Reviewed by:

Approved by:

Revision / Document History

Ver.	Date	Changed by	Modifications
1.0	23/08/2023	1.Harshavardhan Reddy	

Note: Present content to be deleted and revision history to be updated with project's document history, when used in Projects

List of Abbreviations

DFD	**D**ata **F**low **D**iagram
ER	**E**ntity **R**elationship
FHD	**F**unction **H**ierarchy **D**iagram
HLD	**H**igh **L**evel **D**esign
LLD	**L**ow **L**evel **D**esign
GUI	**G**raphical **U**ser **I**nterface
IEEE	**I**nstitute of **E**lectrical and **E**lectronic **E**ngineers
S/W	**S**oft**w**are
SDL	**S**pecification **D**escription **L**anguage
StrD	**Str**uctur**ed**

TABLE OF CONTENTS

1. INTRODUCTION 5

2. DESIGN SCOPE 5

3. DESIGN METHODOLOGY 5

4. DESIGN NOTATIONS 5

5. DESIGN CONSIDERATIONS 5

6. DESIGN OVERVIEW 6

7. DECOMPOSITION 10

DH-1-1 UserEO 10

DH-1-2 DoctorEO 10

DH-1-3 PatientEO 10

DH-1-4 FeedbackEO 10

DH-1-5 AppointmentEO 11

DH-1-6 PrescriptionEO 11

DH-1-7 BillingEO 11

DH-1-8 MessageEO 11

DH-1-9 UserServiceImpl 12

DH-1-10 DoctorServiceImpl 12

DH-1-11 PatientServiceImpl 12

DH-1-12 FeedbackServiceImpl 12

DH-1-13 AppointmentServiceImpl 12

DH-1-14 PrescriptionServiceImpl 13

DH-1-15 BillingServiceImpl 13

DH-1-16 MessageServiceImpl 13

DH-1-17 UserService 13

DH-1-18 DoctorService 13

DH-1-19 PatientService 14

DH-1-20 FeedbackService 14

DH-1-21 AppointmentService 14

DH-1-22 PrescriptionService 14

DH-1-23 BillingService 15

DH-1-24 MessageService 15

DH-1-25 DoctorController 15

DH-1-26 UserController 15

DH-1-27 PatientController 16

DH-1-28 FeedbackController 16

DH-1-29 AppointmentController 16

DH-1-30 PrescriptionController 16

DH-1-31 BillingController 17

DH-1-32 MessageController 17

DH-1-33 AdminClerkApplication 17

DH-1-34 AppointmentsApplication 17

DH-1-35 MessagesApplication 18

DH-1-36 Authentication 18

8. INTERFACE DESIGN 18

8.1 User Interface 18

9. DATA DESIGN 18

9.2 Data structure (data types, arrays, and structures) 18

10. REUSABILITY 18

11. DESIGN ALTERNATIVES 19

13. ADDITIONAL HARDWARE AND SOFTWARE REQUIRED 19

14. TESTING STRATEGY 19

15. TRACEABILITY MATRIX 19

16. REFERENCES 19

1. Introduction

This project, Medical Service Automation System is meant to achieve paperless medical organization. This system will have data entry into electronic format and capture all services at hospital. The project is designed to be implemented on Java platform using Windows Operating System.

2. Design Scope

The scope of the Medical Service Automation System :

1. The admin should have the ability to add details of users, doctors.
2. The admin should have the ability to modify the users and doctor's details when required.
3. The clerk should be able to add In-Patient details and modify those details when required.
4. The doctor should be able to manage their profile.
5. The doctor should be able to manage appointments to patients, view e-prescriptions and interact with the patient using live chat.
6. The patient should be able to manage their profile.
7. The patient should be able to book appointments, view e-prescriptions and can interact with the doctor using live chat.
8. The billing department will calculate the expenses and verify if the patient has an insurance policy and process accordingly.

3. Design Methodology

Object Oriented Analysis and Design (OOAD) methodology has been used for breaking down the specification into functionally independent units.

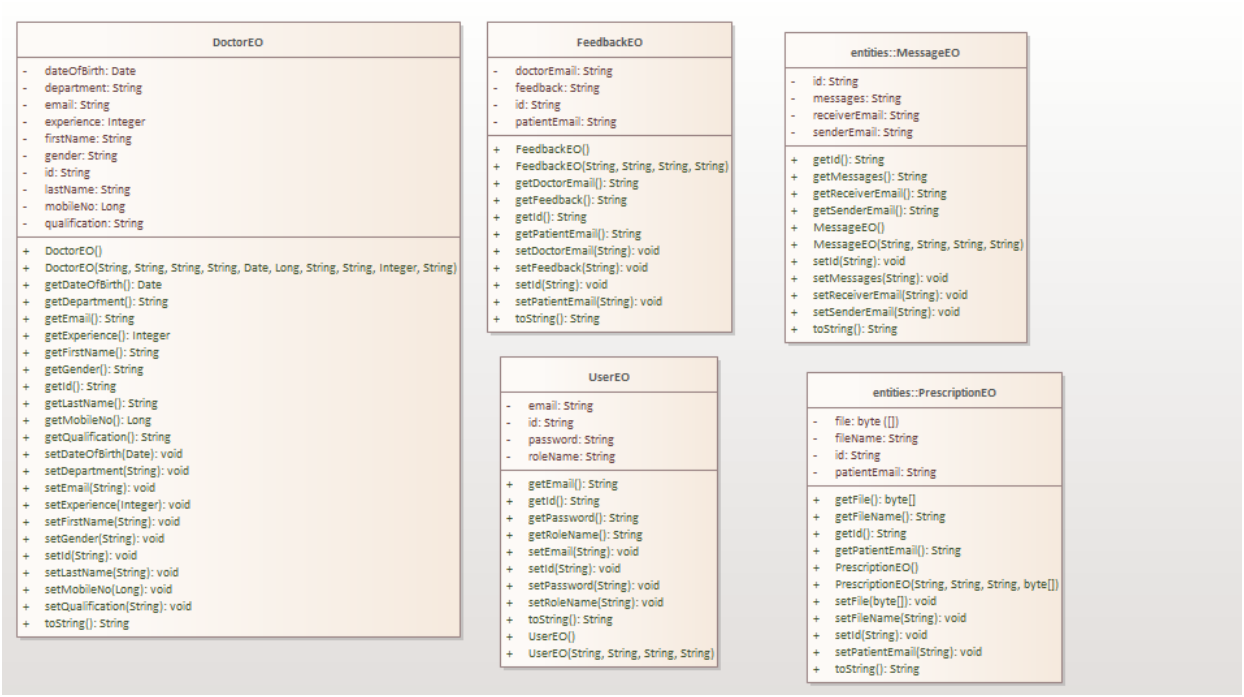
4. Design Notations

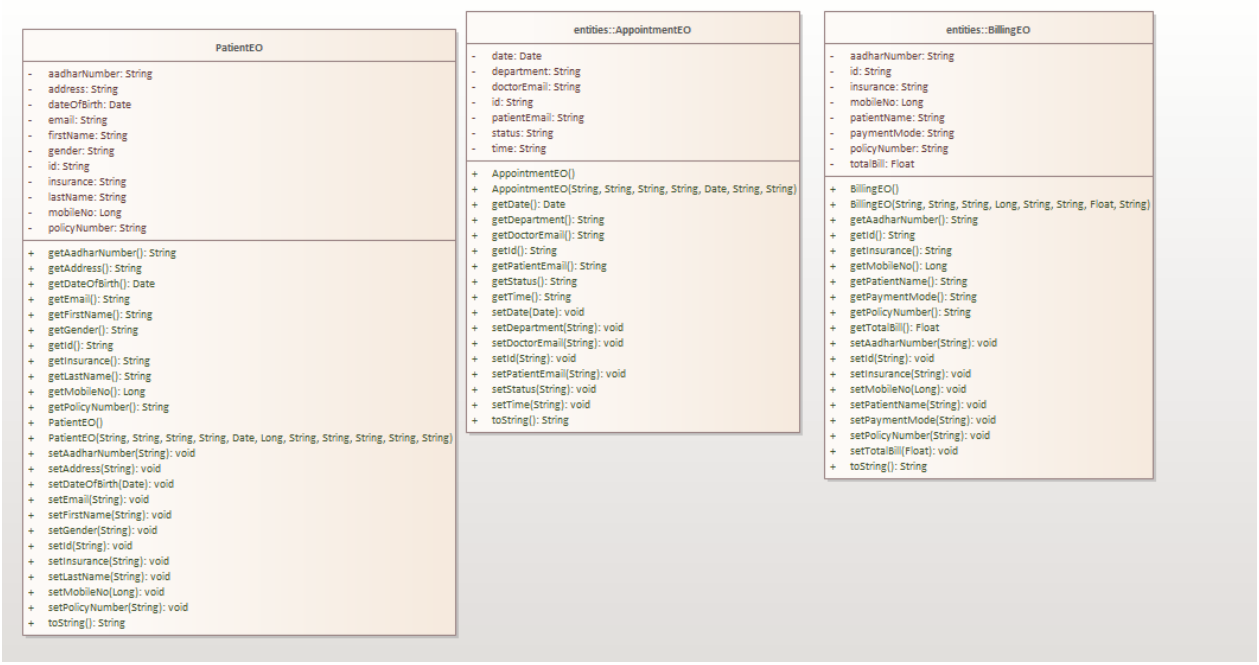
The naming conventions followed conform to Unified Modelling Language (UML) as Object Oriented Analysis and Design (OOAD) is followed.

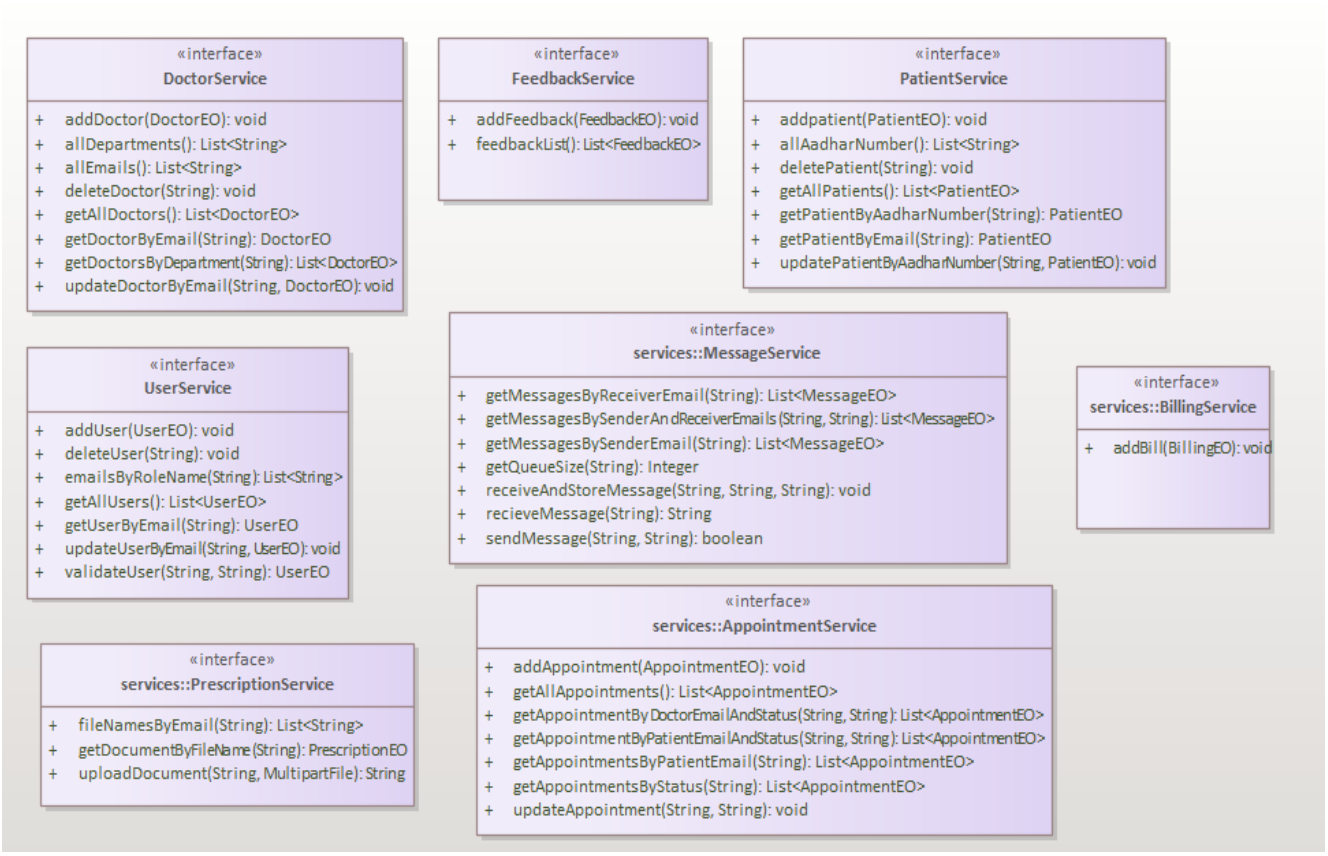
5. Design Considerations

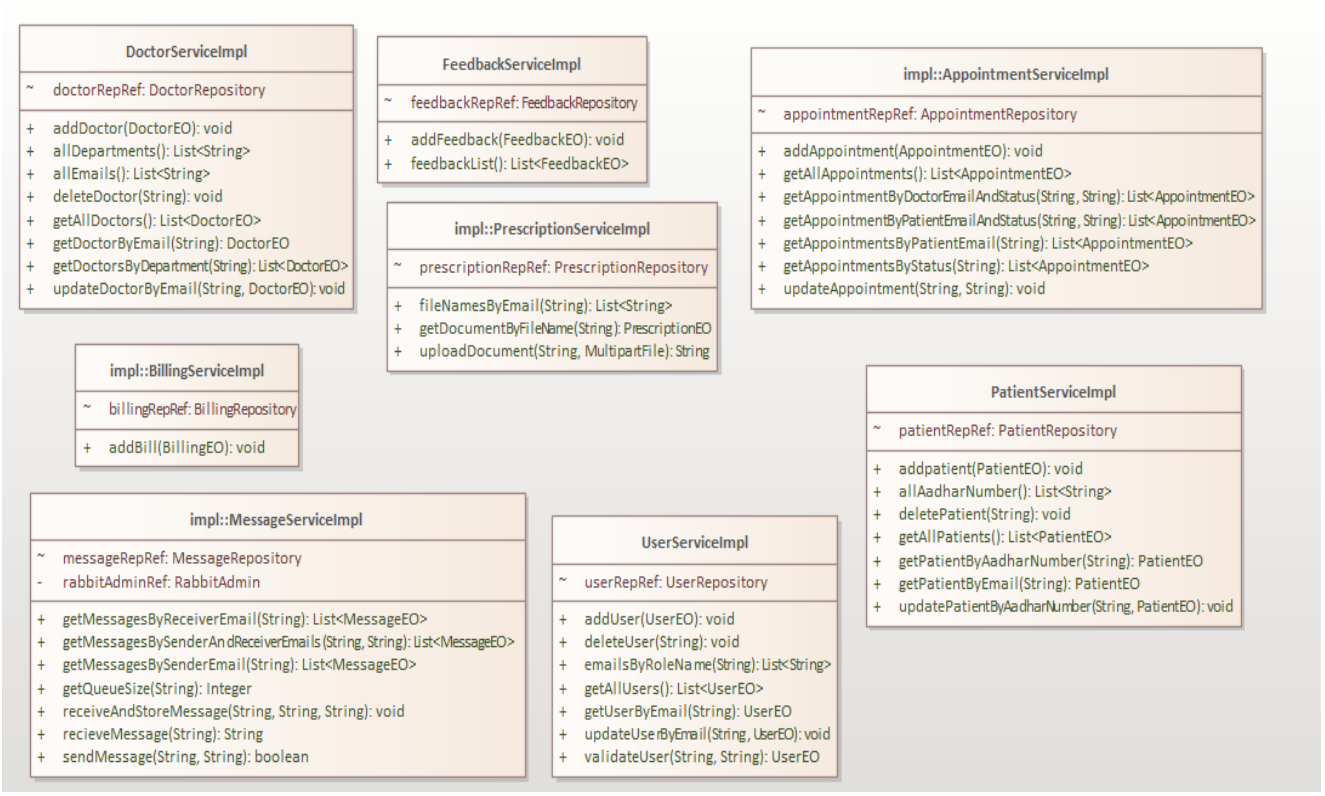
Not Applicable.

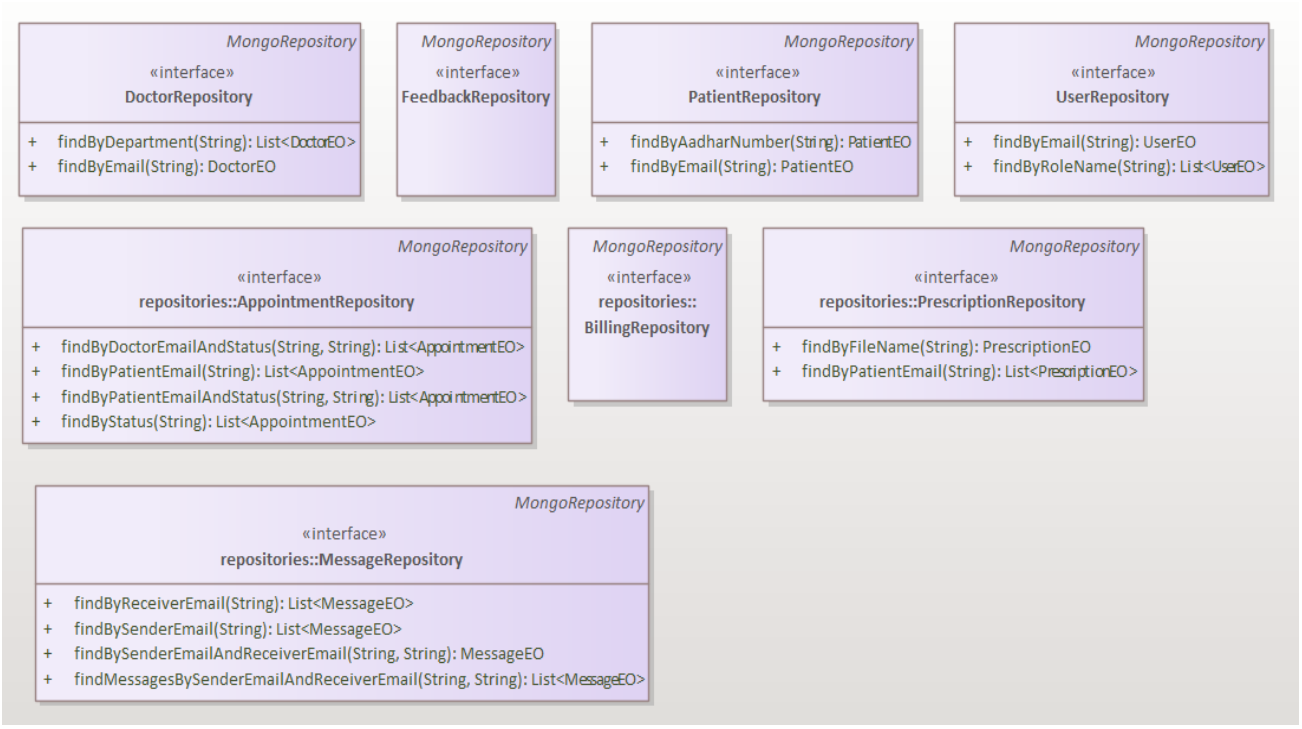
6. Design Overview

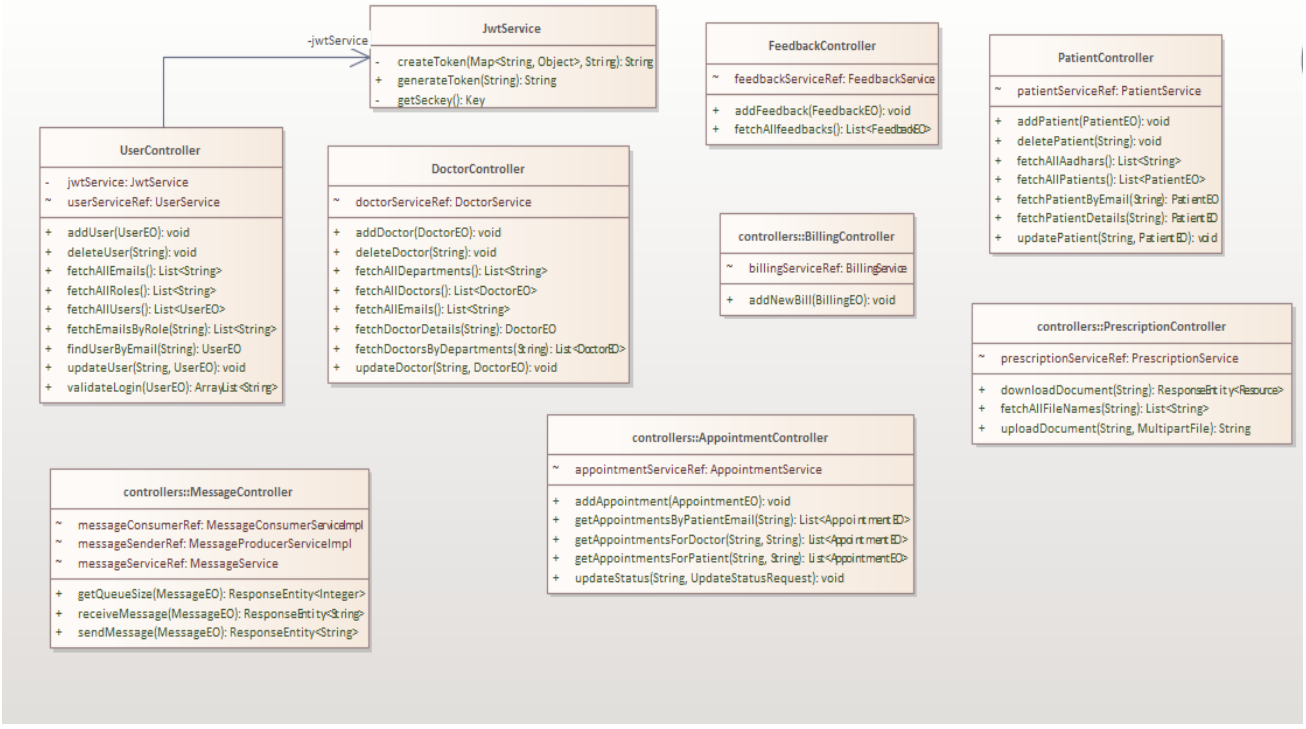


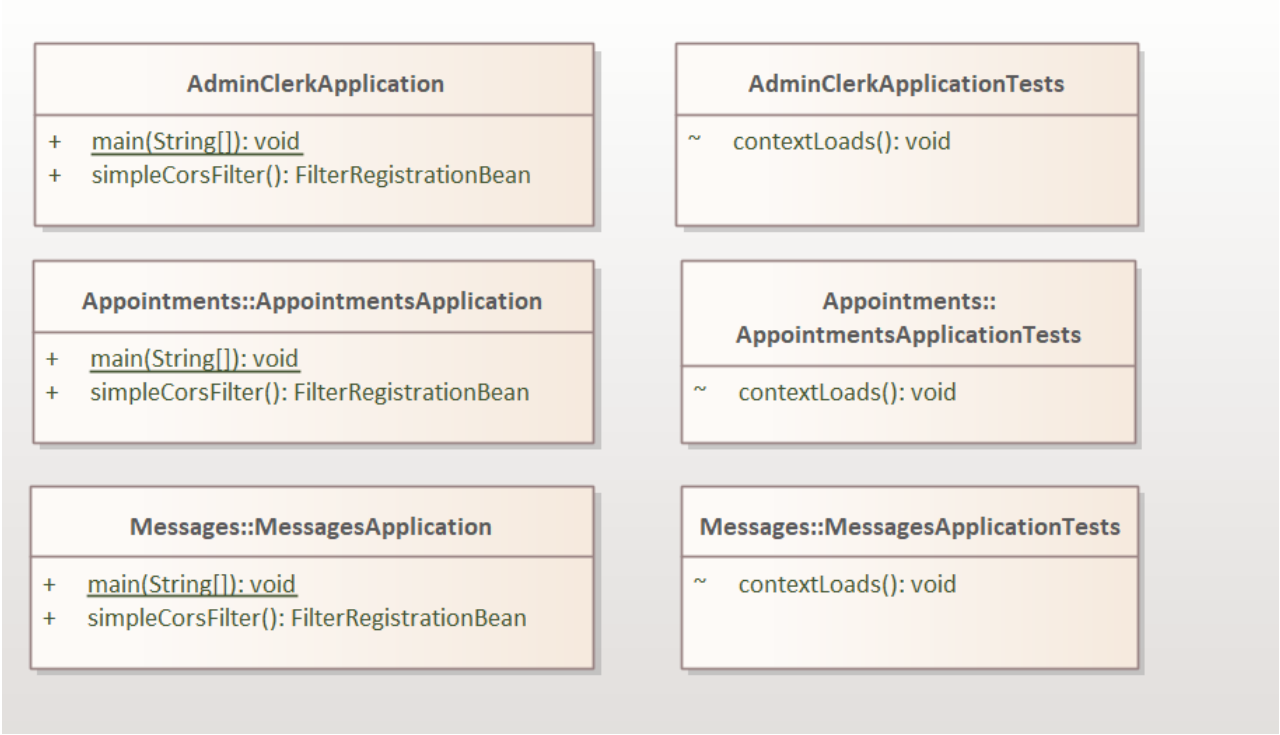

PatientEO
- aadharNumber: String
- address: String
- dateOfBirth: Date
- email: String
- firstName: String
- gender: String
- id: String
- insurance: String
- lastName: String
- mobileNo: Long
- policyNumber: String
+ getAadharNumber(): String
+ getAddress(): String
+ getDateOfBirth(): Date
+ getEmail(): String
+ getFirstName(): String
+ getGender(): String
+ getId(): String
+ getInsurance(): String
+ getLastName(): String
+ getMobileNo(): Long
+ getPolicyNumber(): String
+ PatientEO()
+ PatientEO(String, String, String, String, Date, Long, String, String, String, String, String)
+ setAadharNumber(String): void
+ setAddress(String): void
+ setDateOfBirth(Date): void
+ setEmail(String): void
+ setFirstName(String): void
+ setGender(String): void
+ setId(String): void
+ setInsurance(String): void
+ setLastName(String): void
+ setMobileNo(Long): void
+ setPolicyNumber(String): void
+ toString(): String
entities::AppointmentEO
- date: Date
- department: String
- doctorEmail: String
- id: String
- patientEmail: String
- status: String
- time: String
+ AppointmentEO()
+ AppointmentEO(String, String, String, String, Date, String, String)
+ getDate(): Date
+ getDepartment(): String
+ getDoctorEmail(): String
+ getId(): String
+ getPatientEmail(): String
+ getStatus(): String
+ getTime(): String
+ setDate(Date): void
+ setDepartment(String): void
+ setDoctorEmail(String): void
+ setId(String): void
+ setPatientEmail(String): void
+ setStatus(String): void
+ setTime(String): void
+ toString(): String
entities::BillingEO
- aadharNumber: String
- id: String
- insurance: String
- mobileNo: Long
- patientName: String
- paymentMode: String
- policyNumber: String
- totalBill: Float
+ BillingEO()
+ BillingEO(String, String, String, Long, String, String, Float, String)
+ getAadharNumber(): String
+ getId(): String
+ getInsurance(): String
+ getMobileNo(): Long
+ getPatientName(): String
+ getPaymentMode(): String
+ getPolicyNumber(): String
+ getTotalBill(): Float
+ setAadharNumber(String): void
+ setId(String): void
+ setInsurance(String): void
+ setMobileNo(Long): void
+ setPatientName(String): void
+ setPaymentMode(String): void
+ setPolicyNumber(String): void
+ setTotalBill(Float): void
+ toString(): String


«interface»
DoctorService
+ addDoctor(DoctorEO): void
+ allDepartments(): List<String>
+ allEmails(): List<String>
+ deleteDoctor(String): void
+ getAllDoctors(): List<DoctorEO>
+ getDoctorByEmail(String): DoctorEO
+ getDoctorsByDepartment(String): List<DoctorEO>
+ updateDoctorByEmail(String, DoctorEO): void
«interface»
FeedbackService
+ addFeedback(FeedbackEO): void
+ feedbackList(): List<FeedbackEO>
«interface»
PatientService
+ addpatient(PatientEO): void
+ allAadharNumber(): List<String>
+ deletePatient(String): void
+ getAllPatients(): List<PatientEO>
+ getPatientByAadharNumber(String): PatientEO
+ getPatientByEmail(String): PatientEO
+ updatePatientByAadharNumber(String, PatientEO): void
«interface»
UserService
+ addUser(UserEO): void
+ deleteUser(String): void
+ emailsByRoleName(String): List<String>
+ getAllUsers(): List<UserEO>
+ getUserByEmail(String): UserEO
+ updateUserByEmail(String, UserEO): void
+ validateUser(String, String): UserEO
«interface»
services::MessageService
+ getMessagesByReceiverEmail(String): List<MessageEO>
+ getMessagesBySenderAndReceiverEmails(String, String): List<MessageEO>
+ getMessagesBySenderEmail(String): List<MessageEO>
+ getQueueSize(String): Integer
+ receiveAndStoreMessage(String, String, String): void
+ recieveMessage(String): String
+ sendMessage(String, String): boolean
«interface»
services::BillingService
+ addBill(BillingEO): void
«interface»
services::PrescriptionService
+ fileNamesByEmail(String): List<String>
+ getDocumentByFileName(String): PrescriptionEO
+ uploadDocument(String, MultipartFile): String
«interface»
services::AppointmentService
+ addAppointment(AppointmentEO): void
+ getAllAppointments(): List<AppointmentEO>
+ getAppointmentByDoctorEmailAndStatus(String, String): List<AppointmentEO>
+ getAppointmentByPatientEmailAndStatus(String, String): List<AppointmentEO>
+ getAppointmentsByPatientEmail(String): List<AppointmentEO>
+ getAppointmentsByStatus(String): List<AppointmentEO>
+ updateAppointment(String, String): void


DoctorServiceImpl
~ doctorRepRef: DoctorRepository
+ addDoctor(DoctorEO): void
+ allDepartments(): List<String>
+ allEmails(): List<String>
+ deleteDoctor(String): void
+ getAllDoctors(): List<DoctorEO>
+ getDoctorByEmail(String): DoctorEO
+ getDoctorsByDepartment(String): List<DoctorEO>
+ updateDoctorByEmail(String, DoctorEO): void
FeedbackServiceImpl
~ feedbackRepRef: FeedbackRepository
+ addFeedback(FeedbackEO): void
+ feedbackList(): List<FeedbackEO>
impl::AppointmentServiceImpl
~ appointmentRepRef: AppointmentRepository
+ addAppointment(AppointmentEO): void
+ getAllAppointments(): List<AppointmentEO>
+ getAppointmentByDoctorEmailAndStatus(String, String): List<AppointmentEO>
+ getAppointmentByPatientEmailAndStatus(String, String): List<AppointmentEO>
+ getAppointmentsByPatientEmail(String): List<AppointmentEO>
+ getAppointmentsByStatus(String): List<AppointmentEO>
+ updateAppointment(String, String): void
impl::PrescriptionServiceImpl
~ prescriptionRepRef: PrescriptionRepository
+ fileNamesByEmail(String): List<String>
+ getDocumentByFileName(String): PrescriptionEO
+ uploadDocument(String, MultipartFile): String
impl::BillingServiceImpl
~ billingRepRef: BillingRepository
+ addBill(BillingEO): void
PatientServiceImpl
~ patientRepRef: PatientRepository
+ addpatient(PatientEO): void
+ allAadharNumber(): List<String>
+ deletePatient(String): void
+ getAllPatients(): List<PatientEO>
+ getPatientByAadharNumber(String): PatientEO
+ getPatientByEmail(String): PatientEO
+ updatePatientByAadharNumber(String, PatientEO): void
impl::MessageServiceImpl
~ messageRepRef: MessageRepository
- rabbitAdminRef: RabbitAdmin
+ getMessagesByReceiverEmail(String): List<MessageEO>
+ getMessagesBySenderAndReceiverEmails(String, String): List<MessageEO>
+ getMessagesBySenderEmail(String): List<MessageEO>
+ getQueueSize(String): Integer
+ receiveAndStoreMessage(String, String, String): void
+ recieveMessage(String): String
+ sendMessage(String, String): boolean
UserServiceImpl
~ userRepRef: UserRepository
+ addUser(UserEO): void
+ deleteUser(String): void
+ emailsByRoleName(String): List<String>
+ getAllUsers(): List<UserEO>
+ getUserByEmail(String): UserEO
+ updateUserByEmail(String, UserEO): void
+ validateUser(String, String): UserEO


MongoRepository
«interface»
DoctorRepository
+ findByDepartment(String): List<DoctorEO>
+ findByEmail(String): DoctorEO
MongoRepository
«interface»
FeedbackRepository
MongoRepository
«interface»
PatientRepository
+ findByAadharNumber(String): PatientEO
+ findByEmail(String): PatientEO
MongoRepository
«interface»
UserRepository
+ findByEmail(String): UserEO
+ findByRoleName(String): List<UserEO>
MongoRepository
«interface»
repositories::AppointmentRepository
+ findByDoctorEmailAndStatus(String, String): List<AppointmentEO>
+ findByPatientEmail(String): List<AppointmentEO>
+ findByPatientEmailAndStatus(String, String): List<AppointmentEO>
+ findByStatus(String): List<AppointmentEO>
MongoRepository
«interface»
repositories::
BillingRepository
MongoRepository
«interface»
repositories::PrescriptionRepository
+ findByFileName(String): PrescriptionEO
+ findByPatientEmail(String): List<PrescriptionEO>
MongoRepository
«interface»
repositories::MessageRepository
+ findByReceiverEmail(String): List<MessageEO>
+ findBySenderEmail(String): List<MessageEO>
+ findBySenderEmailAndReceiverEmail(String, String): MessageEO
+ findMessagesBySenderEmailAndReceiverEmail(String, String): List<MessageEO>


JwtService
-jwtService
- createToken(Map<String, Object>, String): String
+ generateToken(String): String
- getSeckey(): Key
FeedbackController
~ feedbackServiceRef: FeedbackService
+ addFeedback(FeedbackEO): void
+ fetchAllfeedbacks(): List<FeedbadEO>
PatientController
~ patientServiceRef: PatientService
+ addPatient(PatientEO): void
+ deletePatient(String): void
+ fetchAllAadhars(): List<String>
+ fetchAllPatients(): List<PatientEO>
+ fetchPatientByEmail(String): PatientEO
+ fetchPatientDetails(String): PatientED
+ updatePatient(String, PatientED): void
UserController
- jwtService: JwtService
~ userServiceRef: UserService
+ addUser(UserEO): void
+ deleteUser(String): void
+ fetchAllEmails(): List<String>
+ fetchAllRoles(): List<String>
+ fetchAllUsers(): List<UserEO>
+ fetchEmailsByRole(String): List<String>
+ findUserByEmail(String): UserEO
+ updateUser(String, UserEO): void
+ validateLogin(UserEO): ArrayList<String>
DoctorController
~ doctorServiceRef: DoctorService
+ addDoctor(DoctorEO): void
+ deleteDoctor(String): void
+ fetchAllDepartments(): List<String>
+ fetchAllDoctors(): List<DoctorEO>
+ fetchAllEmails(): List<String>
+ fetchDoctorDetails(String): DoctorEO
+ fetchDoctorsByDepartments(String): List<DoctorED>
+ updateDoctor(String, DoctorEO): void
controllers::BillingController
~ billingServiceRef: BillingService
+ addNewBill(BillingEO): void
controllers::PrescriptionController
~ prescriptionServiceRef: PrescriptionService
+ downloadDocument(String): ResponseEntity<Resource>
+ fetchAllFileNames(String): List<String>
+ uploadDocument(String, MultipartFile): String
controllers::AppointmentController
~ appointmentServiceRef: AppointmentService
+ addAppointment(AppointmentEO): void
+ getAppointmentsByPatientEmail(String): List<AppointmentED>
+ getAppointmentsForDoctor(String, String): List<AppointmentED>
+ getAppointmentsForPatient(String, String): List<AppointmentED>
+ updateStatus(String, UpdateStatusRequest): void
controllers::MessageController
~ messageConsumerRef: MessageConsumerServiceImpl
~ messageSenderRef: MessageProducerServiceImpl
~ messageServiceRef: MessageService
+ getQueueSize(MessageEO): ResponseEntity<Integer>
+ receiveMessage(MessageEO): ResponseEntity<String>
+ sendMessage(MessageEO): ResponseEntity<String>


AdminClerkApplication
+ main(String[]): void
+ simpleCorsFilter(): FilterRegistrationBean
AdminClerkApplicationTests
~ contextLoads(): void
Appointments::AppointmentsApplication
+ main(String[]): void
+ simpleCorsFilter(): FilterRegistrationBean
Appointments::
AppointmentsApplicationTests
~ contextLoads(): void
Messages::MessagesApplication
+ main(String[]): void
+ simpleCorsFilter(): FilterRegistrationBean
Messages::MessagesApplicationTests
~ contextLoads(): void

7. Decomposition

DH-1-1 UserEO

Inputs: Sets the user details

Outputs: Gets the userdetails.

Scope: Specific

This class is a model class for the collection users.

DH-1-2 DoctorEO

Inputs: Sets the doctor details.

Outputs: Gets the doctor details.

Scope: Specific

This class is a model class for the collection doctor.

DH-1-3 PatientEO

Inputs: Sets the patient details.

Outputs: Gets the patient details.

Scope: Specific

This class is a model class for the collection patient.

DH-1-4 FeedbackEO

Inputs: Sets the feedback details.

Outputs: Gets the feedback details.

Scope: Specific

This class is a model class for the collection feedback.

DH-1-5 AppointmentEO

Inputs: Sets the appointment details

Outputs: Gets the appointment details.

Scope: Specific

This class is a model class for the collection appointment.

DH-1-6 PrescriptionEO

Inputs: Sets the prescription details.

Outputs: Gets the prescription details.

Scope: Specific

This class is a model class for the collection prescription.

DH-1-7 BillingEO

Inputs: Sets the billing details.

Outputs: Gets the billing details.

Scope: Specific

This class is a model class for the collection billing.

DH-1-8 MessageEO

Inputs: Sets the message details.

Outputs: Gets the message details.

Scope: Specific

This class is a model class for the collection messages.

DH-1-9 UserServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Specific

This class implements the UserService interface.

DH-1-10 DoctorServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Generic

This class implements the DoctorService interface.

DH-1-11 PatientServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Specific

This class implements the PatientService interface.

DH-1-12 FeedbackServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Specific

This class implements the FeedbackService interface.

DH-1-13 AppointmentServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Specific

This class implements the AppointmentService interface.

DH-1-14 PrescriptionServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Generic

This class implements the PrescriptionService interface.

DH-1-15 BillingServiceImpl

Inputs: The details to be updated, selected or deleted.

Outputs: The result of selection, updating or deletion.

Scope: Specific

This class implements the BillingService interface.

DH-1-16 MessageServiceImpl

Inputs: The messages to be sent and received.

Outputs: The result of messages sent.

Scope: Specific

This class implements the MessageService interface.

DH-1-17 UserService

Inputs: NA

Outputs: Object of UserServiceImpl class.

Scope: Specific

This class creates an object of UserServiceImpl class.

DH-1-18 DoctorService

Inputs: NA

Outputs: Object of DoctorServiceImpl class.

Scope: Specific

This class creates an object of DoctorServiceImpl class.

DH-1-19 PatientService

Inputs: NA

Outputs: Object of PatientServiceImpl class.

Scope: Specific

This class creates an object of PatientServiceImpl class.

DH-1-20 FeedbackService

Inputs: NA

Outputs: Object of FeedbackServiceImpl class.

Scope: Specific

This class creates an object of FeedbackServiceImpl class.

DH-1-21 AppointmentService

Inputs: NA

Outputs: Object of AppointmentServiceImpl class.

Scope: Specific

This class creates an object of AppointmentServiceImpl class.

DH-1-22 PrescriptionService

Inputs: NA

Outputs: Object of PrescriptionServiceImpl class.

Scope: Specific

This class creates an object of PrescriptionServiceImpl class.

DH-1-23 BillingService

Inputs: NA

Outputs: Object of BillingServiceImpl class.

Scope: Specific

This class creates an object of BillingServiceImpl class.

DH-1-24 MessageService

Inputs: NA

Outputs: Object of MessageServiceImpl class.

Scope: Specific

This class creates an object of MessageServiceImpl class.

DH-1-25 DoctorController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to doctor operations.

DH-1-26 UserController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to user operations.

DH-1-27 PatientController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to patient operations.

DH-1-28 FeedbackController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to feedback operations.

DH-1-29 AppointmentController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to appointment operations.

DH-1-30 PrescriptionController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to prescription operations.

DH-1-31 BillingController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to billing operations.

DH-1-32 MessageController

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used for mapping the URL with respect to message operations.

DH-1-33 AdminClerkApplication

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used to run the application as a microservice.

DH-1-34 AppointmentsApplication

Inputs: NA

Outputs: NA

Scope: Specific.

This class is used to run the application as a microservice.

DH-1-35 MessagesApplication

Inputs:	NA
Outputs:	NA
Scope:	Specific

This class is used to run the application as a microservice.

DH-1-36 Authentication

Inputs:	Username and Password
Outputs:	Result of authentication
Scope:	Generic

This class is used to Authenticate users using details furnished by the user.

8. Interface Design

8.1 User Interface

NA

9. Data design

9.2 Data structure (data types, arrays, and structures)

Not applicable.

10. Reusability

- Not Applicable

11. Design Alternatives

NA

12. Design Feasibility

We have used the OOAD approach in this project. This methodology has been chosen based on our analogy of the user requirements, feasibility study and based on the experience of the co-ordinators. The OOAD assures properties such as reusability, modularity, efficiency.

13. Additional Hardware and Software required

This requirement is based on the future stages of development. Therefore as of now this is not applicable

14. Testing Strategy

The various stages of testing to be followed for our application includes white unit and integration testing.

We will carry out all such testing in a simulated environment only.

15. Traceability Matrix

As per the requirements-HLD tagging shown in the document “Requirement_Traceability.xls” each of the requirements has been mapped to the appropriate classes. Both the requirements and classes have been tagged according to the tag standards of RBIN.

16. References

List of all external sources of information referenced in this document.

Sl. No.	Description	Date	Vers.	Location

1.	Software Requirements Specification Document	29/07/2023	1.0	*SRS.doc*
2.				
3.				

Description, date, and version shall uniquely identify the information source, and the location shall specify where it is to be found.