Patients Queuing System Design

## **✅ Step 1: Requirement Analysis Confirmation (Finalized)**

### **🎯 Confirmed Functional Requirements**

These are system capabilities and features the app must deliver:

1. **Patient Registration & Booking**
   1. Patients can register on-site via a kiosk or online (web/mobile).
   2. Appointment booking is available for pre-scheduling consultations.
2. **Queue Number Assignment**
   1. Automatically generated based on:
      1. Appointment time.
      2. Urgency level.
      3. Patient category (e.g., children, elderly, emergency).
3. **Real-time Queue Monitoring**
   1. Patients view their position in the queue (mobile app, kiosk display).
   2. Doctors and receptionists monitor patient flow and statuses.
4. **Notification System**
   1. SMS and push notifications to:
      1. Confirm registration.
      2. Notify of delays.
      3. Alert when the patient’s turn is near.
5. **Role-based Interfaces**
   1. **Patients**: Booking, queue tracking, notifications, feedback.
   2. **Receptionists**: Registration, queue editing, manual override.
   3. **Doctors**: Queue view, patient info, prioritization options.
   4. **Admins**: Full access, performance reports, and user controls.
6. **Queue Prioritization Logic**
   1. Based on:
      1. **Medical condition severity** (selected by staff during registration).
      2. **Patient type** (emergency, elderly, child, etc.).
7. **Analytics & Reporting**
   1. Metrics include:
      1. Daily patient volume.
      2. Average wait time.
      3. Peak consultation hours.
      4. Dropouts and no-shows.
8. **Mobile App Support**
   1. Built with React-native or React Native.
   2. Includes real-time tracking and multilingual interface.
9. **Help and Guidance Features**
   1. Step-by-step **tutorials** for app usage.
   2. **Help pop-ups** integrated into the UI.
   3. Video/audio assistance for non-literate users (future scope).

### **🚧 Confirmed Non-Functional Requirements**

These govern system performance and quality:

1. **Usability**
   1. Simple, clean UI.
   2. Optimized for both tech-savvy and low-literacy users.
2. **Performance**
   1. Must handle hundreds of concurrent users with <2s response time.
3. **Reliability**
   1. High uptime during working hours.
   2. Error handling for registration and network failures.
4. **Security**
   1. Authentication: Role-based login (patients, doctors, staff).
   2. Data protection: Encryption at rest and in transit.
5. **Scalability**
   1. Designed to expand from OPD to other hospital departments or facilities.
6. **Localization & Language Support**
   1. Primary Language: **Swahili**.
   2. Future support for English toggle option.
7. **Offline Fallback**
   1. Implement **local caching**:
      1. Store recent queue and registration data locally.
      2. Sync with central server when online.

### **📌 Stakeholder Needs Mapping**

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Needs** | **System Feature** |
| **Patients** | Easy access, clear instructions, low wait time | Swahili UI, help popups, real-time updates |
| **Doctors** | See who’s next, prioritize urgent cases | Doctor dashboard, queue detail filters |
| **Receptionists** | Streamlined intake, adjust queue when needed | Registration panel, override controls |
| **Hospital Admins** | Track performance, reduce complaints | Analytics dashboard, feedback reports |

### **❗ Key Assumptions & Design Decisions**

|  |  |
| --- | --- |
| **Item** | **Decision** |
| **Urgency Definition** | Based on **patient condition** + **staff input** at registration |
| **Offline Mode** | Enabled via **local caching**; re-sync when connection resumes |
| **User Guidance** | Help popups, interactive **tutorials** embedded in the app |
| **Primary Language** | **Swahili first**, with **English** as secondary language (future) |

## **✅ Step 2: System Architecture Design (Finalized)**

### **🏗️ 1. Architecture Style**

**Modular, Layered Client-Server Architecture (MVC Pattern)**

* **Frontend**: Mobile-first approach for patients (no web portal).
* **Backend/API Layer**: Central controller for all queue logic and services.
* **Database Layer**: Secure data storage for all queue, patient, and staff data.
* **Offline Sync Layer**: Local caching (mobile-side fallback).
* **Notification Engine**: Sends real-time updates to patients via SMS or push.

### **🔄 2. Data Flow / System Workflow**

Updated to reflect **mobile-first patient access**:

1. **Patient registers** via their **mobile phone app**.
   1. If unavailable, receptionist performs manual registration on their behalf.
2. **Queue number assigned** based on priority rules.
3. Patient receives confirmation and **estimated wait time** on their mobile.
4. **Receptionist monitors queue** and manages exceptions (e.g., late arrivals).
5. **Doctor dashboard** shows prioritized queue and patient details.
6. Data logged centrally for analytics and synced with local cache (offline-ready).

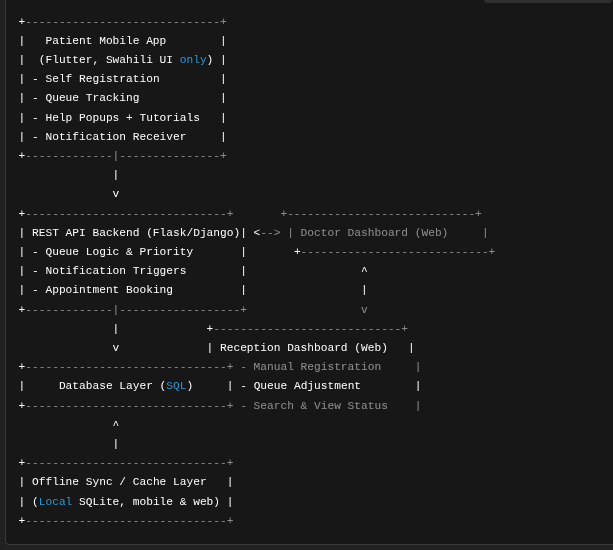
### **🧩 3. Key Components (Updated)**

|  |  |
| --- | --- |
| **Component** | **Purpose/Role** |
| **Mobile App (Patients)** | Registration, queue status, notifications, help UI (in Swahili) |
| **Receptionist Panel** | Manual registration, fallback access for patients without phones |
| **Doctor Dashboard** | Queue monitoring, patient prioritization, marking patients as seen |
| **Queue Engine** | Prioritizes patients based on urgency and category |
| **Notification Module** | Twilio + Firebase for SMS and push alerts |
| **Offline Sync Layer** | Ensures continuity when internet fails (for mobile and reception modules) |
| **Admin Analytics Panel** | For viewing reports, patient flow trends, and system usage |

### **📱 4. Interface Design Overview (Updated)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Interface** | **Platform** | **Target User** | **Key Functions** |
| **Mobile App** | React-native | Patients | Register, track queue, receive alerts, Swahili UI |
| **Reception Dashboard** | Web (React.js) | Receptionists | Manual registration, patient queue control |
| **Doctor Dashboard** | Web (React.js) | Doctors | View queue, prioritize, consult status updates |
| **Admin Panel** | Web (React.js) | Hospital Management | Metrics, logs, staff performance, feedback |

### **📡 5. Updated System Architecture Diagram**



### **🔐 6. System-Level Design Choices**

|  |  |
| --- | --- |
| **Area** | **Design Choice** |
| **Patient Access** | Mobile-first via **React-native app**, Swahili-first, no desktop UI |
| **Fallback Reg.** | Receptionist-assisted registration for patients **without phones** |
| **Queue Handling** | Central queue engine with smart prioritization rules |
| **Notifications** | Twilio (SMS) and Firebase (push) |
| **Offline Mode** | Mobile app and receptionist panel use **local cache**, syncs when online |
| **Language Support** | Swahili-first, with English toggle planned in future |

### **⚙️ Technology Stack Recap (Updated)**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Tools/Tech** | **Purpose** |
| Mobile App | React-native | Patient interface, built in Swahili |
| Web Interface | React.js | Receptionist, Doctor, and Admin UIs |
| Backend | Flask or Django (Python) | Business logic, APIs, integration |
| Database | MySQL or PostgreSQL | Queue, patient, appointment, and logs storage |
| Notifications | Twilio / Firebase | SMS & push alerts |
| Caching | SQLite / SharedPreferences | For offline-first patient app behavior |
| Security | JWT + HTTPS | Authentication, encrypted communication |

## **✅ Step 3: Component and Module Design**

This step defines **what each part of the system does**, how they talk to each other, and how they support user needs. It's the **blueprint for your code structure** and feature implementation.

### **🔧 1. Component Overview**

The system is made up of the following high-level components:

|  |  |
| --- | --- |
| **Component Name** | **Description** |
| **Mobile App (Patient)** | Swahili-first app for self-registration, queue tracking, notifications |
| **Reception Dashboard** | For staff to assist with patient registration and queue management |
| **Doctor Dashboard** | Interface for doctors to view and handle prioritized queue lists |
| **Queue Management Engine** | Core logic for assigning queue numbers and sorting based on urgency |
| **Notification Module** | Sends real-time updates via push or SMS |
| **Analytics Module** | Collects data on usage, wait times, queue efficiency |
| **Offline Sync Manager** | Ensures mobile app and receptionist module function offline |
| **Authentication Module** | Handles logins, permissions, and role-based access |

### **🧱 2. Detailed Module Design**

#### **🔹 A. Patient Mobile App Module (React-native)**

* **Functions**:
  + Account creation and login.
  + Patient self-registration for consultations.
  + Real-time view of queue position.
  + Display of estimated wait time.
  + Notifications for queue updates.
  + Language switcher (future).
  + Help pop-ups and tutorials.
* **Submodules**:
  + RegistrationForm
  + QueueStatusScreen
  + NotificationsHandler
  + HelpAndSupportOverlay
  + OfflineCacheManager

#### **🔹 B. Receptionist Dashboard Module (Web)**

* **Functions**:
  + Register patients without mobile access.
  + Assign queue numbers manually if needed.
  + View/edit current queue.
  + Check appointment history.
  + Sync with doctor schedules.
* **Submodules**:
  + PatientLookupForm
  + ManualQueueAssignment
  + QueueMonitorView
  + OfflineFallbackCache

#### **🔹 C. Doctor Dashboard Module (Web)**

* **Functions**:
  + View incoming patients.
  + Access patient medical notes (if integrated with EHR).
  + Mark patients as seen.
  + Provide feedback for queue adjustment.
* **Submodules**:
  + QueueListView
  + PatientDetailsViewer
  + ConsultationFeedbackForm

#### **🔹 D. Queue Management Engine**

* **Functions**:
  + Prioritize queue based on:
    - Condition severity (from receptionist/staff input).
    - Patient type (elderly, child, emergency).
    - Booking time.
  + Resolve queue position dynamically in real-time.
* **Key Algorithms**:
  + Priority-based sorting.
  + First-come-first-serve fallback.
* **Submodules**:
  + PriorityCalculator
  + QueueAllocator
  + QueueUpdater

#### **🔹 E. Notification Module**

* **Functions**:
  + Sends:
    - SMS (via Twilio)
    - Push notifications (via Firebase)
  + Triggers:
    - Queue number updates.
    - Estimated wait time changes.
    - “It’s your turn” alerts.
* **Submodules**:
  + TwilioSMSService
  + FirebasePushService
  + NotificationTriggerHandler

#### **🔹 F. Analytics & Reporting Module**

* **Functions**:
  + Track:
    - Number of patients served daily.
    - Average waiting time.
    - Missed appointments.
    - Queue efficiency.
  + Generate admin reports.
* **Submodules**:
  + DataCollector
  + StatisticsEngine
  + ReportExporter

#### **🔹 G. Offline Sync Module**

* **Functions**:
  + Locally cache patient registration and queue status.
  + Sync automatically when internet is restored.
* **Tools**:
  + SharedPreferences (React-native)
  + IndexedDB / LocalStorage (Web)

#### **🔹 H. Authentication Module**

* **Functions**:
  + Handles secure login (JWT-based).
  + Role-based access:
    - Patient, Receptionist, Doctor, Admin
  + Prevent unauthorized actions.

### **🔁 3. Component Interactions**

**Example Flow: A new patient registers from their mobile**

1. App submits form to **API**.
2. API invokes **Queue Engine** → assigns queue number.
3. Queue Engine stores data in **Database**.
4. **Notification Module** sends confirmation SMS/push.
5. Patient views queue status in real time via **Mobile App**.

### **🔐 4. Component Security Considerations**

* All communication over **HTTPS**.
* Each module validates user role before execution.
* API access controlled with **JWT tokens**.
* Patient data encrypted in the database.

### **✅ Module Design Summary Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Module** | **Language/Tool** | **Users Affected** | **Main Purpose** |
| Mobile App (React-native) | Javascript/React-native | Patients | Register, track queue, receive updates |
| Reception Panel (React.js) | JavaScript/React | Receptionists | Register and manage queue on behalf |
| Doctor Dashboard | JavaScript/React | Doctors | View and manage prioritized patient queue |
| Queue Engine | Python (Flask/Django) | Backend | Assign and reorder patient queues |
| Notifications | Twilio + Firebase | Patients | Real-time updates via SMS/push |
| Analytics | Python + Charts | Admins | Monitor system usage and performance |
| Offline Sync | SQLite/SharedPrefs | Patients, Receptionists | Maintain usability without internet |
| Authentication | JWT | All | Secure and role-based access control |

## **✅ Step 4: Database (Data) Design**

This section defines:

* **Entities (tables)**
* **Fields/attributes for each table**
* **Relationships between tables**
* **Constraints** and normalization logic

The design supports **queue logic**, **registration**, **notifications**, and **user roles**, optimized for an intelligent, Swahili-first mobile queue management system.

### **🗂️ 1. Core Database Entities**

|  |  |
| --- | --- |
| **Table Name** | **Purpose** |
| Patients | Stores personal details of patients |
| Appointments | Holds appointment and registration info |
| Queue | Manages queue numbers, priorities, and statuses |
| Users | Stores login credentials for doctors, receptionists, and admins |
| Doctors | Doctor-specific data and availability |
| Notifications | Logs all messages sent (SMS or push) |
| AuditLog | Tracks system activities (for analytics and debugging) |

### **🗂️ 1. Core Entities (Now with UUID Primary Keys)**

|  |  |  |
| --- | --- | --- |
| **Table Name** | **Primary Key (Type)** | **Purpose** |
| Patients | patient\_id (UUID) | Stores patient profile information |
| Appointments | appointment\_id (UUID) | Stores consultation booking & urgency details |
| Queue | queue\_id (UUID) | Holds queue number, priority, and current status |
| Users | user\_id (UUID) | Staff/admin login credentials and roles |
| Doctors | doctor\_id (UUID) | Maps user accounts to doctor-specific details |
| Notifications | notification\_id (UUID) | Logs SMS/push messages sent |
| AuditLog | log\_id (UUID) | Stores event/activity logs for analytics/debugging |

### **📋 2. Table Definitions (UUID-based)**

#### **🔹 Patients**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Constraints** |
| patient\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| full\_name | VARCHAR(100) | NOT NULL |
| phone\_number | VARCHAR(20) | UNIQUE, NOT NULL |
| dob | DATE |  |
| gender | ENUM | ('M', 'F', 'Other') |
| registered\_on | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

#### **🔹 Appointments**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Constraints** |
| appointment\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| patient\_id | UUID (FK) | REFERENCES Patients(patient\_id) |
| scheduled\_date | DATE | NOT NULL |
| scheduled\_time | TIME |  |
| urgency\_level | ENUM | ('low', 'medium', 'high') |
| created\_by | UUID (FK) | References Users(user\_id) (nullable) |

#### **🔹 Queue**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Constraints** |
| queue\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| appointment\_id | UUID (FK) | REFERENCES Appointments(appointment\_id) |
| queue\_number | VARCHAR(10) | NOT NULL |
| priority\_score | INT | Used to sort queue |
| status | ENUM | ('waiting', 'notified', 'served', 'skipped') |
| check\_in\_time | TIMESTAMP |  |
| served\_time | TIMESTAMP |  |

#### **🔹 Users (Staff/Admin)**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Constraints** |
| user\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| username | VARCHAR(50) | UNIQUE, NOT NULL |
| password\_hash | VARCHAR(255) | ENCRYPTED |
| role | ENUM | ('doctor', 'receptionist', 'admin') |
| name | VARCHAR(100) |  |

#### **🔹 Doctors**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Constraints** |
| doctor\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| user\_id | UUID (FK) | REFERENCES Users(user\_id) |
| specialty | VARCHAR(50) |  |
| available\_today | BOOLEAN | Default FALSE |

#### **🔹 Notifications**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Constraints** |
| notification\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| appointment\_id | UUID (FK) | REFERENCES Appointments(appointment\_id) |
| channel | ENUM | ('sms', 'push') |
| message | TEXT |  |
| sent\_time | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |
| status | ENUM | ('success', 'failed') |

#### **🔹 AuditLog**

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| log\_id | UUID (PK) | DEFAULT uuid\_generate\_v4() |
| event\_type | VARCHAR(50) | (e.g., 'register', 'notify', 'skip') |
| actor\_id | UUID | ID of user or patient (nullable) |
| description | TEXT |  |
| event\_time | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

### **🔁 3. Relationships (ERD Logic)**

* Patients → Appointments → Queue
* Appointments → Notifications
* Users → Doctors
* Appointments.created\_by may reference a receptionist user\_id
* All logs linked to either patients or users

### **📐 4. Normalization & Constraints**

* **3rd Normal Form (3NF)**: No redundant or duplicate data.
* **Primary Keys**: Each table has a unique ID.
* **Foreign Keys**: Enforced on relationships.
* **Indexes**: Applied on phone\_number, queue\_number, urgency\_level for fast lookups.

## **✅ Step 5: Interface Design**

This step focuses on the **User Interface (UI)** and **User Experience (UX)** for all roles involved in the system:

* Patients (mobile-first)
* Receptionists (web-based)
* Doctors (web-based)
* Admins (web-based)

### **🧑‍⚕️ 1. Design Principles**

|  |  |
| --- | --- |
| **Principle** | **Application** |
| **Mobile-First** | Patient interface built primarily for mobile (React-native) |
| **Swahili-First** | Default language is Swahili with support for English in future updates |
| **Clarity & Simplicity** | Large buttons, icons, and short labels for low-literacy users |
| **Accessibility** | Color contrast, icons, help pop-ups, and possible voice/touch guidance |
| **Role-Based UI** | Interfaces tailored to specific user roles with relevant features only |

### **📱 2. Patient Mobile App UI (React-native)**

#### **🔹 Main Screens:**

|  |  |
| --- | --- |
| **Screen** | **Description** |
| **Home Screen** | Welcome message, action buttons: “Jisajili” (Register), “Angalia Foleni” (Check Queue), “Msaada” (Help) |
| **Registration Screen** | Form with: full name, gender, birth date, phone number, and condition type (emergency, normal, elderly, child) |
| **Queue Status Screen** | Real-time queue number, estimated wait time, doctor name, current position |
| **Notification Center** | List of updates (e.g., “Zamu yako iko karibu” – Your turn is near) |
| **Tutorial / Help** | Popup tutorials with Swahili text + icon animations showing how to register, check status, etc. |
| **Settings** | Language selector (Swahili default), app version, logout |

#### **🖌️ Design Features:**

* Primary color theme: calming green/blue (hospital-friendly)
* Fonts: Bold and large for visibility
* Buttons: Clearly labeled, easy to tap

### **🖥️ 3. Receptionist Web Dashboard (React.js)**

#### **🔹 Key Panels:**

|  |  |
| --- | --- |
| **Section** | **Description** |
| **Quick Register** | Simple form to register patients who don’t have phones |
| **Queue Manager** | Live list of all patients in queue with ability to reorder or skip |
| **Patient Lookup** | Search past visits or upcoming appointments |
| **Offline Indicator** | Shows “Offline Mode” if internet is down (auto-syncs later) |

#### **🖌️ Design Features:**

* Sidebar navigation: “Usajili”, “Foleni”, “Ripoti”
* Toast-style alerts for errors, successes
* Keyboard shortcuts for rapid data entry

### **🧑‍⚕️ 4. Doctor Dashboard (Web)**

#### **🔹 Key Panels:**

|  |  |
| --- | --- |
| **Section** | **Description** |
| **My Queue** | Live list of patients assigned to the doctor, sorted by priority |
| **Patient Details** | Summary card: name, age, urgency level, last appointment (if any) |
| **Mark as Seen** | Button to complete consultation and remove from active queue |
| **Feedback Option** | Dropdown to mark delays, missed patients, or queue concerns |

#### **🖌️ Design Features:**

* Compact and data-dense layout for fast scanning
* Patient cards with visual urgency indicators (red = high, green = low)
* Auto-refresh queue

### **🛠️ 5. Admin Interface**

#### **🔹 Dashboard Features:**

|  |  |
| --- | --- |
| **Panel** | **Description** |
| **Analytics Overview** | Graphs showing patient volume, average wait time, peak hours |
| **User Management** | Create/edit/delete staff accounts |
| **Feedback Logs** | View feedback from doctors or receptionists |
| **Queue Configurator** | Set or adjust priority rules, thresholds |

#### **🖌️ Design Features:**

* Tab-based navigation (Overview, Staff, Queue Rules, Settings)
* Export buttons for CSV/PDF reports
* Access control (admin-only for critical features)

### **🧑‍🎓 6. Help & Onboarding (Cross-interface)**

|  |  |
| --- | --- |
| **Support Tool** | **Description** |
| **Pop-up Tutorials** | Step-by-step guides when a user opens a screen for the first time |
| **Tooltips** | Hover or tap icons with quick tips (“Jina lako kamili hapa”) |
| **FAQ Section** | On mobile: tappable answers in Swahili |
| **Offline Notice** | For patients: "Uko nje ya mtandao, taarifa zako zitahifadhiwa" |

### **🌐 7. Language & Localization**

* **Default language**: Swahili
* **English toggle**: Planned for future
* **Date format**: DD/MM/YYYY
* **Labels** and help text: Culturally localized (e.g., using terms like "Zamu" for queue turn)

### **🔒 8. Security & Access**

|  |  |  |
| --- | --- | --- |
| **Role** | **Interface** | **Access Control** |
| Patient | Mobile only (React-native) | Phone login or session-based auth |
| Receptionist | Web Dashboard | Username + password (JWT token) |
| Doctor | Web Dashboard | Secure login, restricted views |
| Admin | Web + full dashboard | All access, including logs/config |

## **✅ Step 6: Technology Stack Selection (Finalized)**

This tech stack is tailored to:

* Use **React Native** for mobile development (Swahili-first)
* Use **FastAPI + SQLAlchemy** for the backend
* Optimize performance, security, and offline support
* Leverage tools you are already familiar with (JavaScript, Python)

### **📱 1. Mobile App: Patient Interface**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Reason** |
| **Framework** | **React Native** | Cross-platform, JavaScript-based, Swahili UI support, known tech |
| **Language** | JavaScript / TypeScript | Native to React ecosystem |
| **Local Storage** | AsyncStorage / SQLite | For offline mode and queue sync |
| **Navigation** | React Navigation | Clean routing for mobile screens |
| **Push Notifications** | Firebase Cloud Messaging | Lightweight real-time alerts |
| **Design System** | NativeBase or TailwindCSS for RN | Fast UI prototyping, responsive UI |

### **🖥️ 2. Web Dashboards: Receptionist, Doctor, Admin**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Reason** |
| **Framework** | React.js | Fast, modular web UI for role-based dashboards |
| **Styling** | Tailwind CSS / Bootstrap | Quick, responsive, and accessible UI design |
| **State Management** | Redux / Context API | Role-based UI state (doctor, receptionist, admin) |
| **Local Cache** | IndexedDB / LocalStorage | Offline support for reception dashboard |

### **🧠 3. Backend API Server**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Reason** |
| **Framework** | **FastAPI (Python)** | Async-ready, high-performance REST API framework |
| **Language** | Python 3.11+ | Clean syntax, good for async, strong ecosystem |
| **ORM** | **SQLAlchemy** | Powerful, flexible, integrates well with PostgreSQL |
| **Serialization** | Pydantic | Type-safe request/response validation |
| **Authentication** | JWT / OAuth2 (FastAPI) | Secure access for patients, staff, admins |
| **API Docs** | Swagger (auto from FastAPI) | Built-in OpenAPI support for devs |

### **🗃️ 4. Database Layer**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Reason** |
| **DBMS** | **PostgreSQL** | Relational, scalable, strong UUID and JSON support |
| **UUID Support** | uuid-ossp extension | Enables unique IDs across distributed systems |
| **Migration Tool** | Alembic | Manage schema evolution/versioning |
| **Backup/Restore** | pgBackRest / pg\_dump | Reliable database backup utilities |

### **✉️ 5. Notifications**

|  |  |  |
| --- | --- | --- |
| **Channel** | **Technology** | **Purpose** |
| **SMS** | **Twilio** | Real-time alerts for patients without smartphones |
| **Push Notifications** | **Firebase Cloud Messaging** | Notify mobile users (React Native) |
| **Webhook Queueing** | Celery + Redis (optional) | Async job handling for delayed or retrying alerts |

### **☁️ 6. Infrastructure & DevOps**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Purpose** |
| **Hosting** | Render / DigitalOcean / Railway | Easy deployment of FastAPI + PostgreSQL + React |
| **Containerization** | Docker | Standardize environments across dev and prod |
| **CI/CD** | GitHub Actions / GitLab CI | Auto-deploy backend and mobile builds |
| **Monitoring** | Sentry / Prometheus + Grafana | Track crashes, performance, and system logs |
| **Error Logging** | FastAPI Middleware + Sentry | Catch and report API errors |

### **🔐 7. Security Framework**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Description** |
| **Authentication** | JWT (FastAPI Users/OAuth2) | Token-based access (mobile + web) |
| **Transport** | HTTPS (via TLS certs) | Secure communication |
| **Role-based Access** | FastAPI Dependencies | Enforce permissions per route |
| **Input Validation** | Pydantic | Validate all incoming data (e.g., form fields) |
| **Rate Limiting** | SlowAPI / Redis | Protect endpoints from abuse |
| **Data Encryption** | PostgreSQL at rest + HTTPS | Protect patient data |

### **🧪 8. Testing & QA Tools**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology** | **Purpose** |
| **API Testing** | Postman / Pytest | Test endpoint logic and authentication |
| **E2E Testing** | Detox (React Native) | Test patient mobile flows |
| **Unit Testing** | Pytest (FastAPI), Jest (React) | Logic validation for backend and frontend |
| **Mock Data** | Factory Boy / Faker | Generate fake test data for dev/testing |

### **✅ Final Stack Summary Table**

|  |  |
| --- | --- |
| **Layer** | **Tools / Frameworks** |
| **Mobile App** | React Native, AsyncStorage, Firebase, Swahili UI |
| **Web Dashboards** | React.js, Tailwind CSS, IndexedDB |
| **Backend API** | FastAPI, SQLAlchemy, Pydantic, JWT |
| **Database** | PostgreSQL (UUID), Alembic |
| **Notifications** | Firebase (push), Twilio (SMS) |
| **Security** | JWT, HTTPS, Role-based Access |
| **DevOps** | Docker, GitHub Actions, Render/DigitalOcean |
| **Monitoring** | Sentry, FastAPI Logs, Postman |

## **✅ Step 7: Security Design**

### **🔐 1. Authentication & Authorization**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Mechanism** | **Details** |
| **Token-Based Auth** | **JWT (JSON Web Tokens)** | Stateless login sessions for mobile and web users |
| **User Roles** | FastAPI + Role-based permissions | Patients, Doctors, Receptionists, Admins (RBAC model) |
| **Login API** | FastAPI OAuth2 Password Flow | Tokens expire + refresh mechanism (e.g., 15 min expiry) |
| **Mobile Auth** | Phone Number Login (w/ OTP optional) | Minimal friction for patients |

✅ *Why?* Ensures that each user only accesses data meant for them. JWT is easy to integrate with React Native and web.

### **🧱 2. Transport Security**

|  |  |  |
| --- | --- | --- |
| **Item** | **Security Mechanism** | **Details** |
| **TLS Encryption** | HTTPS via SSL Certificate | Encrypts all traffic between clients and servers |
| **HSTS Headers** | Enforced by backend proxy | Prevents downgrade to HTTP |
| **CORS Policy** | FastAPI middleware | Limits frontend API access to trusted origins |

✅ *Why?* Prevents data interception and MITM attacks during communication.

### **🔐 3. Data Protection & Privacy**

|  |  |  |
| --- | --- | --- |
| **Area** | **Technology / Practice** | **Details** |
| **Data at Rest** | PostgreSQL w/ encryption config | Protects stored patient data |
| **Data in Transit** | HTTPS everywhere | Secures form submissions, queue data, notifications |
| **Minimal Data** | Only essential PII collected | No excessive personal info requested |
| **Field Encryption** | Optional encryption for phone\_number, dob, gender using Fernet (Python) or PostgreSQL pgcrypto |  |

✅ *Why?* Healthcare data is sensitive. Encryption at both ends builds trust and complies with best practices.

### **🚫 4. Input Validation & Injection Protection**

|  |  |  |
| --- | --- | --- |
| **Type** | **Technology** | **Protection** |
| **Form Input** | Pydantic (FastAPI) | Validates all user input and payloads |
| **Query Parameters** | FastAPI typed routes | Prevents malformed URL queries |
| **Database Protection** | SQLAlchemy (ORM) | Protects against SQL injection |
| **XSS/CSRF (Web UI)** | React auto-escaping + CSRF tokens | No raw HTML injection from user input |

✅ *Why?* Validating early (at API entry) stops 90% of common attacks.

### **📱 5. Mobile App Security**

|  |  |  |
| --- | --- | --- |
| **Risk Area** | **Protection Mechanism** | **Implementation** |
| **Token Storage** | Encrypted storage (e.g. react-native-encrypted-storage) | Keeps JWT safe from app access |
| **Tampering** | App integrity checks (future scope) | Prevent rooted/JB devices from running app |
| **Sensitive Screenshots** | Disable screen capture for sensitive views | Especially queue and profile pages |

✅ *Why?* Prevents reverse engineering and data leaks through mobile compromise.

### **🚧 6. Rate Limiting & Abuse Prevention**

|  |  |  |
| --- | --- | --- |
| **Endpoint Type** | **Protection** | **Tool** |
| **Login & Registration** | Rate limiting per IP/phone/email | slowapi middleware in FastAPI |
| **SMS Flooding** | Throttling Twilio per number | Redis-based retry queue or cooldown logic |
| **Queue Manipulation** | Auth + input validation | Block bots or malicious users |

✅ *Why?* Even basic systems can be DoSed or abused without safeguards.

### **📜 7. Logging & Monitoring**

|  |  |  |
| --- | --- | --- |
| **Activity** | **Method** | **Tools** |
| **API Requests** | Middleware logging | Store event logs with timestamp, IP, status |
| **Authentication** | Login/logout attempts | Audit via FastAPI + PostgreSQL |
| **Errors** | Real-time error reporting | Sentry (backend & frontend) |
| **User Actions** | Custom audit log table | AuditLog entity in DB |

✅ *Why?* Logging is essential for tracing issues, security audits, and accountability.

### **⚠️ 8. Deployment Security**

|  |  |  |
| --- | --- | --- |
| **Area** | **Control Mechanism** | **Example** |
| **Environment Variables** | .env + secrets management | Use Docker secrets or Render secrets store |
| **Database Access** | Limited IP whitelist | Only API server can talk to DB |
| **File Permissions** | Docker container hardening | Least privilege principles |
| **CI/CD Safety** | Signed builds, test stage enforcement | No auto-deploy without tests passing |

### **✅ Summary: Security Strategy**

|  |  |
| --- | --- |
| **Security Area** | **Key Strategy** |
| Authentication | JWT with role-based access |
| Data Protection | TLS + PostgreSQL encryption + input limits |
| Mobile Security | Token encryption, screenshot protection |
| Rate Limiting | FastAPI + Redis + cooldowns |
| Validation | Pydantic + SQLAlchemy + CSRF-free React |
| Logs & Audits | Sentry + AuditLog + Error tracking |

## **✅ Step 8: Scalability and Performance Planning**

### **⚙️ 1. Scalability Strategy**

Scalability is about preparing your system to **handle growth** in:

* Number of **patients using the mobile app**
* Number of **simultaneous appointments/queues**
* Volume of **real-time notifications**
* Expansion to **other hospitals or departments**

### **🧱 A. Backend Scalability (FastAPI + SQLAlchemy)**

|  |  |
| --- | --- |
| **Strategy** | **Implementation** |
| **Async Support** | Use FastAPI's async endpoints + asyncpg with SQLAlchemy for non-blocking DB access |
| **Horizontal Scaling** | Deploy multiple FastAPI instances behind a load balancer (e.g., NGINX + Gunicorn + Uvicorn workers) |
| **Caching Layer** | Redis to store frequently accessed data (e.g., queue position summaries) |
| **Background Jobs** | Celery + Redis to handle notifications, delays, and reminders off the main thread |
| **Stateless APIs** | Rely on JWTs and centralized DB, so any instance can handle any request |

✅ *Result:* Backend can handle 100s to 1000s of concurrent users with horizontal scaling.

### **🧮 B. Database Performance (PostgreSQL)**

|  |  |
| --- | --- |
| **Strategy** | **Implementation** |
| **Indexing** | Use indexes on queue\_number, scheduled\_date, urgency\_level, phone\_number |
| **Connection Pooling** | Use asyncpg or SQLAlchemy pool with FastAPI to manage DB connections efficiently |
| **UUID Optimization** | Use UUID v4 with default indexing techniques |
| **Partitioning (Future)** | Split large tables (e.g., Queue, Appointments) by date or department |
| **Query Optimization** | Monitor slow queries with pg\_stat\_statements, EXPLAIN plans |

✅ *Result:* Query response times remain fast even with millions of records.

### **📱 C. Mobile App Optimization (React Native)**

|  |  |
| --- | --- |
| **Strategy** | **Implementation** |
| **Local Caching** | Store queue status locally via SQLite or AsyncStorage to reduce API load |
| **Efficient Navigation** | Use React Navigation’s lazy loading and stack clearing to reduce memory usage |
| **Throttled Polling** | Limit how often the app checks queue status (e.g., every 60 seconds) |
| **Offline Sync** | Use background task (e.g., react-native-background-fetch) to sync quietly |

✅ *Result:* Smooth UX for patients even on low-end Android devices and limited networks.

### **🖥️ D. Frontend (Web Dashboards: Receptionist, Doctor, Admin)**

|  |  |
| --- | --- |
| **Strategy** | **Implementation** |
| **Component Lazy Loading** | Use React lazy/suspense for route-level chunking |
| **Pagination + Virtualization** | Use libraries like react-virtualized for long queues or search results |
| **Queue Filtering on Server** | Only load patients relevant to the logged-in doctor or day |
| **Debounce Inputs** | Prevent excessive DB/API hits when searching or updating queues |

✅ *Result:* Dashboards stay responsive even with heavy queue data.

### **🛠️ E. Infrastructure Planning**

|  |  |
| --- | --- |
| **Area** | **Strategy** |
| **Containerization** | Use Docker for backend + React builds |
| **Load Balancing** | NGINX or Render built-in LB for distributing traffic |
| **Auto-Scaling** | Render/DigitalOcean scale FastAPI instances based on CPU/memory |
| **CDN (optional)** | Use Cloudflare or similar for static assets (web admin UI) |

✅ *Result:* Cloud-native scalability out of the box.

### **📊 F. Monitoring & Metrics**

|  |  |
| --- | --- |
| **Metric Tracked** | **Tools/Method** |
| **Queue processing time** | Logged in DB or Prometheus metrics |
| **API response time** | FastAPI middleware + Sentry/APM tools |
| **Database performance** | pg\_stat\_statements, PgHero (optional) |
| **Mobile crash/error reporting** | Firebase Crashlytics / Sentry for React Native |

✅ *Result:* Proactive performance debugging and scaling insights.

### **🧭 G. Future Scalability Features**

|  |  |
| --- | --- |
| **Feature** | **Plan** |
| **Multitenancy** | Add a hospital\_id column to allow support for multiple hospital branches |
| **Queue Service Decoupling** | Extract queue logic into a microservice (e.g., FastAPI + Redis queue engine) |
| **Real-time WebSockets (Optional)** | Use FastAPI + WebSockets for live queue updates (low priority) |

### **✅ Summary: Scalability & Performance Plan**

|  |  |
| --- | --- |
| **Layer** | **Scalable Approach** |
| Backend | Async FastAPI + Redis + Uvicorn workers |
| Database | Indexed PostgreSQL with UUID + partitioning |
| Mobile | React Native + caching + throttled sync |
| Dashboards | Lazy loading, pagination, server-side filter |
| Infrastructure | Docker + Load Balancing + Auto-scaling |
| Monitoring | Sentry + Logs + pg\_stat\_statements |

## **✅ Step 10: Design Documentation**

### **📚 1. Core Documents to Create**

|  |  |
| --- | --- |
| **Document Name** | **Description** |
| **Software Requirements Spec (SRS)** | All functional & non-functional requirements, user roles, feature lists |
| **System Architecture Document** | Overview of backend, frontend, DB, and how components interact |
| **Component Design Document** | Detailed module breakdown (API, Mobile, Queue Engine, etc.) |
| **Database Schema (ERD + SQL)** | Visual + SQL schema of all tables with UUIDs, constraints, relationships |
| **API Specification (OpenAPI)** | Auto-generated or manually written REST endpoint docs |
| **UI/UX Wireframes & Flow Diagrams** | Visuals of patient app, receptionist, doctor, and admin panels |
| **Security Strategy Document** | How roles, tokens, encryption, and validation are handled |
| **Offline Sync & Caching Plan** | Strategy and flow for handling disconnections in mobile/web |
| **Scalability Plan** | Load balancing, DB optimization, multi-hospital support |
| **Testing Plan** | Test strategy, tools, roles, example cases, CI/CD hooks |
| **Deployment Guide** | How to build, configure, and launch the app and backend |
| **Change Log** | Versioned updates with dates and summaries |
| **Future Work & Roadmap** | Planned upgrades: multilingual toggle, EHR integration, AI queue logic |

### **🧾 2. Recommended Structure (Table of Contents Format)**

📁 Documentation/  
├── 📄 01\_Requirements\_Specification.pdf  
├── 📄 02\_System\_Architecture\_Diagram.pdf  
├── 📄 03\_Module\_Design\_Details.pdf  
├── 📄 04\_Database\_Schema\_ERD.pdf  
├── 📄 05\_API\_Specification\_OpenAPI.yaml  
├── 📄 06\_UI\_Wireframes\_and\_Flows.pdf  
├── 📄 07\_Security\_Design.pdf  
├── 📄 08\_Offline\_Sync\_Strategy.pdf  
├── 📄 09\_Scalability\_and\_Performance\_Plan.pdf  
├── 📄 10\_Testing\_Strategy.pdf  
├── 📄 11\_Deployment\_Guide.md  
├── 📄 12\_Change\_Log.md  
├── 📄 13\_Future\_Improvements\_and\_Expansion.pdf

### **🛠️ 3. Auto-Generated vs Manual**

|  |  |
| --- | --- |
| **Artifact** | **Recommended Tool** |
| OpenAPI Spec | FastAPI (auto at /docs or /openapi.json) |
| ERD (Database Diagram) | dbdiagram.io / DrawSQL / PgAdmin ERD |
| Wireframes | Figma / Balsamiq / Pen + DALL·E (done ✅) |
| API Testing/Docs | Postman Collections + Swagger UI |
| Flow Diagrams | Lucidchart, Excalidraw, diagrams.net |
| Version Control & Docs Hosting | GitHub + Markdown/README + PDF export |

### **📄 4. Special Sections to Highlight in Your Docs**

#### **🔹 System Goals & Context**

* Problem statement (e.g., hospital wait times)
* Scope (OPD department only for now)
* Stakeholders (patients, doctors, receptionists, admin)

#### **🔹 User Personas**

* A patient with a low-end smartphone, using Swahili UI
* A receptionist with limited digital training
* A doctor needing a fast view of who's next
* An admin pulling performance reports

#### **🔹 Language & Accessibility**

* Swahili-first content, optional English toggle
* Visual guidance (help popups)
* Offline capability explained visually

### **✅ What You’ve Already Covered (and Can Now Document)**

|  |  |  |
| --- | --- | --- |
| **Section** | **Status** | **Source** |
| Requirements Analysis | ✅ Complete | Step 1 summary |
| Architecture & Components | ✅ Complete | Steps 2–4 |
| Tech Stack | ✅ Finalized | Step 6 update |
| Security & Performance | ✅ Designed | Steps 7–8 |
| Wireframes | ✅ Generated | Patient + Receptionist views done via AI |
| Testing Plan | ✅ Documented | Step 9 |