Project Report: Book Reminder Application

**1. Real World Problem Identification**

In today's busy world, people often struggle to keep track of the books they want to read or remember vital details about those books. Without a structured way to record books, authors, and reminder dates, readers may forget about books they have intended to read or lose motivation over time. A reliable way to track book details and set reminders for when to read them would make book tracking easier and more organized.

**2. Proposed Solution**

This Book Reminder application allows users to add books they want to remember, along with author names and the date/time they wish to be reminded about the book. The application offers a simple, intuitive interface to input this information and save it in a local SQLite database, making the data accessible even offline. When users return to the app, they can view the list of saved books and their reminders, providing a convenient way to manage reading goals.

The app features:

* A main screen with a grid layout highlighting added books.
* A floating action button to add new books.
* A form page to enter the book's name, author, and set a reminder time.
* SQLite database integration to store book information locally on the device.

**3. Responsive User Interfaces**

The app is designed to be responsive and adaptable across different screen sizes and orientations. The grid layout for displaying books adjusts based on the screen dimensions, ensuring a smooth user experience on both small and large screens. Here are screenshots of the app on various screen sizes:

* **Mobile (Portrait Mode)**: Displays a grid of books in two columns.
* **Tablet (Landscape Mode)**: Adapts to a larger screen with multiple columns for better use of space.
* **Web/Desktop**: If run on the web, the layout further adjusts to utilize more screen space and maintain readability.

*(Note: Actual screenshots can be inserted here for better documentation.)*

**4. Data Storage**

**Chosen Database**: SQLite

**Justification**: SQLite is a lightweight, serverless, and fully self-contained SQL database engine. It is ideal for applications that require persistent, local storage without the need for cloud-based synchronization. In the case of this book reminder app, SQLite offers a robust solution for storing book information (title, author, and reminder time) directly on the user's device. It ensures that users can access their data offline, making it especially convenient for a single-user app that does not require real-time data synchronization across devices.

SQLite also has the advantage of being widely supported and highly efficient for small to medium-sized datasets, making it well-suited to this app's needs.

**5. APIs/Packages/Plug-ins**

* **sqflite**: This package provides an interface for working with SQLite databases in Flutter. It allows CRUD (Create, Read, Update, Delete) operations, which are essential for managing the book records in the app.
* **Path Provider**: The path\_provider package is used to get the location of the database file on the device. It helps store the SQLite database in a suitable directory for consistent data persistence.
* **Image Picker (if used)**: If a book cover image is added to the form, the image\_picker package would allow users to select images from their device gallery or capture latest photos.
* **Flutter Form Validation**: Flutter's TextFormField widgets and validation help ensure that users enter the required details for book name and author name.

**6. Issues and Bugs Encountered and Resolved during Development**

1. **Database Path Setup Issue**
   * **Resolution**: Encountered issues in determining the correct path for storing the SQLite database file. Resolved by using the path\_provider package, which provides a platform-specific path for storing local files and database files.
2. **Data Persistence Testing**
   * **Resolution**: Ensured data was saved correctly by testing CRUD operations individually. Encountered issues with the data not showing up on app restart, which was resolved by re-initializing the database connection on app load.
3. **UI Overflow on Smaller Screens**
   * **Resolution**: Used responsive widgets like Expanded and Flexible, adjusted the layout to accommodate different screen sizes, and wrapped main components in a SingleChildScrollView to prevent overflow issues on smaller screens.
4. **Form Validation Errors**
   * **Resolution**: Added custom validation logic within TextFormField widgets to ensure that users enter required information before submitting the form.

**Problems**

Did not added the scaffold.

