Medical Scribe: A React-Based Healthcare Assistant for Docathon

MedEngineers Team No: 15

May 18, 2025

Presented at Hackathon on May 18, 2025

Abstract

The Medical Scribe application is a modern healthcare assistant designed to streamline clinical documentation for doctors. Built using React for the frontend, Node.js with SQLite for the backend, and integrated with AI-powered APIs (Google Cloud Speech-to-Text and Hugging Face), this application offers real-time speech-to-text transcription, automated structuring and summarization of consultation notes, and efficient management of patients, appointments, and templates. Developed for a hackathon, this project demonstrates the potential of AI to reduce administrative burdens in healthcare, allowing doctors to focus on patient care.

1 Introduction

The Medical Scribe application addresses a critical challenge in healthcare: the time-consuming process of clinical documentation. Doctors often spend significant time on administrative tasks, detracting from patient care. Our solution automates this process by providing a user-friendly interface to manage patients, schedule appointments, create templates, and generate consultation notes using AI. The application leverages React for a responsive frontend, Node.js with SQLite for data persistence, and external APIs for AI functionalities like speech-to-text and natural language processing (NLP).

2 Project Overview

The Medical Scribe application is a full-stack web application designed to assist doctors in managing their daily tasks efficiently. It includes the following key features:

- **Dashboard**: Displays statistics (total patients, appointments, notes) and allows creating/deleting appointments.
- **Patients**: Enables adding, viewing, and deleting patient records.
- **AI Scribe**: Uses real-time speech-to-text (via Google Cloud Speech-to-Text API) to transcribe doctor-patient conversations, structures the notes into sections (e.g., Chief Complaint, History), and summarizes them using Hugging Face's NLP API.

- **Templates**: Allows doctors to create, view, and delete reusable templates for common consultation scenarios.

The project was developed within the constraints of a hackathon, focusing on functionality, user experience, and integration of AI technologies to demonstrate innovation in healthcare.

3 Features

3.1 Dashboard

- Displays key statistics: total patients, appointments today, saved time (static at 6 hours), and total consultation notes.
- Allows doctors to create new appointments by selecting a patient, specifying a time, and adding a purpose.
- Lists today's appointments with options to start an AI Scribe session or delete the appointment.

3.2 Patients

- Provides a form to add new patients with fields for name, gender, age, medical history, and last visit date.
- Displays a searchable table of patients with their details.
- Includes action buttons to start an AI Scribe session, view the patient profile (mocked), or delete the patient.

3.3 AI Scribe

- Enables real-time speech-to-text transcription using the Google Cloud Speech-to-Text API.
- Processes the transcribed text to structure it into sections: Chief Complaint, History of Present Illness, Physical Examination, Assessment/Diagnosis, and Plan. This uses a rule-based keyword approach on the backend.
- Summarizes the consultation note using Hugging Face's `facebook/bart-large-cnn` model via their API.
- Allows doctors to save the note to the database and view previously saved notes.

3.4 Templates

- Offers a form to create templates with a name, category, and content.
- Displays a searchable grid of templates with options to preview, use, or delete them.

4 Technology Stack

4.1 Frontend

- **React**: Used for building a dynamic and responsive user interface with components for each feature (Dashboard, Patients, AI Scribe, Templates).
- **Tailwind CSS**: Provides utility-first styling for a professional and modern UI, matching the design requirements of a healthcare application.
- **React Router**: Handles navigation between different pages (Dashboard, Patients, AI Scribe, Templates).
- **Axios**: Facilitates HTTP requests to the backend API.

4.2 Backend

- **Node.js with Express**: Powers the backend server, handling API endpoints for CRUD operations and AI processing.
- **SQLite**: Stores data for patients, appointments, templates, and consultation notes in a lightweight database.
- **Dotenv**: Manages environment variables for API keys securely.

4.3 AI Integration

- **Google Cloud Speech-to-Text API**: Converts audio recordings of doctor-patient conversations into text in real-time.
- **Hugging Face API**: Uses the `facebook/bart-large-cnn` model for summarizing consultation notes.
- **Axios**: Used in the backend to make API calls to Google Cloud and Hugging Face.

5 Implementation Details

5.1 Directory Structure

The project follows a clear structure to separate frontend and backend concerns:

- `medical-scribe/`
  - `backend/`
    - `server.js`: Backend logic with API endpoints.
    - `database.db`: SQLite database.
    - `package.json`: Backend dependencies.
    - `.env`: Stores API keys.
  - `frontend/`
    - `src/`

        - `App.js`: Main React component with routing.
        - `components/`: Contains React components (`Sidebar.js`, `Dashboard.js`, `Patients.js`, `AIScribe.js`, `Templates.js`).
        - `index.js`: Entry point for React.
        - `index.css`: Global styles with Tailwind.
    - `package.json`: Frontend dependencies.
    - `tailwind.config.js`: Tailwind configuration.
    - `postcss.config.js`: PostCSS configuration.

5.2 Backend Implementation

The backend (`server.js`) is built using Express and SQLite. It includes:

- **Database Setup**: Creates tables for patients, appointments, templates, and consultation notes with initial seed data.
- **API Endpoints**:
  - `GET /api/patients`: Retrieves all patients.
  - `POST /api/patients`: Adds a new patient.
  - `DELETE /api/patients/:id`: Deletes a patient.
  - `GET /api/appointments`, `POST /api/appointments`, `DELETE /api/appointments/:id`: Manages appointments.
  - `GET /api/templates`, `POST /api/templates`, `DELETE /api/templates/:id`: Manages templates.
  - `POST /api/speech-to-text`: Converts audio to text using Google Cloud Speech-to-Text API.
  - `POST /api/nlp`: Structures and summarizes text using a rule-based approach and Hugging Face API.
  - `POST /api/scribe`, `GET /api/consultation_notes`: Manages consultation notes.
- **AI Integration**: Uses `@google-cloud/speech` for speech-to-text and `axios` to call Hugging Face's API for summarization.

5.3 Frontend Implementation

The frontend is built with React and structured into reusable components:

- **Sidebar**: A navigation component with links to Dashboard, Patients, AI Scribe, and Templates.
- **Dashboard**: Displays stats and allows appointment management.

- **Patients**: Implements a form for adding patients and a table for viewing/deleting them.
- **AI Scribe**: Uses the browser's `MediaRecorder` API to record audio, sends it to the backend for transcription, processes the text for structuring and summarization, and saves the note.
- **Templates**: Provides a form to create templates and a grid to view/delete them.

5.4 AI Scribe Workflow

1. Doctor selects a patient and clicks "Start Recording."
2. Audio is recorded using `MediaRecorder` and sent as a base64-encoded string to the backend.
3. Backend calls Google Cloud Speech-to-Text API to transcribe the audio.
4. Transcribed text is returned to the frontend and displayed in a textarea.
5. Doctor clicks "Process" to send the text to the backend's `/api/nlp` endpoint.
6. Backend structures the text using keyword matching (e.g., "presents with" for Chief Complaint) and summarizes it using Hugging Face's API.
7. Structured note is displayed in the frontend, and the doctor can save it to the database.

6 Challenges Faced

- **API Integration**: Setting up Google Cloud Speech-to-Text required configuring the API key and handling audio encoding (WEBM_OPUS). Rate limits and quota management were also concerns.
- **NLP Structuring**: The rule-based approach for structuring notes (e.g., Chief Complaint, History) is limited and may miss nuanced medical terms. A medical-specific NLP model would be ideal but wasn't feasible within hackathon constraints.
- **Real-Time Audio Recording**: Ensuring compatibility across browsers for `MediaRecorder` and handling microphone permissions was challenging.
- **Time Constraints**: Balancing feature development, UI design, and API integration within the hackathon timeline required prioritization.

7 Future Improvements

- **Advanced NLP**: Integrate a medical-specific NLP model (e.g., ClinicalBERT) for better structuring of consultation notes.
- **User Authentication**: Add login functionality to secure the application and manage multiple doctors.
- **Export Functionality**: Allow doctors to export consultation notes as PDFs.

- **Scalability**: Replace SQLite with a more robust database like PostgreSQL for production use.
- **Mobile Support**: Optimize the UI for mobile devices with a responsive design.

8 Conclusion

The Medical Scribe application demonstrates the power of AI in healthcare by automating clinical documentation. By integrating real-time speech-to-text and NLP capabilities, it reduces the administrative burden on doctors, allowing them to focus on patient care. Built using React, Node.js, and external APIs, this project showcases a practical solution developed within the constraints of a hackathon. With future improvements, it has the potential to become a valuable tool in clinical settings.

9 Team Contributions

- **Team Member 1**: Designed the frontend UI and implemented React components.
- **Team Member 2**: Set up the backend with Express and SQLite, integrated Google Cloud Speech-to-Text API.
- **Team Member 3**: Implemented the AI Scribe feature, including audio recording and Hugging Face API integration.
- **Team Member 4**: Tested the application, documented the project, and prepared the presentation.