🔗 MediHelp

---

1. Introduction

1.1 Purpose

This document outlines the complete Site Reliability Engineering (SRE) plan and API specifications for the MediHelp+ project. It unifies backend service requirements, frontend reliability plans, API endpoints, third-party integrations, and monitoring strategies. It ensures that the MediHelp+ platform remains highly available, reliable, secure, and scalable for end users.

1.2 Document Conventions

Following IEEE SRS and SRE best practices, the document is divided into sections for functional and non-functional requirements, system features, SRE strategies, and constraints.

1.3 Intended Audience

- Backend and frontend developers
- Hackathon organizers and evaluators

1.4 Scope

Covers:

- Full backend API and database structure
- Frontend user interface responsibilities
- Monitoring (backend, frontend, AI services)
- Auto-recovery mechanisms

- Alerts and real-time incident response
- Security and performance targets

---

2. Overall Description

2.1 Product Perspective

MediHelp+ backend is a RESTful Django 5.2 system with Django REST Framework (DRF 3.16), JWT authentication (SimpleJWT), PostgreSQL database (for production), and auto-generated OpenAPI documentation using drf-spectacular. Frontend is built with ReactJS using TailwindCSS for UI and Sentry/Web Vitals for monitoring.

2.2 Product Functions

- **User Authentication:** Registration, login, password management
- **Symptom Checker:** Submit symptoms, AI-based response (using Gemini API)
- **Symptom Check History:** View past health checks
- **First Aid & Home Remedies:** Searchable database
- **Educational Content:** Health articles and videos
- **Monitoring and Alerts:** Automated uptime checks, AI API health checking
- **Recovery:** Auto-restart backend, frontend fallback modals

2.3 User Characteristics

- General users seeking quick health support
- Admin users managing content and user records
- Developers maintaining reliability

2.4 Constraints

- Django 5.2, DRF 3.16, PostgreSQL production database
- JWT-based authentication
- HTTPS enforced on all endpoints
- OpenAPI 3 documentation via drf-spectacular
- Google Gemini API coupon usage for AI tasks
- Map-related emergency services temporarily delayed

---

3. Specific Requirements

3.1 Backend API Specifications

3.1.1 Authentication APIs

Endpoint	Method	Purpose
/api/auth/register/	POST	Register a new user
/api/auth/login/	POST	Login and get JWT tokens
/api/auth/logout/	POST	Logout and invalidate token
/api/auth/token/refresh/	POST	Refresh JWT access token

3.1.2 Symptom Checker APIs

Endpoint	Method	Purpose
/api/symptoms/check/	POST	Submit symptoms, get AI diagnosis
/api/symptoms/history/	GET	View past symptom checks
/api/symptoms/history/{id}/	GET	View specific symptom record

3.1.3 First Aid and Home Remedies APIs

Endpoint	Method	Purpose
/api/firstaid/	GET	List available first-aid topics/remedies
/api/firstaid/{id}/	GET	Get detailed first-aid info
/api/firstaid/?q=keyword	GET	Search/filter remedies

3.1.4 Educational Content APIs

Endpoint	Method	Purpose
/api/content/articles/	GET	List educational articles
/api/content/articles/{id}/	GET	Get article details

/api/content/videos/	GET	List educational videos
/api/content/videos/{id}/	GET	Get video details
/api/content/articles/ (admin)	POST/PUT/DELETE	Manage articles

3.1.5 Documentation APIs

Endpoint	Method	Purpose
/api/schema/	GET	Get OpenAPI schema
/api/docs/	GET	SwaggerUI or Redoc API explorer

3.1.6 Monitoring APIs

Endpoint	Method	Purpose
/api/healthz/	GET	Health check endpoint for Uptime Kuma

3.2 Frontend Development Responsibilities

Components to Build Immediately (Parallel to Backend)

Component	Purpose	When to Start	Backend Dependency

Authentication Pages (Login/Register)	Allow user login, signup	Day 1	Needs /api/auth/register/ and /api/auth/login/
Symptom Checker UI	Form to submit symptoms, view AI results	Day 1–2	Needs /api/symptoms/check/
Symptom History UI	Show list of previous symptom checks	Day 2–3	Needs /api/symptoms/history/
First Aid Search Page	Browse/search first-aid instructions	Day 2–4	Needs /api/firstaid/
Educational Content List	Show health articles and videos	Day 3–4	Needs /api/content/articles/ and /api/content/videos/
Error Handling & Recovery Modals	Inform users of failures (network, backend)	Day 1	Can be done independently
Web Vitals + Sentry Integration	Monitor frontend performance and crashes	Day 1	No backend dependency

Temporary Delays

- Map & Nearby Hospitals feature is **delayed**.
- Emergency location tracking **delayed** until after main features are working.

Frontend Waiting Strategy

Task	Wait for Backend API Completion?
Authentication UI	Yes, wait for register/login APIs
Symptom Checker UI	Can start form design, wait for symptom check API
First Aid UI	Can design list layout, wait for /api/firstaid/
Educational Content UI	Can design early based on dummy data
Monitoring (Sentry/Web Vitals)	No need to wait

---

4. SRE Strategies (Reliability, Monitoring, Recovery)

4.1 Backend Monitoring

Metric	Tool	Threshold	Action
API Health (/api/healthz)	Uptime Kuma	2 failed checks	Discord alert
PostgreSQL Availability	`pg_isready` cron	No response 2 min	Discord alert

API Error Rate (500s)	Django logs	>5% requests	Discord alert
AI API Latency (Gemini)	Django middleware	>2s response time	Log warning
AI API Failures	Django middleware	>1 failure/min	Discord alert

4.2 Frontend Monitoring

Metric	Tool	Threshold	Action
Javascript Errors	Sentry	Any critical error	Discord alert
LCP (Largest Contentful Paint)	Web Vitals	>2.5s	Log warning
CLS (Cumulative Layout Shift)	Web Vitals	>0.1	Log warning
FID (First Input Delay)	Web Vitals	>100ms	Log warning

4.3 Recovery and Automation

- PM2 (or systemd) auto-restarts Django backend on crash.
- Healthchecks.io bots for scheduled pings and downtime alerts.
- Frontend crash modals suggest users refresh instead of breaking.

4.4 AI API Handling

- Google Gemini API is used for AI functions
- Fallback message shown if Gemini API is unavailable.
- Errors logged separately for AI API monitoring.

---

5. Data Models Overview

- CustomUser: User authentication and roles.
- Symptom: User-reported symptoms.
- Condition: Medical conditions mapped to symptoms.
- Severity: Severity rating system.
- SymptomCheck: Records of symptom submissions.
- FirstAidInstruction: First-aid procedure database.
- HomeRemedy: Home-based treatment entries.
- Article: Health educational articles.
- Video: Health educational videos.

---

6. Technology Stack

- Backend: Django 5.2 + Django REST Framework 3.16
- Authentication: JWT (SimpleJWT)
- Database: SQLite (development) and PostgreSQL (production)

- Frontend: ReactJS + TailwindCSS
- Monitoring: Uptime Kuma, Sentry, Healthchecks.io, PM2
- AI APIs: Google Gemini APIs
- Environment: django-environ for configuration

✅ Final Notes

- APIs must be RESTful, returning JSON with standard HTTP status codes.
- Frontend must handle API failures gracefully and stay user-friendly.
- System must maintain ≥99% uptime during hackathon.
- Security of JWT tokens, password storage, and user health data is critical.
- Full OpenAPI 3 documentation must be live and updated.
- Emergency maps and hospital location services are temporarily delayed for post-hackathon versions.
