PROJECT REPORT

Cura AI - AI-Powered Symptom Checker with Multilingual Support

SUBMITTED BY:

NASIRA RIAZ

SUBMITTED TO:

MA'AM AIMAN

SUBMITTED ON:

AUGUST 18, 2025

TABLE OF CONTENTS

TABLE OF FIGURES

Introduction

The objective of this project was to develop Cura AI, a full-stack web application designed to serve as an intelligent, AI-powered symptom checker. The primary goal was to create a tool that not only analyzes user-reported symptoms to suggest potential conditions but also provides crucial multilingual support (English and Urdu) through voice commands and text-to-speech output. The application is intended to serve as a helpful first point of reference, providing AI-generated guidance, static first-aid information, and directions to nearby medical facilities based on the user's live location.

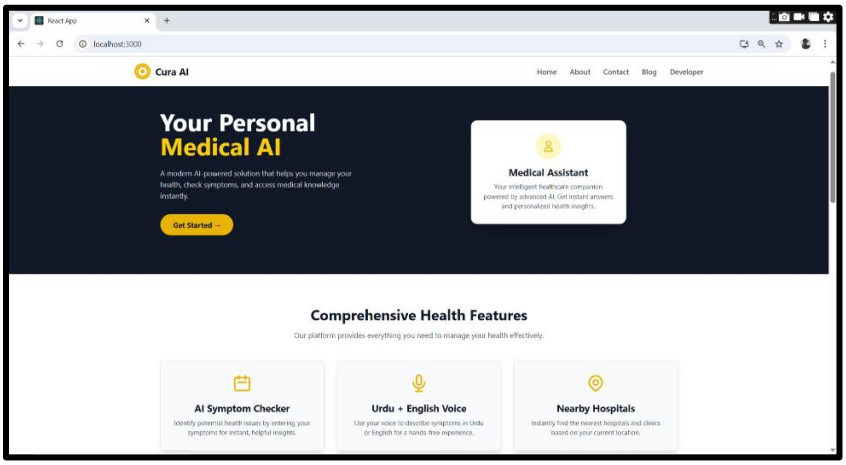

Figure 1 Homepage of the Cura AI Application

Project Architecture and Technology Stack

I designed the project with a modern MERN-stack architecture, separating the frontend and backend into distinct, independently runnable applications to ensure a clean separation of concerns.

- **Frontend:** The frontend was built with **React.js** to create a dynamic and interactive user interface. **Tailwind CSS** was chosen for styling to enable rapid, utility-first design and ensure full responsiveness. Client-side routing was managed by **React Router**.
- **Backend:** The backend was developed with **Node.js** and the **Express.js** framework to build a robust REST API. This API serves as the bridge between the frontend, the database, and the external AI services.

- **Database: MongoDB** was selected as the database, accessed via the Mongoose ODM. It was used to store and retrieve static data, such as first-aid information corresponding to various medical conditions.
- **External APIs:** The application relies on several key APIs: the **Hugging Face Inference API** for the core AI analysis, the browser's **Web Speech API** for voice recognition, and **OpenStreetMap** (via the Overpass API) for geolocation data.

Development Steps and Task Implementation

The development process was broken down into a series of logical tasks, starting with the backend services and then building the frontend to consume them.

Task 1: Backend API and Database Setup

The initial phase focused on building the server-side infrastructure. I set up a Node.js server using the Express.js framework and established the basic API routes needed for the application. Following this, I designed a simple Mongoose schema for storing first-aid instructions and connected the server to a MongoDB Atlas database. The database connection string and other sensitive keys were secured in a .env file from the beginning to prevent accidental exposure.

Task 2: AI Model Integration and Multilingual Voice Support

This was the core task of the project. I implemented several key features:

- **AI Integration:** I created a specific API endpoint on the backend that accepts a list of symptoms. This endpoint makes a secure, server-to-server request to the Hugging Face Inference API, passing the symptoms to a language model. The AI's response is then processed on my server and sent back to the frontend.
- **Voice Input:** On the frontend, I used the browser's native **Web Speech API**. I configured it to listen for voice input in both English (en-US) and Urdu (ur-PK), allowing users to state their symptoms verbally.
- **Text-to-Speech (TTS):** To provide results in both languages, I created a custom backend endpoint that takes the English text from the AI analysis, translates it to Urdu, synthesizes it into an audio file, and sends the audio back to the frontend for playback.

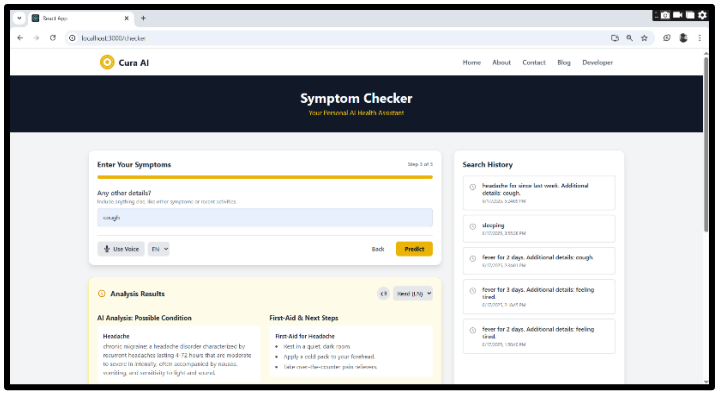

Figure 2 The Core AI Symptom Analysis Interface

Task 3: Building the React Frontend and User Interface

With the backend services in place, I built the user interface using React. I created a multi-step form for manual symptom entry as an alternative to voice input. The entire UI was styled with **Tailwind CSS**, adopting a mobile-first approach to ensure full responsiveness. To improve the user experience, I used **Framer Motion** to add subtle animations and implemented skeleton loading states to give users feedback while waiting for API responses.

Task 4: Geolocation and Mapping

To implement the "find nearby hospitals" feature, I used the browser's **Geolocation API** to get the user's current latitude and longitude. These coordinates were then used to query the OpenStreetMap database (via the Overpass API) for medical facilities. On the frontend, I used the react-leaflet library to display these locations on an interactive map, including options to filter by specialty.

Task 5: Implementing Client-Side Features

To complete the feature set, I added two important client-side functionalities:

- **Search History:** I used the idb-keyval library to interact with the browser's IndexedDB, allowing me to save a user's recent search history locally for quick re-access.
- **Error Handling:** I implemented robust error handling for scenarios like a user denying microphone or location permissions, or if an API call failed, ensuring the application would fail gracefully without crashing.

Challenges Faced and Solutions

During the project, I encountered a significant challenge related to security and version control.

- **Challenge: Secret Key Exposure:** In an early stage of development, I accidentally committed the .env file containing my secret Hugging Face API key. Even after I added .env to the .gitignore file, the key remained in the Git history. When I attempted to push my code, GitHub's Push Protection feature correctly blocked the push, warning me that I was exposing a secret.
- **Solution:** GitHub provided a special bypass link that allowed me to temporarily disable the protection for this academic project. This was a critical learning experience in understanding the importance of setting up .gitignore correctly from the very start of a project and the value of modern security features built into platforms like GitHub.

Conclusion

The Cura AI project was a comprehensive exercise in full-stack development, demonstrating the integration of a React frontend, a Node.js backend, a NoSQL database, and multiple third-party and browser-native APIs. Through this project, I have successfully demonstrated my ability to build a complex, feature-rich application that solves a real-world problem. The process of implementing advanced features like multilingual voice support and resolving critical security issues has significantly enhanced my practical skills as a web developer.

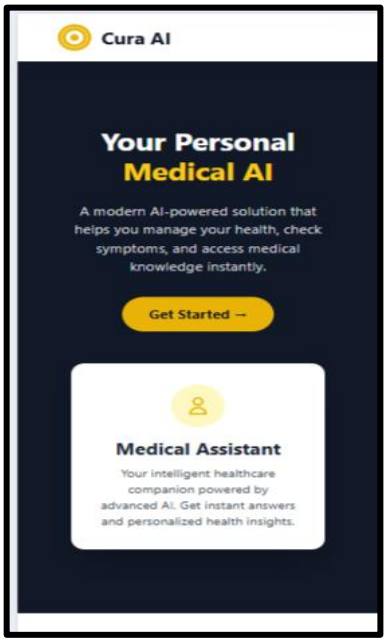

Figure 3 The Application's Responsive Mobile View