AI Medical Assistant

Design Document Submission

⚠ **IMPORTANT: The GITHUB repo should JUST have the PDF for the Design Document Submission.**

Design Document

The complete design document is available in `DESIGN_DOCUMENT.md`.

To create the PDF submission:

1. Open `DESIGN_DOCUMENT.md` in your preferred markdown editor
2. Export/Print to PDF format
3. Name the file: `AI_Medical_Assistant_Design_Document.pdf`
4. Upload the PDF to this repository

Project Overview

An AI-powered medical assistant application that provides intelligent health consultations, symptom tracking, appointment management, and access to medical resources. Built with modern web technologies and integrated with AI capabilities for natural language health interactions.

This comprehensive platform combines cutting-edge AI technology with medical information management to provide users with:

- **Intelligent Medical Conversations**: Real-time AI-powered health consultations using Google Gemini and OpenAI models
- **Complete Health Management**: Track symptoms, medications, appointments, and medical history
- **Hospital Intelligence**: Advanced web scraping system that collects and indexes hospital information
- **Doctor Directory**: Find and review healthcare providers
- **Emergency Support**: Critical symptom detection with emergency guidance

Project URL: https://lovable.dev/projects/f7e3ea07-2a44-4095-a4e6-da0fd18ff9d3

Documentation:

- ARCHITECTURE.md - System architecture and technical design
- SETUP_GUIDE.md - Complete setup and configuration guide
- API_DOCUMENTATION.md - API reference and examples
- DESIGN_DOCUMENT.md - Full project design document
- TESTING_GUIDE.md - Testing strategy and implementation

Features

Core Functionality

- **AI Health Chat**: Natural language conversations about health concerns with streaming responses using Lovable AI Gateway
- **Symptom Tracking**: Log and monitor symptoms over time with severity indicators and temperature tracking
- **Appointment Management**: Schedule and track medical appointments with recurring appointments and email reminders
- **Medical History**: Comprehensive health profile including:
    - Chronic conditions with severity tracking
    - Current and historical medications
    - Allergies with reaction severity
    - Family medical history
- **Doctor Directory**: Browse, search, and review healthcare providers by specialty
- **Hospital Management System**:
    - Add and manage hospital information
    - **Advanced Web Scraping**: Three-tier scraping system:
        1. **Advanced Native Scraper** (Free) - Deep crawling up to 100 pages with metadata extraction
        2. **Firecrawl Integration** - Cloud-based scraping with JavaScript rendering
        3. **ScraperAPI Integration** - Proxy rotation and CAPTCHA handling
    - Bulk scraping with progress tracking
    - Visual indicators for scraped vs unscraped hospitals
    - API key testing before save
    - Automatic page type classification (services, contact, emergency, etc.)
- **Health Resources**: Access curated medical articles and information
- **Emergency Guidance**: Critical symptoms checker with emergency contact management and protocols
- **Profile Management**: Secure user profiles with avatar upload to Supabase Storage

Technical Features

- Real-time AI chat with streaming responses
- User authentication and authorization
- Responsive design (mobile, tablet, desktop)
- Dark/light mode support
- Secure data storage with Row Level Security (RLS)
- RESTful API integration
- Edge functions for AI processing

Tech Stack

Frontend

- **React 18.3** - UI framework
- **TypeScript** - Type-safe development
- **Vite** - Build tool and dev server
- **Tailwind CSS** - Utility-first styling
- **Shadcn/ui** - Component library

- **React Router** - Client-side routing
- **React Query** - Server state management
- **Lucide React** - Icon library
- **Sonner** - Toast notifications

Backend (Lovable Cloud)

- **Supabase** - Backend as a Service
  - PostgreSQL database
  - Authentication
  - Row Level Security (RLS)
  - Edge Functions
  - Real-time subscriptions
- **Deno** - Edge function runtime

AI Integration

- **Lovable AI Gateway** - Pre-configured AI access (no API key required)
- **Supported Models**:
  - google/gemini-2.5-flash (default) - Balanced performance and cost
  - google/gemini-2.5-pro - Advanced reasoning for complex medical queries
  - google/gemini-3-pro-preview - Next-generation model
  - google/gemini-2.5-flash-lite - Fastest for simple queries
  - openai/gpt-5 - OpenAI's flagship model
  - openai/gpt-5-mini - Cost-effective OpenAI
  - openai/gpt-5-nano - High-speed OpenAI
- **Features**:
  - Streaming responses (token-by-token rendering)
  - Context-aware conversations
  - Medical knowledge base
  - Structured data extraction via tool calling
  - Rate limiting and credit management

Hospital Scraping System

- **Three-Tier Scraping Approach**:
  1. **Advanced Native Scraper** (Built-in, Free):
     - No API key required
     - Recursive crawling (max 100 pages, depth 3)
     - Intelligent content extraction with improved selectors
     - Metadata extraction (emails, phones, JSON-LD structured data)
     - Page type classification (emergency, services, doctors, contact, etc.)
     - Contact information detection from page content
  2. **Firecrawl API** (Recommended):
     - Superior scraping accuracy
     - JavaScript rendering support
     - Better handling of dynamic content

        - API key from https://firecrawl.dev
    3. **ScraperAPI** (Alternative):
        - Proxy rotation and CAPTCHA solving
        - Reliable for blocked websites
        - API key from https://scraperapi.com
- **Bulk Scraping**: Select multiple hospitals and scrape in batch with progress tracking
- **API Key Testing**: Verify API keys before saving
- **Visual Indicators**: Green "Scraped", Amber "Not Scraped", Blue "Auto" badges
- **Analytics**: Track scraping statistics, success rates, and methods used

Requirements

System Requirements

- Node.js 18+ or Bun runtime
- Modern web browser (Chrome, Firefox, Safari, Edge)
- Internet connection

Environment Variables

The following environment variables are automatically configured via Lovable Cloud:

```
VITE_SUPABASE_URL=your-supabase-url
VITE_SUPABASE_PUBLISHABLE_KEY=your-anon-key
VITE_SUPABASE_PROJECT_ID=your-project-id
```

API Keys (Optional)

For enhanced AI features, you can add:

- OpenAI API Key (optional, Lovable AI is default)
- Google AI API Key (optional)

Installation & Setup

1. Clone the Repository

```
git clone [repository-url]
cd ai-medical-assistant
```

2. Install Dependencies

```
npm install
# or
bun install
```

3. Environment Configuration

The `.env` file is automatically configured when using Lovable Cloud. If self-hosting:

- Copy `.env.example` to `.env`
- Fill in your Supabase credentials

4. Database Setup

Database migrations are automatically applied via Lovable Cloud. The schema includes:

- `profiles` - User profiles
- `appointments` - Medical appointments
- `medications` - Medication tracking
- `allergies` - Allergy records
- `chronic_conditions` - Long-term health conditions
- `family_history` - Family medical history
- `doctors` - Healthcare provider directory
- `hospitals` - Hospital information
- `hospital_pages` - Scraped hospital data
- `doctor_reviews` - Provider ratings and reviews

5. Start Development Server

```
npm run dev
# or
bun run dev
```

The application will be available at `http://localhost:5173`

Compilation & Build (Production)

When you're ready to prepare the app for production you should compile (build) it. Compiling produces an optimized output (the `dist/` folder) that's smaller, faster, and ready to deploy to a static host or CDN.

- Install dependencies (if not already installed):

```
npm install
# or
bun install
```

- Build the production bundle using Vite:

```
npm run build
# or
bun run build
```

- Preview the production build locally (serves the contents of `dist/`):

```
npm run preview
# or
bun run preview
```

What the compile step does and why it matters:

- Bundling & Code Splitting: Combines modules into efficient bundles and separates code for lazy-loaded routes to reduce initial download size.
- Minification & Compression: Removes whitespace, shortens identifiers and applies other optimizations so assets are smaller.
- Tree Shaking: Eliminates unused code from your bundles, lowering payload size.
- Transpilation & Compatibility: Converts modern JS/TS features to work across target browsers (if configured), improving compatibility.
- Hashing & Cacheability: Emits hashed filenames for long-term caching and better cache invalidation when files change.
- Removes Dev-Only Code: Dev helpers, warnings and source maps (unless enabled) are not included by default, improving runtime speed and reducing exposure of internal details.
- Deterministic Production Output: Produces a predictable `dist/` folder you can deploy to hosting providers (Vercel, Netlify, Cloudflare Pages, S3, etc.).

When to use `dev` vs `build`/`preview`:

- `npm run dev` / `bun run dev`: Fast developer server with hot-reload for active development.
- `npm run build`: Produces the production-ready assets — use this before deploying.
- `npm run preview`: Serve the compiled `dist/` locally to verify the production build behaviour (routing, asset loading, environment differences).

Notes for TypeScript projects:

- The Vite build will include compiled TypeScript output; however, if you want a separate type-check step use `tsc --noEmit` or a CI job to enforce type safety before publishing.

Development

Project Structure

```
├── src/
│   ├── components/        # React components
│   │   ├── ui/           # Shadcn UI components
```

```
│   │   └── ...           # Feature components
│   ├── pages/            # Route pages
│   ├── contexts/         # React contexts
│   ├── hooks/            # Custom hooks
│   ├── integrations/     # External integrations
│   │   └── supabase/    # Supabase client & types
│   ├── data/            # Static data
│   └── lib/             # Utility functions
├── supabase/
│   ├── functions/        # Edge functions
│   └── config.toml       # Supabase configuration
└── public/              # Static assets
```

Available Scripts

- `npm run dev` - Start development server
- `npm run build` - Build for production
- `npm run preview` - Preview production build
- `npm run lint` - Run ESLint

Edge Functions

Located in `supabase/functions/`:

- `chat` - AI chat functionality
- `hospital-assistant` - Hospital information queries
- `scrape-hospital` - Hospital data scraping

Edge functions are automatically deployed via Lovable Cloud.

Authentication

The app uses Supabase Auth with email/password authentication:

- Email auto-confirmation enabled for development
- Row Level Security (RLS) policies enforce data access
- Protected routes require authentication
- Session management with automatic refresh

Database Security

Row Level Security (RLS)

All tables have RLS policies ensuring:

- Users can only access their own data
- Public data (doctors, hospitals) is readable by all
- Write operations require authentication
- Admin operations are restricted

Data Privacy

- HIPAA compliance considerations
- Encrypted data at rest
- Secure API communications (HTTPS)
- No sensitive data in client-side storage

Testing

Test Setup

The project uses **Vitest** for unit testing and **React Testing Library** for component testing.

For comprehensive testing documentation, see TESTING_GUIDE.md

Test files are located alongside their source files in `__tests__` directories:

```
src/
├── components/
│   ├── __tests__/
│   │   ├── ChatMessage.test.tsx
│   │   └── QuickResponseButtons.test.tsx
│   └── ui/
│       └── __tests__/
│           └── button.test.tsx
└── lib/
    └── __tests__/
        └── utils.test.ts
```

Running Tests

```
# Run all tests
npm test

# Run tests in watch mode
npm run test:watch

# Run tests with coverage
npm run test:coverage

# Run tests with UI
npm run test:ui
```

Test Scripts Configuration

Add these scripts to your `package.json`:

```
{
  "scripts": {
    "test": "vitest run",
    "test:watch": "vitest",
    "test:ui": "vitest --ui",
    "test:coverage": "vitest run --coverage",
    "test:integration": "vitest run src/test/integration",
    "test:e2e": "playwright test",
    "test:e2e:ui": "playwright test --ui",
    "test:visual": "playwright test e2e/visual-regression.spec.ts"
  }
}
```

Writing Tests

Example test structure:

```
import { describe, it, expect, vi } from 'vitest';
import { render } from '@/test/utils';
import { screen } from '@testing-library/dom';
import userEvent from '@testing-library/user-event';
import { YourComponent } from '../YourComponent';

describe('YourComponent', () => {
  it('renders correctly', () => {
    render(<YourComponent />);
    expect(screen.getByText('Expected Text')).toBeInTheDocument();
  });

  it('handles user interactions', async () => {
    const handleClick = vi.fn();
    const user = userEvent.setup();

    render(<YourComponent onClick={handleClick} />);
    await user.click(screen.getByRole('button'));

    expect(handleClick).toHaveBeenCalledTimes(1);
  });
});
```

Integration Tests

Integration tests use **MSW (Mock Service Worker)** to mock API endpoints:

Location: `src/test/integration/`

Mock handlers are defined in `src/test/mocks/handlers.ts` and include:

- Supabase Auth endpoints

- Chat edge function
- Database queries (appointments, doctors, hospitals)

Example integration test:

```
import { describe, it, expect, beforeAll, afterAll, afterEach } from 'vitest';
import { server } from '../mocks/server';

describe('Chat Integration Tests', () => {
  beforeAll(() => server.listen());
  afterEach(() => server.resetHandlers());
  afterAll(() => server.close());

  it('sends a message and receives a response', async () => {
    // Test implementation
  });
});
```

End-to-End Tests

E2E tests use **Playwright** to test complete user flows:

Location: `e2e/`

Test categories:

- `auth.spec.ts` - Authentication flows
- `navigation.spec.ts` - Navigation and routing
- `emergency.spec.ts` - Emergency page functionality
- `visual-regression.spec.ts` - Visual regression testing with Percy

Running E2E tests:

```
# Install Playwright browsers (first time only)
npx playwright install

# Run all E2E tests
npx playwright test

# Run E2E tests in UI mode
npx playwright test --ui

# Run specific test file
npx playwright test e2e/auth.spec.ts

# Run tests in specific browser
npx playwright test --project=chromium
```

Coverage Thresholds

Coverage thresholds are enforced in `vitest.config.ts`:

```
coverage: {
  thresholds: {
    lines: 80,
    functions: 80,
    branches: 80,
    statements: 80,
  },
}
```

Tests will fail if coverage drops below 80% for any metric.

Visual Regression Testing

Visual regression testing uses **Percy** to catch unintended UI changes:

Setup:

1. Sign up at percy.io
2. Get your `PERCY_TOKEN`
3. Add to environment variables or CI/CD secrets

Percy snapshots are automatically captured during E2E tests:

```
import percySnapshot from '@percy/playwright';

await percySnapshot(page, 'Page Name - Desktop');
```

Configuration in `.percy.yml`:

- Multiple viewport widths (mobile, tablet, desktop)
- CSS to hide dynamic content
- Minimum height settings

Test Utilities

Custom render function with providers (`src/test/utils.tsx`):

- Wraps components with React Router
- Includes React Query provider
- Provides consistent test environment

CI/CD Integration

GitHub Actions workflows:

Unit & Integration Tests (`.github/workflows/test.yml`):

- Tests on Node.js 18.x and 20.x
- Linting checks
- Build verification
- Coverage reports with thresholds enforcement
- Optional Codecov integration

E2E Tests (`.github/workflows/e2e.yml`):

- Playwright tests across multiple browsers
- Percy visual regression tests
- Automatic artifact upload for test results and reports

Both workflows trigger on:

- Push to `main` or `develop` branches
- Pull requests to `main` or `develop` branches

Required Secrets for CI/CD:

- `CODECOV_TOKEN` (optional) - For coverage reporting
- `PERCY_TOKEN` (optional) - For visual regression testing

Deployment

Via Lovable

The app is automatically deployed via Lovable's hosting platform.

Self-Hosting

1. Build the application:

```
npm run build
```

2. Deploy the `dist` folder to your hosting provider
3. Configure environment variables
4. Set up Supabase project separately
5. Deploy edge functions to Supabase

Supported Platforms

- Vercel
- Netlify
- Cloudflare Pages
- AWS Amplify
- Traditional web hosting

Testing

Manual Testing Checklist

- [ ] User registration and login
- [ ] AI chat functionality
- [ ] Symptom logging
- [ ] Appointment creation/editing
- [ ] Medical history CRUD operations
- [ ] Doctor search and reviews
- [ ] Hospital finder
- [ ] Responsive design on mobile/tablet
- [ ] Dark/light mode switching

Automated Testing

Currently, the project uses manual testing. Future enhancements include:

- Unit tests with Vitest
- Component tests with React Testing Library
- E2E tests with Playwright

Browser Support

- Chrome/Edge (latest 2 versions)
- Firefox (latest 2 versions)
- Safari (latest 2 versions)
- Mobile browsers (iOS Safari, Chrome Mobile)

Performance

- Lazy loading for routes
- Code splitting
- Optimized images
- Streaming AI responses
- Efficient database queries with indexes

Accessibility

- Semantic HTML
- ARIA labels
- Keyboard navigation
- Screen reader support
- Color contrast compliance (WCAG 2.1 AA)

Contributing

1. Fork the repository

2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

License

Disclaimer

⚠ **Medical Disclaimer**: This application is for educational and informational purposes only. It is not intended to be a substitute for professional medical advice, diagnosis, or treatment. Always seek the advice of your physician or other qualified health provider with any questions you may have regarding a medical condition.

Support

For issues and questions:

- Create an issue in the GitHub repository
- Contact via Lovable platform
- Review documentation in `DESIGN_DOCUMENT.md`

Acknowledgments

- Built with Lovable
- UI components from Shadcn/ui
- Icons from Lucide
- Backend powered by Supabase
- AI capabilities via Google Gemini & OpenAI

Version History

See the commit history for detailed version information.

Last Updated: November 2025
Status: Active Development