Web Task

Objective

Build a web-based Library Management System that simulates standard member-side library operations. The app should have a clean, responsive UI and clear user workflows.

Authentication (simple)

- Basic sign-up / login for library members.

Requirements (must implement)

- Catalog: Browse, filter, and sort books.
- Book Data (read-only in app): Display book metadata, categories/shelves, and availability.
- Users can borrow, return, renew, and reserve books — the system tracks due dates, availability, fines (only calculated, not paid), and a basic queue for reservations.
- Reading: View free books in-browser (PDF viewer or iframe) and support PDF uploads for free titles during development/seeding.
- Clean UI: Responsive, accessible, intuitive navigation and empty-state handling.
- State & Validation: Prevent double-borrows, show reservation position, block renewals if reserved by another user, etc.

Suggested Routes / Pages

- / — Home / Featured books
- /login, /register — Auth
- /catalog — Catalog & search
- /book/:id — Book detail (borrow, reserve, read)
- /reader/:id — PDF reader / iframe view
- /dashboard — Member dashboard (active loans, reservations, fines history)

Example Workflows

- Catalog seeding (dev script) → categories/shelves applied → visible in catalog.
- Borrow: member requests borrow → loan record created → due date set → availability updates.
- Return: member returns → availability updates → system calculates any logical fine (record only).
- Renew: allowed if not reserved and under policy limits.
- Read free book: open in PDF viewer / iframe (no payment).

Deliverables

- Source code (frontend + backend).
- Short report: architecture overview, routes, database schema, lending rules, seeding approach, and PDF viewer integration.

Bonus (extra credit)

- Razorpay integration for optional payments (e.g., fines or paid rentals/purchases): Create a mock checkout for fines or rentals, handle success/failure webhooks, and record transactions. (Keep core app fully usable without payments.)
- Search enhancements: autocomplete, advanced filters, or full-text search.

(Note: You may create your own apis or use public apis. If public apis used then a good frontend is expected)

AIML Task

Welcome to the Co-Comm interviews! You are required to complete one of the following tasks (GenAI or ML). Please read the requirements carefully and submit your solutions as per the submission process.

Task 1: GenAI – RAG with LangChain

Requirements

- Implement a Retrieval-Augmented Generation (RAG) system using LangChain.
- Allow the user to upload PDFs.
- Use embeddings + vector database for document storage and retrieval.
- Generate contextual answers to user queries using the uploaded PDFs.
- Have a UI where the user can chat with the system.

Bonus Points

- Responsive & professional UI (clean, intuitive, and mobile-friendly).
- Context management: Previous conversation + PDF context should persist for longer queries.
- Smooth user experience with clear error handling.

Task 2: Machine Learning – Movie Recommendation System

Dataset: IMDB Dataset of Top 1000 Movies and TV Shows (Kaggle)

Requirements

- Preprocess and clean the dataset.
- Implement a recommendation algorithm (Content-based, Collaborative Filtering, or Hybrid approach).
- Be able to recommend movies/shows based on user input (e.g., title or genre).

Bonus Points

- Frontend Integration – Build a simple web interface where users can input a movie/show and get recommendations.
- Well-documented code with clear explanation of approach.

Submission Process

- You can choose to complete either Task 1 (GenAI) or Task 2 (ML).
- Create a public GitHub repository with a folder for the chosen task.

- Each folder should contain:
- Source code
- README.md with instructions on how to run the project
- .env.example file if environment variables are used
- Any requirements files (requirements.txt, package.json, etc.)
- Provide a README in the root folder describing:
- Your approach for the task
- Tools/libraries used
- Any challenges faced and how you solved them
- Optional but recommended: Deploy your project (Streamlit, HuggingFace Spaces, Vercel, Netlify, etc.) and add the demo link in the README.

Evaluation Criteria

- Functionality (Does it work as expected?)
- Code Quality & Documentation
- UI/UX (for Task 1 and Bonus in Task 2)
- Creativity & Bonus Implementation

App Task - 1

Overview

The app should simulate a tool for managing tasks in team projects. The goal is to allow users to assign tasks to team members, track their status, and perform basic management operations.

Key Requirements

- Frontend: Flutter OR React Native (latest stable version recommended).
- Backend: Any backend service for data storage and CRUD operations.
- State Management: Use a robust solution for state management (Preferably Provider).
- Cross-Platform: App should work on both Android and iOS devices/simulators.
- Error Handling: Manage network failures and invalid inputs gracefully.

Submission

- GitHub repository link with the complete code.
- A README.md file containing:
- Setup instructions.
- Backend configuration details.
- API keys or mock data (if applicable).
- During the interview, you will demo the app and walk through your code.
- Focus on clean, modular code that is easy for a team to collaborate on.

Core Features (Mandatory)

1. User Authentication

- Implement a basic login/signup system.
- Authentication can be via email/password.
- Predefined team members may be used for simplicity.

2. Task Creation (Create)

A screen/form to create a new task.

Fields:

- Task title
- Description
- Assignee (from a list of team members)
- Due date (date picker)
- Initial status (e.g., Pending)

- Validate required fields before saving.

3. Task List View (Read)

- Home screen displaying all tasks fetched from the backend.
- Show:
- Title
- Assignee
- Due Date
- Status
- Support basic sorting/filtering (by status or assignee).

4. Task Details and Update (Update)

- Tapping a task opens a detail screen.
- Allow editing/updating:
- Status (e.g., Pending → In Progress → Completed)
- Description
- Assignee
- Sync updates with backend and refresh UI.

5. Task Deletion (Delete)

- Option to delete tasks from the list or detail screen.
- Show confirmation dialog before deletion.

6. State Management & Sync

- Use a state management solution (e.g Provider).
- Handle:
- Loading states
- Errors
- Real-time UI updates after CRUD operations
- Show indicators for API calls and errors (e.g., “No internet connection”).

Bonus Features (Optional, Extra Points)

- Offline Support with Local Storage:
- Use Hive, SQLite, or SharedPreferences.
- Cache tasks or allow offline drafts.
- Sync when back online.
- Notifications:
- Local notifications for due dates or new assignments.

- Push notifications (via Firebase Cloud Messaging).
- User Profiles/Team Management:
- Screen to manage team members.
- Show tasks filtered by user (e.g., "My Tasks").
- Search & Advanced Filtering:
- Search bar to filter tasks by title/description.
- Date-based filters.
- Highlight overdue tasks.

Evaluation Criteria

- Clean, maintainable Flutter code.
- Proper backend integration with CRUD operations.
- Correct use of state management.
- Error handling and user-friendly UI.
- Bonus features (if implemented).

Deliverables

- GitHub Repository with full source code.
- README.md with clear setup instructions.
- Backend/API setup details.
- Working demo during the interview.

App Task - 2

Overview

This alternative assignment is UI-focused and emphasizes local data persistence without requiring a backend. The goal is to demonstrate strong UI/UX skills,

Key Requirements

- Frontend: Flutter OR React Native (latest stable version recommended).
- Local Storage: Use Hive, SQLite, or SharedPreferences to store tasks.
- State Management: Must use Provider for handling state changes.
- Notifications: Implement local notifications to remind users of task due dates.
- Cross-Platform: Should work on both Android and iOS.

Submission

- GitHub repository link with full code.
- A README.md with setup and instructions.
- Demo during the interview.

Core Features (Mandatory)

1. Task Creation (Create)

A form to create a task with fields:

- Title
- Description
- Due Date (date picker)
- Status (Pending by default)
- Save tasks to local storage.

2. Task List View (Read)

- Display tasks stored locally.
- Show:
- Title
- Due Date
- Status
- Tasks should be visually organized (e.g., Pending, In Progress, Completed).

3. Task Update

- Edit task details (status, description, due date).
- Reflect changes immediately in UI and update local storage.

4. Task Deletion

- Delete a task from storage.
- Confirm deletion with a dialog.

5. Notifications

- Use flutter_local_notifications to remind users about upcoming deadlines.

6. Search & Filter

- Add a search bar to filter tasks by title or description.
- Provide filters by status (Pending, In Progress, Completed).

Bonus Features

- Dark/Light mode toggle.
- Sort tasks by due date.
- Overdue task highlighting.

Evaluation Criteria

- Clean, intuitive, Professional UI design.
- Correct use of Provider for state management.
- Efficient use of local storage.
- Functional local notifications.
- Search and filter implementation.

Deliverables

- GitHub Repository with full source code.
- README.md with clear setup instructions.
- Working demo during the interview.

Do one of the two tasks to demonstrate your skills, no need to do both the tasks.