Reviewed by:
Approved by:

Revision / Document History

Ver.	Date	Changed by	Modifications
1.0	30/06/2025	Srigowri N	

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**uctured

TABLE OF CONTENTS

1. Introduction

The purpose of the Claim Processing System is to develop a robust, user-friendly, and efficient web application using React that streamlines the entire insurance claim lifecycle for Sun Health and Allied Insurance. This system aims to automate and simplify the submission, validation, tracking, and approval of insurance claims by providing a centralized platform for all stakeholders including claimants, insurance coordinators, claims officers, medical reviewers, and approvers. The system will enhance operational efficiency, reduce manual errors, improve transparency, and speed up claim settlements, thereby increasing customer satisfaction and trust. By integrating role-based access and workflow automation, the project seeks to ensure secure and compliant handling of sensitive claim data while supporting timely decision-making.

2. Design Scope

The Claim Processing System will be designed to support end-to-end digital management of insurance claims across multiple user roles. It will include intuitive interfaces for Claimants, Policy Holders, and Insurance Coordinators to submit claims with necessary documentation, along with real-time tracking dashboards to monitor claim status. The system will provide Claims Processing Officers with tools for claim validation, documentation checks, and fraud detection, while enabling Medical Practitioners and Internal Medical Reviewers to verify treatment authenticity and upload medical proofs. A comprehensive approval workflow will be incorporated for Claims Approvers, Managers, and Regional Heads to review and approve high-value claims, including escalation and revalidation options. Additionally, the system will offer robust user and role management for secure access control, automated notifications to keep stakeholders informed, and advanced reporting and analytics capabilities for performance monitoring and fraud analysis. Overall, the design will focus on efficient workflows, clear communication, and secure, role-based access throughout the claim lifecycle.

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

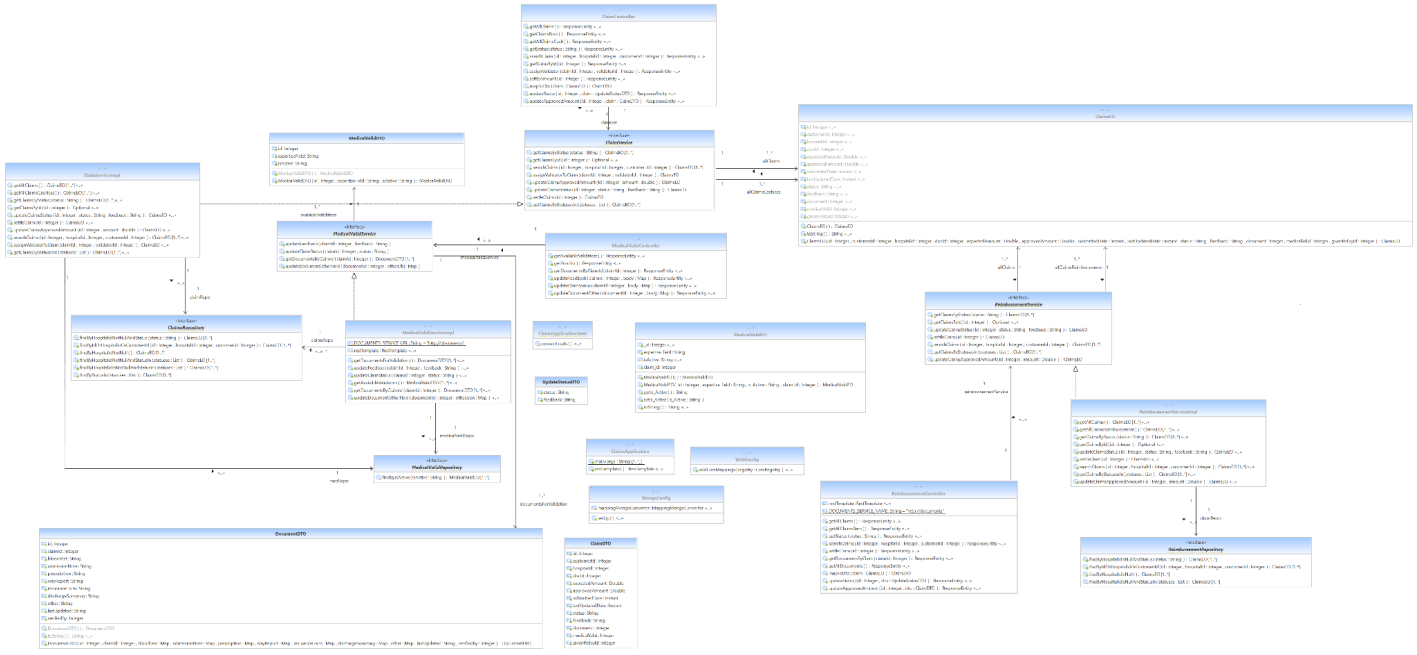
5. Design Considerations

Not Applicable

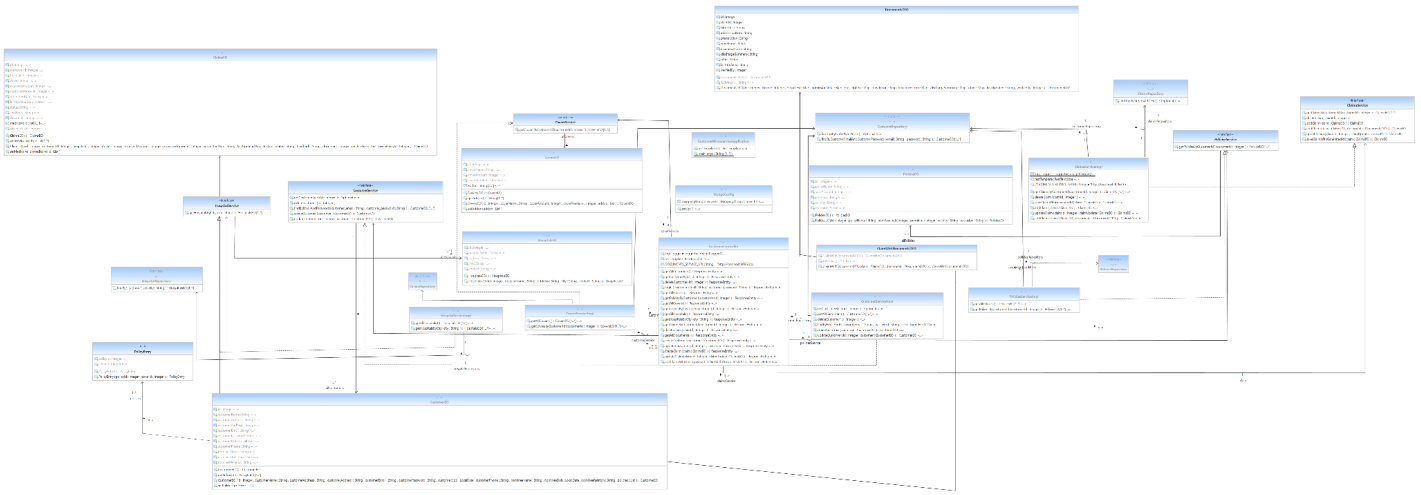
6. Design Overview

Admin Microservice

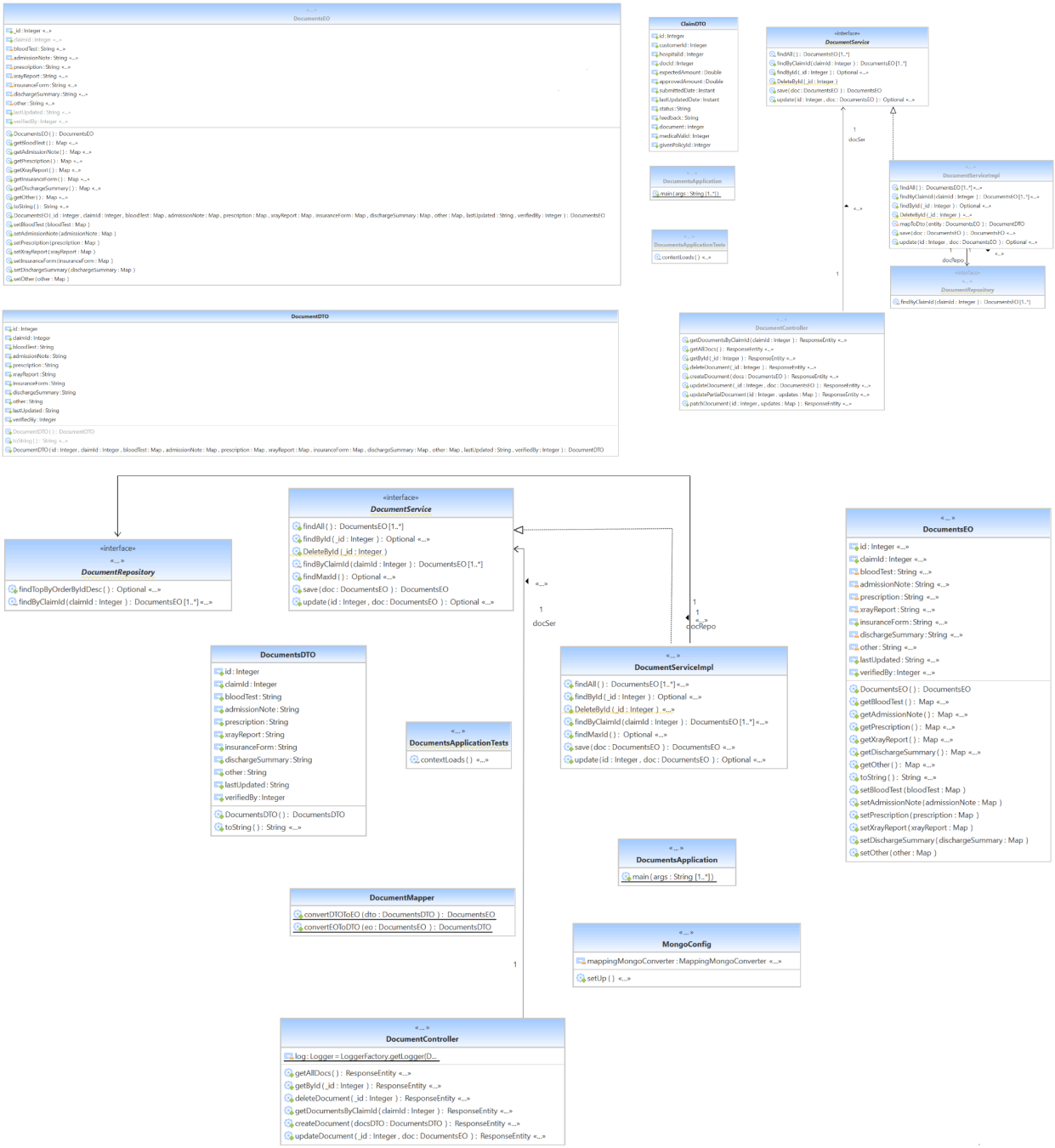
Claims Microservice



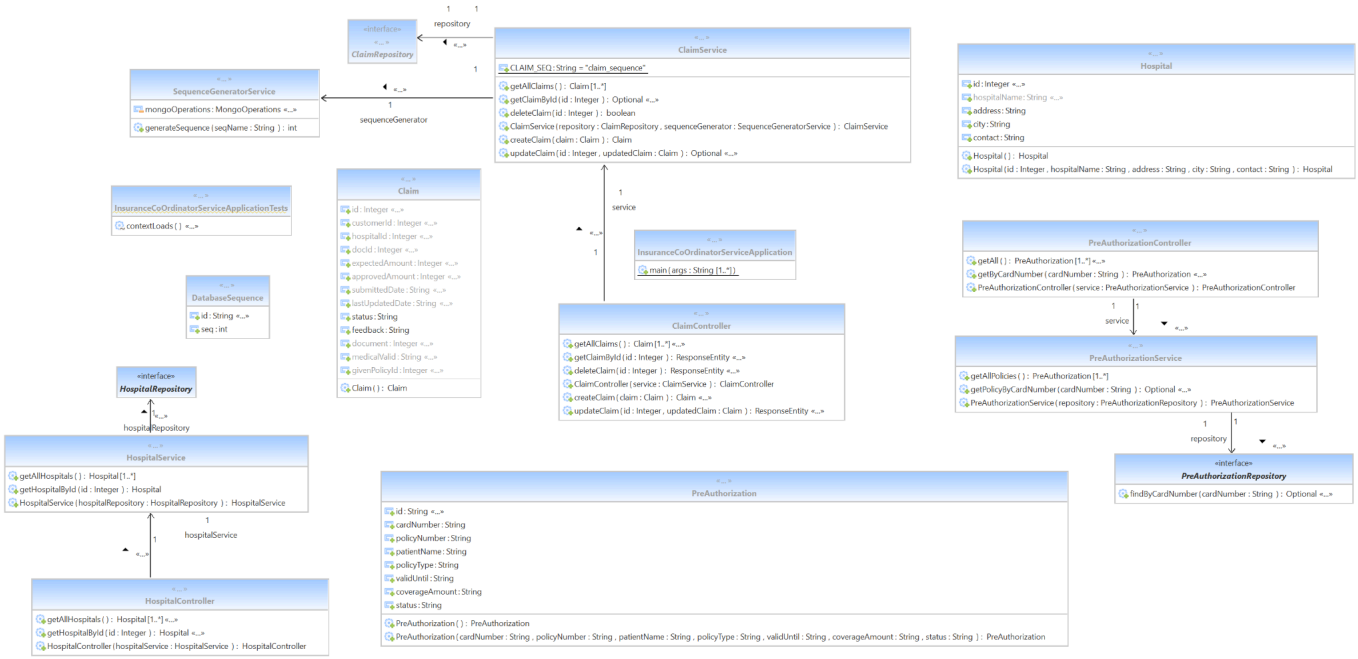
Customer Microservice



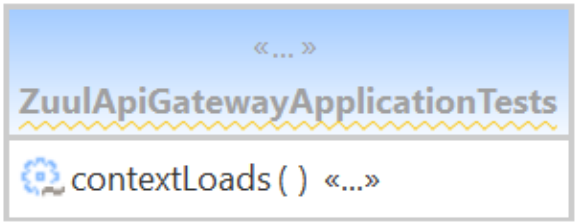
Documents Microservice



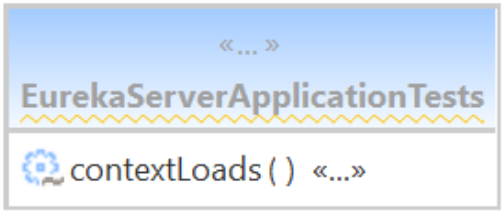
Insurance Co-Ordinator Microservice



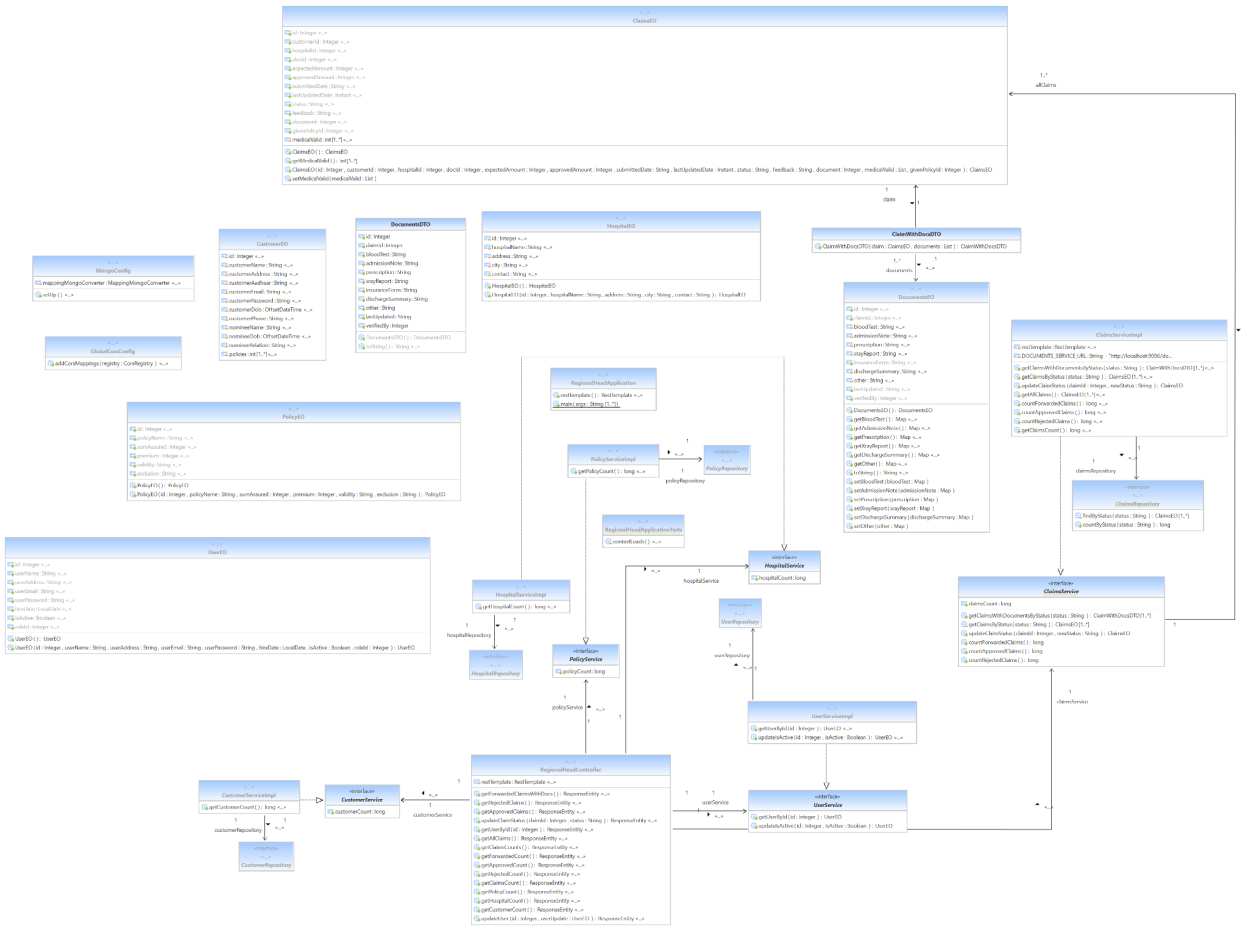
API Gateway Microservice



Eureka Microservice



Regional Head Microservice



7. Decomposition

DH-1-1 CoverEO

Inputs: Set cover details including id, cover name, cover amount, and cover premium.
Outputs: Get cover details from the database.
Scope: Specific

The Cover class (com.cps.admin.model.Cover) represents insurance cover options stored in the MongoDB collection covers. Its models cover attributes such as name, insured amount, and premium cost tied to insurance policies.

DH-1-2 MedicalValidatorEO

Inputs: Set id, field of expertise, active status, and associated claim ID.
Outputs: Retrieve medical validator information.
Scope: Specific

The MedicalValidator class (com.cps.admin.model.MedicalValidator) models experts who verify medical claims. Stored in the medicalValid MongoDB collection, this entity tracks their expertise, active status, and linked claims for validation purposes.

DH-1-3 PolicyEO

Inputs: Set id, policy name, sum assured, premium, validity period, and exclusions.
Outputs: Retrieve policy details from the database.
Scope: Specific

The Policy class (com.cps.admin.model.Policy) models insurance policies stored in the MongoDB collection policies. It includes core policy information such as coverage sums, premiums due, validity, and exclusions.

DH-1-4 PolicyHolderEO

Inputs: id, personal information, password, nominee details, and linked policies list.
Outputs: Retrieve a comprehensive customer profile with linked policies from the DB.
Scope: Specific

The PolicyHolder class (com.cps.admin.model.PolicyHolder) represents customers who hold insurance policies. Stored in the customers collection, it contains detailed personal data, nominee info, security credentials, and embedded references to policies and covers.

DH-1-5 RoleEO

Inputs: Set role details including id, role name, and role description.
Outputs: Retrieve role information for access control.
Scope: Specific

The Role class (com.cps.admin.model.Role) models user roles within the system to manage permissions. Stored in the roles MongoDB collection, it defines role names and descriptions used to assign user privileges.

DH-1-6 UserEO

Inputs: id, name, address, email, password, hire date, active status & assigned role ID.
Outputs: Retrieve user information including role associations.
Scope: Specific

The User class (com.cps.admin.model.User) represents system users with their credentials and metadata. Stored in the users collection, it includes user profile data, authentication fields, employment details, active status, and role linkage for access management.

DH-1-7 ClaimsEO

Inputs: Set claim details including claim id, customer ID, hospital ID, doctor ID, expected claim amount, approved claim amount, claim submission date, last updated date, claim status, feedback, associated document ID, medical
Outputs: Retrieve all claim-related details from the MongoDB claims collection.
Scope: Specific

The ClaimsEO class (training.iqgateway.model.ClaimsEO) is a model class representing the insurance claim entity stored in the MongoDB claims collection. It captures the comprehensive claim data such as the claimant's identification, hospital and doctor associations, financial details (expected and approved amounts), timestamps for claim submission and updates, claim status and feedback, as well as references to supporting documents, medical validations, and policies involved. This entity forms the core data structure for insurance claim processing workflows.

DH-1-8 CustomerEO

Inputs: Set customer details including id, name, address, Aadhaar number, email, password, date of birth, phone, nominee details (name, date of birth, relation), and a list of linked policies each with policy and cover IDs.
Outputs: Retrieve complete customer profile with nominee and linked policies from the MongoDB customers collection.
Scope: Specific
The CustomerEO class (traning.iqgateway.entities.CustomerEO) models insurance customers stored in the customers collection. It captures personal details, contact information, nominee data, and policy associations, facilitating comprehensive customer profile management in the claims and policy systems.

DH-1-9 HospitalEO

Inputs: id, hospital name, address, city, and contact information.
Outputs: Retrieve hospital information from the MongoDB hospitals collection.
Scope: Specific
Scope: Specific
The HospitalsEO class (traning.iqgateway.entities.HospitalsEO) represents hospitals participating in the insurance ecosystem. It stores essential identifiers and contact details, enabling hospital information management and linking in claims processing.

DH-1-10 DocumentsEO

Inputs: id, linked claim ID, and multiple medical and administrative document categories (blood test, admission note, prescription, x-ray report, insurance form, discharge summary, other documents) stored as key-value maps; also includes last updated timestamp and verifier user ID.
Outputs: Retrieve detailed documents and verification metadata from the MongoDB documents collection.
Scope: Specific
The DocumentsEO class (training.iqgateway.model.DocumentsEO) encapsulates various claim-related documents organized by category as flexible maps, linked to specific claims. It supports audit and verification tracking for submitted documents within the insurance claim workflow.

DH-1-11 DatabaseSequenceEO

Inputs: Set sequence identifier (name) and current integer sequence value.
Outputs: Retrieve sequence state used for generating unique IDs.
Scope: Specific
The DatabaseSequence class (com.example.claim.model.DatabaseSequence) models a sequence counter stored in the counters MongoDB collection. It is typically used for generating sequential unique identifiers for entities across the system in a thread-safe manner.

DH-1-12 Customer Controller

Inputs: HTTP request payloads including customer, policies, covers, hospitals, claims data, JSON bodies, and query parameters.
Outputs: HTTP responses with entity data or status codes for CRUD operations, login, and cross-service document handling.
Scope: Specific

The CustomerController class (traning.iqgateway.controller.CustomerController) exposes REST APIs for managing customers, policies, covers, hospitals, and claims. It integrates services for CRUD operations, login authentication, and interactions with a document microservice.

DH-1-13 ClaimsServiceImpl

Inputs: ClaimsEO objects and optional DocumentsDTO for claim creation, updates, and deletions.
Outputs: Persisted ClaimsEO entities with generated IDs, filtered claim lists, updated claims, and deletion acknowledgments.
Scope: Specific

The ClaimsServiceImpl class (traning.iqgateway.serviceImpl.ClaimsServiceImpl) implements insurance claim business logic, including ID generation, claim lifecycle management, and integration with the Documents microservice for document uploads and linkage.

DH-1-14 CoversServiceImpl

Inputs: Customer ID to fetch linked covers.
Outputs: Lists of CoversEO either all covers or filtered by customer association.
Scope: Specific

The CoversServiceImpl class (traning.iqgateway.serviceImpl.CoversServiceImpl) provides functionalities for retrieving and filtering insurance covers from the database based on policy ownership by customers.

DH-1-15 CustomerServiceImpl

Inputs: CustomerEO objects and identifiers for CRUD and login operations.
Outputs: Saved, updated, or retrieved CustomerEO entities and search results by email/password.
Scope: Specific

The CustomerServiceImpl class (traning.iqgateway.serviceImpl.CustomerServiceImpl) manages customer profile persistence, ID generation, CRUD operations, and credential-based authentication.

DH-1-16 HospitalServiceImpl

Inputs: Query parameters such as city names for filtering hospitals.
Outputs: Lists of HospitalsEO either all hospitals or filtered by city.
Scope: Specific

The HospitalServiceImpl class (traning.iqgateway.serviceImpl.HospitalServiceImpl) handles retrieval of hospital data, supporting queries for all hospitals or city-specific listings.

DH-1-17 PoliciesServiceImpl

Inputs: Customer ID to retrieve linked policies.
Outputs: Lists of PoliciesEO all policies or filtered by customer association.
Scope: Specific

The PoliciesServiceImpl class (traning.iqgateway.serviceImpl.PoliciesServiceImpl) manages insurance policy data access, fetching all policies or those linked to a customer through embedded policy references.

DH-1-18 DocumentController

Inputs: HTTP requests with path variables, JSON request bodies containing document fields, or partial update maps.
Outputs: HTTP responses with document data, status codes for CRUD operations, and error handling messages.
Scope: Specific

The DocumentController class (training.iqgateway.controller.DocumentController) exposes REST APIs for managing insurance claim documents stored in MongoDB. It supports full and partial document retrieval, creation, update, patch, and deletion, integrating flexible JSON mapping to support dynamic fields such as nested document maps, with CORS enabled for front-end integration.

DH-1-19 DocumentServiceImpl

Inputs: CRUD operations and query parameters such as claim ID or document ID.
Outputs: Persisted entities, optional updated documents, and lists of documents matching queries.
Scope: Specific

The DocumentServiceImpl class (training.iqgateway.service.impl.DocumentServiceImpl) implements the business logic for document management. It handles CRUD persistence using a MongoDB repository, supports partial updates by merging document fields, and maps entity data to DTOs for API response formatting.

DH-1-20 ClaimController

Inputs: HTTP requests with claim data such as IDs, updated claim details, and JSON request bodies.
Outputs: HTTP responses with claim entities, lists, or status codes for CRUD operations.
Scope: Specific

The ClaimController class (com.example.claim.controller.ClaimController) exposes REST APIs to manage insurance claims. It provides endpoints for creating, retrieving, updating, and deleting claims while handling request-response mapping for integration with front-end applications.

DH-1-21 PreAuthorizationController

Inputs: HTTP requests for retrieving pre-authorization policies by list or specific card number.
Outputs: Lists of PreAuthorization entities or single policy objects returned as JSON responses.
Scope: Specific

The PreAuthorizationController class (com.example.preauth.controller.PreAuthorizationController) provides REST APIs to fetch all pre-authorization policies or a specific policy by card number, facilitating access to pre-authorization data for claims processing.

DH-1-22 ClaimServiceImpl

Inputs: Claim entities and claim IDs for CRUD operations and sequence generation requests.
Outputs: Persisted Claim entities, optional retrieved claims, updated claims, or deletion status.
Scope: Specific

The ClaimService class (com.example.claim.service.ClaimService) implements core business logic for managing claims. It handles creating claims with sequence-generated IDs, updates, retrievals, and deletions, while updating timestamp fields accordingly.

DH-1-23 SequenceGeneratorService

Inputs: Sequence name strings for generating numeric sequence values.
Outputs: Incremented integer sequence numbers used for generating unique IDs.
Scope: Specific

The SequenceGeneratorService class (com.example.claim.service.SequenceGeneratorService) uses MongoDB operations to safely increment and retrieve sequence values, supporting unique ID generation across entities in the system.

DH-1-24 HospitalController

Inputs: HTTP requests for all hospitals or hospital by ID.
Outputs: List of Hospital entities or single Hospital object.
Scope: Specific

The HospitalController class (com.example.hospital.controller.HospitalController) provides REST endpoints to retrieve all hospital records or a specific hospital by its identifier to support claim and network provider workflows.

DH-1-25 PreAuthorizationService

Inputs: Repository queries for pre-authorization policies or lookups by card number.
Outputs: Lists of PreAuthorization entities or optional single policy results.
Scope: Specific

The PreAuthorizationService class (com.example.preauth.service.PreAuthorizationService) handles database interactions to retrieve pre-authorization policies, facilitating access and validation during claim evaluations.

DH-1-26 HospitalServiceImpl

Inputs: Repository queries for hospital records by ID or all records.
Outputs: Lists of Hospital entities or single Hospital objects (nullable if not found).
Scope: Specific

The HospitalService class (com.example.hospital.service.HospitalService) supports hospital data retrieval for claims and policy administration by interfacing with the HospitalRepository to fetch hospital entities.

DH-1-27 RegionalHeadController

Inputs: HTTP requests with claim IDs, user IDs, status parameters, and JSON request bodies for updates.
Outputs: HTTP responses with claim data, user data, count results, or status codes for CRUD and aggregation operations.
Scope: Specific

The RegionalHeadController class (traning.iqgateway.controller.RegionalHeadController) exposes APIs for regional heads to manage claims, users, policies, hospitals, and customers. It supports approving, rejecting, forwarding claims, updating claim statuses, activating/deactivating users, and provides aggregate counts for dashboard insights.

DH-1-28 AdminController

Inputs: HTTP requests with user IDs, JSON request bodies for user details, or login credentials.
Outputs: User entities, lists of users, login responses, or status codes for create, read, update, delete operations.
Scope: Specific

The AdminController class (com.cps.admin.controller.AdminController) provides administrative APIs for managing system users. It includes CRUD endpoints, login authentication using email and password, and interacts with UserService to manage user accounts for the system.

DH-1-29 MedicalValidatorController

Inputs: HTTP requests with validator IDs and JSON request bodies containing medical validator details.
Outputs: MedicalValidator entities, lists, or status codes for CRUD operations.
Scope: Specific

The MedicalValidatorController class (com.cps.admin.controller.MedicalValidatorController) manages medical validators through REST APIs. It supports creation, retrieval, update, and deletion of validators, used for claim medical validation and fraud detection workflows.

DH-1-30 PolicyHolderController

Inputs: Policy holder IDs and JSON request bodies containing policy holder information.
Outputs: PolicyHolder entities, lists of policy holders, or status codes for CRUD operations.
It supports creation, retrieval, updating, and deletion of policy holder records via integration with PolicyHolderService.

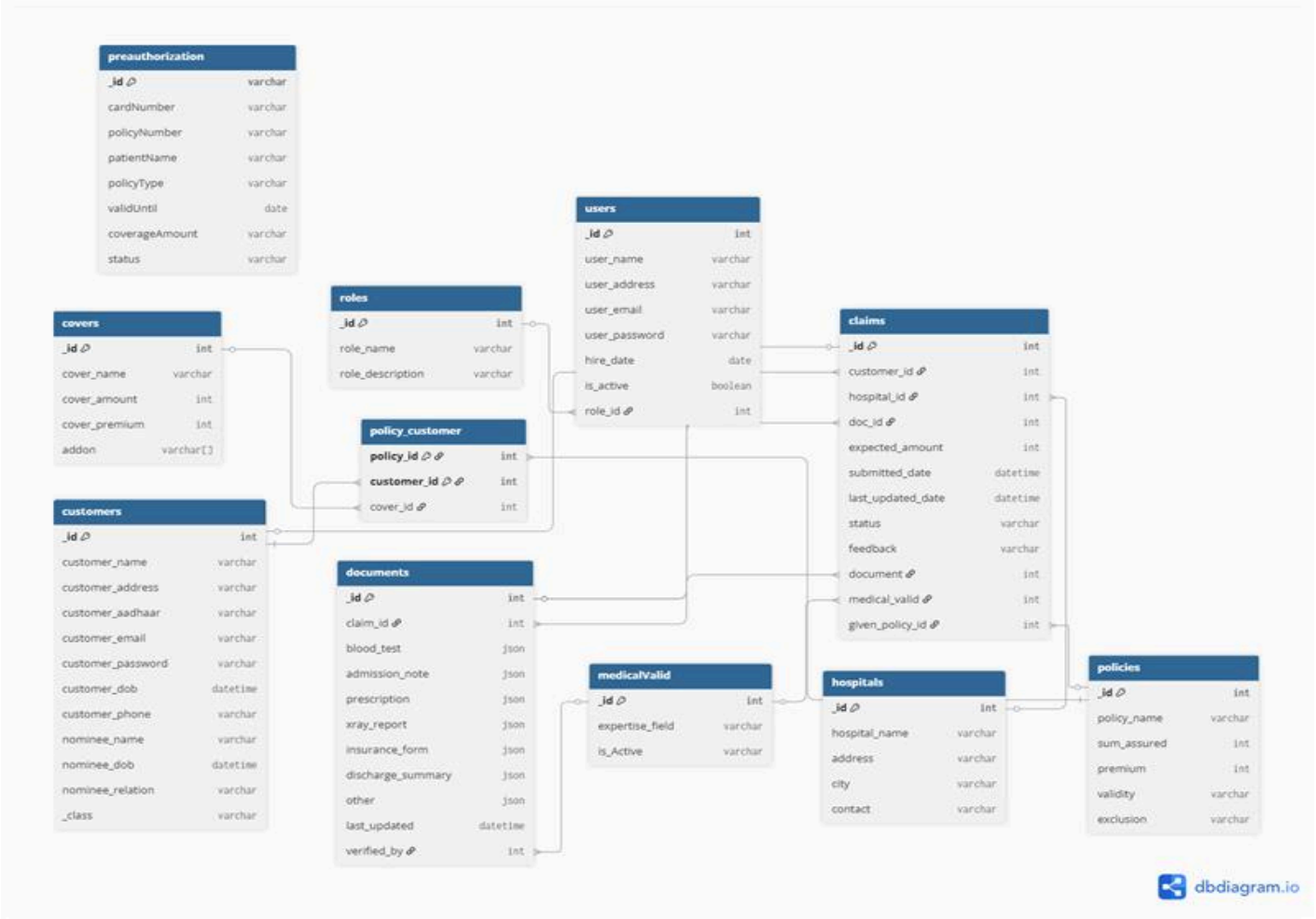
8. Interface Design

Not applicable.

8.1 User Interface

Not applicable.

9. Data design



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

Not applicable.

10. Reusability

- Documents
- Pre Authorization

11. Design Alternatives

Not applicable.

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. It has been seen that several other project groups developing similar projects have chosen the same methodology.

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.

16. References

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

Sl. No.	Description	Date	Vers.	Location
1.	SRS Document	23/06/2025	1.0	*SRS.doc*
2.	High Level Design	30/06/2025	1.0	
3.	Low Level Design	30/06/2025	1.0	
4.	Readme	30/06/2025		