BSc (Hons) in Information Technology

IT Project – IT2080

Activity 02 – Requirements Engineering

Topic : Unified System for Insurance Claim Management

Group no : ITP25_B4_96

Campus : Malabe

Reg No	Name	Email	Contact No
IT23834774	NATH I M N	it23834774@my.sliit.lk	+94 77 429 0347
IT23836440	FERNANDO PULLE N S	it23836440@my.sliit.lk	+94 70 311 4390
IT23830332	PATHIRANA P U O R	it23830332@my.sliit.lk	+94 71 679 2331
IT23725010	PERERA B I V	it23725010@my.sliit.lk	+94 70 134 6360
IT23828766	SENARATHNA P G R M	it23828766@my.sliit.lk	+94 71 777 6610

SLIIT
Discover Your Future

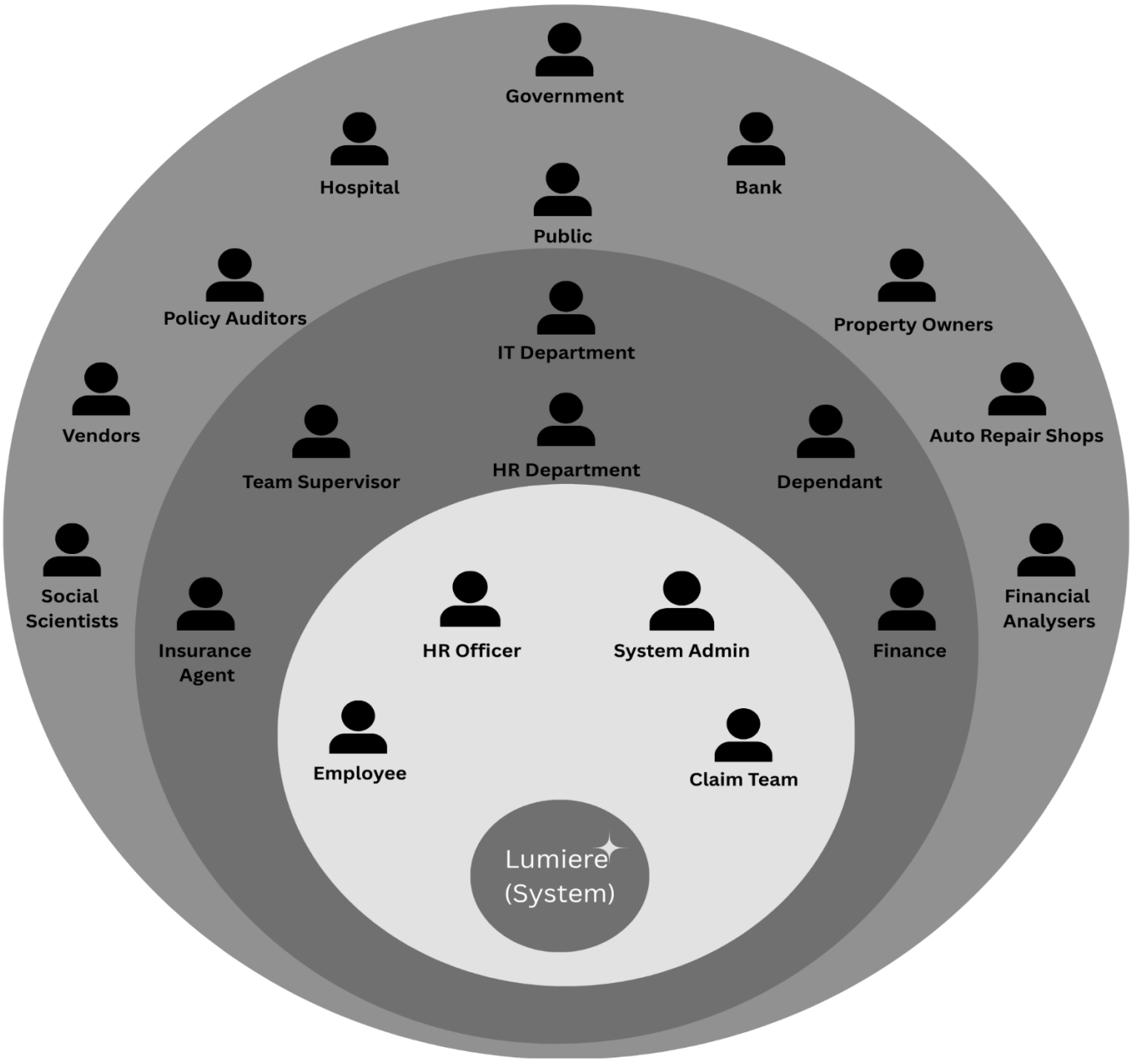
BSc (Hons) in Information Technology

IT Project – IT2080

Activity 02 – Requirements Engineering

Contents

ONION DIAGRAM



Functional Requirements for main stakeholders

Stakeholder	Functional Requirement
Employees	Register and securely log in to the system
	Upload identity verification documents (NIC/Passport)
	View own insurance policies and remaining claim limits
	Submit insurance claims with supporting documents
	Track the status of submitted claims
	Communicate with HR via built-in messaging
	Receive notifications (e.g., claim status updates, document requests)
HR Department	Secure login for HR users
	Review and validate submitted claims
	Forward valid claims to assigned insurance agents
	View employee insurance history and ID verification status
	Respond to employee messages
	Receive alerts when new claims are submitted or pending
Insurance Agents	Secure login for agents
	Access claims forwarded by HR
	Review claim details and uploaded documents
	Approve, reject, or request additional documents for a claim
	Enter final claim decision and reimbursement amount
	Add internal notes visible to HR
Admin	Verify and activate HR and agent accounts
	Manage system settings (policy types, coverage categories, claim limits)
	Monitor logs and analytics for claims processing
	Configure chatbot knowledge base and notification templates
All Users	Enable secure internal messaging between stakeholders
	Implement chatbot support for FAQs and claim guidance
	Handle secure document uploads linked to claim IDs
	Generate and deliver in-app, email, or SMS notifications
	Track individual and group policy usage (claims vs. total limit)

Non-functional requirements

Stakeholder	Non-Functional Requirement
Employees	Usability: Interface should be intuitive for claim submission, status tracking, and document upload
	Security: Personal and claim data must be encrypted and protected from unauthorized access
	Performance: The system should allow smooth upload of documents and load dashboards quickly
	Accessibility: System should be accessible via both desktop and mobile devices
	Reliability: Users must be able to access the system without unexpected downtimes
HR Department	Usability: HR dashboard should enable quick filtering, sorting, and viewing of employee claims
	Security: Only authorized HR personnel can view/edit employee data
	Performance: System should handle large numbers of concurrent claim reviews without lag
	Auditability: All claim interactions must be logged for internal auditing
Insurance Agents	Usability: Clear interface for reviewing claims, viewing attached documents, and entering decisions
	Security: Access to only forwarded claims; ability to add notes securely
	Performance: Capable of processing multiple claims in parallel efficiently
	Availability: System should be accessible at all times for real-time decision processing
Admin	Security: Role-based access control for all user types (employees, HR, agents)
	Performance: Tools should allow for real-time monitoring and management
	Maintainability: Easy deployment of feature updates and patches
	Scalability: Ability to support increasing number of users and claim volume
System-Wide	Localization: Interface should support multiple languages if needed
	Backup & Recovery: Regular backups of data and fast recovery in case of failure
	Integration: Should integrate seamlessly with external systems (payment gateways, identity verification, etc.)
	Chatbot: Fast and context-aware responses; ability to be updated with new queries
	Notification System: Should deliver alerts instantly with retry mechanisms in case of failure

Technical requirements

Frontend - react.js (Tailwind CSS, material UI for UI)

Backend - node.js (express.js)

Database - MongoDB

USE CASE DIAGRAM

Use cases

Use Case 1: System Admin Creates a Group Policy

Actor: System Admin

Preconditions: Admin is authenticated

Trigger: Admin selects “Create Group Policy” option

Main Flow:

- Admin navigates to the “Policies” section.
- Clicks “Create Group Policy”.
- Fills in required fields:
  - Policy name, description
  - Insurance provider
  - Coverage details (life, health, accident)
  - Sum assured, duration, premium terms
- Clicks Save.
- System validates inputs and stores group policy in database.
- Confirmation message shown.
  Postconditions: New group policy is available to assign to employees.

Use Case 2: HR Officer Creates Employee Account & Assigns Group Policy

Actor: HR Officer

Preconditions: HR Officer is authenticated

Trigger: HR chooses "Add New Employee"

Main Flow:

- HR navigates to "Employee Management".
- Clicks "Add New Employee".
- Fills in:
  - Name, NIC, contact info, email
  - Employee type (regular/executive)
  - Initial temporary password
  - Assigns relevant group policy/policies
- Clicks Create.
- System stores employee info and sends credentials via email.
- HR instructs employee to reset password on first login.
  Postconditions: Employee account is created and linked to policy.

Use Case 3: Employee Submits Claim with Documents

Actor: Employee

Preconditions: Employee is logged in and has an active policy

Trigger: Employee selects "Submit Claim"

Main Flow:

- Employee goes to "Submit Claim".
- Selects policy (if multiple assigned).
- Chooses claim type: Life / Vehicle.
- Fills in form: event date, description, etc.
- Uploads required documents (death certificate, bills, etc.).
- Clicks Submit.
- System stores claim and links it to the employee and policy.
- Notification is sent to HR Officer for review.
  Postconditions: Claim is now in "Pending Review" state.

Use Case 4: HR Officer Reviews and Forwards Claim

Actor: HR Officer

Preconditions: A claim has been submitted

Trigger: HR views pending claims

Main Flow:

- HR opens a submitted claim.
- Views documents, claim info, policy details.
- Fills in additional required fields (like formatting, summarizing).
- Optionally adds internal notes or flags inconsistencies.
- Clicks Forward to Agent.
- System updates claim status to forwarded.
- Insurance agent is notified.
  Postconditions: Claim is now assigned to an insurance agent.

Use Case 5: Employee Sends Inquiry, HR Responds

Actor: Employee, HR Officer

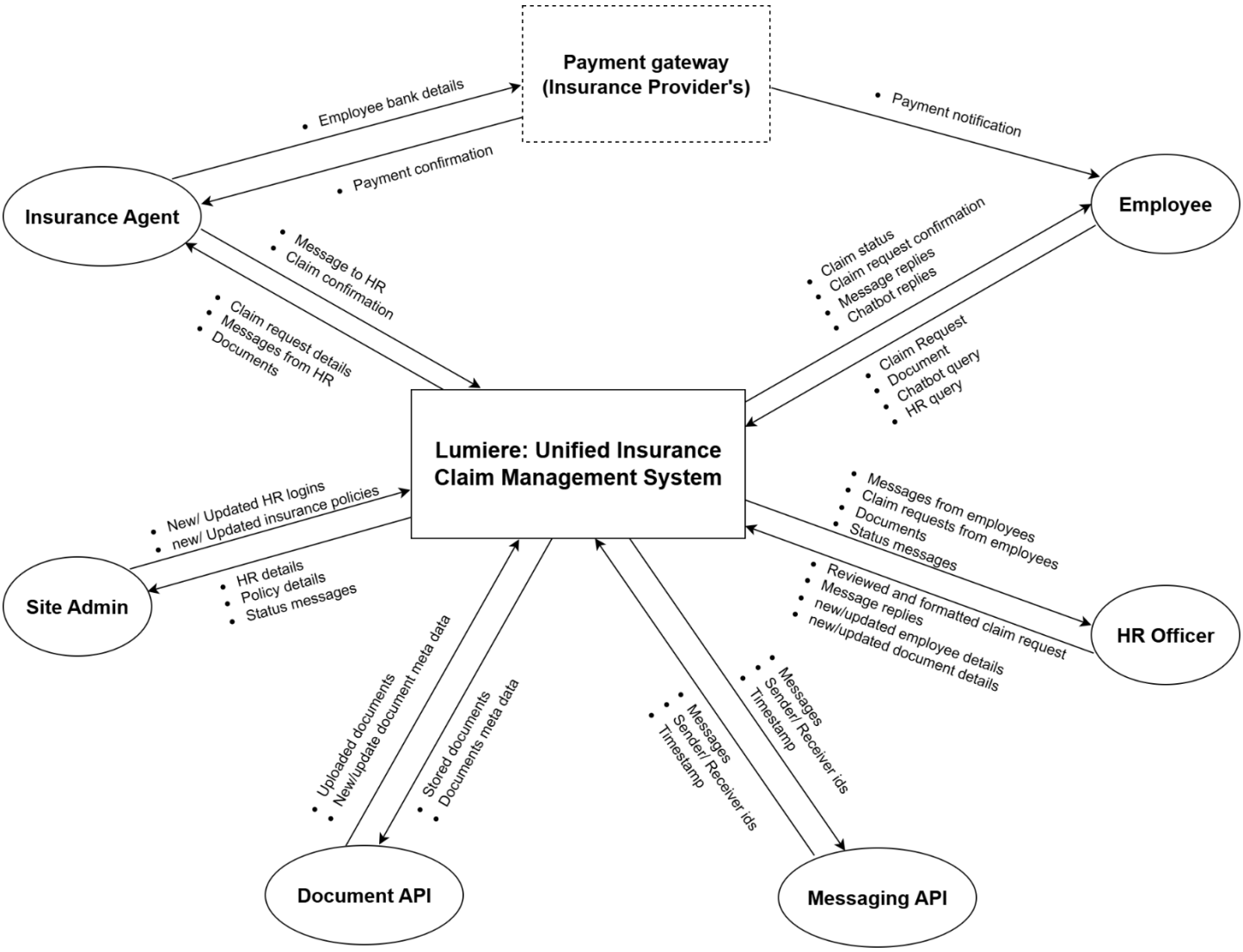
Preconditions: Both are authenticated

Trigger: Employee opens chat and starts a new message

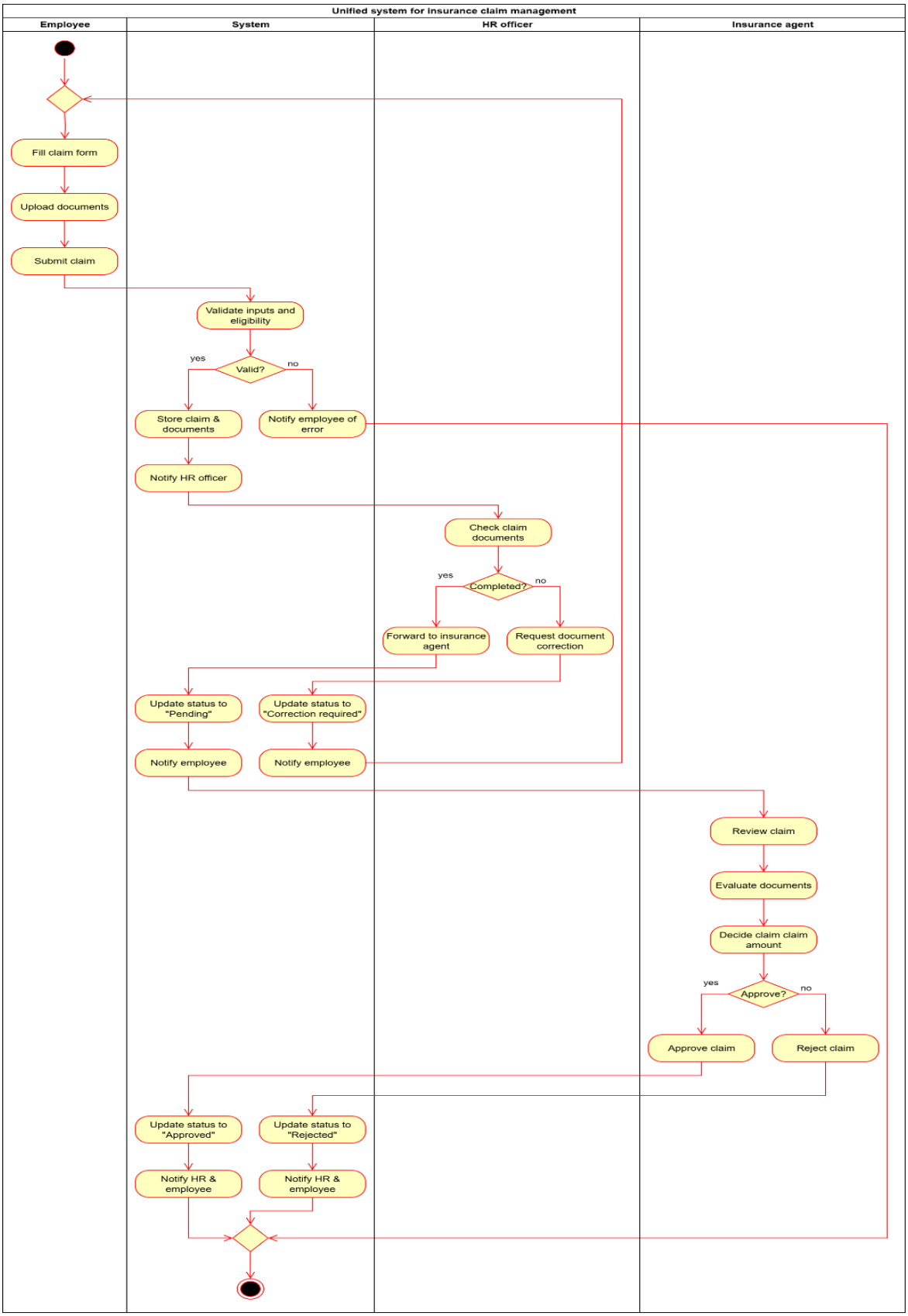
Main Flow:

- Employee opens “Messages”.
- Writes inquiry (e.g., claim delay, clarification).
- System stores message and sends to HR inbox.
- HR receives notification.
- HR opens chat, reads message, and responds.
- Employee receives and views reply in message history.
  Postconditions: Message thread is updated and visible to both.

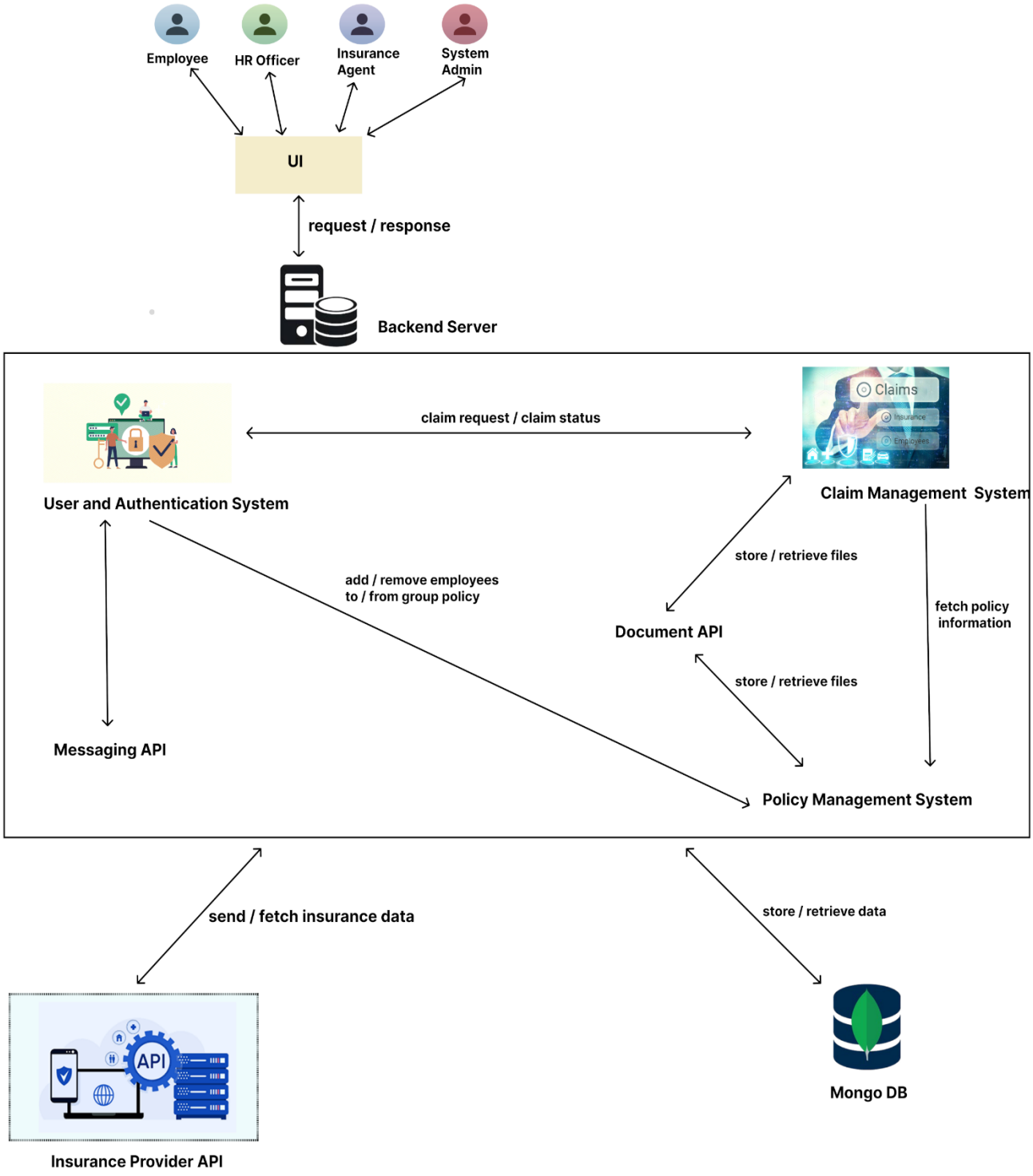
DATA FLOW DIAGRAM



SYSTEM ACTIVITY DIAGRAM



SYSTEM DIAGRAM



Project planning as a team

The development of this unified Claim Management System (Lumiere) is intended to give employees, insurance agents, and administrators an easy-to-use and efficient digital system for managing insurance policies, filing claims, and monitoring their status. Our team collected the requirements for this system through research on the difficulties encountered by employees in using manual or legacy claim processes, as well as the administrative costs borne by insurers and HR departments. We've specifically contacted an HR officer at Janashakthi group as our client and received special requests and specifications.

Through thorough discussions and requirement gathering sessions, both face-to-face and through online media, we were able to identify key features required to facilitate claim submission, verification, and communication between stakeholders. After thorough analysis of the system requirements, we were able to effectively allocate work to the team members responsible for modules such as user authentication, policy management, claim processing, contact support, notification system, and administrative tools. This collaborative planning enabled role clarity and a common vision for system development

Function	**Team Member**
Document and chatbot functions	NATH I M N
Claim management function	FERNANDO PULLE N S
Messaging and notification functions	PATHIRANA P U O R
User management function	PERERA B I V
Policy management function	SENARATHNA P G R M

Agile Development Approach

In developing the Employee Insurance Claim Management System, our team utilized the Agile Methodology in order to preserve flexibility, continuous improvement, and stakeholder collaboration within the project cycle.

Scrum Meetings

▪ **Stand-Ups** - We do a short daily stand-up where each team member states their progress, any blockers, and their work for the day. This allows for transparency and quick resolution of issues.

▪ **Sprint Planning** - At the beginning of a sprint, we discuss what features and functionalities we want to prioritize, like claim submission, policy management, or notification triggers. The tasks are then broken down into manageable units and assigned to relevant team members so that they can be executed in the most efficient manner.

Sprint Reviews

▪ Towards the end of each sprint, we conduct sprint reviews to check the features developed such as the claim form module, user registration, or admin dashboard. We gather feedback to ascertain that all deliverables are according to user requirements and technical expectations, and necessary adjustments are planned for future sprints.

Sprint Planning

Sprint	Goal	Tasks	Deliverables
Sprint 1 (Weeks 1-2)	System foundation - design and setup	• Design the database schema for all major collections • Set up MongoDB hosting (MongoDB Atlas) • Initialize Git repository and create feature branches • Design base UI/UX • Set up Node.js backend project structure with basic User Management API	• Initial schema design document • Git repository with separate branches • UI/UX prototype for core UI components (Figma) • Running backend with basic structure • Authenticated user flow working via API (bcrypt for password hashing)
Sprint 2 (Weeks 3-4)	Core backend development - Claims and Policies APIs	• Implement Claims API and Policies API (CRUD + validations) • Extend User Management API to support roles and authorization • Build React components for login and static pages	• Functional backend APIs for Claims and Policies • Component-based layout and routing setup in React • Postman collection for existing routes
Sprint 3 (Weeks 5-6)	Support systems - Documents and Messaging + Feedback	• Implement Document Upload API (file storage with metadata) • Implement Messaging API • Integrate document logic into Claims flow • Develop core functional React components • Collect client feedback on the current implementation	• Document and Messaging APIs ready and tested • Integrated backend and frontend logic • React components for submitting claims and messaging • Mid-sprint client feedback report
Sprint 4 (Weeks 7-8)	AI and Notifications modules	• Build LLM-powered Chatbot API (using gemini API) • Implement Notifications API (email or in-app alerts or both) • Develop React components for chatbot interaction and alerts	• Working chatbot module with sample queries • Notification triggers on claim updates/messages • Chat and notification components integrated in UI

Sprint 5 (Weeks 9-10)	System integration and refinement	• Integrate all frontend and backend modules end-to-end • Secure backend routes with proper role checks • Add validation and clear error feedback on all forms • Improve UI styling and flow consistency	• Fully functional and integrated system • Role-based access control working correctly • Consistent and user-friendly UI
Sprint 6 (Weeks 11-12)	Final feedback, testing, and deployment	• Collect final feedback from client and refine system • Conduct full system testing (Unit, Integration, System) • Fix bugs, polish UX, and finalize features • Deploy the app to a cloud host (Azure)	• Finalized and tested app • Client-approved final build • Live deployed version with access credentials

Communication & Collaboration

Project management

As a project management tool, we have decided to use **Clickup** to track tasks, and progress, and assign responsibilities among members.

Version control

We have decided to use **Git (GitHub)** to manage code and prevent conflicts and set up clear branching and continuously improving strategies.

Communication

We use **Google Meet, Zoom** and **WhatsApp** for team communication. Moreover, we have established a dedicated WhatsApp group with all the members in the team.

Code Reviews

For the development, our team has decided to use the MERN stack. We schedule regular code reviews where teammates can review each other's work and suggest improvements and they are reviewed by the team leader.