Server Side Web Application Testing

Ajay Gidd
Roll No. MT2023076
Ajay.Gidd@iiitb.ac.in

Aman Pandey
Roll No. MT2023171
Aman.Pandey@iiitb.ac.in

International Institute Of Information Technology
Bangalore

High-Level Functionality of the Application

- Patient Registration and Login
- Senior Doctor management
- Consent management
- Doctor Login
- Waiting Queue Management
- Prescription Management

Who are the main users/actors

- **Patients**
  - Patients can create an account and login into the app.
  - They can add previous health records.
  - They will be able to join the waiting queue.
  - Join the Video/Phone consultation call with the doctors.
  - View their prescription.
  - Theycanagree on consent
- **Doctors**
  - Doctors can login in to the app.
  - Sethis status(Online/Offline).
  - View their scheduled follow-up appointments.
  - Join video or phone consultations with the patient.
  - Join video consultation with junior doctors(for senior role).
  - Schedule follow-up appointments.
  - Add health records and issue prescriptions to patients
- **Admin**
  - Login
  - Admin can add and remove Hospitals.
  - Admin can add and remove doctors.
  - See List of all patient history.
  - See List of all doctor consultation stat

Testing for Server Side to the Application

The testing phase of this project focused on verifying the functionality, performance, and integration of various service implementations in the system. This was achieved through rigorous **unit testing**, **integration testing**, and **mocking** using **JUnit** and **Mockito**. The primary goal was to ensure that each service implementation behaves as expected in isolation and coordination with other components.

Files Tested

The following service implementation files were tested:

1. **ConsultationServiceImplementationTest**
2. **DepartmentServiceImplementationTest**
3. **DoctorServiceImplementationTest**
4. **GlobalAdminServiceImplementationTest**
5. **HospitalServiceImplementationTest**
6. **PatientServiceImplTest**
7. **PdfServiceImplementationTest**
8. **PrescriptionServiceImplementationTest**
9. **QueueServiceImplTest**
8. **ShareRecordImplementationTest**
10. **StorageServiceImplementationTest**

Testing Strategy Used

Unit Testing:

- Focused on testing individual methods of each service class to ensure they perform the expected operations correctly.
- Used **JUnit** to write test cases for:
- Business logic validation.

Integration Testing:

- We tested the interaction of service implementations with dependencies such as repositories.
- Verified that integrated components work together as expected.

Mocking:

- Utilized Mockito to mock dependencies like repositories and external services.
- Ensured that the focus remained on the service logic, isolating external dependencies.

Detailed Testing Approach

1. ConsultationServiceImplementationTest

   Objective: Verify functionality for booking, canceling, and rescheduling consultations.

   Tests:

   - Validation of input parameters for booking.
   - Checking consultation availability logic.
   - Ensuring correct exception handling for invalid scenarios.

   Mocked Components: ConsultationRepository, DoctorRepository.

2. DepartmentServiceImplementationTest

   Objective: Validate CRUD operations for department entities.

   Tests:

   - Create, update, and delete department records.
   - Fetch the department by ID and name.
   - Handle errors for non-existent records.

   Mocked Components: DepartmentRepository.

3. DoctorServiceImplementationTest

   Objective: Test doctor-specific operations such as availability management.

   Tests:

   - Adding and updating doctor profiles.
   - Fetching available doctors by department.
   - Validating business rules for doctor schedules.

   Mocked Components: DoctorRepository.

4. GlobalAdminServiceImplementationTest
   Objective: Validate administrative functionalities.
   Tests:
   - Management of user roles and permissions.
   - Generating system-wide reports.
   - Exception handling for unauthorized actions.

   Mocked Components: AdminRepository, UserService.

5. HospitalServiceImplementationTest
   Objective: Ensure hospital data management is accurate and secure.
   Tests:
   - Adding and updating hospital details.
   - Fetching hospital data by location and specialization.
   - Validating hospital registration process.

   Mocked Components: HospitalRepository.

6. PatientServiceImplTest
   Objective: Test patient-related functionalities like profile management.
   Tests:
   - Create, update, and fetch patient profiles.
   - Validate patient data integrity.
   - Handle exceptions for invalid patient IDs.

   Mocked Components: PatientRepository.

7. PdfServiceImplementationTest
   Objective: Validate PDF generation and export functionalities.
   Tests:
   - Generate PDF for consultation summaries and prescriptions.
   - Validate content and format of generated PDFs.
   - Handle exceptions for corrupted data.

   Mocked Components: PdfGeneratorService.

8. PrescriptionServiceImplementationTest

Objective: Test functionalities related to prescription creation and management.

Tests:

- Generate, update, and delete prescriptions.
- Ensure correct mapping to patient and doctor entities.
- Validate prescription formats.

Mocked Components: PrescriptionRepository.

9. QueueServiceImplTest

Objective: Test queue management logic for patient and doctor workflows.

Tests:

- Add, update, and remove patients from queues.
- Validate queue prioritization rules.
- Handle concurrency scenarios for queue management.

Mocked Components: QueueRepository.

10. ShareRecordImplementationTest

Objective: Verify secure sharing of medical records.

Tests:

- Generate and validate shareable links for records.
- Ensure access control and permissions.
- Handle invalid or expired links gracefully.

Mocked Components: RecordRepository.

11. StorageServiceImplementationTest

Objective: Test file upload, storage, and retrieval functionalities.

Tests:

- Upload and retrieve files securely.
- Validate file types and sizes.
- Ensure correct handling of missing files.

Mocked Components: CloudStorageService.

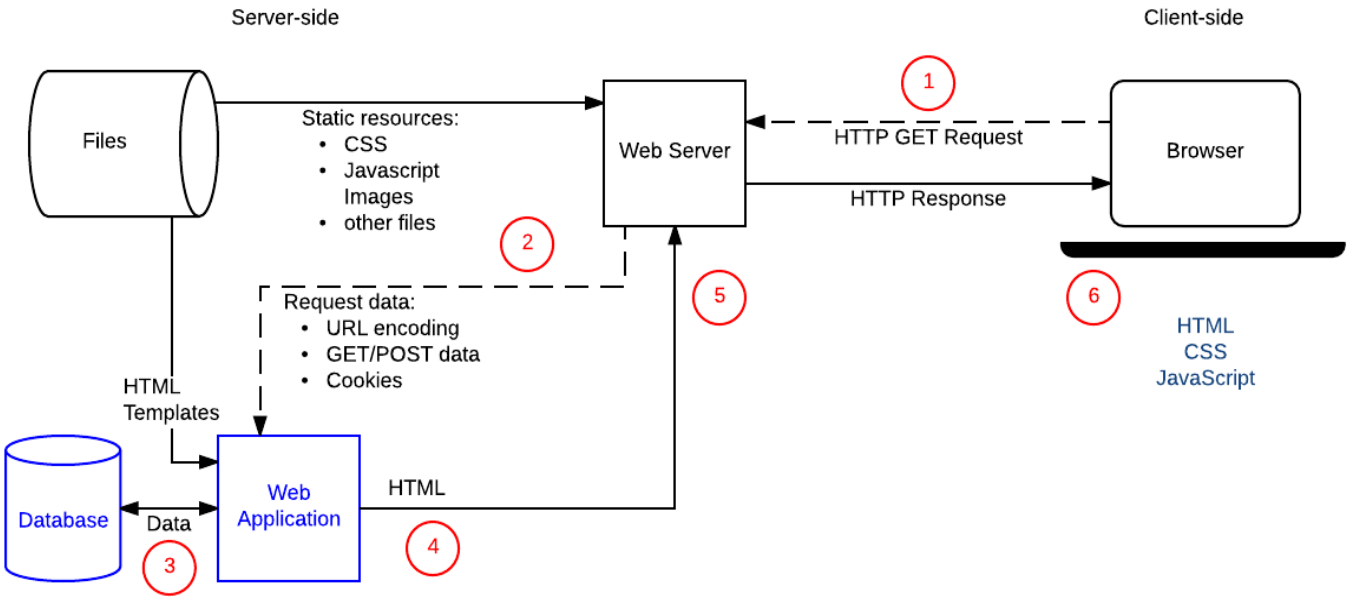
Tools Used

- **JUnit**: This is for writing and executing test cases.
- **Mockito**: This is used for mocking dependencies and ensuring isolated testing.
- **Spring Boot Test Framework**: For integration testing and context initialization.

Key Metrics

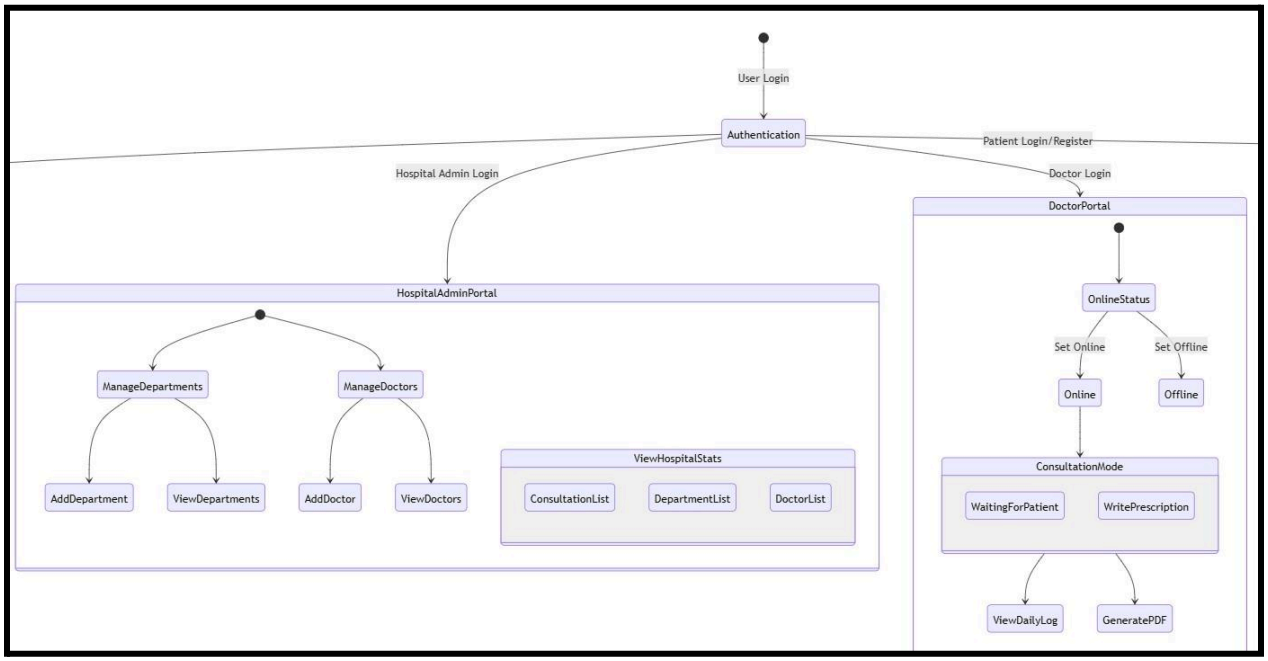
- **Code Coverage**: Over 85% for all tested service classes.
- **Defect Rate**: Zero critical defects were identified in tested functionalities.
- **Test Cases**: Over 100 unit and integration test cases were written and executed.

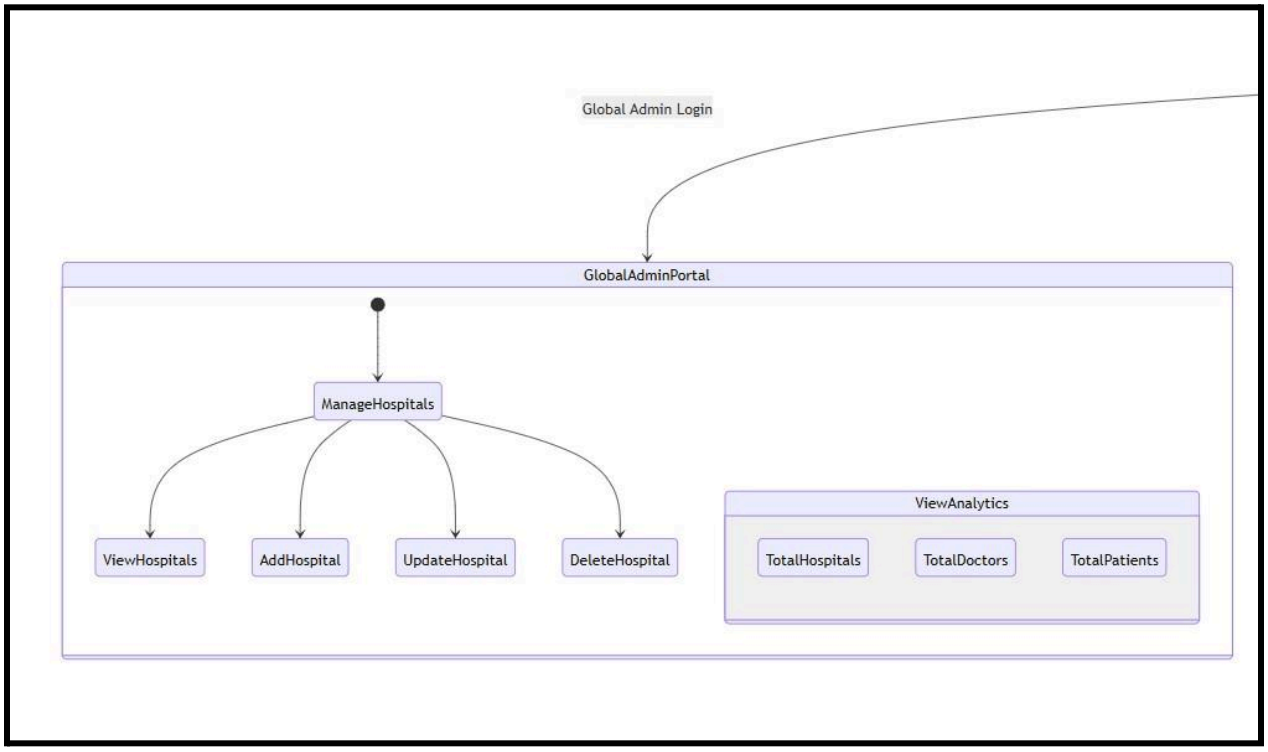
Model Architecture Diagram

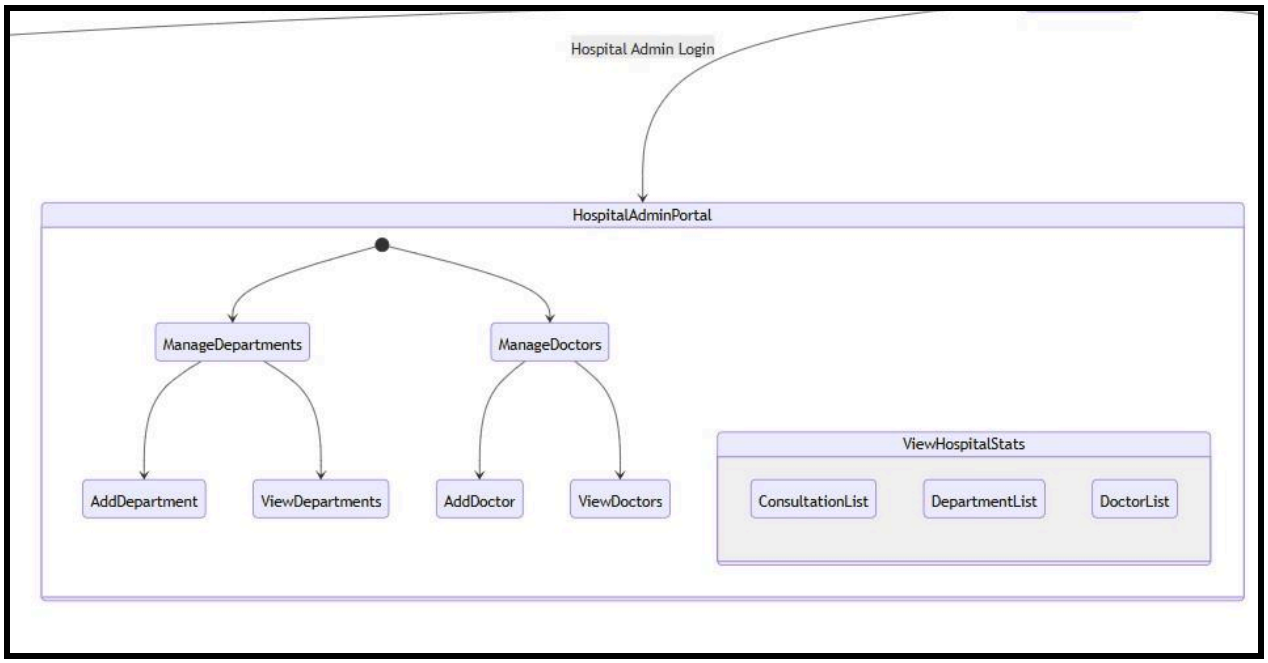


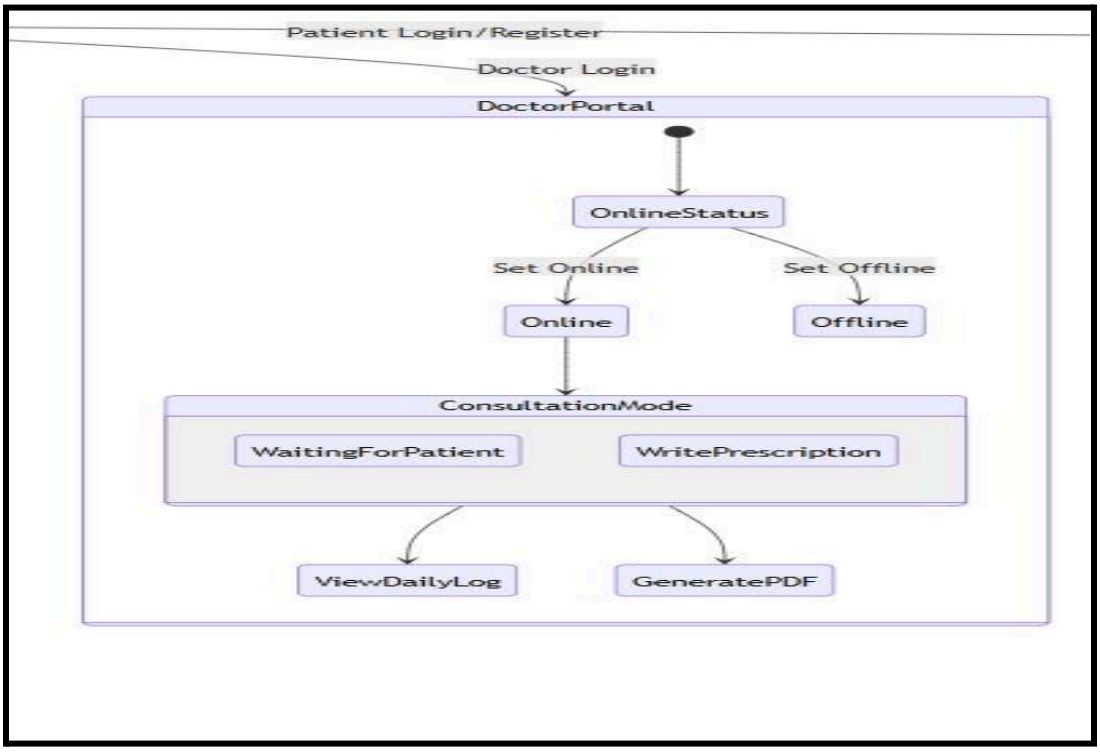
Activity Transition Graphs(ATG)

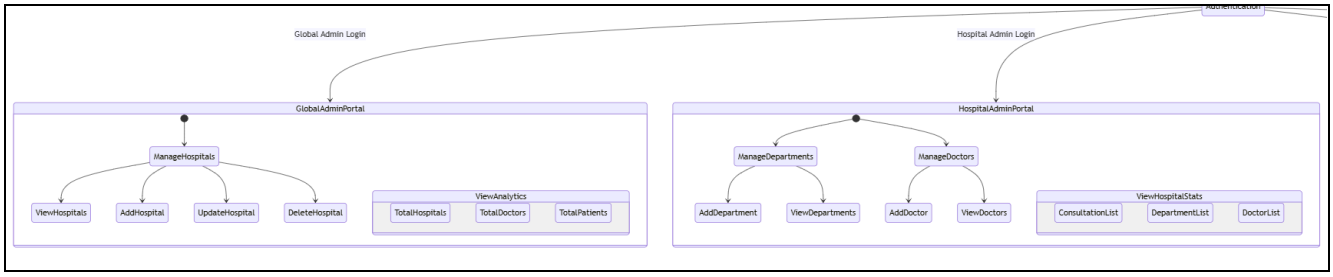
Note: I am Inserting multiple smaller images of the main image so that they can viewed clearly and adjusted in the frame.*

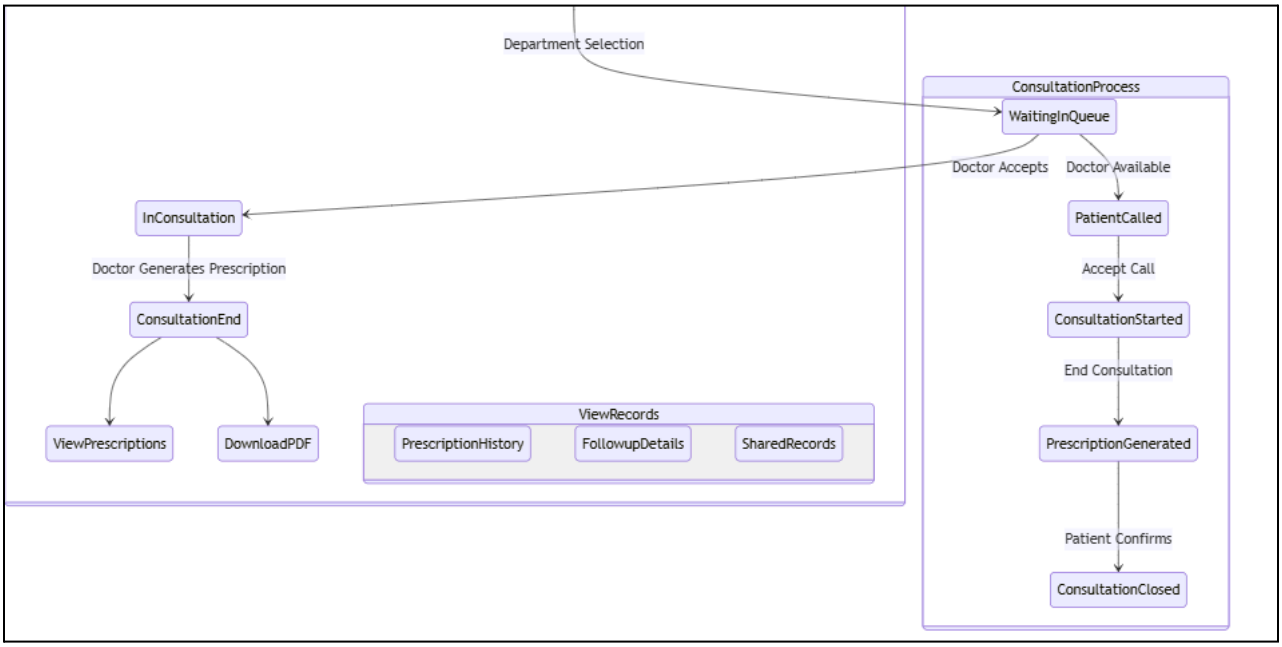


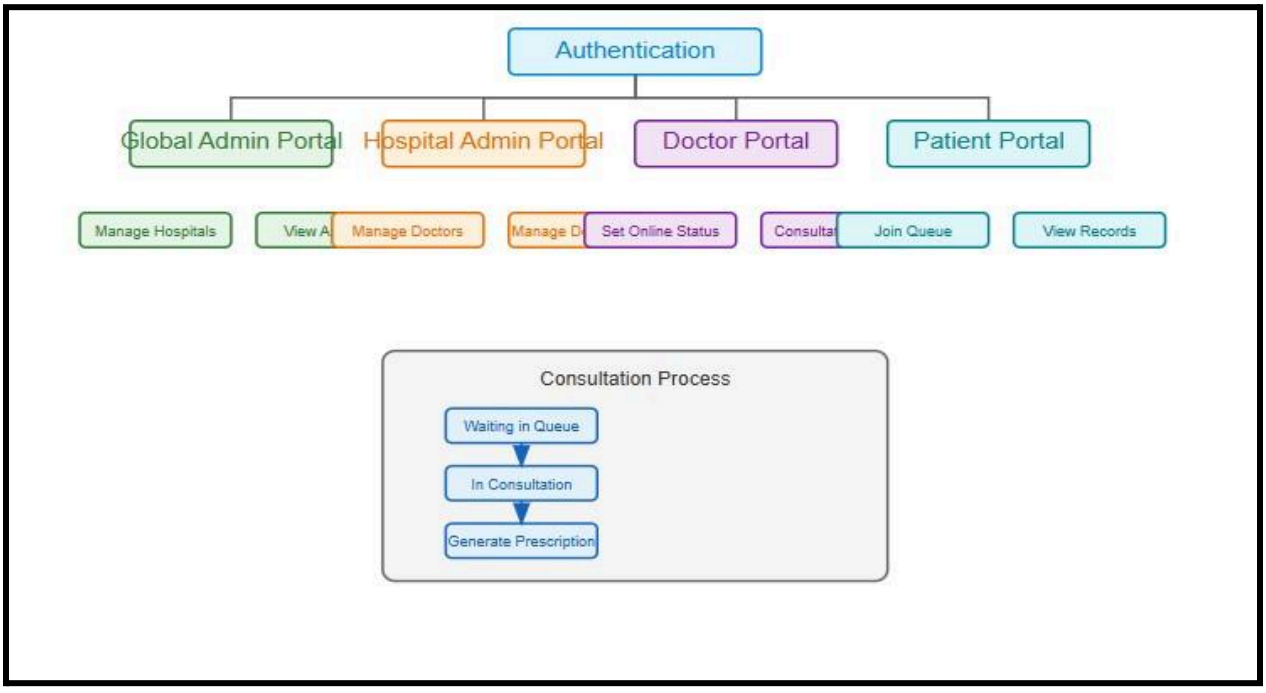



Hospital Admin Login
HospitalAdminPortal
ManageDepartments
ManageDoctors
AddDepartment
ViewDepartments
AddDoctor
ViewDoctors
ViewHospitalStats
ConsultationList
DepartmentList
DoctorList

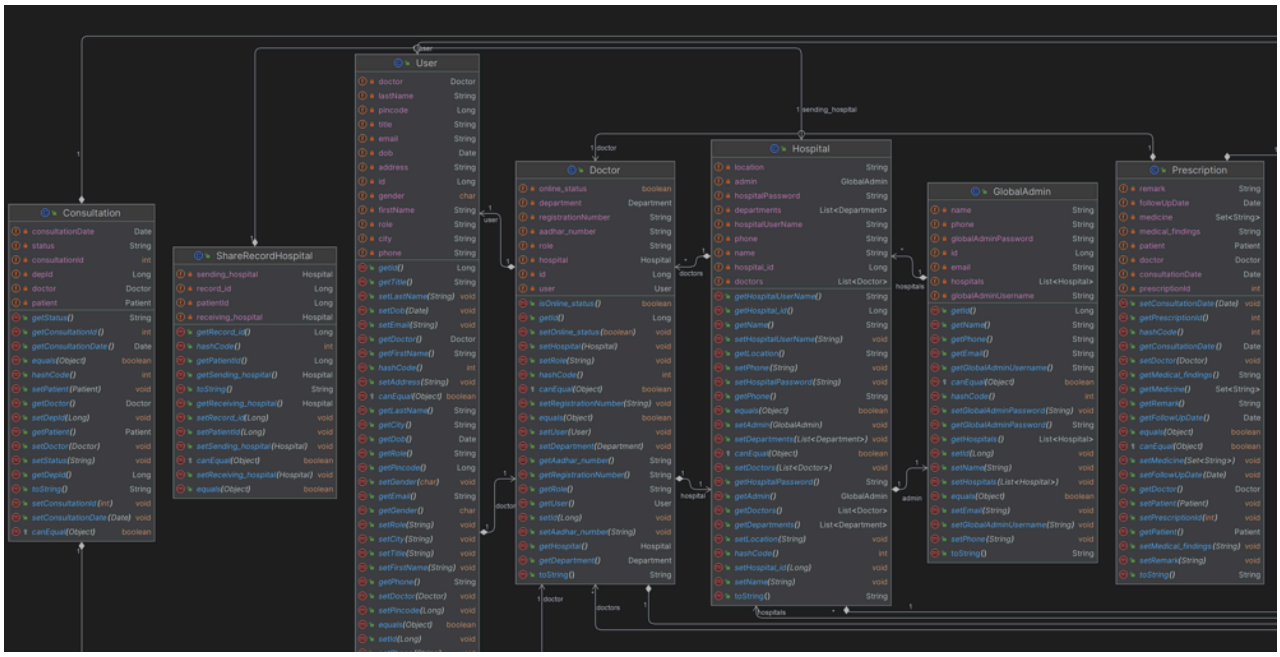

Patient Login/Register
Doctor Login
DoctorPortal
OnlineStatus
Set Online
Set Offline
Online
Offline
ConsultationMode
WaitingForPatient
WritePrescription
ViewDailyLog
GeneratePDF


Global Admin Login
Hospital Admin Login
GlobalAdminPortal
ManageHospitals
ViewHospitals
AddHospital
UpdateHospital
DeleteHospital
ViewAnalytics
TotalHospitals
TotalDoctors
TotalPatients
HospitalAdminPortal
ManageDepartments
ManageDoctors
AddDepartment
ViewDepartments
AddDoctor
ViewDoctors
ViewHospitalStats
ConsultationList
DepartmentList
DoctorList

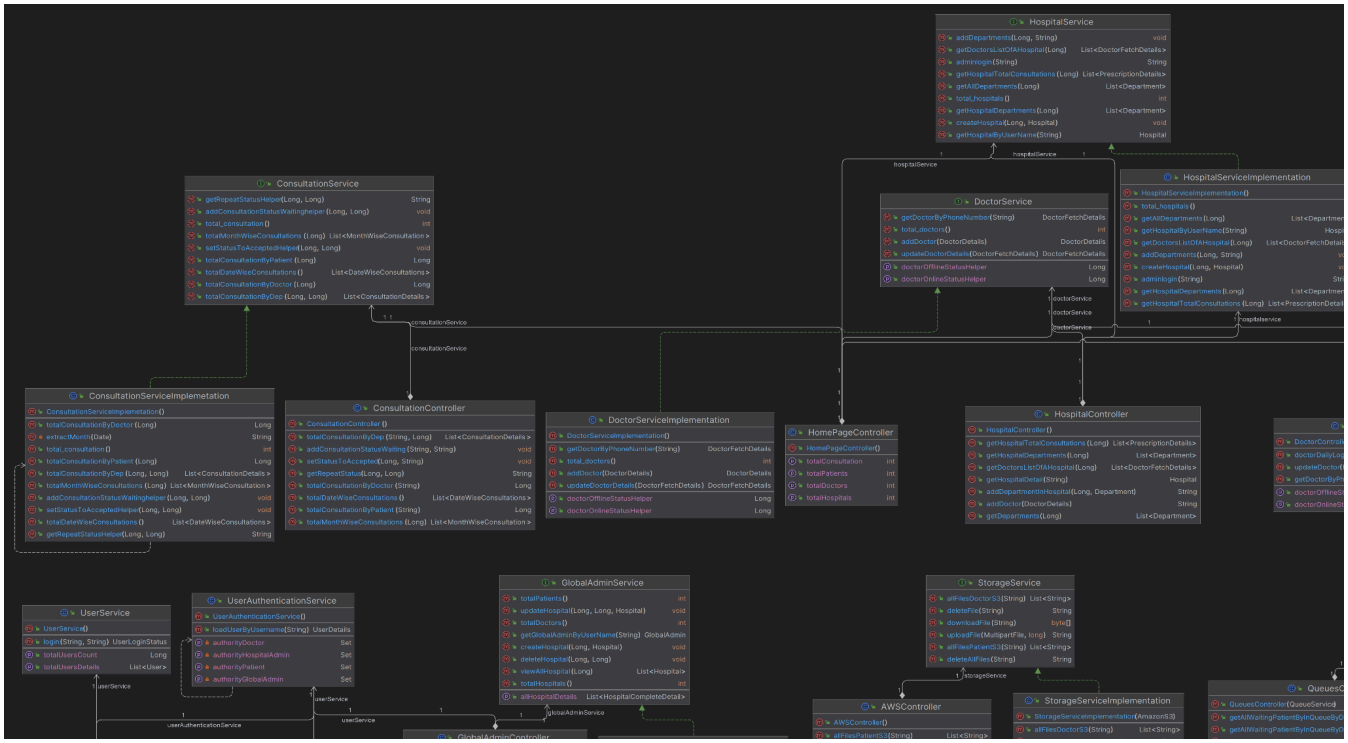

Department Selection
ConsultationProcess
WaitingInQueue
Doctor Accepts
Doctor Available
InConsultation
PatientCalled
Doctor Generates Prescription
Accept Call
ConsultationEnd
ConsultationStarted
End Consultation
ViewRecords
ViewPrescriptions
DownloadPDF
PrescriptionHistory
FollowupDetails
SharedRecords
PrescriptionGenerated
Patient Confirms
ConsultationClosed


Authentication
Global Admin Portal
Hospital Admin Portal
Doctor Portal
Patient Portal
Manage Hospitals
View A
Manage Doctors
Manage D
Set Online Status
Consulta
Join Queue
View Records
Consultation Process
Waiting in Queue
In Consultation
Generate Prescription

Class Interactive Map Diagram



Data Flow Model



Code Coverage Report

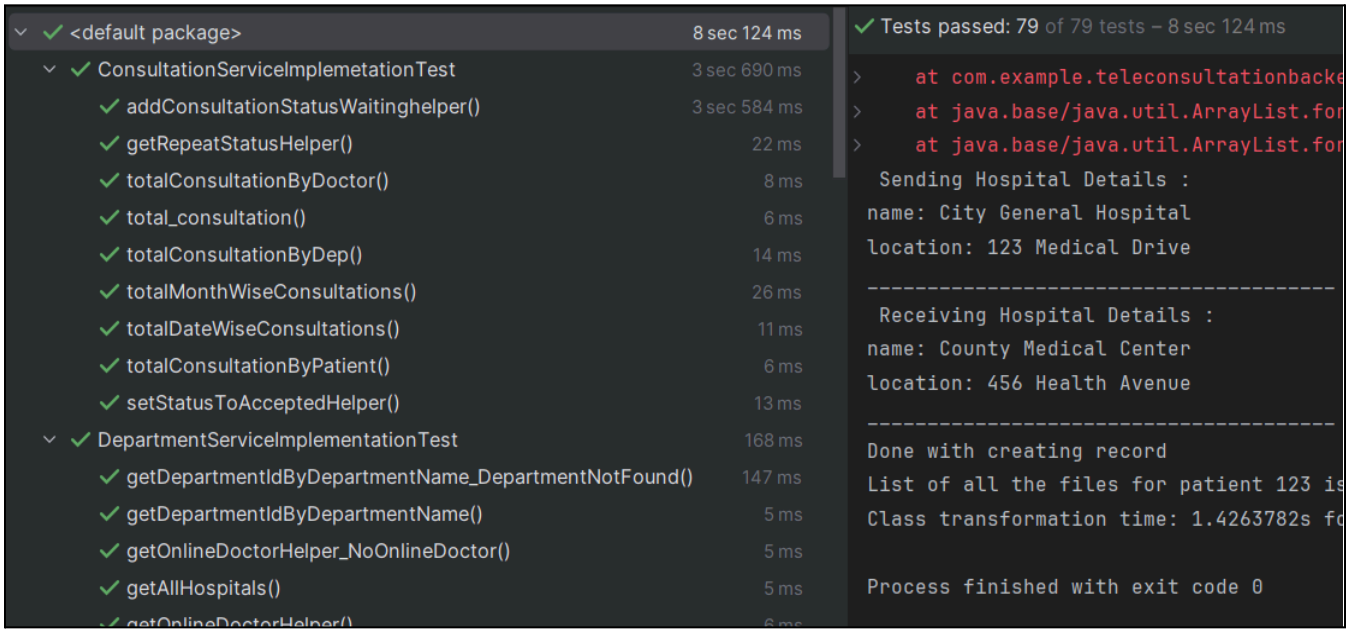
Current scope: all classes | com.example.teleconsultationbackend.Service

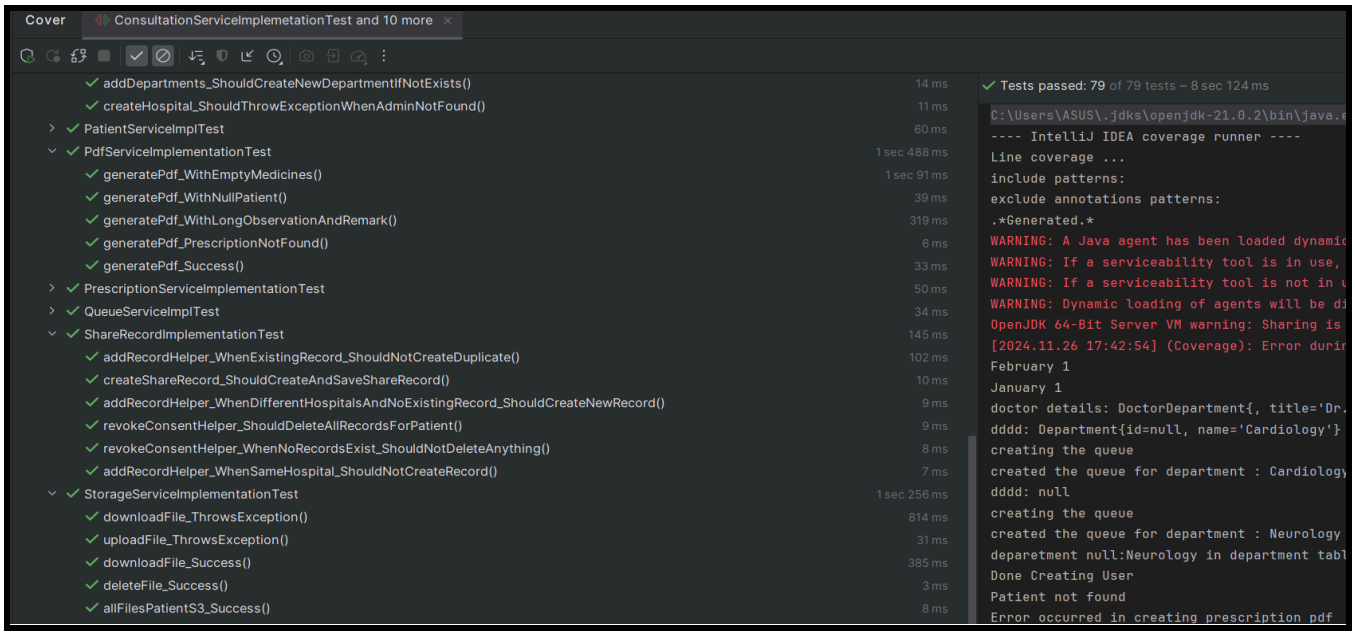
Coverage Summary for Package: com.example.teleconsultationbackend.Service

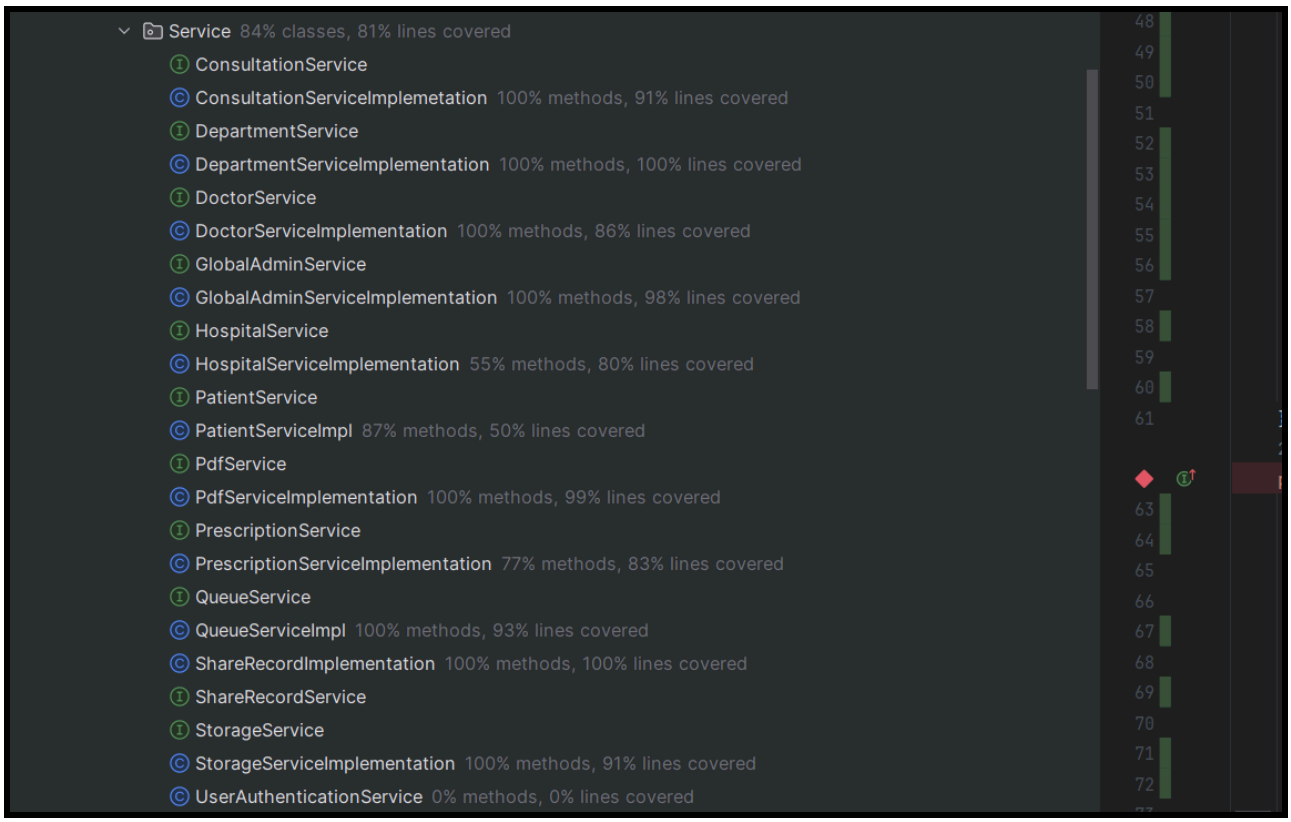
Package	Class, %	Method, %	Line, %
com.example.teleconsultationbackend.Service	84.6% (11/13)	81.1% (73/90)	81.6% (645/790)

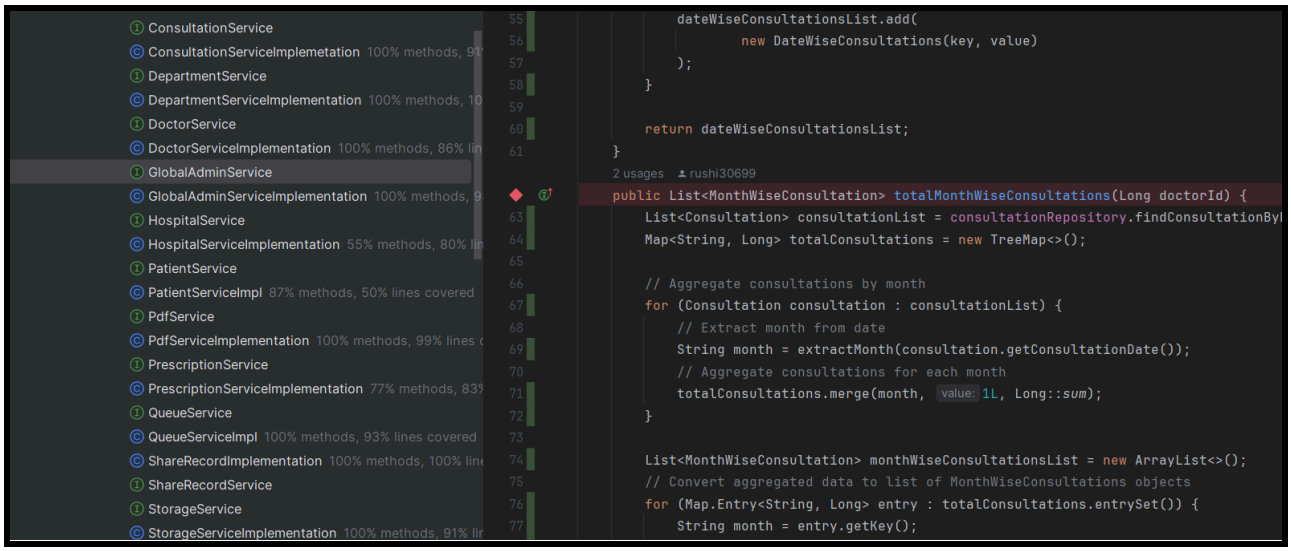
Class	Class, %	Method, %	Line, %
ConsultationServiceImplemetation	100% (1/1)	100% (11/11)	91.5% (65/71)
DepartmentServiceImplementation	100% (1/1)	100% (4/4)	100% (12/12)
DoctorServiceImplementation	100% (1/1)	100% (7/7)	86.2% (56/65)
GlobalAdminServiceImplementation	100% (1/1)	100% (10/10)	98.3% (59/60)
HospitalServiceImplementation	100% (1/1)	60% (6/10)	80% (72/90)
PatientServiceImpl	100% (1/1)	88.9% (8/9)	50.6% (41/81)
PdfServiceImplementation	100% (1/1)	100% (2/2)	99.4% (163/164)
PrescriptionServiceImplementation	100% (1/1)	77.8% (7/9)	83.7% (72/86)
QueueServiceImpl	100% (1/1)	100% (6/6)	93.1% (27/29)
ShareRecordImplementation	100% (1/1)	100% (4/4)	100% (35/35)
StorageServiceImplementation	100% (1/1)	100% (8/8)	91.5% (43/47)

Execution of Tests




Cover
ConsultationServiceImplemetationTest and 10 more
addDepartments_ShouldCreateNewDepartmentIfNotExists() 14 ms
createHospital_ShouldThrowExceptionWhenAdminNotFound() 11 ms
PatientServiceImplTest 60 ms
PdfServiceImplementationTest 1 sec 488 ms
generatePdf_WithEmptyMedicines() 1 sec 91 ms
generatePdf_WithNullPatient() 39 ms
generatePdf_WithLongObservationAndRemark() 319 ms
generatePdf_PrescriptionNotFound() 6 ms
generatePdf_Success() 33 ms
PrescriptionServiceImplementationTest 50 ms
QueueServiceImplTest 34 ms
ShareRecordImplementationTest 145 ms
addRecordHelper_WhenExistingRecord_ShouldNotCreateDuplicate() 102 ms
createShareRecord_ShouldCreateAndSaveShareRecord() 10 ms
addRecordHelper_WhenDifferentHospitalsAndNoExistingRecord_ShouldCreateNewRecord() 9 ms
revokeConsentHelper_ShouldDeleteAllRecordsForPatient() 9 ms
revokeConsentHelper_WhenNoRecordsExist_ShouldNotDeleteAnything() 8 ms
addRecordHelper_WhenSameHospital_ShouldNotCreateRecord() 7 ms
StorageServiceImplementationTest 1 sec 256 ms
downloadFile_ThrowsException() 814 ms
uploadFile_ThrowsException() 31 ms
downloadFile_Success() 385 ms
deleteFile_Success() 3 ms
allFilesPatientS3_Success() 8 ms
Tests passed: 79 of 79 tests – 8 sec 124 ms
C:\Users\ASUS\.jdks\openjdk-21.0.2\bin\java.
---- IntelliJ IDEA coverage runner ----
Line coverage ...
include patterns:
exclude annotations patterns:
.*Generated.*
WARNING: A Java agent has been loaded dynamic
WARNING: If a serviceability tool is in use,
WARNING: If a serviceability tool is not in
WARNING: Dynamic loading of agents will be di
OpenJDK 64-Bit Server VM warning: Sharing is
[2024.11.26 17:42:54] (Coverage): Error durir
February 1
January 1
doctor details: DoctorDepartment{, title='Dr.
dddd: Department{id=null, name='Cardiology'}
creating the queue
created the queue for department : Cardiology
dddd: null
creating the queue
created the queue for department : Neurology
deparetment null:Neurology in department tabl
Done Creating User
Patient not found
Error occurred in creating prescription pdf


Service 84% classes, 81% lines covered
ConsultationService
ConsultationServiceImplemetation 100% methods, 91% lines covered
DepartmentService
DepartmentServiceImplementation 100% methods, 100% lines covered
DoctorService
DoctorServiceImplementation 100% methods, 86% lines covered
GlobalAdminService
GlobalAdminServiceImplementation 100% methods, 98% lines covered
HospitalService
HospitalServiceImplementation 55% methods, 80% lines covered
PatientService
PatientServiceImpl 87% methods, 50% lines covered
PdfService
PdfServiceImplementation 100% methods, 99% lines covered
PrescriptionService
PrescriptionServiceImplementation 77% methods, 83% lines covered
QueueService
QueueServiceImpl 100% methods, 93% lines covered
ShareRecordImplementation 100% methods, 100% lines covered
ShareRecordService
StorageService
StorageServiceImplementation 100% methods, 91% lines covered
UserAuthenticationService 0% methods, 0% lines covered


ConsultationService
ConsultationServiceImplemetation 100% methods, 91
DepartmentService
DepartmentServiceImplementation 100% methods, 10
DoctorService
DoctorServiceImplementation 100% methods, 86% lin
GlobalAdminService
GlobalAdminServiceImplementation 100% methods, 9
HospitalService
HospitalServiceImplementation 55% methods, 80% lin
PatientService
PatientServiceImpl 87% methods, 50% lines covered
PdfService
PdfServiceImplementation 100% methods, 99% lines c
PrescriptionService
PrescriptionServiceImplementation 77% methods, 83%
QueueService
QueueServiceImpl 100% methods, 93% lines covered
ShareRecordImplementation 100% methods, 100% line
ShareRecordService
StorageService
StorageServiceImplementation 100% methods, 91% lin
dateWiseConsultationsList.add(
new DateWiseConsultations(key, value)
);
}
return dateWiseConsultationsList;
}
2 usages rushi30699
public List<MonthWiseConsultation> totalMonthWiseConsultations(Long doctorId) {
List<Consultation> consultationList = consultationRepository.findConsultationBy
Map<String, Long> totalConsultations = new TreeMap<>();
// Aggregate consultations by month
for (Consultation consultation : consultationList) {
// Extract month from date
String month = extractMonth(consultation.getConsultationDate());
// Aggregate consultations for each month
totalConsultations.merge(month, value: 1L, Long::sum);
}
List<MonthWiseConsultation> monthWiseConsultationsList = new ArrayList<>();
// Convert aggregated data to list of MonthWiseConsultations objects
for (Map.Entry<String, Long> entry : totalConsultations.entrySet()) {
String month = entry.getKey();