# News Reader App

## 1. App Architecture and Design

**I have used MVVM for separation of concerns:**

- **Model**: Represents the news article data.
- **ViewModel**: Handles logic and communicates with the network layer.
- **View**: SwiftUI views displaying data and handling user interactions.

**I have Incorporated SOLID principles:**

- **S**ingle Responsibility: Separate each responsibility into distinct classes/modules.
- **O**pen-Closed: Ensure components are open for extension but closed for modification.
- **L**iskov Substitution: Use protocols for ViewModel and services.
- **I**nterface Segregation: Keep interfaces lean and specific.
- **D**ependency Inversion: Use dependency injection for testability and flexibility.

---

## 2. Key Features Implementation

### a. Browsing Articles

- Fetch articles using a public API like NewsAPI.
- Display a list of articles with titles and summaries in a **LazyVStack**.
- Integrate pull-to-refresh functionality using SwiftUI.

### b. Article Details

- Navigate to a detailed view using **NavigationLink**.
- Show full article content in a scrollable view.

### c. Bookmarking

- Maintain bookmarks in persistent storage like **UserDefaults**.
- Show bookmarked articles in a separate view.
- Allow users to add/remove bookmarks.

### d. Filtering by Category

- Use a **SegmentedControl** for category selection.

- Fetch articles filtered by the selected category.

**e. Sharing via Bluetooth**

- Implement Bluetooth Low Energy (BLE) using **CoreBluetooth**.
- Allow nearby devices to receive shared article summaries or links.

---

## 3. Technical Considerations

**a. Network Layer**

- Use **Combine** for handling asynchronous tasks and state updates.
- Create a generic network manager with robust error handling.
- Map API responses to Swift structs using **Codable**.

**b. Dependency Injection**

- Inject dependencies (e.g., NetworkManager) into ViewModels.
- Use a DI framework like **Resolver** for managing dependencies.

**c. Test Cases**

- Write **unit tests** for ViewModel and network calls.
- Write **UI tests** for major user flows using **XCTest**.

**d. Error Handling**

- Display user-friendly error messages using SwiftUI alerts.
- Handle API failures, empty states, and Bluetooth errors gracefully.

---

## 4. BLE Integration

- Implement a BLE central manager to discover and connect to nearby devices.
- Use BLE characteristics for transferring article data.
- Handle permissions and device states (powered on/off).

---

## 5. Testing Strategy

- Unit test ViewModels with mocked dependencies.

- UI test navigation, bookmarking, and BLE sharing functionality.
- Validate network calls using stubbed responses.