SRS Project Kairo (Καιρός)

Team Members: Enmanuel De Los Santos Cruz
Description of project: Kairo is a full-stack, voice-first web application designed to be an intelligent journal and reflective assistant. It uses a sophisticated AI pipeline to transcribe a user's spoken entries and then analyzes them for sentiment and key themes. The user can then have a natural language conversation with their own journal, asking questions and discovering patterns in their own thoughts.

Section 1:
Introduction (WHY this idea?): Traditional journaling is an effective tool for self-reflection, but it has two major points of friction: it's tedious to write consistently, and the resulting log of text is difficult to analyze for meaningful insights. Most people stop journaling because it feels like a chore, and even if they persist, their journal remains a passive collection of data rather than an active tool for growth.

Purpose (WHAT your project/app will accomplish): Kairo solves both of these problems. First, it lowers the barrier to entry by being "voice-first," allowing a user to capture their thoughts as naturally as talking. Second, it acts as a personal data analyst, automatically running an AI pipeline on each entry to add structure (like sentiment) and enabling a powerful Q&A feature. The purpose is to transform a journal from a simple logbook into an active, conversational partner that helps a user understand themselves better.

Scope (What your project will do and what it will not): This project will focus on delivering a secure, single-user web application.

IN SCOPE: A secure user authentication system, voice recording and transcription, saving entries to a database, performing sentiment analysis, and a conversational Q&A interface.

OUT OF SCOPE: This project will not be a mobile-native (iOS/Android) application. It will not support multiple users per account, image/video uploads, or real-time collaborative journaling.

Technologies Used:

Backend: Python 3.9, FastAPI, Uvicorn (ASGI Server)

Frontend: React.js (JavaScript/HTML/CSS)

Database: PostgreSQL

AI/ML: Hugging Face transformers library, openai/whisper-base (for speech-to-text), bert-base-uncased (for text analysis), librosa (for audio processing)

Real-time Communication: WebSockets

System Dependencies: Homebrew, FFmpeg

This list defines the Minimum Viable Product (MVP).

REQ-M1: User Authentication. The system shall allow a new user to create a secure account with an email and a password.

Success Measure & Demonstration: I will show the "Sign Up" page, enter a new email and password, and show that a new user record is successfully created in the PostgreSQL database with a hashed password.

REQ-M2: User Login. The system shall allow an existing user to log in with their email and password.

Success Measure & Demonstration: I will show the "Login" page. I will first attempt to log in with an incorrect password and show the error. I will then log in with the correct credentials and be redirected to the main journal page.

REQ-M3: Voice Entry Creation. The system shall allow a logged-in user to record an audio journal entry from their browser.

Success Measure & Demonstration: I will press the "Record" button in the web interface, speak a sentence ("This is my first test entry"), and press "Stop." The audio data will be successfully captured by the browser.

REQ-M4: Entry Processing & Saving. The system shall process and save the audio entry, associating it with the logged-in user.

Success Measure & Demonstration: After recording, I will show the whisper-base model transcribing the audio. I will then check the database and show that the new text ("This is my first test entry") is saved in the journal_entries table, linked to my user ID.

REQ-M5: View All Entries. The system shall display a reverse-chronological list of all past journal entries for the user.

Success Measure & Demonstration: I will log in and the main page will display a list of all my previously saved entries, with the newest entry ("This is my first test entry") at the top.

REQ-M6: Entry Deletion. The system shall allow a user to permanently delete a specific journal entry.

Success Measure & Demonstration: I will click a "Delete" icon next to an entry. The system will ask for confirmation. After confirming, the entry will disappear from the list, and I will show that the corresponding row has been deleted from the database.

Section 2b: Stretch Requirements
This list defines additional features that make the project exceptional.

REQ-S1: Sentiment Analysis. The system shall automatically perform and display sentiment analysis for each entry.

Success Measure & Demonstration: When a new entry is saved, the bert-base model will analyze it. I will record "I am having a wonderful day," and the entry will appear in the list with a "Positive" tag. I will then record "This is a bad day," and it will show a "Negative" tag.

REQ-S2: Keyword Search. The system shall allow a user to search their entries for a specific keyword.

Success Measure & Demonstration: I will use a search bar to type the word "test." The main list will filter to show only the entries that contain the word "test."

REQ-S3: Conversational Q&A. The system shall allow a user to ask a natural language question about their journal.

Success Measure & Demonstration: I will go to a Q&A interface and type, "What was I talking about last week?" The system will use its AI pipeline (RAG) to find and display the relevant journal entries from that time period.

REQ-S4: Real-time Interaction. The system shall use WebSockets for real-time, conversational interaction.

Success Measure & Demonstration: I will demonstrate the Q&A interface. When I ask a question, the answer will appear on the page instantly without the page having to do a full reload, showing a live, two-way connection.

Section 2c: Weekly Schedule
Note: Project is recovering from a 2-week delay in Weeks 3-4 due to environment setup. This schedule reflects an accelerated plan to get back on track.

Weeks 1-2 (9/20 - 10/3): Planning. Project ideation, proposal drafting, and submission.

Weeks 3-5 (10/4 - 10/24): Troubleshooting. (Lost time) Solved major technical blockers, stabilized environment, installed all dependencies (FFmpeg), and successfully benchmarked all AI models.

Week 6 (10/25 - 10/31): Backend Foundation (Phase 1). Install FastAPI/Uvicorn. Create "Hello World" server. Design PostgreSQL database schema.

Week 7 (11/1 - 11/7): Backend Auth. Build and test /register and /login API endpoints with password hashing and JWT token generation.

Week 8 (11/8 - 11/14): Backend Core. Build and test API endpoints for creating, retrieving, and deleting journal entries. Connect all endpoints to the PostgreSQL database.

Week 9 (11/15 - 11/21): Frontend Foundation (Phase 2). Set up React.js project. Build the reusable components (buttons, layout). Create the Login and Registration pages.

Week 10 (11/22 - 11/28): Frontend-Backend Integration. Connect the React frontend to the FastAPI backend. Prove that a user can successfully log in and that the auth token is stored.

Week 11 (11/29 - 12/5): Core Feature Integration (Phase 3). Build the main journal page. Implement browser-based audio recording and send the audio to the backend. Display the list of transcribed entries.

Week 12 (12/6 - 12/12): AI Integration (Phase 4). Integrate the bert-base model to perform sentiment analysis (Stretch Goal 1). Implement the keyword search feature (Stretch Goal 2).

Week 13 (12/13 - 12/19): AI Q&A. Implement the RAG pipeline for the conversational Q&A feature (Stretch Goal 3). Refine the UI/UX.

Week 14 (12/20 - 12/26): Final Polish. Bug fixing, final testing, and preparation for the final project demonstration.

Section 3: Design Overview of the Product
Workflow:

First-Time User: The user visits the web app and is presented with a Login/Register page. They register for a new account.

Login: The user logs in. The server validates their credentials and sends back a JSON Web Token (JWT). The React app stores this token.

Main Page: The user is directed to their main journal page. The app uses the JWT to request and display all their past entries.

Record Entry: The user clicks "Record," speaks their entry, and clicks "Stop."

Processing: The React app sends the audio data (via WebSocket or REST) to the FastAPI server.

AI Pipeline: The server: a. Uses librosa to format the audio. b. Uses whisper-base to transcribe it to text. c. Uses bert-base to analyze the text for sentiment. d. Saves the text, timestamp, and sentiment to the PostgreSQL database.

Reflection: The user can then use the search bar or the Q&A interface to ask questions about their entries.

Resources:

Architecture: This is a multi-tier, client-server application.

Front-End (Client): A React.js single-page application that runs in the user's browser. It handles all UI and user interaction.

Back-End (Server): A FastAPI (Python) application that serves as the API. It handles all business logic, AI processing, and database communication.

Database: A PostgreSQL database that stores all user data.

AI Models: Hugging Face Transformers (Whisper and BERT) are loaded by the FastAPI server to perform their tasks.

Interconnection: The Front-End (React) communicates with the Back-End (FastAPI) using HTTPS (for REST APIs) and WebSockets (for real-time data). The Back-End communicates with the Database (PostgreSQL) directly.

Data at Rest:

Data will be stored in a PostgreSQL database.

User Profile Data: This will be stored in a users table. The user's password shall be stored as a secure hash (e.g., using passlib), never as plaintext.

Application Data: This will be stored in a journal_entries table. This table will contain the entry ID, the user ID (as a foreign key), the timestamp, the transcribed text, and the calculated sentiment score.

Data on the Wire:

All communication between the React client and the FastAPI server will be encrypted using HTTPS.

Standard user data (login, fetching entries) will be transmitted as JSON payloads over RESTful API endpoints.

Real-time audio uploads and Q&A responses will be transmitted over a WebSocket connection for low-latency, bidirectional communication. All AI processing occurs on the server.

Data State: Here is a diagram representing the flow of information for the core "Create Entry" function.

HMI/HCI/GUI: The user interface will be minimalist and clean, focusing on the task at hand.

Login Screen: A simple form with "Email" and "Password" fields, and a "Login" button.

Main Journal Screen: A two-column layout. The left column will be a list of past entries (showing date and a snippet). The right column will be the main interaction area, featuring a large "Record" button. When not recording, this area will house the Q&A chat interface.

Entry List Item: Each item in the list will show the date, the first line of the text, and a small visual tag (e.g., 😊, 😐, 😞) representing the sentiment.

Section 4: Verification
Demo: My demonstration will follow a complete user story:

I will show the login page and log in as a test user.

I will show the main journal page, which will have a few pre-existing entries.

I will press "Record" and say: "I am finally getting my backend server to work, which makes me feel very positive and productive."

I will press "Stop." The new entry will appear at the top of the list almost instantly, correctly transcribed, and showing a "Positive" (😊) sentiment tag.

I will then go to the Q&A box and type: "How was I feeling about my server?"

The system will present the entry I just recorded as the top result, successfully demonstrating the full, end-to-end data pipeline.

Testing: This section maps each requirement to its specific, testable acceptance criteria.

REQ-M1: A test user can be created. The password in the database must be a non-plaintext hash. Attempting to create a user with a duplicate email must fail.

REQ-M2: A user with valid credentials must receive a JWT. A user with an invalid password must receive a 401 Unauthorized error.

REQ-M3: Clicking "Record" must activate the browser's microphone. Clicking "Stop" must produce a usable audio data object.

REQ-M4: A 15-second audio clip must be transcribed by Whisper in under 3 seconds. The resulting text must be saved to the correct user's account in the database.

REQ-M5: Logging in must immediately populate a list of entries. Creating a new entry must place it at the top of this list without requiring a page refresh.

REQ-M6: Clicking "Delete" and "Confirm" must remove the entry from the UI. The corresponding row in the journal_entries table must be deleted.

REQ-S1: The text "I love this" must be classified as "Positive." The text "This is terrible" must be classified as "Negative."

REQ-S2: Given 10 entries, 3 of which contain the word "Kairo," searching for "Kairo" must return only those 3 entries.

REQ-S3: Given an entry: "I worked on my Kairo project today," asking "What did I work on today?" must retrieve this entry.

REQ-S4: During the Q&A demo, the browser's "Network" tab must show data being transmitted over a "WS" (WebSocket) connection, not a new HTTP request.

Sources/Citation/Resources Links:

FastAPI: https://fastapi.tiangolo.com/

React: https://react.dev/

Hugging Face Transformers: https://huggingface.co/docs/transformers/index

PostgreSQL: https://www.postgresql.org/docs/

Uvicorn: https://www.uvicorn.org/