Danielle Shokrian

Medical Office Staff Scheduler - Project Pitch

Section 1: Scheduling Problem

Medical office scheduling is a time-consuming, error-prone task that drains administrative productivity. Office managers spend **5-10 hours per week** manually coordinating staff schedules across spreadsheets, paper calendars, or disjointed systems. This manual approach leads to scheduling conflicts, coverage gaps, and staff frustration.

Target Users

Primary User: Office Administrators/Practice Managers

- Juggle multiple staff schedules across different roles (doctors, nurses, techs)
- Manually check for conflicts by cross-referencing multiple sources
- Scramble to fill last-minute gaps when someone calls in sick or requests PTO
- Lose track of time-off requests buried in emails or voicemails

Secondary Users: Medical Staff (Doctors, Nurses, Technicians)

- Lack visibility into their own schedules without asking the office manager
- Submit time-off requests via email/text that sometimes get overlooked
- Miss opportunities to pick up extra shifts because they don't know what's available
- Experience scheduling conflicts that could have been prevented

Solution

This application centralizes scheduling into one organized place, transforming a scattered, manual process into an efficient, automated workflow.

AI-Powered Optimization: The app integrates OpenAI API to generate intelligent shift assignment suggestions

As an office administrator, I want to:

- Create a weekly schedule by assigning staff to shifts, so that I can coordinate coverage across all roles
- See visual warnings when I try to schedule someone who's already working, so that I avoid double-booking conflicts
- View a list of all open shifts with AI-generated staffing suggestions, so that I can quickly fill gaps with qualified staff
- Approve or deny time-off requests in one place, so that I don't lose track of who's out when

As a medical staff member, I want to:

- View my upcoming shifts in a calendar, so that I don't have to ask the manager when I'm working
- Submit time-off requests digitally and see their approval status, so that I can plan PTO confidently
- See available shifts I'm qualified for, so that I can pick up extra hours when I want them

---

Section 2: Problem-Solving Process

Step-by-Step Development Process

Step 1: Planning

- Define functional requirements from user stories
- Sketch outline for key user interfaces (schedule calendar, staff list, time-off form)
- Map out RESTful API endpoints for all CRUD operations
- Select technology stack and set up development environment

Step 2: Design & Implementation

- Create SQLAlchemy models with relationships
- Implement Flask API routes for all CRUD operations
- Add error handling for edge cases (double-booking, invalid dates)
- Test API endpoints using Postman

Step 3: Implementation

- Design prompts for OpenAI API to generate schedule suggestions
- Create backend endpoint that calls OpenAI and processes responses
- Build logic to accept/reject suggestions and update actual schedule

Step 4: Implementation

- Build React component structure and routing
- Implement forms for creating/editing staff, shifts, and time-off requests
- Create calendar view to visualize weekly schedule
- Connect frontend to backend API using fetch
- Display conflict warnings and AI suggestions to users

Step 5: Testing

- Test all CRUD operations for each resource
- Validate conflict detection catches scheduling errors
- Verify AI suggestions are stored and displayed correctly

- Test error handling with invalid inputs

Step 6: Deployment & Maintenance

- Write README with setup instructions and feature overview
- Document API endpoints and data models
- Create user guide or demo script
- Deploy to Render
- Prepare demo presentation highlighting key features

Outline

1. Admin logs in → Dashboard showing weekly schedule overview

2. Admin clicks "Add Staff" → Form to create new staff member

3. Admin creates shifts → Selects staff, date, time → System checks for conflicts

├─ If conflict: Display error, prevent creation

└─ If valid: Save to database, update calendar view

4. Staff submits time-off request → Appears in admin's approval queue

5. Admin reviews requests → Approve/Deny → Staff sees updated status

6. Admin views open shifts → Clicks "Get AI Suggestions"

7. System calls OpenAI → Generates staffing recommendations → Stores in DB

8. Admin reviews suggestions → Accepts one → Creates shift automatically

Tools & Technologies

Backend Stack:

- Flask, SQLAlchemy, PostgreSQL, Flask-CORS, OpenAI API

Frontend Stack:

- React, React Router, useState/useEffect, react-calendar, CSS/Tailwind

Development Tools:

- Git/GitHub, Postman, VS Code

Section 3: Timeline and Scope

Development Timeline (7 Days, ~50-60 Hours)

Days 1-2: Planning & Backend (16 hours)

- Finalize data models and wireframes
- Set up Flask, PostgreSQL, and SQLAlchemy
- Create all four models (Staff, Shift, TimeOffRequest, AISuggestion)
- Build all CRUD API routes
- Implement conflict detection logic
- Test endpoints with Postman

Days 3-4: AI Integration & Core Backend Features (16 hours)

- Integrate OpenAI API for schedule suggestions
- Build endpoint to generate and store AI responses
- Add validation and error handling
- Complete backend testing
- Begin React setup and basic routing

Days 5-6: Frontend Development (16 hours)

- Build all React components (StaffList, ScheduleCalendar, TimeOffManager, AISuggestions)
- Implement all CRUD operations in UI
- Connect frontend to backend via Axios
- Add calendar view and conflict warnings
- Style with CSS/Tailwind for clean, professional look

Day 7: Testing, Documentation & Demo Prep (8-12 hours)

- End-to-end testing of all features
- Bug fixes and edge case handling
- Write README with setup instructions and screenshots
- Polish UI and practice demo presentation