BBC Scrapper Venue Kings Design Choices and Time Spent on Each Section

1. Website Chosen

- **Chosen Website**: BBC News
- **Reasoning**:
    - I chose **BBC News** because it offers a variety of news categories (e.g., Technology, Business, Sports, General) with structured content, making it ideal for web scraping. The content is diverse and covers multiple fields, which is important for this task.
    - The website is well-organized and offers clear categories, which facilitates efficient data extraction.
    - The project can be extended easily to scrape different sections like BBC Sport, which adds value and variety to the data.
- **Time Spent**: 1 hour to explore the website, decide on the categories to scrape, and identify the key fields needed.

2. Web Scraping (Using Selenium)

- **Reasoning for Tool Choice**:
    - Selenium can handle dynamic content, which was necessary for scraping BBC News, as the site loads content via JavaScript.
    - **Headless Mode** was chosen for efficiency as it runs the browser in the background without rendering the graphical interface, which speeds up scraping.
- **Design Choices**:
    - There are multiple categories as well as BBC Sports section which has different HTML structure so I had to write separate methods for that.

    - Implemented error handling to manage timeouts, missing elements, and unexpected errors.
    - Scraped key fields: headline, category, description, last_updated, tag and URL of the article.
- **Time Spent**:
    - 2 hours for configuring Selenium, scraping data, and testing.
    - 2 hours for debugging dynamic content scraping and handling pagination.

3. Database Design (SQLite)

- **Reasoning for Tool Choice**:
    - I chose **SQLite** because it is lightweight, easy to set up, and requires minimal configuration, making it perfect for small projects. It also avoids the complexity of setting up a full database system for this task.
    - **SQLAlchemy ORM** was used for ease of database interaction, enabling me to manage data in an object-oriented way and ensuring a more efficient data query process.
- **Design Choices**:
    - **Schema**: The database schema includes a table for storing scraped news items. Each entry has fields like `headline`, `category`, `subcategory`, `description, image_url, tag`, and `last_updated`.
    - **Data Integrity**: Ensured that no duplicate entries were added, and each article has a unique identifier (e.g., article URL).
    - **Pagination**: Handled efficiently by retrieving a set number of articles per page.
- **Time Spent**:
    - 1 hour for setting up the SQLite database and structuring the tables.
    - 1 hour for designing the schema and ensuring proper data integrity and handling pagination.

4. Backend API (Flask REST API)

- **Reasoning for Tool Choice**:
    - I chose **Flask** because it is lightweight and simple, ideal for a small web scraping API. It allows rapid development of RESTful endpoints with minimal setup.
    - Flask provides a clear route structure and is easy to integrate with **SQLAlchemy**, which I am using for database interactions.
- **Design Choices**:
    - Created the following API routes:
        - `GET /scrape-news`: Initiates the scraping process.
        - `GET /data`: Retrieves all scraped data with pagination and filtering by category/subcategory.
        - `GET /data/<category>`: Retrieves data filtered by category.
        - `GET /data/<category>/<sub_category>`: Retrieves data filtered by category and subcategory.
        - `DELETE /delete/<id>`: Allows deletion of a specific article.
    - Ensured proper pagination support and filtering options in the responses for better scalability and usability.
- **Time Spent**:
    - 2 hours to design and implement the Flask API, including routes and response formatting.
    - 1 hour for testing and debugging the API, including pagination and filtering.

5. Frontend UI (React.js)

- **Reasoning for Tool Choice**:
    - **React.js** was chosen because of its component-based architecture, which makes it easy to build and maintain the UI.
    - React's virtual DOM ensures efficient re-rendering of components, which is important for dynamic data display.

- **Design Choices**:
    - Implemented dropdown menus for category and subcategory selection.
    - Added pagination controls to allow users to navigate through large sets of articles.
    - Displayed news articles dynamically based on the selected filters.

- **Time Spent**:
    - 3 hours to develop the UI components, set up the React app, and integrate API calls.
    - 2 hours for UI testing and debugging, including responsive design adjustments and filter functionality.

6. AI Tool Usage

- **AI Assistance**:
    - **Web Scraping**: I used AI to help identify the best class names and CSS selectors for scraping news data from the BBC website. Some prompts used include:
        - "Can you help me find the correct CSS selectors for the BBC News I will provide a news card?"
        - "What are the best strategies for web scraping dynamic content with Selenium?"
    - **ChromeDriver Debugging**: AI was used to troubleshoot issues with ChromeDriver not working correctly. Sample prompts used include:
        - "How do I resolve issues with ChromeDriver not launching in headless mode?"
        - "Which version will work for Chrome should I update it to the latest?"
    - **Documentation**: AI tools were used to refine the README.md and ensure all details, such as installation, usage, and features, were clear. Some example prompts include:

        - "Can you generate a detailed README.md for a web scraping project?"
        - "How should I document the installation process for a Flask app?"
    - **Debugging News Fetching**: During development, I used AI to resolve issues where some sports news could not be fetched. Prompts included:
        - "What might cause certain elements not to be scraped on the BBC innovation page?"
    -
- **Time Spent**:
    - 1 hour using AI for debugging issues with web scraping.
    - 30 minutes for documentation assistance and generating content for README.md.

7. Time Breakdown

- **Web Scraping**: 4 hours
- **Database Setup**: 2 hours
- **Backend API Development**: 3 hours
- **Frontend Development**: 5 hours
- **AI Tool Assistance**: 1.5 hours
- **Total Time Spent**: 15.5 hours