Addis Ababa University

College of Natural and Computational Sciences

Department of Computer Science

Digital Queue Management System for Tikur Anbessa Hospital

Software Design Specification

Team Members

Name	ID NO.
1. Adonias Abiyot	UGR/0796/15
2. Eyob Assayie	UGR/1219/16
3. Lawgaw sefineh	UGR/8107/17
4. Samuel kinfe	UGR/2027/16
5. Sawel yohannes	UGR/2969/16
6. Tsegaye shewamare	UGR/2048/16

ADVISOR: Aderaw Semma

Date; Saturday, 29 November

Contents

List of Tables

List of Figures

Definitions, Acronyms, Abbreviations

Admin – A system user responsible for managing departments, users, and queue configurations.

API – Application Programming Interface; a set of functions enabling communication between software components.

Audit Log – A secure record of user actions, system changes, and data access.

Authentication – Verification of a user's identity before granting system access.

Authorization – Permission control determining what system resources a user can access.

BMI – Body Mass Index; a medical calculation based on height and weight.

DB – Database; a structured storage system for persistent data.

DQMS – Digital Queue Management System; the system designed for Black Lion Hospital.

EMR – Electronic Medical Record; a digital medical record system used by hospitals.

EWT – Estimated Wait Time; dynamically calculated time prediction for patient service.

FHIR – Fast Healthcare Interoperability Resources; a standard for healthcare data exchange.

HL7 – Health Level Seven; international standards for the exchange of medical information.

ICT – Information and Communication Technology.

In-Progress – Queue state indicating a patient is currently being served.

Kiosk – A self-service device used by patients for registration and queue requests.

Node.js – JavaScript runtime used for server-side application logic.

PostgreSQL – Relational database system storing persistent application data.

Queue Entry – A digital record representing a patient's position and state in the queue.

SMS – Short Message Service; text notification mechanism used by the system.

UI – User Interface; the front-end system presented to users.

UML – Unified Modeling Language; a standardized modeling notation used in software design.

WAN – Wide Area Network; the hospital's internal communication network.

1. Introduction

This Software Design Specification (SDS) provides a comprehensive technical design for the Digital Queue Management System (DQMS) developed for the Ethiopian Black Lion Hospital. It expands upon the Software Requirements Specification (SRS) by translating functional and non-functional requirements into architectural structures, component designs, data models, and interaction diagrams. This document serves as the primary guide for developers, system analysts, testers, and ICT technicians in constructing and maintaining the system.

The SDS ensures that all system components, interfaces, and operational flows are clearly defined and traceable to the requirements, ensuring consistency, reliability, and a shared understanding of the system's technical blueprint.

1.1 Purpose

The purpose of this System Design document is to translate the business requirements and hospital workflow processes identified in the SRS into a detailed technical design that will guide the development of the Digital Queue Management System (DQMS). It serves as a bridge between requirement analysis and actual system implementation Specifically, the SDS aims to;
- Define the system's architecture, components, modules, and interactions.
- Describe data structures, processing logic, and workflow behavior.
- Provide developers with a clear blueprint for coding and system integration.
- Support testers in validating system functionality and performance.
- Ensure long-term maintainability, scalability, and alignment with hospital operations.

1.2 General Overview

The Digital Queue Management System (DQMS) is designed to automate, monitor, and streamline patient queue operations at Black Lion Hospital. The system operates within the hospital's infrastructure and interacts with various hardware and software components such as:
- Patient self-service kiosks
- Digital display screens
- Administrative workstations
- Local servers and optional cloud backup services

The system uses a layered architectural approach composed of:
1. Presentation Layer – handles user interfaces for patients and staff.
2. Application/Business Logic Layer – processes queue logic, token management, and service flow.
3. Data Layer – manages data storage related to patients, queues, logs, and system usage.

The high-level context diagram from the SRS has been updated to reflect improved understanding of subsystem interactions. Changes include clearer separation between the kiosk subsystem and the administrative queue management interface, and the addition of a real-time display communication service.

Design goals include:
- Minimizing patient waiting time and congestion
- Ensuring real-time and accurate queue information
- Maintaining system reliability during peak hospital hours
- Supporting scalability for multiple specialized departments
- Ensuring security and controlled access to administrative functions

1.3 Development Methods & Contingencies

The design of the DQMS uses an Object-Oriented Design (OOD) methodology supported by Unified Modeling Language (UML). This approach ensures modularity, reusability, and maintainability through clearly defined classes and object interactions.

Methods considered:
- Structured Design: efficient for simple workflows but not suitable for the modular and scalable needs of DQMS.
- Prototyping Model: helpful for interface refinement but insufficient for overall system structure.
- Object-Oriented Design (Selected): provides strong abstraction, better scalability, and fits well with UML documentation.

UML artifacts used include:
- Use Case Diagrams
- Class Diagrams
- Activity Diagrams
- Sequence Diagrams
- Deployment Diagrams

Contingencies that may influence system design include:
- Changes in hospital workflow or patient processing policy
- Delays in obtaining integration details from third-party display system vendors
- Infrastructure limitations such as poor local network reliability
- Future need to integrate with Electronic Medical Record (EMR) systems

Workarounds and adaptation measures:
- Maintain modular subsystem boundaries to isolate potential changes
- Use API-based communication for flexible integration
- Include fallback offline modes for kiosk operations
- Use iterative development cycles to incorporate evolving requirements

2. System Architecture

2.1 Subsystem Decomposition

The Digital Queue Management System (QMS) adopts a modular, layered architecture that separates concerns and promotes maintainability. The system is decomposed into four primary subsystems that work collaboratively to deliver the required functionality while ensuring scalability, security, and interoperability with existing hospital systems.

Architecture Overview:

The system follows a client-server model with a web-based frontend and a robust backend service layer. This decomposition enables clear separation between user interface concerns, business logic, data management, and external system integrations. Each subsystem is designed to be independently testable and deployable, following the principles of high cohesion and low coupling.

Subsystem Responsibilities:

1. **User Interface (UI) Subsystem**: Serves as the primary interaction point for all users including patients, doctors, lab technicians, and administrators. It provides responsive web interfaces optimized for various devices and manages real-time display updates for public screens. This subsystem handles both the patient-facing portal for queue management and the staff-facing dashboard for queue administration.
2. **Application Logic Subsystem**: Implements the core business rules and workflow management. This subsystem handles user authentication and authorization, queue processing algorithms, notification scheduling, and coordinates all system operations. It contains the essential business logic for queue allocation, wait time calculations, and automatic queue management functions.
3. **Data Management Subsystem**: Provides persistent storage and retrieval of all system data including patient information, queue records, user accounts, and audit logs. Ensures data integrity and supports transaction management while maintaining compliance with data retention policies specified in the SRS.
4. **External Integration Subsystem**: Manages all external communications and data exchanges. This includes integration with the hospital's Ewket EMR system using HL7 FHIR standards and communication with SMS gateway services for patient notifications. This subsystem ensures seamless interoperability with existing hospital systems while maintaining data consistency.

UML Component Diagram

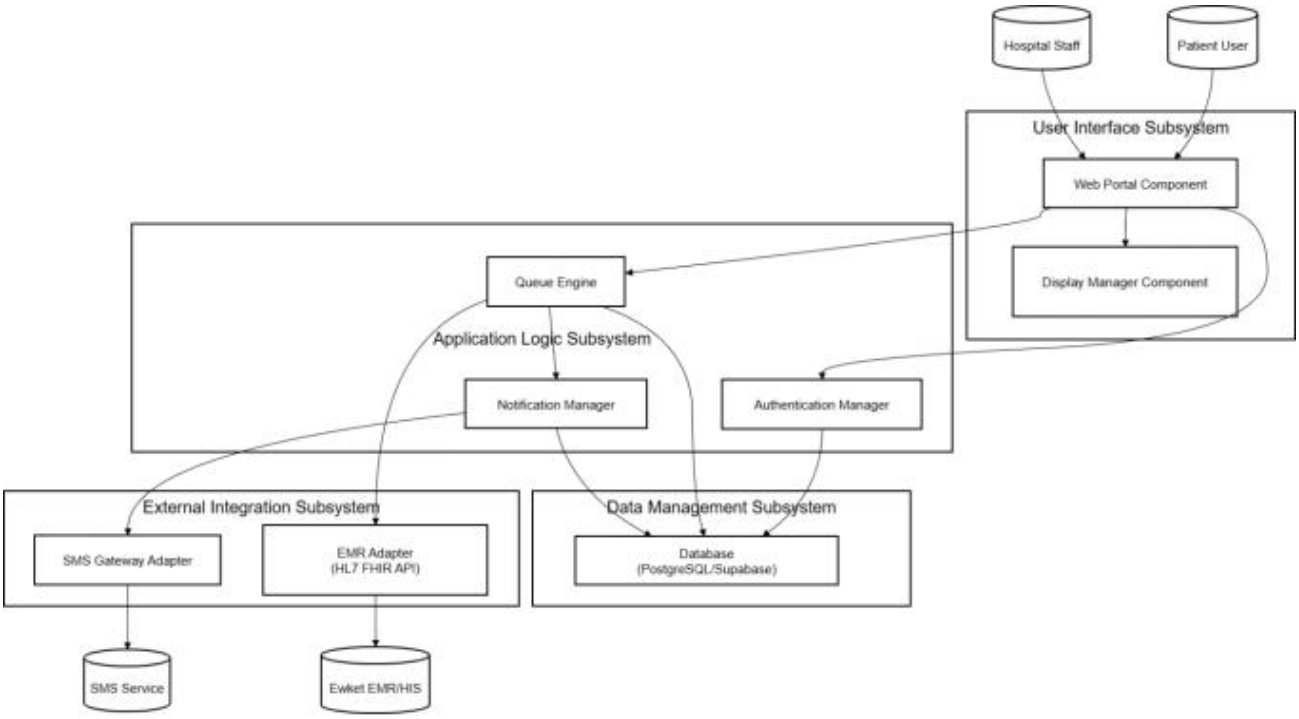

Figure 2.1

2.2 Hardware/Software Mapping

The deployment architecture leverages the hospital's existing IT infrastructure while ensuring high availability and performance. The system is designed to operate within the hospital's Wide Area Network (WAN), connecting multiple buildings and departments seamlessly.

Infrastructure Strategy:
The deployment follows a centralized server model with distributed client access points. All backend services are hosted on dedicated servers within the hospital's data center, while client access is provided through standard web browsers on various devices. This approach minimizes deployment complexity and leverages existing hospital network infrastructure.

Network Configuration:

- **Hospital WAN**: Provides connectivity between all departments and the central server
- **Client Network Segments**: Separate VLANs for administrative workstations, public displays, and patient WiFi

Software Stack Specification:

- **Operating System**: Ubuntu Linux 20.04 LTS for server components
- **Web Server**: Nginx for static content and reverse proxy
- **Application Runtime**: Node.js with Express.js framework
- **Database**: PostgreSQL with Supabase management layer
- **Client Software**: Chrome browser (v129.x+) for optimal compatibility

UML Deployment Diagram

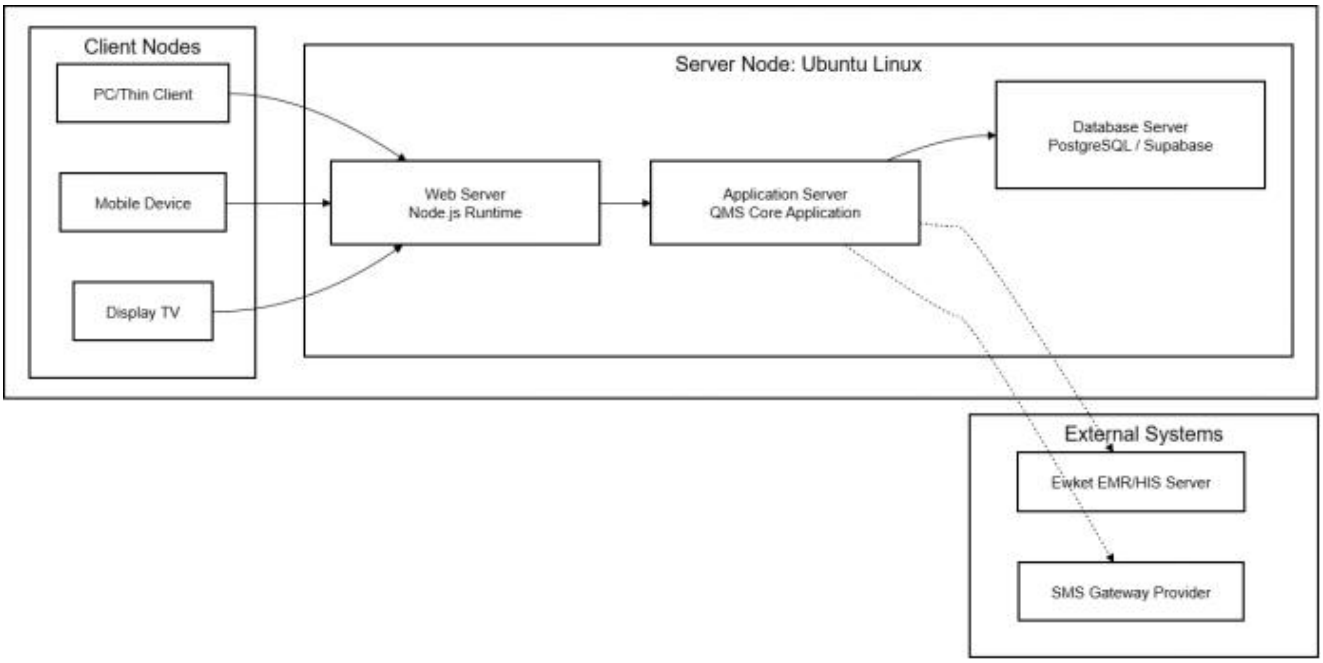

Figure 2.2

Deployment Notes:

- All internal communications use encrypted protocols (HTTPS/TLS 1.2+)
- Database server operates within a private subnet for enhanced security
- Load balancing capabilities are built-in for future scalability
- Regular automated backups are configured for disaster recovery
- Monitoring agents are deployed for performance tracking and alerting

This architecture ensures a clear separation of concerns, supports the specified functional and non-functional requirements, and aligns with the technical constraints outlined in the SRS while providing a foundation for future expansion and integration.

2.3 Access Control

Access control defines how different users interact with the Digital Queue Management System and ensures that sensitive health-related information is accessed only by authorized personnel. The system uses a **Role-Based Access Control (RBAC)** model, where each user role is assigned specific permissions according to its responsibilities within the hospital workflow.

1. Access Control Model

The system implements **RBAC (Role-Based Access Control)** with four primary roles:

1. Patient

2. Doctor

3. Lab Technician

4. Administrator

Each role is granted access only to the operations necessary for performing its tasks. All access rights follow the **least-privilege principle**, ensuring users can only perform what is essential for their responsibilities.

2. User Roles and Permissions

2.1 Patient

Allowed actions:

- Register or log in to the system.
- Request a queue number (online or assisted on-site).
- View queue status in real time.
- Cancel a queue request.
- Receive SMS or screen notifications.

Restrictions:

- Cannot view other patients' information.
- Cannot modify queue assignments or system configurations.

2.2 Doctor

Allowed actions:

- Log in to the system.
- Mark availability (available / not available).
- View assigned patients in the queue.
- Call the next patient.
- Update queue progress.

Restrictions:

- Cannot manage user accounts.
- Cannot assign queues manually.
- Cannot modify lab results.

2.3 Lab Technician

Allowed actions:

- Log in to the system.
- Update laboratory test status/results.
- View queue data related to lab tests.

Restrictions:

- Cannot manage users.
- Cannot call next patient for doctors' queue.
- Cannot assign patients to doctors.

2.4 Administrator

Allowed actions (full control):

- Add, update, or remove users (staff & patients).
- Assign patients to available doctors.
- Generate or modify queue numbers (for on-site requests).
- Monitor system-wide queue status.
- View and manage logs.
- Configure notifications and system settings.

Restrictions:

- Cannot change medical information or lab results.

3. Authentication Requirements

To ensure secure access, the system enforces:

- **Unique username & password** for all users.
- **Password hashing** (bcrypt / SHA-256).
- **Account lockout** after multiple failed login attempts (configurable).
- **Automatic logout after 10 minutes of inactivity.**
- **HTTPS/TLS 1.2+** for all data transmission.
- **Two-step verification** (optional future enhancement).

4. Authorization Enforcement

Authorization is enforced at **three levels**:

4.1 UI-Level Access Control

- Menus, screens, and buttons are shown only when the user's role permits.

4.2 API-Level Authorization

Each backend endpoint validates:

- User identity (session token / JWT)
- User role
- Permission scope

4.3 Database-Level Access Control

- Separate tables for sensitive data.
- Only admin-level operations can modify user records.
- Foreign key constraints ensure integrity.
- Access validated before any write operation.

5. Audit Logging and Monitoring

The system logs:

- Login and logout attempts
- Queue creation/modification
- Patient assignment changes
- Lab test updates

- Account management actions

These logs:

- Are accessible only to **Administrators**
- Support post-incident investigations
- Comply with **Ethiopian EMR security requirements**

6. Compliance and Security Policies

The access control design aligns with:

- **Ethiopian EMR Standards**
- **National Health Data Security Guidelines**
- **ISO/IEC 27001 security principles**
- **Least Privilege and Need-to-Know principles**

3. Object Model - Dynamic Diagrams

3.1 Class Diagram

This class diagram models the Digital Queue Management System's structure. It defines key entities and their relationships. It serves as a blueprint for developers to implement the system's object-oriented architecture. The diagram ensures all functional requirements from the project scope are properly encapsulated in classes and methods. It provides a visual guide for database design and system development while maintaining UML standards.

Core Classes:

· **User**: userId, name, phone, email (Base class)

· **Patient**: patientId, registrationDate (inherits User)

· **Staff:** staffId, role, department (inherits User)

· **Admin/Doctor/LabTechnician** (inherit Staff with role-specific attributes)

· **Queue:** queueId, status, queueNumber, department

· **Notification:** notificationId, message, type, sentTime

· **BMI/TestResults**: Store medical data and calculations

Key Relationships:

· Patient has Queues & Notifications

· Staff roles manage system components

· Database stores all entities

· Inheritance hierarchy for Users

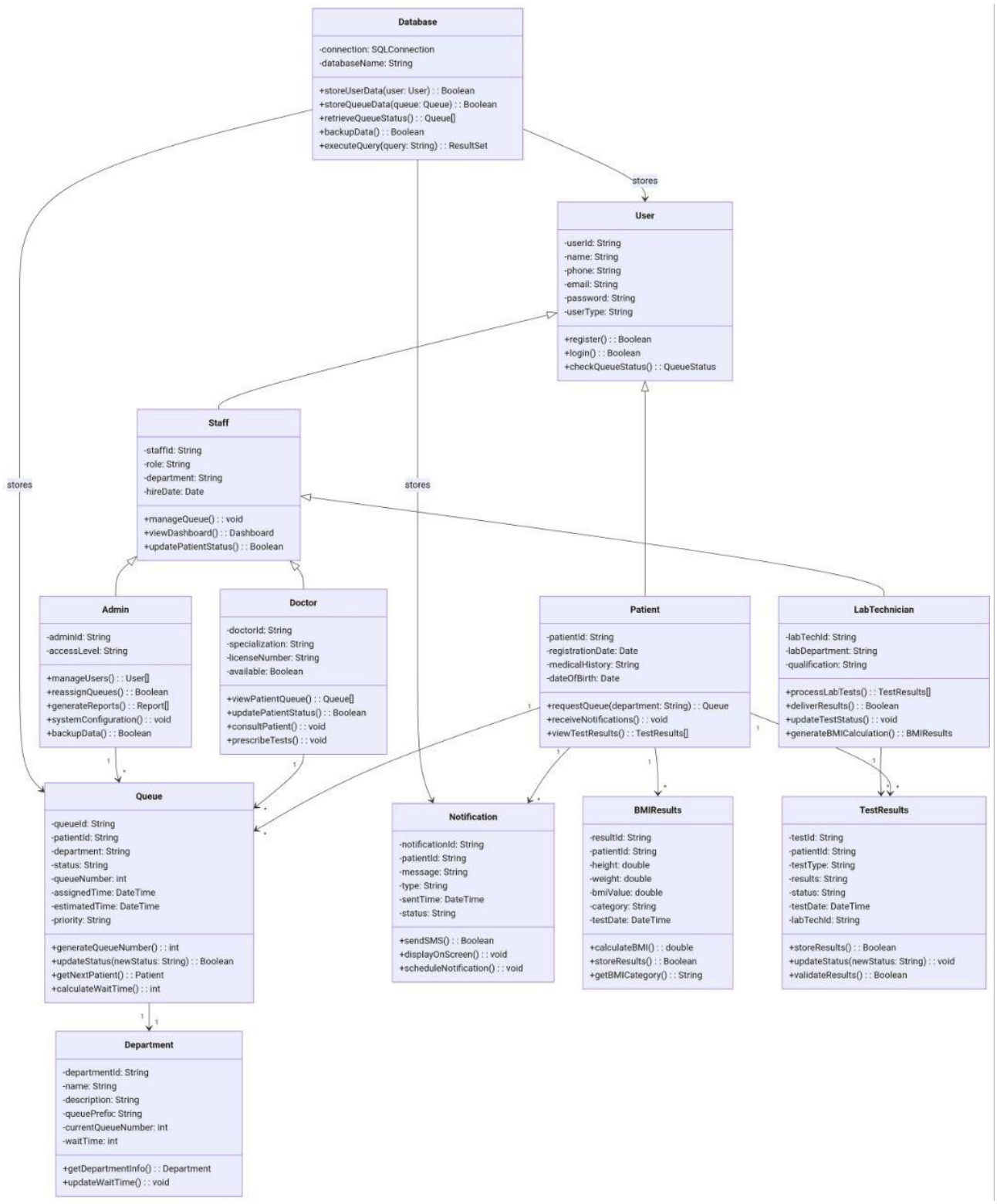
Database
-connection: SQLConnection
-databaseName: String
+storeUserData(user: User) : : Boolean
+storeQueueData(queue: Queue) : : Boolean
+retrieveQueueStatus() : : Queue[]
+backupData() : : Boolean
+executeQuery(query: String) : : ResultSet
stores
User
-userId: String
-name: String
-phone: String
-email: String
-password: String
-userType: String
+register() : : Boolean
+login() : : Boolean
+checkQueueStatus() : : QueueStatus
Staff
-staffId: String
-role: String
-department: String
-hireDate: Date
+manageQueue() : : void
+viewDashboard() : : Dashboard
+updatePatientStatus() : : Boolean
stores
stores
Admin
-adminId: String
-accessLevel: String
+manageUsers() : : User[]
+reassignQueues() : : Boolean
+generateReports() : : Report[]
+systemConfiguration() : : void
+backupData() : : Boolean
Doctor
-doctorId: String
-specialization: String
-licenseNumber: String
-available: Boolean
+viewPatientQueue() : : Queue[]
+updatePatientStatus() : : Boolean
+consultPatient() : : void
+prescribeTests() : : void
Patient
-patientId: String
-registrationDate: Date
-medicalHistory: String
-dateOfBirth: Date
+requestQueue(department: String) : : Queue
+receiveNotifications() : : void
+viewTestResults() : : TestResults[]
LabTechnician
-labTechId: String
-labDepartment: String
-qualification: String
+processLabTests() : : TestResults[]
+deliverResults() : : Boolean
+updateTestStatus() : : void
+generateBMICalculation() : : BMIResults
Queue
-queueId: String
-patientId: String
-department: String
-status: String
-queueNumber: int
-assignedTime: DateTime
-estimatedTime: DateTime
-priority: String
+generateQueueNumber() : : int
+updateStatus(newStatus: String) : : Boolean
+getNextPatient() : : Patient
+calculateWaitTime() : : int
Notification
-notificationId: String
-patientId: String
-message: String
-type: String
-sentTime: DateTime
-status: String
+sendSMS() : : Boolean
+displayOnScreen() : : void
+scheduleNotification() : : void
BMIResults
-resultId: String
-patientId: String
-height: double
-weight: double
-bmiValue: double
-category: String
-testDate: DateTime
+calculateBMI() : : double
+storeResults() : : Boolean
+getBMICategory() : : String
TestResults
-testId: String
-patientId: String
-testType: String
-results: String
-status: String
-testDate: DateTime
-labTechId: String
+storeResults() : : Boolean
+updateStatus(newStatus: String) : : void
+validateResults() : : Boolean
Department
-departmentId: String
-name: String
-description: String
-queuePrefix: String
-currentQueueNumber: int
-waitTime: int
+getDepartmentInfo() : : Department
+updateWaitTime() : : void

Figure 3.1

User

- **Attributes**: userId (unique identifier), name (full name), phone (contact number), email (email address), password (login credential), userType (role classification)
- **Methods**: register() (create account), login() (authenticate user), checkQueueStatus() (view queue position)

Patient

- **Attributes**: patientId (medical record number), registrationDate (sign-up date), medicalHistory (past treatments), dateOfBirth (birth date)
- **Methods:** requestQueue() (get queue number), receiveNotifications() (get alerts), viewTestResults() (access lab reports)

Staff

- **Attributes:** staffId (employee ID), role (job position), department (work unit), hireDate (employment start date)
- **Methods:** manageQueue() (control patient flow), viewDashboard() (see system overview), updatePatientStatus() (change patient state)

Admin

- **Attributes:** adminId (administrator ID), accessLevel (permission level)
- **Methods:** manageUsers() (handle user accounts), reassignQueues() (redirect queues), generateReports() (create analytics), systemConfiguration() (adjust settings)

Doctor

- **Attributes:** doctorId (physician ID), specialization (medical field), licenseNumber (medical license), available (status)
- **Methods**: viewPatientQueue() (see patient list), updatePatientStatus() (mark consultation complete), consultPatient() (examine patient), prescribeTests() (order lab tests)

LabTechnician

- **Attributes**: labTechId (technician ID), labDepartment (lab section), qualification (certifications)
- **Methods**: processLabTests() (analyze samples), deliverResults() (send test outcomes), updateTestStatus() (change test state), generateBMICalculation() (compute BMI)

Queue

- **Attributes**: queueId (queue identifier), patientId (patient reference), department (service point), status (current state), queueNumber (position), assignedTime (creation time), estimatedTime (expected time), priority (urgency level)
- **Methods:** generateQueueNumber() (create queue position), updateStatus() (change queue state), getNextPatient() (call next patient), calculateWaitTime() (estimate waiting time)

Notification

- **Attributes**: notificationId (alert ID), patientId (recipient), message (content), type (SMS/screen), sentTime (dispatch time), status (delivery state)
- **Methods**: sendSMS() (text message), displayOnScreen() (screen display), scheduleNotification() (time alerts)

Department

- **Attributes**: departmentId (unit ID), name (department name), description (unit purpose), queuePrefix (queue coding), currentQueueNumber (active number), waitTime (average wait)
- **Methods**: getDepartmentInfo() (retrieve details), updateWaitTime() (adjust wait estimates)

BMIResults

- **Attributes**: resultId (BMI record ID), patientId (patient reference), height (patient height), weight (patient weight), bmiValue (calculated BMI), category (weight classification), testDate (assessment date)
- **Methods**: calculateBMI() (compute BMI), storeResults() (save data), getBMICategory() (determine weight class)

TestResults

- **Attributes**: testId (lab test ID), patientId (patient reference), testType (lab test name), results (findings), status (completion state), testDate (test date), labTechId (technician reference)
- **Methods:** storeResults() (save outcomes), updateStatus() (change test state), validateResults() (verify accuracy)

Database

- **Attributes:** connection (database link), databaseName (database identifier)
- **Methods:** storeUserData() (save user info), storeQueueData() (save queue info), retrieveQueueStatus() (get queue state), backupData() (create backups), executeQuery() (run database commands)

The Dynamic Model supplements the static Class Diagram by illustrating the system's time-dependent behavior and the lifecycle of core entities. This section details the system's operational flow using the Sequence Diagram (3.2.1) and the State Chart Diagram (3.2.2).

3.2.1 Sequence Diagram: Patient Check-in and Queue Assignment

The Sequence Diagram illustrates the temporal order of object interactions for the **Patient Check-in and Queue Assignment** use case. This process is initiated by the Receptionist upon patient arrival and involves the core system components responsible for data verification and service assignment. The diagram is critical for validating the system's compliance with functional requirement 2.1 (Patient Registration).

The sequence explicitly details the conditional logic (alt fragment) required to handle both existing patient verification against the **Patient Records (DB)** and the persistent registration of new patient data. The successful completion of this sequence results in a confirmed QueueEntry object being created in the **Queue Records (DB)**, and the assigned identifier and Estimated Wait Time (EWT) being communicated back to the Receptionist.

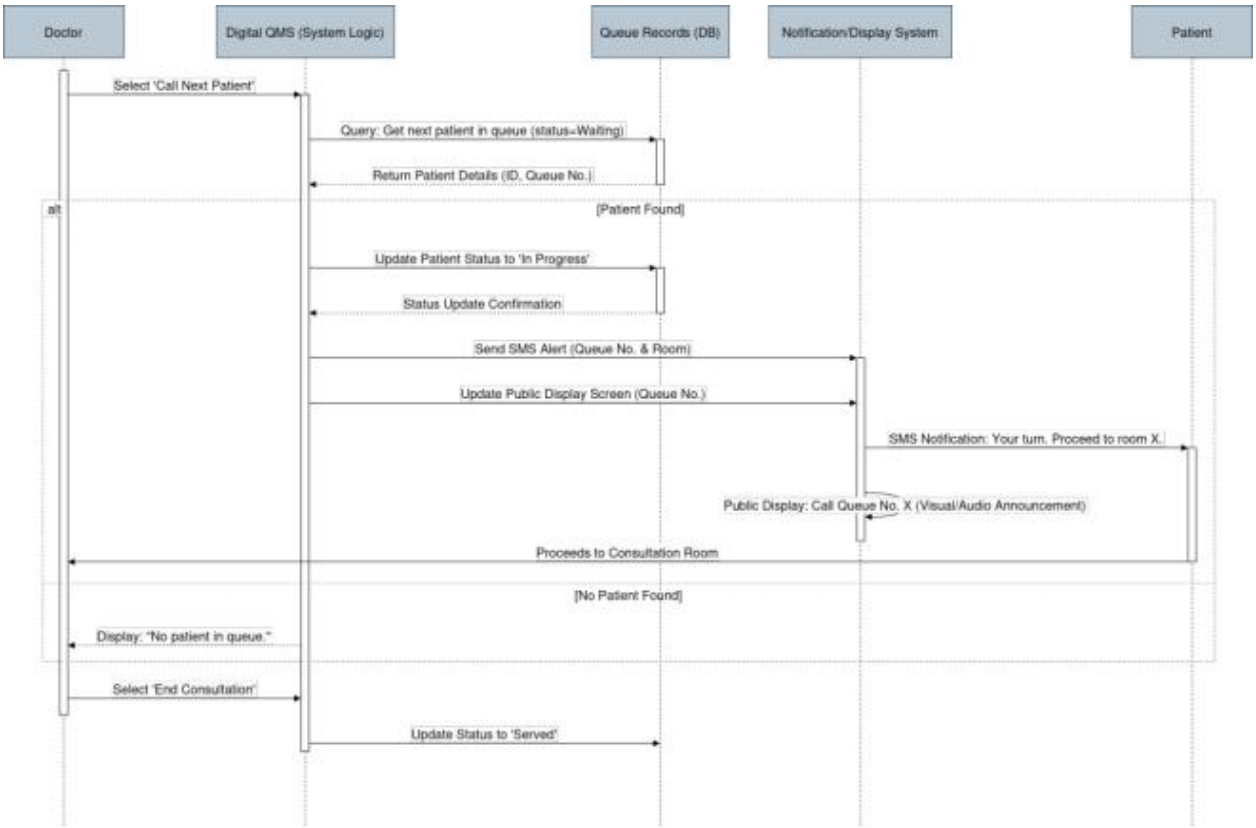

Figure 3.2

Description of Flow:

1. **Initiation:** The Patient provides necessary identification to the Receptionist, initiating the sequence of object interactions.
2. **Verification:** The Receptionist triggers the QMS - Core Services to perform a lookup in the centralized Patient Records (DB).
3. **Conditional Registration:** An alternative flow is executed if the patient is not found (alt fragment). This path includes the collection of required demographic data by the Receptionist and the subsequent secure persistence of a new patient record in the Patient Records (DB).
4. **Service Request:** The Receptionist determines the required service specialty (e.g., Cardiology, General Practice) and requests queue allocation from the QMS.
5. **Assignment & EWT Calculation:** The QMS instructs the Queue Records (DB) to create a new QueueEntry and calculate the Estimated Wait Time (EWT) based on current departmental load, active doctor count, and historical service metrics.
6. **Confirmation:** The final queue identifier and EWT are returned to the QMS, which relays them to the Receptionist for communication to the patient.

3.2.2 State Chart Diagram: Queue Entry Lifecycle

The State Chart Diagram models the behavior and permissible transitions of the core system entity, the **QueueEntry** object. This diagram is essential for defining the integrity of the real-time queue data and ensures that a patient's status can only transition through defined, valid states, aligning with functional requirement 2.3 (Real-Time Status Monitoring).

The lifecycle starts with the initial **Waiting** state upon check-in and concludes with the terminal states of **Complete** or **Cancelled**. The diagram explicitly models the critical events (e.g., Doctor/Nurse Calls Patient, Visit Concluded, Interruption/Reschedule) that trigger state changes and demonstrates the system's resilience by modeling the non-terminal transition from **InProgress** back to **Waiting** to support operational scenarios like temporary patient hold or transfer.

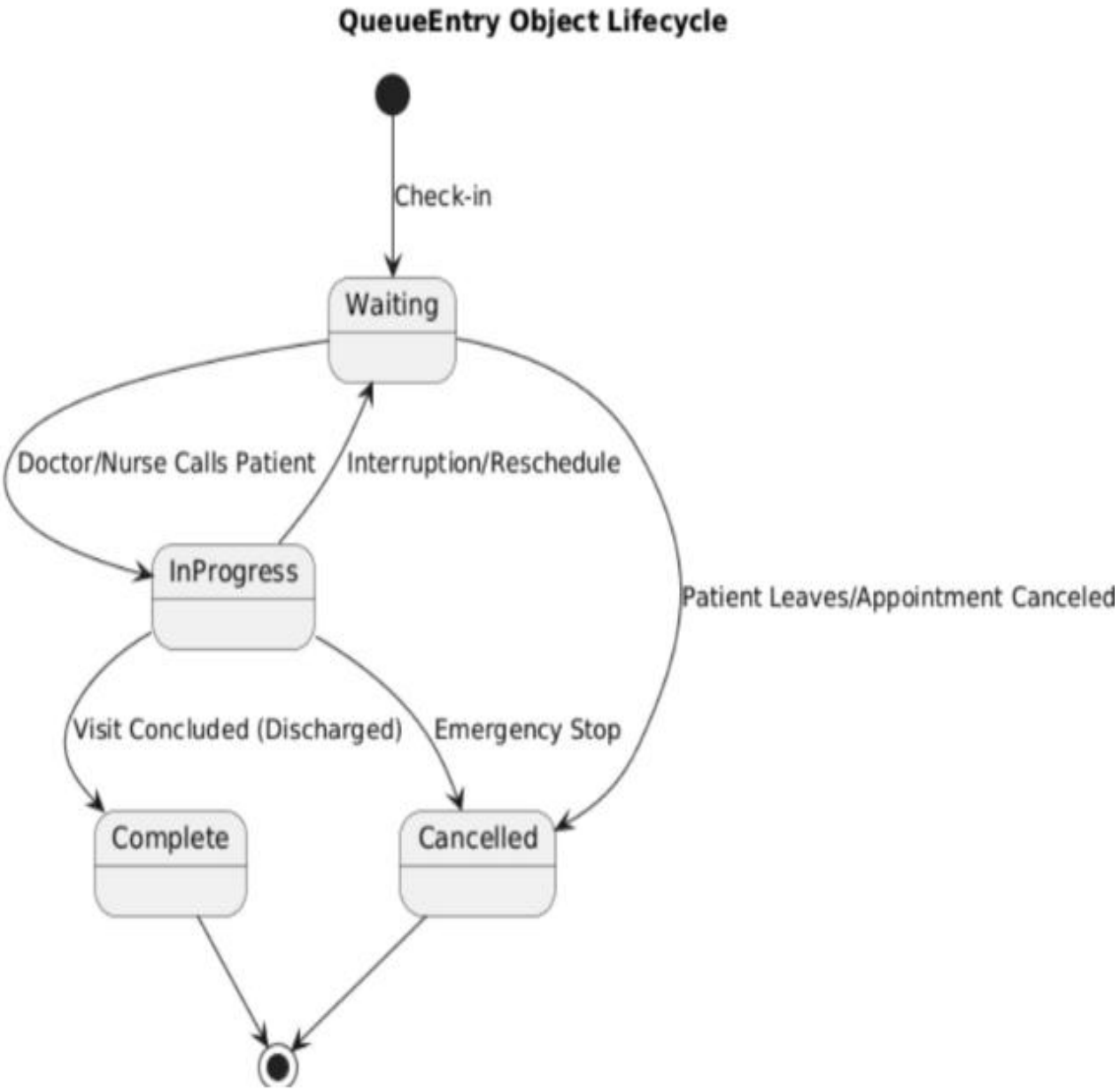
QueueEntry Object Lifecycle
Check-in
Waiting
Doctor/Nurse Calls Patient
Interruption/Reschedule
InProgress
Patient Leaves/Appointment Canceled
Visit Concluded (Discharged)
Emergency Stop
Complete
Cancelled

Figure 3.3

State	Description	Triggering Event(s)
Waiting	The entry is active, the patient is physically waiting, and the entry is available for doctor selection.	Initial Check-in
InProgress	The patient has been called by the Doctor/Nurse and is currently occupying a consultation slot. This state locks the entry from further selection attempts.	Doctor/Nurse Calls Patient
Complete	The consultation or service is successfully finished. This is a terminal state, triggering final archival and service reporting functions.	Visit Concluded (Discharged)
Cancelled	The queue entry has been prematurely terminated due to patient withdrawal, administrative action, or no-show. This is a terminal state.	Patient Leaves/Appointment Canceled, Emergency Stop
Interruption	(Transition from InProgress back to Waiting) Indicates a temporary hold or transfer, making the entry available for re-selection once the patient returns to the waiting area.	Interruption/Reschedule

4. Detailed Design

This section provides the detailed internal design of the major system components identified in the class diagrams, sequence diagrams, and state machine models. Each class is described in terms of its attributes, operations, visibility, constraints, and behavioral responsibilities. The design ensures that all components align with the system's functional requirements and support future scalability and maintainability.

4.1 Class Descriptions

4.1.1 Patient Class

Table 4.1 Patient Class

Attribute	Type	Visibility
PatientID	String	Private
FullName	String	Public
PhoneNumber	String	Public
Department	String	Public
QueueNumber	Integer	Public
Status	String	Public
RegisteredAt	DateTime	Private

Table 4.2 Attribute Description: Patient Class

Attribute	Type	Invariant / Constraint
FullName	String	Must not be null; alphabetic characters only.
PhoneNumber	String	Must be Ethiopian format (09XXXXXXXX or +2519XXXXXXX).
Department	String	Must match valid departments (Cardiology, OPD, Lab, etc.).
QueueNumber	Integer	Unique per patient; auto-generated.
Status	String	Must be one of: *Waiting, Assigned, Completed, and Cancelled*.

Table 4.3 Operation Description: Patient Class

Operation	Visibility	Return	Arguments	Pre-Condition	Post-Condition
registerPatient()	Public	void	FullName, Phone	Patient not previously registered	Patient record created
requestQueueNumber()	Public	Integer	Department	Patient verified	Queue number generated & stored
cancelQueue()	Public	void	QueueNumber	QueueNumber exists & active	Queue status = "Cancelled"

4.1.2 Queue Manager Class

Table 4.4 Queue Manager Class

Attribute	Type	Visibility
QueueList	List	Private
NextQueueNumber	Integer	Private

Table 4.5 Operation Description: QueueManager

Operation	Visibility	Return	Argument	Pre-Condition	Post-Condition
generateQueueNumber()	Public	Integer	Department	Department must exist	New queue number generated
updateQueueStatus()	Public	void	QueueID, Status	QueueID exists	Status updated
assignToDoctor()	Public	void	QueueID, DoctorID	Doctor available	Patient assigned

4.1.3 Doctor Class

Table 4.6 Doctor Class

Attribute	Type	Visibility
DoctorID	String	Private
Name	String	Public
Department	String	Public
AvailabilityStatus	String	Public

Table 4.7 Operation Description: Doctor Class

Operation	Visibility	Return	Pre-Condition	Post-Condition
viewAssignedPatients()	Public	List	Doctor logged in	Assigned list displayed
updatePatientStatus()	Public	void	Patient assigned	Patient status updated to Completed / In-Progress

4.1.4 Admin Class

Table 4.8 Admin Class

Attribute	Type	Visibility
AdminID	String	Private
Name	String	Public
Role	String	Public

Table 4.9 Operation Description: Admin Class

Operation	Visibility	Return	Argument	Pre-Condition	Post-Condition
registerPatient()	Public	void	Patient data	Valid data	Patient added
manageDepartmentQueue()	Public	void	Department	Queue exists	Queue updated
reassignQueue()	Public	void	QueueID, DoctorID	Doctor available	Queue reassigned

4.1.5 Notification Service Class

Table 4.10 Notification Service Class

Attribute	Type	Visibility
SMSGateway	String	Private
MessageTemplate	String	Private

Table 4.11 Operation Description: Notification Service

Operation	Visibility	Return	Argument	Pre-Condition	Post-Condition
sendSMS()	Public	Boolean	Phone, Message	Valid phone number	SMS delivered
pushScreenNotification()	Public	void	QueueID	QueueID active	Real-time update displayed

4.1.6 Authentication Class

Table 4.12 Authentication Class

Attribute	Type	Visibility
Username	String	Public
PasswordHash	String	Private
Role	String	Public

Table 4.13 Operation Description: Authentication

Operation	Visibility	Return	Pre-Condition	Post-Condition
login()	Public	Boolean	Valid credentials	Session created
logout()	Public	void	User logged in	Session destroyed

4.2 System Component Interaction Summary

- **Patient** registers → **QueueManager** generates a queue number.
- **QueueManager** stores queue data in the **Database**.
- When patient is close to their turn:
  - **NotificationService** sends SMS and on-screen alerts.
- **Admin** manages queue flow:
  - Cancels requests.
  - Reassigns patients to available doctors.
- **Doctors** update patient status via **Doctor Dashboard**.

4.3 Compliance with EMR Standards

- Modular architecture supports future EMR integration.
- Data exchange supports HL7 / FHIR for interoperability.
- Secure login, RBAC, and encrypted communication meet Ethiopian health IT regulations.

References

1. Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.

2. Pressman, R. S., & Maxim, B. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill.

3. Health Level Seven International. (2023). *HL7 FHIR Release 4.0.1 Specification*. Retrieved from the HL7 Official Documentation.

4. Ministry of Health Ethiopia. (2023). *Ewket EMR Integration and Interoperability Guidelines*. Addis Ababa: MoH.

5. IEEE Computer Society. (2009). *IEEE Std 1016-2009 – IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*.