SDE Intern Assignment

plum

SDE Intern Assignment

Instructions:

- Please **choose any 1 of the following 4** problem statements for your submission
- Read the **submission guidelines** and **evaluation criteria** carefully for the chosen problem statement as mentioned below.
- **Submission Timeline:** You have a total of **4 days** to complete the assignment. The solution needs to be submitted by the 5th day from receiving the problem statement.

Submission Instructions:

- Provide a working demo (screen recording for mobile or hosted interactive link for web).
- Share source code (GitHub repo or ZIP).
- Include a README file with:
  - Prompts used and refinements made.
  - Architecture explanation and state management choices.
  - Screenshots of key screens or screen recordings (for mobile).
  - Known issues and potential improvements.

Evaluation Criteria:

- AI prompt quality and consistency of output.
- UI/UX polish, smooth navigation, error/loading states.
- Code quality, modularity, and component reusability.
- Proper async handling (loading indicators, retries).
- Creativity and attention to detail.

Problem Statement 1: AI-Based Health News Curator

Problem Statement 1: AI-Based Health News Curator

Task: Summarize and simplify health news articles into a daily feed.

Expected Flow:

- Screen 1: Load mock news articles or RSS dump.
- Screen 2: AI summarizes each into 2-line TL;DR + 3 key takeaways.
- Screen 3: Display feed with pagination and pull-to-refresh.
- Screen 4: Expand article →AI rewrites in simpler, friendly tone.

AI Usage Guidance:

Focus on clean, consistent summary formatting. Handle regenerate and refresh states gracefully.

Sample README – AI-Based Health News Curator:

1. Project Setup & Demo

Web: Run `npm install && npm start` to launch locally.
Mobile:
- For iOS: open project in Xcode and run on simulator/device.
- For Android: run `./gradlew assembleDebug` or launch via Android Studio.
Demo: Provide a screen recording (for mobile) or hosted link (for web).

2. Problem Understanding

Summarize your understanding of the problem and mention assumptions made.

3. AI Prompts & Iterations

Document your initial prompts, issues faced, and refined prompts for better results.

4. Architecture & Code Structure

- `App.tsx` or mobile `NavigationHost` manages navigation.
- Separate components/screens for each step.
- `aiService.ts` / `AIClient.swift` / `AIRepository.kt` handles AI calls.
- Use React Context, SwiftUI ObservableObject, or Jetpack ViewModel for state management.

5. <u>Screenshots / Screen Recording</u>

Attach screenshots (web) or screen recording (mobile) covering all screens.

6. <u>Known Issues / Improvements</u>

List bugs or limitations and how you'd improve them with more time.

7. <u>Bonus Work</u>

Mention any extra polish or features added (animations, dark mode, etc.).

Problem Statement 2: AI-Assisted Knowledge Quiz

Problem Statement 2: AI-Assisted Knowledge Quiz

Task: Generate quiz questions using AI and display an interactive, navigable quiz experience.

Expected Flow:

- Screen 1: Topic selection screen (e.g. Wellness, Tech Trends).
- Screen 2: AI generates 5 MCQs with options and correct answer flag. Show loader while fetching.
- Screen 3: Display questions one by one with next/previous navigation and progress bar.
- Screen 4: On completion, AI generates a custom feedback message based on score.

AI Usage Guidance:
Prompts must produce consistent JSON output. Add error handling and retry if malformed. Build reusable question component.

Sample README – AI-Assisted Knowledge Quiz:

1. Project Setup & Demo

Web: Run `npm install && npm start` to launch locally.
Mobile:
- For iOS: open project in Xcode and run on simulator/device.
- For Android: run `./gradlew assembleDebug` or launch via Android Studio.
Demo: Provide a screen recording (for mobile) or hosted link (for web).

2. Problem Understanding

Summarize your understanding of the problem and mention assumptions made.

3. AI Prompts & Iterations

Document your initial prompts, issues faced, and refined prompts for better results.

4. Architecture & Code Structure

- `App.tsx` or mobile `NavigationHost` manages navigation.
- Separate components/screens for each step.
- `aiService.ts` / `AIClient.swift` / `AIRepository.kt` handles AI calls.
- Use React Context, SwiftUI ObservableObject, or Jetpack ViewModel for state management.

5. Screenshots / Screen Recording

Attach screenshots (web) or screen recording (mobile) covering all screens.

6. Known Issues / Improvements

List bugs or limitations and how you'd improve them with more time.

7. Bonus Work

Mention any extra polish or features added (animations, dark mode, etc.).

Problem Statement 3: AI-Generated Wellness Board

<u>Problem Statement 3:</u> AI-Generated Wellness Recommendation Board

Task: Display a personalized board of health tips and allow users to explore and save them.

Expected Flow:

- Screen 1: Profile capture (age, gender, goal selection).
- Screen 2: AI generates 5 tips. Display as scrollable cards with icons and short titles.
- Screen 3: Tap card → AI generates longer explanation & step-by-step advice.
- Screen 4: Allow saving favorite tips and persist locally.

AI Usage Guidance:

Ensure prompts generate concise, engaging recommendations. Implement regenerate option for fresh results.

Sample README – AI-Generated Wellness Recommendation Board

1. <u>Project Setup & Demo</u>

Web: Run `npm install && npm start` to launch locally.
Mobile:
- For iOS: open project in Xcode and run on simulator/device.
- For Android: run `./gradlew assembleDebug` or launch via Android Studio.
Demo: Provide a screen recording (for mobile) or hosted link (for web).

2. <u>Problem Understanding</u>

Summarize your understanding of the problem and mention assumptions made.

3. <u>AI Prompts & Iterations</u>

Document your initial prompts, issues faced, and refined prompts for better results.

4. <u>Architecture & Code Structure</u>

- `App.tsx` or mobile `NavigationHost` manages navigation.
- Separate components/screens for each step.
- `aiService.ts` / `AIClient.swift` / `AIRepository.kt` handles AI calls.
- Use React Context, SwiftUI ObservableObject, or Jetpack ViewModel for state management.

5. Screenshots / Screen Recording

Attach screenshots (web) or screen recording (mobile) covering all screens.

6. Known Issues / Improvements

List bugs or limitations and how you'd improve them with more time.

7. Bonus Work

Mention any extra polish or features added (animations, dark mode, etc.).

Problem Statement 4: AI-Powered Benefits Discovery

Problem Statement 4: AI-Powered Benefits Discovery Flow

Task: Your task is to create a multi-screen flow where an employee enters a health-related need in free text, AI classifies it into a benefit category, and recommends suitable benefits with a generated step-by-step action plan.

Expected Flow:

- Screen 1: Free-text input screen with friendly placeholder (e.g. 'I have tooth pain, what can I do?').
- Screen 2: AI classification into categories like Dental, Mental Health, Vision. Show a loading indicator during processing.
- Screen 3: Display 2–4 benefit cards with mock data (title, coverage, description).
- Screen 4: On selecting a benefit, AI generates a 3-step action plan explaining how to avail the benefit.

AI Usage Guidance:

Prompts must be designed to return just the category name and then structured step-by-step instructions. Include fallback prompts for unrecognized inputs and provide a regenerate option for users.

Sample README – AI-Powered Benefits Discovery Flow:

1. Project Setup & Demo

Web: Run `npm install && npm start` to launch the React app locally.
Mobile:
- For iOS: open project in Xcode and run on simulator/device.
- For Android: run `./gradlew assembleDebug` or launch via Android Studio.
Demo: Provide a screen recording (for mobile) or hosted link (for web).

2. Problem Understanding

Goal: Classify employee health needs into categories and display benefits with action plans.

Assumption: Benefits are loaded from mock JSON, no real backend call.

3. AI Prompts & Iterations

Initial Prompt: 'Classify the following text into Dental, OPD, Vision, Mental Health.'
Issue: Returned verbose explanation.
Refined Prompt: 'Return ONLY the category name from {Dental, OPD, Vision, Mental Health} that matches the text: {user_input}. Nothing else.'

4. <u>Architecture & Code Structure</u>

- `App.tsx` or equivalent mobile `NavigationHost` manages navigation.
- `BenefitInput`, `BenefitList`, `BenefitDetails` as separate components/screens.
- `aiService.ts` / `AIClient.swift` / `AIRepository.kt` handles AI calls and retries.
- Use state management (React Context, SwiftUI ObservableObject, Jetpack ViewModel) to persist selected benefit.

5. <u>Screenshots / Screen Recording</u>

Attach screenshots for web or provide a screen recording for mobile app flow.

6. <u>Known Issues / Improvements</u>

- Occasionally misclassifies when input is vague.
- Next improvement: add clarifying question fallback.

7. <u>Bonus Work</u>

Added Lottie-based loading animation and optional dark mode support.