Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the core implementation details for **MedSync - Secure Healthcare Appointment Platform**. MedSync enables secure appointment booking, user authentication, role-based access, medical record uploads, and real-time notifications, with analytics and user management for admins. The system leverages Next.js, React, Node.js, Express, PostgreSQL, MongoDB, JWT, and Multer.

---

1. System Components

Component	Technology	Key Responsibilities
Frontend	Next.js, React, TS, Tailwind	UI, user flows, dashboard, notifications
API Server	Node.js, Express, TS	RESTful APIs, business logic, auth, file handling
Auth Service	JWT	User authentication, token issuance/validation
Relational DB	PostgreSQL	Users, appointments, analytics
NoSQL DB	MongoDB	Medical records, file metadata
File Storage	Multer (local/cloud)	Secure file uploads/downloads
Notification Svc	WebSockets/Push	Real-time notifications
Admin Dashboard	Next.js, React	Analytics, user/appointment management

---

2. Class/Interface Overview

Class/Interface	Description	Key Methods/Attributes
`UserController`	User registration, login, profile	`register()`, `login()`, `getProfile()`
`AppointmentController`	CRUD for appointments	`create()`, `update()`, `cancel()`, `listByUser()`
`AuthMiddleware`	JWT validation, role checks	`verifyToken()`, `requireRole(role)`
`FileController`	Medical record upload/download	`upload()`, `download()`, `listFiles()`
`NotificationService`	Real-time event push	`sendToUser()`, `broadcast()`
`AdminController`	Analytics, user mgmt	`getStats()`, `listUsers()`, `banUser()`

Relationships:

- Controllers use services (e.g., Auth, Notification).
- Middleware protects routes.
- Data models map to DB schemas.

---

3. Data Structure Overview

Model	Storage	Key Fields
User	PostgreSQL	id, name, email, passwordHash, role, status
Appointment	PostgreSQL	id, patientId, doctorId, datetime, status
MedicalRecord	MongoDB	_id, userId, fileUrl, fileType, uploadedAt
Notification	MongoDB	_id, userId, message, type, read, createdAt
Analytics	PostgreSQL	id, metric, value, timestamp

---

4. Algorithms/Logic

User Authentication (JWT):

```
POST /login:
  - Validate credentials
  - If valid: issue JWT (userId, role, exp)
  - Return token
Middleware verifyToken:
  - Extract JWT from header
  - Verify signature & expiry
  - Attach user info to request
```

Appointment Booking:

```
POST /appointments:
  - Auth: require patient role
  - Validate doctor availability
  - Create appointment (PostgreSQL)
  - Notify doctor (NotificationService)
```

File Upload (Medical Records):

```
POST /records/upload:
  - Auth: require user
  - Multer handles file upload
  - Store file metadata in MongoDB
  - Return file reference
```

---

5. Error Handling

Scenario	Handling Approach
Invalid credentials/auth failure	401 Unauthorized, error message
Access denied (role)	403 Forbidden, error message
Resource not found	404 Not Found, error message
Validation errors	400 Bad Request, details in response
File upload errors	400/500, log error, user-friendly message
DB/Service unavailable	503 Service Unavailable, retry logic/logging

---

End of Document