Student Number: 2307932

Student Name: Binod Bhandari

Module: 7cc012: Mobile Application Development

Produce a project plan for the development of the app.

Produce Software Functional Specification Document from the User Requirements above, for  
the Mobile App.

Produce a Software Design Document in any standard design document format, e.g. UML,  
Flow Charts, etc., for the app.

Develop the app. Test all implemented functionality. Record and document your test results,  
as well as any major development decisions and issues.

Be ready and able to demonstrate the Mobile App and explain the code in an oral viva. If you  
are unable to demonstrate and answer questions on your code satisfactorily, you will not  
pass the module.

……………………….. ……………………………..

**Project Brief: BlogApp - Mobile Application for Creating Blog.**

**Introduction:**

In this mobile app project, our goal is to help users manage their blogs without always needing the internet. Like finding workers, blogging can also be tricky without reliable internet. So, we're creating an app that lets users write, edit, and share blog posts even offline. It's all about making blogging easier for everyone.

**Project Overview:**

In the context of the blogging app project, this application aims to revolutionize how individuals and organizations manage their blogging activities. This app serves as a comprehensive platform for creating, editing, and sharing blog content offline, catering to users with varying levels of technical expertise. By aggregating blog data and providing seamless functionality, this app streamlines the blogging process, enabling users to efficiently share their thoughts and stories with the world.

**Goals and Objectives:**

Facilitate individuals and organizations in efficiently managing their blogging activities offline.

• Establish a user-friendly platform that caters to bloggers of varying technical proficiency levels.

• Provide a seamless and transparent system for creating, editing, and sharing blog content.

• Enhance the accessibility of blogging by enabling users to work offline and synchronize data when connected to the internet.

• Foster a vibrant community of bloggers by offering features that encourage collaboration and sharing of ideas.

• Empower users to have full control over their blog content, including easy management of posts and media attachments.

• Ensure reliability and efficiency in the offline functionality, allowing users to create and edit content anytime, anywhere.

• Promote engagement and interaction by implementing search features for exploring blog archives and discovering relevant content.

• Enhance the user experience through intuitive design and smooth navigation, ensuring a pleasant blogging experience.

**Technologies Used:**

* Programming Language: Dart - A popular JavaScript library for building user interfaces.
* Framework: Flutter - A PHP web application framework known for its simplicity and elegance.
* Database: Sqlite - A community-developed fork of MySQL, providing reliability and performance.

**Key Features:**

* Offline Blog Creation: Allow users to create, edit, and manage blog posts without requiring an internet connection.
* Local Database Storage: Store blog data locally on the device to ensure accessibility even when offline.
* Text Input: Provide a user-friendly interface for inputting blog titles and content.
* Image Attachment: Enable users to attach photos or images to their blog posts from the device's photo gallery or camera.
* Blog Management: Allow users to view, edit, and delete individual blog posts, as well as manage the overall list of blog items.
* Search Functionality: Implement a search feature to enable users to search for specific text within their blog posts or across the entire blog archive.
* Sharing Mechanism: Enable users to share individual blog posts via email using the standard platform "Share" mechanism.
* Group Deletion: Allow users to select and delete multiple blog items simultaneously for efficient management.
* Date Stamp: Automatically record the date of blog entry for each post to maintain chronological order. • User-Friendly Interface: Design an intuitive and easy-to-navigate interface suitable for users with varying levels of technical proficiency.

**Risks:**

* Data Loss: Risk of data loss due to local storage issues; mitigated by regular backups.
* Limited Offline Functionality: Users may experience feature limitations offline; clear guidance provided.
* Compatibility Issues: Possible inconsistencies across devices; thorough testing planned.
* User Adoption: Risk of low adoption if app doesn't meet expectations; continuous feedback loop established.
* Technical Challenges: Development may face bugs, performance issues; agile approach adopted for quick resolution.
* Regulatory Compliance: Legal risks due to non-compliance with data protection laws; ongoing monitoring ensured.
* Network Dependency: Certain features require network access; alternatives provided for offline use.

**Cost Considerations:**

* Development Costs: Investment required for development tools, software licenses, and hiring developers.
* Infrastructure Costs: Expenses for server hosting, database management, and maintenance.
* Operational Costs: Ongoing expenses for platform updates, customer support, and marketing efforts.
* Resource Allocation: Strategies for efficient resource allocation to optimize budget utilization.
* Cost Management: Measures to monitor and control expenses throughout the project lifecycle.
* Revenue Generation: Exploration of monetization strategies such as subscription models or in-app purchases to offset costs.
* Sustainability Plan: Long-term strategies for financial sustainability and scalability of the project.

**Project Development life cycle:**

The MeroKam app for Android and its database server uses a method called the Iterative Model. This means they start with basic features and keep improving them until the whole system is ready.

Instead of making a full list of what the app should do at the beginning, it starts with some features. Then, as it gets worked on, more features are added in each round.

The process has four steps: gathering what the app needs to do, designing how it will work, building and testing it, and then checking to see if it meets what's needed and making any necessary changes.

A diagram of a process

Description automatically generated

**Project planning and feasibility study:**

Before making any software, we need to understand what the system needs and plan the project based on that. We create a feasibility report to do a detailed analysis of the project idea. This report looks at three main things:

* 1. Economic feasibility: We check if the project can be built at a reasonable cost. We figure out if we need to reduce costs or not.
  2. Technical feasibility: We see if the software we're making is technically sound. This means it should work well with the system it's meant for.
  3. Organization feasibility: We check if the people in the organization can use the software. We especially look at whether employees, like receptionists, can use it easily.

**Requirement analysis:**

This phase is crucial in the Software Development Life Cycle (SDLC). It involves gathering detailed reports from various fields to find business solutions or create systems. It's essential to gather all requirements to avoid problems later. Clear understanding of how stock is currently recorded and how the owner wants it recorded in the new system is crucial in this phase.

**System design:**

Design refers to the modules used to build the software. There are different modules such as E-R diagram, DFD’s, flowchart, etc. So, it’s very difficult to code. Thus, we have used the concept of object-oriented programming. In this program the modules in the main menu and subdivision are the functions.

**Coding:** Based on the system design, we did the coding of the system. The coding is done using the programming language i.e. java and python programming language.

Testing: Testing is like finding and fixing mistakes in software. We test to make sure the program works right and to figure out why it might not. Testing helps make sure the software is good quality and works correctly. We do this by running the program with different types of data and checking if it behaves the way it's supposed to. So, when we finish testing, we know our software is error-free and works well.

**Debugging:**

We have debugged our program for finding the reducing the number of bugs, or defects, in a computer or a piece of electronic hardware thus making it behave as expected.

Although each debugging experience is unique, there are certain general principles we have applied in debugging. The basic steps that we have used in our program are: -

* Recognize that a bug exists.
* Isolate the source of the bug.
* Identify the cause of the bugs.
* Determine a fix for the bugs.
* Apply the fix and test it.

**Implementation:** System implementation generally focuses on the coding and installation of the system. Our system implementation is composed of activities, which are coding, testing and installation. The purpose of these steps is to convert the physical system specification into working and reliable software.

Coding is the process whereby the physical design specification created by the analysis team is turned into working computer code by the programming team. Once the coding has begun the testing procedure can begin and proceeds in parallel.

**Documentation and Evaluation:** We have documented all the activity performed during development of this system, which will be very helpful in the future modifications or changes. As per the time being if the vendor wants to make some amendments (changes) in the existing program of his system, the developer should edit the programs per his requirements. This phase of fulfilling the demands comes up with evaluation phase.

Manpower and Limitations:

Manpower:

The required human resource to operate the software: -

* Basic English knowledge.
* Familiar with basic android mobile system.
* Familiar with basic use of internet.

**Limitations:**

* Currently application is available in android platform only.
* Needed android mobile with at least Lollipop operating system version or higher.
* Should have good basic knowledge of the internet and Smartphone.
* User need to know privacy and policies.
* Should have basic knowledge of computer-based system.

**Cost Estimation:**

|  |  |  |
| --- | --- | --- |
| Sn | Activities | Price(Nrs) |
| 1. | Google play console | 3000 |
| 2. | Database server | 2000 |
| 3. | Data collection | 1000 |
| 4. | Internet and collaboration | 1000 |
| 5. | Designing and Testing | 2000 |
| 6. | Stationary related Expenses | 1000 |
| 7. | Miscellaneous Expenses | 1500 |
|  | Total | 11500 |

**Duration:**

|  |  |
| --- | --- |
| ACTIVITIES | DURATION(DAYS) |
| PROBLEM ANALYSIS | 5 |
| REQUIREMENT ANALYSIS | 20 |
| DESIGNING | 15 |
| CODING AND TESTING | 30 |
| IMPLEMENTATION | 5 |
| DOCUMENTATION | 15 |
| TOTAL ESTIMATES (DURATION) | 90 |

**Use Case Diagram:**

A use case diagram illustrates how different parts of a system interact. It's a tool used in system analysis to understand and organize requirements. It's commonly used in Unified Modeling Language (UML), a standard notation for modeling real-world objects and systems. Use cases can vary from planning requirements to testing software products. The diagram includes four main components: the system boundary, actors (individuals involved), use cases (specific actions), and the relationships between them. It looks similar to a flowchart and uses symbols to represent system elements.

A diagram of a company

Description automatically generated

**Data Flow Diagram:**

A Data Flow Diagram (DFD) visually depicts how a system processes, stores, and transfers data within itself and with its environment. It's a helpful communication tool between users and system designers. DFDs start with a broad overview and can be expanded into more detailed diagrams. They show functions or processes and how data flows between them. This helps understand the system's structure and operations.

A diagram of a company

Description automatically generated

**Gantt Chart:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | TASK | 2020 | | | | | | | | | | | | | | |
| S.N. | AUG. | | | | SEP. | | | | | | OCT. | | | | |
| 1st | 2nd | 3rd | 4th | | 1st | 2nd | 3rd | 4th | 1st | | 2nd | 3rd | 4th |
| 1. | Requirement Analysis |  |  |  |  | |  |  |  |  |  | |  |  |  |
|  |  |  |
| 2. | Designing |  |  |  |  | |  |  |  |  |  | |  |  |  |
|  |  | |  |  |  |  |
| 3. | Coding |  |  |  |  | |  |  |  |  |  | |  |  |  |
|  | |  |  |  |  |  | |  |
| 4. | Testing |  |  |  |  | |  |  |  |  |  | |  |  |  |
|  | |  |  |  |
| 5. | Implementation |  |  |  |  | |  |  |  |  |  | |  |  |  |
|  |  |  |
| 6. | Documentation |  |  |  |  | |  |  |  |  |  | |  |  |  |
|  |

**Conclusion:**

MeroKam aims to bridge the gap between workers and employers, providing a convenient and efficient platform for connecting individuals and organizations with short-term employment opportunities. By leveraging the power of web technologies such as React.js, Laravel, and MariaDB, MeroKam seeks to revolutionize the way workers and employers interact in Nepal's job market.

……………………….gimini …………………………..

**Project Plan**

1. **Definition**

**Project Scope:** Outline project boundaries, deliverables, and constraints.

**Project Boundaries**

* **Focus:** Development of a core offline blogging client with the ability to create, edit, manage, and share blog entries locally.
* **Excluded Features (for this initial scope):**
  + Cloud synchronization features for cross-device access.
  + Commenting or social interaction on blog posts.
  + Advanced WYSIWYG rich-text formatting in the editor (focus on plain text for now).
  + Integration with online blogging platforms.

**Deliverables**

* **Project Plan:** Document outlining timeline, milestones, resource requirements, and risk management.
* **Software Functional Specification:** Detailed description of user requirements and associated software functionalities.
* **Software Design Document:** Technical design outlining architecture, database structure, and key design choices (UML, flowcharts, etc.).
* **Flutter Mobile Application:** The developed application with core functionality, testable on both Android and iOS.
* **Test Documentation:** Records of test cases, results, and any bug fixes.
* **User Guide (Optional):** A simple manual if time and resources permit.

**Constraints**

* **Offline Functionality:** All core app features (except email sharing) must work without network connectivity.
* **Cross-Platform:** Code portability between Android and iOS is paramount.
* **Technology:** Flutter is the mandatory development framework.
* **Budget:** Assume a limited student project budget – consider open-source libraries wherever possible.
* **Time:** Presume development time aligns with academic or project deadlines.

* + **Goals and Objectives:** Define measurable success criteria.

Absolutely! Here's a breakdown of goals and objectives for the offline blogging app, along with how to measure their success:

**Goals**

* **Provide a reliable offline blogging experience.** Users should be able to seamlessly express their thoughts and ideas without network dependency.
* **Create an intuitive and user-friendly interface.** The app should be easy to learn and navigate, encouraging frequent use.
* **Ensure data persistence.** Blog entries must be saved reliably, protecting the user's work even in the event of app closure or device restarts.
* **Facilitate easy sharing.** Users should have a straightforward way to share their completed blog entries via email.

**Objectives & Success Criteria**

1. **Offline Functionality**
   * **Metric:** All core features (creating, editing, deleting, searching) function as intended with zero network connectivity.
   * **Measurement:** Conduct tests with device in airplane mode.
2. **Usability**
   * **Metric:** Average task completion time for new users (e.g., creating a new blog entry).
   * **Measurement:** Usability testing with a small group of target users, timing their tasks.
   * **Metric:** Positive user feedback on the UI's ease of use.
   * **Measurement:** Short survey or feedback feature built into the app.
3. **Data Reliability**
   * **Metric:** Zero reported instances of data loss during normal use.
   * **Measurement:** Extensive testing, involving app restarts, simulated crashes, and diverse usage patterns.
4. **Seamless Sharing**
   * **Metric:** Successful transmission of blog content via email through the 'Share' function.
   * **Measurement:** Testing the share function and verifying email delivery with content.

**Additional Considerations**

* **Performance:** App responsiveness under load (with a significant number of stored blog entries).
* **Security:** Consider basic measures to protect local data, especially if dealing with potentially sensitive content.

**Important Note:** Success criteria can be adjusted based on the project's scale and available time for testing.

**Stakeholders**

* **Project Owner:**
  + This could be you as the student undertaking the project, or your instructor/professor if it's an assigned task.
  + *Role:* Overall responsibility for project vision, securing resources (if any), resolving major roadblocks, and final approval.
* **Users:**
  + People who would use a mobile blogging app for offline functionality. This might include:
    - Journalers or diarists
    - Travelers who want to document experiences with limited connectivity
    - Writers who prefer distraction-free creation environments
  + *Role:* Provide input on desired features during the requirements stage. Might participate in usability testing.
* **Development Team**
  + Could be just you as the sole developer, or a small team if this is a collaborative project.
  + *Role:* Design, code implementation, testing, debugging, and documentation.
* **Potential Additional Stakeholders**
  + **Technical Advisor:** An instructor or a mentor if you need guidance on specific technical challenges.
  + **Industry Expert:** A professional blogger or content creator might offer insights if the project aims for wider use.

**Roles and Responsibilities**

It's important to define more granular roles within those stakeholder groups:

* **Project Owner**
  + Approves the project plan and budget.
  + Sets success criteria and makes final decisions.
* **Users**
  + Provide feedback on requirements and the app's usability.
* **Development Team**
  + **Lead Developer:** Assigns tasks, oversees technical decisions.
  + **Developers:** Build individual features, write test cases, fix bugs.
  + **Tester (if separate role):** Designs test plans and executes test cases.

1. **Planning**
   * **Requirements Breakdown:** Create a detailed Work Breakdown Structure (WBS) to organize tasks.

**1. Project Planning**

* **1.1 Define Project Scope**
  + 1.1.1 Outline deliverables, boundaries, and constraints
  + 1.1.2 Define goals and success metrics
  + 1.1.3 Identify stakeholders
* **1.2 Create Project Schedule**
  + 1.2.1 Develop task list and dependencies
  + 1.2.2 Estimate task durations
  + 1.2.3 Create a timeline with milestones (e.g., Gantt chart)
* **1.3 Resource Allocation**
  + 1.3.1 Determine team size and skill requirements
  + 1.3.2 Secure development hardware/software
* **1.4 Risk Assessment**
  + 1.4.1 Identify potential risks (technical, schedule, etc.)
  + 1.4.2 Develop mitigation plans

**2. Analysis and Design**

* **2.1 Requirements Gathering**
  + 2.1.1 Detail functional specifications
  + 2.1.2 Conduct user research (if feasible)
* **2.2 Software Design**
  + 2.2.1 Select architectural pattern (e.g., MVC)
  + 2.2.2 Design UI mockups or wireframes
  + 2.2.3 Design the database schema

**3. Development**

* **3.1 Set up Development Environment**
  + 3.1.1 Install Flutter and required tools
  + 3.1.2 Set up version control (e.g., Git)
* **3.2 UI Implementation**
  + 3.2.1 Build navigation structure (tabs, list views)
  + 3.2.2 Develop blog creation/editing forms
  + 3.2.3 Implement blog display and search
* **3.3 Database Integration**
  + 3.3.1 Set up local database (SQLite)
  + 3.3.2 Create data access layer (read, write, delete)
* **3.4 Image Handling**
  + 3.4.1 Integrate image picker (camera and gallery)
  + 3.4.2 Handle image storage and display
* **3.5 Implement Sharing**
  + 3.5.1 Integrate with device's email/share function

**4. Testing**

* **4.1 Develop Test Plans**
  + 4.1.1 Create unit test cases
  + 4.1.2 Create integration test cases
  + 4.1.3 Design system-level test scenarios
* **4.2 Execute Testing**
  + 4.2.1 Run test cases, record results
  + 4.2.2 Fix identified bugs
* **4.3 Usability Testing (if time allows)**
  + 4.3.1 Conduct user tests, gather feedback

**5. Deployment (Depending on project scope)**

* **5.1 App Store Preparation**
  + 5.1.1 Meet Apple/Google Play requirements
  + 5.1.2 Create app store listings
* **5.2 App Submission**

**6. Documentation**

* **6.1 Technical Documentation**
  + 6.1.1 Comment code
  + 6.1.2 Create diagrams (if design docs are extensive)
* **6.2 User Guide (optional)**

**Notes:**

* **Level of Detail:** You can break down tasks further for more granularity.
* **WBS Software:** Project management tools (e.g., Trello, Asana) can visually organize your WBS.
* **Iterative Approach:** Development might necessitate revisiting analysis/design if you face major hurdles.

* + **Schedule and Milestones:** Estimate task durations and dependencies, creating a timeline with key milestones

**Framework**

1. **Task Durations:**
   * Break down each task from the WBS into its smallest manageable unit.
   * Estimate in hours or half-day increments. It's better to overestimate slightly and have buffer time.
   * Consider your experience with Flutter: If you're new, add padding to your estimates for the learning curve.
2. **Dependencies:**
   * Identify sequential tasks: Tasks that cannot begin until another task is complete.
   * Identify potential parallel tasks: Tasks that can be worked on simultaneously.
3. **Milestones:**
   * Mark major completions:
     + Project Plan finalized
     + Design phase complete
     + Core functionality developed (e.g., create, edit, delete blogs)
     + All features complete
     + Testing and bug fixing complete

**Example Timeline (Hypothetical)**

Assuming you're a student with some Flutter experience, here's a VERY rough timeline. You'll need to adjust this drastically based on your own context.

* **Week 1**
  + Project Planning (2 days)
  + Requirement Gathering & Initial Design (3 days)
* **Week 2**
  + Set up Development Environment (1 day)
  + Begin Core UI and Database Work (parallel tasks, 4 days)
* **Week 3**
  + Continue UI and Database Development (4 days)
  + Image Handling Implementation (1 day)
* **Week 4**
  + Sharing Feature Implementation (2 days)
  + Thorough Testing and Bug Fixes (3 days)
* **Week 5**
  + Documentation (2 days)
  + Buffer/Contingency Time or Optional Features (3 days)

**Important Notes:**

* **Gantt Charts:** Use project management software to visualize the timeline and dependencies.
* **Time Constraints:** Be realistic about your weekly time commitment with other coursework.
* **Testing:** Testing time might increase depending on the complexity of your implementation.
* **Flexibility:** Be prepared to adjust and pivot as you progress.

**Guidance for Estimating Task Durations**

* **Reflect on Past Projects:** If you've done similar work before, use that experience as a baseline.
* **Break Down the Unfamiliar:** If you face a completely new type of task, break it down into smaller research and prototyping steps.
* **Get Second Opinions (if possible):** Discuss estimates with a mentor or fellow student working on a similar project.

* + **Resource Allocation:** Determine team needs, software, hardware, and other resources.

**Team Needs**

* **Developer (likely yourself):** Programming in Flutter, UI design, database work, testing.
* **Potential Additional Roles**
  + UI/UX Designer (if you want a highly polished look and feel).
  + Dedicated Tester (especially if the project gets complex).

**Software**

* **Flutter SDK:** Core development framework.
* **IDE or Code Editor:** Visual Studio Code or Android Studio are popular choices.
* **SQLite Plugin:** For local database handling (e.g., the sqflite package).
* **Image Handling Libraries:** If needed (consider built-in Flutter options first).
* **Email Sharing Plugin:** (e.g., share\_plus).

**Hardware**

* **Development Computer:** Any laptop or desktop capable of running your chosen IDE smoothly.
* **Android/iOS Device or Simulators:** For testing on both platforms.

**Other Resources**

* **Version Control:** Git, along with a hosting service like GitHub or Bitbucket.
* **Project Management Tools:** Trello, Asana, or similar for task tracking.

* + **Budgeting:** Project development, testing, and potential deployment costs.

**Development Costs**

* **Software:** The good news is that the core development tools (Flutter, IDEs, essential plugins) should be free to use.
* **Hardware:** You likely already own a suitable development computer. If not, this could be a significant cost.
* **Developer Time:** The major 'cost' is your own time investment. Consider what your time translates to if this were a paid project.

**Testing Costs**

* **Devices:** If you don't own both Android and iOS devices, factor in the potential cost of acquiring a testing device or using cloud-based testing services.
* **User Testing:** If you conduct user testing, consider if there are any small incentives (gift cards, etc.) for participants.

**Potential Deployment Costs**

* **App Store Fees:** Google Play and Apple App Store have developer accounts with associated fees.
* **Marketing:** Even minimal promotion of your app might incur small costs (website, social media ads).

* + **Risk Management:** Identify potential risks and create plans to mitigate them.

**Technical Risks**

* **Data Loss:**
  + **Mitigation:** Thoroughly test database operations (saving, restoring, unexpected app closure). Implement backup mechanisms if possible.
* **Cross-Platform Incompatibility:**
  + **Mitigation:** Test regularly on both Android and iOS simulators/devices. Utilize Flutter's built-in tools for platform-awareness.
* **Performance Issues:** (with many blog entries)
  + **Mitigation:** Optimize database queries, consider image compression techniques.
* **Scope Creep:** Adding too many features
  + **Mitigation:** Prioritize the core functionality, keep the initial scope well-defined.

**Project Management Risks**

* **Missed Deadlines:**
  + **Mitigation:** Set realistic timelines with buffers. Break down tasks into small manageable units. Track progress regularly.
* **Underestimating task complexity:**
  + **Mitigation:** Do small prototypes or research spikes for unfamiliar features. Get advice from a mentor if available.

**External Risks**

* **Changes in Flutter Framework:**
  + **Mitigation:** Stay updated on Flutter releases, adapt the code if major updates break functionality.
* **Loss of work:** (computer failure)
  + **Mitigation:** Use version control (Git) and back up your project consistently.

**Risk Management Plan**

**Risk Prioritization:** Focus on mitigating high-impact and probable risks first.

* **Communication:** If working in a team, have open channels to discuss potential risks.
* **Contingency Planning:** Have basic plans for "what if" scenarios (e.g., how you might re-scope the project if facing significant delays).

1. **Execution**
   * **Software Design:** (covered in more detail below).
   * **Development:** Use Flutter to build the UI and core functionality, adhering to portability requirements.
   * **Testing:** Create test cases for unit, integration, and system-wide testing. Record results thoroughly.
   * **Iteration:** Revise code and design based on testing results.
2. **Monitoring and Control**
   * **Progress Tracking:** Use project management tools to monitor against timelines and budget.
   * **Quality Control:** Implement code reviews and maintain a bug tracking system.
   * **Change Management:** Create a formal change control process.
3. **Closure**
   * **Documentation:** Finalize technical documentation and user manuals.
   * **Deployment (if applicable):** Prepare for app store submission.
   * **Project Retrospective:** Review successes and opportunities for improvement.

**Software Functional Specification**

**Features**

* **Blog Creation:**
  + Input fields for Title, Date (auto-populated), Body Text, Image.
  + Image attachment from Gallery or Camera.
* **Blog Management:**
  + Listing blog entries (titles, preview text, date).
  + Viewing individual blog entries.
  + Editing (title, body text, image swap).
  + Deleting a single entry.
  + Bulk-deleting selected entries.
  + Search by text (within titles or full body text).
* **Offline functionality**
  + Data persistence using SQLite or similar local database.
* **Email Sharing**
  + Integration with the device's default 'Share' mechanism.

**User Interface**

* Intuitive navigation (tabs, lists, clear buttons).
* Visual hierarchy for blog listing.
* Clean WYSIWYG-like editor, if resources permit, otherwise, plain text with image preview.

**Software Design**

* **Model-View-Controller (MVC) Architecture:** Promotes code separation and maintainability.
* **State Management:** Consider Flutter's built-in options (Provider) or more robust solutions (BLoC) depending on project complexity.
* **Image Handling:**
  + Implement file selection and preview.
  + Consider image compression for storage optimization.
* **Database Design:** Schema for blog entries with appropriate data types.

**Development Considerations**

* **Flutter Widgets:** Familiarize yourself with layouts, input fields, image display, and list views.
* **Database Integration:** Utilize the sqflite package for Flutter.
* **Email Integration:** Use Flutter's share\_plus or similar package to interact with the default email client.
* **Cross-platform Compatibility:** Careful testing on both Android and iOS simulators/devices.