# 🏥 **MediWay – Smart Health Management System**

### Updated Technical & Implementation Architecture (Spring Boot + React)

## ****1) Backend Folder Structure****

com.mediway.backend

├── MediWayBackendApplication.java

├── config/

│ ├── SecurityConfig.java

│ ├── JwtAuthenticationFilter.java

│ └── CorsConfig.java

├── controller/

│ ├── AuthController.java

│ ├── UserController.java

│ ├── AppointmentController.java

│ ├── PaymentController.java

│ ├── ReportController.java

│ └── MedicalRecordController.java

├── dto/

│ ├── request/

│ │ ├── LoginRequest.java

│ │ ├── AppointmentRequest.java

│ │ ├── PaymentRequest.java

│ │ ├── ReportRequest.java

│ │ └── MedicalRecordRequest.java

│ └── response/

│ ├── AuthResponse.java

│ ├── AppointmentResponse.java

│ ├── PaymentResponse.java

│ ├── ReportResponse.java

│ └── MedicalRecordResponse.java

├── entity/

│ ├── User.java

│ ├── Patient.java

│ ├── Doctor.java

│ ├── Appointment.java

│ ├── Payment.java

│ ├── Report.java

│ └── MedicalRecord.java

├── repository/

│ ├── UserRepository.java

│ ├── AppointmentRepository.java

│ ├── PaymentRepository.java

│ ├── ReportRepository.java

│ └── MedicalRecordRepository.java

├── service/

│ ├── AuthService.java

│ ├── AppointmentService.java

│ ├── PaymentService.java

│ ├── ReportService.java

│ └── MedicalRecordService.java

├── security/

│ ├── JwtUtil.java

│ └── UserDetailsServiceImpl.java

└── exception/

├── GlobalExceptionHandler.java

└── ResourceNotFoundException.java

## ****2) High-Level Technical Architecture (Spring Boot Stack)****

* **Frontend:** React (web) — consumes REST APIs, consistent with wireframes/storyboards.
* **Backend:** Java Spring Boot (REST API)
  + Spring Boot 3.x
  + Spring Web (REST Controllers)
  + Spring Data JPA (Hibernate)
  + PostgreSQL (primary relational DB)
  + Spring Security (JWT-based authentication)
  + Swagger / Springdoc OpenAPI for documentation
  + JUnit + Mockito + MockMvc for testing
  + Flyway or Liquibase for DB migrations
* **External Integrations:**
  + Payment sandbox (Stripe/PayHere test mode)
  + Optional email notification service (SendGrid/Mailgun)
* **Hosting / CI/CD:**
  + GitHub Actions for CI
  + Deploy backend on Render / Railway / Heroku / AWS Elastic Beanstalk

## ****3) Database Design (ERD Summary)****

### ****Primary Tables (PK = Primary Key, FK = Foreign Key)****

1. **patients**
   * patient\_id (UUID, PK)
   * full\_name
   * email (unique)
   * phone
   * dob (date)
   * gender
   * address
   * height\_cm
   * weight\_kg
   * created\_at, updated\_at
2. **doctors**
   * doctor\_id (UUID, PK)
   * full\_name
   * specialization
   * contact
   * availability (JSON / separate schedule table)
3. **doctor\_schedules**
   * schedule\_id (PK)
   * doctor\_id (FK → doctors)
   * date (date)
   * time\_slot (e.g., "09:00–09:30")
   * slot\_status (available/booked)
4. **appointments**
   * appointment\_id (UUID, PK)
   * patient\_id (FK → patients)
   * doctor\_id (FK → doctors)
   * schedule\_id (FK → doctor\_schedules)
   * status (BOOKED / CANCELLED / COMPLETED)
   * reason
   * created\_at
5. **payments**
   * payment\_id (UUID, PK)
   * appointment\_id (FK → appointments)
   * amount (decimal)
   * method (CARD / INSURANCE / CASH)
   * transaction\_id
   * status (PENDING / SUCCESS / FAILED)
   * paid\_at
6. **medical\_records**
   * record\_id (UUID, PK)
   * patient\_id (FK → patients)
   * doctor\_id (FK → doctors)
   * diagnosis
   * medications
   * notes
   * created\_at
   * updated\_at
7. **reports (metadata)**
   * report\_id (UUID, PK)
   * generated\_by (admin\_id)
   * report\_type
   * filters (JSON)
   * generated\_at
   * file\_path
8. **admins**
   * admin\_id (PK)
   * name
   * email
   * password\_hash
   * role

### ****Relationships****

* patients 1..\* → appointments
* doctors 1..\* → doctor\_schedules
* appointments 1..1 → payments
* patients 1..\* → medical\_records
* doctors 1..\* → medical\_records
* admins 1..\* → reports

### ****Indexes****

* index on patients.email
* index on doctor\_schedules(doctor\_id, date)
* index on appointments(patient\_id, doctor\_id, status)
* index on medical\_records(patient\_id, doctor\_id)

## ****4) REST API Endpoints (Spring Boot Controllers)****

### ****Appointments (Use Case A – Scheduling)****

* POST /api/appointments
  + Request: patientId, doctorId, scheduleId, reason
  + Behavior: check slot availability → create appointment → mark schedule as booked
* PUT /api/appointments/{appointmentId}
  + Update or cancel appointment (status change)
* GET /api/appointments/patient/{patientId}
  + List patient’s appointments
* GET /api/doctors
  + List doctors by specialization
* GET /api/doctors/{id}/schedules
  + Retrieve available slots

### ****Payments (Use Case B – Payment Handling)****

* GET /api/payments/patient/{id} — list unpaid bills
* POST /api/payments — process payment (card/insurance)
* GET /api/payments/{id}/receipt — view receipt
* PUT /api/payments/{id}/refund — optional refund endpoint

### ****Reports (Use Case C – Statistical Reports)****

* POST /api/reports/generate
  + Accept filters → generate PDF/CSV report → save metadata
* GET /api/reports
  + List all generated reports
* GET /api/reports/{id}/download
  + Download selected report file

### ****Medical Records (Use Case D – Manage Patient Medical Records)****

* GET /api/patients/{id}/records — view patient’s medical history
* POST /api/patients/{id}/records — add new record (diagnosis, prescriptions, notes)
* PUT /api/records/{recordId} — update existing record
* GET /api/records/{recordId} — view detailed record
* GET /api/patients/search?query= — search by patient ID or QR code

## ****5) Suggested Improvements to Original Design (for Group Report)****

1. **Separate doctor\_schedules table** → Prevents double-booking and enables accurate concurrency control.
2. **Add medical\_records table** → Enables healthcare domain depth and better data traceability.
3. **Normalize payments table** → Ensures safe retries and idempotent payment processing.
4. **Store report filters as JSON** → Allows reproducible hospital analytics.
5. **Introduce audit logging** for medical records → Tracks who modified what and when.

## ****6) Division of Work – Team Allocation****

Each member implements one substantial business use case with:

* Complete CRUD functionality
* Meaningful tests (≥80% coverage)
* Consistent UI + documentation + database migrations

| **Member** | **Use Case** | **Key Responsibilities** |
| --- | --- | --- |
| **Shalon Fernando** | Appointment Scheduling | Doctor scheduling, booking logic, concurrency control |
| **Shirantha** | Statistical Reports | Generate PDF/CSV hospital insights and analytics |
| **Navodya** | Manage Patient Medical Records | CRUD operations for diagnoses, treatments, and prescriptions |
| **Nipuni** | Payment Handling | Integrate payment gateway sandbox, handle transactions, generate receipts |

## ****7) Implementation & Testing Guidelines****

* **Project Structure:**
* src/main/java/com/mediwey
* /config
* /controller
* /service
* /repository
* /entity
* /dto
* /exception
* src/test/java/...
* **Database Migrations:**
  + Use Flyway. Each member adds migration for their tables.
* **Validation:**
  + Use @Valid annotations in DTOs and request payloads.
* **Transactions:**
  + Annotate service methods with @Transactional.
* **Testing Tools:**
  + @DataJpaTest for repository
  + @SpringBootTest and MockMvc for controllers
  + Mockito for service logic
* **Coverage Target:**
  + ≥80% per individual use case (Jacoco report)
* **API Documentation:**
  + Auto-generate with Swagger (Springdoc OpenAPI)
* **Error Handling:**
  + Centralized @ControllerAdvice for exceptions
* **Security:**
  + Minimal JWT authentication (for doctor/staff/admin roles)

## ****8) Collaboration Workflow****

1. Create branches per feature:  
   feature/<member>-<usecase>
2. Push commits incrementally (entity → service → controller → test).
3. Each member opens one PR for their module.
4. Peer-review each other’s PRs before merging into dev.
5. Integrate and test combined system before demo.

Each PR must include:

* Endpoints implemented
* Migration files
* Test coverage screenshots
* Postman collection / sample curl commands

## ****9) Submission Checklist (Aligned with Assignment Rubric)****

| **Deliverable** | **Description** |
| --- | --- |
| **Group Report** | Critique + suggested improvements + updated UML diagrams + UI designs |
| **Individual Code** | One complete use case per member, with unit/integration tests |
| **Test Coverage** | Jacoco HTML report showing ≥80% coverage for implemented module |
| **Demo Video / Screenshots** | Show the working use case end-to-end |
| **API Documentation** | Generated Swagger or Postman collection |
| **Database Scripts** | Flyway migration files for all entities |