EBOOK MANAGEMENT SYSTEM

13.12.2024

PAUL PARKER
C00244601

Overview

For my Computer Science final project, I developed a full-stack ebook library application designed for seamless ebook management. The platform combines modern web development tools like Next.js, Tailwind CSS, and PostgreSQL with Prisma ORM, delivering a user-friendly experience.

Project Proposal

Project(Systems & Services) Proposal Submission:

My proposal for the project is to develop a library management system as a website, featuring two user roles: administrators (librarians) and patrons (users). The site will include a registration and login page, enabling users to create accounts, access their respective portals, and log out.

The system will include core features such as book management, user management, search functionality, data visualisation, and a map feature highlighting selected library locations across Ireland. Administrators will have access to a dedicated portal that presents a dashboard for managing books, allowing them to add, edit, and delete entries in the database through user-friendly forms. Patrons will be directed to a separate dashboard where they can manage their accounts, search for books by title, author, or genre, and explore a map showcasing library locations. This design structure ensures both administrators and users have intuitive access to the system's various functionalities.

The website will be developed using frontend technologies such as HTML, CSS, PHP, and JavaScript (with the React framework). The backend will utilise Python, JSON API, GeoJSON, an SQLite database, Java, and the Django framework. The database will be prepopulated with ebooks downloaded from Project Gutenberg to demonstrate the core features of the system. Additionally, the JSON web API will be sourced from Gutendex, enabling the fetching of data from the Gutenberg website. The website will utilise this data to display visualisations on the homepage, highlighting metrics such as book popularity, the number of books by genre, and by author. This information, sourced from the API, updates daily, allowing for dynamic and real-time visualisations.

The project will also utilise additional APIs from data.gov.ie, specifically from the library datasets. These datasets provide information on the locations of various libraries across Ireland. The project will use GeoJSON to visualise these locations on a map, marking them with pins to enhance user navigation and accessibility. The project will fetch 8 APIs from library datasets: Libraries DLR, Wicklow Branch Libraries, Wexford Libraries, Libraries FCC, Libraries - Roscommon, GCC Libraries, Library Services SDCC, cork-city-council-libraries. (API Gateway). The initial scope of the project will be restrained to basic features. If correctly and successfully implemented, the project scope will be expanded to implement additional features.

Project Presentation

Paul Parker

HDIP IN SCIENCE AND COMPUTING
C00244601

THE PROJECT

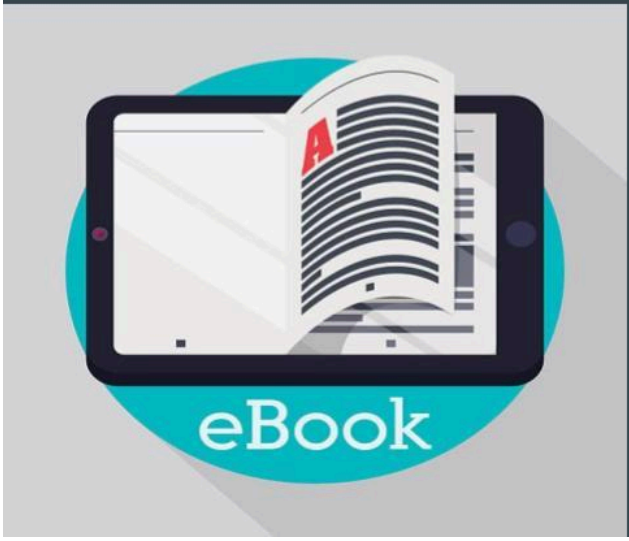
THE PROJECT

The project aims to develop a web application for managing an ebook library system.

The system will include core features such as book management, user management, search functionality, data visualisation, and a map feature highlighting selected library locations across Ireland.

Administrators will have access to a dedicated portal that presents a dashboard for managing books, allowing them to add, edit, and delete entries in the database through user-friendly forms. Patrons will be directed to a separate dashboard where they can manage their accounts, search for books by title, author, or genre, and explore a map showcasing library locations.

PROJECT OBJECTIVES



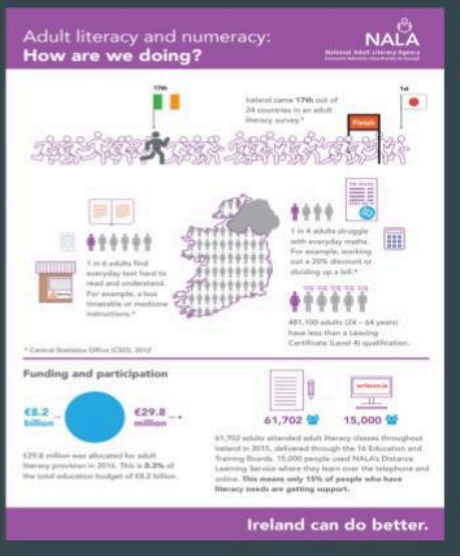
1. Apply the skills you have acquired.
2. Establish a project scope.
3. Establish a project timeline.
4. Set project targets and goals.
5. Research phase.
6. Production phase.
7. Documentation.
8. Testing.
9. Demo Product.

Background

Literacy and numeracy statistics: The OECD Adult Skills Survey shows that 17.9% or about 1 in 6, Irish adults are at or below level 1 on a five level literacy scale. At this level a person may be unable to understand basic written information.

Literacy is like a muscle. If you don't use reading and writing skills every day you can get out of practise.

There is also a stigma attached to unmet literacy, numeracy and digital literacy needs. Often people feel too embarrassed to return to learning and go to great extremes to hide their needs from their friends and family, which exacerbates the problem for them.



Background

Unmet literacy needs also costs public services, businesses and the economy millions each year.

There are 17.1 million visits made to public libraries each year in Ireland.

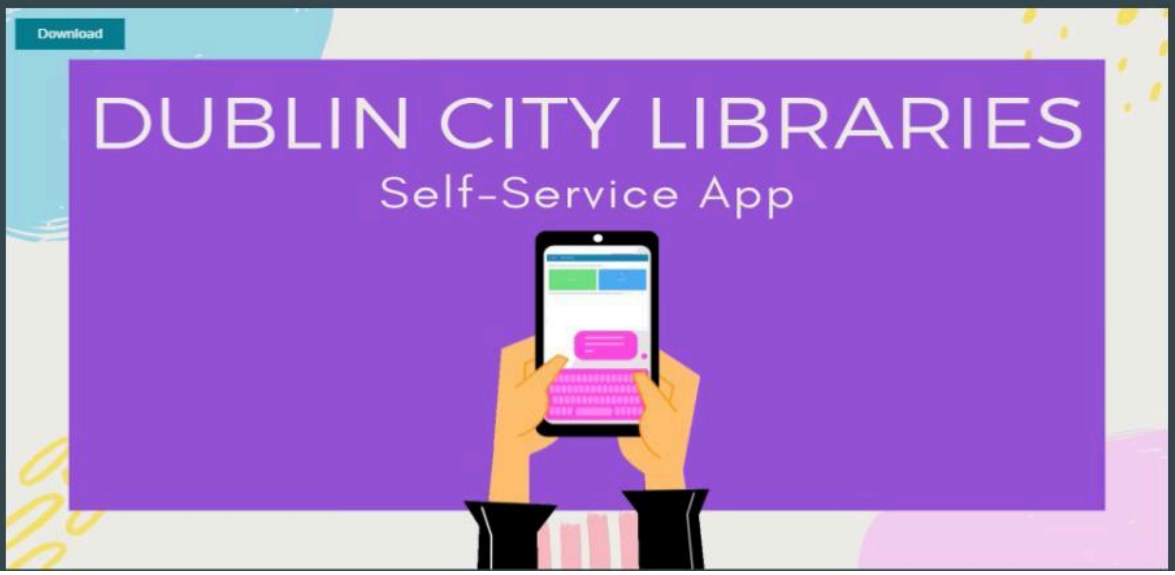
There are 19.3 million books, audio books, CDs and DVDs borrowed from public libraries annually.

There are 336 branch libraries and 31.5 mobile libraries.

SOURCES:
https://www.nala.ie/literacy-and-numeracy-in-ireland/#:~:text=Literacy%20and%20numeracy%20statistics,below%20level%201%20for%20numeracy.

https://www.askaboutireland.ie/libraries/public-libraries/fast-facts-and-figures/#:~:text=Fast%20Facts%20%26%20Figures,and%20via%20free%20wi%2Dfi.

Research



Research

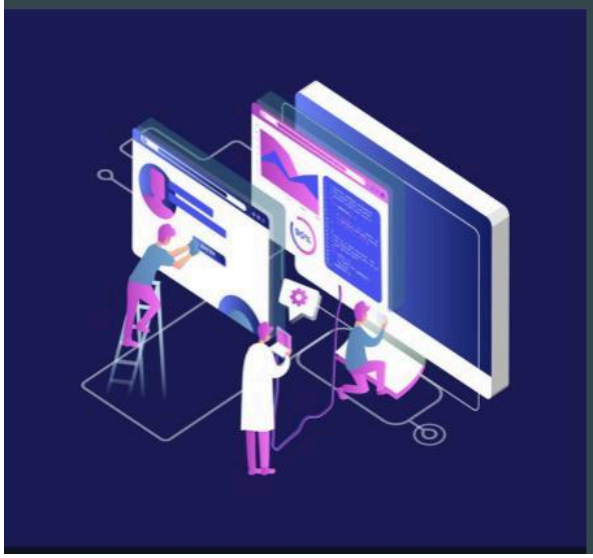
The library App gives you a quick and easy way to manage your account and search the library catalogue on your smartphone or tablet. Once you're a member you can use the App to:

- access and manage your library account.
- issue and return items.
- reserve and renew items.
- locate your nearest branch library and see its opening hours.
- view a list of upcoming library events.
- download eBooks, eAudiobooks, eMagazines, eComics and eGraphic novels.
- access online resources.

SOURCES:

https://www.dublincity.ie/residential/libraries/using-your-library/libraries-mobile-app

TECHNOLOGIES



Frontend technologies:

HTML, CSS, PHP, JavaScript.

Backend Technologies:

Python, JSON API, GeoJSON, Java.

Django framework.

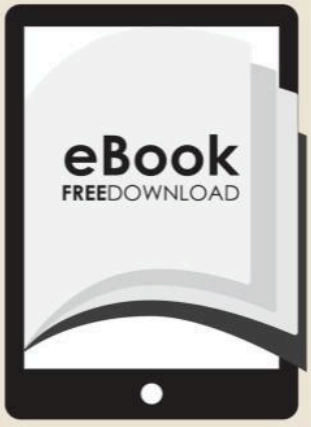
Postgres database.

APPLICATION FUNCTION

Project Gutenberg is a library of over 70,000 free eBooks, including such titles as Frankenstein, Pride and Prejudice, Moby Dick

The database will be populated with ebooks downloaded from Project Gutenberg. The JSON web API will be sourced from Gutendex, enabling the fetching of data from the Gutenberg website.

The website will utilise this data to display visualisations highlighting metrics such as book popularity, the number of books by genre, and by author. This information, sourced from the API, updates daily, allowing for dynamic and real-time visualisations.



APPLICATION FUNCTION

The project will also utilise additional APIs from data.gov.ie, specifically from the library datasets.

These datasets provide information on the locations of various libraries across Ireland. The project will use GeoJSON to visualise these locations on a map, marking them with pins to enhance user navigation and accessibility.

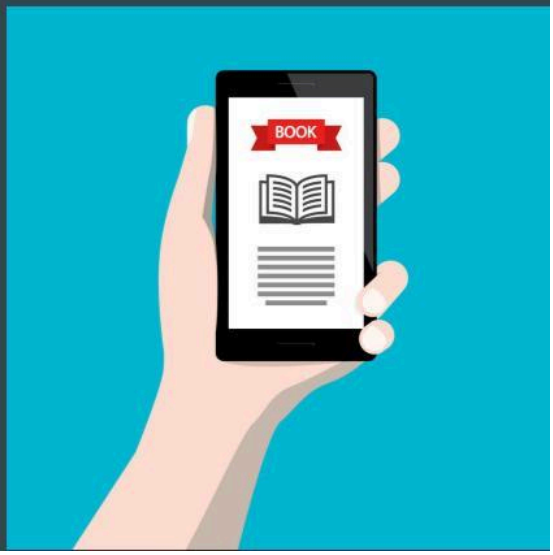
The project will fetch 8 APIs from library datasets: Libraries DLR, Wicklow Branch Libraries, Wexford Libraries, Libraries FCC, Libraries - Roscommon, GCC Libraries, Library Services SDCC, cork-city-council-libraries.

PROJECT OUTCOMES

Provide an eLearning tool that complements existing public resources, like the Dublin City Council Libraries mobile app.

This application enables administrators to upload, edit, and delete free ebooks, allowing users to access and enjoy a diverse selection of free ebooks available online.

It aims to promote engagement with ebooks to enhance adult literacy and serve as a valuable public learning resource.



QUESTIONS?

THANK YOU

Project Aim

My goal is to develop an ebook library application hosted on a website that will allow users to sign up, create an account, and access a personalised user dashboard. Through this dashboard, users will be able to read ebooks using an integrated ebook reader, while also managing their profile, which includes their profile picture, username, and email address. The website will also include features such as visualisations powered by data from the Gutendex API. These visualisations will help users to identify popular books from Project Gutenberg's website, and allow users to make informed decisions on which ebooks to engage with.

One notable feature of the application will be an interactive map displaying various library locations across Ireland, marked by pins. By selecting a pin, users can view details about the library, such as its address, contact information, and operating hours.

The aim of this application is to be used as an e-learning tool for both educators and active learners.

User Needs Research

https://www.nala.ie/literacy-and-numeracy-in-ireland/#:~:text=Literacy%20and%20numeracy%20statistics,below%20level%201%20for%20numeracy

https://www.askaboutireland.ie/libraries/public-libraries/fast-facts-and-figures/#:~:text=Fast%20Facts%20%26%20Figures,and%20via%20free%20wi%2Dfi

https://www.oecd.org/content/dam/oecd/en/publications/reports/2019/11/the-survey-of-adult-skills_d7f1bc16/f70238c7-en.pdf

This application is designed to encourage adult readers in Ireland to embrace digital reading. According to the OECD Adult Skills Survey, 17.9% of Irish adults, or approximately 1 in 6, score at or below level 1 on a five-level literacy scale. There are 17.1 million visits made to public libraries each year in Ireland. There are 19.3 million books, audio books, CDs and DVDs borrowed from public libraries annually. There are

336 branch libraries and 31.5 mobile libraries in Ireland. This application aims to complement existing public resources to help raise literacy levels and promote reading among Irish adults. Existing applications and websites that provide free ebooks attract a diverse user base, spanning from young to old. Barriers to entry include a lack of awareness about available resources, as well as outdated and complex user interfaces that hinder the ease of finding and using existing platforms. These non user friendly interfaces make it challenging for users to navigate the content, diminishing engagement and reducing the likelihood of them returning to the application or website. The goal is to encourage and motivate users to embrace digital reading through this ebook library application, by addressing these barriers to entry and making the platform as user friendly and engaging as possible.

User Needs

- Websites offering free ebooks can be too text heavy and lose the user's interest quickly.
- Users want an interactive and visual friendly way to engage with the topic.
- Users want a visual representation of popular ebooks rather than long lists of text.
- Users want an easy to navigate user interface that quickly engages the user in the topic.
- Users want the option to pursue the information if they so desire/are interested, rather than be bombarded with that information when they first engage with the website.

- Users want the ability to access quick and visual friendly information on nearby libraries through a map component that gives the user the ability to find relevant information.

- Users want an ability to manage and customise their profile through a user specific dashboard.

- Users want the ability to view the website through many different devices and viewports.

User Persona

Thomas the Student

Age: 17

Gender: Male

Interests: Reading

Occupation: Student

Goals: Wants to access ebooks such as ‘A Dolls House’ for his studies for his leaving cert. Wants the ability to study on his phone while travelling to and from school.

Ciara the English Teacher

Age: 28

Gender: Female

Interests: Teaching and reading

Occupation: Teacher

Goals: Wants to access free ebooks so she can create lesson plans based on classical literature greats for her english students.

Gerald the Delivery Driver

Age: 36

Gender: Male

Interests: Sports and Television

Occupation: Delivery driver

Goals: Wants to improve his literacy level by reading ebooks on his phone while on his break.

Mary the Senior Citizen

Age: 68

Gender: Female

Interests: Reading and knitting

Occupation: Retired

Goals: Wants to pursue her favourite pastime of reading but finds it difficult to read traditional books anymore, finds reading on her tablet much more easy.

Problem Statement

Young students seek an engaging and intuitive way to read ebooks on their devices, with easy access to free resources for their English studies. Teachers desire a learning platform that allows them to create lesson plans and keeps students focused through a user-friendly website or application. Text-heavy websites can cause users to become distracted and disengaged. To serve as an effective e-learning tool, the website must capture and maintain users' attention while encouraging interaction. Adult learners, on the other hand, want an engaging way to improve their literacy skills, helping them overcome the stigma of below-average literacy among their peers.

Objectives

User interaction through the website will help to engage the user with the topic, where learning is interactive and fun. A visual first approach that seeks to engage the user through interactive means. User centric design and layout to ensure navigational issues for the user is minimised, thus allowing for various age groups and varying levels of computer literacy to be able to use the website. This user centric design allows for the user to further pursue the topic if they have sufficient interest. This will ensure minimum barriers of entry for the user. By creating a website that can be used as an e-learning tool and resource to co-exist with public resources in Ireland, this will provide the opportunity for people of all ages to engage with the website.

Ideation

1. We should provide pin markers for each library location on the map of Ireland, to allow further interaction and accessing relevant information if the user so desires.

2. Users do not have a sufficient attention span to parse through text heavy websites of existing ebook websites. We should instead have a visual first method to engage the user via interaction.

3. Could the website layout be designed to resemble popular modern applications? Since the target audience includes students, teachers, and adult learners, adopting a layout similar to widely used apps could make the platform feel familiar. This familiarity could help engage users more effectively by aligning with the interface styles they are already accustomed to.

4. With enough time and resources, we could implement a scraping tool that fetches data from existing public library applications in Ireland. This would allow users to search for desired ebooks and see if it's available in a library close to them.

5. Learning about a new topic is like starting a new video game. The more engaging and interactive the experience is from the start, the more likely users are to keep playing and in this case, keep reading. Just like in a game, if the user is hooked early with exciting challenges and smooth navigation, they'll be motivated to continue exploring and leveling up their knowledge.

6. We offer several different filter options in the visualisations section. This way the user understands different metrics about the ebooks on offer.

7. Most useful: A map of Ireland with pins showcasing information about that specific library location.

8. Most desirable: A scraping tool to fetch data from existing public resources, to allow the application to be a complimentary e-learning tool.

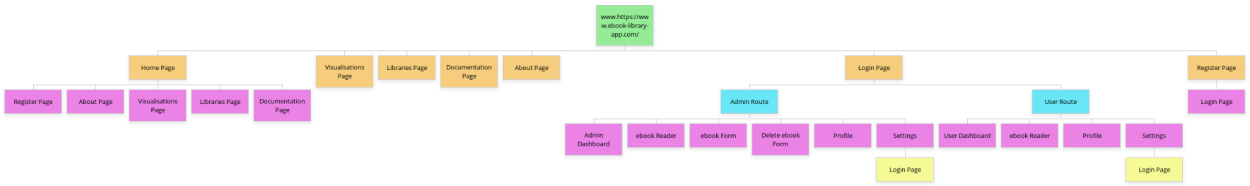
9. Easiest for most people to use: Visual representations of the ebook popularity by download count, allowing the user to make an informed choice of which ebook to read.

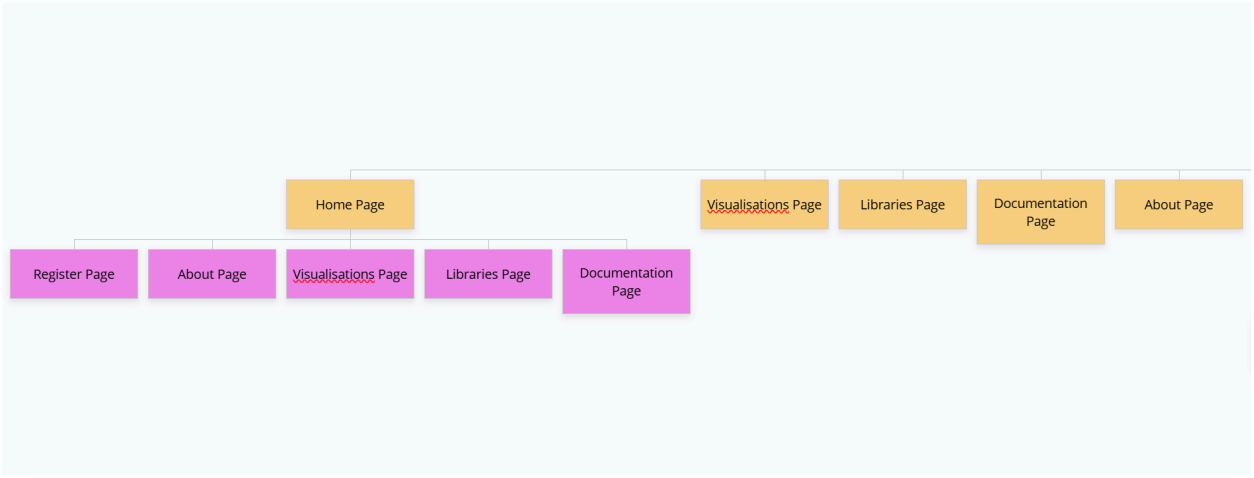
10. Mobile responsive design will allow for interaction across various devices, thus removing barriers of entry.

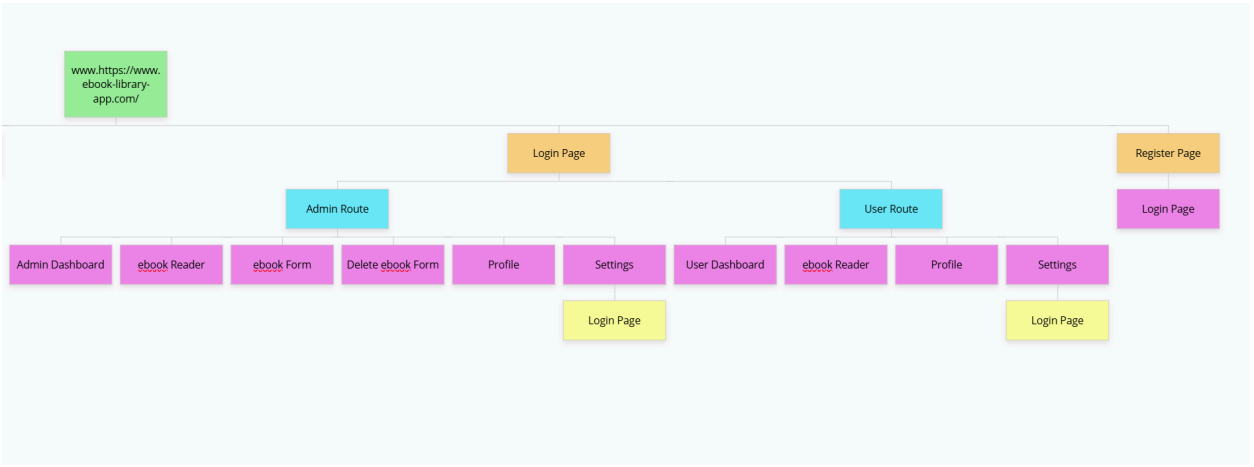
11. Most functional: An integrated ebook reader that allows users to quickly access different ebooks.

12. Most sustainable: Vector based graphics across the website allowing for interaction across various devices.

Sitemap







Page Descriptions

Page	Items

Home/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. Opacity animation on heading and paragraph on page load. 6. Get started button links to sign up page. 7. Learn more button links to about page. 8. Interactive Libraries Map Learn more button links to libraries page. 9. Data Learn more button links to visualisations page. 10. Image carousel changes image on click of right or left arrow. 11. Opacity animation on heading and paragraph on section entering id section coming into view. 12. Project Documentation Learn more button links to Documentation page. 13. Opacity animation on heading and paragraph on section entering id section coming into view.

Page	Items
Home/page.tsx-part 2	1. onHover event increases the scale of the img container. 2. Opacity animation on review cards on the section id coming into view. 3. onClick Function on up arrow svg that returns to the top of the page. 4. Footer Section, links to Visualisations, Libraries, Documentation, and About pages.

Page	Items
Visualisations/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. Select dropdown button filters barchart on user selection. 6. onHover event on the bar chart creates a popup dialog box that displays book title and download count. 7. Footer Section, links to Visualisations, Libraries, Documentation, and About pages.

Page	Items
Libraries/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. onClick event on map markers pin that creates popup dialog box displaying text information about relevant library 6. Footer Section, links to Visualisations, Libraries, Documentation, and About pages.

Page	Items
Documentation/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. onClick event on embedded link that creates a new window with documentation displayed. 6. Footer Section, links to Visualisations, Libraries, Documentation, and About pages.

Page	Items
About/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. onClick event on accordion component causes user selected part to display text. 6. Footer Section, links to Visualisations, Libraries, Documentation, and About pages.

Page	Items
Login/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. onClick event on the eye icon displays the user's password field. 6. Don’t have an account button links to the registration page. 7. onSubmit event on Login button routes user to relevant dashboard. 8. On successful login a JWT token is created to authenticate the user's session.

Page	Items
Register/page.tsx	1. Navigation Bar, links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 2. Mobile hamburger menu, drop downs links to Home, Visualisation, Libraries, Documentation, About, Sign up, Log in pages. 3. Clickable Header Element, links to Home page. 4. onScroll event listener makes the navbar appear on the up scroll and disappear on the down scroll. 5. onClick event on the eye icon displays the user's password field. 6. Already have an account button links to the login page. 7. onSubmit event on Register button creates users account and stores their details in the database.

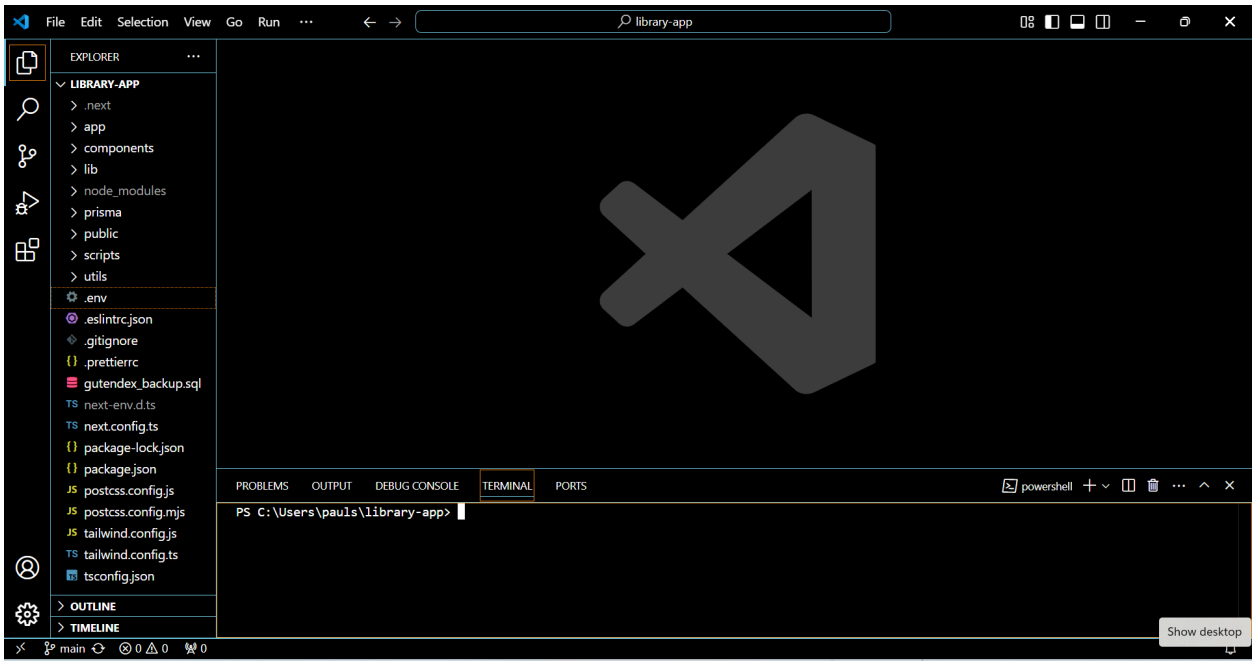
Page	Items

AdminDashboard/page.tsx	1. onClick event opens tab menu displaying a list of possible tabs for the user to navigate to. 2. Ebook reader tab: Select dropdown button lists all possible ebooks to open in the ebook reader, onClick event fetches relevant ebooks from the database and displays its properties. 3. Ebook Form tab: Choose file button on click creates dialog box for users to upload the ebook file and relevant cover page image. onSubmit event on submit button commits the imputed details from the form and adds the ebook and its properties to the database. onHover event on cover page when ebook loads. 4. Delete ebook Form tab: Select button dropdown lists all available ebooks for admin to select and on selection Delete Ebook appears and onClick deletes relevant ebooks from the database. 5. Profile tab: onClick event for Change Profile Picture button creates a popup dialog box that displays several images for the user to choose from. onHover event creates a blue ring around the image. onClick event changes the user's profile picture and commits the change to the user table in the database. onClick event for Close button closes the popup dialog box. 6. Settings tab: onClick event for Logout button routes user back to login page. Clears JWT token from the session

Page	Items
UserDashboard/page.tsx	1. onClick event opens tab menu displaying a list of possible tabs for the user to navigate to. 2. Ebook reader tab: Select dropdown button lists all possible ebooks to open in the ebook reader, onClick event fetches relevant ebooks from the database and displays its properties. onHover event on cover page when ebook loads. 3. Profile tab: onClick event for Change Profile Picture button creates a popup dialog box that displays several images for the user to choose from. onHover event creates a blue ring around the image. onClick event changes the user's profile picture and commits the change to the user table in the database. onClick event for Close button closes the popup dialog box. 4. Settings tab: onClick event for Logout button routes user back to login page. Clears JWT token from the session

Project Setup

This project was built in Visual Studio Code using NEXT.js by Vercel - The React Framework.

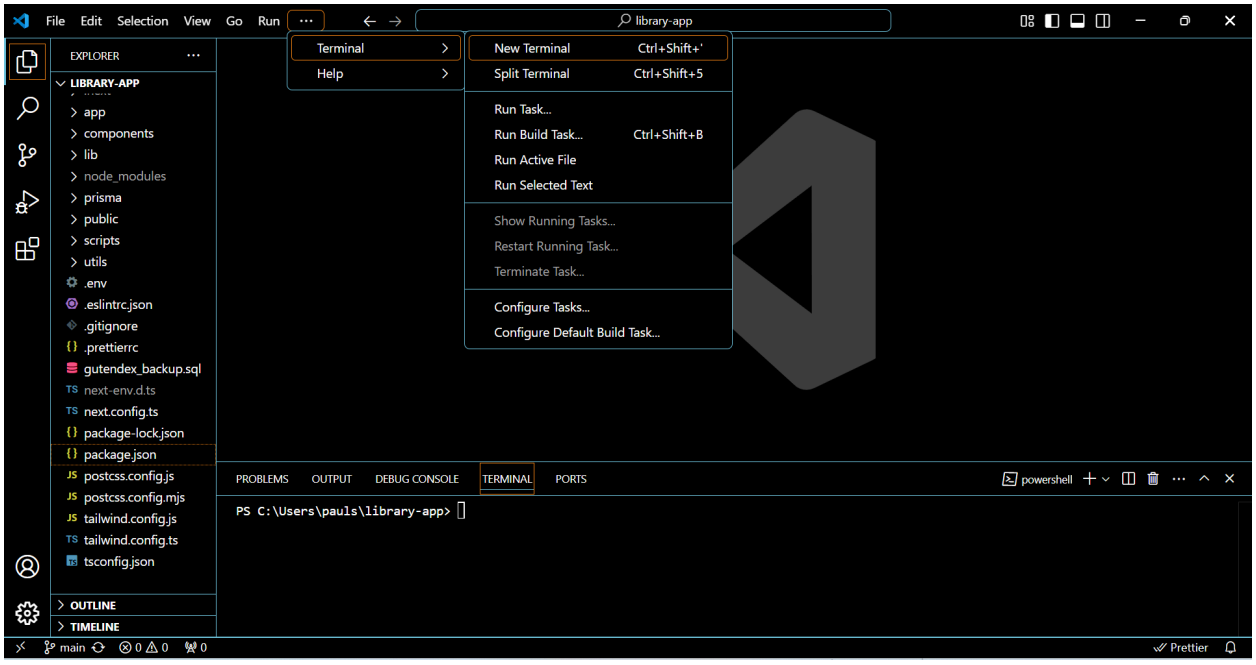


Install Node.js from nodejs.org.

Install Visual Studio Code.

Launch Visual Code and create a new project folder.

Open the terminal in the project.



In the terminal type the following: npx create-next-app@latest my-next-app

Replace ‘my-next-app’ with your project folder name.

You will then be prompted by the command line interface:

TypeScript? (yes/no)

ESLint? (yes/no)

Tailwind CSS? (yes/no)

src/ directory? (yes/no)

App Router? (yes/no for /app directory)

The project utilises yes for all of these except App Router.

The project uses the following the dependencies:

```
{
  "name": "library-app",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "build:full": "npm run build",
    "start": "next start",
    "lint": "next lint",
    "hashPassword": "node scripts/hashPassword.js"
  },
  "dependencies": {
    "@gsap/react": "^2.1.1",
    "@heroicons/react": "^2.2.0",
    "@material-tailwind/react": "^2.1.10",
    "@prisma/client": "^6.0.1",
    "@supabase/supabase-js": "^2.47.7",
```

```
    "bcrypt": "^5.1.1",
    "bcryptjs": "^2.4.3",
    "d3": "^7.9.0",
    "dotenv": "^16.4.7",
    "fs-extra": "^11.2.0",
    "gsap": "^3.12.5",
    "jsonwebtoken": "^9.0.2",
    "leaflet": "^1.9.4",
    "next": "^15.1.0",
    "pg": "^8.13.1",
    "react": "^18.0.0",
    "react-cookie": "^7.2.2",
    "react-cookie-consent": "^9.0.0",
    "react-dom": "^18.0.0",
    "react-icons": "^5.4.0",
    "react-leaflet": "^4.2.1"
  },
  "devDependencies": {
    "@types/bcryptjs": "^2.4.6",
    "@types/d3": "^7.4.3",
    "@types/jsonwebtoken": "^9.0.7",
    "@types/leaflet": "^1.9.15",
    "@types/node": "^20",
    "@types/pg": "^8.11.10",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "@typescript-eslint/eslint-plugin": "^8.18.0",
```

```
    "@typescript-eslint/parser": "^8.18.0",
    "autoprefixer": "^10.4.20",
    "eslint": "^8.57.1",
    "eslint-config-next": "^13.4.19",
    "postcss": "^8.4.49",
    "prettier": "^3.3.3",
    "prettier-plugin-tailwindcss": "^0.6.8",
    "prisma": "^6.0.1",
    "tailwindcss": "^3.4.15",
    "typescript": "^5"
  }
}
```

These can be installed using the npx and npm install command inside the Visual Studio Code terminal.

The project was built in a local development environment using the npm run dev command - Local:http://localhost:3000.

Gutendex API

The project uses the Gutendex API to download catalog information from Project Gutenberg, an online library of free ebooks. Gutendex uses Django to download catalog data and serve it in a simple JSON REST API. Project Gutenberg has no such public API of its own, but it publishes nightly archives of complicated XML files. Gutendex downloads these files, stores their data in a database, and publishes the data in a simpler format. The database in the context of this project is a postgres database.

Follow the link https://github.com/garethbjohnson/gutendex?tab=readme-ov-file

Click on the green code dropdown button and download the zip file. Then extract the folder in your desired location.

To make this work for this project I followed the installation guide https://github.com/garethbjohnson/gutendex/wiki/Installation-Guide.

This required me to the following:

1) Install Prerequisites:

   Download and install Python from python.org.

   Download and install PostgreSQL from postgresql.org.

2) Open command prompt and enter the following commands:

   Switch to the PostgreSQL shell: psql -U postgres - You will be prompted to enter the PostgreSQL password during installation, make sure to make a note of this value.

3) Create a database: CREATE DATABASE gutendex; - Replace ‘gutendex’ with the desired name for your database.

4) Then create a user and set a password: CREATE USER gutendex WITH PASSWORD 'yourpassword'; - Replace ‘gutendex’ and ‘yourpassword’ with your desired values.

5) Then grant privileges to the user: GRANT ALL PRIVILEGES ON DATABASE gutendex TO gutendex; - Replace ‘gutendex’ values with your database and user values.

6) Then exit the psql shell: \q

7) Then enter the ‘gutendex-master’ folder and type ‘cmd’ into the path directory window in the top of the file explorer window, this will launch command prompt.

8) Then install Virtualenv: pip install virtualenv

9) Then create a virtual environment: python -m venv venv

10) Then activate the virtual environment by navigating to the directory: ‘venv\Scripts\activate’ and entering ‘activate’

11) Then install the required packages: pip install -r requirements.txt

12) Then deactivate the virtual environment: deactivate

13) Then set up the environmental variables, copy the .env.template file to a new .env file: copy gutendex\.env.template gutendex\.env

14) Then open the .env file in any text editor, such as Notepad ++ and replace the placeholder values DATABASE_URL, DATABASE_USER, DATABASE_PASSWORD, etc with your relevant values.

15) Then migrate the database, ensure you're in the root directory, and the virtual environment is activated. Type type ‘cmd’ into the path directory window in the top of the file explorer window, this will launch command prompt: python manage.py migrate

16) Then populate the database: python manage.py updatecatalog

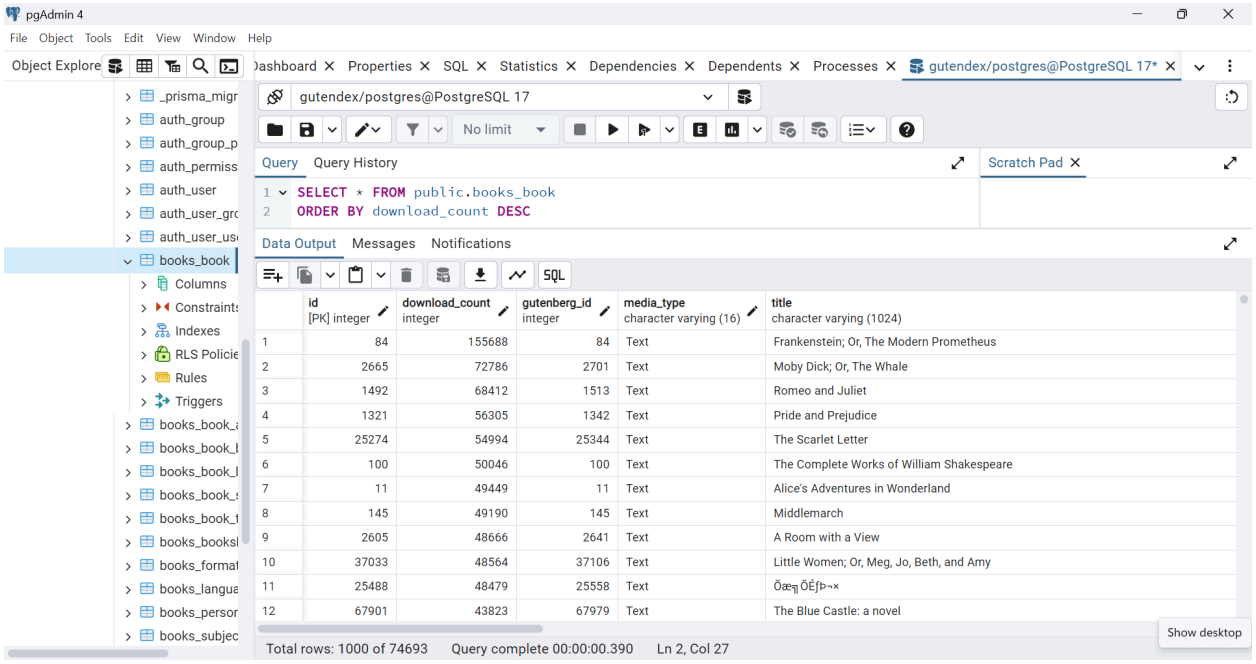
This downloads a file archive of Project Gutenberg's catalog data and decompresses the files into a new directory, catalog_files. It places the contained files in catalog_files/rdf, and it stores a log in catalog_files/log. If your database already contains catalog data, the above command will update it with any new or

updated data from Project Gutenberg. This process takes several minutes to complete.

After completing all these above steps the postgres database will be populated with catalog data from Project Gutenberg.

Initial Visualisation of the Gutendex API Data

For my project, I wanted to understand how I would visualise the catalog data. By using pgAdmin4 I was able to view the catalog data in my database. By querying my database with the following SQL command 'SELECT * FROM public.books_book ORDER BY download_count DESC' I was able to view the 'books_book' table and sort it by the 'download_count' column in descending order from the highest value first.



	id [PK] integer	download_count integer	gutenberg_id integer	media_type character varying (16)	title character varying (1024)
1	84	155688	84	Text	Frankenstein; Or, The Modern Prometheus
2	2665	72786	2701	Text	Moby Dick; Or, The Whale
3	1492	68412	1513	Text	Romeo and Juliet
4	1321	56305	1342	Text	Pride and Prejudice
5	25274	54994	25344	Text	The Scarlet Letter
6	100	50046	100	Text	The Complete Works of William Shakespeare
7	11	49449	11	Text	Alice's Adventures in Wonderland
8	145	49190	145	Text	Middlemarch
9	2605	48666	2641	Text	A Room with a View
10	37033	48564	37106	Text	Little Women; Or, Meg, Jo, Beth, and Amy
11	25488	48479	25558	Text	Õœ╗ÖÉƒÞ¬×
12	67901	43823	67979	Text	The Blue Castle: a novel

This helped me understand my data. My next step was to create a python script to visualise this data. First open command prompt and install the following python packages:

1. psycopg2 is used to interact with the PostgreSQL database: pip install psycopg2
2. pandas is used for data manipulation and analysis: pip install pandas

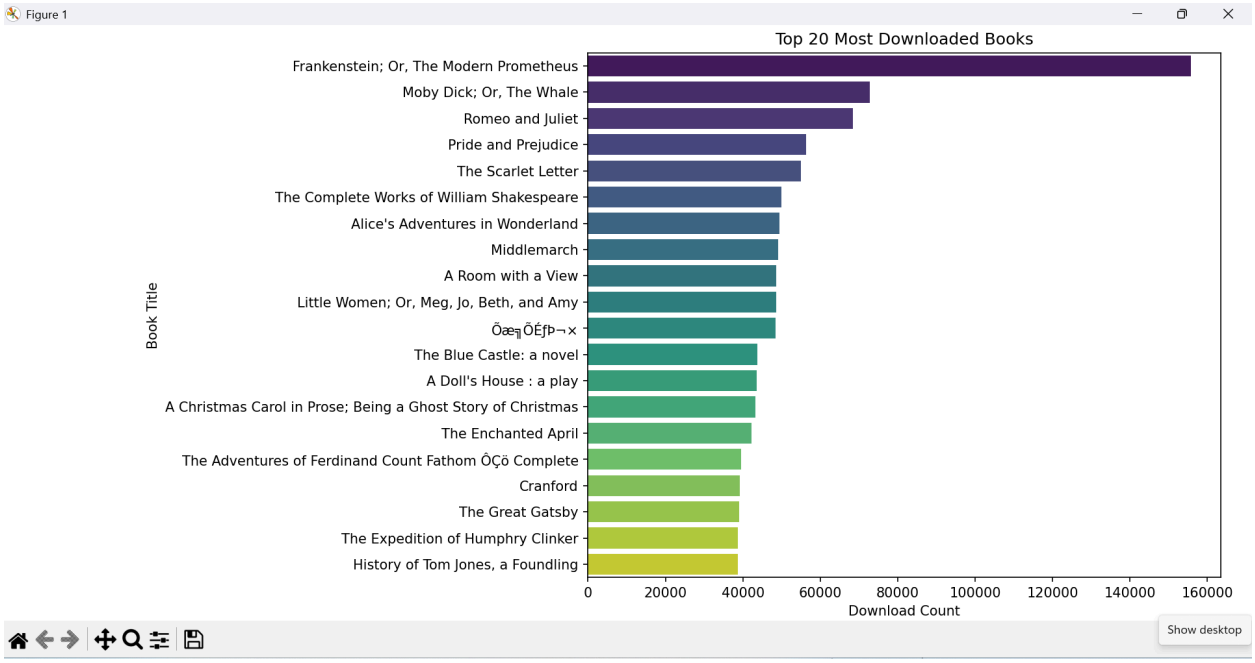
3. matplotlib is used for creating static, interactive, and animated visualisations: pip install matplotlib
4. seaborn is used for statistical data visualisation and works on top of matplotlib: pip install seaborn

Python Script:

```
# Import libraries
import psycopg2  # Interact with the PostgreSQL database
import pandas as pd  # Data manipulation and analysis
import matplotlib.pyplot as plt  # Plotting library for creating visualizations
import seaborn as sns  # Statistical data visualization


# Create 'Database' class
class Database:
    # Initialiae the Database class with connection parameters 'self'. 'db_name', 'user', 'password'. 'host', 'port'
    def __init__(self, db_name, user, password, host="localhost", port="5432"):
        # Establish a connection to the PostgreSQL database
        self.conn = psycopg2.connect(
            dbname=db_name, user=user, password=password, host=host, port=port
        )
        # Create a cursor object to interact with the database
        self.cursor = self.conn.cursor()


    # Method to fetch the books from the 'books_book' table and sort by 'download_count'
    def fetch_download_counts(self, limit=20):
        self.cursor.execute("""
            SELECT title, download_count
```

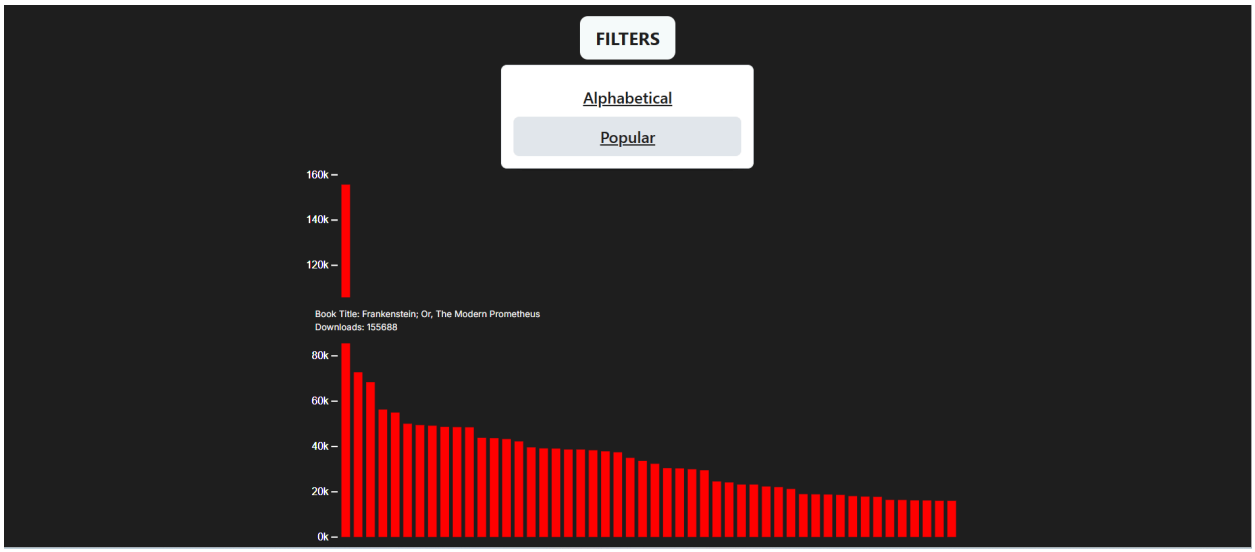
```
            FROM books_book
            ORDER BY download_count DESC
            LIMIT %s
        """, (limit,))
        return self.cursor.fetchall()  # Return top download count with corresponding titles

    # Close the database connection
    def close(self):
        self.conn.close()

# Function to visualise the top 20 most downloaded books
def visualize_download_counts(data):
    # Convert data into a DataFrame for plotting
    df = pd.DataFrame(data, columns=["title", "download_count"])

    # Set up the visualisation styles for plotting
    plt.figure(figsize=(10, 8))
    sns.barplot(y='title', x='download_count', data=df, hue='title', legend=False,
palette='viridis')
    plt.title('Top 20 Most Downloaded Books')  # Visualisation title
    plt.xlabel('Download Count')  # X-axis label
    plt.ylabel('Book Title')  # Y-axis label
    plt.tight_layout()  # Adjust layout to prevent overlap

    # Display the plot
    plt.show()
```

```
# Main function to display visualisation
def main():
    # Database credentials
    db_name = "your_db_name"  # Database name - Replace 'your_db_name' with your database name value
    user = "your_db_user"  # Database user - Replace 'your_db_user' with your database name value
    password = "your_db_password"  # User password - Replace 'your_db_password' with your database name value

    # Create a Database object with the database credentials parameters
    db = Database(db_name, user, password)

    # Fetch and visualise download counts for the top 20 books
    download_data = db.fetch_download_counts()
    print("\nVisualising top 20 most downloaded books...")
    visualize_download_counts(download_data)

    # Close the database connection
    db.close()

if __name__ == "__main__":
    main()
```



Visualisations by D3.js

Then using D3.js library and the code shown below in the barchart.tsx component and visualisations API, I was able to create visualisations of the catalog book information and represent them in a bar chart format. The user has two filter options to view the data.



```
// Render the component client-side
'use client';


// Import React hooks and import d3 library for data visualization
import { useEffect, useRef } from 'react';
import * as d3 from 'd3';


// Define the data fetched from the API
interface Point {
  title: string;
  download_count: number;
}


// Specify that 'data' is an array of Point objects
const BarChart = ({ data }: { data: Point[] }) => {
  const svgRef = useRef<SVGSVGElement | null>(null); // Create a reference
to the SVG element for drawing the bar chart


  useEffect(() => {
    // If data is empty do not render an empty bar chart
    if (data.length === 0) return;


    // Define bar chart dimensions and margins
    const width = 640;
    const height = 400;
    const marginTop = 20;
    const marginRight = 0;
    const marginBottom = 30;
```

```
    const marginLeft = 40;


    // Map book titles on the x-axis
    const x = d3.scaleBand()
      .domain(data.map(d => d.title)) // Use book titles for the x-axis
      .range([marginLeft, width - marginRight]) // Sets the scale from
left margin to available width.
      .padding(0.3); // Padding between bars


    // Mapping download count to the y-axis
    const y = d3.scaleLinear()
      .domain([0, d3.max(data, d => d.download_count) || 0]) // Use the
max download count for y-axis
      .nice() // Adjusts the scale to use rounded values
      .range([height - marginBottom, marginTop]); //Sets the scale from
bottom margin to top margin.


    // Select the SVG element and set its attributes for size, style, and
appearance
    const svg = d3.select(svgRef.current)
      .attr("viewBox", [0, 0, width, height])
      .attr("style", "max-width: 100%; height: auto; font: 10px
sans-serif; overflow: visible;");


    // Plot the 'data' array on the bar chart, using the 'title' as a
unique identifier for each bar
    const bars = svg.selectAll("rect")
      // @ts-expect-error data type d is unknown[] causing typescript to
raise error over title property not being guaranteed
```

```
      .data(data, d => d.title);

    // Create and animate bars with transition effects
    bars.join("rect")
      // @ts-expect-error data type d is unknown[] causing typescript to
raise error over title property not being guaranteed
      .attr("x", d => x(d.title))
      .attr("y", d => y(d.download_count))
      .attr("height", d => y(0) - y(d.download_count))
      .attr("width", x.bandwidth())
      .attr("fill", "red")
      .transition()
      .duration(750)
      .delay((d, i) => i * 20);

    // Add Y-axis and style the text
    svg.append("g")
      .attr("transform", `translate(${marginLeft},0)`)
      // @ts-expect-error data type d is unknown[] causing typescript to
raise error over title property not being guaranteed
      .call(d3.axisLeft(y).tickFormat(d => `${d / 1000}k`))
      .call(g => g.select(".domain").remove())
      .selectAll("text")
      .style("fill", "white");

    // Remove x-axis labels to avoid overlapping text
    svg.selectAll(".x-axis").remove();
```

```
// Style and position the information box
const tooltip = d3.select("body")
  .append("div")
  .style("position", "absolute")
  .style("background-color", "#212121")
  .style("color", "#fff")
  .style("padding", "10px")
  .style("font-size", "9px")
  .style("border-radius", "10px")
  .style("visibility", "hidden");


// Show information box on hover
bars.on("mouseenter", function (event, d) {
  tooltip
    .style("visibility", "visible")
    .html(`Book Title: ${d.title}<br>Downloads: ${d.download_count}`);
})
// Information box position on hover
.on("mousemove", function (event) {
  tooltip
    .style("top", (event.pageY + 10) + "px")
    .style("left", (event.pageX - 40) + "px");
})
// Information box disappears on mouse off
.on("mouseleave", function () {
  tooltip.style("visibility", "hidden");
});
```

```
  }, [data]); // Re-run effect when `data` changes

  return (
    <div className='min-h-80 lg:min-h-screen mt-20'>
      <svg ref={svgRef}></svg> {/* Render the bar chart */}
    </div>
  );
};

export default BarChart; // Export the BarChart component

import { NextResponse } from "next/server"; // Import NextResponse from
'next/server'
import prisma from "../../../lib/prisma"; // Import Prisma client to
interact with the database

// GET function
export async function GET(request: Request) {
  // Parse the request URL to access query parameters
  const url = new URL(request.url);

  // Get page and limit from query parameters
  const page = parseInt(url.searchParams.get("page") || "1", 10); //
Default to page 1
  const limit = parseInt(url.searchParams.get("limit") || "50", 10); //
Default to 50 items per page
  const offset = (page - 1) * limit; // Calculate how many records to skip
based on the current page and the limit per page
```

```
  try {
    // Fetch books, sorted by download count
    const books = await prisma.books_book.findMany({
      select: {
        title: true, // Select only the 'title' field
        download_count: true, // Select the 'download_count' field
      },
      orderBy: {
        download_count: 'desc', // Order books by download count in
descending order
      },
      skip: offset, // Skip the records from previous pages based on the
offset
      take: limit, // Get 'limit' number of books
    });

    // Get the total number of books
    const totalBooks = await prisma.books_book.count();

    // Return the books data, total count, and page information
    return NextResponse.json({
      data: books, // Array of fetched books
      total: totalBooks, // Total number of books
      page, // Current page number
      totalPages: Math.ceil(totalBooks / limit), // Calculate the total
number of pages
    });
```

```
  } catch {
    // Return error message
    return NextResponse.json({ error: 'An unexpected error occurred' });
  }
}
```

APIs

The APIS used throughout the project:

Delete Ebook API

```
// Import NextResponse from 'next/server'
import { NextResponse } from "next/server";

// Import prisma to interact with the database
import prisma from "../../../lib/prisma";

// resetEbookSequence function
async function resetEbookSequence() {
// Handle DELETE request to remove an ebook and reset the sequence
const maxId = await prisma.ebook.aggregate({
_max: {
id: true, // Get the highest 'id' value
},
});
```

```
// Set the next sequence ID for 'Ebook' table, defaulting to 1 if no
                        records exist
            const nextId = (maxId._max.id || 0) + 1;
             // Set 'Ebook' sequence to the next ID
               await prisma.$executeRawUnsafe(`
      SELECT setval('public."Ebook_id_seq"', ${nextId}, false);
                            `);
                            }


                    // DELETE function
        export async function DELETE(request: Request) {
                           try {
              // Get query parameters from the request URL
      const ebookId = new URL(request.url).searchParams.get("id");


                  // Ensure ebookId exists
                      if (!ebookId) {
       return NextResponse.json({ error: "Ebook ID is required" }, {
         status: 400 }); // Status code 400 for bad request
                             }


              // Delete the ebook from the database
        const deletedEbook = await prisma.ebook.delete({
                 where: { id: Number(ebookId) },
                           });


                    // Reset the sequence
                  await resetEbookSequence();
```

```
        // Return the deleted ebook as a response
        return NextResponse.json(deletedEbook);
    } catch {
        // Error message
        return NextResponse.json({ error: "An error occurred while deleting the ebook" }, { status: 500 }); // Status code 500 for internal server error
    }
}
```

Ebook Form API

```
import { NextResponse } from 'next/server'; // Import NextResponse from 'next/server'
import { PrismaClient } from '@prisma/client'; // Importing PrismaClient to interact with the database
import { createClient } from '@supabase/supabase-js'; // Import Supabase client

// Creating an instance of PrismaClient to communicate with the database
const prisma = new PrismaClient();

// Initialize Supabase client
const supabase = createClient(
    'https://tshuqzoktjjlooqfenvm.supabase.co', // Supabase Project URL
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InRzaHVxem9rdGpqbG9vcWZlbnZtIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTczNDIxMDY2NiwiZXhwIjoyMDQ5Nzg2NjY2fQ.JGFT5rSkyoym7MiG_Pmo5IlWLxi16ZvmtP15_57UT3U' // 'service_role' Supabase API key
```

```
);


// POST function
export async function POST(request: Request) {
try {
// Parsing the incoming form data
const formData = await request.formData();

// Extract input data from the form
const title = formData.get("title") as string; // Get the title of the ebook
const author = formData.get("author") as string; // Get the author of the ebook
const description = formData.get("description") as string; // Get the description of the ebook
const publishedAt = new Date(formData.get("publishedAt") as string); // Get the publish date of the ebook

// Extracting the ebook file and cover page file from the form data
const file = formData.get("file") as File;
const coverPage = formData.get("coverPage") as File;

// Function to upload files to Supabase Storage
const uploadToSupabase = async (file: File, folder: string): Promise<string> => {
// Upload the file to Supabase Storage buckets
const { data, error } = await supabase
.storage
.from(folder) // Supabase bucket
```

```
            .upload(`${folder}/${file.name}`, file.stream(), {
          upsert: true, // If the file already exists then overwrite it
             duplex: 'half', // Allow file upload with duplex stream
                                    });

                                 if (error) {
          throw new Error(`Supabase upload error: ${error.message}`);
                                    }

           // Ensure the file path exists and return the public URL
                                if (data?.path) {
                                    return
`https://tshuqzoktjjlooqfenvm.supabase.co/storage/v1/object/public/${folde
                                r}/${data.path}`;
                                } else {
       throw new Error("Failed to get file URL after uploading ebook form
                                 data.");
                                    }
                                    };

         // Upload the ebook file and cover page to Supabase Storage
           const fileUrl = await uploadToSupabase(file, 'ebooks');
    const coverPageUrl = await uploadToSupabase(coverPage, 'coverPages');

              // Save the ebook information to the database
                const ebook = await prisma.ebook.create({
                                 data: {
                                   title,
```

```
                author,
              description,
              publishedAt,
                fileUrl,
              coverPageUrl,
                },
                });

            // Success message
return NextResponse.json({ message: "Ebook added successfully!", ebook
                });
              } catch {
            // Error message
return NextResponse.json({ error: "Something went wrong" }, { status:
        500 }); // Status code 500 for internal server error
                }
                }
```

Fetch Ebooks API

```
        // Import NextResponse from 'next/server'
       import { NextResponse } from 'next/server';

      // Import prisma to interact with the database
        import { prisma } from '../../../lib/db';

              // GET function
            export async function GET() {
// Fetch all ebooks from the database using Prisma's findMany method
```

```
const ebooks = await prisma.ebook.findMany();


  // Return ebooks data as a JSON response

     return NextResponse.json(ebooks);

                          }
```

User Profile API

```
// Import Prisma client to interact with the database

      import { prisma } from '../../../lib/db';


                  // GET function

      export async function GET(req: Request) {

                // Get the request URL

            const url = new URL(req.url);


         // Get 'userId' from the query string

       const userId = url.searchParams.get('userId');


// If 'userId' is missing, return a 400 error status code

                    if (!userId) {

return new Response(JSON.stringify({ error: 'User ID is required' }),
                          {

        status: 400, // Status code 400 for bad request

                          });

                          }


// Query the database to find the user by 'userId' and select the
                    'profilePicture'
```

```
            const user = await prisma.user.findUnique({
          where: { id: parseInt(userId) }, // Find user by 'id'
    select: { profilePicture: true }, // Only select the 'profilePicture'
                                field
                                  });


     // If user is found return profile data, if not return null value
                            return new Response(
                  JSON.stringify({ user: user || null }),
                  { status: 200 } // Successfully updated
                                  );
                                  }
```

Login API

```
 import { NextResponse } from 'next/server'; // Import NextResponse from
                              'next/server'
import { prisma } from '@/lib/db'; // Import Prisma client for interacting
                            with the database
    import bcrypt from 'bcryptjs'; // Import for hashing and comparing
                                passwords
 import jwt from 'jsonwebtoken'; // Import for creating and verifying JSON
                                Web Tokens


                            // POST function
             export async function POST(request: Request) {
                                  try {
      // Parse incoming JSON request data to get the email and password
              const { email, password } = await request.json();
```

```
    // Validate and ensure that both email and password are provided
                    if (!email || !password) {
    return NextResponse.json({ error: 'Email and password are required'
       }, { status: 400 }); // Status code 400 for bad request
                                }

        // Find the user by the provided email in the database
              const user = await prisma.user.findUnique({
                          where: { email },
                                });

      // If no user is found with the given email, return an error
                            if (!user) {
    return NextResponse.json({ error: 'Invalid email or password' }, {
         status: 400 }); // Status code 400 for bad request
                                }

       // Compare the provided password with the hashed password
const isPasswordValid = await bcrypt.compare(password, user.password);

                    if (!isPasswordValid) {
    return NextResponse.json({ error: 'Invalid email or password' }, {
         status: 400 }); // Status code 400 for bad request
                                }

        // Fetch JWT secret from environment variable .env file
              const jwtSecret = process.env.JWT_SECRET;
```

```
                    if (!jwtSecret) {
            // If JWT secret is missing, throw an error
    throw new Error("JWT_SECRET is not defined in the environment
            variables. Please check the .env file");
                                }

  // Generate a JWT token with user details if authentication is
                            successful
                        const token = jwt.sign(
                                 {
                              userId: user.id,
                            isAdmin: user.isAdmin,
                           username: user.username,
                              email: user.email,
                         profilePicture: user.profilePicture,
                                 },
                jwtSecret, // Secret key to encode the token
                { expiresIn: '1h' } // Token expiration time
                                );

 // Return a success message with the token and user's admin status
                        return NextResponse.json({
                       message: 'Login successful!',
                                 token,
                            isAdmin: user.isAdmin,
                        profilePicture: user.profilePicture,
                                });
```

```
    } catch {
        // Return error message
        return NextResponse.json({ error: 'Something went wrong' }, { status: 500 }); // Status code 500 for internal server error
    }
}
```

Register API

```
import { NextResponse } from 'next/server'; // Import NextResponse from 'next/server'
import { prisma } from '@/lib/db'; // Import Prisma client for interacting with the database
import bcrypt from 'bcryptjs'; // Import for hashing and comparing passwords

// POST function
export async function POST(request: Request) {
    try {
        // Parse the request data
        const body = await request.json();

        // Extract username, email, and password
        const { username, email, password } = body;

        // Validate that username, email, and password are provided
        if (!username || !email || !password) {
            return NextResponse.json({ error: "Username, email, and password are required" }, { status: 400 }); // Status code 400 for bad request
```

```
                                    }


    // Check if the email or username already exists in the database
            const existingUser = await prisma.user.findFirst({
                                    where: {
                                      OR: [
                        { email }, // Check if email exists
                    { username }, // Check if username exists
                                       ],
                                     },
                                    });


       // If either the email or username is taken, return an error
                                if (existingUser) {
    return NextResponse.json({ error: "Email or username already exists"
       }, { status: 400 }); // Status code 400 for bad request
                                    }


         // Hash the password before saving it to the database
         const hashedPassword = await bcrypt.hash(password, 10);


        // Create the new user in the User table in the database
                        await prisma.user.create({
                                    data: {
                                    username,
                                      email,
                                  password: hashedPassword,
                                     },
```

```
    });

    // Return a success message
    return NextResponse.json({ message: "Registration successful!" });

  } catch {
    // Return error message
    return NextResponse.json({ error: "Something went wrong" }, { status:
      500 }); // Status code 500 for internal server error
  }
}
```

Ebook Reader API

```
// Import NextRequest and NextResponse from 'next/server' and import
    necessary modules
import { NextRequest, NextResponse } from "next/server";
import { PrismaClient } from "@prisma/client";

// Import prisma to interact with the database
const prisma = new PrismaClient();

// GET function
export async function GET(req: NextRequest) {
  // Get the query parameters from the URL of the request
  const { searchParams } = new URL(req.url);
  const id = searchParams.get("id");

  // Check if id is a valid number
```

```
const parsedId = parseInt(id ?? '', 10);  // Convert id to an integer,
                  defaulting to NaN if id is null

                        if (isNaN(parsedId)) {

                       return NextResponse.json(

    { error: "Invalid 'id' parameter" },  // Error message for invalid
                                    'id'

           { status: 400 }  // Status code 400 for bad request

                                    );

                                    }


   // Fetch the ebook data from the database using the provided id

            const ebook = await prisma.ebook.findUnique({

 where: { id: parsedId },  // Query the database for the ebook with the
                                  given id

                                    });


         // If the ebook is not found, return error message

                                if (!ebook) {

                       return NextResponse.json(

                    { error: "Ebook not found" },

            { status: 404 }  // Status code 404 for not found

                                    );

                                    }


                // Return the ebook details if found

                     return NextResponse.json({

                      id: ebook.id,  // Ebook ID

                  title: ebook.title,  // Ebook title
```

```
            description: ebook.description,  // Ebook description
              fileUrl: ebook.fileUrl,  // URL to the ebook file
          coverPageUrl: ebook.coverPageUrl,  // URL to the cover page image
                                    });
                                    }
```

User Profile Picture API

```
  import { prisma } from '../../../lib/db'; // Import Prisma client to
                        interact with the database
import jwt from 'jsonwebtoken'; // Import for creating and verifying JSON
                                Web Tokens


            // Define an interface for the DecodedToken object
                        interface DecodedToken {
          userId: number; // User ID extracted from the decoded token
                                    }


                            // POST function
                export async function POST(req: Request) {
           // Extract profile picture URL and token from the request
              const { profilePicture, token } = await req.json();


                        // Decode the JWT token
            const decodedToken = jwt.decode(token) as DecodedToken;


            // Check if the token is valid and contains a userId
                if (!decodedToken || !decodedToken.userId) {
          return new Response(JSON.stringify({ error: 'Unauthorized' }), {
```

```
            status: 401, // Unauthorized if token is invalid
                                });
                                }


                              try {
          // Update the user's profile picture in the database
             const updatedUser = await prisma.user.update({
           where: { id: decodedToken.userId }, // Find user by ID
                              data: {
               profilePicture, // Set new profile picture URL
                                },
                                });


                        // Success message
                      return new Response(
    JSON.stringify({ message: 'Profile picture updated', user:
                        updatedUser }),
                                {
                   status: 200, // Successfully updated
                                }
                               );
                            } catch {
                      // Return error message
                        return new Response(
        JSON.stringify({ error: 'Failed to update profile picture' }),
         { status: 500 } // Status code 500 for internal server error
                               );
                               }
```

```
}
```

Visualisations API

```
import { NextResponse } from "next/server"; // Import NextResponse from
'next/server'
import prisma from "../../../lib/prisma"; // Import Prisma client to
interact with the database


// GET function
export async function GET(request: Request) {
// Parse the request URL to access query parameters
const url = new URL(request.url);


// Get page and limit from query parameters
const page = parseInt(url.searchParams.get("page") || "1", 10); //
Default to page 1
const limit = parseInt(url.searchParams.get("limit") || "50", 10); //
Default to 50 items per page
const offset = (page - 1) * limit; // Calculate how many records to skip
based on the current page and the limit per page


try {
// Fetch books, sorted by download count
const books = await prisma.books_book.findMany({
select: {
title: true, // Select only the 'title' field
download_count: true, // Select the 'download_count' field
},
```

```
                orderBy: {
        download_count: 'desc', // Order books by download count in
                        descending order
                    },
    skip: offset, // Skip the records from previous pages based on the
                        offset
                take: limit, // Get 'limit' number of books
                    });


                // Get the total number of books
            const totalBooks = await prisma.books_book.count();


      // Return the books data, total count, and page information
                    return NextResponse.json({
                  data: books, // Array of fetched books
                total: totalBooks, // Total number of books
                    page, // Current page number
     totalPages: Math.ceil(totalBooks / limit), // Calculate the total
                        number of pages
                    });
                } catch {
                // Return error message
  return NextResponse.json({ error: 'An unexpected error occurred' });
                    }
                }
```

GEOJSON

The map of Ireland feature is populated with marker pins that display relevant information on the selected library. This is achieved using Geojson files taken from the https://data.gov.ie/dataset/?q=libraries&api=true&sort=score+desc%2C+metadata_created+desc

This is sample of the GEOJSON dataset utilised in the project:


```
{
  "type" : "FeatureCollection",
  "features" : [
    {
      "type" : "Feature",
      "id" : 1,
      "geometry" : {
        "type" : "Point",
        "coordinates" : [
          -6.1754458622377966,
          53.300837391838719
        ]
      },
      "properties" : {
        "OBJECTID" : 1,
        "OBJECTID_1" : 6,
        "Name" : "Blackrock",
        "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
        "BUILDING_N" : "BLACKROCK LIBRARY",
        "EIRCODE" : "A94YF76",
```

```
      "Address1" : "BLACKROCK LIBRARY",
      "ADDR_LINE1" : "48 MAIN STREET",
      "Town" : "BLACKROCK",
      "County" : "CO. DUBLIN",
      "ITM_EAST" : 721614.60400000005,
      "ITM_NORTH" : 729391.20999999996,
      "long" : -6.1754458620000001,
      "lat" : 53.300837389999998,
      "Phone" : "(01) 2888117",
      "Email" : "blackrocklib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 2,
    "geometry" : {
      "type" : "Point",
      "coordinates" : [
        -6.149560986395195,
        53.259856091286352
      ]
    },
    "properties" : {
      "OBJECTID" : 2,
      "OBJECTID_1" : 7,
      "Name" : "Cabinteely",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
```

```
      "BUILDING_N" : "CABINTEELY LIBRARY",
      "EIRCODE" : "D18W773",
      "Address1" : "Old Bray Road",
      "ADDR_LINE1" : "Cabinteely",
      "Town" : "Dublin 18",
      "County" : "CO. DUBLIN",
      "ITM_EAST" : 723457.84860000003,
      "ITM_NORTH" : 724876.2108,
      "long" : -6.149560986,
      "lat" : 53.25985609,
      "Phone" : "(01) 2855363",
      "Email" : "cabinteelylib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 3,
    "geometry" : {
      "type" : "Point",
      "coordinates" : [
        -6.104578237942925,
        53.277326488647063
      ]
    },
    "properties" : {
      "OBJECTID" : 3,
      "OBJECTID_1" : 5,
```

```
      "Name" : "Dalkey",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "BUILDING_N" : "DALKEY BRANCH LIBRARY",
      "EIRCODE" : "A96AK28",
      "Address1" : "DALKEY BRANCH LIBRARY",
      "ADDR_LINE1" : "40/41 CASTLE STREET",
      "Town" : "DALKEY",
      "County" : "CO. DUBLIN",
      "ITM_EAST" : 726407.15599999996,
      "ITM_NORTH" : 726898.49899999995,
      "long" : -6.1045782380000002,
      "lat" : 53.27732649,
      "Phone" : "(01) 2855317 / 2855277",
      "Email" : "dalkeylib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 4,
    "geometry" : {
      "type" : "Point",
      "coordinates" : [
        -6.1646566118279349,
        53.277503711984188
      ]
    },
    "properties" : {
```

```
      "OBJECTID" : 4,
      "OBJECTID_1" : 8,
      "Name" : "Deansgrange",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "BUILDING_N" : "DEANSGRANGE LIBRARY",
      "EIRCODE" : "D18NY58",
      "Address1" : "Clonkeen Drive",
      "ADDR_LINE1" : "Deansgrange ",
      "Town" : "Dublin 18",
      "County" : "CO. DUBLIN",
      "ITM_EAST" : 722400.37210000004,
      "ITM_NORTH" : 726813.63210000005,
      "long" : -6.1646566119999999,
      "lat" : 53.277503709999998,
      "Phone" : "(01) 2850860",
      "Email" : "deansgrangelib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 5,
    "geometry" : {
      "type" : "Point",
      "coordinates" : [
        -6.2468782302331674,
        53.292656190729183
      ]
```

```
    },
    "properties" : {
      "OBJECTID" : 5,
      "OBJECTID_1" : 1,
      "Name" : "Dundrum",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "BUILDING_N" : "CARNEGIE LIBRARY",
      "EIRCODE" : "D14VP97",
      "Address1" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "ADDR_LINE1" : "CARNEGIE LIBRARY",
      "Town" : "CHURCHTOWN ROAD UPPER",
      "County" : "DUBLIN 14",
      "ITM_EAST" : 716876.08900000004,
      "ITM_NORTH" : 728361.77899999998,
      "long" : -6.2468782300000001,
      "lat" : 53.292656190000002,
      "Phone" : "(01) 2985000",
      "Email" : "dundrumlib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 6,
    "geometry" : {
      "type" : "Point",
      "coordinates" : [
        -6.1270298745477891,
```

```
        53.233535674954254
      ]
    },
    "properties" : {
      "OBJECTID" : 6,
      "OBJECTID_1" : 3,
      "Name" : "Shankill",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "BUILDING_N" : "SHANKILL LIBRARY",
      "EIRCODE" : "D18DX43",
      "Address1" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "ADDR_LINE1" : "SHANKILL LIBRARY",
      "Town" : "LIBRARY ROAD",
      "County" : "DUBLIN 18",
      "ITM_EAST" : 725037.65399999998,
      "ITM_NORTH" : 721987.10999999999,
      "long" : -6.1270298749999998,
      "lat" : 53.233535670000002,
      "Phone" : "(01) 2823081",
      "Email" : "shankilllib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 7,
    "geometry" : {
      "type" : "Point",
```

```
      "coordinates" : [
        -6.1966525279151421,
        53.289817324132031
      ]
    },
    "properties" : {
      "OBJECTID" : 7,
      "OBJECTID_1" : 4,
      "Name" : "Stillorgan",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "BUILDING_N" : "STILLORGAN PUBLIC LIBRARY",
      "EIRCODE" : "A94XT02",
      "Address1" : "STILLORGAN PUBLIC LIBRARY",
      "ADDR_LINE1" : "SAINT LAURENCE'S PARK",
      "Town" : "STILLORGAN",
      "County" : "CO. DUBLIN",
      "ITM_EAST" : 720232.16799999995,
      "ITM_NORTH" : 728129.27099999995,
      "long" : -6.1966525280000004,
      "lat" : 53.289817319999997,
      "Phone" : "(01) 2889655",
      "Email" : "stillorganlib@dlrcoco.ie"
    }
  },
  {
    "type" : "Feature",
    "id" : 8,
```

```
    "geometry" : {
      "type" : "Point",
      "coordinates" : [
        -6.1317844492839653,
        53.292912776865442
      ]
    },
    "properties" : {
      "OBJECTID" : 8,
      "OBJECTID_1" : 2,
      "Name" : "DLR LEXICON",
      "ORGANISATI" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "BUILDING_N" : "THE DLR LEXICON LIBRARY",
      "EIRCODE" : "A96H283",
      "Address1" : "DÚN LAOGHAIRE-RATHDOWN COUNTY COUNCIL",
      "ADDR_LINE1" : "THE DLR LEXICON LIBRARY",
      "Town" : "HAIGH TERRACE",
      "County" : "DUN LAOGHAIRE",
      "ITM_EAST" : 724547.59600000002,
      "ITM_NORTH" : 728584.77000000002,
      "long" : -6.1317844490000004,
      "lat" : 53.292912780000002,
      "Phone" : "(01) 2147975",
      "Email" : "dlrlexiconlib@dlrcoco.ie"
    }
  }
]
```

```
}
```

The GEOJSON properties are declared inside the map component like this:

```
// Define data types for GeoJSON feature properties and geometry
interface LibraryProperties {
  Name: string;
  Address1: string;
  Town: string;
  County: string;
  Eircode: string;
  Phone: string;
  Email: string;
  Website: string;
  Opening_Hours_Monday: string;
  Opening_Hours_Tuesday: string;
  Opening_Hours_Wednesday: string;
  Opening_Hours_Thursday: string;
  Opening_Hours_Friday: string;
  Opening_Hours_Saturday: string;
}

interface LibraryGeometry {
  coordinates: [number, number]; // [longitude, latitude]
}
```

The GEOJSON files are stored inside the map component using an array:

```
// Array of GeoJSON file paths to load library locations
  const geojsonFiles = [
    '/geojson/wexford_libraries.geojson',
```

```
    '/geojson/Libraries_Roscommon.geojson',
    '/geojson/Wicklow_Libraries_WIW.geojson',
    '/geojson/librarylocationsdlr.geojson',
    '/geojson/Library_Services_SDCC.geojson',
    '/geojson/Libraries_FCC.geojson',
    '/geojson/GCC_Libraries.geojson',
    '/geojson/cork-city-council-libraries.geojson',
  ];
```

The GEOJSON files are then looped through and assigned their key value pair properties and this is how the map of Ireland feature is created and displayed:

```
useEffect(() => {
    // Initialize the map when the component loads
    if (mapRef.current && !mapInstance.current) {
      // Create a Leaflet map instance and set the initial view and zoom
level
      mapInstance.current = L.map(mapRef.current).setView([52.655778,
-6.65652], 8);

      // Add OpenStreetMap tile layer to the map
      L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; OpenStreetMap contributors',
      }).addTo(mapInstance.current);

      // Function to load and process GeoJSON data
      const loadGeojson = (filePath: string) => {
        fetch(filePath) // Fetch GeoJSON file
          .then((response) => response.json()) // Parse the response as
JSON
```

```
          .then((geojsonData: { features: LibraryFeature[] }) => {
            // Iterate through each feature in the GeoJSON data
            geojsonData.features.forEach((feature) => {
              const { coordinates } = feature.geometry; // Extract
coordinates (longitude, latitude)
              const {
                Name,
                Address1,
                Town,
                County,
                Eircode,
                Phone,
                Email,
                Website,
                Opening_Hours_Monday,
                Opening_Hours_Tuesday,
                Opening_Hours_Wednesday,
                Opening_Hours_Thursday,
                Opening_Hours_Friday,
                Opening_Hours_Saturday,
              } = feature.properties; // Extract other properties, the
library details

              // Create popup content box with relevant library details
              const popupContent = `
                <div style="overflow-y: scroll; max-width: 200px;
max-height: 200px; word-wrap: break-word; padding: 12px;">
```

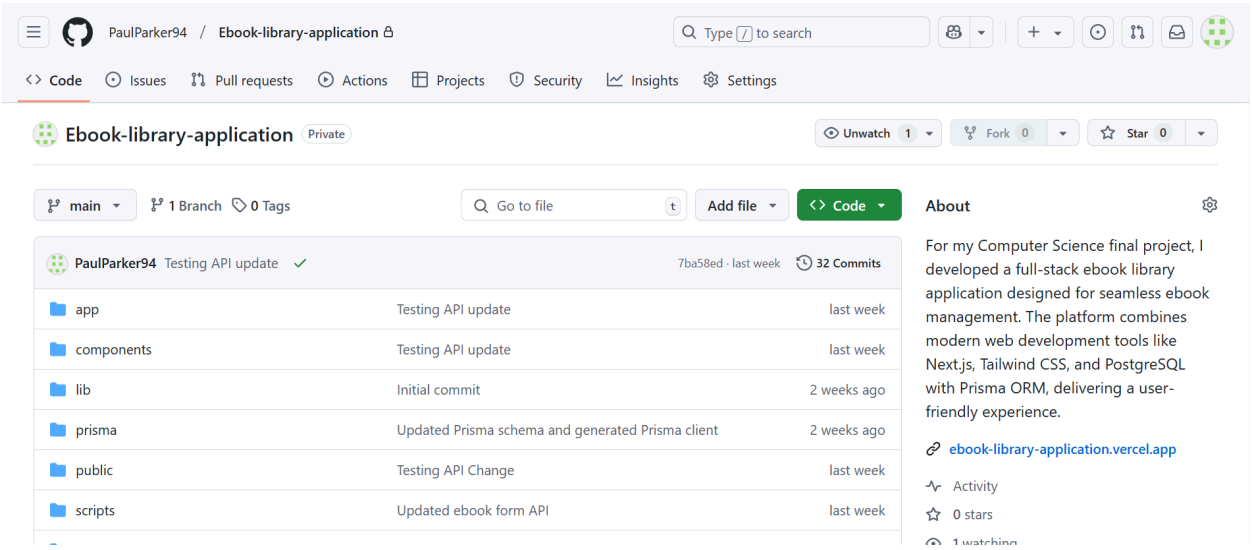
```
                <h3 style="text-decoration: underline; font-size: 15px;
font-weight: bold;">${Name}</h3>

                <p><strong>Address:</strong> ${Address1}, ${Town},
${County}, ${Eircode}</p>

                <p><strong>Phone:</strong> ${Phone}</p>

                <p><strong>Email:</strong> <a
href="mailto:${Email}">${Email}</a></p>

                <p><strong>Website:</strong> <a href="${Website}"
target="_blank">${Website}</a></p>

                <p><strong>Opening Hours (Monday):</strong>
${Opening_Hours_Monday}</p>

                <p><strong>Opening Hours (Tuesday):</strong>
${Opening_Hours_Tuesday}</p>

                <p><strong>Opening Hours (Wednesday):</strong>
${Opening_Hours_Wednesday}</p>

                <p><strong>Opening Hours (Thursday):</strong>
${Opening_Hours_Thursday}</p>

                <p><strong>Opening Hours (Friday):</strong>
${Opening_Hours_Friday}</p>

                <p><strong>Opening Hours (Saturday):</strong>
${Opening_Hours_Saturday}</p>

                <p><strong>Opening Hours (Sunday):</strong> Closed</p>

                <p><strong>Opening Hours (Bank Holidays):</strong>
Closed</p>

              </div>

            `;


            // Add a marker on the map for each feature's coordinates

            L.marker([coordinates[1], coordinates[0]]) // Create a
marker at the given coordinates

              .addTo(mapInstance.current!) // Add marker to the map
```

```
                .bindPopup(popupContent); // Bind the popup content to the
marker

            });

          })

      };


      // Load all the GeoJSON files

      geojsonFiles.forEach((file) => {

        loadGeojson(file);

      });

    }
```

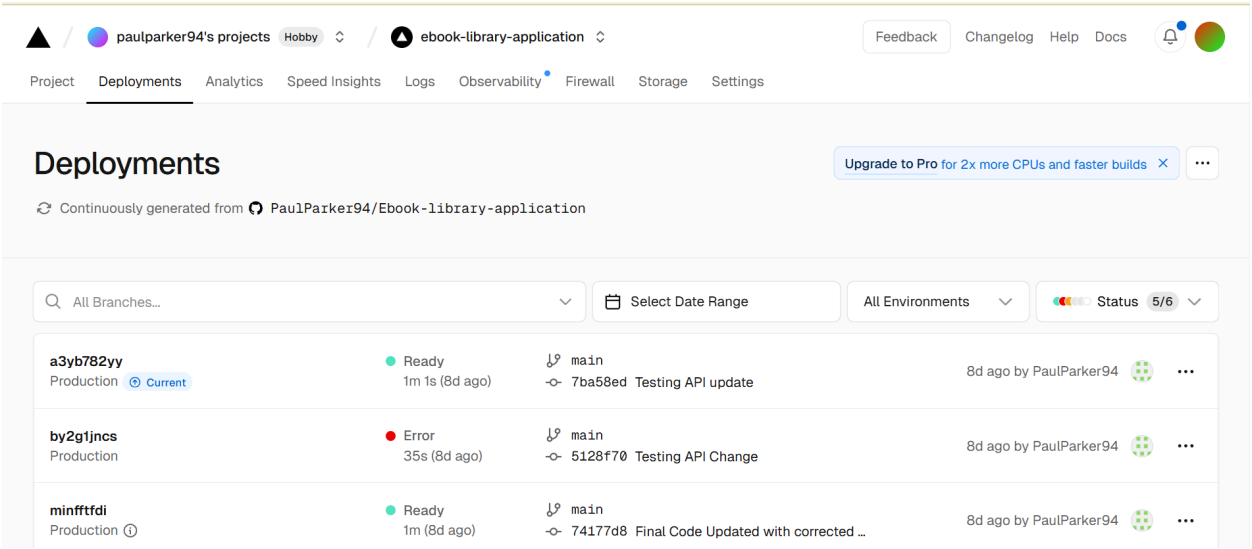
Production

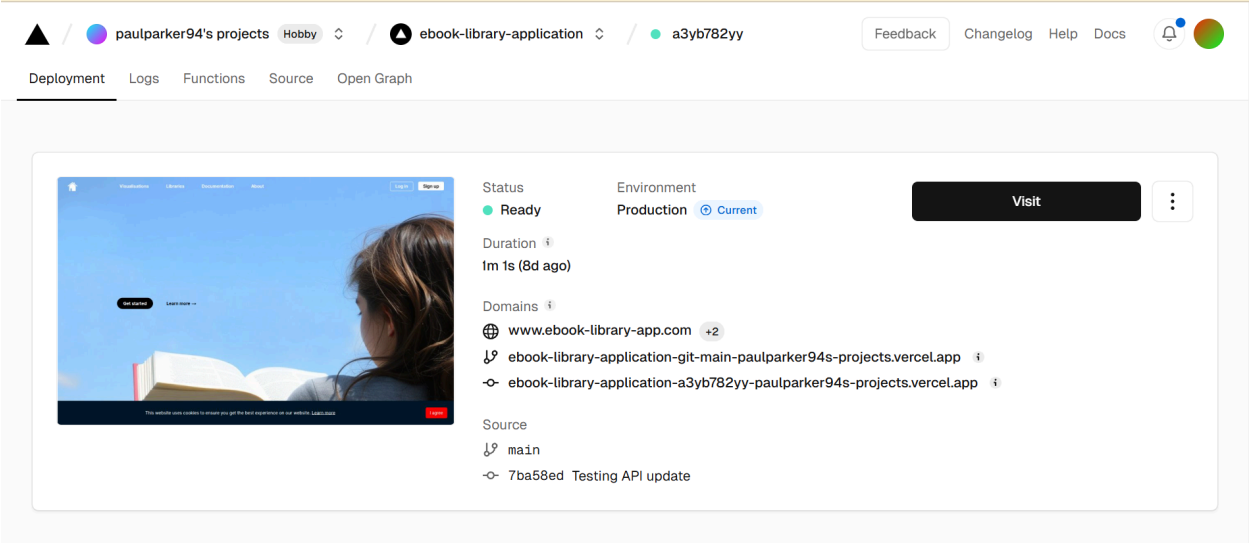
Once the project was completed in the local development environment, the next step was to push it to production. To achieve this I used a Github repository to push my code to from my visual studio code using the following commands inside the command terminal.

1. git add .
2. git commit -m “your_message_here”
3. git push origin main



Then I created and linked my Vercel account to my Github account. This allowed me to automatically deploy the project everytime I pushed the code to github.





The next step was to create and link my Supabase database to my Vercel project. This allowed me to autogenerate my environmental variables which allowed my project to communicate to my database. I also added my own environmental variables such as my JWT token.

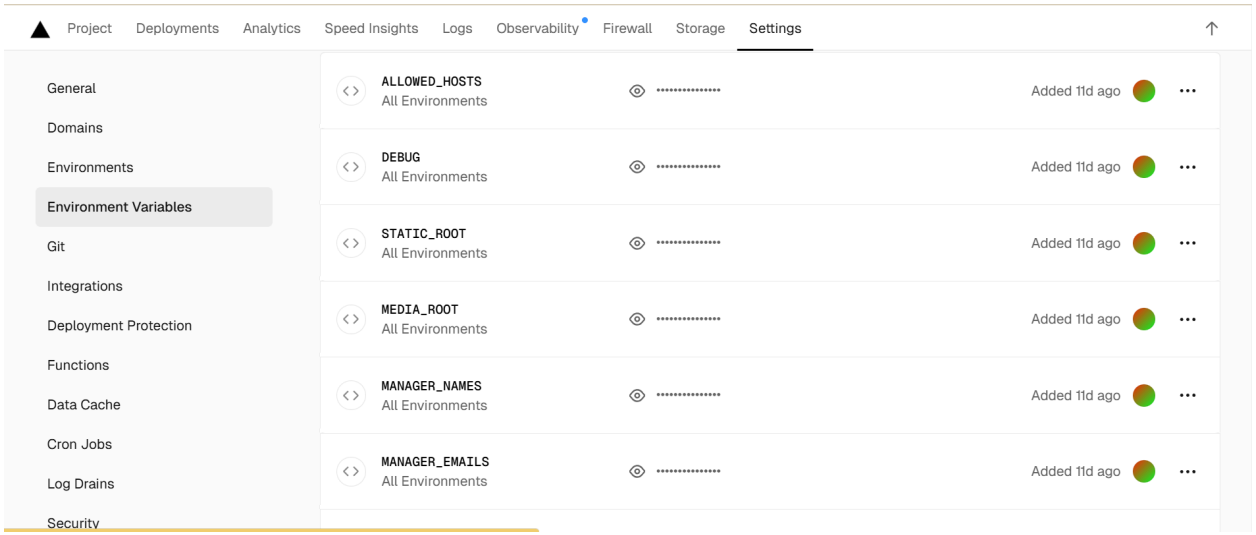
I used ‘npx prisma migrate deploy’ to migrate my schema to my Supabase database, by changing the DATABASE_URL value inside my environmental variables to the Supabase connection API string it now could communicate directly with my Supabase Database

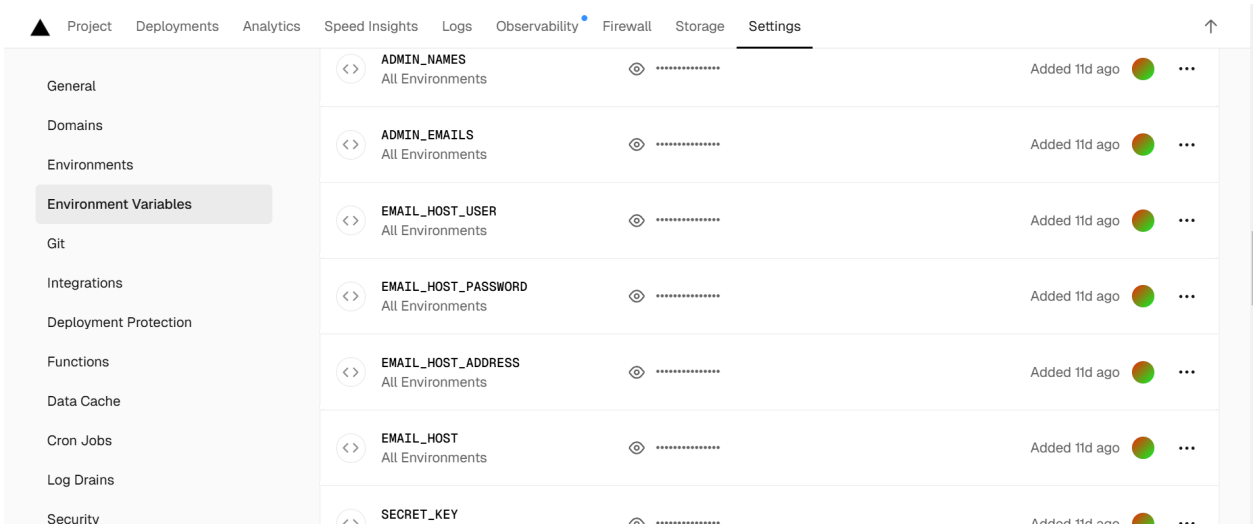
I used the following command inside my Visual studio terminal to pull the data from my local production postgres database:

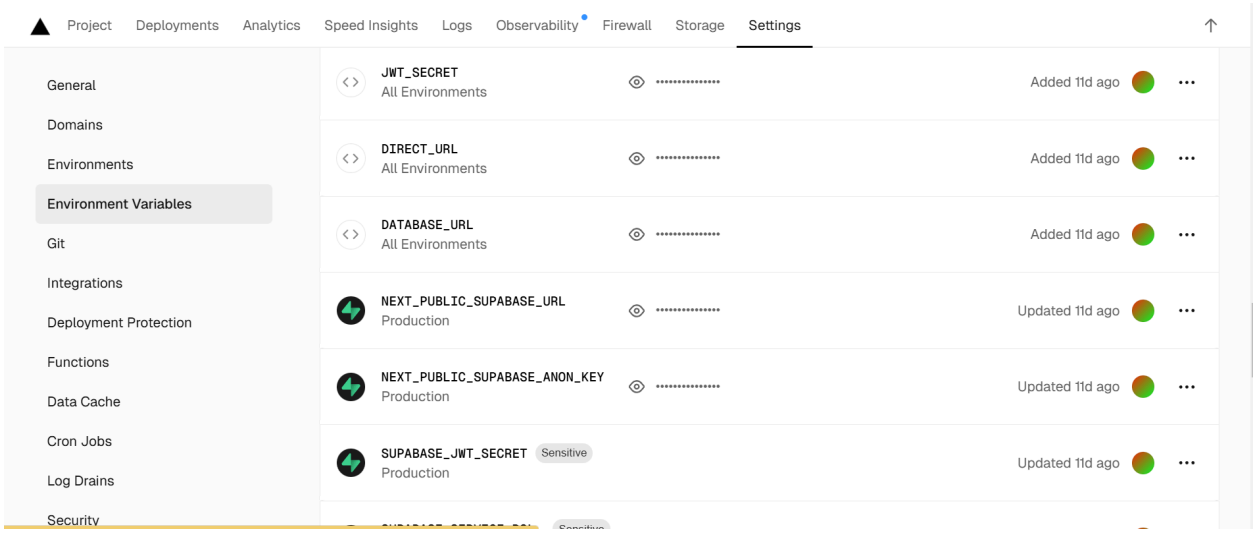
pg_dump --username=your_username --host=your_host --port=your_port --file=database_backup.sql your_database_name -replace placeholder values with relevant details.

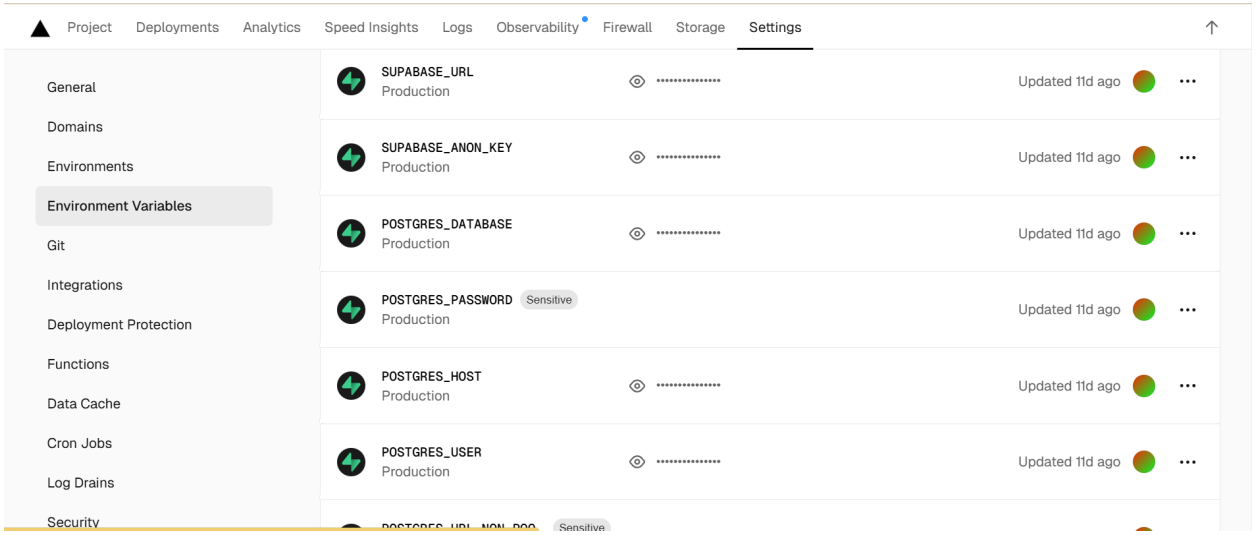
This created a sql file with all the data which I then pushed to my Supabase database from my terminal inside my visual studio code, which now emulated my local production database.

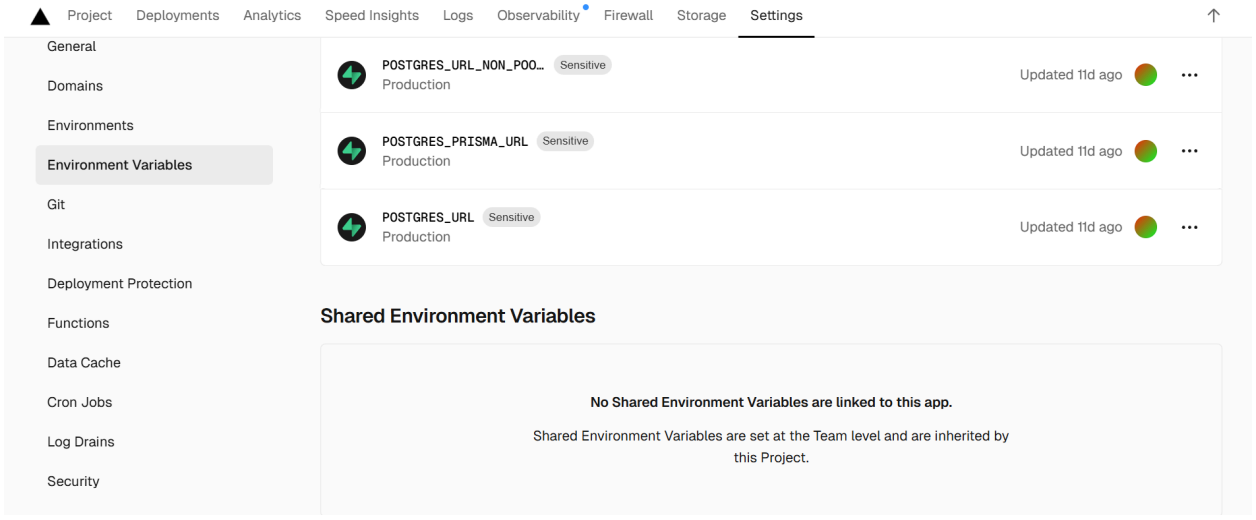
psql --host=supabase_host --port=5432 --username=supabase_user --dbname=supabase_db_name --file=data.sql - replace placeholder values with relevant details.


Project
Deployments
Analytics
Speed Insights
Logs
Observability
Firewall
Storage
Settings
General
Domains
Environments
Environment Variables
Git
Integrations
Deployment Protection
Functions
Data Cache
Cron Jobs
Log Drains
Security
ALLOWED_HOSTS
All Environments
DEBUG
All Environments
STATIC_ROOT
All Environments
MEDIA_ROOT
All Environments
MANAGER_NAMES
All Environments
MANAGER_EMAILS
All Environments
Added 11d ago

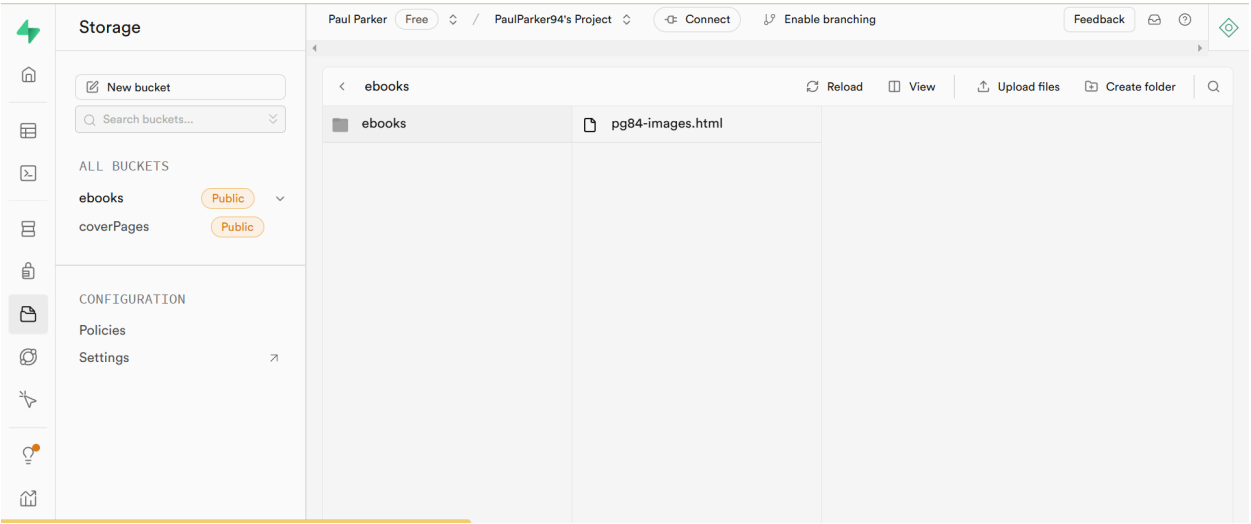

Project
Deployments
Analytics
Speed Insights
Logs
Observability
Firewall
Storage
Settings
General
Domains
Environments
Environment Variables
Git
Integrations
Deployment Protection
Functions
Data Cache
Cron Jobs
Log Drains
Security
ADMIN_NAMES
All Environments
ADMIN_EMAILS
All Environments
EMAIL_HOST_USER
All Environments
EMAIL_HOST_PASSWORD
All Environments
EMAIL_HOST_ADDRESS
All Environments
EMAIL_HOST
All Environments
SECRET_KEY
Added 11d ago


Project
Deployments
Analytics
Speed Insights
Logs
Observability
Firewall
Storage
Settings
General
Domains
Environments
Environment Variables
Git
Integrations
Deployment Protection
Functions
Data Cache
Cron Jobs
Log Drains
Security
JWT_SECRET
All Environments
Added 11d ago
DIRECT_URL
All Environments
Added 11d ago
DATABASE_URL
All Environments
Added 11d ago
NEXT_PUBLIC_SUPABASE_URL
Production
Updated 11d ago
NEXT_PUBLIC_SUPABASE_ANON_KEY
Production
Updated 11d ago
SUPABASE_JWT_SECRET
Sensitive
Production
Updated 11d ago


Project
Deployments
Analytics
Speed Insights
Logs
Observability
Firewall
Storage
Settings
General
Domains
Environments
Environment Variables
Git
Integrations
Deployment Protection
Functions
Data Cache
Cron Jobs
Log Drains
Security
SUPABASE_URL
Production
Updated 11d ago
SUPABASE_ANON_KEY
Production
Updated 11d ago
POSTGRES_DATABASE
Production
Updated 11d ago
POSTGRES_PASSWORD
Sensitive
Production
Updated 11d ago
POSTGRES_HOST
Production
Updated 11d ago
POSTGRES_USER
Production
Updated 11d ago

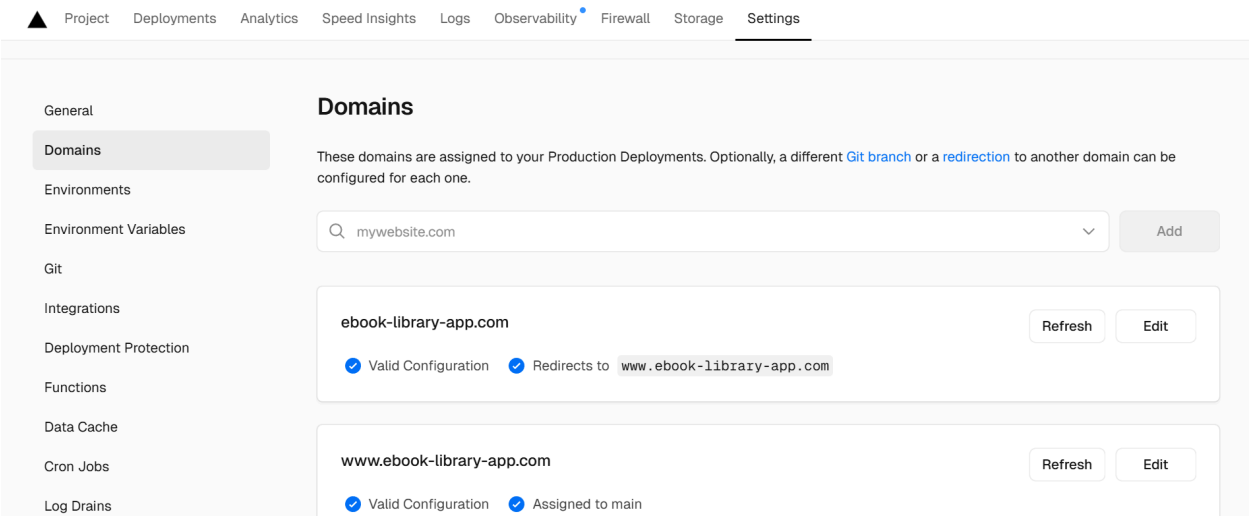

Project
Deployments
Analytics
Speed Insights
Logs
Observability
Firewall
Storage
Settings
General
Domains
Environments
Environment Variables
Git
Integrations
Deployment Protection
Functions
Data Cache
Cron Jobs
Log Drains
Security
POSTGRES_URL_NON_POO…
Sensitive
Production
Updated 11d ago
POSTGRES_PRISMA_URL
Sensitive
Production
Updated 11d ago
POSTGRES_URL
Sensitive
Production
Updated 11d ago
Shared Environment Variables
No Shared Environment Variables are linked to this app.
Shared Environment Variables are set at the Team level and are inherited by this Project.

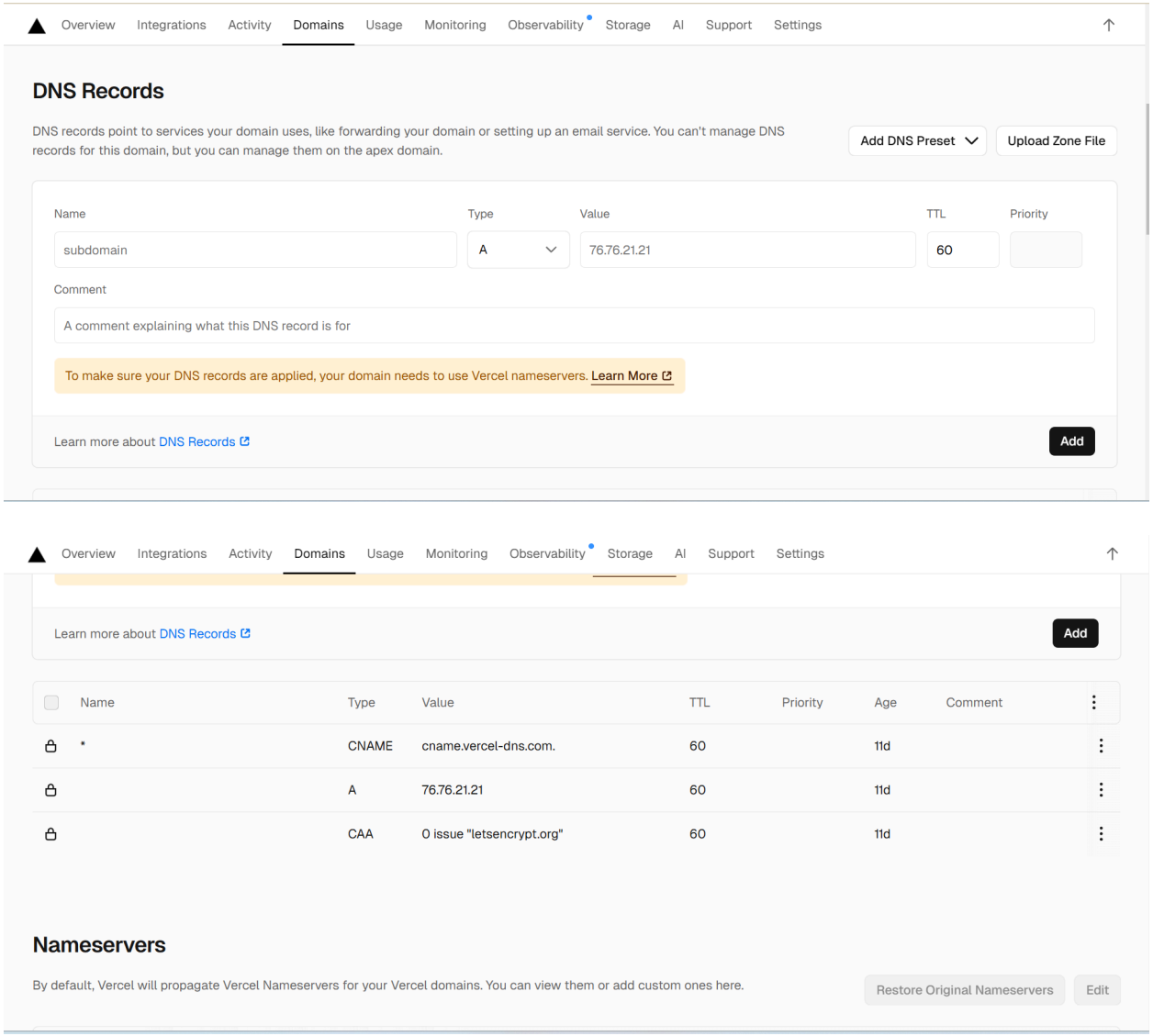
The next step was to create my storage buckets on my Supabase project to store the ebooks uploads.



Then I purchased a domain name from GoDaddy https://www.godaddy.com/en-ie/offers/godaddy?isc=sem3year&countryview=1&currencyType=EUR&cdtl=c_15415727694.g_138386971815.k_kwd-88659201.a_68462489351 0.d_c.ctv_g&bnb=b&gad_source=1&gclid=CjwKCAiAmrS7BhBJEiwAei59i-xz76_uStL aLNps4hsCKW0-fjbKHJ_CIFCGeySG6zb3tDXttvYMtxoCgGcQAvD_BwE

To link this to my Vercel project, I went to settings on my Project and selected the domains option. Then I set the DNS records to correspond with my GoDaddy values. After setting these values my project was fully deployed and live.





Heuristic Evaluation

Observation 1:

Problem: Review card overlaps with book cover images when viewing a smaller viewport.

Severity: Cosmetic: need not be fixed.

Heuristic: Consistency & standards. Strive for consistency. Aesthetic & minimalistic design

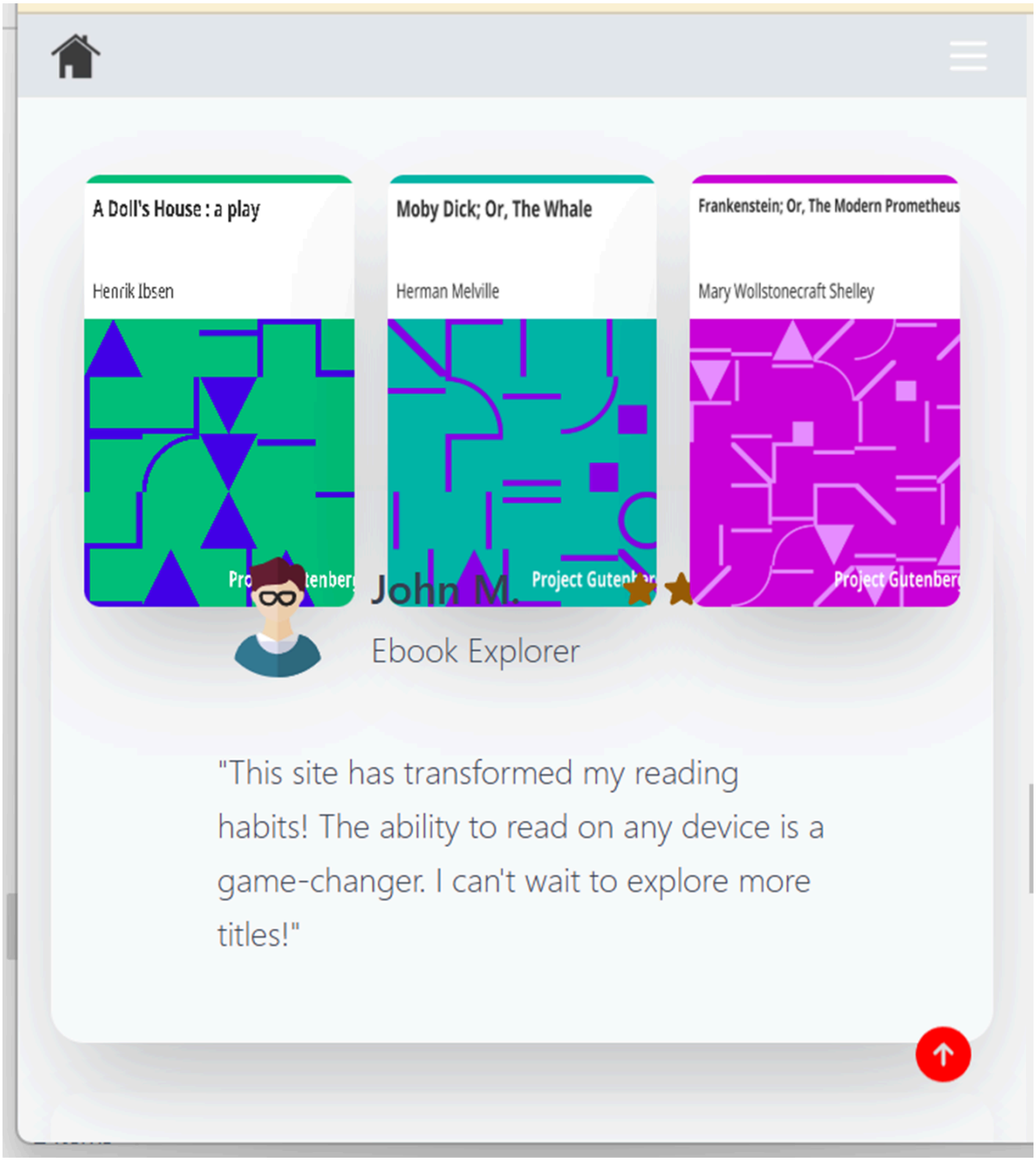
Description: Overlapping component elements, the margin-top value is causing the problem.

Recommendation: Tweak code of container div to ensure this doesn’t overlap:

From: mt-[-8rem]

To: mt-[-3rem].

Screenshot:



Observation 2:

Problem: Popup dialog box needs to be scaled for mobile viewports to ensure it's not too large.

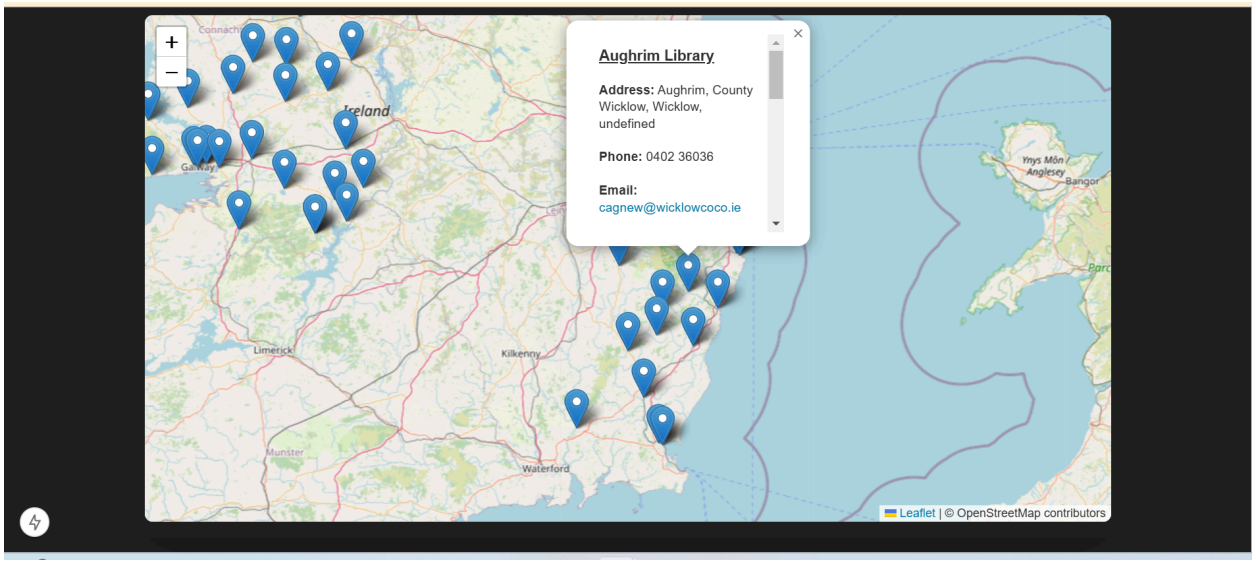
Severity: Cosmetic: need not be fixed.

Heuristic: Consistency & standards. Strive for consistency. User control & freedom. Aesthetic & minimalistic design. Support internal locus of control.

Description: Popup dialog is not scaled correctly for mobile viewports.

Recommendation: Add code to parent CSS class: style="overflow-y: scroll; max-width: 200px; max-height: 200px; word-wrap: break-word; padding: 12px;"

Screenshot:



Observation 3:

Problem: Value fields in the popup dialog box are returning null values.

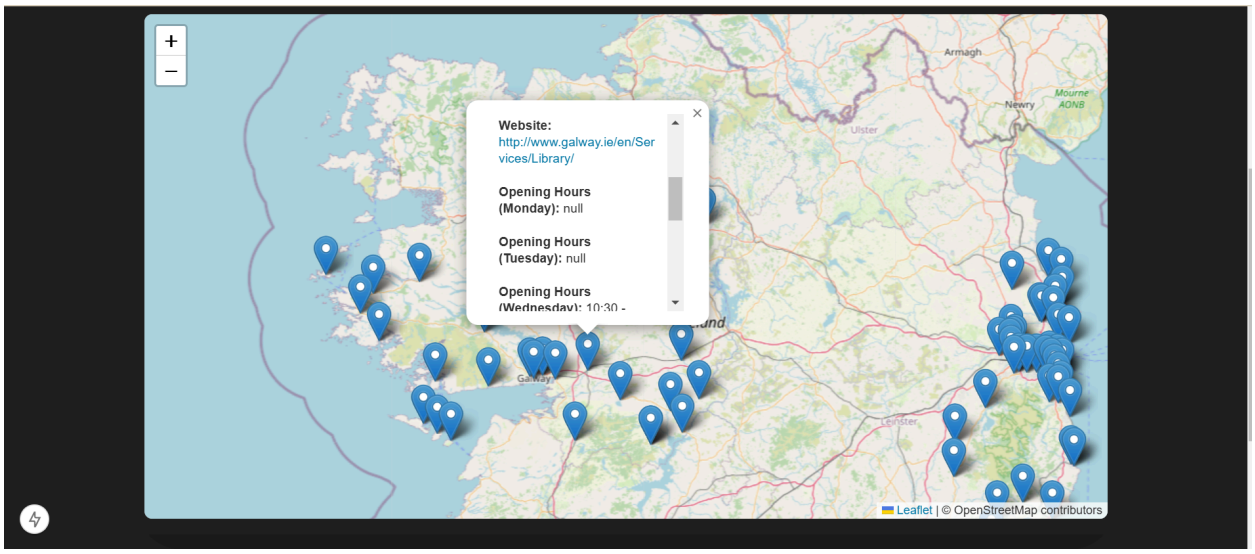
Severity: Minor: needs fixing but low priority.

Heuristic: Consistency & standards. Strive for consistency.

Description: Null value fields in the popup box are caused by undefined values inside the GEOJSON file.

Recommendation: Manually edit the GEOJSON files to ensure these null values are defined.

Screenshot:



Website Test Plan - Agile Methodology

1. **Introduction:** This document outlines the test plan for evaluating the user experience (UX), user interface (UI), and usability of the ebook-library-app website. The purpose is to identify areas of improvement to enhance user satisfaction and functionality.

2. **Business Background:** ebook-library-app is a website created for a final year Computer Science project. The website serves as a elearning platform to coexist with public resources for users to access a catalog of free ebooks. The aim of the platform is to improve adult literacy levels.

3. **Test Objectives:** The primary objectives of this test plan are to: Evaluate the UX/UI design elements of the website. Identify usability issues affecting user interaction and navigation. Provide recommendations for enhancing the overall user experience.

4. **Scope Inclusions:** Assessment of homepage layout, navigation, and content organization. Evaluation of map of Ireland markers ease of use. Analysis of responsiveness and compatibility across devices and browsers. Exclusions: Back-end functionality testing. Security vulnerabilities assessment (covered separately). In-depth performance analysis (covered separately).

5. **Test Types Identified:** UI Testing Usability Testing Compatibility Testing

6. **Problems Perceived:** Confusion between different map markers. Complexity in accessing the user dashboard. Inconsistencies in the map markers value fields.

7. **Architecture:** The website architecture consists of multiple pages interconnected through a navigation menu. It includes dynamic content sections such as visualisations page, and map of Ireland page.

8. **Environment:** The testing environment will include various devices (desktop, laptop, mobile) and browsers (Chrome, Firefox, Safari) to ensure compatibility.

9. **Assumptions:** The website is fully functional and accessible during the testing period. Testers have basic knowledge of UX/UI principles and testing methodologies.

10. **Functionality Constraints and Resolutions:** Constraint 1: Confusion between different map markers. Constraint 2: Inconsistencies in the map markers value fields. Resolution: Implement a unified value field for all map markers pins to ensure relevant information is displayed on every pin. Risk Identified & Mitigation Planned: Risk: Potential user frustration due to complex navigation. Mitigation: Simplify navigation structure and provide clear pathways to essential information.

11. **Security:** Security testing will be conducted separately to identify vulnerabilities and ensure data protection.

12. **Performance:** Performance testing will be conducted separately to assess website speed and responsiveness under different load conditions.

13. **Usability:** Constraints and Resolutions: Constraint 1: Complexity in accessing the user dashboard. Constraint 2: Inconsistent user interface elements. Resolution: Streamline user navigation and standardise UI elements for consistency. Risk Identified & Mitigation Planned: Risk: User dissatisfaction due to poor usability. Mitigation: Implement user-centric design principles and conduct user testing for feedback.

14. **Compatibility:** Compatibility testing will ensure the website functions seamlessly across various devices and browsers.

15. **Test Team Organisation:** Testers will be assigned specific tasks related to UI/UX evaluation, usability testing, and compatibility testing.

16. **Schedule:** Testing activities will be conducted over a two-week period, with regular progress updates and reporting.

17. **Defects Classification Mechanism:** Type of Defects: Critical Major Minor Cosmetics Defects Logging and Status Changing Mechanism: Defects will be logged using a standardized template and tracked until resolution.

18. **Configuration Management:** Version control and configuration management will ensure consistency across testing environments.

19. **Release Criteria:** Criteria for website release will be defined based on the resolution of critical defects and satisfactory usability performance. This test plan provides a comprehensive framework for evaluating the ebook-library-app website's UX/UI design and usability aspects. It aims to address identified issues and enhance user satisfaction and experience.

Gutendex API Update Dynamic Data

Batch File:

```
@echo off

rem Replace 'C:\Relative\to_My\path\myApps\idle3.12.exe' Replace 'idle3.12' with the installed iteration of the IDLE executable if it's different. This directory path can be found by viewing where IDLE is installed on your device.

rem Replace 'C:\Relative\to_My\path\myfolder\updatecatalog.py' with the path to your directory to your Python script

rem I have left my relative paths as an example of how to correctly use the batch file.

set IDLE_EXECUTABLE="C:\Users\pauls\AppData\Local\Microsoft\WindowsApps\idle3.12.exe"

set SCRIPT_PATH="C:\Users\pauls\gutendex-master\books\management\commands\updatecatalog.py"


rem Run IDLE with the script
```

%IDLE_EXECUTABLE% -r %SCRIPT_PATH%

pause

Save this above file as Script_automation.bat

HOW TO AUTOMATE PYTHON SCRIPT TO RUN EACH DAY AT A SET TIME.

1.Automate the python script to run automatically each day at a set time by using the batch file 'Script_automation'.

2.First you need to change the relative paths within the batch file 'Script_automation'. Right click on the batch file and edit the file with free programs such as Notepad++ or Notepad. Follow the instructions within the batch file to achieve this.

3.To automate the script click start and type 'Task Scheduler' into the search and press enter.

4.This will launch 'Task Scheduler'. Click on 'Task Scheduler Library'.

5.Click on 'Create Basic Task' from the right navigational pane.

6.Give the task a name e.g., 'Python Gutendex Data Script' and a description if you desire.

7.Click next and choose your trigger, which in this case is 'Daily'.

8.Click next and choose a date for the script to start on and set the time that you wish for the script to run at. Set the value to '1' for the 'Recur every: Days' dialog box.

9.Click next and for Action choose Start a Program.

10.Click next and in Start a Program, type 'cmd.exe' into the Program/script section.

11.In the Add a argument section type 'start /c "C:\Relative\to_My\path\myfolder\Script_automation.bat"'. Replace 'C:\Relative\to_My\path\myfolder\' with the path to your directory.

12.In the Start in (optional) section type 'C:\Relative\to_My\path\myfolder\'. Replace 'C:\Relative\to_My\path\myfolder\' with the path to your directory.

13.Click next and in the Finish section tick the checkbox for 'Open the Properties dialog for this task when I click finish'.

14.Click Finish

15.The Properties dialog box will now open, move to the general view tab.

16.Under the security heading, click the radio button option for 'Run whether user is logged on or not'.

17.Next move to the Conditions view tab and look to the power heading.

18.'Start the task only if the computer is on AC power' and the subtask of 'Stop if the computer switches to battery power'. These features can be left on or off.

19.'Wake the computer to run this task'. This feature can be left on or off. Note this feature can be dependent on whether you have clicked the above condition of Start the task only if the computer is on AC power or 'Stop if the computer switches to battery power'.

20.Click OK to finish and close out the Properties dialog box.

21.Enter your password for the relevant user in the dialog box.

22.Next time you turn the computer on the batch file will run automatically each day at a set time. The computer must be powered on for the task to run if you have not clicked on the above condition of 'Wake the computer to run this task'.

After following these above steps the script will execute daily and update the Gutendex API catalog information and update the data in the Supabase database. This will ensure that the data displayed in the visualisations section of the website will show dynamic data that is up to date reflecting the latest updates from the Project Gutenberg website.

I would like to express my gratitude to Kevin Conway for his exceptional guidance and support throughout this project. I also extend my thanks to Tracy Cassells for her outstanding teaching in previous semesters.

Full Code Used For Project

```
// Import the components from the components directory
import Navbar from "../components/Navbar";
import Hero from "../components/Hero";
import { Footer } from "../components/Footer";


// Create and export the HomePage function
export default function HomePage() {
  return (
    // Flexbox layout container
    <div className="flex flex-col min-h-screen">
      {/* Navbar */}
      <Navbar />
      {/* Main content */}
      <main className="flex-grow">
        <Hero />
      </main>
      {/* Footer */}
      <Footer />
    </div>
  );
}


// Render the component client-side
"use client";
```

```
//Import React hooks and import necessary modules and components
import { useEffect, useState, useCallback } from "react";
import Navbar from "../../components/Navbar";
import BarChart from "../../components/BarChart";
import { Footer } from "../../components/Footer";
import {
  Menu,
  MenuHandler,
  MenuList,
  MenuItem,
  Button,
} from "@material-tailwind/react";

// Define the data fetched from the API
interface Point {
  title: string;
  download_count: number;
}

// Specify the format of the API data
interface ApiResponse {
  data: Point[];
}

const VisualisationsPage = () => {
  // State variables for points, sorted points, error, loading, and sort
order
```

```
  const [points, setPoints] = useState<Point[]>([]); // Ensure points
state is always an array
  const [sortedPoints, setSortedPoints] = useState<Point[]>([]);
  const [error] = useState<string>(""); // Error state to handle API
errors
  const [loading, setLoading] = useState<boolean>(true); // Loading state
to show loading message
  const [sortOrder, setSortOrder] = useState<string>("frequency"); //
Default sorting order


  // Handle sorting order change when user selects an option of Popular or
Alphabetical
  const handleSortChange = (order: string) => {
    setSortOrder(order);
  };


  // Sort the data based on selected order
  const sortData = useCallback((order: string) => {
    if (points.length === 0) return; // If no data is available return


    // Initialise new variable as empty array and store sorted data
    let sortedData: Point[] = [];
    if (order === "alphabetical") {
      sortedData = [...points].sort((a, b) =>
a.title.localeCompare(b.title)); // Sort alphabetically by comparing
strings with .localeCompare
    } else if (order === "frequency") {
      sortedData = [...points].sort((a, b) => b.download_count -
a.download_count); // Sorting is done by subtracting `a.download_count`
```

```
from `b.download_count` to arrange the elements from highest to lowest
download count
    }

    setSortedPoints(sortedData); // Update the state variable with the
newly sorted array sortedData
  }, [points]); // The sortData function will run whenever the points
array changes

  // Fetch data from the API when the component loads
  useEffect(() => {
    const fetchData = async () => {
      setLoading(true); // Set loading to true before fetching
      try {
        const response = await fetch("/api/visualisations"); // Fetch data
from the API
        const data: ApiResponse = await response.json(); // Parse the json
response

        // Check if data.data is a valid array, otherwise use an empty
array.
        const formattedData = Array.isArray(data.data) ? data.data : [];
        setPoints(formattedData);  // Set points with the fetched data

      } finally {
        setLoading(false); // After the data fetch, set loading to false
to indicate the process is complete.
      }
    };
    fetchData();
```

```
  }, []); // Run only once when the component is first loaded


  // Sort data whenever `points` or `sortOrder` changes
  useEffect(() => {
    if (points.length > 0) {
      sortData(sortOrder); // Sort data based on the current sort order
    }
  }, [points, sortOrder, sortData]); // Re-run the effect when points, sortOrder, or sortData change


  return (
    <div className="flex flex-col items-center justify-center min-h-screen bg-custom-gray mb-[-2rem]">
      <Navbar /> {/* Navbar */}
      <main className="flex w-full px-8 py-20 justify-center items-center text-white">
        {loading && <p>Loading data...</p>} {/* Show loading message if data is loading */}


        {/* Render the content only when data has finished loading, no error has occurred, and there are data points to display. */}
        {!loading && !error && sortedPoints.length > 0 && (
          <div className="max-w-2xl w-full mx-auto mt-12 lg:mt-28 text-center">
            <h1 className="font-sans font-bold tracking-tight text-white underline text-2xl sm:text-5xl md:text-6xl lg:text-4xl 2xl:text-8xl 2xl:mt-52">
              Project Gutenberg Most Downloaded
            </h1>
            <div className="my-8">
```

```
            {/* Filter Menu for sorting options */}
            <Menu animate={{ mount: { y: 0 }, unmount: { y: 25 } }}>
              <MenuHandler>
                <Button className="bg-gray-50 text-custom-gray my-4
items-start text-sm sm:text-base md:text-lg px-3 sm:px-5 md:px-4 py-2
sm:py-2" {...({} as React.ComponentProps<typeof Button>)}>
                  Filters
                </Button>
              </MenuHandler>
              <MenuList className="w-auto sm:w-48 md:w-64" {...({} as
React.ComponentProps<typeof MenuList>)}>
                <MenuItem
                  className={`flex items-center justify-center
font-semibold text-custom-gray underline hover:text-gray-500
hover:focus:text-gray-500 py-2 text-sm sm:text-base ${sortOrder ===
"alphabetical" ? "bg-gray-200" : ""}`}
                  onClick={() => handleSortChange("alphabetical")}
{...({} as React.ComponentProps<typeof MenuItem>)}
                >
                  Alphabetical
                </MenuItem>
                <MenuItem
                  className={`flex items-center justify-center
font-semibold text-custom-gray underline hover:text-gray-500
hover:focus:text-gray-500 py-2 text-sm sm:text-base ${sortOrder ===
"frequency" ? "bg-gray-200" : ""}`}
                  onClick={() => handleSortChange("frequency")} {...({}
as React.ComponentProps<typeof MenuItem>)}
                >
                  Popular
```

```
                </MenuItem>
              </MenuList>
            </Menu>
          </div>
          <BarChart data={sortedPoints} /> {/* Render BarChart with
sorted data */}
        </div>
      )}

      {/* A No data available message when there is no data to display.
*/}
      {!loading && !error && sortedPoints.length === 0 && (
        <p>No data available to display</p>
      )}
    </main>
    {/* Footer */}
    <Footer />
  </div>
 );
};

export default VisualisationsPage; // Export the VisualisationsPage
component

// Render the component client-side
"use client";
```

```
// Import the components from the components directory and import
necessary modules
import dynamic from "next/dynamic";
import Navbar from "../../components/Navbar";
import { Footer } from "../../components/Footer";

// Disable server-side rendering for the Map component
const Map = dynamic(() => import("../../components/Map"), { ssr: false });

// Create and export the LibrariesPage function
export default function LibrariesPage() {
  return (
    // Flexbox layout container
    <div className="flex flex-col min-h-screen bg-custom-gray">
      {/* Navbar */}
      <Navbar />
      {/* Main content */}
      <main className="flex-grow">
        <div className="w-full max-w-2xl mx-auto text-center mt-16
lg:mt-28">
          {/* Heading */}
          <h1 className="font-sans font-bold tracking-tight underline
text-white text-2xl sm:text-5xl md:text-6xl lg:text-4xl 2xl:text-8xl
2xl:mt-52 py-6">
            Discover Libraries Around Ireland
          </h1>
          {/* Subheading */}
```

```
          <p className="mt-2 mx-16 lg:mx-20 text-xs sm:text-lg md:text-xl
lg:text-base 2xl:text-3xl leading-7 sm:leading-8 text-white
font-extralight text-center 2xl:mt-20">
            Connect with a community of readers and resources in libraries
across Ireland.
          </p>
        </div>
        {/* Map Container */}
        <div className="flex justify-center items-center w-full py-16">
          {/* Map container */}
          <div className="rounded-3xl bg-custom-gray px-4 py-4 mt-[-2rem]
w-full sm:w-2/3 md:w-3/4 lg:w-4/5 min-h-64 2xl:min-h-96 shadow-2xl">
            <Map /> {/* Map Component */}
          </div>
        </div>
      </main>
      {/* Footer */}
      <Footer />
    </div>
  );
}


// Render the component client-side
"use client";


// Import necessary components from the components directory
import Navbar from "../../components/Navbar";
import { DocumentationViewer } from "../../components/DocumentDisplayer";
```

```
import { Footer } from "../../components/Footer";


// Create and export the DocumentationPage function
export default function DocumentationPage() {
  return (
    // Flexbox layout container
    <div className="flex flex-col max-h-64 lg:min-h-screen
bg-custom-gray">
      {/* Navbar */}
      <Navbar />
      <main className="flex-grow relative">
        <div className="w-full max-w-2xl mx-auto text-center mt-16
lg:mt-28">
          {/* Main heading */}
          <h1 className="font-sans font-bold tracking-tight underline
text-white text-2xl sm:text-5xl md:text-6xl lg:text-4xl 2xl:text-8xl
2xl:mt-52 py-6">
            Project Documentation
          </h1>
          {/* Subheading */}
          <p className="pb-2 lg:mt-2 mx-16 lg:mx-20 text-xs sm:text-lg
md:text-xl lg:text-base 2xl:text-3xl leading-7 sm:leading-8 text-white
font-extralight text-center 2xl:mt-20">
            Check out the documentation below.
          </p>
        </div>
        <div className="relative min-w-full px-auto py-2 z-10">
          {/* Documentation viewer */}
          <DocumentationViewer />
```

```
        </div>
      </main>
      {/* Footer */}
      <Footer />
    </div>
  );
}

// Render the component client-side
"use client";

// Import the components from the components directory
import Navbar from "../../components/Navbar";
import AboutPage from "../../components/AboutPage";
import { DefaultAccordion } from "../../components/DefaultAccordion";
import { Footer } from "../../components/Footer";

// Create and export the AboutHome function
export default function AboutHome() {
  return (
    // Flexbox layout container
    <div className="flex flex-col min-h-screen bg-custom-black">
      {/* Navbar */}
      <Navbar />
      {/* Main content */}
      <main className="flex-grow">
        {/* About Page */}
```

```
        <AboutPage />
        {/* Accordion Section */}
        <div className="flex justify-center items-center w-full py-24">
          <div className="box z-12 rounded-lg bg-custom-black px-4 py-4
mt-[-8rem] w-4/5 sm:w-2/3 md:w-3/4 lg:w-6/7 min-h-64 2xl:min-h-96
shadow-2xl flex flex-wrap">
            {/* Accordion */}
            <DefaultAccordion />
          </div>
        </div>
      </main>
      {/* Footer */}
      <Footer />
    </div>
  );
}

// Import necessary styles and components
import ".//../globals.css";
import Navbar from "../../components/Navbar"; // Import Navbar component
import LoginForm from "../../components/LoginForm"; // Import LoginForm
component

// Create and export the Login function
export default function Login() {
  return (
    // Outer container
```

```
    <div className="bg-custom-gray px-4 pt-32 lg:pt-24 lg:px-8
min-h-screen flex-col">
      {/* Navbar */}
      <Navbar />
      {/* Main content  */}
      <div className="mx-auto max-w-2xl py-8 sm:py-24 lg:py-12
text-center">
        {/* Main heading */}
        <h1 className="text-5xl sm:text-4xl md:text-5xl font-bold
tracking-tight text-white underline">
          Login
        </h1>
        {/* Subheading */}
        <p className="mt-8 sm:mt-6 text-xs sm:text-sm font-light overline
leading-6 sm:leading-8 pb-8 text-gray-200">
          Enter your information below.
        </p>
        {/* Login form */}
        <LoginForm />
      </div>
    </div>
  );
}


// Import necessary styles and components
import ".//../globals.css";
import Navbar from "../../components/Navbar"; // Import the Navbar
component
```

```
import RegisterForm from "../../components/RegisterForm"; // Import the
RegisterForm component


// Create and export the Register function
export default function Register() {
  return (
    // Outer container
    <div className="bg-custom-gray px-4 pt-20 lg:pt-14 lg:px-8
min-h-screen">
      {/* Navbar */}
      <Navbar />
      {/* Main content */}
      <div className="mx-auto max-w-2xl py-8 sm:py-24 lg:py-12
text-center">
        {/* Main heading */}
        <h1 className="text-5xl sm:text-4xl md:text-5xl font-bold
tracking-tight text-white underline">
          Register
        </h1>
        {/* Subheading */}
        <p className="mt-8 sm:mt-6 text-xs sm:text-sm font-light overline
leading-6 sm:leading-8 pb-8 text-gray-200">
          Enter your information below.
        </p>
        {/* Register form */}
        <RegisterForm />
      </div>
    </div>
  );
```

```
}


@tailwind base;
@tailwind components;
@tailwind utilities;


/* globals.css */
@layer base {
  input:-webkit-autofill {
    background-color: var(--custom-gray) !important;
    color: var(--custom-text) !important;
    -webkit-box-shadow: 0 0 0px 1000px var(--custom-gray) inset
!important;
  }


  input:-webkit-autofill:focus,
  input:-webkit-autofill:hover {
    background-color: var(--custom-gray) !important;
    color: var(--custom-text) !important;
  }


  input:-moz-autofill {
    background-color: var(--custom-gray) !important;
    color: var(--custom-text) !important;
  }
}


  .leaflet-container {
```

```
    z-index: 0 !important; /* Ensure it stays behind other elements */
  }




// Importing the Link component from next/link for client-side navigation between pages
import Link from "next/link";


export default function PrivacyPolicy() {
    return (
      // Flexbox layout container
      <div className="min-h-screen flex flex-col items-center justify-center text-center p-2">
        {/* Main heading */}
        <h1 className="text-3xl font-bold underline mt-[-6rem]">Privacy Policy</h1>
        {/* Subheading */}
        <p className="mt-4 text-base lg:text-lg">Please check back soon for the full privacy policy.</p>
        {/* Return Link */}
        <Link href="/" className="px-5 py-2 text-sm 2xl:text-2xl font-semibold leading-6 text-black hover:focus:text-gray-600 hover:text-gray-600 hover:underline">
          Return <span aria-hidden="true">→</span>
        </Link>
      </div>
    );
}
```

```
// Render the component client-side
"use client";

// Import the components from the components directory
import { AdminDashboardTabs } from "@/components/AdminDashboard";

const AdminDashboard = () => {
  return (
    // Outer container
    <div className="p-10 bg-custom-gray">
      <AdminDashboardTabs />
    </div>
  );
};

export default AdminDashboard; // Export AdminDashboard

// Render the component client-side
"use client";

// Import the components from the components directory
import { UserDashboardTabs } from "@/components/UserDashboard";

const UserDashboard = () => {
  return (
    // Outer container
```

```
    <div className="p-10 bg-custom-gray">
      <UserDashboardTabs />
    </div>
  );
};

export default UserDashboard; // Export UserDashboard

// Import NextResponse from 'next/server'
import { NextResponse } from "next/server";

// Import prisma to interact with the database
import prisma from "../../../lib/prisma";

// resetEbookSequence function
async function resetEbookSequence() {
  // Handle DELETE request to remove an ebook and reset the sequence
  const maxId = await prisma.ebook.aggregate({
    _max: {
      id: true, // Get the highest 'id' value
    },
  });

  // Set the next sequence ID for 'Ebook' table, defaulting to 1 if no
records exist
  const nextId = (maxId._max.id || 0) + 1;
  // Set 'Ebook' sequence to the next ID
  await prisma.$executeRawUnsafe(`
```

```
    SELECT setval('public."Ebook_id_seq"', ${nextId}, false);
  `);
}

// DELETE function
export async function DELETE(request: Request) {
  try {
    // Get query parameters from the request URL
    const ebookId = new URL(request.url).searchParams.get("id");

    // Ensure ebookId exists
    if (!ebookId) {
      return NextResponse.json({ error: "Ebook ID is required" }, {
status: 400 }); // Status code 400 for bad request
    }

    // Delete the ebook from the database
    const deletedEbook = await prisma.ebook.delete({
      where: { id: Number(ebookId) },
    });

    // Reset the sequence
    await resetEbookSequence();

    // Return the deleted ebook as a response
    return NextResponse.json(deletedEbook);
  } catch {
    // Error message
```

```
    return NextResponse.json({ error: "An error occurred while deleting
the ebook" }, { status: 500 }); // Status code 500 for internal server
error
  }
}

import { NextResponse } from 'next/server'; // Import NextResponse from
'next/server'
import { PrismaClient } from '@prisma/client'; // Importing PrismaClient
to interact with the database
import { createClient } from '@supabase/supabase-js'; // Import Supabase
client

// Creating an instance of PrismaClient to communicate with the database
const prisma = new PrismaClient();

// Initialize Supabase client
const supabase = createClient(
  'https://tshuqzoktjjlooqfenvm.supabase.co', // Supabase Project URL

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InRz
aHVxem9rdGpqbG9vcWZlbnZtIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTczNDIxMD
Y2NiwiZXhwIjoyMDQ5Nzg2NjY2fQ.JGFT5rSkyoym7MiG_Pmo5IlWLxi16ZvmtP15_57UT3U'
// 'service_role' Supabase API key
);

// POST function
export async function POST(request: Request) {
  try {
    // Parsing the incoming form data
```

```
    const formData = await request.formData();


    // Extract input data from the form
    const title = formData.get("title") as string; // Get the title of the
ebook
    const author = formData.get("author") as string; // Get the author of
the ebook
    const description = formData.get("description") as string; // Get the
description of the ebook
    const publishedAt = new Date(formData.get("publishedAt") as string);
// Get the publish date of the ebook


    // Extracting the ebook file and cover page file from the form data
    const file = formData.get("file") as File;
    const coverPage = formData.get("coverPage") as File;


    // Function to upload files to Supabase Storage
    const uploadToSupabase = async (file: File, folder: string):
Promise<string> => {
      // Upload the file to Supabase Storage buckets
      const { data, error } = await supabase
        .storage
        .from(folder) // Supabase bucket
        .upload(`${folder}/${file.name}`, file.stream(), {
          upsert: true, // If the file already exists then overwrite it
          duplex: 'half', // Allow file upload with duplex stream
        });


      if (error) {
```

```
        throw new Error(`Supabase upload error: ${error.message}`);
      }

      // Ensure the file path exists and return the public URL
      if (data?.path) {
        return
`https://tshuqzoktjjlooqfenvm.supabase.co/storage/v1/object/public/${folde
r}/${data.path}`;
      } else {
        throw new Error("Failed to get file URL after uploading ebook form
data.");
      }
    };

    // Upload the ebook file and cover page to Supabase Storage
    const fileUrl = await uploadToSupabase(file, 'ebooks');
    const coverPageUrl = await uploadToSupabase(coverPage, 'coverPages');

    // Save the ebook information to the database
    const ebook = await prisma.ebook.create({
      data: {
        title,
        author,
        description,
        publishedAt,
        fileUrl,
        coverPageUrl,
      },
```

```
    });

    // Success message
    return NextResponse.json({ message: "Ebook added successfully!", ebook
});
  } catch {
    // Error message
    return NextResponse.json({ error: "Something went wrong" }, { status:
500 }); // Status code 500 for internal server error
  }
}

// Import NextResponse from 'next/server'
import { NextResponse } from 'next/server';

// Import prisma to interact with the database
import { prisma } from '../../../lib/db';

// GET function
export async function GET() {
  // Fetch all ebooks from the database using Prisma's findMany method
  const ebooks = await prisma.ebook.findMany();

  // Return ebooks data as a JSON response
  return NextResponse.json(ebooks);
}

// Import Prisma client to interact with the database
```

```
import { prisma } from '../../../lib/db';


// GET function
export async function GET(req: Request) {
  // Get the request URL
  const url = new URL(req.url);


  // Get 'userId' from the query string
  const userId = url.searchParams.get('userId');


  // If 'userId' is missing, return a 400 error status code
  if (!userId) {
    return new Response(JSON.stringify({ error: 'User ID is required' }),
{
      status: 400, // Status code 400 for bad request
    });
  }


  // Query the database to find the user by 'userId' and select the
'profilePicture'
  const user = await prisma.user.findUnique({
    where: { id: parseInt(userId) }, // Find user by 'id'
    select: { profilePicture: true }, // Only select the 'profilePicture'
field
  });


  // If user is found return profile data, if not return null value
  return new Response(
```

```
    JSON.stringify({ user: user || null }),
    { status: 200 } // Successfully updated
  );
}

import { NextResponse } from 'next/server'; // Import NextResponse from 'next/server'
import { prisma } from '@/lib/db'; // Import Prisma client for interacting with the database
import bcrypt from 'bcryptjs'; // Import for hashing and comparing passwords
import jwt from 'jsonwebtoken'; // Import for creating and verifying JSON Web Tokens

// POST function
export async function POST(request: Request) {
  try {
    // Parse incoming JSON request data to get the email and password
    const { email, password } = await request.json();

    // Validate and ensure that both email and password are provided
    if (!email || !password) {
      return NextResponse.json({ error: 'Email and password are required' }, { status: 400 }); // Status code 400 for bad request
    }

    // Find the user by the provided email in the database
    const user = await prisma.user.findUnique({
      where: { email },
```

```
    });

    // If no user is found with the given email, return an error
    if (!user) {
      return NextResponse.json({ error: 'Invalid email or password' }, {
status: 400 }); // Status code 400 for bad request
    }

    // Compare the provided password with the hashed password
    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {
      return NextResponse.json({ error: 'Invalid email or password' }, {
status: 400 }); // Status code 400 for bad request
    }

    // Fetch JWT secret from environment variable .env file
    const jwtSecret = process.env.JWT_SECRET;

    if (!jwtSecret) {
      // If JWT secret is missing, throw an error
      throw new Error("JWT_SECRET is not defined in the environment
variables. Please check the .env file");
    }

    // Generate a JWT token with user details if authentication is
successful
    const token = jwt.sign(
```

```
      {
        userId: user.id,
        isAdmin: user.isAdmin,
        username: user.username,
        email: user.email,
        profilePicture: user.profilePicture,
      },
      jwtSecret, // Secret key to encode the token
      { expiresIn: '1h' } // Token expiration time
    );

    // Return a success message with the token and user's admin status
    return NextResponse.json({
      message: 'Login successful!',
      token,
      isAdmin: user.isAdmin,
      profilePicture: user.profilePicture,
    });
  } catch {
    // Return error message
    return NextResponse.json({ error: 'Something went wrong' }, { status:
500 }); // Status code 500 for internal server error
  }
}

// Import NextRequest and NextResponse from 'next/server' and import
necessary modules
import { NextRequest, NextResponse } from "next/server";
```

```
import { PrismaClient } from "@prisma/client";


// Import prisma to interact with the database
const prisma = new PrismaClient();


// GET function
export async function GET(req: NextRequest) {
  // Get the query parameters from the URL of the request
  const { searchParams } = new URL(req.url);
  const id = searchParams.get("id");


  // Check if id is a valid number
  const parsedId = parseInt(id ?? '', 10);  // Convert id to an integer,
defaulting to NaN if id is null
  if (isNaN(parsedId)) {
    return NextResponse.json(
      { error: "Invalid 'id' parameter" },  // Error message for invalid
'id'
      { status: 400 }  // Status code 400 for bad request
    );
  }


  // Fetch the ebook data from the database using the provided id
  const ebook = await prisma.ebook.findUnique({
    where: { id: parsedId },  // Query the database for the ebook with the
given id
  });
```

```
  // If the ebook is not found, return error message
  if (!ebook) {
    return NextResponse.json(
      { error: "Ebook not found" },
      { status: 404 }  // Status code 404 for not found
    );
  }

  // Return the ebook details if found
  return NextResponse.json({
    id: ebook.id,  // Ebook ID
    title: ebook.title,  // Ebook title
    description: ebook.description,  // Ebook description
    fileUrl: ebook.fileUrl,  // URL to the ebook file
    coverPageUrl: ebook.coverPageUrl,  // URL to the cover page image
  });
}

import { NextResponse } from 'next/server'; // Import NextResponse from
'next/server'
import { prisma } from '@/lib/db'; // Import Prisma client for interacting
with the database
import bcrypt from 'bcryptjs'; // Import for hashing and comparing
passwords

// POST function
export async function POST(request: Request) {
  try {
```

```
    // Parse the request data
    const body = await request.json();

    // Extract username, email, and password
    const { username, email, password } = body;

    // Validate that username, email, and password are provided
    if (!username || !email || !password) {
      return NextResponse.json({ error: "Username, email, and password are
required" }, { status: 400 }); // Status code 400 for bad request
    }

    // Check if the email or username already exists in the database
    const existingUser = await prisma.user.findFirst({
      where: {
        OR: [
          { email }, // Check if email exists
          { username }, // Check if username exists
        ],
      },
    });

    // If either the email or username is taken, return an error
    if (existingUser) {
      return NextResponse.json({ error: "Email or username already exists"
}, { status: 400 }); // Status code 400 for bad request
    }
```

```
    // Hash the password before saving it to the database
    const hashedPassword = await bcrypt.hash(password, 10);

    // Create the new user in the User table in the database
    await prisma.user.create({
      data: {
        username,
        email,
        password: hashedPassword,
      },
    });

    // Return a success message
    return NextResponse.json({ message: "Registration successful!" });

  } catch {
    // Return error message
    return NextResponse.json({ error: "Something went wrong" }, { status:
500 }); // Status code 500 for internal server error
  }
}

import { prisma } from '../../../lib/db'; // Import Prisma client to
interact with the database
import jwt from 'jsonwebtoken'; // Import for creating and verifying JSON
Web Tokens

// Define an interface for the DecodedToken object
```

```
interface DecodedToken {
  userId: number; // User ID extracted from the decoded token
}

// POST function
export async function POST(req: Request) {
  // Extract profile picture URL and token from the request
  const { profilePicture, token } = await req.json();

  // Decode the JWT token
  const decodedToken = jwt.decode(token) as DecodedToken;

  // Check if the token is valid and contains a userId
  if (!decodedToken || !decodedToken.userId) {
    return new Response(JSON.stringify({ error: 'Unauthorized' }), {
      status: 401, // Unauthorized if token is invalid
    });
  }

  try {
    // Update the user's profile picture in the database
    const updatedUser = await prisma.user.update({
      where: { id: decodedToken.userId }, // Find user by ID
      data: {
        profilePicture, // Set new profile picture URL
      },
    });
```

```
    // Success message
    return new Response(
      JSON.stringify({ message: 'Profile picture updated', user:
updatedUser }),
      {
        status: 200, // Successfully updated
      }
    );
  } catch {
    // Return error message
    return new Response(
      JSON.stringify({ error: 'Failed to update profile picture' }),
      { status: 500 } // Status code 500 for internal server error
    );
  }
}

import { NextResponse } from "next/server"; // Import NextResponse from
'next/server'
import prisma from "../../../lib/prisma"; // Import Prisma client to
interact with the database

// GET function
export async function GET(request: Request) {
  // Parse the request URL to access query parameters
  const url = new URL(request.url);
```

```
  // Get page and limit from query parameters
  const page = parseInt(url.searchParams.get("page") || "1", 10); //
Default to page 1
  const limit = parseInt(url.searchParams.get("limit") || "50", 10); //
Default to 50 items per page
  const offset = (page - 1) * limit; // Calculate how many records to skip
based on the current page and the limit per page


  try {
    // Fetch books, sorted by download count
    const books = await prisma.books_book.findMany({
      select: {
        title: true, // Select only the 'title' field
        download_count: true, // Select the 'download_count' field
      },
      orderBy: {
        download_count: 'desc', // Order books by download count in
descending order
      },
      skip: offset, // Skip the records from previous pages based on the
offset
      take: limit, // Get 'limit' number of books
    });


    // Get the total number of books
    const totalBooks = await prisma.books_book.count();


    // Return the books data, total count, and page information
    return NextResponse.json({
```

```
      data: books, // Array of fetched books
      total: totalBooks, // Total number of books
      page, // Current page number
      totalPages: Math.ceil(totalBooks / limit), // Calculate the total
number of pages
    });
  } catch {
    // Return error message
    return NextResponse.json({ error: 'An unexpected error occurred' });
  }
}

// Render the component client-side
"use client";

// AboutPage component
export default function AboutPage() {
  return (
    <div className="bg-custom-gray">
      {/* Hero Section */}
      <div className="relative isolate px-6 pt-0 lg:px-48 bg-about-pattern
bg-no-repeat bg-center bg-cover min-h-[80vh] lg:min-h-screen w-full">
        {/* Container for the hero content */}
        <div className="mx-auto max-w-full">
          <div className="flex items-center justify-center
lg:justify-between pt-2">
            {/* Intro Header */}
```

```
            <div className="mt-24 text-center lg:mt-32 lg:text-left
pb-24">
              {/* Hero Heading */}
              <h1 className="font-sans font-bold tracking-tight text-white
underline text-4xl sm:text-5xl md:text-6xl lg:text-7xl 2xl:text-8xl
2xl:mt-52">
                ABOUT US
              </h1>

              {/* Hero Subheading */}
              <p className="mt-4 lg:mt-8 text-xs sm:text-base md:text-xl
lg:text-sm 2xl:text-3xl lg:w-96 max-w-40 sm:max-w-md md:max-w-lg leading-7
sm:leading-8 text-white font-extralight lg:font-light 2xl:mt-20">
                Explore a vast collection of ebooks and immerse yourself
in the world of endless reading possibilities.
              </p>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}

// Render the component client-side
"use client";

// Import React hooks and import necessary modules
import {
```

```
  Input,
  Button,
  Typography,
} from "@material-tailwind/react";
import { useState, useRef } from "react";

const AddEbookForm = () => {
  // State hooks for managing form values
  const [title, setTitle] = useState(""); // To store the book title
  const [author, setAuthor] = useState(""); // To store the author name
  const [description, setDescription] = useState(""); // To store the
description
  const [publishedAt, setPublishedAt] = useState(""); // To store the
publish date
  const [file, setFile] = useState<File | null>(null); // To store the
ebook file
  const [coverPage, setCoverPage] = useState<File | null>(null); // To
store the cover page file
  const [message, setMessage] = useState(""); // To store messages such as
success or error messages

  // References for file input fields
  const fileInputRef = useRef<HTMLInputElement | null>(null);
  const coverPageInputRef = useRef<HTMLInputElement | null>(null);

  // Submit handler for the form
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault(); // Prevent the form from submitting the default
way
```

```
    setMessage(""); // Clear previous messages

    const formData = new FormData(); // Create a new FormData object to handle form data
    formData.append("title", title); // Add title to the form data
    formData.append("author", author); // Add author to the form data
    formData.append("description", description); // Add description to the form data
    formData.append("publishedAt", publishedAt); // Add published date to the form data

    // Append files if they exist
    if (file) formData.append("file", file); // If a file is selected, append it
    if (coverPage) formData.append("coverPage", coverPage); // If a cover page is selected, append it

    try {
      const response = await fetch("/api/ebooks", { // Make a POST request to the server
        method: "POST", // Use POST method to send the form data
        body: formData, // Send the form data as the body of the request
      });

      if (response.ok) { // Check if the response was successful
        setMessage("Ebook added successfully!"); // Set a success message

        // Reset form fields
        setTitle("");
```

```
        setAuthor("");
        setDescription("");
        setPublishedAt("");
        setFile(null);
        setCoverPage(null);

        // Reset the file inputs using references
        if (fileInputRef.current) fileInputRef.current.value = "";
        if (coverPageInputRef.current) coverPageInputRef.current.value =
"";

        // Reload the page after submission
        window.location.reload();
      }
    } catch {
      setMessage("Error: Unable to add ebook."); // If the fetch request
fails, show an error message
    }
  }

  return (
    <form
      onSubmit={handleSubmit} // Handle form submission
      className="relative flex flex-col items-center justify-center mb-2
w-full px-4 sm:px-6 lg:px-74"
    >
      <div className="mb-1 flex flex-col items-center justify-center gap-6
w-full max-w-lg min-h-screen">
```

```
        {/* Book Title Input */}
        <Typography variant="h6" color="blue-gray" className="pt-3 text-lg
underline" {...({} as React.ComponentProps<typeof Typography>)}>
          Book Title
        </Typography>
        <Input
          type="text"
          id="title"
          size="lg"
          placeholder="Sample Title"
          value={title}
          onChange={(e) => setTitle(e.target.value)} // Update title state
          required
          className="bg-white !border-t-blue-gray-200
focus:!border-t-gray-500"
          labelProps={{
            className: "before:content-none after:content-none",
          }} {...({} as React.ComponentProps<typeof Input>)}
        />


        {/* Author Input */}
        <Typography variant="h6" color="blue-gray" className="pt-3 text-lg
underline" {...({} as React.ComponentProps<typeof Typography>)}>
          Author
        </Typography>
        <Input
          type="text"
          id="author"
```

```
          size="lg"
          placeholder="Sample Author"
          value={author}
          onChange={(e) => setAuthor(e.target.value)} // Update author
state
          required
          className="bg-white !border-t-blue-gray-200
focus:!border-t-gray-500"
          labelProps={{
            className: "before:content-none after:content-none",
          }} {...({} as React.ComponentProps<typeof Input>)}
        />


        {/* Description Input */}
        <Typography variant="h6" color="black" className="pt-3 text-lg
underline" {...({} as React.ComponentProps<typeof Typography>)}>
          Description
        </Typography>
        <textarea
          id="description"
          placeholder="Sample Description"
          value={description}
          onChange={(e) => setDescription(e.target.value)} // Update
description state
          required
          className="!border-t-blue-gray-500 focus:!border-t-gray-500
w-full pb-16"
        ></textarea>
```

```
        {/* Publish Date Input */}
        <label
          htmlFor="publishedAt"
          className="block text-black text-base underline"
        >
          Published Date
        </label>
        <input
          type="date"
          id="publishedAt"
          value={publishedAt}
          onChange={(e) => setPublishedAt(e.target.value)} // Update
published date state
          required
          className="mt-1"
        />


        {/* Ebook File Input */}
        <label
          htmlFor="file"
          className="block text-black text-base underline"
        >
          Ebook File
        </label>
        <input
          type="file"
          id="file"
          accept=".html"
```

```
          onChange={(e) => setFile(e.target.files?.[0] || null)} // Update
file state
          required
          className="mt-1 block lg:px-24 w-52 lg:w-96"
          ref={fileInputRef} // Reference for file input
        />

        {/* Cover Page Input */}
        <label
          htmlFor="coverPage"
          className="block text-black text-base underline"
        >
          Cover Page
        </label>
        <input
          type="file"
          id="coverPage"
          accept="image/*"
          onChange={(e) => setCoverPage(e.target.files?.[0] || null)} //
Update cover page state
          required
          className="mt-1 block lg:px-24 w-52 lg:w-96"
          ref={coverPageInputRef} // Reference for cover page input
        />

        {/* Submit Button */}
        <Button type="submit" className="m-6 w-3/5 lg:w-4/5 sm:w-auto"
{...({} as React.ComponentProps<typeof Button>)}>
```

```
          Submit
        </Button>
      </div>

      {/* Success or error message */}
      <div className="lg:mt-2">
        {message && <p className="absolute lg:bottom-0 left-1/2 transform
-translate-x-1/2 text-sm text-red-500">{message}</p>}
      </div>
    </form>
  );
};

export default AddEbookForm; // Export the form component

// Import React hooks, components and necessary modules
import Image from "next/image";
import React, { useState, useEffect } from "react";
import {
  Drawer,
  Tabs,
  TabsHeader,
  Tab,
  Button,
  Select,
  Option
} from "@material-tailwind/react";
```

```
import { Square3Stack3DIcon, BookOpenIcon, DocumentTextIcon, TrashIcon,
UserCircleIcon, Cog6ToothIcon } from "@heroicons/react/24/solid";

import EbookReader from "./EbookReader";

import AddEbookForm from "./AddEbookForm";

import DeleteEbookForm from "./DeleteEbookForm";

import { AdminProfileCard } from "./AdminProfileCard";

import LogoutButton from "./Logout";

import { AdminOverviewTab } from "./AdminOverviewTab";


// Define an interface for the Ebook object

interface Ebook {

  id: number;

  title: string;

  author: string;

  description: string;

  publishedAt: string;

  fileUrl: string;

  coverPageUrl: string;

}


// Main AdminDashboardTabs component

export function AdminDashboardTabs() {

  const [open, setOpen] = useState(false); // State for drawer visibility

  const [selectedTab, setSelectedTab] = useState("dashboard"); // State to
track selected tab

  const [ebookId, setEbookId] = useState<number | null>(null); // Track
the selected ebook ID
```

```
  const [ebooks, setEbooks] = useState<Ebook[]>([]); // State to store
list of ebooks
  const [loading, setLoading] = useState<boolean>(false); // Loading state
for fetching ebooks
  const [error, setError] = useState<string | null>(null); // State to
track errors


  // Function to open the drawer
  const openDrawer = () => setOpen(true);


  // Function to close the drawer
  const closeDrawer = () => setOpen(false);


  // Data for each tab in the sidebar: labels, values, and icons
  const data = [
    { label: "Dashboard", value: "dashboard", icon: Square3Stack3DIcon },
    { label: "ebook Reader", value: "ebookreader", icon: BookOpenIcon },
    { label: "ebook Form", value: "ebookform", icon: DocumentTextIcon },
    { label: "Delete ebook Form", value: "deleteEbookform", icon:
TrashIcon },
    { label: "Profile", value: "profile", icon: UserCircleIcon },
    { label: "Settings", value: "settings", icon: Cog6ToothIcon },
  ];


  // Fetch ebooks on component load
  useEffect(() => {
    const fetchEbooks = async () => {
      setLoading(true); // Set loading to true while fetching
```

```
      const response = await fetch("/api/fetch");
      const data = await response.json();
      setEbooks(data); // Store fetched ebooks in state
      setLoading(false); // Set loading to false once fetching is complete
    };

    fetchEbooks();
  }, []); // Empty dependency array ensures it runs only once when the
component loads

  // Function to handle ebook deletion
  const handleDeleteEbook = async (ebookId: number) => {
    if (!ebookId) return;

    // Send a Delete request to the server to remove the ebook with the
specified ID
    const response = await fetch(`/api/delete?id=${ebookId}`, { method:
"DELETE" });

    // If the response was unsuccessful set an error message
    if (!response.ok) {
      const errorData = await response.json();
      setError(errorData.error || "Failed to delete ebook");
      return;
    }

    // If deletion is successful, update the state
```

```
    setEbooks((prevEbooks) => prevEbooks.filter((ebook) => ebook.id !==
ebookId));
    setEbookId(null); // Reset the selected ebook ID
  };

  // Function to select an ebook by ID
  const handleSelectEbook = (id: number) => {
    setEbookId(id);
  };

  // Function to render the relevant content based on selected tab
  const renderContent = () => {
    switch (selectedTab) {
      case "dashboard":
        return <AdminOverviewTab />;
      case "ebookreader":
        return (
          <div className="flex w-full h-full flex-col gap-6 py-4 px-2">
            <h2 className="text-lg lg:text-2xl font-bold text-black
underline mb-4 ml-4">Choose a ebook:</h2>

            {/* Loading or error states */}
            {loading && <div className="text-center text-gray-600">Loading
ebooks...</div>}
            {error && <div className="text-center
text-red-500">{error}</div>}

            {/* Ebook selection dropdown */}
```

```
            <Select
              size="lg"
              label="Select Ebook"
              value={ebookId?.toString() || ""}
              onChange={(value) => {
                // Convert the selected value to a number and update the
ebookId state
                const selectedId = Number(value);
                setEbookId(selectedId);
              }}
              className="mb-6" {...({} as React.ComponentProps<typeof
Select>)}
            >
              {/* Iterate over the ebooks array and create an option for
each ebook */}
              {ebooks.map((ebook) => (
                <Option key={ebook.id} value={ebook.id.toString()}>
                  {ebook.title}
                </Option>
              ))}
            </Select>

            {/* If an ebook is selected, display the book details and
cover page */}
            {ebookId && (
              <div className="text-center">
                <h3 className="text-x1 lg:text-3x1 font-semibold overline
mt-4">{ebooks.find((ebook) => ebook.id === ebookId)?.title}</h3>
```

```
                <p className="text-sm lg:text-base text-black underline
py-2 mt-4">Author: {ebooks.find((ebook) => ebook.id ===
ebookId)?.author}</p>

                <p className="text-sm lg:text-base text-justify
text-gray-600 font-medium mt-4 px-10 lg:px-60">Description:
{ebooks.find((ebook) => ebook.id === ebookId)?.description}</p>


                {/* Show cover image only when an ebook is selected */}

                {ebooks.find((ebook) => ebook.id ===
ebookId)?.coverPageUrl && (

                  <Image

                    src={ebooks.find((ebook) => ebook.id ===
ebookId)?.coverPageUrl || "/default-image.jpg"}

                    alt={ebooks.find((ebook) => ebook.id ===
ebookId)?.title || "Default Title"}

                    className="mt-12 h-80 w-72 lg:h-96 lg:w-80 mx-auto
hover:-translate-y-1 hover:scale-110 duration-300"

                    width={288}

                    height={320}

                  />

                )}


                {/* Display the EbookReader component */}

                <div className="overflow-hidden mt-4">

                  <EbookReader ebookId={ebookId} />

                </div>

              </div>

            )}

          </div>

        );
```

```
    case "ebookform":
      return <AddEbookForm />;
    case "deleteEbookform":
      return (
        <DeleteEbookForm
          ebooks={ebooks}
          ebookId={ebookId}
          onDelete={handleDeleteEbook}
          onSelect={handleSelectEbook}
        />
      );
    case "profile":
      return <AdminProfileCard />;
    case "settings":
      return <LogoutButton />;
    default:
      return null; // Return nothing if no tab is selected
  }
};

// Function to handle tab change
const handleTabChange = (value: string) => {
  setSelectedTab(value);
  closeDrawer();
};

return (
```

```
    <>
      {/* Button to open the tab drawer */}
      <div className="flex items-center justify-center">
        <Button className="text-sm font-semibold" color="red"
onClick={openDrawer} {...({} as React.ComponentProps<typeof Button>)}>
          Open Menu
        </Button>
      </div>

      {/* Drawer component to show tabs */}
      <Drawer {...({} as React.ComponentProps<typeof Drawer>)} open={open}
onClose={closeDrawer} className="p-4 w-64">
        <Tabs value={selectedTab} orientation="vertical">
          <TabsHeader {...({} as React.ComponentProps<typeof
TabsHeader>)}>
            {data.map(({ label, value, icon }) => (
              <Tab {...({} as React.ComponentProps<typeof Tab>)}
key={value} value={value} onClick={() => handleTabChange(value)}>
                <div className="flex items-center gap-6 m-2">
                  {React.createElement(icon, { className: "w-5 h-5" })}
{/* Render the tab icon */}
                  {label} {/* Render the tab label */}
                </div>
              </Tab>
            ))}
          </TabsHeader>
        </Tabs>
      </Drawer>
```

```
      {/* Main content area */}
      <div className="rounded-2xl lg:p-8 bg-gray-50 flex flex-col
min-h-screen">
        {renderContent()} {/* Render content based on selected tab */}
      </div>
    </>
  );
}

// Import the component from the components directory
import { AdminTimeline } from "./AdminTimelineCard";

// AdminOverviewTab component
export function AdminOverviewTab() {
  return (
    // Flexbox layout container
    <div className="flex flex-row items-center justify-center">
      {/* Admin Timeline */}
      <AdminTimeline />
    </div>
  );
}

// Import React hooks, token and necessary modules
import Image from 'next/image';
import { useEffect, useState } from 'react';
import jwt from 'jsonwebtoken';
import {
```

```
  Card,
  CardHeader,
  CardBody,
  CardFooter,
  Typography,
  Button,
} from "@material-tailwind/react";

// Define an interface for the token object
interface DecodedToken {
  userId: string;
  username: string;
  email: string;
}

// AdminProfileCard component
export function AdminProfileCard() {
  const [username, setUsername] = useState(''); // State to store the username
  const [email, setEmail] = useState(''); // State to store the email
  const [selectedImage, setSelectedImage] = useState(''); // State for selected profile image, initially empty value
  const [showPopup, setShowPopup] = useState(false); // State to toggle Popup visibility
  const [errorMessage, setErrorMessage] = useState(''); // State for error message
  const [successMessage, setSuccessMessage] = useState(''); // State for success message
```

```
  // Image pool
  const imagePool = [
    '../images/avatar-man.svg',
    '../images/avatar-woman-5.svg',
    '../images/avatar-man-1.svg',
    '../images/avatar-woman-6.svg',
    '../images/avatar-man-2.svg',
    '../images/avatar-woman-7.svg',
    '../images/avatar-man-3.svg',
    '../images/avatar-woman-8.svg',
    '../images/avatar-man-4.svg',
    '../images/avatar-woman.svg',
    '../images/avatar-woman-1.svg',
    '../images/avatar-woman-2.svg',
    '../images/avatar-woman-3.svg',
    '../images/avatar-woman-4.svg',
  ];


  useEffect(() => {
    const token = localStorage.getItem("token"); // Retrieve the token
from localStorage


    if (token) {
      try {
        const decodedToken = jwt.decode(token) as DecodedToken; // Decode
the token with the declared interface


        if (decodedToken && decodedToken.userId) {
```

```
          setUsername(decodedToken.username || 'Default Username');
          setEmail(decodedToken.email || 'Default Email');
          // Fetch the profile picture from the API
          fetch(`/api/getUserProfile?userId=${decodedToken.userId}`)
            .then((response) => response.json()) // Parse the JSON
response
            .then((data) => {
              // Set the user's profile picture or a default image
              if (data.user) {
                setSelectedImage(data.user.profilePicture ||
'../images/avatar-man-1.svg');
              }
            });
        }
      } catch {
        setErrorMessage('Error decoding the token');
      }
    }
  }, []); // Runs once on component load


  // Handle image selection
  const handleImageSelect = async (image: string) => {
    setSelectedImage(image); // Update selected image
    setShowPopup(false); // Close the Popup


    // Get the token from localStorage
    const token = localStorage.getItem("token");
    if (token) {
```

```
      try {
        // Sending a POST request to update the profile picture in the
database
        const response = await fetch('/api/updateProfile', {
          method: 'POST', // Sending POST request to the updateProfile API
          headers: {
            'Content-Type': 'application/json', // Setting request body
type
          },
          body: JSON.stringify({ profilePicture: image, token }), // Send
selected image and token
        });

        const data = await response.json();
        if (response.ok) {
          setSelectedImage(data.user.profilePicture); // Set the updated
profile picture
          setSuccessMessage(data.message || 'Profile picture updated
successfully!'); // Set success message
          setTimeout(() => setSuccessMessage(''), 3000); // Clear message
after 3 seconds
        } else {
          setErrorMessage(data.error || 'Failed to update profile
picture');
        }
      } catch {
        setErrorMessage('Error updating profile picture');
      }
    }
```

```
  };


  return (
    <div className="flex items-center justify-center min-h-screen p-2">
      <Card className="min-h-screen w-full sm:w-4/5 md:w-2/5" {...({} as
React.ComponentProps<typeof Card>)}>
        <CardHeader floated={false} className="h-60 lg:h-80" {...({} as
React.ComponentProps<typeof CardHeader>)}>
          {/* If selectedImage is not set, fallback to default */}
          <Image
            src={selectedImage || '/images/avatar-man-1.svg'}
            alt="profile-picture"
            width={500}
            height={300}
            className="rounded-full"
          />
        </CardHeader>
        <CardBody className="text-center" {...({} as
React.ComponentProps<typeof CardBody>)}>
          <Typography variant="h4" color="blue-gray" className="underline
my-4 lg:my-2 text-x1 lg:text-3xl" {...({} as React.ComponentProps<typeof
Typography>)}>
            {username}
          </Typography>
          <Typography color="blue-gray" className="mt-12 text-lg
lg:text-2xl" {...({} as React.ComponentProps<typeof Typography>)}>
            {email}
          </Typography>
        </CardBody>
```

```
        <CardFooter className="flex justify-center gap-7 pt-2" {...({} as
React.ComponentProps<typeof CardFooter>)}>
          <Button
            onClick={() => setShowPopup(true)} // Open the Popup
            className="text-xs mt-20" {...({} as
React.ComponentProps<typeof Button>)}
          >
            Change Profile Picture
          </Button>
        </CardFooter>
      </Card>

      {/* Error message */}
      {errorMessage && (
        <div className="text-center text-red-500 mt-4">
          <Typography {...({} as React.ComponentProps<typeof
Typography>)}>{errorMessage}</Typography>
        </div>
      )}

      {/* Success message */}
      {successMessage && (
        <div className="lg:mt-2">
          <p className="absolute bottom-20 lg:bottom-[-5rem] left-1/2
transform -translate-x-1/2 text-sm text-green-500">
            {successMessage}
          </p>
        </div>
```

```
      )}


      {/* Popup for image selection */}
      {showPopup && (
        <div className="fixed inset-0 pb-2 flex items-center
justify-center lg:items-center lg:justify-center bg-black bg-opacity-50">
          <div className="bg-white p-2 sm:p-6 rounded-lg shadow-lg w-9/12
sm:w-3/4 md:w-1/2 lg:w-2/3 max-h-[90vh] overflow-y-auto">
            <Typography variant="h6" className="mb-4 text-center
underline" {...({} as React.ComponentProps<typeof Typography>)}>
              Choose a Profile Picture
            </Typography>
            <div className="flex flex-wrap justify-center gap-4">
              {imagePool.map((image, index) => (
                <Image
                  key={index}
                  src={image}
                  alt={`avatar ${index + 1}`}
                  className="w-16 h-16 lg:w-24 lg:h-24 rounded-full
cursor-pointer border-2 border-transparent hover:border-blue-500"
                  width={100}
                  height={100}
                  onClick={() => handleImageSelect(image)} // Update image
on click
                />
              ))}
            </div>
            <div className="mt-8 flex justify-center">
```

```
              <Button onClick={() => setShowPopup(false)} color="red"
{...({} as React.ComponentProps<typeof Button>)}>
                Close
              </Button>
            </div>
          </div>
        </div>
      )}
    </div>
  );
}


// Import necessary modules and icons
import {
    Timeline,
    TimelineItem,
    TimelineConnector,
    TimelineHeader,
    TimelineIcon,
    TimelineBody,
    Typography,
  } from "@material-tailwind/react";
  import { HomeIcon, BookOpenIcon, PhotoIcon } from
"@heroicons/react/24/solid";


  // AdminTimeline component
  export function AdminTimeline() {
    return (
```

```
      <div className="w-full max-w-[32rem] p-6 sm:px-8">
        <Timeline>
          <TimelineItem>
            <TimelineConnector />
            <TimelineHeader>
              <TimelineIcon className="p-2">
                <HomeIcon className="h-4 w-4" />
              </TimelineIcon>
              <Typography variant="h5" color="blue-gray"
className="text-lg lg:text-xl" {...({} as React.ComponentProps<typeof
Typography>)}>
                Welcome, Admin, to Your Dashboard!
              </Typography>
            </TimelineHeader>
            <TimelineBody className="pb-8 text-left">
              <Typography color="gray" className="text-sm lg:text-base
font-normal text-gray-600" {...({} as React.ComponentProps<typeof
Typography>)}>
                Welcome to ShelfSpace, Admin! Thank you for signing up and
joining our community. We&#39;re excited to have you on board! Your admin
dashboard is your central hub—manage the eBook collection, upload new
content, and ensure everything runs smoothly. Let&apos;s get started on
making the platform even better!
              </Typography>
            </TimelineBody>
          </TimelineItem>
          <TimelineItem>
            <TimelineConnector />
            <TimelineHeader>
              <TimelineIcon className="p-2">
```

```
              <BookOpenIcon className="h-4 w-4" />
            </TimelineIcon>
            <Typography variant="h5" color="blue-gray"
className="text-lg lg:text-xl" {...({} as React.ComponentProps<typeof
Typography>)}>
              Manage eBook Collection
            </Typography>
          </TimelineHeader>
          <TimelineBody className="pb-8">
            <Typography color="gray" className="text-sm lg:text-base
font-normal text-gray-600" {...({} as React.ComponentProps<typeof
Typography>)}>
              As an admin, you can easily upload new eBooks using the
eBook form, expanding the library for users to enjoy! Simply fill out the
form to add fresh content and make it available to everyone. You also have
the ability to delete any unwanted entries, keeping the collection
organized and up-to-date.
            </Typography>
          </TimelineBody>
        </TimelineItem>
        <TimelineItem>
          <TimelineHeader>
            <TimelineIcon className="p-2">
              <PhotoIcon className="h-4 w-4" />
            </TimelineIcon>
            <Typography variant="h5" color="blue-gray"
className="text-lg lg:text-xl" {...({} as React.ComponentProps<typeof
Typography>)}>
              Personalize Your Admin Profile!
            </Typography>
```

```
            </TimelineHeader>
            <TimelineBody>
              <Typography color="gray" className="text-sm lg:text-base
font-normal text-gray-600" {...({} as React.ComponentProps<typeof
Typography>)}>
                Check out your profile tab to personalize your admin
account! You can easily edit your avatar to reflect your unique style.
Make your profile truly yours and add a personalized touch. Visit your
profile now and make it stand out as an admin!
              </Typography>
            </TimelineBody>
          </TimelineItem>
        </Timeline>
      </div>
    );
  }

// Render the component client-side
'use client';

// Import React hooks and import d3 library for data visualization
import { useEffect, useRef } from 'react';
import * as d3 from 'd3';

// Define the data fetched from the API
interface Point {
  title: string;
  download_count: number;
}
```

```
// Specify that 'data' is an array of Point objects
const BarChart = ({ data }: { data: Point[] }) => {
  const svgRef = useRef<SVGSVGElement | null>(null); // Create a reference
to the SVG element for drawing the bar chart

  useEffect(() => {
    // If data is empty do not render an empty bar chart
    if (data.length === 0) return;

    // Define bar chart dimensions and margins
    const width = 640;
    const height = 400;
    const marginTop = 20;
    const marginRight = 0;
    const marginBottom = 30;
    const marginLeft = 40;

    // Map book titles on the x-axis
    const x = d3.scaleBand()
      .domain(data.map(d => d.title)) // Use book titles for the x-axis
      .range([marginLeft, width - marginRight]) // Sets the scale from
left margin to available width.
      .padding(0.3); // Padding between bars

    // Mapping download count to the y-axis
    const y = d3.scaleLinear()
```

```
      .domain([0, d3.max(data, d => d.download_count) || 0]) // Use the
max download count for y-axis
      .nice() // Adjusts the scale to use rounded values
      .range([height - marginBottom, marginTop]); //Sets the scale from
bottom margin to top margin.


    // Select the SVG element and set its attributes for size, style, and
appearance
    const svg = d3.select(svgRef.current)
      .attr("viewBox", [0, 0, width, height])
      .attr("style", "max-width: 100%; height: auto; font: 10px
sans-serif; overflow: visible;");


    // Plot the 'data' array on the bar chart, using the 'title' as a
unique identifier for each bar
    const bars = svg.selectAll("rect")
      // @ts-expect-error data type d is unknown[] causing typescript to
raise error over title property not being guaranteed
      .data(data, d => d.title);


    // Create and animate bars with transition effects
    bars.join("rect")
      // @ts-expect-error data type d is unknown[] causing typescript to
raise error over title property not being guaranteed
      .attr("x", d => x(d.title))
      .attr("y", d => y(d.download_count))
      .attr("height", d => y(0) - y(d.download_count))
      .attr("width", x.bandwidth())
      .attr("fill", "red")
```

```
      .transition()
      .duration(750)
      .delay((d, i) => i * 20);

    // Add Y-axis and style the text
    svg.append("g")
      .attr("transform", `translate(${marginLeft},0)`)
      // @ts-expect-error data type d is unknown[] causing typescript to
raise error over title property not being guaranteed
      .call(d3.axisLeft(y).tickFormat(d => `${d / 1000}k`))
      .call(g => g.select(".domain").remove())
      .selectAll("text")
      .style("fill", "white");

    // Remove x-axis labels to avoid overlapping text
    svg.selectAll(".x-axis").remove();

    // Style and position the information box
    const tooltip = d3.select("body")
      .append("div")
      .style("position", "absolute")
      .style("background-color", "#212121")
      .style("color", "#fff")
      .style("padding", "10px")
      .style("font-size", "9px")
      .style("border-radius", "10px")
      .style("visibility", "hidden");
```

```
    // Show information box on hover
    bars.on("mouseenter", function (event, d) {
      tooltip
        .style("visibility", "visible")
        .html(`Book Title: ${d.title}<br>Downloads: ${d.download_count}`);
    })
    // Information box position on hover
    .on("mousemove", function (event) {
      tooltip
        .style("top", (event.pageY + 10) + "px")
        .style("left", (event.pageX - 40) + "px");
    })
    // Information box disappears on mouse off
    .on("mouseleave", function () {
      tooltip.style("visibility", "hidden");
    });

  }, [data]); // Re-run effect when `data` changes

  return (
    <div className='min-h-80 lg:min-h-screen mt-20'>
      <svg ref={svgRef}></svg> {/* Render the bar chart */}
    </div>
  );
};

export default BarChart; // Export the BarChart component
```

```
// Render the component client-side
"use client";

// Import Necessary modules
import CookieConsent from "react-cookie-consent";

// CookieConsentBanner component that displays a banner asking users to
consent to cookie usage
const CookieConsentBanner = () => {
  return (
    <CookieConsent
      location="bottom"
      buttonText="I agree"
      cookieName="user-consent"
      style={{
        background: "#021728",
        color: "#fff",
        fontSize: "14px",
        padding: "6px",
        textAlign: "center",
      }}
      buttonStyle={{
        background: "#ff0000",
        color: "#fff",
        fontSize: "13px",
        borderRadius: "5px",
        padding: "8px 14px",
```

```
      }}
      expires={182}
    >
      This website uses cookies to ensure you get the best experience on
our website.{" "}
      <a href="/privacy" className="underline hover:text-gray-300
hover:focus:text-gray-300">
        Learn more
      </a>
    </CookieConsent>
  );
};

export default CookieConsentBanner; // Export the CookieConsentBanner
component

// Import React hooks and import necessary modules
import React from "react";
import {
  Accordion,
  AccordionHeader,
  AccordionBody,
} from "@material-tailwind/react"; // Import Material Tailwind Accordion
components

// DefaultAccordion component
export function DefaultAccordion() {
  // State to manage which accordion is open
```

```
  const [open, setOpen] = React.useState(1);


  // Function to handle opening and closing the accordion
  // @ts-expect-error raising unnecessary error
  const handleOpen = (value) => setOpen(open === value ? 0 : value); //
Toggle the opened state based on value


  return (
    <>
      {/* Accordion 1 */}
      <Accordion {...({} as React.ComponentProps<typeof Accordion>)}
open={open === 1}>
        <AccordionHeader
          className="bg-custom-black text-white text-sm lg:text-lg"
          onClick={() => handleOpen(1)} {...({} as
React.ComponentProps<typeof AccordionHeader>)}
        >
          What is Project Gutenberg?
        </AccordionHeader>
        <AccordionBody className="bg-custom-black text-white text-xs
lg:text-base">
          Project Gutenberg is a library of over 70,000 free eBooks.
Project
          Gutenberg was the first provider of free electronic books, or
eBooks.
          Michael Hart, founder of Project Gutenberg, invented eBooks in
1971 and
          his memory continues to inspire the creation of eBooks and
related
```

```
          content today. Project Gutenberg is a volunteer effort to
digitize and
          archive cultural works, as well as to &quot;encourage the
creation and
          distribution of eBooks.&quot;
        </AccordionBody>
      </Accordion>


      {/* Accordion 2 */}
      <Accordion {...({} as React.ComponentProps<typeof Accordion>)}
open={open === 2}>
        <AccordionHeader
          className="bg-custom-black text-white text-sm lg:text-lg"
          onClick={() => handleOpen(2) } {...({} as
React.ComponentProps<typeof AccordionHeader>)}
        >
          What is the Gutendex API?
        </AccordionHeader>
        <AccordionBody className="bg-custom-black text-white text-xs
lg:text-base">
          Gutendex is a simple, self-hosted web API for serving book
catalog
          information from Project Gutenberg, an online library of free
ebooks.
          Project Gutenberg can be a useful source of literature, but its
large
          size makes it difficult to access and analyse it on a large
scale.
          Thus, an API of its catalog information is useful for automating
these
```

```
          tasks.
        </AccordionBody>
      </Accordion>

      {/* Accordion 3 */}
      <Accordion {...({} as React.ComponentProps<typeof Accordion>)}
open={open === 3}>
        <AccordionHeader
          className="bg-custom-black text-white text-sm lg:text-lg"
          onClick={() => handleOpen(3)} {...({} as
React.ComponentProps<typeof AccordionHeader>)}
        >
          How does the Gutendex API work?
        </AccordionHeader>
        <AccordionBody className="bg-custom-black text-white text-xs
lg:text-base">
          Gutendex uses Django to download catalog data and serve it in a
simple
          JSON REST API.
          <br />
          Project Gutenberg has no such public API of its own, but it
publishes
          nightly archives of complicated XML files. Gutendex downloads
these
          files, stores their data in a database, and publishes the data
in a
          simpler format.
        </AccordionBody>
      </Accordion>
```

```
      {/* Accordion 4 */}
      <Accordion {...({} as React.ComponentProps<typeof Accordion>)}
open={open === 4}>
        <AccordionHeader
          className="bg-custom-black text-white text-sm lg:text-lg"
          onClick={() => handleOpen(4)} {...({} as
React.ComponentProps<typeof AccordionHeader>)}
        >
          Our Mission Statement
        </AccordionHeader>
        <AccordionBody className="bg-custom-black text-white text-xs
lg:text-base">
          Our mission is to empower adults in Ireland by providing
accessible,
          diverse, and engaging eBooks that foster a love of reading and
promote
          lifelong learning. We strive to enhance literacy rates, offering
a wide
          range of resources that support personal growth, education, and
          empowerment, ensuring that every individual has the opportunity
to
          improve their reading skills and broaden their knowledge.
        </AccordionBody>
      </Accordion>
    </>
  );
}
```

```
// Import React hooks and necessary modules
import React from "react";
import { Button, Select, Option } from "@material-tailwind/react";

// Define an interface for the ebook object
interface Ebook {
  id: number;
  title: string;
}

// Define the interface for DeleteEbookForm
interface DeleteEbookForm {
  ebooks: Ebook[]; // List of ebooks to display in the dropdown
  ebookId: number | null; // The currently selected ebook ID
  onDelete: (ebookId: number) => void; // Callback function to handle ebook deletion
  onSelect: (ebookId: number) => void; // Callback function to handle ebook selection
}

// DeleteEbookForm component handles ebook selection and deletion using properties
const DeleteEbookForm: React.FC<DeleteEbookForm> = ({ ebooks, ebookId, onDelete, onSelect }) => {
  // handleDelete function calls the onDelete callback and reloads the page after deletion
  const handleDelete = (ebookId: number) => {
    onDelete(ebookId);
    window.location.reload();
```

```
  };


  return (
    <div className="flex flex-col items-center justify-center w-full
max-w-xl mx-auto p-4">
      {/* Header */}
      <h2 className="text-xl lg:text-2xl font-semibold mb-4 text-center
underline">Delete Ebook</h2>


      {/* Select dropdown */}
      <Select
        label="Select Ebook to Delete"
        value={ebookId?.toString() || ""} // Set the value to the
currently selected ebook ID
        onChange={(e) => onSelect(Number(e))} // When a new option is
chosen update selection
        className="mb-4 w-full" {...({} as React.ComponentProps<typeof
Select>)}
      >
        {/* Create an option for each ebook  */}
        {ebooks.map((ebook) => (
          <Option key={ebook.id} value={ebook.id.toString()}>
            {ebook.title} {/* Display ebook title */}
          </Option>
        ))}
      </Select>


      {/* Show delete button on selection of ebook */}
      {ebookId && (
```

```
        <div className="flex justify-center w-2/5 lg:w-full mt-6">
          <Button color="red" onClick={() => handleDelete(ebookId)}
className="w-full sm:w-auto" {...({} as React.ComponentProps<typeof
Button>)}>
            Delete Ebook
          </Button>
        </div>
      )}
    </div>
  );
};

export default DeleteEbookForm; // Export the DeleteEbookForm component

// Import necessary modules
import Image from 'next/image';
import {
  Card,
  CardBody,
  CardFooter,
  Typography,
  Button,
} from "@material-tailwind/react";

// CardWithLink component
export function CardWithLink() {
  return (
```

```
    <Card className="mt-6 w-96" {...({} as React.ComponentProps<typeof
Card>)}> {/* Define the card container */}
      <CardBody {...({} as React.ComponentProps<typeof CardBody>)}>
      <Image
        src="/images/document-logo.svg"
        alt="logo-ct"
        className="w-14 h-16"
        width={56}
        height={64}
      />

        <Typography
          variant="h5"
          color="blue-gray"
          className="mt-2 mb-2" {...({} as React.ComponentProps<typeof
Typography>)}
        >
          Project Documentation
        </Typography> {/* Title of the card */}

        <Typography {...({} as React.ComponentProps<typeof Typography>)}>
          Explore the project documentation to learn more about the
features, development process, and technical details of the eBook website.
        </Typography> {/* Description of the card */}
      </CardBody>

      <CardFooter className="pt-0" {...({} as React.ComponentProps<typeof
CardFooter>)}> {/* Footer section of the card */}
```

```
        <a href="/documentation" className="inline-block">
          <Button
            size="sm"
            variant="text"
            className="flex items-center gap-2" {...({} as
React.ComponentProps<typeof Button>)}
          >
            Learn More


            <svg
              xmlns="http://www.w3.org/2000/svg"
              fill="none"
              viewBox="0 0 24 24"
              strokeWidth={2}
              stroke="currentColor"
              className="h-4 w-4"
            >
              <path
                strokeLinecap="round"
                strokeLinejoin="round"
                d="M17.25 8.25L21 12m0 0l-3.75 3.75M21 12H3"
              />
            </svg>
          </Button>
        </a>
      </CardFooter>
    </Card>
  );
```

```
}


// DocumentationViewer component
export function DocumentationViewer() {
  return (
    <div className="flex flex-col relative pb-10 lg:pb-14 lg:mt-3
items-center justify-center bg-custom-gray overflow-hidden">
      <a
        href="/Project_Living_Document.pdf"
        target="_blank"
        rel="noopener noreferrer"
        className="text-xs lg:text-base font-normal lg:font-medium pb-6
text-red-500 underline mt-6 block hover:text-red-400"
      >
      Open PDF in a new window
      </a>


      {/* Embeds the PDF document inside an iframe */}
      <iframe
        src="/Project_Living_Document.pdf"
        className="w-full sm:w-11/12 md:w-4/5 h-[75vh] sm:h-[90vh]
md:h-[100vh] lg:h-[125vh] max-w-5xl rounded-lg shadow-lg"
        style={{
          minHeight: '65vh',
          width: '85%',
          height: '100%',
          border: 'none',
        }}
```

```
        title="Documentation Viewer"
      />
    </div>
  );
}



// Import React hooks
import React, { useState, useEffect } from "react";

// Defining the props interface for EbookReader component
interface EbookReaderProps {
  ebookId: number;
}

// Define an interface for the EbookData object
interface EbookData {
  id: number;
  title: string;
  description: string;
  publishedAt: Date;
  fileUrl: string; // URL to the file in Supabase storage bucket
}

// EbookReader component for displaying an ebook based on the ebookId
const EbookReader: React.FC<EbookReaderProps> = ({ ebookId }) => {
  // State to store the ebook data fetched from the API
```

```
  const [ebookData, setEbookData] = useState<EbookData | null>(null);

  // State to handle loading and error messages
  const [status, setStatus] = useState<{ loading: boolean; error: string | null }>({
    loading: true,
    error: null,
  });

  // This state will hold the fetched file content
  const [fileContent, setFileContent] = useState<string | null>(null);

  // useEffect hook to fetch ebook data when the ebookId changes
  useEffect(() => {
    const fetchEbook = async () => {
      setStatus({ loading: true, error: null }); // Reset loading and error states
      try {
        // Fetches data from the API and updates the loading and error states
        const response = await fetch(`/api/reader?id=${ebookId}`);
        if (!response.ok) {
          throw new Error("Failed to fetch ebook data");
        }

        // Parses the fetched ebook data and updates the state
        const data: EbookData = await response.json();
        setEbookData(data);
```

```
      setStatus({ loading: false, error: null });

      // Fetch the file content from the URL
      const fileResponse = await fetch(data.fileUrl);
      if (!fileResponse.ok) {
        throw new Error("Failed to fetch file content");
      }

      // Assume the file is HTML
      const fileText = await fileResponse.text();
      setFileContent(fileText); // Store the file content in state
    } catch {
      setStatus({ loading: false, error:"Failed to fetch ebook data" });
    }
  };

  fetchEbook();
}, [ebookId]); // Runs when ebookId changes

// Show a loading message
if (status.loading) return <div>Loading...</div>;

// If there's an error, show the error message
if (status.error) return <div>Error: {status.error}</div>;

return (
  <div className="text-center justify-center min-h-screen">
```

```
      {/* If fileUrl is present, display the ebook content */}
      {ebookData?.fileUrl ? (
        <div>
          {/* If file content is loaded, render it in an iframe */}
          {fileContent ? (
            <iframe
              srcDoc={fileContent} // Using srcDoc for inline HTML
rendering
              width="100%"
              height="600"
              title={ebookData.title}
            />
          ) : (
            <div>Loading file content...</div>
          )}
        </div>
      ) : (
        <div className="text-red-500 py-4">No file available for this
ebook</div>
      )}
    </div>
  );
};

export default EbookReader; // Export the EbookReader component

// Render the component client-side
"use client";
```

```
// Import necessary modules
import Image from 'next/image';
import { Typography } from "@material-tailwind/react";

// Footer Component
export function Footer() {
  return (
    // Footer container
    <footer className="w-full bg-gray-50">
      {/* Top section of the footer */}
      <div className="flex flex-row items-center flex-wrap justify-between
bg-gray-50 text-black text-center p-6">
        {/* Logo Section */}
        <div className="flex justify-start">
        <Image
          src="/images/logo.svg"
          alt="logo-ct"
          className="w-8 h-6"
          width={32}
          height={24}
        />
        </div>

        {/* Navigation Links */}
        <ul className="flex flex-wrap items-center justify-center py-0
lg:py-2 gap-y-2 gap-x-6 flex-1">
          <li>
```

```
            <Typography
              as="a"
              href="/visualisations"
              color="blue-gray"
              className="text-sm font-normal transition-colors
hover:text-gray-500 focus:text-gray-500" {...({} as
React.ComponentProps<typeof Typography>)}
            >
              Visualisations
            </Typography>
          </li>
          <li>
            <Typography
              as="a"
              href="/libraries"
              color="blue-gray"
              className="text-sm font-normal transition-colors
hover:text-gray-500 focus:text-gray-500" {...({} as
React.ComponentProps<typeof Typography>)}
            >
              Libraries
            </Typography>
          </li>
          <li>
            <Typography
              as="a"
              href="/documentation"
              color="blue-gray"
```

```
              className="text-sm font-normal transition-colors
hover:text-gray-500 focus:text-gray-500" {...({} as
React.ComponentProps<typeof Typography>)}
            >
              Documentation
            </Typography>
          </li>
          <li>
            <Typography
              as="a"
              href="/about"
              color="blue-gray"
              className="text-sm font-normal transition-colors
hover:text-gray-500 focus:text-gray-500" {...({} as
React.ComponentProps<typeof Typography>)}
            >
              About
            </Typography>
          </li>
        </ul>
      </div>

      {/* Divider */}
      <hr className="border-blue-gray-50" />

      {/* Footer bottom text */}
      <Typography
        color="blue-gray"
```

```
        className="text-center text-sm font-normal text-black py-4"
{...({} as React.ComponentProps<typeof Typography>)}
      >
        &copy; 2024 ShelfSpace
      </Typography>
    </footer>
  );
}


// Render the component client-side
"use client";


// Import the components from the components directory and import
necessary modules
import Image from 'next/image';
import { useEffect } from "react";
import { CardDefault } from "./InfoCard";
import { CarouselCustomNavigation } from "./ImageCarousel";
import { CardWithLink } from "./DocumentCard";
import { TestimonialCard } from "./UserReviewCard";
import { HorizontalCard } from "./PreviewCard";
import { gsap } from "gsap";
import { ScrollTrigger } from "gsap/dist/ScrollTrigger";


// Hero Component
export default function Hero() {
  // Scroll to a section function triggered by the up arrow click
  const scrollToSection = (sectionId: string) => {
```

```
  const sectionElement = document.getElementById(sectionId);
  if (sectionElement) {
    sectionElement.scrollIntoView({ behavior: "smooth" });
  }
};

useEffect(() => {
  // Register the ScrollTrigger plugin for scroll-based animations
  gsap.registerPlugin(ScrollTrigger);

  // Animate <h1> in #textSection when it comes into view
  gsap.fromTo(
    "#textSection h1",
    {
      opacity: 0, // Start as invisible
      y: 50, // Animation starting place
    },
    {
      opacity: 1, // End with full opacity
      y: 0, // Move to original position
      duration: 3, // Duration of animation
      ease: "power1.out", // Ease for smooth animation
      scrollTrigger: {
        trigger: "#textSection", // Animation trigger
        start: "top 75%", // Animation starts
        once: true, // Animation runs only once
      },
```

```
    }
  );
  // Animate <h1> in #homeSection when it comes into view
  gsap.fromTo(
    "#homeSection h1",
    {
      opacity: 0, // Start as invisible
      y: 50, // Animation starting place
    },
    {
      opacity: 1, // End with full opacity
      y: 0, // Move to original position
      duration: 2, // Duration of animation
      ease: "power1.out", // Ease for smooth animation
      scrollTrigger: {
        trigger: "#homeSection", // Trigger animation when #imageSection
is in view
        start: "top 75%", // Animation starts
        once: true, // Animation runs only once
      },
    }
  );

  // Animate <p> in #imageSection when it comes into view
  gsap.fromTo(
    "#homeSection p",
    {
      opacity: 0, // Start as invisible
```

```
    y: 30, // Animation starting place
  },
  {
    opacity: 1, // End with full opacity
    y: 0, // Move to original position
    duration: 1, // Duration of animation
    delay: 0.2, // Delay after <h1> animation
    ease: "power1.out", // Ease for smooth animation
    scrollTrigger: {
      trigger: "#homeSection", // Animation trigger
      start: "top 75%", // Animation starts
      once: true, // Animation runs only once
    },
  }
);

// Animate <h1> in #imageSection when it comes into view
gsap.fromTo(
  "#imageSection h1",
  {
    opacity: 0, // Start as invisible
    y: 50, // Animation starting place
  },
  {
    opacity: 1, // End with full opacity
    y: 0, // Move to original position
    duration: 3, // Duration of animation
```

```
    ease: "power1.out", // Ease for smooth animation
    scrollTrigger: {
      trigger: "#imageSection", // Animation trigger
      start: "top 75%", // Animation starts
      once: true, // Animation runs only once
    },
  }
);

// Animate <p> in #textSection when it comes into view
gsap.fromTo(
  "#textSection p",
  {
    opacity: 0, // Start as invisible
    y: 30, // Animation starting place
  },
  {
    opacity: 1, // End with full opacity
    y: 0, // Move to original position
    duration: 3, // Duration of animation
    delay: 0.2, // Delay after <h1> animation
    ease: "power1.out", // Ease for smooth animation
    scrollTrigger: {
      trigger: "#textSection", // Animation trigger
      start: "top 75%", // Animation starts
      once: true, // Animation runs only once
    },
```

```
    }
  );

  // Animate <p> in #imageSection when it comes into view
  gsap.fromTo(
    "#imageSection p",
    {
      opacity: 0, // Start as invisible
      y: 30, // Animation starting place
    },
    {
      opacity: 1, // End with full opacity
      y: 0, // Move to original position
      duration: 2, // Duration of animation
      delay: 0.2, // Delay after <h1> animation
      ease: "power1.out", // Ease for smooth animation
      scrollTrigger: {
        trigger: "#imageSection", // Animation trigger
        start: "top 75%", // Animation starts
        once: true, // Animation runs only once
      },
    }
  );
}, []); // Empty dependency array ensures this effect runs only once

return (
  <div className="bg-gray-50">
```

```
    {/* Scroll Up Arrow */}
    <Image
      className="fixed right-6 bottom-6 lg:justify-end h-8 w-auto z-50
animate-bounce cursor-pointer"
      src="/images/arrow-up.svg"
      alt="Arrow Up"
      width={32}
      height={32}
      onClick={() => scrollToSection("homeSection")} // Call
scrollToSection when clicked
    />


      {/* Hero Section */}
      <div className="relative isolate px-6 pt-0 lg:px-48 bg-hero-pattern
bg-no-repeat bg-center bg-cover lg:min-h-screen w-full" id="homeSection">
        <div className="mx-auto max-w-full">
          <div className="flex items-center justify-between pt-2">
            {/* Intro Header */}
            <div className="mt-24 text-left pb-24">
              {/* Hero Heading */}
              <h1 className="font-sans font-bold tracking-tight text-white
text-3xl sm:text-5xl md:text-6xl lg:text-7xl 2xl:text-8xl 2xl:mt-52">
                CHANGE THE WAY YOU READ
              </h1>


              {/* Hero Subheading */}
              <p className="mt-8 text-sm sm:text-lg md:text-xl lg:text-xl
2xl:text-3xl max-w-sm sm:max-w-md md:max-w-lg leading-7 sm:leading-8
text-white font-normal 2xl:mt-20">
```

```
              Explore a vast catalogue of ebooks. Sign up for free.
            </p>


            {/* Link Buttons */}
            <div className="mt-20 flex items-center justify-start
gap-x-6">
              {/* Get Started Button */}
              <a
                href="/register"
                className="rounded-full bg-black px-5 py-2 text-sm
2xl:text-2xl font-semibold text-white shadow-sm hover:bg-gray-800
2xl:mt-20">
                Get started
              </a>


              {/* Learn More Button */}
              <a
                href="/about"
                className="px-5 py-2 text-sm 2xl:text-2xl font-semibold
leading-6 text-black hover:focus:text-gray-600 hover:text-gray-600
hover:underline 2xl:mt-20">
                Learn more <span aria-hidden="true">→</span>
              </a>
            </div>
          </div>
        </div>
      </div>
```

```
      {/* Cards Section */}
      <div className="relative bg-gray-50 min-h-screen lg:mb-[-28rem]
pt-24 pb-24">
        {/* Responsive Grid for Cards */}
        <section
          className="grid grid-cols-1 lg:grid-cols-3 lg:grid-rows-3
lg:grid-flow-col gap-4 justify-center items-center mx-4 mb-[-2rem] lg:mb-0
lg:pb-10 sm:mx-10 lg:mx-20" >
          {/* First Card */}
          <div className="rounded-2xl lg:row-span-3 bg-white
lg:mt-[-40rem] min-h-64 sm:min-h-[16rem] lg:min-h-[20rem] shadow-2xl
text-sm font-semibold text-black flex items-center justify-center">
            <CardDefault />
          </div>


          {/* Second Card */}
          <div className="rounded-2xl lg:col-span-2 bg-white min-h-64
sm:min-h-[16rem] lg:min-h-[20rem] shadow-2xl text-sm font-semibold
text-black flex items-center justify-center">
            <HorizontalCard />
          </div>


          {/* Third Card */}
          <div className="rounded-2xl lg:col-span-3 bg-white min-h-48
sm:min-h-[16rem] lg:min-h-[20rem] shadow-2xl text-sm font-semibold
text-black flex items-center justify-center">
            <CarouselCustomNavigation />
          </div>
        </section>
      </div>
```

```
      {/* Text Section */}
      <div className="relative bg-custom-gradient min-h-[600px]
sm:min-h-screen pt-2 pb-14 w-full" id="textSection">
        <div className="flex justify-center items-center text-right
mx-auto py-16 px-4 sm:py-14 sm:px-6 lg:px-8">
          <div className="max-w-screen-md px-4">
            {/* Hero Heading */}
            <h1 className="text-center text-4xl text-white font-black
tracking-normal sm:text-5xl lg:text-5xl">
              READ, EXPLORE, AND ENGAGE IN A WHOLE NEW WAY!
            </h1>

            {/* Hero Subheading */}
            <p className="mt-16 max-w-2xl mx-auto text-justify text-white
text-sm sm:text-lg 2xl:text-4xl leading-7 font-sans font-bold">
              Experience a new way to read and engage with books like
never before! Our platform combines cutting-edge technology to offer a
personalized reading experience, featuring interactive elements, real-time
updates, and a community-driven approach. Dive deeper into the world of
literature with dynamic data visualizations powered by data from Project
Gutenberg, offering insights into trends, popular titles, and reading
patterns. Whether you&#39;re reading for pleasure or learning, our
innovative approach makes exploring eBooks more immersive, engaging, and
data-driven than traditional methods. Join today to unlock a whole new
dimension of digital reading!
            </p>
          </div>
        </div>
      </div>
```

```
      <div className="flex items-center justify-center w-full px-4 pb-12">
        <CardWithLink />
      </div>

      {/* Book Cover Container Section */}
      <div className="relative min-h-screen pt-2" id="imageSection">
        <div className="mx-auto py-6 px-4 sm:py-16 sm:px-6 lg:px-8">
          <div className="px-4">
            {/* Image Section Heading */}
            <h1 className="text-center text-4xl sm:text-4xl md:text-5xl
lg:text-5xl 2xl:text-7xl font-black tracking-normal 2xl:mt-14">
              DISCOVER A WORLD OF KNOWLEDGE
            </h1>

            {/* Book Cover Section Description */}
            <p className="text-justify mt-6 sm:mt-8 md:mt-10 2xl:mt-20
max-w-lg sm:max-w-x1 md:max-w-2xl mx-auto text-sm sm:text-lg md:text-xl
lg:text-base 2xl:text-3xl leading-6 sm:leading-7 lg:leading-8 font-sans
font-semibold">
              Looking to explore a world of eBooks at your fingertips? Our
platform offers a seamless experience for discovering, reading, and
enjoying eBooks across all genres. Join a growing community of readers and
access an ever-expanding library of titles, with such classics as seen
below.
            </p>

            {/* Book Cover Grid */}
            <section className="w-full flex justify-center items-center
gap-4 pt-16 2xl:mt-20">
              {/* First Book Cover */}
```

```
            <div className="box z-12 rounded-lg bg-white bg-book-cover1
bg-cover w-full sm:w-1/3 md:w-1/4 lg:w-1/5 min-h-52 lg:h-80 2xl:min-h-96
shadow-2xl text-sm font-semibold text-black flex items-center
justify-center hover:bg-black hover:-translate-y-1 hover:scale-110
duration-300">
            </div>


            {/* Second Book Cover */}
            <div className="box z-11 rounded-lg bg-white bg-book-cover2
bg-cover w-full sm:w-1/3 md:w-1/4 lg:w-1/5 min-h-52 lg:h-80 2xl:min-h-96
shadow-2xl text-sm font-semibold text-black flex items-center
justify-center hover:bg-black hover:-translate-y-1 hover:scale-110
duration-300">
            </div>


            {/* Third Book Cover */}
            <div className="box z-10 rounded-lg bg-white bg-book-cover3
bg-cover w-full sm:w-1/3 md:w-1/4 lg:w-1/5 min-h-52 lg:h-80 2xl:min-h-96
shadow-2xl text-sm font-semibold text-black flex items-center
justify-center hover:bg-black hover:-translate-y-1 hover:scale-110
duration-300">
            </div>
          </section>
        </div>
      </div>
    </div>


    <div className="container mx-auto mt-[-8rem] lg:mt-0 px-4 lg:px-16
py-12">
      {/* Testimonial Cards Container */}
      <div className="space-y-6">
```

```
          <div className="flex rounded-2xl items-center justify-center
min-w-64 shadow-2xl text-sm font-semibold text-black">
            {/* Testimonial Card 1 */}
            <TestimonialCard
              name="John M."
              title="Ebook Explorer"
              imageUrl="/images/avatar-man-1.svg"
              rating={5}
              testimonial="This site has transformed my reading habits!
The ability to read on any device is a game-changer. I can't wait to
explore more titles!"
            />


          </div>
          <div className="flex rounded-2xl items-center justify-center
min-w-64 shadow-2xl text-sm font-semibold text-black">
            {/* Testimonial Card 2 */}
            <TestimonialCard
              name="Saoirse W"
              title="Literature Lover"
              imageUrl="/images/avatar-woman-1.svg"
              rating={5}
              testimonial="I absolutely love this ebook platform! The
selection of books is fantastic, and the reading experience is seamless.
I've been able to find so many hidden gems that I wouldn't have discovered
otherwise. Highly recommend it to any book lover!"
            />
          </div>
          <div className="flex rounded-2xl items-center justify-center
min-w-64 shadow-2xl text-sm font-semibold text-black">
```

```
            {/* Testimonial Card 3 */}
            <TestimonialCard
              name="Deirdre M."
              title="Story Seeker"
              imageUrl="/images/avatar-woman.svg"
              rating={5}
              testimonial="A must-have for any ebook reader! The site is so user-friendly, and I love that I can read books on the go. There's a huge selection, and I always find something new to enjoy. Excellent experience overall!"
            />
          </div>
          <div className="flex rounded-2xl items-center justify-center min-w-64 shadow-2xl text-sm font-semibold text-black">
            {/* Testimonial Card 4 */}
            <TestimonialCard
              name="Michael T."
              title="World Wanderer"
              imageUrl="/images/avatar-man.svg"
              rating={4}
              testimonial="As an avid reader, I've tried many ebook sites, but this one stands out. The books are high quality, the site is fast, and there are always great recommendations."
            />
          </div>
        </div>
      </div>


    </div>// Main container close
```

```
  );
}



"use client";


// Import necessary modules
import { useState } from "react";
import Image from "next/image";
import { Carousel } from "@material-tailwind/react"; // Importing the
Carousel component from Material Tailwind


// CarouselCustomNavigation component
export function CarouselCustomNavigation() {
  const [imagesLoaded, setImagesLoaded] = useState(false); // Track if all
images are loaded


  return (
    <div className="relative w-full h-[400px]"> {/* Container with fixed
height */}
      {!imagesLoaded && (
        <div className="absolute inset-0 flex items-center justify-center
bg-gray-200">
          <span className="text-gray-500">Loading...</span>
        </div>
      )}
      <Carousel
        className={`rounded-xl transition-opacity ${
```

```
          imagesLoaded ? "opacity-100" : "opacity-0"
        }`} {...({} as React.ComponentProps<typeof Carousel>)}
        navigation={({ setActiveIndex, activeIndex, length }) => (
          <div className="absolute bottom-4 left-2/4 z-50 flex
-translate-x-2/4 gap-2">
            {new Array(length).fill("").map((_, i) => (
              <span
                key={i} // Unique key for each navigation dot
                className={`block h-1 cursor-pointer rounded-2xl
transition-all ${
                  activeIndex === i ? "w-8 bg-white" : "w-4 bg-white/50"
                }`}
                onClick={() => setActiveIndex(i)} // Set active index when
UI nav element is clicked
              />
            ))}
          </div>
        )}
      >
        {/* Images in the carousel */}
        <Image
          src="/images/reading-image2.jpg"
          alt="image 1"
          className="w-full h-full object-cover"
          width={500}
          height={500}
          onLoad={() => setImagesLoaded(true)} // Set imagesLoaded to true
once image loads
```

```
        />
        <Image
          src="/images/reading-image.jpg"
          alt="image 2"
          className="w-full h-full object-cover"
          width={500}
          height={500}
          onLoad={() => setImagesLoaded(true)}
        />
        <Image
          src="/images/reading-image1.jpg"
          alt="image 3"
          className="w-full h-full object-cover"
          width={500}
          height={500}
          onLoad={() => setImagesLoaded(true)}
        />
      </Carousel>
    </div>
  );
}

// Render the component client-side
"use client";

// Import Necessary modules
import Image from 'next/image';
```

```
import {
  Card,
  CardHeader,
  CardBody,
  CardFooter,
  Typography,
} from "@material-tailwind/react";

// CardDefault component
export function CardDefault() {
  return (
    <Card className="mt-6 w-96" {...({} as React.ComponentProps<typeof
Card>)}>
      {/* Card Header with image */}
      <CardHeader color="blue-gray" className="relative h-56" {...({} as
React.ComponentProps<typeof CardHeader>)}>
      <Image
        src="/images/map-of-ireland.jpg"
        alt="Map Image"
        width={500}
        height={300}
      />
      </CardHeader>

      {/* Card Body with title and description */}
      <CardBody {...({} as React.ComponentProps<typeof CardBody>)}>
        <Typography variant="h5" color="blue-gray" className="mb-2"
{...({} as React.ComponentProps<typeof Typography>)}>
```

```
          Interactive Libraries Map
        </Typography>
        <Typography {...({} as React.ComponentProps<typeof Typography>)}>
          Check out our Libraries page to explore an interactive map of
Ireland, showcasing library locations.
          Discover nearby libraries, their services, and more, making it
easier to find your next visit!
        </Typography>
      </CardBody>

      {/* Card Footer with 'Learn more' link */}
      <CardFooter className="pt-0 flex justify-center text-center" {...({}
as React.ComponentProps<typeof CardFooter>)}>
        <a
          href="/libraries"
          className="px-5 py-2 text-sm 2xl:text-2xl font-semibold
leading-6 text-black hover:focus:text-gray-600 hover:text-gray-600
hover:underline 2xl:mt-20"
        >
          Learn more <span aria-hidden="true">→</span>
        </a>
      </CardFooter>
    </Card>
  );
}

// Render the component client-side
"use client";
```

```
// Import React hooks, necessary modules and icons
import { useState } from 'react';
import { useRouter } from 'next/navigation';
import {
  Input,
  Button,
} from "@material-tailwind/react";
import { FaEye, FaEyeSlash } from "react-icons/fa";


const LoginForm = () => {
  const [email, setEmail] = useState(''); // State to store the email
  const [password, setPassword] = useState(''); // State to store the
password
  const [showPassword, setShowPassword] = useState(false); // Toggles
password visibility
  const [message, setMessage] = useState(''); // Stores the success or
error message


  // Router hook to navigate after successful login
  const router = useRouter();


  // Handle form submission
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault(); // Prevent default form submission
    setMessage(''); // Clear previous messages


    try {
```

```
      // Sending login request to the API which checks the Postgres
database for existing tables
      const response = await fetch('/api/login', {
        method: 'POST', // Sending POST request to the login API
        headers: {
          'Content-Type': 'application/json', // Setting request body type
        },
        body: JSON.stringify({ email, password }), // Convert email, and
password to a JSON string
      });


      if (response.ok) { // Check if login was successful
        const data = await response.json();
        localStorage.setItem("token", data.token); // Store the JWT token
in localStorage
        setMessage('Login successful!'); // Set success message
        setEmail(''); // Clear email input field
        setPassword(''); // Clear password input field


        // Redirect route based on user role: admin or user
        if (data.isAdmin) {
          router.push('/dashboard/admin'); // Redirect to admin dashboard
        } else {
          router.push('/dashboard/user'); // Redirect to user dashboard
        }
      } else {
        // Handle errors for invalid details
        const errorData = await response.json();
```

```
        setMessage(`Error: ${errorData.error}`);
      }
    } catch {
      // Handle errors
      setMessage('Error: Unable to log in.');
    }
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-6 flex flex-col
items-center">

      {/* Email Input */}
      <div className="w-3/6 pt-2 pb-2">
        <Input
          variant="standard"
          color="white"
          type="email"
          id="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)} // Update email on
change
          required
          label="Email Address" {...({} as React.ComponentProps<typeof
Input>)}
        />
      </div>
```

```
      {/* Password Input */}
      <div className="w-3/6 pb-2 relative">
        <Input
          variant="standard"
          color="white"
          type={showPassword ? "text" : "password"} // Toggle password
visibility
          id="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)} // Update password
on change
          required
          label="Password" {...({} as React.ComponentProps<typeof Input>)}
        />

        {/* Eye Icon */}
        <button
          type="button"
          onClick={() => setShowPassword(!showPassword)} // Toggle
password visibility
          className="absolute top-1/2 right-[-3rem] lg:right-[-2rem]
transform -translate-y-1/2 text-gray-500"
          aria-label="Toggle password visibility"
        >
          {showPassword ? <FaEyeSlash /> : <FaEye />}
        </button>

      </div>
```

```
      {/* No Account Link */}
      <a
        href="/register"
        className="px-5 py-0 text-xs 2xl:text-2xl font-semibold leading-6 text-white hover:focus:text-gray-300 hover:text-gray-300 hover:underline 2xl:mt-20">
        Don&#39;t have an account? <span aria-hidden="true">→</span>
      </a>

      {/* Submit Button */}
      <Button variant="outlined" color="white" type="submit" {...({} as React.ComponentProps<typeof Button>)}>
        Login
      </Button>

      {/* Display error or success message */}
      {message && <p className="mt-2 text-sm text-red-500">{message}</p>}
    </form>
  );
};

export default LoginForm; // Export LoginForm component

// Import necessary modules
import { Button } from "@material-tailwind/react";

// LogoutButton component to handle user logout
```

```
const LogoutButton = () => {
  const logout = () => {
    // Remove token from localStorage
    localStorage.removeItem('token');

    // Redirect the user to the login page
    window.location.href = '/login';
  };

  return (
    <div className="flex flex-col items-center justify-center min-h-screen
p-2">
      <p className="text-center justify-center text-sm lg:text-lg
font-normal text-black mb-4 w-4/5">
        Finished your session? Logout using the button below.
      </p>
      <Button onClick={logout} className="mt-6 w-2/5 lg:w-1/5 sm:w-auto"
{...({} as React.ComponentProps<typeof Button>)}>
        Logout
      </Button>
    </div>
  );
};

export default LogoutButton; // Export LogoutButton component

// Import React hooks, Leaflet library, and Leaflet's CSS for proper
styling
```

```
import React, { useEffect, useRef } from 'react';
import 'leaflet/dist/leaflet.css';
import L from 'leaflet';

// Configure custom marker icons for Leaflet
L.Icon.Default.mergeOptions({
  iconUrl: '/images/marker-icon.png',
  iconRetinaUrl: '/images/marker-icon-2x.png',
  shadowUrl: '/images/marker-shadow.png',
});

// Define data types for GeoJSON feature properties and geometry
interface LibraryProperties {
  Name: string;
  Address1: string;
  Town: string;
  County: string;
  Eircode: string;
  Phone: string;
  Email: string;
  Website: string;
  Opening_Hours_Monday: string;
  Opening_Hours_Tuesday: string;
  Opening_Hours_Wednesday: string;
  Opening_Hours_Thursday: string;
  Opening_Hours_Friday: string;
  Opening_Hours_Saturday: string;
```

```
}


interface LibraryGeometry {
  coordinates: [number, number]; // [longitude, latitude]
}


// Represents a library with its geometry and properties
interface LibraryFeature {
  geometry: LibraryGeometry;
  properties: LibraryProperties;
}


// References for the map container and Leaflet map instance
const MapComponent: React.FC = () => {
  const mapRef = useRef<HTMLDivElement | null>(null); // Reference to the
map container element
  const mapInstance = useRef<L.Map | null>(null); // Holds the Leaflet map
instance


  // Array of GeoJSON file paths to load library locations
  const geojsonFiles = [
    '/geojson/wexford_libraries.geojson',
    '/geojson/Libraries_Roscommon.geojson',
    '/geojson/Wicklow_Libraries_WIW.geojson',
    '/geojson/librarylocationsdlr.geojson',
    '/geojson/Library_Services_SDCC.geojson',
    '/geojson/Libraries_FCC.geojson',
    '/geojson/GCC_Libraries.geojson',
```

```
    '/geojson/cork-city-council-libraries.geojson',
  ];

  useEffect(() => {
    // Initialize the map when the component loads
    if (mapRef.current && !mapInstance.current) {
      // Create a Leaflet map instance and set the initial view and zoom
level
      mapInstance.current = L.map(mapRef.current).setView([52.655778,
-6.65652], 8);

      // Add OpenStreetMap tile layer to the map
      L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; OpenStreetMap contributors',
      }).addTo(mapInstance.current);

      // Function to load and process GeoJSON data
      const loadGeojson = (filePath: string) => {
        fetch(filePath) // Fetch GeoJSON file
          .then((response) => response.json()) // Parse the response as
JSON
          .then((geojsonData: { features: LibraryFeature[] }) => {
            // Iterate through each feature in the GeoJSON data
            geojsonData.features.forEach((feature) => {
              const { coordinates } = feature.geometry; // Extract
coordinates (longitude, latitude)
              const {
                Name,
```

```
                Address1,
                Town,
                County,
                Eircode,
                Phone,
                Email,
                Website,
                Opening_Hours_Monday,
                Opening_Hours_Tuesday,
                Opening_Hours_Wednesday,
                Opening_Hours_Thursday,
                Opening_Hours_Friday,
                Opening_Hours_Saturday,
              } = feature.properties; // Extract other properties, the
library details


              // Create popup content box with relevant library details
              const popupContent = `
                <div style="overflow-y: scroll; max-width: 200px;
max-height: 200px; word-wrap: break-word; padding: 12px;">
                  <h3 style="text-decoration: underline; font-size: 15px;
font-weight: bold;">${Name}</h3>
                  <p><strong>Address:</strong> ${Address1}, ${Town},
${County}, ${Eircode}</p>
                  <p><strong>Phone:</strong> ${Phone}</p>
                  <p><strong>Email:</strong> <a
href="mailto:${Email}">${Email}</a></p>
                  <p><strong>Website:</strong> <a href="${Website}"
target="_blank">${Website}</a></p>
```

```
                <p><strong>Opening Hours (Monday):</strong>
${Opening_Hours_Monday}</p>
                <p><strong>Opening Hours (Tuesday):</strong>
${Opening_Hours_Tuesday}</p>
                <p><strong>Opening Hours (Wednesday):</strong>
${Opening_Hours_Wednesday}</p>
                <p><strong>Opening Hours (Thursday):</strong>
${Opening_Hours_Thursday}</p>
                <p><strong>Opening Hours (Friday):</strong>
${Opening_Hours_Friday}</p>
                <p><strong>Opening Hours (Saturday):</strong>
${Opening_Hours_Saturday}</p>
                <p><strong>Opening Hours (Sunday):</strong> Closed</p>
                <p><strong>Opening Hours (Bank Holidays):</strong>
Closed</p>
              </div>
            `;

            // Add a marker on the map for each feature's coordinates
            L.marker([coordinates[1], coordinates[0]]) // Create a
marker at the given coordinates
              .addTo(mapInstance.current!) // Add marker to the map
              .bindPopup(popupContent); // Bind the popup content to the
marker
          });
        })
    };

    // Load all the GeoJSON files
    geojsonFiles.forEach((file) => {
```

```
        loadGeojson(file);
      });
    }

    // Clean up the map instance when the component is disconneted
    return () => {
      if (mapInstance.current) {
        mapInstance.current.remove(); // Remove the map instance
        mapInstance.current = null; // Set the map instance to null
      }
    };
  }, []); // Empty dependency array ensures this effect runs once on load

  return (
    // Map container element
    <div
      ref={mapRef}
      className="w-full rounded-lg overflow-hidden h-[100vw] sm:h-[50vw]
lg:h-[40vw]" // Tailwind classes for styling
    />
  );
};

export default MapComponent; // Export the MapComponent component

// Render the component client-side
"use client";
```

```
// Import React hooks and import the Link component for navigation between
pages
import Image from 'next/image';
import React, { useState, useEffect } from "react";
import Link from "next/link";


const Navbar = () => {
  // State for showing/hiding the navbar on scroll
  const [isVisible, setIsVisible] = useState(true);
  // State to track the last scroll position
  const [lastScrollY, setLastScrollY] = useState(0);
  // State to control the navbar background colour
  const [bgColor, setBgColor] = useState("bg-transparent");
  // State to control font colour
  const [fontColor, setFontColor] = useState("text-white");
  // State to control button text colour
  const [buttonColor, setButtonColor] = useState("text-white");
  // State to track the hamburger menu open/close state
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  // State to control the logo source
  const [logoSrc, setLogoSrc] = useState("/images/home_icon_w.png");


  // Handle scroll to change navbar styles based on scroll position
  useEffect(() => {
    const handleScroll = () => {
      const currentScrollY = window.scrollY;


      if (currentScrollY === 0) {
```

```
      // Reset styles when at the top of the page
      setBgColor("bg-transparent");
      setFontColor("text-white");
      setButtonColor("text-white");
      setLogoSrc("/images/home_icon_w.png");
    } else if (currentScrollY > lastScrollY) {
      // Hide navbar on scroll down
      setIsVisible(false);
      setBgColor("bg-transparent");
    } else {
      // Show navbar on scroll up and change styles
      setIsVisible(true);
      setBgColor("bg-gray-200");
      setFontColor("text-black");
      setButtonColor("text-black");
      setLogoSrc("/images/home_icon.png");
    }

    // Update the last scroll position
    setLastScrollY(currentScrollY);
  };

  // Scroll event handler
  window.addEventListener("scroll", handleScroll);
  return () => {
    window.removeEventListener("scroll", handleScroll);
  };
```

```
  }, [lastScrollY]);


  // Toggle hamburger menu for mobile screens
  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  };


  return (
    <nav
      className={`fixed top-0 left-0 w-full z-50 ${bgColor} ${fontColor}
transition-transform duration-300 ${
        isVisible ? "translate-y-0" : "-translate-y-full"
      }`}
    >
      <div className="flex items-center justify-between px-4 py-2
lg:gap-x-6 lg:px-8">
        {/* Logo */}
        <div>
          <Link href="/" className="flex items-center animate-pulse">
          <Image
            src={logoSrc}
            alt="Logo"
            className="h-auto w-auto"
            width={24}
            height={24}
          />
          </Link>
        </div>
```

```
{/* Hamburger Icon for mobile screens */}
<div className="lg:hidden flex items-center">
  <button
    onClick={toggleMenu}
    className="text-white focus:outline-none"
  >
    {/* Hamburger icon */}
    <svg
      xmlns="http://www.w3.org/2000/svg"
      fill="none"
      viewBox="0 0 24 24"
      stroke="currentColor"
      className="h-6 w-6"
    >
      <path
        strokeLinecap="round"
        strokeLinejoin="round"
        strokeWidth="2"
        d="M4 6h16M4 12h16M4 18h16"
      />
    </svg>
  </button>
</div>

{/* Desktop links */}
<div className="hidden lg:flex lg:gap-x-6 mr-auto pl-24">
```

```
          <Link
            href="/visualisations"
            className="rounded-full px-4 py-3 text-sm font-semibold
hover:underline hover:-translate-y-1 hover:scale-60 duration-300"
          >
            Visualisations
          </Link>
          <Link
            href="/libraries"
            className="rounded-full px-4 py-3 text-sm font-semibold
hover:underline hover:-translate-y-1 hover:scale-60 duration-300"
          >
            Libraries
          </Link>
          <Link
            href="/documentation"
            className="rounded-full px-4 py-3 text-sm font-semibold
hover:underline hover:-translate-y-1 hover:scale-60 duration-300"
          >
            Documentation
          </Link>
          <Link
            href="/about"
            className="rounded-full px-4 py-3 text-sm font-semibold
hover:underline hover:-translate-y-1 hover:scale-60 duration-300"
          >
            About
          </Link>
```

```
        </div>


        {/* Desktop buttons */}
        <div className="hidden lg:flex lg:gap-x-4">
          <Link href="/login">
            <button
              className={`px-4 py-1 text-sm font-semibold border
border-white rounded hover:bg-white hover:text-gray-800 ${buttonColor}`}
            >
              Log in
            </button>
          </Link>
          <Link href="/register">
            <button
              className={`px-4 py-1 text-sm font-semibold rounded
hover:bg-gray-500 ${
                buttonColor === "text-white"
                  ? "bg-white text-black"
                  : "bg-gray-800 text-white"
              }`}
            >
              Sign up
            </button>
          </Link>
        </div>
      </div>


      {/* Mobile menu */}
```

```
      <div
        className={`lg:hidden ${isMenuOpen ? "block" : "hidden"}
bg-custom-gray text-white py-4 px-6`}
      >
        <div className="flex flex-col items-end">
        <Link href="/visualisations" className="block py-2 text-sm
hover:underline hover:text-gray-300 hover:focus:text-gray-300">
          Visualisations
        </Link>
        <Link href="/libraries" className="block py-2 text-sm
hover:underline hover:text-gray-300 hover:focus:text-gray-300">
          Libraries
        </Link>
        <Link href="/documentation" className="block py-2 text-sm
hover:underline hover:text-gray-300 hover:focus:text-gray-300">
          Documentation
        </Link>
        <Link href="/about" className="block py-2 text-sm hover:underline
hover:text-gray-300 hover:focus:text-gray-300">
          About
        </Link>
        <Link href="/login" className="block py-2 text-sm hover:underline
hover:text-gray-300 hover:focus:text-gray-300">
          Log in
        </Link>
        <Link href="/register" className="block py-2 text-sm
hover:underline hover:text-gray-300 hover:focus:text-gray-300">
          Sign up
        </Link>
```

```
      </div>
    </div>
  </nav>
  );
};

export default Navbar; // Export the Navbar component

// Import the components from the components directory
import { TimelineWithIcon } from "./TimelineCard";

// OverviewTab component
export function OverviewTab() {
  return (
    // Flexbox layout container
    <div className="flex flex-row items-center justify-center">
      <TimelineWithIcon />
    </div>
  );
}

// Import Necessary modules
import Image from 'next/image';
import {
  Card,
  CardHeader,
  CardBody,
```

```
  Typography,
} from "@material-tailwind/react";


// HorizontalCard component
export function HorizontalCard() {
  return (
    <Card className="w-full max-w-[48rem] flex-col sm:flex-row" {...({} as
React.ComponentProps<typeof Card>)}>


      {/* Card Body */}
      <CardBody className="sm:w-3/5" {...({} as
React.ComponentProps<typeof CardBody>)}>


        {/* Title section */}
        <Typography
          variant="h6"
          color="gray"
          className="mb-4 uppercase text-center sm:text-center" {...({} as
React.ComponentProps<typeof Typography>)}
        >
          DATA
        </Typography>


        {/* Heading for visualizations link */}
        <Typography
          variant="h4"
          color="blue-gray"
```

```
          className="mb-8 lg:mb-4 text-center sm:text-center" {...({} as
React.ComponentProps<typeof Typography>)}
        >
          Visualisations using data taken from Project Gutenberg
        </Typography>

        {/* Description */}
        <Typography
          color="gray"
          className="mb-4 font-normal text-justify sm:text-left" {...({}
as React.ComponentProps<typeof Typography>)}
        >
          The visualizations are powered by data retrieved from the
Gutendex API,
          which fetches book catalog information from the Project
Gutenberg website.
          This data is then stored in a PostgreSQL database, enabling
dynamic and
          interactive visualisations for exploring the book collection.
        </Typography>

        {/* Learn More link */}
        <div className="flex justify-center">
          <a
            href="/visualisations"
            className="px-5 py-2 text-sm 2xl:text-2xl font-semibold
leading-6 text-black hover:focus:text-gray-600 hover:text-gray-600
hover:underline 2xl:mt-20"
          >
```

```
            Learn more <span aria-hidden="true">→</span>
          </a>
        </div>
      </CardBody>

      {/* Image */}
      <CardHeader
        shadow={false}
        floated={false}
        className="ml-2 lg:mt-0 mt-[-2rem] w-full sm:w-2/5 shrink-0
rounded-t-none sm:rounded-l-none" {...({} as React.ComponentProps<typeof
CardHeader>)}
      >
        <Image
          src="/images/Bar-Chart-Vertical.png"
          alt="card-image"
          className="h-full w-full object-cover"
          width={500}
          height={500}
        />
      </CardHeader>
    </Card>
  );
}

// Render the component client-side
"use client";
```

```
// Import React hooks, necessary modules and icons
import { useState } from 'react';
import {
  Input,
  Button,
} from "@material-tailwind/react";
import { FaEye, FaEyeSlash } from "react-icons/fa";


const RegisterForm = () => {
  const [username, setUsername] = useState(""); // Stores the username entered by the user
  const [email, setEmail] = useState(''); // Stores the email entered by the user
  const [password, setPassword] = useState(''); // Stores the password entered by the user
  const [confirmPassword, setConfirmPassword] = useState(''); // Stores the confirmed password
  const [showPassword, setShowPassword] = useState(false); // Toggles password visibility
  const [message, setMessage] = useState(''); // Stores the success or error message


  // Handle form submission
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault(); // Prevent default form submission
    setMessage(''); // Clear previous messages


    // Check if password and confirm password match
    if (password !== confirmPassword) {
```

```
      setMessage('Error: Passwords do not match.');
      return; // If passwords don't match stop form submission
    }

    try {
      // Send registration data to the API which updates the tables in
Postgres database
      const response = await fetch('/api/register', {
        method: 'POST', // Sending POST request to the register API
        headers: {
          'Content-Type': 'application/json', // Setting request body type
        },
        body: JSON.stringify({ username, email, password }), // Convert
username, email, and password to a JSON string
      });

      // Check if the registration was successful
      if (response.ok) {
        setMessage('Registration successful!'); // Success message
        setUsername(''); // Clear username field
        setEmail(''); // Clear email field
        setPassword(''); // Clear password field
        setConfirmPassword(''); // Clear confirm password field
      } else {
        const errorData = await response.json();
        setMessage(`Error: ${errorData.error}`); // Error message
      }
    } catch {
```

```
      setMessage('Error: Unable to register.'); // Error message
    }
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-6 flex flex-col
items-center">

      {/* Username Input */}
      <div className="w-3/6 pt-2 pb-2">
        <Input
          variant="standard"
          color="white"
          type="text"
          id="username"
          value={username}
          onChange={(e) => setUsername(e.target.value)} // Update Username
on change
          required
          label="Username" {...({} as React.ComponentProps<typeof Input>)}
        />
      </div>

      {/* Email Input */}
      <div className="w-3/6 pt-2 pb-2">
        <Input
          variant="standard"
          color="white"
```

```
          type="email"
          id="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)} // Update email on
change
          required
          label="Email Address" {...({} as React.ComponentProps<typeof
Input>)}
        />
      </div>

      {/* Password Input */}
      <div className="w-3/6 pt-2 pb-2 relative">
        <Input
          variant="standard"
          color="white"
          type={showPassword ? "text" : "password"} // Toggle password
visibility
          id="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)} // Update password
on change
          required
          label="Password" {...({} as React.ComponentProps<typeof Input>)}
        />

        {/* Eye Icon */}
        <button
```

```
          type="button"
          onClick={() => setShowPassword(!showPassword)} // Toggle
password visibility
          className="absolute top-1/2 right-[-3rem] lg:right-[-2rem]
transform -translate-y-1/2 text-gray-500"
          aria-label="Toggle password visibility"
        >
          {showPassword ? <FaEyeSlash /> : <FaEye />}
        </button>
      </div>

      {/* Confirm Password Input */}
      <div className="w-3/6 pt-2 pb-2 relative">
        <Input
          variant="standard"
          color="white"
          type={showPassword ? "text" : "password"} // Toggle password
visibility
          id="confirmPassword"
          value={confirmPassword}
          onChange={(e) => setConfirmPassword(e.target.value)} // Update
confirm password on change
          required
          label="Confirm Password" {...({} as React.ComponentProps<typeof
Input>)}
        />

        {/* Eye Icon */}
        <button
```

```
          type="button"
          onClick={() => setShowPassword(!showPassword)} // Toggle
password visibility
          className="absolute top-1/2 right-[-3rem] lg:right-[-2rem]
transform -translate-y-1/2 text-gray-500"
          aria-label="Toggle password visibility"
        >
          {showPassword ? <FaEyeSlash /> : <FaEye />}
        </button>
      </div>

      {/* Already Have Account Link */}
      <a
        href="/login"
        className="px-5 py-2 text-xs 2xl:text-2xl font-semibold leading-6
text-white hover:focus:text-gray-300 hover:text-gray-300 hover:underline
2xl:mt-20"
      >
        Already have an account? <span aria-hidden="true">→</span>
      </a>

      {/* Submit Button */}
      <Button variant="outlined" color="white" type="submit" {...({} as
React.ComponentProps<typeof Button>)}>
        Register
      </Button>

      {/* Display error or success message */}
      {message && <p className="mt-2 text-sm text-red-500">{message}</p>}
```

```
    </form>
  );
};

export default RegisterForm; // Export the RegisterForm component

// Import necessary modules and Icons
import {
  Timeline,
  TimelineItem,
  TimelineConnector,
  TimelineHeader,
  TimelineIcon,
  TimelineBody,
  Typography,
} from "@material-tailwind/react";
import { HomeIcon, BookOpenIcon, PhotoIcon } from
"@heroicons/react/24/solid";

// TimelineWithIcon component
export function TimelineWithIcon() {
  return (
    <div className="w-full max-w-[32rem] p-4 sm:px-8">
      <Timeline>
        <TimelineItem>
          <TimelineConnector />
          <TimelineHeader>
            <TimelineIcon className="p-2">
```

```
            <HomeIcon className="h-4 w-4" />
          </TimelineIcon>
          <Typography variant="h5" color="blue-gray" className="text-lg
lg:text-xl" {...({} as React.ComponentProps<typeof Typography>)}>
            Welcome to Your Dashboard!
          </Typography>
        </TimelineHeader>
        <TimelineBody className="pb-8 text-left">
          <Typography color="gray" className="text-sm lg:text-base
font-normal text-gray-600" {...({} as React.ComponentProps<typeof
Typography>)}>
            Welcome to ShelfSpace! Thank you for signing up and joining
our community. We&#39;re thrilled to have you on board! Your dashboard is
your central hub—explore features, manage your account, and get the most
out of your experience. Let’s get started on achieving your goals
together!
          </Typography>
        </TimelineBody>
      </TimelineItem>
      <TimelineItem>
        <TimelineConnector />
        <TimelineHeader>
          <TimelineIcon className="p-2">
            <BookOpenIcon className="h-4 w-4" />
          </TimelineIcon>
          <Typography variant="h5" color="blue-gray" className="text-lg
lg:text-xl" {...({} as React.ComponentProps<typeof Typography>)}>
            Explore Our eBook Collection!
          </Typography>
        </TimelineHeader>
```

```
          <TimelineBody className="pb-8">
            <Typography color="gray" className="text-sm lg:text-base
font-normal text-gray-600" {...({} as React.ComponentProps<typeof
Typography>)}>
              Explore our extensive collection of eBooks, carefully
curated for your reading pleasure! With the eBook reader, you can dive
into new stories, learn something new, or relax with your favorite
genres—all at your fingertips. Start exploring today and enjoy a world of
knowledge and entertainment!
            </Typography>
          </TimelineBody>
        </TimelineItem>
        <TimelineItem>
          <TimelineHeader>
            <TimelineIcon className="p-2">
              <PhotoIcon className="h-4 w-4" />
            </TimelineIcon>
            <Typography variant="h5" color="blue-gray" className="text-lg
lg:text-xl" {...({} as React.ComponentProps<typeof Typography>)}>
              Customize Your Profile!
            </Typography>
          </TimelineHeader>
          <TimelineBody>
            <Typography color="gray" className="text-sm lg:text-base
font-normal text-gray-600" {...({} as React.ComponentProps<typeof
Typography>)}>
              Check out your profile tab to personalise your account! You
can easily edit your avatar to reflect your unique style. Make your
profile truly yours and keep it fresh with a personalized touch. Visit
your profile now and make it stand out!
            </Typography>
```

```
          </TimelineBody>
        </TimelineItem>
      </Timeline>
    </div>
  );
}


// Import React hooks, components and necessary modules
import Image from 'next/image';
import React, { useState, useEffect } from "react";
import {
  Drawer,
  Tabs,
  TabsHeader,
  Tab,
  Button,
  Select,
  Option
} from "@material-tailwind/react";
import { Square3Stack3DIcon, BookOpenIcon, UserCircleIcon, Cog6ToothIcon }
from "@heroicons/react/24/solid";
import EbookReader from "./EbookReader";
import { UserProfileCard } from "./UserProfileCard";
import LogoutButton from "./Logout";
import { OverviewTab } from "./OverviewTab";


// Define an interface for the Ebook object
interface Ebook {
```

```
  id: number;
  title: string;
  author: string;
  description: string;
  publishedAt: string;
  fileUrl: string;
  coverPageUrl: string;
}


// Main UserDashboardTabs component
export function UserDashboardTabs() {
  const [open, setOpen] = useState(false); // State for drawer visibility
  const [selectedTab, setSelectedTab] = useState("dashboard"); // State to
track selected tab
  const [ebookId, setEbookId] = useState<number | null>(null); // Track
the selected ebook ID
  const [ebooks, setEbooks] = useState<Ebook[]>([]); // State to store
list of ebooks
  const [loading, setLoading] = useState<boolean>(false); // Loading state
for fetching ebooks
  const [error] = useState<string | null>(null); // State to track errors


  // Function to open the drawer
  const openDrawer = () => setOpen(true);


  // Function to close the drawer
  const closeDrawer = () => setOpen(false);
```

```
  // Data for each tab in the sidebar, including labels, values, and icons
  const data = [
    { label: "Dashboard", value: "dashboard", icon: Square3Stack3DIcon },
    { label: "ebook Reader", value: "ebookreader", icon: BookOpenIcon },
    { label: "Profile", value: "profile", icon: UserCircleIcon },
    { label: "Settings", value: "settings", icon: Cog6ToothIcon },
  ];

  // Fetch ebooks on component load
  useEffect(() => {
    const fetchEbooks = async () => {
      setLoading(true); // Set loading to true while fetching
      const response = await fetch("/api/fetch");
      const data = await response.json();
      setEbooks(data); // Store fetched ebooks in state
      setLoading(false); // Set loading to false once fetching is complete
    };

    fetchEbooks();
  }, []); // Empty dependency array ensures it runs only once when the
component loads

  // Function to render the relevant content based on selected tab
  const renderContent = () => {
    switch (selectedTab) {
      case "dashboard":
        return <OverviewTab />;
      case "ebookreader":
```

```
        return (
          <div className="flex w-full h-full flex-col gap-6 py-4 px-2">
            <h2 className="text-lg lg:text-2xl font-bold text-black
underline mb-4 ml-4">Choose a ebook:</h2>

            {/* Loading or error states */}
            {loading && <div className="text-center text-gray-600">Loading
ebooks...</div>}
            {error && <div className="text-center
text-red-500">{error}</div>}

            {/* Ebook selection dropdown */}
            <Select
              size="lg"
              label="Select Ebook"
              value={ebookId?.toString() || ""}
              onChange={(value) => {
                // Convert the selected value to a number and update the
ebookId state
                const selectedId = Number(value);
                setEbookId(selectedId);
              }}
              className="mb-6" {...({} as React.ComponentProps<typeof
Select>)}
            >
              {/* Iterate over the ebooks array and create an option for
each ebook */}
              {ebooks.map((ebook) => (
                <Option key={ebook.id} value={ebook.id.toString()}>
```

```
                  {ebook.title}
                </Option>
              ))}
            </Select>

            {/* If an ebook is selected, display the book details and
cover page */}
            {ebookId && (
              <div className="text-center">
                <h3 className="text-x1 lg:text-3x1 font-semibold overline
mt-4">{ebooks.find((ebook) => ebook.id === ebookId)?.title}</h3>
                <p className="text-sm lg:text-base text-black underline
py-2 mt-4">Author: {ebooks.find((ebook) => ebook.id ===
ebookId)?.author}</p>
                <p className="text-sm lg:text-base text-justify
text-gray-600 font-medium mt-4 px-10 lg:px-60">Description:
{ebooks.find((ebook) => ebook.id === ebookId)?.description}</p>

                {/* Show cover image only when an ebook is selected */}
                {ebooks.find((ebook) => ebook.id ===
ebookId)?.coverPageUrl && (
                <Image
                src={ebooks.find((ebook) => ebook.id ===
ebookId)?.coverPageUrl || '/default-image.jpg'}
                alt={ebooks.find((ebook) => ebook.id === ebookId)?.title
|| 'Default Title'}
                className="mt-12 h-80 w-72 lg:h-96 lg:w-80 mx-auto
hover:-translate-y-1 hover:scale-110 duration-300"
                width={288}
                 height={320}
```

```
                />
                )}

                {/* Display the EbookReader component */}
                <div className="overflow-hidden mt-4">
                  <EbookReader ebookId={ebookId} />
                </div>
              </div>
            )}
          </div>
        );
      case "profile":
        return <UserProfileCard />;
      case "settings":
        return <LogoutButton />;
      default:
        return null; // Return nothing if no tab is selected
    }
  };

  // Function to handle tab change
  const handleTabChange = (value: string) => {
    setSelectedTab(value);
    closeDrawer(); // Close the drawer after selecting a tab
  };

  return (
```

```
    <>
      {/* Button to open the drawer */}
      <div className="flex items-center justify-center">
        <Button className="text-sm font-semibold" color="red"
onClick={openDrawer} {...({} as React.ComponentProps<typeof Button>)}>
          Open Menu
        </Button>
      </div>

      {/* Drawer component to show tabs */}
      <Drawer {...({} as React.ComponentProps<typeof Drawer>)} open={open}
onClose={closeDrawer} className="p-4 w-64">
        <Tabs value={selectedTab} orientation="vertical">
          <TabsHeader {...({} as React.ComponentProps<typeof
TabsHeader>)}>
            {data.map(({ label, value, icon }) => (
              <Tab {...({} as React.ComponentProps<typeof Tab>)}
key={value} value={value} onClick={() => handleTabChange(value)}>
                <div className="flex items-center gap-6 m-2">
                  {React.createElement(icon, { className: "w-5 h-5" })}
{/* Render the tab icon */}
                  {label} {/* Render the tab label */}
                </div>
              </Tab>
            ))}
          </TabsHeader>
        </Tabs>
      </Drawer>
```

```
      {/* Main content area */}
      <div className="rounded-2xl lg:p-8 bg-gray-50 flex flex-col
min-h-screen">
        {renderContent()} {/* Render content based on selected tab */}
      </div>
    </>
  );
}

// Import React hooks, token and necessary modules
import Image from 'next/image';
import { useEffect, useState } from 'react'; // Import useState and
useEffect hooks
import jwt from 'jsonwebtoken'; // Import the jwt library to decode the
token
import {
  Card,
  CardHeader,
  CardBody,
  CardFooter,
  Typography,
  Button,
} from "@material-tailwind/react";

// Define an interface for the token object
interface DecodedToken {
  userId: string;
  username: string;
```

```
  email: string;
}


export function UserProfileCard() {
  const [username, setUsername] = useState(''); // State to store the
username
  const [email, setEmail] = useState(''); // State to store the email
  const [selectedImage, setSelectedImage] = useState(''); // State for
selected profile image, initially empty value
  const [showPopup, setShowPopup] = useState(false); // State to toggle
Popup visibility
  const [errorMessage, setErrorMessage] = useState(''); // State for error
message
  const [successMessage, setSuccessMessage] = useState(''); // State for
success message


  // Image pool
  const imagePool = [
    '../images/avatar-man.svg',
    '../images/avatar-woman-5.svg',
    '../images/avatar-man-1.svg',
    '../images/avatar-woman-6.svg',
    '../images/avatar-man-2.svg',
    '../images/avatar-woman-7.svg',
    '../images/avatar-man-3.svg',
    '../images/avatar-woman-8.svg',
    '../images/avatar-man-4.svg',
    '../images/avatar-woman.svg',
    '../images/avatar-woman-1.svg',
```

```
    '../images/avatar-woman-2.svg',
    '../images/avatar-woman-3.svg',
    '../images/avatar-woman-4.svg',
  ];

  useEffect(() => {
    const token = localStorage.getItem("token"); // Retrieve the token
from localStorage

    if (token) {
      try {
        const decodedToken = jwt.decode(token) as DecodedToken; // Decode
the token with the declared interface

        if (decodedToken && decodedToken.userId) {
          setUsername(decodedToken.username || 'Default Username');
          setEmail(decodedToken.email || 'Default Email');
          // Fetch the profile picture from the API
          fetch(`/api/getUserProfile?userId=${decodedToken.userId}`)
            .then((response) => response.json()) // Parse the JSON
response
            .then((data) => {
              // Set the user's profile picture or a default image
              if (data.user) {
                setSelectedImage(data.user.profilePicture ||
'../images/avatar-man-1.svg');
              }
            });
```

```
        }
      } catch {
        setErrorMessage('Error decoding the token');
      }
    }
  }, []); // Runs once on component load


  // Handle image selection
  const handleImageSelect = async (image: string) => {
    setSelectedImage(image); // Update selected image
    setShowPopup(false); // Close the Popup


    // Get the token from localStorage
    const token = localStorage.getItem("token");
    if (token) {
      try {
        // Sending a POST request to update the profile picture in the
database
        const response = await fetch('/api/updateProfile', {
          method: 'POST', // Sending POST request to the updateProfile API
          headers: {
            'Content-Type': 'application/json', // Setting request body
type
          },
          body: JSON.stringify({ profilePicture: image, token }), // Send
selected image and token
        });
```

```
        const data = await response.json();
        if (response.ok) {
          setSelectedImage(data.user.profilePicture); // Set the updated
profile picture
          setSuccessMessage(data.message || 'Profile picture updated
successfully!'); // Set success message
          setTimeout(() => setSuccessMessage(''), 3000); // Clear message
after 3 seconds
        } else {
          setErrorMessage(data.error || 'Failed to update profile
picture');
        }
      } catch (error) {
        console.error('Error during fetch:', error);
        setErrorMessage('Error updating profile picture');
      }
    }
  };

  return (
    <div className="flex items-center justify-center min-h-screen p-2">
      <Card className="min-h-screen w-full sm:w-4/5 md:w-2/5" {...({} as
React.ComponentProps<typeof Card>)}>
        <CardHeader floated={false} className="h-60 lg:h-80" {...({} as
React.ComponentProps<typeof CardHeader>)}>
          {/* If selectedImage is not set, fallback to default */}
          <Image
            src={selectedImage || '/images/avatar-man-1.svg'}
            alt="profile-picture"
```

```
            width={500}
            height={300}
            className="rounded-full"
          />
        </CardHeader>
        <CardBody className="text-center" {...({} as
React.ComponentProps<typeof CardBody>)}>
          <Typography variant="h4" color="blue-gray" className="underline
my-4 lg:my-2 text-x1 lg:text-3xl" {...({} as React.ComponentProps<typeof
Typography>)}>
            {username}
          </Typography>
          <Typography color="blue-gray" className="mt-12 text-lg
lg:text-2xl" {...({} as React.ComponentProps<typeof Typography>)}>
            {email}
          </Typography>
        </CardBody>
        <CardFooter className="flex justify-center gap-7 pt-2" {...({} as
React.ComponentProps<typeof CardFooter>)}>
          <Button
            onClick={() => setShowPopup(true)} // Open the Popup
            className="text-xs mt-20" {...({} as
React.ComponentProps<typeof Button>)}>
            Change Profile Picture
          </Button>
        </CardFooter>
      </Card>

      {/* Error message */}
```

```
      {errorMessage && (
        <div className="text-center text-red-500 mt-4">
          <Typography {...({} as React.ComponentProps<typeof
Typography>)}>{errorMessage}</Typography>
        </div>
      )}

      {/* Success message */}
      {successMessage && (
        <div className="lg:mt-2">
          <p className="absolute bottom-20 lg:bottom-[-5rem] left-1/2
transform -translate-x-1/2 text-sm text-green-500">
            {successMessage}
          </p>
        </div>
      )}

      {/* Popup for image selection */}
      {showPopup && (
        <div className="fixed inset-0 pb-2 flex items-center
justify-center lg:items-center lg:justify-center bg-black bg-opacity-50">
          <div className="bg-white p-2 sm:p-6 rounded-lg shadow-lg w-9/12
sm:w-3/4 md:w-1/2 lg:w-2/3 max-h-[90vh] overflow-y-auto">
            <Typography variant="h6" className="mb-4 text-center
underline" {...({} as React.ComponentProps<typeof Typography>)}>
              Choose a Profile Picture
            </Typography>
            <div className="flex flex-wrap justify-center gap-4">
              {imagePool.map((image, index) => (
```

```
                <Image
                  key={index}
                  src={image}
                  alt={`avatar ${index + 1}`}
                  className="w-16 h-16 lg:w-24 lg:h-24 rounded-full
cursor-pointer border-2 border-transparent hover:border-blue-500"
                  width={100}
                  height={100}
                  onClick={() => handleImageSelect(image)} // Update image
on click
                 />
              ))}
            </div>
            <div className="mt-8 flex justify-center">
              <Button onClick={() => setShowPopup(false)} color="red"
{...({} as React.ComponentProps<typeof Button>)}>
                Close
              </Button>
            </div>
          </div>
        </div>
      )}
    </div>
  );
}

// Render the component client-side
"use client";
```

```
// Import React hooks, necessary modules and plugins for animations
import React, { useEffect } from "react";
import { gsap } from "gsap";
import { ScrollTrigger } from "gsap/dist/ScrollTrigger";
import {
  Card,
  CardHeader,
  CardBody,
  Typography,
  Avatar,
} from "@material-tailwind/react";

// StarIcon component
function StarIcon() {
  return (
    <svg
      xmlns="http://www.w3.org/2000/svg"
      viewBox="0 0 24 24"
      fill="currentColor"
      className="h-5 w-5 text-yellow-700"
    >
      <path
        fillRule="evenodd"
        d="M10.788 3.21c.448-1.077 1.976-1.077 2.424 0l2.082 5.007
5.404.433c1.164.093 1.636 1.545.749 2.305l-4.117 3.527 1.257 5.273c.271
1.136-.964 2.033-1.96 1.425L12 18.354 7.373
```

```
21.18c-.996.608-2.231-.29-1.96-1.425l1.257-5.273-4.117-3.527c-.887-.76-.41
5-2.212.749-2.305l5.404-.433 2.082-5.006z"
        clipRule="evenodd"
      />
    </svg>
  );
}

// Define an interface for the TestimonialCardProperties object
interface TestimonialCardProperties {
  name: string;
  title: string;
  imageUrl: string;
  rating: number; // Rating score
  testimonial: string;
}

// TestimonialCard component
export function TestimonialCard({
  name,
  title,
  imageUrl,
  rating,
  testimonial,
}: TestimonialCardProperties) {
  useEffect(() => {
    // Register the ScrollTrigger plugin for scroll-based animations
    gsap.registerPlugin(ScrollTrigger);
```

```
    // Convert all elements with the class "testimonial-card" into an
array using GSAP's toArray method
    const testimonialCards = gsap.utils.toArray(".testimonial-card");

    testimonialCards.forEach((card) => {
      // Animate each card when it enters the viewport
      gsap.fromTo(
        // @ts-expect-error raising unnecessary error
        card,
        {
          opacity: 0, // Initial state
          y: 50, // Initial vertical position
        },
        {
          opacity: 1, // Final state
          y: 0, // Reset vertical position
          duration: 0.5, // Duration of the animation
          ease: "power1.out", // Easing function for smoothness
          scrollTrigger: {
            trigger: card, // Animation trigger
            start: "top 80%", // Trigger when 80% of the card is visible
            once: true, // Trigger only once
          },
        }
      );
    });
```

```
  }, []); // Empty dependency array ensures this effect runs only once on
load


  // Dynamically generate star icons based on the rating
  const stars = Array.from({ length: 5 }, (_, index) => (
    <StarIcon key={index} /> // Return a new star icon for each iteration
  ));


  return (
    <Card
      color="transparent"
      shadow={false}
      className="testimonial-card w-full max-w-[26rem] mx-20 my-4" {...({}
as React.ComponentProps<typeof Card>)}
    >
      <CardHeader
        color="transparent"
        floated={false}
        shadow={false}
        className="mx-0 flex items-center gap-4 pt-0 pb-8" {...({} as
React.ComponentProps<typeof CardHeader>)}
      >
        {/* Avatar and Name Section */}
        <Avatar
          size="lg"
          variant="circular"
          src={imageUrl}
          alt={name} {...({} as React.ComponentProps<typeof Avatar>)}
```

```
        />
        <div className="flex w-full flex-col gap-0.5">
          <div className="flex items-center justify-between">
            <Typography variant="h5" color="blue-gray" {...({} as
React.ComponentProps<typeof Typography>)}>
              {name} {/* Display the name of the person */}
            </Typography>
            <div className="flex items-center gap-0">
              {stars.slice(0, rating)} {/* Display the number of stars
based on rating */}
            </div>
          </div>
          <Typography color="blue-gray" {...({} as
React.ComponentProps<typeof Typography>)}>{title}</Typography> {/* Display
the title */}
        </div>
      </CardHeader>
      <CardBody className="mb-6 p-0" {...({} as
React.ComponentProps<typeof CardBody>)}>
        <Typography {...({} as React.ComponentProps<typeof Typography>)}>
          &quot;{testimonial}&quot; {/* Display the testimonial message
*/}
        </Typography>
      </CardBody>
    </Card>
  );
}
```

```
// Import the PrismaClient from the Prisma client library to interact with the database
import { PrismaClient } from '@prisma/client';

// Instantiate a new PrismaClient instance, to make queries to the database
const prisma = new PrismaClient();

// Export the PrismaClient instance
export { prisma };

// Import the PrismaClient from the Prisma client library to interact with the database
import { PrismaClient } from '@prisma/client';

// Instantiate a new PrismaClient instance to interact with the database
const prisma = new PrismaClient();

// Export the Prisma instance
export default prisma;

// Import necessay modules for interactive input and for hashing the password
import readline from 'readline';
import bcrypt from 'bcryptjs';

// Create an interface to read input from the terminal
const rl = readline.createInterface({
```

```
  input: process.stdin,
  output: process.stdout
});

// Prompt the user for the password
rl.question('Enter the password to hash: ', (password) => {
  if (!password) {
    // If no password is entered, show an error message and close the
interface
    console.log('Please provide a password to hash.');
    rl.close();
    return;
  }

  // Hash the password using bcrypt
  const hashedPassword = bcrypt.hashSync(password, 10);

  // Output the hashed password
  console.log('Hashed Password:', hashedPassword);

  // Close the readline interface
  rl.close();
});

// Import the Prisma client to interact with the database
import prisma from '../lib/prisma';
```

```
// Function to reset the 'Ebook' sequence in the PostgreSQL database, to
avoid autoincrement id error related to adding and removing ebook entries
to the database
export async function resetEbookSequence() {
  try {
    // Retrieve the maximum 'id' value from the 'Ebook' table
    const maxId = await prisma.ebook.aggregate({
      _max: {
        id: true, // Get the highest 'id' value from the 'Ebook' table
      },
    });

    // Reset the 'Ebook' sequence to the max 'id' or 1 if none exist.
    await prisma.$executeRawUnsafe(`
      SELECT setval('public."Ebook_id_seq"', ${maxId._max.id || 1},
false);
    `);
  } catch  {
    // If there is an error throw the error
    throw new Error("Failed to reset sequence.");
  }
}
```

Schema:

```
generator client {
  provider = "prisma-client-js"
}
```

```
datasource db {
  provider  = "postgresql"
  url       = env("DATABASE_URL")
  directUrl = env("DIRECT_URL")
}

model Ebook {
  id           Int      @id @default(autoincrement())
  title        String
  author       String
  description  String
  publishedAt  DateTime
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  fileUrl      String
  coverPageUrl String
}

model User {
  id             Int      @id @default(autoincrement())
  email          String   @unique
  password       String
  isAdmin        Boolean  @default(false)
  updatedAt      DateTime @updatedAt
  createdAt      DateTime @default(now())
```

```
  username       String   @unique
  profilePicture String?
}

model auth_group {
  id                     Int                     @id
@default(autoincrement())
  name                   String                  @unique
@db.VarChar(150)
  auth_group_permissions auth_group_permissions[]
  auth_user_groups       auth_user_groups[]

  @@index([name], map: "auth_group_name_a6ea08ec_like")
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model auth_group_permissions {
  id              Int             @id
@default(autoincrement())
  group_id        Int
  permission_id   Int
  auth_permission auth_permission @relation(fields:
[permission_id], references: [id], onDelete: NoAction,
onUpdate: NoAction, map:
"auth_group_permissio_permission_id_84c5c92e_fk_auth_perm")
```

```
  auth_group      auth_group      @relation(fields:
[group_id], references: [id], onDelete: NoAction, onUpdate:
NoAction, map:
"auth_group_permissions_group_id_b120cbf9_fk_auth_group_id")


  @@unique([group_id, permission_id], map:
"auth_group_permissions_group_id_permission_id_0cd325b0_uniq
")
  @@index([group_id], map:
"auth_group_permissions_group_id_b120cbf9")
  @@index([permission_id], map:
"auth_group_permissions_permission_id_84c5c92e")
}


/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model auth_permission {
  id                          Int
@id @default(autoincrement())
  name                        String
@db.VarChar(255)
  content_type_id             Int
  codename                    String
@db.VarChar(100)
  auth_group_permissions      auth_group_permissions[]
  django_content_type         django_content_type
@relation(fields: [content_type_id], references: [id],
```

```
onDelete: NoAction, onUpdate: NoAction, map:
"auth_permission_content_type_id_2f476e4b_fk_django_co")
  auth_user_user_permissions auth_user_user_permissions[]


  @@unique([content_type_id, codename], map:
"auth_permission_content_type_id_codename_01ab375a_uniq")
  @@index([content_type_id], map:
"auth_permission_content_type_id_2f476e4b")
}


model auth_user {
  id                          Int
@id @default(autoincrement())
  password                    String
@db.VarChar(128)
  last_login                  DateTime?
@db.Timestamptz(6)
  is_superuser                Boolean
  username                    String
@unique @db.VarChar(150)
  first_name                  String
@db.VarChar(150)
  last_name                   String
@db.VarChar(150)
  email                       String
@db.VarChar(254)
  is_staff                    Boolean
  is_active                   Boolean
```

```
  date_joined                DateTime
@db.Timestamptz(6)
  auth_user_groups           auth_user_groups[]
  auth_user_user_permissions auth_user_user_permissions[]
  django_admin_log           django_admin_log[]

  @@index([username], map:
"auth_user_username_6821ab7c_like")
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model auth_user_groups {
  id         Int        @id @default(autoincrement())
  user_id    Int
  group_id   Int
  auth_group auth_group @relation(fields: [group_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map: "auth_user_groups_group_id_97559544_fk_auth_group_id")
  auth_user  auth_user  @relation(fields: [user_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map: "auth_user_groups_user_id_6a12ed8b_fk_auth_user_id")

  @@unique([user_id, group_id], map:
"auth_user_groups_user_id_group_id_94350c0c_uniq")
  @@index([group_id], map:
"auth_user_groups_group_id_97559544")
```

```
  @@index([user_id], map:
"auth_user_groups_user_id_6a12ed8b")
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model auth_user_user_permissions {
  id              Int             @id
@default(autoincrement())
  user_id         Int
  permission_id   Int
  auth_permission auth_permission @relation(fields:
[permission_id], references: [id], onDelete: NoAction,
onUpdate: NoAction, map:
"auth_user_user_permi_permission_id_1fbb5f2c_fk_auth_perm")
  auth_user       auth_user       @relation(fields:
[user_id], references: [id], onDelete: NoAction, onUpdate:
NoAction, map:
"auth_user_user_permissions_user_id_a95ead1b_fk_auth_user_id
")

  @@unique([user_id, permission_id], map:
"auth_user_user_permissions_user_id_permission_id_14a6b632_u
niq")
  @@index([permission_id], map:
"auth_user_user_permissions_permission_id_1fbb5f2c")
  @@index([user_id], map:
"auth_user_user_permissions_user_id_a95ead1b")
```

```
}

/// This table contains check constraints and requires
additional setup for migrations. Visit
https://pris.ly/d/check-constraints for more info.
model books_book {
  id                     Int                     @id
@default(autoincrement())
  download_count         Int?
  gutenberg_id           Int                     @unique
  media_type             String
@db.VarChar(16)
  title                  String?
@db.VarChar(1024)
  copyright              Boolean?
  books_book_authors     books_book_authors[]
  books_book_bookshelves books_book_bookshelves[]
  books_book_languages   books_book_languages[]
  books_book_subjects    books_book_subjects[]
  books_book_translators books_book_translators[]
  books_format           books_format[]
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model books_book_authors {
```

```
  id           Int          @id @default(autoincrement())
  book_id      Int
  person_id    Int
  books_book   books_book   @relation(fields: [book_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map: "books_book_authors_book_id_ed3433e7_fk_books_book_id")
  books_person books_person @relation(fields: [person_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map:
"books_book_authors_person_id_feffc563_fk_books_person_id")

  @@unique([book_id, person_id], map:
"books_book_authors_book_id_author_id_8714badb_uniq")
  @@index([person_id], map:
"books_book_authors_author_id_984f1ab8")
  @@index([book_id], map:
"books_book_authors_book_id_ed3433e7")
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model books_book_bookshelves {
  id              Int             @id
@default(autoincrement())
  book_id         Int
  bookshelf_id    Int
  books_bookshelf books_bookshelf @relation(fields:
[bookshelf_id], references: [id], onDelete: NoAction,
```

```
onUpdate: NoAction, map:
"books_book_bookshelv_bookshelf_id_80cc77c5_fk_books_boo")
  books_book      books_book      @relation(fields:
[book_id], references: [id], onDelete: NoAction, onUpdate:
NoAction, map:
"books_book_bookshelves_book_id_f820ff72_fk_books_book_id")


  @@unique([book_id, bookshelf_id], map:
"books_book_bookshelves_book_id_bookshelf_id_6016a70a_uniq")
  @@index([book_id], map:
"books_book_bookshelves_book_id_f820ff72")
  @@index([bookshelf_id], map:
"books_book_bookshelves_bookshelf_id_80cc77c5")
}


/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model books_book_languages {
  id             Int           @id
@default(autoincrement())
  book_id        Int
  language_id    Int
  books_book     books_book     @relation(fields: [book_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map:
"books_book_languages_book_id_e833b1f4_fk_books_book_id")
  books_language books_language @relation(fields:
[language_id], references: [id], onDelete: NoAction,
onUpdate: NoAction, map:
```

```
"books_book_languages_language_id_e9f60572_fk_books_language
_id")


  @@unique([book_id, language_id], map:
"books_book_languages_book_id_language_id_554fdccb_uniq")
  @@index([book_id], map:
"books_book_languages_book_id_e833b1f4")
  @@index([language_id], map:
"books_book_languages_language_id_e9f60572")
}


/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model books_book_subjects {
  id            Int           @id @default(autoincrement())
  book_id       Int
  subject_id    Int
  books_book    books_book    @relation(fields: [book_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map:
"books_book_subjects_book_id_a578cff2_fk_books_book_id")
  books_subject books_subject @relation(fields:
[subject_id], references: [id], onDelete: NoAction,
onUpdate: NoAction, map:
"books_book_subjects_subject_id_7445958f_fk_books_subject_id
")
```

```
  @@unique([book_id, subject_id], map:
"books_book_subjects_book_id_subject_id_74dcf64a_uniq")
  @@index([book_id], map:
"books_book_subjects_book_id_a578cff2")
  @@index([subject_id], map:
"books_book_subjects_subject_id_7445958f")
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model books_book_translators {
  id           Int          @id @default(autoincrement())
  book_id      Int
  person_id    Int
  books_book   books_book   @relation(fields: [book_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map:
"books_book_translators_book_id_7b8cd893_fk_books_book_id")
  books_person books_person @relation(fields: [person_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map:
"books_book_translators_person_id_ad262373_fk_books_person_i
d")

  @@unique([book_id, person_id], map:
"books_book_translators_book_id_person_id_fd7c4b79_uniq")
  @@index([book_id], map:
"books_book_translators_book_id_7b8cd893")
```

```
  @@index([person_id], map:
"books_book_translators_person_id_ad262373")
}

model books_bookshelf {
  id                    Int                    @id
@default(autoincrement())
  name                  String                 @unique
@db.VarChar(64)
  books_book_bookshelves books_book_bookshelves[]

  @@index([name], map: "books_bookshelf_name_2642cad6_like")
}

/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model books_format {
  id         Int        @id @default(autoincrement())
  mime_type  String     @db.VarChar(32)
  url        String     @db.VarChar(256)
  book_id    Int
  books_book books_book @relation(fields: [book_id],
references: [id], onDelete: NoAction, onUpdate: NoAction,
map: "books_format_book_id_b948fa34_fk_books_book_id")

  @@index([book_id], map: "books_format_book_id_b948fa34")
```

```
}

model books_language {
  id                    Int                    @id
@default(autoincrement())
  code                  String                 @unique
@db.VarChar(4)
  books_book_languages books_book_languages[]

  @@index([code], map: "books_language_code_217c406c_like")
}

model books_person {
  id                     Int                      @id(map:
"books_author_pkey") @default(autoincrement())
  birth_year             Int?
@db.SmallInt
  death_year             Int?
@db.SmallInt
  name                   String
@db.VarChar(128)
  books_book_authors     books_book_authors[]
  books_book_translators books_book_translators[]
}

model books_subject {
```

```
  id                    Int                    @id
@default(autoincrement())
  name                  String                 @db.VarChar(256)
  books_book_subjects books_book_subjects[]
}

/// This table contains check constraints and requires
additional setup for migrations. Visit
https://pris.ly/d/check-constraints for more info.
/// This model has constraints using non-default deferring
rules and requires additional setup for migrations. Visit
https://pris.ly/d/constraint-deferring for more info.
model django_admin_log {
  id                    Int                    @id
@default(autoincrement())
  action_time           DateTime
@db.Timestamptz(6)
  object_id             String?
  object_repr           String                 @db.VarChar(200)
  action_flag           Int                    @db.SmallInt
  change_message        String
  content_type_id       Int?
  user_id               Int
  django_content_type django_content_type? @relation(fields:
[content_type_id], references: [id], onDelete: NoAction,
onUpdate: NoAction, map:
"django_admin_log_content_type_id_c4bce8eb_fk_django_co")
```

```
  auth_user           auth_user           @relation(fields:
[user_id], references: [id], onDelete: NoAction, onUpdate:
NoAction, map:
"django_admin_log_user_id_c564eba6_fk_auth_user_id")

  @@index([content_type_id], map:
"django_admin_log_content_type_id_c4bce8eb")
  @@index([user_id], map:
"django_admin_log_user_id_c564eba6")
}

model django_content_type {
  id               Int                @id
@default(autoincrement())
  app_label        String             @db.VarChar(100)
  model            String             @db.VarChar(100)
  auth_permission  auth_permission[]
  django_admin_log django_admin_log[]

  @@unique([app_label, model], map:
"django_content_type_app_label_model_76bd3d3b_uniq")
}

model django_migrations {
  id      Int      @id @default(autoincrement())
  app     String   @db.VarChar(255)
  name    String   @db.VarChar(255)
```

```
  applied DateTime @db.Timestamptz(6)
}

model django_session {
  session_key  String   @id @db.VarChar(40)
  session_data String
  expire_date  DateTime @db.Timestamptz(6)

  @@index([expire_date], map:
"django_session_expire_date_a5c62663")
  @@index([session_key], map:
"django_session_session_key_c0390e0f_like")
}
```