|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Desktop computer icon computer #AD , #AFFILIATE, #paid, #icon, #computer, # Desktop | Computer icon, Computer, Image computer  IS 413 Assignment 2 | Abstract  This assignment is designed to equip students with the essential skills and knowledge required for web design and development in the context of addressing real-world business challenges. The primary objectives of this assignment encompass an exploration of the foundational principles that underlie the creation of efficient web-based business systems. A particular focus is placed on web design tailored for optimal usability, the creation of prototypes to facilitate development, and the effective management of projects in the realm of web-based solutions.   |  |  |  |  | | --- | --- | --- | --- | | Student ID | Name | %Contribution | Signature | | S11133165 | Sarvesh Chand | 100 % | **S.C** | | S11165518 | Goel Lal | 100 % |  | | S11148835 | Sweta Gaunder | 100 % |  | |

Table of Contents

[1. Project Overview 1](#_Toc148436604)

[About WheelWise Insurance 1](#_Toc148436605)

[Problem Statement 1](#_Toc148436606)

[Project Scope 2](#_Toc148436607)

[Target Market 3](#_Toc148436608)

[2. Objectives 4](#_Toc148436609)

[3. Constraints and Assumptions 5](#_Toc148436610)

[Constraints 5](#_Toc148436611)

[Assumptions 6](#_Toc148436612)

[4. Projected Benefits 7](#_Toc148436613)

[5. System Architecture 8](#_Toc148436614)

[User Roles and Access Control 10](#_Toc148436615)

[Site Map 10](#_Toc148436616)

[6. System Features 13](#_Toc148436617)

[a. Policy Management: 13](#_Toc148436618)

[Functional Requirements 13](#_Toc148436619)

[Entity Relationship Diagram 14](#_Toc148436620)

[API End Points 15](#_Toc148436621)

[Workflow 15](#_Toc148436622)

[User Interface 17](#_Toc148436623)

[b. Premium Management: 18](#_Toc148436624)

[Functional Requirements 18](#_Toc148436625)

[Entity Relationship Diagram 19](#_Toc148436626)

[API End Points 19](#_Toc148436627)

[Workflow 21](#_Toc148436628)

[User Interface 22](#_Toc148436629)

[c. Customer Management: 23](#_Toc148436630)

[Functional Requirements 23](#_Toc148436631)

[Entity Relationship Diagram 24](#_Toc148436632)

[API End Points 25](#_Toc148436633)

[Workflow 26](#_Toc148436634)

[User Interface 27](#_Toc148436635)

[d. Reporting: 29](#_Toc148436636)

[Functional Requirements 29](#_Toc148436637)

[Workflow 29](#_Toc148436638)

[User Interface 30](#_Toc148436639)

[e. Payment Management 31](#_Toc148436640)

[Function Requirement 31](#_Toc148436641)

[Entity Relationship Diagram 32](#_Toc148436642)

[32](#_Toc148436643)

[API End Points 33](#_Toc148436644)

[Workflow 34](#_Toc148436645)

[User Interface 35](#_Toc148436646)

[f. Customer Requests Upgrade 37](#_Toc148436647)

[Function Requirement 37](#_Toc148436648)

[Entity Relationship Diagram 39](#_Toc148436649)

[API End Points 40](#_Toc148436650)

[Work Flow 42](#_Toc148436651)

[User Interface 42](#_Toc148436652)

[g. Claim Management 46](#_Toc148436653)

[Function Requirement 46](#_Toc148436654)

[Entity Relationship Diagram 47](#_Toc148436655)

[API End Points 47](#_Toc148436656)

[Work Flow 49](#_Toc148436657)

[User Interface 50](#_Toc148436658)

[h. Cancellation and Renewal 53](#_Toc148436659)

[Function Requirement 53](#_Toc148436660)

[Entity Relationship Diagram 54](#_Toc148436661)

[API End Points 55](#_Toc148436662)

[Work Flow 56](#_Toc148436663)

[User Interface 57](#_Toc148436664)

[7. System Functionalities 59](#_Toc148436665)

[Premium Calculation 59](#_Toc148436666)

[Claim Management Process 60](#_Toc148436667)

[Handling Customer Request 60](#_Toc148436668)

[8.Software Metrics 63](#_Toc148436669)

[9.Security 64](#_Toc148436670)

[Data Encryption 64](#_Toc148436671)

[Authentication 64](#_Toc148436672)

[Authorization 64](#_Toc148436673)

[10.Testing 65](#_Toc148436674)

[Functional Testing 65](#_Toc148436675)

[11. Deployment 67](#_Toc148436676)

[System Requirements 67](#_Toc148436677)

[Hardware Requirements 67](#_Toc148436678)

[Software Requirements 67](#_Toc148436679)

[Installation and Configuration 67](#_Toc148436680)

[Training 68](#_Toc148436681)

[Maintenance and Support 68](#_Toc148436682)

[12. Change Management and Requests 69](#_Toc148436683)

[Change request template. 69](#_Toc148436684)

[13. Conclusion 70](#_Toc148436685)

[14. Reference 71](#_Toc148436686)

# Project Overview

## About WheelWise Insurance

WheelWise is a leading car insurance provider that is dedicated to offering its customers comprehensive and affordable coverage. Its mission is to protect its customers from the financial risks associated with car ownership and accidents while providing exceptional customer service and support.

Founded in 2023, WheelWise has quickly become a trusted name in the car insurance industry. Its team of experienced and knowledgeable professionals works tirelessly to ensure that customers receive the best coverage tailored to their individual needs. WheelWise takes pride in its ability to adapt to the ever-changing landscape of the automotive and insurance industries.

At WheelWise, there is a strong belief in transparency and honesty. The company strives to provide customers with all the information they need to make informed decisions about their car insurance coverage. It aims to build a user-friendly platform allows customers to easily compare quotes and choose the best policy to suit their needs and budget.

WheelWise understands the importance of trust and reliability in the car insurance industry. That's why the company is committed to delivering exceptional customer service and support, ensuring that customers feel confident and secure in their choice of insurance provider. WheelWise values the relationships it has built with its customers and looks forward to serving their car insurance needs for years to come.

## 

## Problem Statement

WheelWise, a reputable car insurance provider, recognizes the growing need for an efficient and user-friendly web-based application to cater to both customer and employee requirements. The current absence of such a platform poses several challenges that hinder the company's ability to streamline and enhance its services. These challenges include:

**Inefficient Customer Interactions**: WheelWise currently relies on traditional methods of customer interaction, such as phone calls and emails, which can be time-consuming and less responsive to customer needs. This inefficiency can lead to delays in policy issuance, claims processing, and general inquiries, impacting customer satisfaction.

**Data Management and Accessibility**: Without a centralized web-based platform, customer and policy data are distributed across multiple systems, making it difficult for employees to access, update, and utilize the information efficiently. This fragmented approach can result in errors, data inconsistencies, and difficulties in offering tailored insurance solutions.

**Limited Customer Self-Service Options**: The absence of a customer-facing web application restricts clients from performing essential tasks independently, such as obtaining insurance quotes, policy management, and claims submissions. This lack of self-service options not only inconveniences customers but also places additional strain on employee workloads.

**Competitive Disadvantage**: In an increasingly digital world, not having a web-based application puts WheelWise at a competitive disadvantage. Many of its competitors already offer such platforms, which can attract tech-savvy customers and result in a loss of market share.

To remain at the forefront of the car insurance industry and better serve both customers and employees, WheelWise needs to develop a comprehensive web-based application that addresses these challenges. This solution will empower customers to manage their policies with ease, enable employees to work more efficiently, enhance data management and accessibility, and ultimately strengthen the company's competitive position in the market.

## Project Scope

The Wheel Wise IMS has been developed and designed to provide a comprehensive solution for managing and administering car insurance policies. The new system is web-based, which means that it can be accessed from any location with an internet connection. The user-friendly interface makes it easy for users to navigate through the system and perform various tasks.

The policy management feature of the IMS allows insurers to create, modify, and terminate policies. This feature also allows insurers to view policy details, including coverage type, policy limits, and deductibles. The claim management feature allows insurers to manage claims efficiently by allowing users to file claims, view claims details, and track the progress of claims.

The premium management feature allows insurers to manage premiums and billing efficiently. This feature allows insurers to calculate premiums based on policy details, view payment history, and track payment status. The payment management feature allows insurers to receive and process payments from customers easily.

The customer management feature allows insurers to manage customer information, including contact details, policy information, and claims history. This feature also allows insurers to view customer interactions with the company, including phone calls, emails, and chat sessions.

The reporting capabilities of the IMS allow insurers to generate various reports, including policy reports, claim reports, premium reports, and customer reports. These reports provide valuable insights into the company's performance and help to identify areas for improvement.

The non-functional requirements of the system, such as performance, security, and compatibility, have been carefully considered during the design and development of the IMS. The system has been designed to ensure optimal performance, with minimal downtime and fast response times. The system is also designed to be secure, with robust encryption and access controls. The system is compatible with various browsers and devices, making it accessible to a wide range of users.

The Wheel Wise IMS is a comprehensive solution for managing and administering car insurance policies. The system offers numerous features and benefits, including policy management, claim management, premium management, payment management, customer management, and reporting capabilities. The system is designed to be user-friendly, efficient, and compliant with relevant regulations. This technical document provides a detailed overview of the system's functionalities, design, and benefits, serving as a valuable reference for stakeholders involved in the design, development, and implementation of the Wheel Wise IMS.

## Target Market

The WheelWise website is designed to cater to a diverse range of individuals, primarily focusing on those interested in car insurance services and the efficient management of their policies. The target audience for the WheelWise website includes:

**Car Owners and Drivers**: This group comprises the core target audience. WheelWise aims to serve individuals who own or drive cars and require car insurance coverage. This includes a wide spectrum of drivers, from experienced vehicle owners to new drivers seeking insurance for the first time.

**Insurance Seekers**: The website is also geared toward individuals actively seeking car insurance solutions. Whether they are in the process of purchasing a new vehicle or looking to switch insurance providers, WheelWise aims to provide them with comprehensive information and user-friendly tools to obtain quotes and explore insurance options.

**Existing WheelWise Customers**: WheelWise customers who have already purchased policies are a significant part of the target audience. The website serves as a platform for them to manage their existing policies, file claims, and access various customer support services.

**Insurance Professionals and Agents**: While the primary focus is on customers, the website may also provide a dedicated section or portal for insurance professionals and agents associated with WheelWise. This segment of the audience may require tools and resources to assist them in their roles.

**Individuals Seeking Information**: The website is a valuable resource for individuals seeking information about car insurance, industry trends, and best practices. Whether they are in the initial stages of car ownership or curious about insurance-related topics, WheelWise offers informative content.

By targeting this diverse audience, the WheelWise website aims to provide a comprehensive online experience that meets the needs of car owners, insurance seekers, existing customers, and those seeking information about the automotive insurance industry. It emphasizes accessibility, ease of use, and informative content to engage and serve a wide range of users.

# Objectives

The objectives of the WheelWise IMS are to:

* Simplify policy management for insurers by enabling them to create, view, update, and delete car insurance policies with different coverage types.
* Enhance claim management by allowing insurers and claims staff to manage claims related to the three main coverage types.
* Provide customers with an intuitive platform to view and manage their car insurance policies, file claims, and track claim status.
* Improve premium management by offering insurers a dashboard to manage and update premium amounts for various car insurance policies.
* Ensure seamless integration with payment gateways for processing premium payments and claim settlements.
* Enhance customer management by allowing authorized users to create, modify, and search customer records.
* Offer reporting capabilities to generate policy, claims, and customer reports in various formats, such as PDF.

# Constraints and Assumptions

## Constraints

Some of the constraint identified for the project are:

**Budget Constraints**: Limited financial resources for the project could impact the scope, features, and quality of the web application.

**Time Constraints**: Meeting project deadlines and launch dates is crucial, and delays could result in missed opportunities or customer dissatisfaction.

**Regulatory Compliance**: The car insurance industry is heavily regulated. Ensuring the application complies with all relevant laws and regulations is a constraint that may impact development.

**Security and Privacy**: Safeguarding customer and company data is essential. Security and privacy constraints may require additional measures and testing, potentially affecting the project timeline.

**Data Migration**: Migrating existing customer data to the new platform while maintaining data integrity is a challenging task.

**Technical Constraints**: The technology stack chosen for the web application may impose constraints, such as compatibility issues or limitations in scalability.

**User Accessibility**: Ensuring the application is accessible to individuals with disabilities may be a regulatory requirement and a design constraint.

**User Experience (UX) Design**: The web application must be user-friendly and intuitive, which can be challenging to achieve and may affect development time.

**Mobile Responsiveness**: The application may need to be responsive to various devices and screen sizes, which can be technically demanding.

**Content and Data Accuracy**: Maintaining accurate and up-to-date content and data is essential but can be resource-intensive.

**Testing and Quality Assurance**: Rigorous testing is crucial to ensure the application's functionality and security, and this can impose constraints on the project timeline.

**Scalability**: Preparing the application for potential growth and increased user load can be a constraint on development efforts.

**Change Management**: Preparing employees for the transition to the new system and addressing resistance to change is a constraint that should be considered.

**Vendor and Third-Party Dependencies**: If third-party services or vendors are involved, their constraints and timelines need to be coordinated.

These constraints should be carefully managed to ensure the successful development and implementation of the web-based application for both customers and employees.

## Assumptions

Here are some common assumptions for this project:

**Stakeholder Support**: It is assumed that there is strong support from key stakeholders, including senior management, for the development of the web-based application.

**Adequate Budget**: The project assumes that there is a budget in place to cover development, testing, and implementation costs.

**Availability of Skilled Workforce**: It's assumed that the project team, including developers, designers, and testers, has the required skills and expertise to complete the project successfully.

**Data Availability**: Necessary data, such as customer information and policy details, is assumed to be accessible for integration into the application.

**Hardware and Software Infrastructure**: Adequate hardware and software resources are assumed to be available to support the development and hosting of the web application.

**User Adoption**: It is assumed that customers and employees will embrace the new application and use it effectively once it is launched.

**Network and Connectivity**: The project assumes that users will have reliable internet access and the necessary network infrastructure to access the web application.

**Testing and Quality Assurance**: Adequate time and resources are assumed to be allocated for thorough testing and quality assurance to identify and resolve issues.

**Project Timeline**: The project assumes that the timeline is realistic and that milestones can be achieved according to the schedule.

**Change Management**: The project assumes that change management strategies will be effective in helping employees transition to the new system.

**Vendor Reliability**: If third-party vendors are involved, it is assumed that they will deliver products and services as agreed upon.

**Competitive Landscape**: Assumptions about WheelWise's competitive positioning, market trends, and customer expectations are considered when designing the application.

**Scalability**: The project assumes that the infrastructure and architecture can accommodate future growth and increasing user demand.

**Data Security**: It is assumed that security measures, such as encryption and access controls, will be effective in protecting user and company data.

These assumptions serve as a foundation for project planning, but they should be regularly reviewed and validated throughout the project to ensure they remain accurate. Adjustments may be necessary as the project progresses and new information becomes available.

# Projected Benefits

**Increased Efficiency**: By automating various processes and reducing manual tasks for insurers, claims staff, and customers, the IMS will improve efficiency and productivity.

**Enhanced Customer Satisfaction**: A user-friendly interface and streamlined processes will result in a seamless experience for customers managing their car insurance policies and claims, leading to higher satisfaction levels.

**Improved Decision Making**: Reporting capabilities will provide valuable insights into policy and claims data, enabling insurers and claims staff to make more informed decisions.

**Reduced Operational Costs**: The IMS will help insurers and claims staff manage car insurance policies and claims more effectively, leading to reduced operational costs and improved profitability.

**Regulatory Compliance**: Adhering to relevant regulations, such as IRDA and PCI DSS, the IMS will ensure that insurers remain compliant with industry standards, minimizing the risk of fines or penalties.

The Wheel Wise Insurance Management System offers a tailored and efficient solution for managing car insurance policies, claims, and customer interactions, ultimately resulting in a better overall experience for insurers and policyholders. By investing in the development and implementation of the IMS, insurers can expect to see significant improvements in operational efficiency, customer satisfaction, and regulatory compliance.

# System Architecture

The overall system architecture for the Wheel Wise Insurance Management System (IMS) can be divided into three main layers: Presentation Layer, Business Logic Layer, and Data Access Layer. These layers interact with each other to provide the desired functionality and ensure a modular and maintainable design.

1. Presentation Layer:

This layer is responsible for providing the user interface (UI) through which users interact with the system. The UI is developed using React JS using the Material UI Library and is compatible with modern web browsers. It communicates with the Business Logic Layer through APIs to request and receive data, which is then displayed to the user.

1. Business Logic Layer:

The Business Logic Layer, also known as the Application Layer, contains the core logic and processing of the IMS. It is built using a server-side language such as ASP.NET. This layer is responsible for processing user requests received from the Presentation Layer and interacting with the Data Access Layer to retrieve or store data. It also ensures that the appropriate business rules, validations, and security measures are in place.

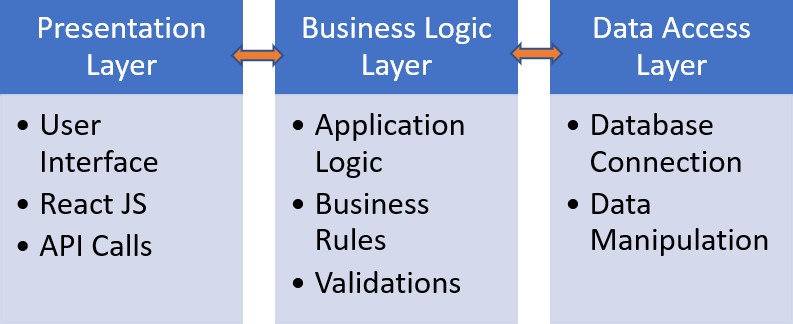
1. Data Access Layer:

The Data Access Layer is responsible for managing the connection and interactions with the database. It handles data storage, retrieval, and manipulation using a database management system (DBMS) like Microsoft SQL Server. This layer abstracts the database operations from the Business Logic Layer, allowing for easy updates or changes to the database without affecting the application logic.

The relationships between these layers can be described as follows:

* The Presentation Layer sends requests for data or actions to the Business Logic Layer through APIs.
* The Business Logic Layer processes these requests, applying any necessary business rules, validations, or security measures, and then communicates with the Data Access Layer to retrieve or store the required data.
* The Data Access Layer interacts with the database to perform the requested operations and returns the data or results to the Business Logic Layer.
* The Business Logic Layer then sends the processed data or results back to the Presentation Layer, which displays the information to the user.

The diagram below illustrates the overall system architecture, including the components, layers, and their relationships:



This modular architecture ensures that each layer has a distinct responsibility, promoting maintainability, scalability, and flexibility within the Wheel Wise Insurance Management System.

Apart from the above layered architecture, the Wheel Wise Insurance Management System utilizes a combination of the following software architecture styles:

Client-Server Architecture: The IMS follows a client-server model, where the client (web browser) interacts with the server-side application through API calls. The server processes the requests, communicates with the database, and returns the results to the client.

Service-Oriented Architecture (SOA): The system exposes its functionalities through APIs, allowing different components of the system to communicate with each other using standardized protocols. This approach enables flexibility, reusability, and scalability of the system components.

While the Wheel Wise Insurance Management System primarily employs a combination of Layered, Client- Server, and Service-Oriented Architectures, it can be extended or adapted to incorporate other architecture styles, such as Microservices or Event-Driven, based on the specific requirements and scalability needs of the system.

In the Wheel Wise Insurance Management System, Aspect-Oriented Software Development (AOSD) is employed to enhance modularity and maintainability by effectively separating cross-cutting concerns from the core functionality. AOSD allows us to encapsulate these concerns, such as logging, security, and performance monitoring, into separate aspects that can be woven into the main system's codebase at specific points. This approach reduces code redundancy and complexity, resulting in a cleaner and more manageable system architecture. By utilizing AOSD, the IMS is better equipped to handle changes in requirements or functionality, enabling developers to focus on the core business logic without being hindered by the intricacies of cross-cutting concerns.

## User Roles and Access Control

The Wheel Wise Insurance Management System implements role-based access control to ensure that users can only access the functionalities relevant to their role. The system defines the following user roles:

1. Administrator: Responsible for managing the overall system, including user management, system configuration, and reporting.
2. Insurer: Manages car insurance policies, premiums, customers, and claims.
3. Claims Staff: Handles claim processing, including approving or rejecting claims.
4. Customer: Accesses and manages their car insurance policies, files claims, and tracks claim status.

## Site Map

Administrator Level

Insurer Level

Claim Staff Level

Customer Level

# System Features

## a. Policy Management:

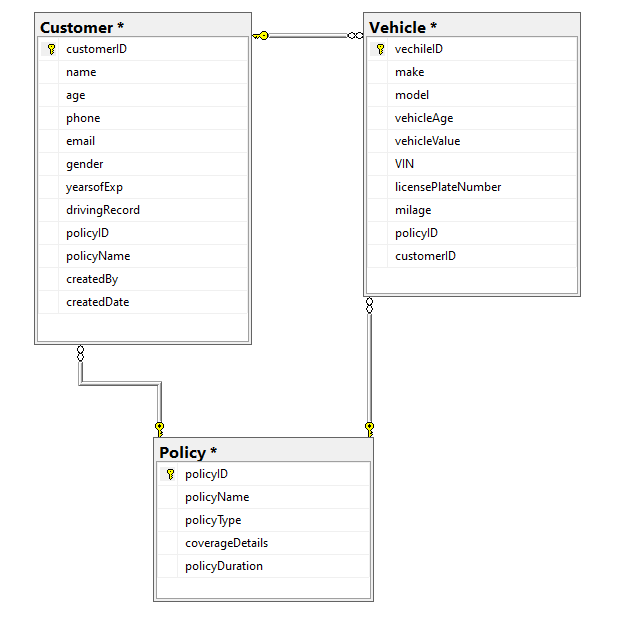
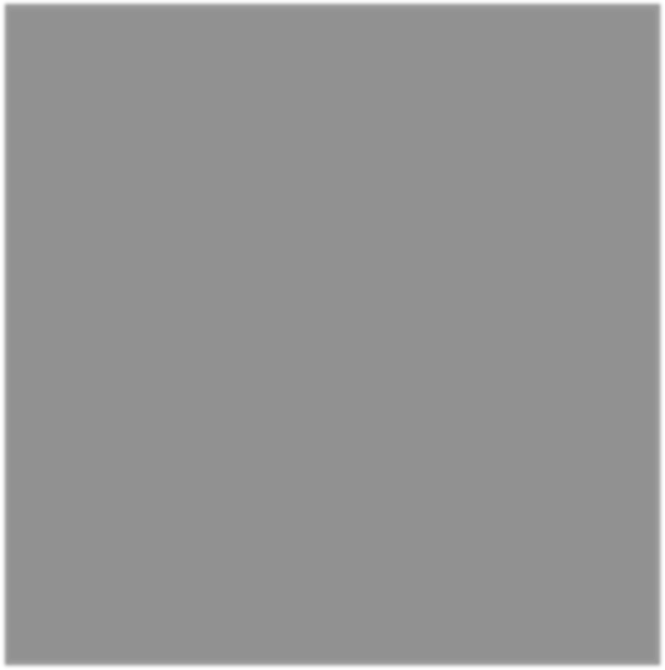
### Functional Requirements

The functional requirements of the Policy Management feature in the Wheel Wise Insurance Management System are as follows:

1. Create Policy: The system should allow authorized users (e.g., insurers) to create new car insurance policies with different coverage types, such as basic liability, comprehensive, and collision coverage. Users should be able to input necessary policy details, including Customer information, vehicle details, coverage selections, policy start and end dates, and premium amounts.
2. View Policy: Authorized users should be able to view the details of existing car insurance policies, including policy number, Customer information, vehicle details, coverage information, policy duration, and premium amounts.
3. Update Policy: The system should enable authorized users to update the details of existing car insurance policies, such as modifying coverage types, adjusting policy durations, or updating Customer and vehicle information. Any changes made to the policy should be recorded in the system for tracking and auditing purposes.
4. Delete Policy: Authorized users should be able to delete existing car insurance policies when necessary, such as in cases of policy cancellation or erroneous policy creation. Deletion should be handled with appropriate safeguards to prevent accidental data loss.
5. Search and Filter Policies: The system should provide authorized users with the ability to search for specific policies using various criteria, such as policy number, Customer name, vehicle information, or coverage type. Additionally, users should be able to filter and sort the list of policies based on different attributes, such as policy creation date, coverage type, or policy status.

The data model for policies in the Wheel Wise Insurance Management System consists of several entities, their attributes, and the relationships between them. The main entities in this data model are Policy, Customer, and Vehicle.

### Entity Relationship Diagram



### API End Points

Below are the API endpoints related to policy management, including their expected input parameters, output, and error handling:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| API  Function | Method | API Call | Input Parameters | Output | Error Handling |
| 1. Get all policies | GET | http://localhost:5179/ap i/Policy | None | A list of policy  objects | If no policies are found, return an  empty list |
| 2. Create a new policy | POST | http://localhost:5179/ap i/Policy | Policy object (JSON format) | The created policy object with a new ID | If input data is invalid or incomplete, return a 400 Bad Request status with an error  message |
| 3. Get a specific policy by ID | GET | http://localhost:5179/ap i/Policy/{id} | Policy ID (integer) | The policy object with the  specified ID | If the policy ID is not found, return a 404 Not Found status with an error message |
| 4.  Update a policy by ID | PUT | http://localhost:5179/ap i/Policy/{id} | Policy ID (integer), updated policy object (JSON format) | The updated policy object | If the policy ID is not found, return a 404 Not Found status with an error message. If input data is invalid or incomplete, return a  400 Bad Request  status with an error message. |
| 5. Delete a policy by ID | DELETE | http://localhost:5179/ap i/Policy/{id} | Policy ID (integer) | A  confirmatio message indicating successful  deletion | If the policy ID is not found, return a 404 Not Found status with an error message |

### Workflow

The typical workflow for policy management in the Wheel Wise Insurance Management System involves several user roles, each with specific permissions, to ensure a secure and efficient process. The main user roles include Insurers, Claims Staff, and Customer.

1. Insurer:

Permissions:

* Create, view, update, and delete policies.
* Manage policy details, including coverage types, policy duration, and premium amounts.
* Search and filter policies.

Workflow:

* 1. The insurer logs in to the system with their credentials.
  2. The insurer creates a new policy by entering customer information, vehicle details, coverage selections, policy duration, and premium amounts.
  3. The insurer can view existing policies, modify policy details, or delete policies as needed.

1. Claims Staff:

Permissions:

* View policy details.
* Search and filter policies.

Workflow:

* 1. The claims staff logs in to the system with their credentials.
  2. The claims staff can view policy details when processing claims to verify coverage types, policy duration, and other relevant information.
  3. The claims staff can search for specific policies using various criteria and filter or sort the list of policies based on different attributes.

1. Customer:

Permissions:

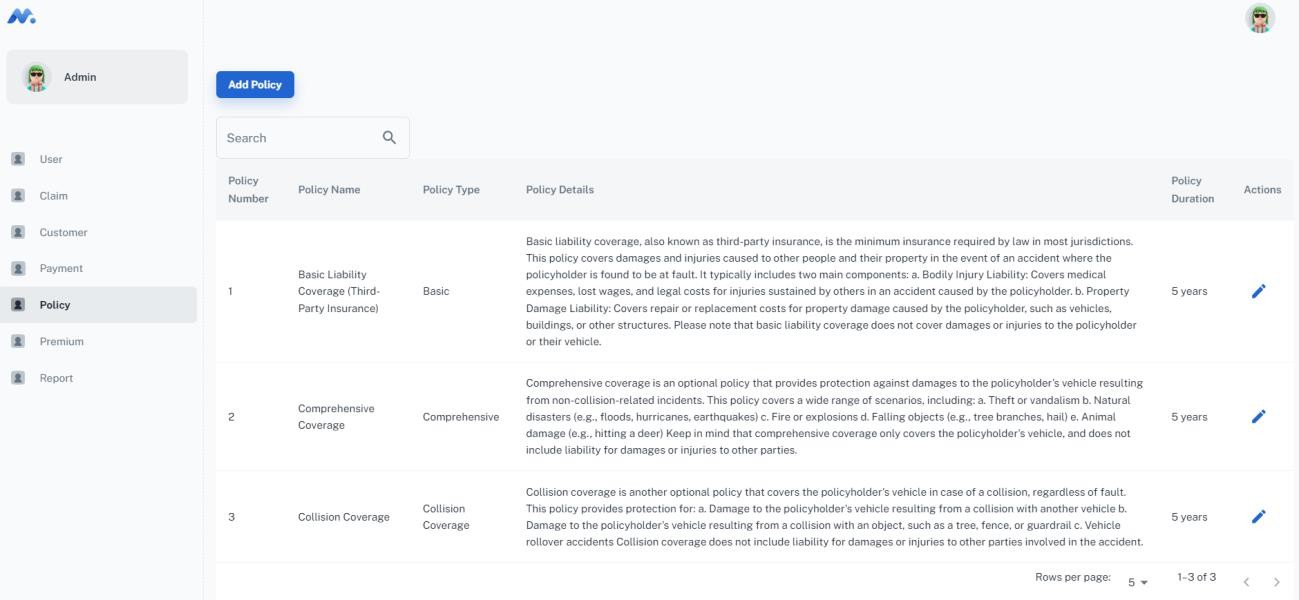
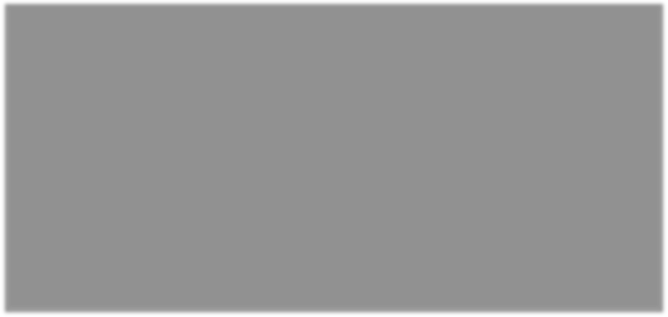
* View their own policies.
* File claims.
* Track claim status.

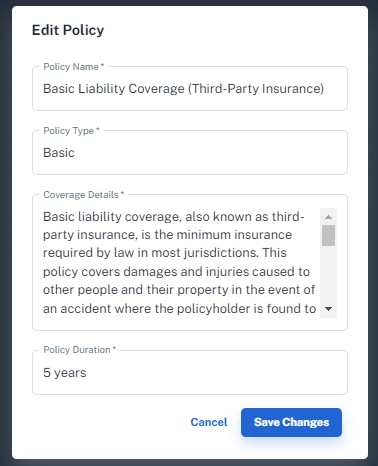
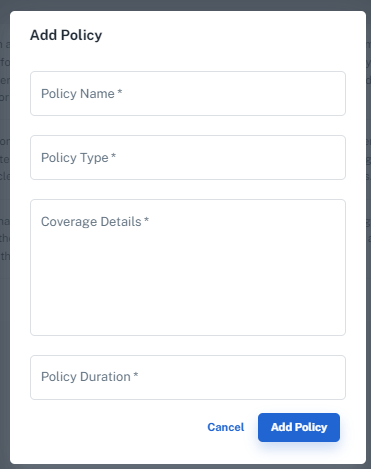
Workflow:

* 1. The customer logs in to the system with their credentials.
  2. The customer can view the details of their own car insurance policies, including policy number, coverage information, policy duration, and premium amounts.
  3. The customer can file claims related to their policies and track the status of their claims.

This typical workflow for policy management ensures that each user role has the appropriate permissions to perform their specific tasks, providing a secure and streamlined process for managing car insurance policies in the Wheel Wise Insurance Management System.

### User Interface



## b. Premium Management:

### Functional Requirements

The functional requirements of the Premium Management feature in the Wheel Wise Insurance Management System are as follows:

Calculate Premiums: The system should automatically calculate insurance premiums for various car insurance policies based on factors such as vehicle information and customers driving history.

Update Premium Amounts: Authorized users (e.g., insurers) should be able to view and update premium amounts for various car insurance policies as needed, such as when customer risk factors change.

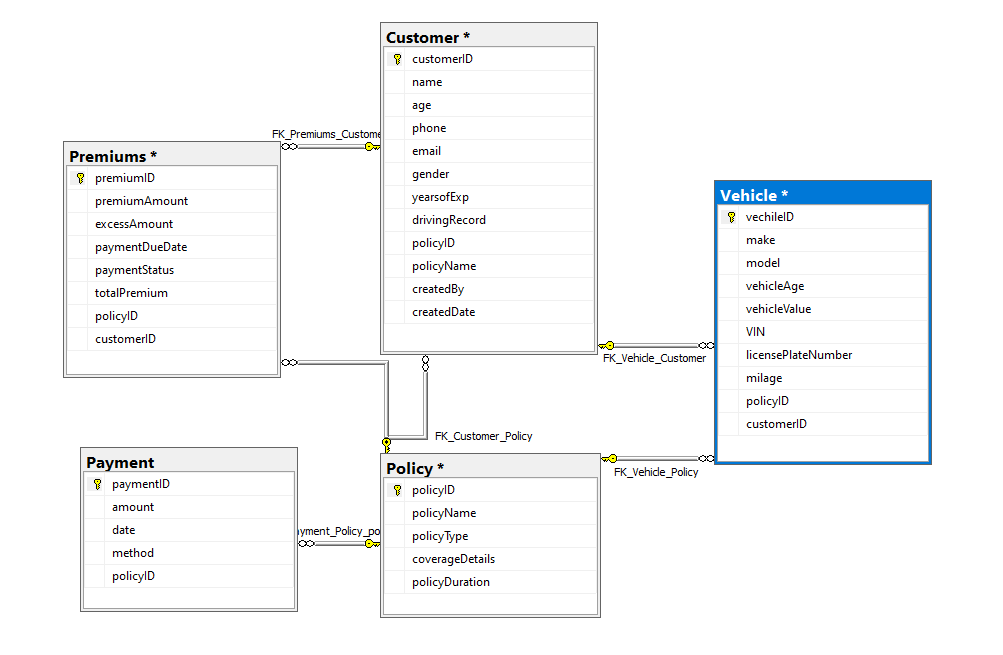
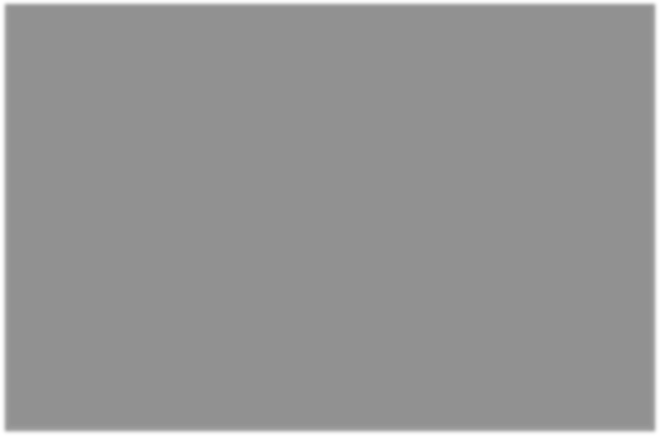
Process Premium Payments: The system should facilitate the processing of premium payments by integrating with payment gateways to support various payment methods, such as credit cards, debit cards, and electronic funds transfers (EFT).

Manage Payment Schedules: Authorized users should be able to manage payment schedules for customers, including setting up recurring payment plans and sending payment reminders or notifications.

Record Payment Transactions: The system should record all premium payment transactions, including payment dates, amounts, methods, and confirmation numbers, to maintain an accurate payment history for each customer.

Handle Payment Failures: The system should provide mechanisms to handle payment failures, such as sending payment failure notifications to customers, retrying failed payments, or applying late payment fees as per the policy terms.

### Entity Relationship Diagram



### API End Points

Below are the API endpoints related to premium management, including their expected input parameters, output, and error handling:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| API  Function | Method | API Call | Input Parameter  s | Output | Error Handling |
| 1. Get all premium s | GET | http://localhost:5179/api/Premiums | None | A list of premium objects | If no  premiums are found, return an  empty list |
| 2. Create a new premium | POST | http://localhost:5179/api/Premiums | Premium object (JSON  format) | The created premium object with a new ID | If input data is invalid or incomplete  , return a  400 Bad Request status with an error  message |
| 3. Get a specific premium by ID | GET | http://localhost:5179/api/Premiums/{i d} | Premium ID  (integer) | The premium object with the specified ID | If the  premium ID is not found, return a  404 Not Found status with an error  message |
| 4. Update a premium by ID | PUT | http://localhost:5179/api/Premiums/{i d} | Premium ID  (integer), updated premium object (JSON  format) | The updated premium object | If the  premium ID is not found, return a  404 Not Found status with an error message. If input data is invalid or incomplete  , return a  400 Bad Request  status with an error message. |
| 5. Delete | DELET | http://localhost:5179/api/Premiums/{i | Premium | A | If the |
| a | E | d} | ID | confirmation | premium |
| premium |  |  | (integer) | message | ID is not |
| by ID |  |  |  | indicating | found, |
|  |  |  |  | successful | return a |
|  |  |  |  | deletion | 404 Not |
|  |  |  |  |  | Found |
|  |  |  |  |  | status with |
|  |  |  |  |  | an error |
|  |  |  |  |  | message |

### Workflow

The typical workflow for premium management in the Wheel Wise Insurance Management System involves several user roles, each with specific permissions, to ensure a secure and efficient process. The main user roles include Insurers (or Insurance Agents) and customers (Customers).

Insurer:

Permissions:

Calculate and update premium amounts for various car insurance policies. Manage payment schedules for customers.

Generate premium and payment reports. Workflow:

1. The insurer logs in to the system with their credentials.
2. The system calculates, or insurer updates premium amounts for car insurance policies based on coverage type, limits, deductibles, vehicle information, and the customer's driving history.
3. The system sets up and manages payment schedules for customers on an annual basis and sends payment reminders or notifications.

Customer:

Permissions:

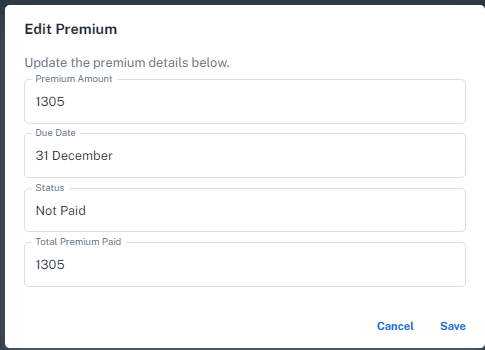
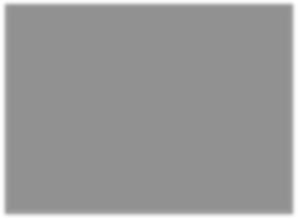
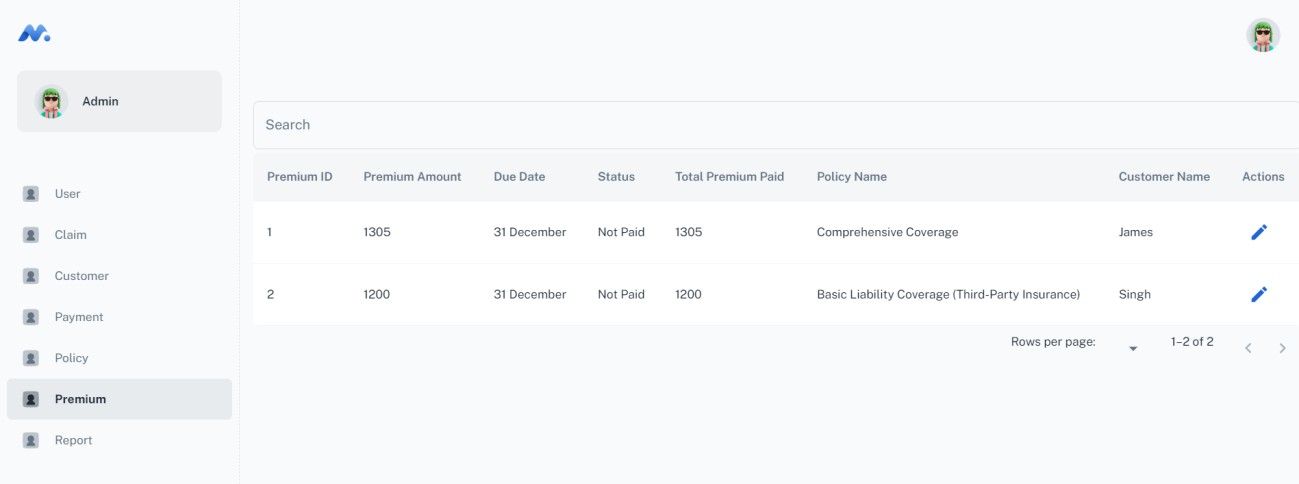
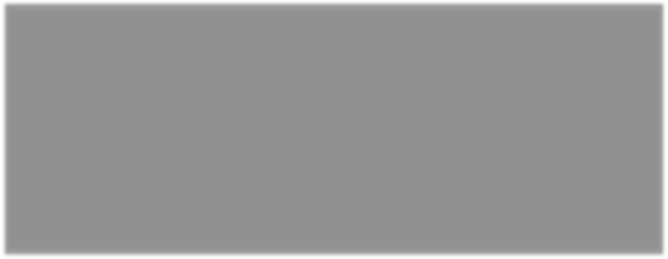
View premium details and payment schedules. Make premium payments.

Workflow:

1. The Customer logs in to the system with their credentials.
2. The Customer views their premium details and payment schedules for their car insurance policy.
3. The Customer makes premium payments using the integrated payment gateway, which supports various payment methods.

The system records the payment transaction details, updates the next payment due date, and sends a payment confirmation to the customer.

### User Interface



## c. Customer Management:

### Functional Requirements

The functional requirements of the Customer Management feature in the Wheel Wise Insurance Management System are as follows:

Create Customer Profile: The system should allow authorized users (e.g., insurers) to create new customer profiles with essential customer details, including name, date of birth, address, phone number, and email address.

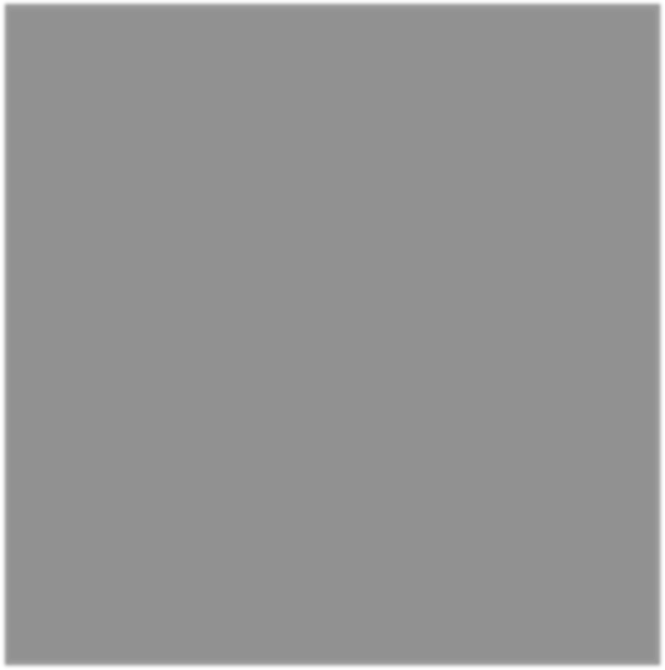
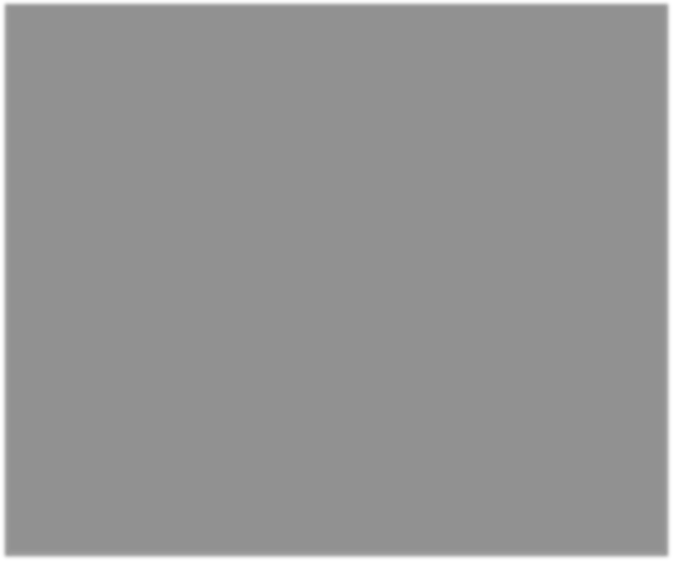
View Customer Profile: Authorized users should be able to view the details of existing customer profiles, including customer identification numbers, names, dates of birth, addresses, phone numbers, and email addresses.

Update Customer Profile: The system should enable authorized users to update the details of existing customer profiles, such as modifying personal information, updating contact details, or adding secondary contact information. Any changes made to the customer profile should be recorded in the system for tracking and auditing purposes.

Delete Customer Profile: Authorized users should be able to delete existing customer profiles when necessary, such as in cases of duplicate profiles or erroneous profile creation. Deletion should be handled with appropriate safeguards to prevent accidental data loss.

Search and Filter Customers: The system should provide authorized users with the ability to search for specific customers using various criteria, such as name, customer identification number, phone number, or email address. Additionally, users should be able to filter and sort the list of customers based on different attributes, such as account creation date or the number of active policies.

### Entity Relationship Diagram



### API End Points

Below are the API endpoints related to customer management, including their expected input parameters, output, and error handling:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| API  Function | Method | API Call | Input  Parameter s | Output | Error Handling |
| 1. Get all customer s | GET | http://localhost:5179/api/Customer | None | A list of customer objects | If no  customers are found,  return an empty list |
| 2. Create a new customer | POST | http://localhost:5179/api/Customer | Customer object (JSON  format) | The created customer object with a new ID | If input data is invalid or incomplete  , return a  400 Bad Request status with an error  message |
| 3. Get a specific customer by ID | GET | http://localhost:5179/api/Customer/{i d} | Customer ID  (integer) | The customer object with the specified ID | If the  customer ID is not found, return a  404 Not Found status with an error  message |
| 4. Update a customer by ID | PUT | http://localhost:5179/api/Customer/{i d} | Customer ID  (integer), updated customer object (JSON  format) | The updated customer object | If the  customer ID is not found, return a  404 Not Found status with an error message. If input data is invalid or incomplete  , return a |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 400 Bad Request status with an error  message. |
| 5. Delete | DELET | http://localhost:5179/api/Customer/{i | Customer | A | If the |
| a | E | d} | ID | confirmation | customer |
| customer |  |  | (integer) | message | ID is not |
| by ID |  |  |  | indicating | found, |
|  |  |  |  | successful | return a |
|  |  |  |  | deletion | 404 Not |
|  |  |  |  |  | Found |
|  |  |  |  |  | status with |
|  |  |  |  |  | an error |
|  |  |  |  |  | message |

### Workflow

The typical workflow for customer management in the Wheel Wise Insurance Management System involves several user roles, each with specific permissions, to ensure a secure and efficient process. The main user roles include Insurers, Claims Staff, and Customers.

1. Insurer:

Permissions:

* Create, view, update, and delete customer profiles.
* Manage customer details, including personal information and contact details.
* Search and filter customer profiles.

Workflow:

* 1. The insurer logs in to the system with their credentials.
  2. The insurer creates a new customer profile by entering essential customer details.
  3. The insurer can view existing customer profiles, modify customer details, or delete customer profiles as needed.
  4. The insurer can search for specific customers using various criteria and filter or sort the list of customers based on different attributes.

1. Claims Staff:

Permissions:

* View customer profiles.
* Search and filter customer profiles.

Workflow:

* 1. The claims staff logs in to the system with their credentials.
  2. The claims staff can view customer profiles when processing claims to verify customer information and contact details.
  3. The claims staff can search for specific customers using various criteria and filter or sort the list of customers based on different attributes.

1. Customer:

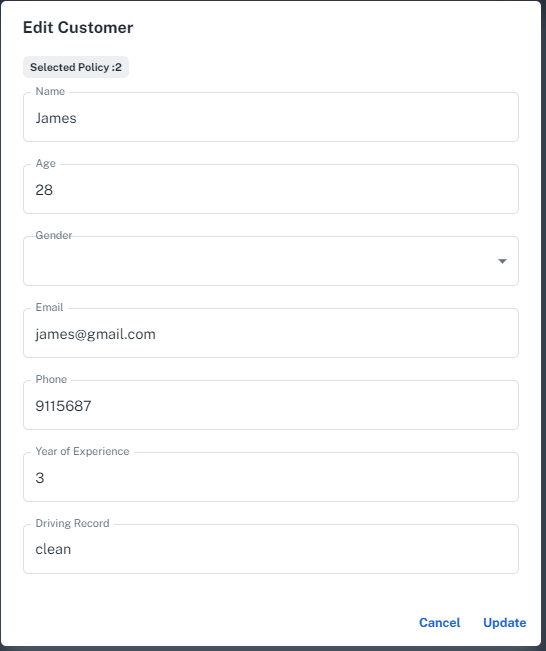
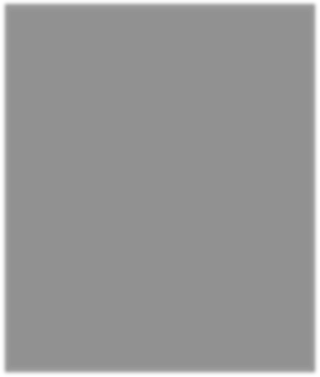
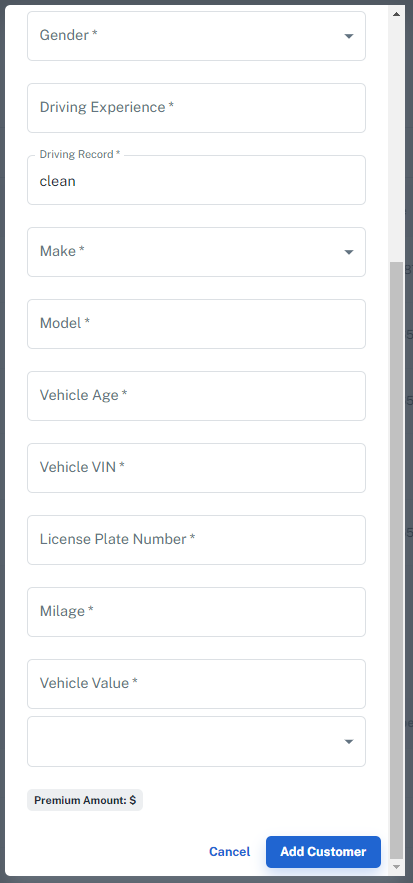
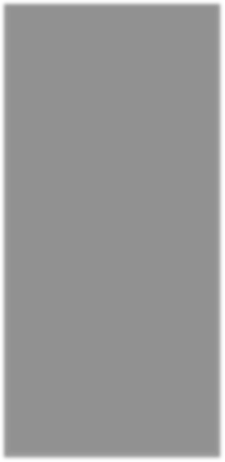
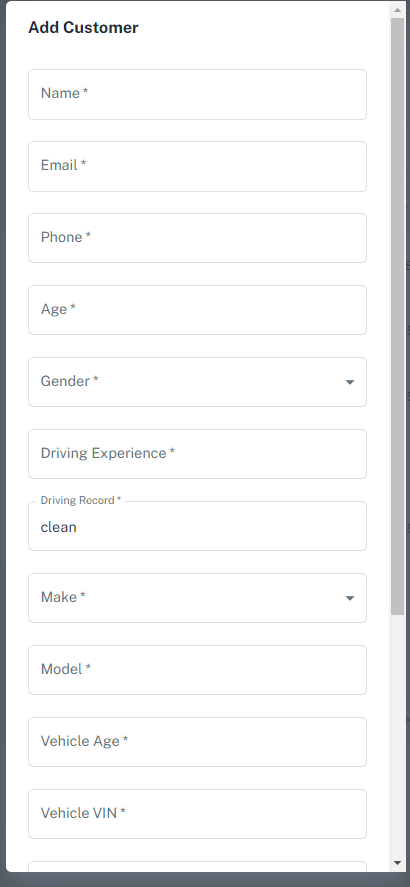
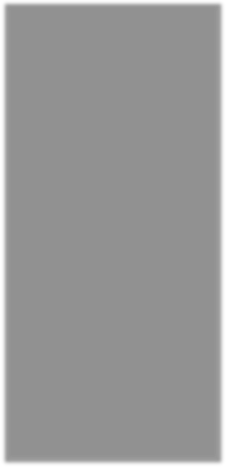
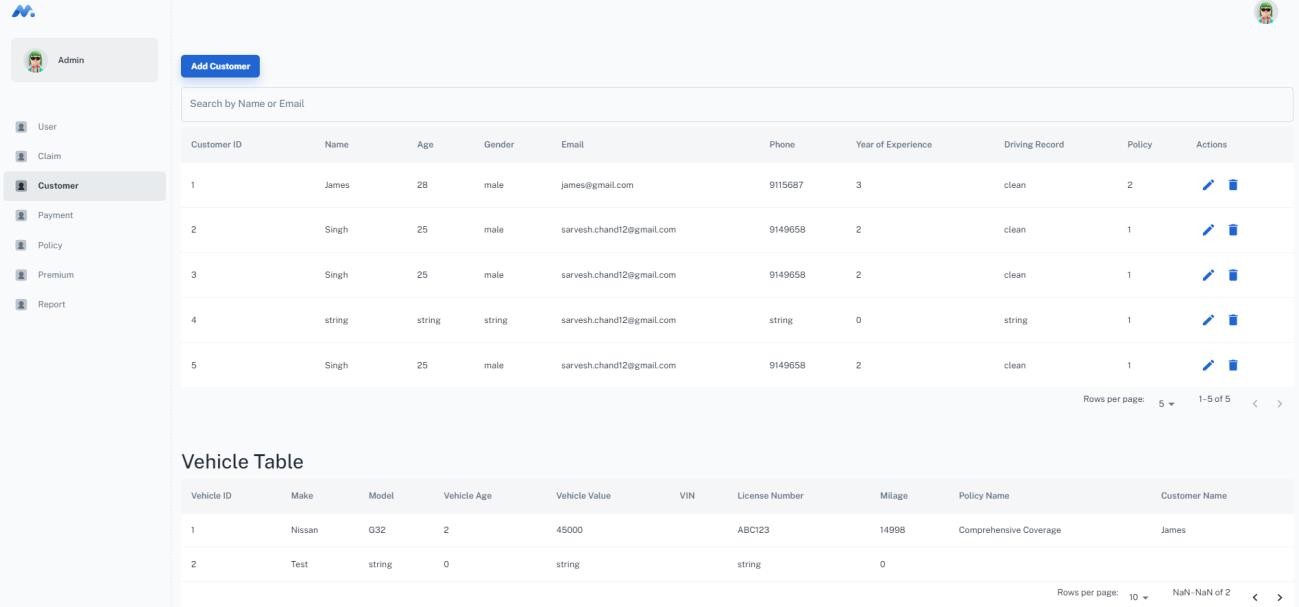
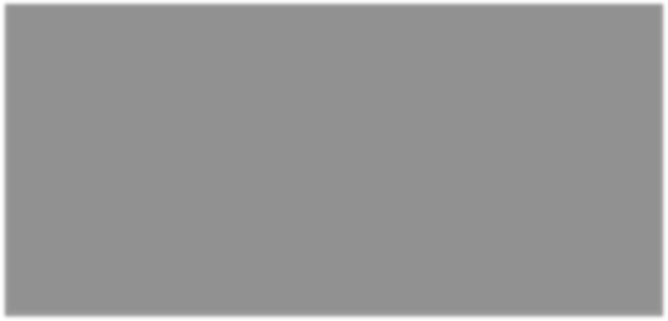
Permissions:

- View and update their own customer profile.

Workflow:

* 1. The customer logs in to the system with their credentials.
  2. The customer can view their own customer profile, including personal information and contact details.

### User Interface



## d. Reporting:

### Functional Requirements

The functional requirements of the Reporting feature in the Wheel Wise Insurance Management System are as follows:

1. Generate Reports: The system should allow authorized users to generate various reports related to policies, claims, and customers, providing valuable insights into business operations and performance.
2. Report Types: The system should support multiple types of reports, including:

* Policy Reports: Reports on the types policies and how much customers and premiums each policy has.
* Claims Reports: Reports on open claims, closed claims, approved claims, rejected claims, and claim processing time.
* Customer Reports: Reports on new customers, customer demographics, and customer policy distribution.

3.. Export Report Data: The system should allow authorized users to export report data in PDF format. The reporting feature utilizes the API’s of other features such as policy and customer to generate the reports.

### Workflow

The typical workflow for generating and viewing reports in the Wheel Wise Insurance Management System involves several user roles, each with specific permissions, to ensure a secure and efficient process. The main user roles include Admins, Insurers and Claims Staff.

1. Admins:

Permissions:

* Generate policy and customer reports.
* View and export generated reports in PDF format.

Workflow:

* 1. The admins logs in to the system with their credentials.
  2. The admin selects the desired report type (policy or customer) and specifies any required filters or parameters.
  3. The system generates the report based on the provided parameters and displays it to the admin.
  4. The insurer can view the report and export it in PDF format for further analysis or sharing with external parties.

1. Insurer:

Permissions:

* Generate policy and customer reports.
* View and export generated reports in PDF format.

Workflow:

* 1. The insurer logs in to the system with their credentials.
  2. The insurer selects the desired report type (policy or customer) and specifies any required filters or parameters.
  3. The system generates the report based on the provided parameters and displays it to the insurer.
  4. The insurer can view the report and export it in PDF format for further analysis or sharing with external parties.

1. Claims Staff:

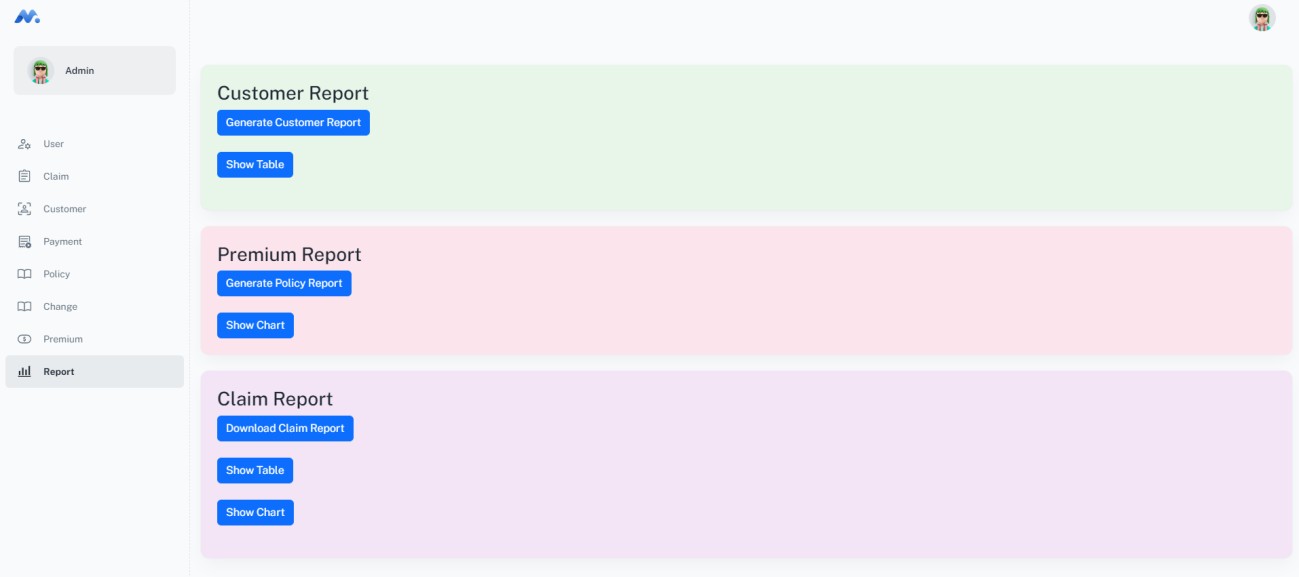
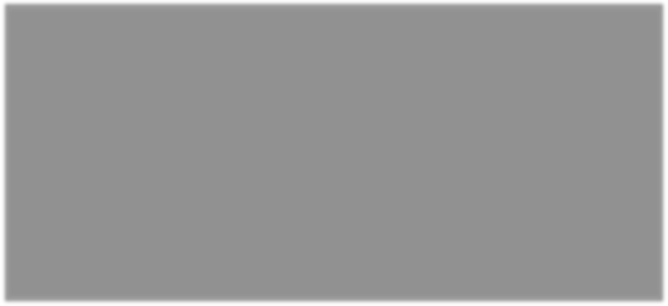
Permissions:

* Generate claims reports.
* View and export generated reports in PDF format.

Workflow:

* 1. The claims staff logs in to the system with their credentials.
  2. The claims staff selects the claims report type and specifies any required filters or parameters.
  3. The system generates the report based on the provided parameters and displays it to the claims staff.
  4. The claims staff can view the report and export it in PDF format for further analysis or sharing with external parties.

### User Interface



## Payment Management

### Function Requirement

The functional requirements of the Payment feature in the Wheel Wise Insurance Management System are as follows:

A Payment Receipt:

The system should integrate with one or more payment gateways to facilitate the receiving of payments from customers. This feature will allow customers to make payments for their policy premiums and any other charges directly from their dashboard.

Upon successful payment, the system should automatically update the status of the corresponding policy or claim. For policy premiums, the status may be updated to "Paid," and for claims, the status could be updated to "Settled".

The system should also generate a digital receipt for each payment, which can be sent to the customer via email or can be made available for download from the customer's dashboard.

1. Payment Refunds:

In situations where a refund is necessary, such as when a policy is cancelled or a claim is rejected, the system should have the functionality to facilitate the refund process.

When a refund is initiated, the system should automatically update the status of the related policy or claim to reflect the initiation of the refund process. Once the refund has been successfully processed through the payment gateway, the status should then be updated to "Refunded".

1. Payment Tracking and History:

The system should keep track of all payment activities, including payments received and refunds issued. This information should be stored in a way that it can easily be retrieved for review or audit purposes.

1. Secure Payment Processing:

The system should ensure that all payment information is processed securely. This means using encryption to protect sensitive data, such as credit card numbers, and ensuring that the system is compliant with all relevant payment card industry (PCI) standards.

1. Payment Notifications:

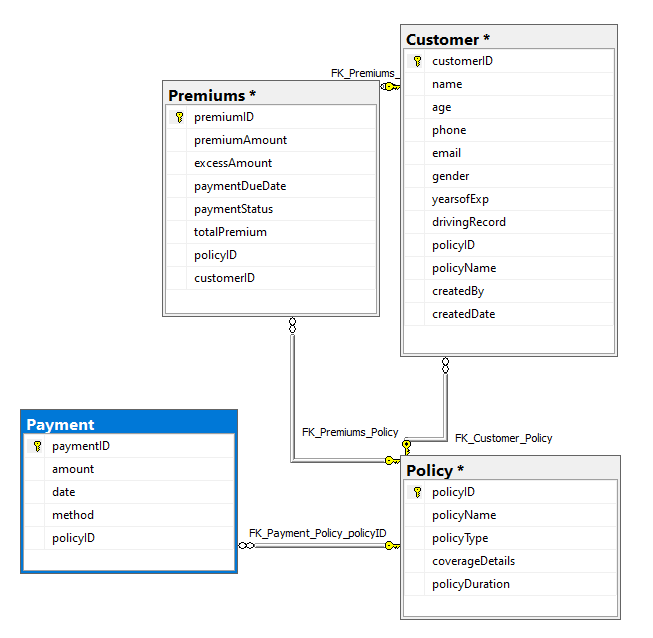
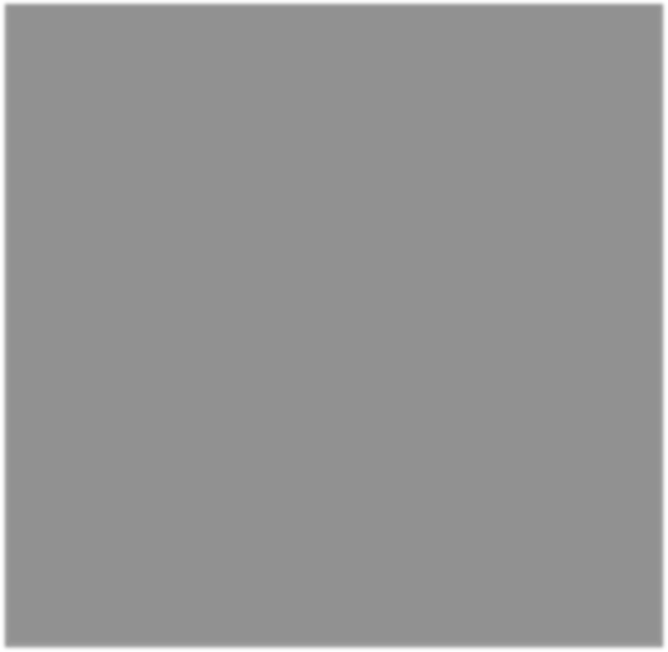
Whenever a payment is made or a refund is issued, the system should automatically generate a notification. This notification should be sent to the customer, and it should provide details about the transaction, including the amount, the date, and the associated policy or claim.

1. Payment Failures:

In cases where a payment fails due to insufficient funds, expired credit card, or any other reason, the system should notify the customer about the failed transaction and provide suggestions for resolving the issue.

### Entity Relationship Diagram

### 



### API End Points

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API**  **Function** | **Method** | **API Call** | **Input**  **Parameters** | **Output** | **Error Handling** |
| 1. Get all payments | GET | /api/Payment | None | A list of payment objects | If no payments are found, return an empty list |
| 2. Create a new payment | POST | /api/Payment | Payment object (JSON format) | The created payment object with a new ID | If input data is invalid or incomplete, return a 400 Bad Request status with an error message |
| 3. Get a specific payment by ID | GET | /api/Payment  /{id} | Payment ID (integer) | The payment object with the specified ID | If the payment ID is not found, return a 404 Not Found status with an error message |
| 4.  Update a payment by ID | PUT | /api/Payment t/{id} | Payment ID (integer), updated payment object (JSON format) | The updated payment object | If the payment ID is not found, return a 404 Not Found status with an error message. If input data is invalid or incomplete, return a 400 Bad Request status with an error message. |
| 5. Delete a payment by ID | DELETE | /api/Payment/{id} | Payment ID (integer) | A  confirmation message indicating successful deletion | If the payment ID is not found, return a 404 Not Found status with an error message |

### Workflow

#### Receive Payment

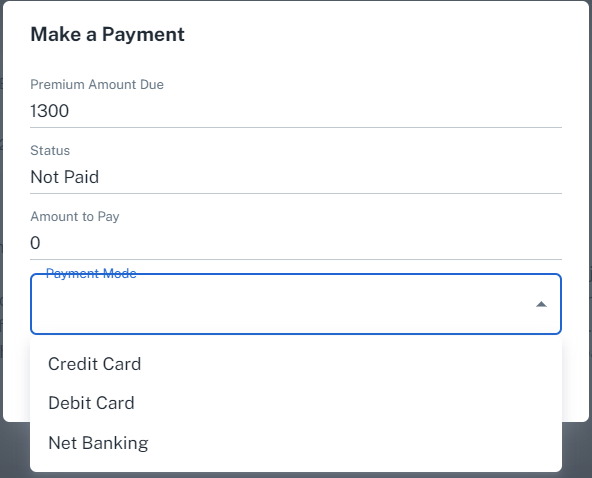
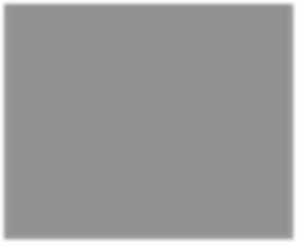
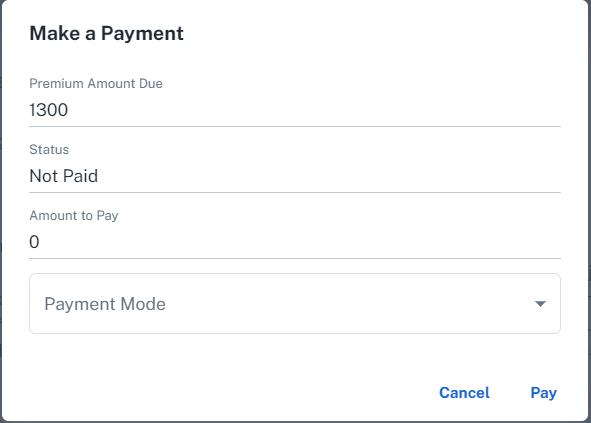
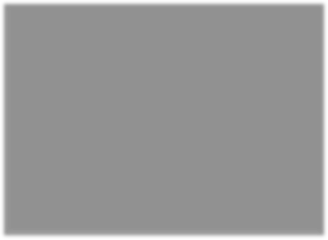
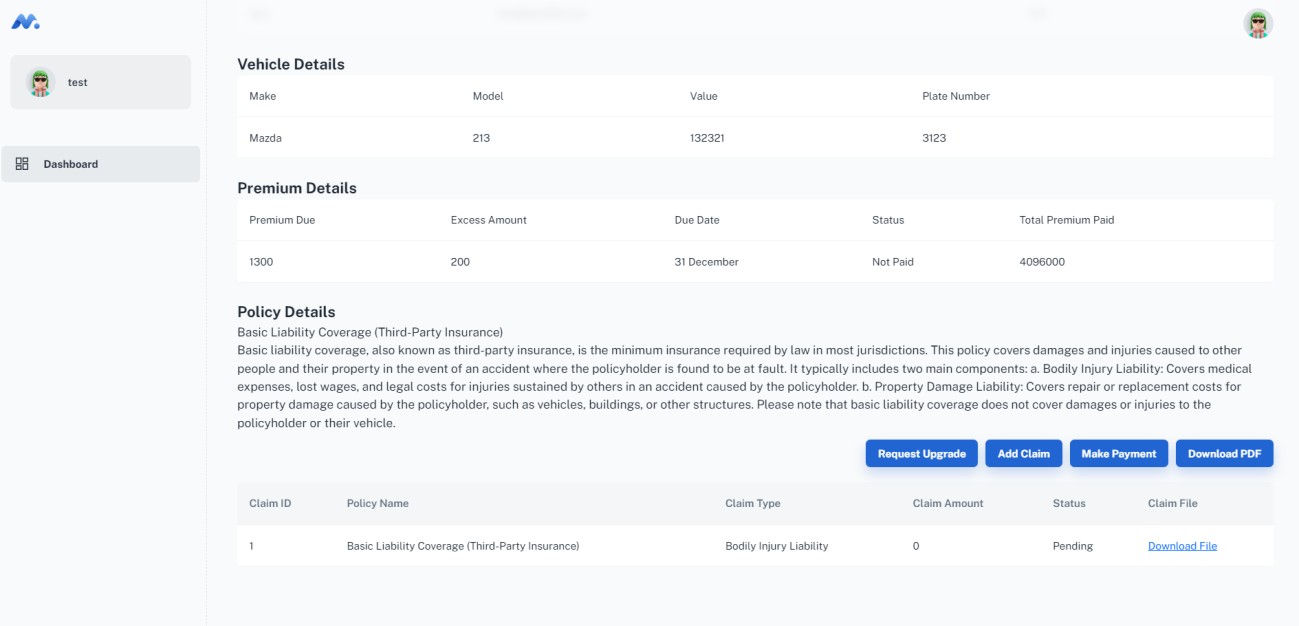
* 1. User initiates the payment process by submitting a payment request. This request should include the necessary information such as customer ID, policy number, payment details etc.
  2. The system validates the request. If the request is valid, it proceeds to the next step. If not, it sends an error message to the user.
  3. The system sends the payment details to the integrated payment gateway for processing.
  4. The payment gateway processes the payment and sends a response back to the system. This response contains the payment status and transaction details.
  5. Based on the payment status received from the gateway, the system updates the premium and/or claim status accordingly. If the payment is successful, the status is updated to 'paid'. If not, the status is updated to 'payment failed'.
  6. The system sends a response back to the user with the payment status and updated policy or claim status.

#### Refund Payment

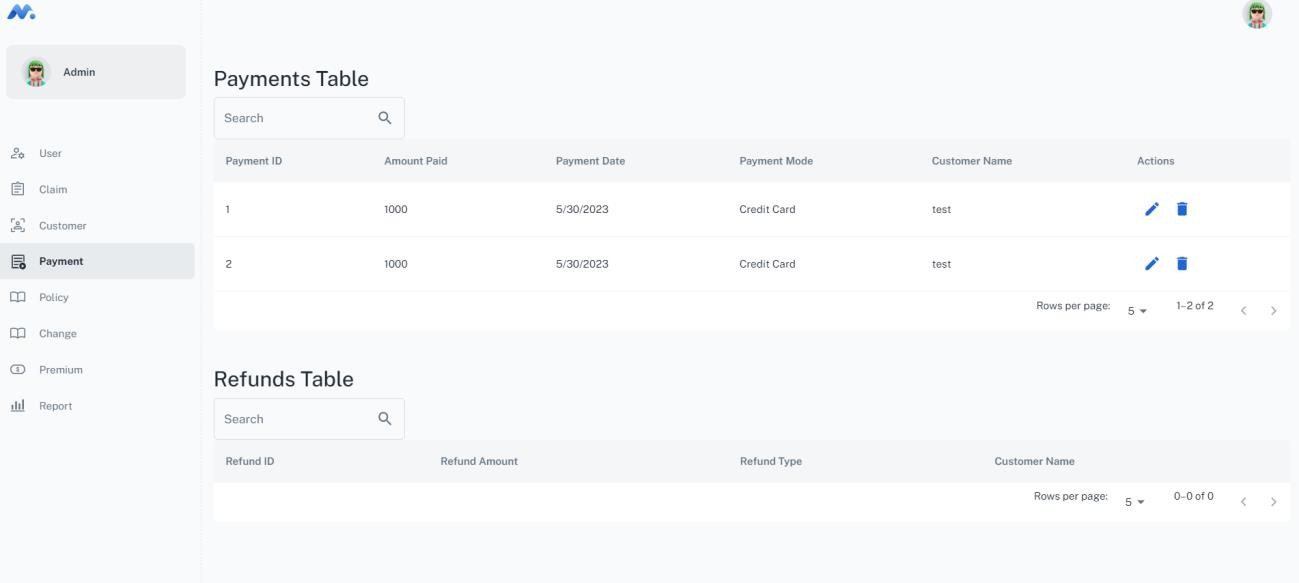
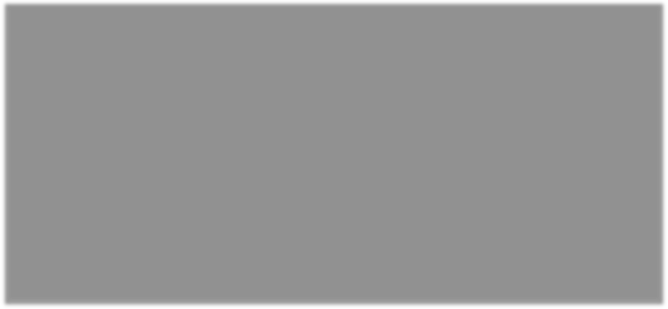
* 1. A request for refund is initiated by the user or system due to policy cancellations or claim rejections. This request includes necessary details like customer ID, policy number, and reason for refund.
  2. The system validates the refund request. If it's valid, the system proceeds to the next step. If not, an error message is sent to the user.
  3. The system initiates a refund transaction through the integrated payment gateway, sending along the necessary payment details.
  4. The payment gateway processes the refund and sends back a response to the system with the refund status and transaction details.
  5. Based on the refund status received from the gateway, the system updates the policy or claim status accordingly. If the refund is successful, the status is updated to 'refunded'. If not, the status is updated to 'refund failed'.
  6. The system sends a response back to the user with the refund status and updated policy or claim status.

### User Interface

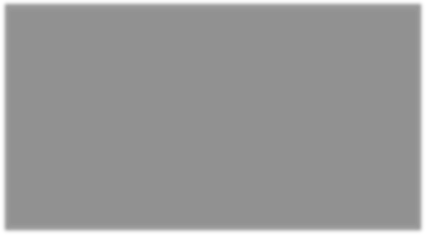
#### Customer Portal



#### Admin Portal



#### Refunds

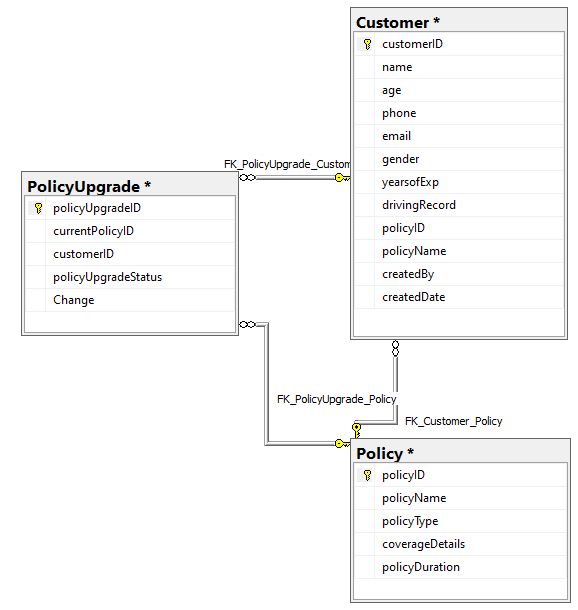
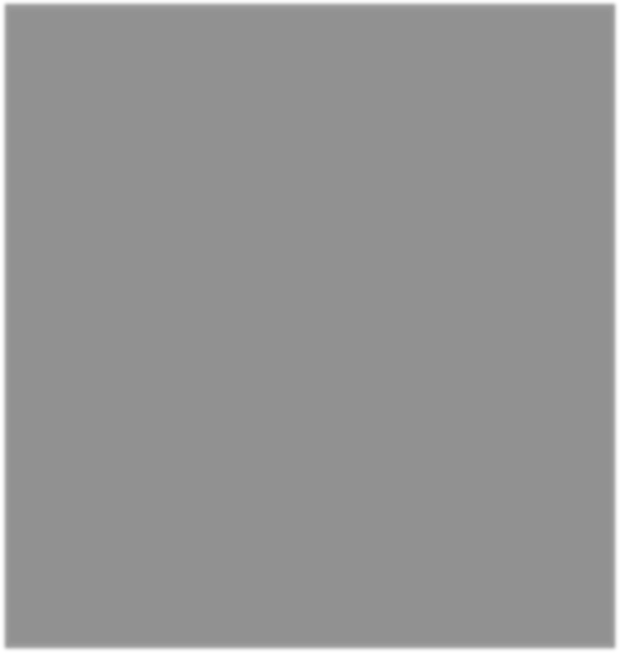


## Customer Requests Upgrade

### Function Requirement

* + 1. The system should provide a "Request Upgrade" option in the browser interface for customers to initiate an upgrade of their existing policies.
    2. The system should determine the available upgrade options based on the insurer's policies and present them to the customer for selection.
    3. In Scenario 1, the system should allow customers to edit their existing policies by upgrading from a Standard (Basic) cover plan to an Executive (Advanced) cover plan. This upgrade should include any additional features or benefits associated with the Executive cover plan.
    4. In Scenario 2, the system should enable customers to edit their existing policies by increasing or decreasing the sum insured and excess amount. These changes should be reflected in the premiums charged for the policy.
    5. In Scenario 3, the system should support the creation of multiple policies for a customer, with a group discount applied to the premiums. The system should calculate the discounted premiums based on the number of policies associated with the customer's account.
    6. The system should validate the customer's eligibility for each upgrade option based on the insurer's policies and business rules.
    7. The system should ensure that all upgrades and changes to policies are accurately recorded and reflected in the customer's account and relevant policy documents.
    8. The system should generate notifications or alerts to inform the insurer's staff about customer requests for policy upgrades, enabling them to review and process the requests in a timely manner.
    9. The system should provide appropriate error handling and validation to prevent invalid or conflicting upgrade requests.
    10. The system should maintain an audit trail of all upgrade requests, including the details of the original policy, requested changes, approval status, and any additional information related to the upgrade process.
    11. The system should generate updated policy documents reflecting the changes made through the upgrade process and provide them to the customer.
    12. The system should ensure the security and privacy of customer data throughout the upgrade process, adhering to relevant data protection regulations.
    13. The system should provide a user-friendly interface that guides customers through the upgrade process and provides clear instructions and feedback at each step.
    14. The system should be scalable and capable of handling a high volume of upgrade requests simultaneously to ensure efficient processing and minimal downtime.
    15. The system should integrate with relevant internal systems and databases to retrieve and update policy information accurately and in real-time during the upgrade process.

### Entity Relationship Diagram



### API End Points

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API**  **Function** | **Method** | **API Call** | **Input Parameters** | **Output** | **Error Handling** |
| Get all policy upgrades | GET | http://localhost:5179/api/PolicyUpgrade | None | A list of policy upgrade objects | If no policy upgrades are found, return an empty list |
| Create a new policy upgrade | POST | http://localhost:5179/api/PolicyUpgrade | Policy upgrade object (JSON  format) | The created policy upgrade object with a new ID | If input data is invalid or incomplete, return a 400 Bad Request status with an error message |
| Get a specific policy upgrade by ID | GET | http://localhost:5179/api/PolicyUpgrade/{id} | Policy upgrade ID (integer) | The policy upgrade object with the specified ID | If the policy upgrade ID is not found, return a 404 Not Found status with an error  message |
| Update a policy upgrade by ID | PUT | http://localhost:5179/api/PolicyUpgrade/{id} | Policy upgrade ID (integer), updated policy upgrade object (JSON  format) | The updated policy upgrade object | If the policy upgrade ID is not found, return a 404 Not Found status with an error message. If input data is invalid or incomplete, return a 400 Bad Request status with an error message |
| Delete a policy upgrade by ID | DELETE | http://localhost:5179/api/PolicyUpgrade/{id} | Policy upgrade ID (integer) | A  confirmation message indicating successful deletion | If the policy upgrade ID is not found, return a 404 Not Found status with an error  message |

### Work Flow

#### Request Upgrade:

* 1. On the customer's policy page, there should be a "Request Upgrade" button. This will direct the customer to an Upgrade Request form.
  2. The form should show the details of the customer's current policy and the possible upgrade options. These options will depend on the existing policy's coverage and limits.

#### Upgrade Options:

The form should allow for the following scenarios:

#### Scenario 1 - Upgrade Cover Plan:

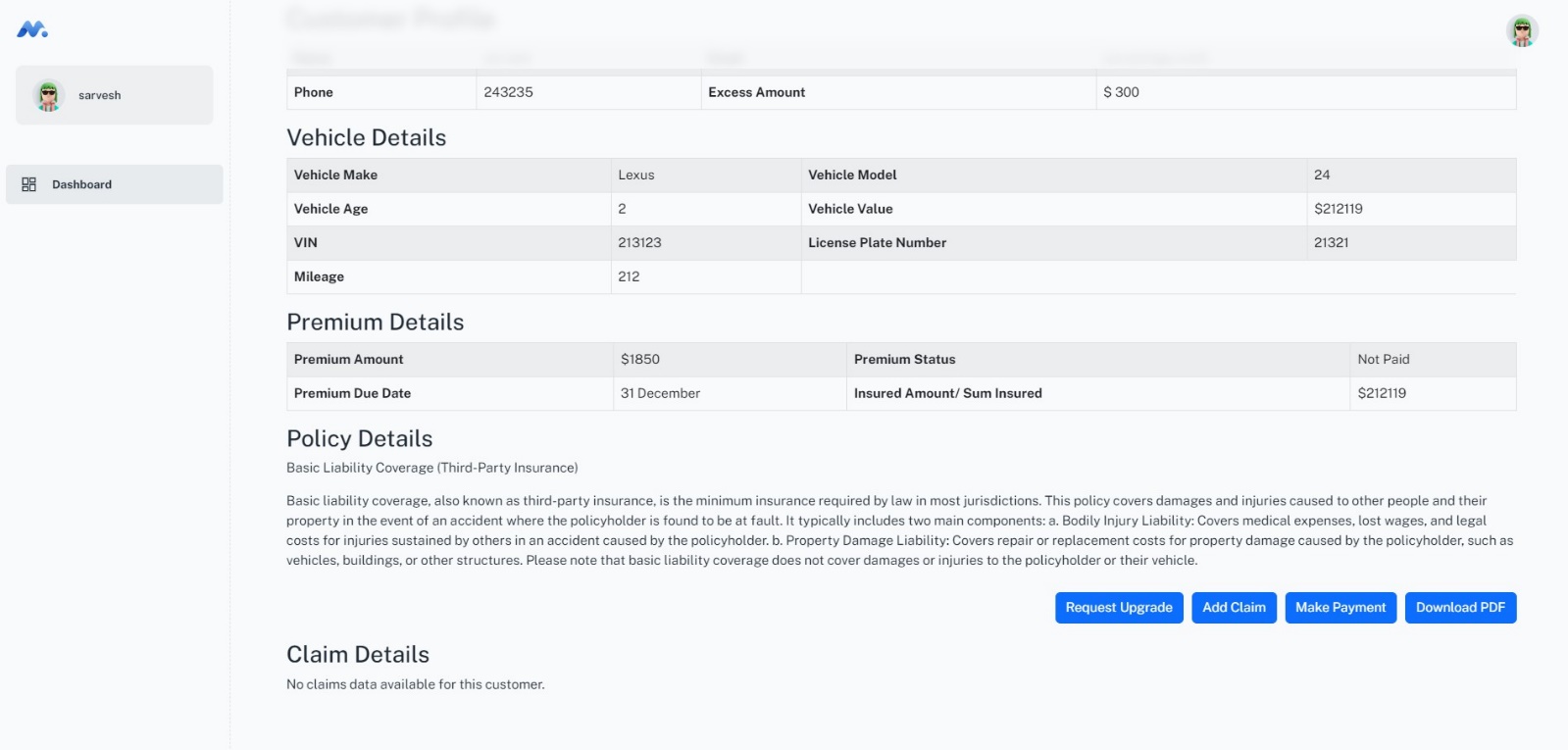
* 1. If the current policy is a Standard (Basic) cover plan, the form should include an option to upgrade to an Executive (Advanced) cover plan.
  2. On selection, the system should calculate the new premium for the

Executive cover plan and display it to the customer.

* 1. If the customer accepts the new premium, they can submit the form to request the upgrade.

#### Scenario 2 - Adjust Sum Insured and Excess Amount:

* 1. The form should include fields to adjust the sum insured and excess amount, along with an explanation of how these changes would affect the premium.
  2. As the customer adjusts these values, the system should dynamically calculate and display the updated premium.
  3. If the customer is satisfied with the changes, they can submit the form to request the upgrade.



#### Scenario 3 - Multiple Policies with Group Discount:

* 1. If the customer has multiple policies, the form should include an option to apply a group discount to the premiums.
  2. On selection, the system should calculate the total premium with the group discount and display it to the customer.
  3. If the customer accepts the new total premium, they can submit the form to request the upgrade.

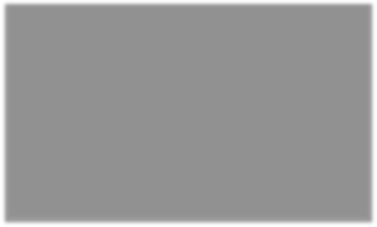
#### Submission and Processing:

* 1. Once the customer submits the form, the system should send the upgrade request to the insurer for approval.
  2. The system should then update the customer's policy details as per the upgrade request and recalculate the premium accordingly.
  3. The customer should be notified of the status of their upgrade request and any changes to their policy and premium.

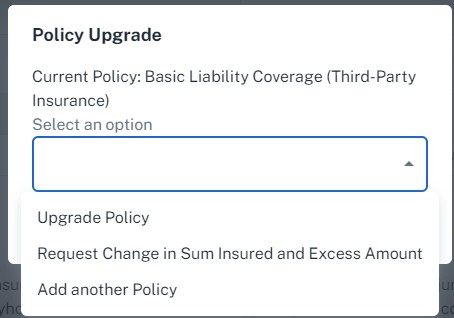
### User Interface

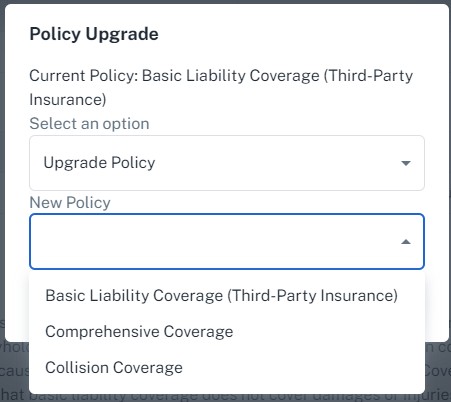
Customer View

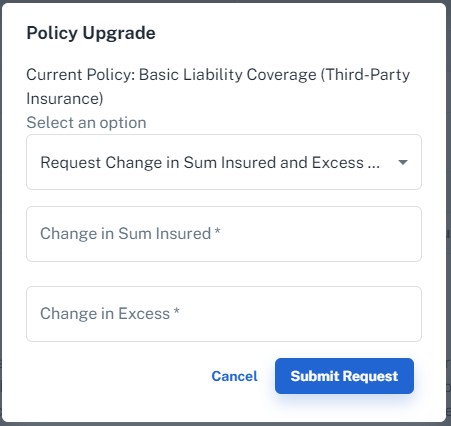
To Request Upgrade

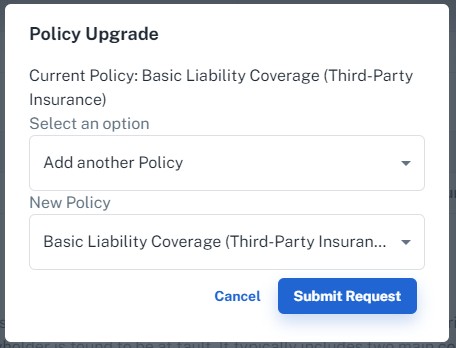


Select Scenario

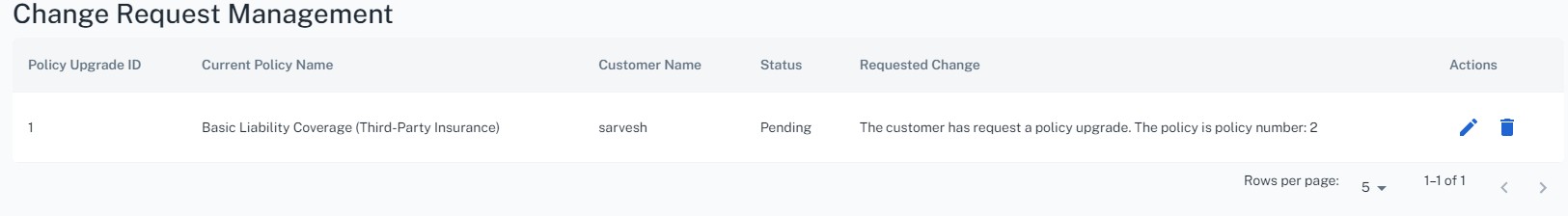
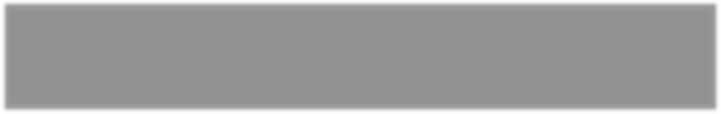


Scenario 1

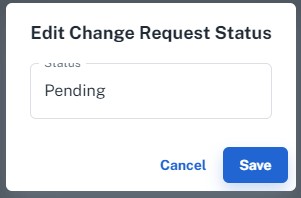
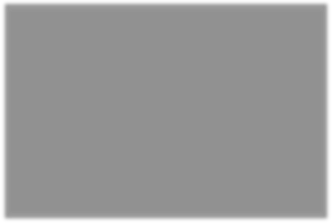
Scenario 2

Scenario 3

Insurer View



After the change has been implemented



## Claim Management

### Function Requirement

1. Claim Creation:

The system should provide a feature that allows insurers to create a new claim on behalf of a policyholder. This should involve the input of claim details such as the policy number, claim amount, claim type, and claim date.

Upon successful claim creation, the system should assign a unique claim number to the new claim and update the claim status to "Pending". The policyholder should also be notified of the claim's creation, and the claim details should be made available for them to view from their dashboard.

1. Claim Viewing:

The system should offer a search feature that allows insurers to locate and view a claim's details using a claim number, policy number, or customer name.

The claim's details, including its current status and any associated documents or evidence, should be presented in a clear and organized manner. The claim's history, including any status changes and notes added by claims staff, should also be viewable.

1. Claim Updating:

Insurers should be able to update a claim's details, including its status. This could involve approving or rejecting the claim based on the provided evidence and the terms of the policyholder's insurance policy.

Whenever a claim is updated, the system should automatically notify the policyholder of the changes. If a claim is approved, the system might also initiate the payment process, depending on the insurer's policies and procedures.

1. Claim Deletion:

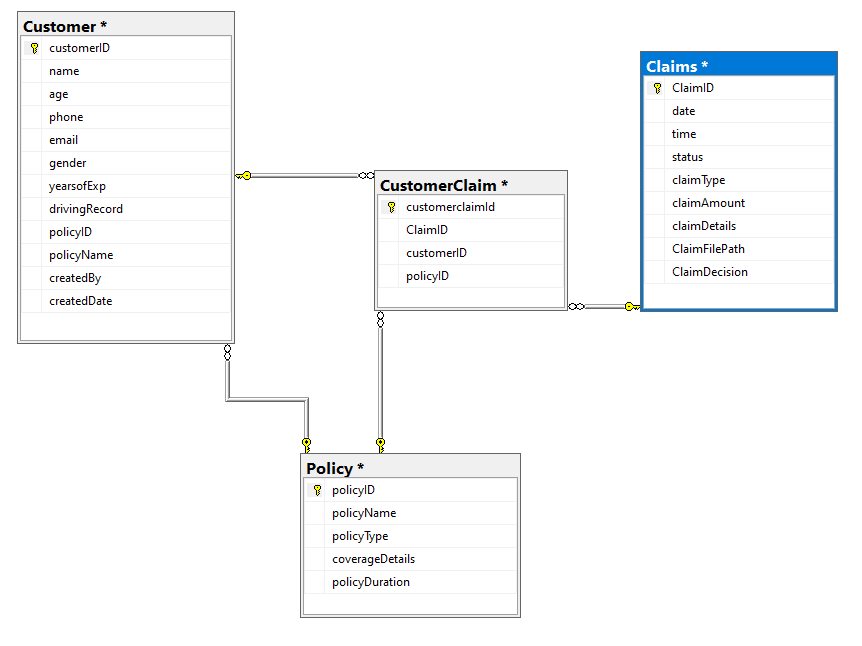
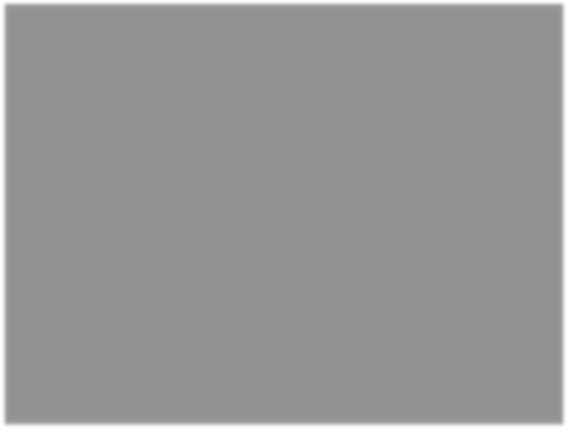
In situations where a claim is found to be invalid or is created in error, insurers should have the ability to delete the claim from the system. This should involve providing the claim number and possibly a reason for the deletion.

The system should ensure that all associated data is removed and that the policyholder is notified of the claim's deletion. However, for audit purposes, the system might keep a record of the deletion, including the claim number, the deletion date, and the reason for deletion.

1. Claim History and Auditing:

The system should keep a detailed record of all claim activities, including claim creation, updates, and deletion. This claim history should be viewable by insurers and should be designed in a way that it can be used for auditing purposes if needed.

### Entity Relationship Diagram



### API End Points

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| API  Function | Method | API Call | Input Parameters | Output | Error Handling |
| Get all claims | GET | http://localhost:5179/api/Claims | None | A list of claim objects | If no claims are found,  return an empty list |
| Create a new claim | POST | http://localhost:5179/api/Claims | Claim object (JSON  format) | The created claim object with a new ID | If input data is invalid or incomplete, return a 400 Bad Request status with an error  message |
| Get a specific claim by ID | GET | http://localhost:5179/api/Claims/{id} | Claim ID (integer) | The claim object with the specified ID | If the claim ID is not found, return a 404 Not Found status with  an error message |
| Update a claim by ID | PUT | http://localhost:5179/api/Claims/{id} | Claim ID (integer), updated claim object (JSON  format) | The updated claim object | If the claim ID is not found, return a 404 Not Found status with an error message. If input data is invalid or incomplete, return a 400 Bad Request status with an error  message |
| Delete a claim by ID | DELETE | http://localhost:5179/api/Claims/{id} | Claim ID (integer) | A  confirmation message indicating successful deletion | If the claim ID is not found, return a 404 Not Found status with an error  message |

### Work Flow

* 1. Claim Creation:

An insurer logs into the Insurance Management System (IMS). The insurer navigates to the "Create Claim" section.

The insurer inputs the claim details (policy number, claim amount, claim type, claim date) into the designated fields.

The system validates the provided details and checks the policy number against the database.

Upon validation, the system generates a unique claim number, assigns it to the new claim, and sets the claim status as "Pending".

The insurer submits the claim, and the system saves it into the database.

The system sends a notification to the policyholder regarding the creation of the claim and updates the claim details on the policyholder's dashboard.

* 1. Claim Viewing:

The insurer navigates to the "View Claim" section.

The insurer inputs the claim number, policy number, or customer name into the search field. The system retrieves the corresponding claim details from the database.

The system displays the claim details, including status and any associated documents or evidence, in a user- friendly format. The claim history, including any status changes and notes, are also displayed.

* 1. Claim Updating:

The insurer navigates to the "Update Claim" section.

The insurer searches for the claim using the claim number, policy number, or customer name.

The system displays the claim details. The insurer can edit the claim details and change the claim status (approve/reject).

The insurer submits the updated claim details, and the system saves the changes into the database.

The system sends a notification to the policyholder about the changes made to their claim. If approved, the system initiates the payment process.

* 1. Claim Deletion:

The insurer navigates to the "Delete Claim" section.

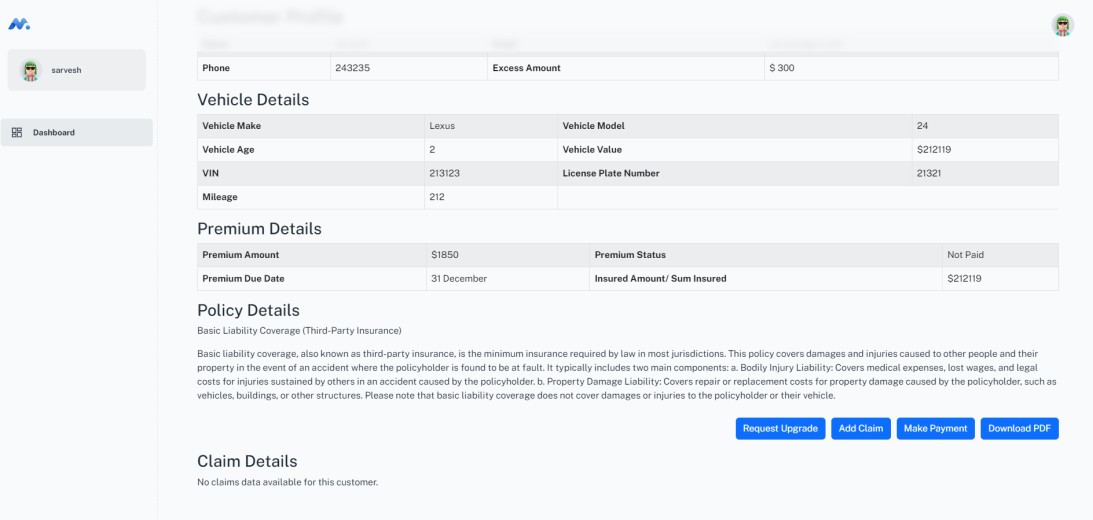
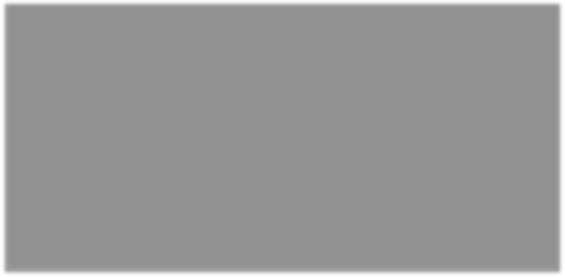
The insurer inputs the claim number and provides a reason for deletion. The system validates the claim number and deletion reason.

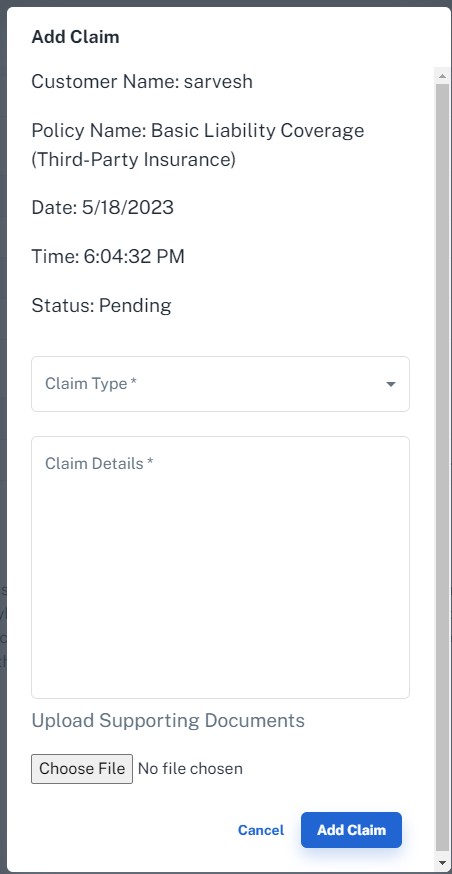
Upon validation, the system removes the claim from the active claims database and sends a notification to the policyholder about the claim deletion.

The system keeps a record of the deletion, including the claim number, deletion date, and reason for deletion, for auditing purposes.

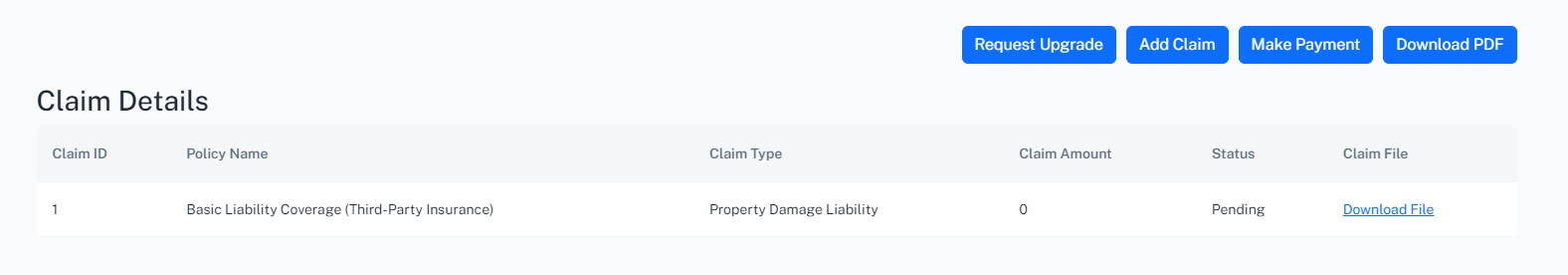
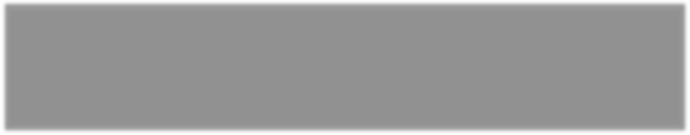
### User Interface

Customer View

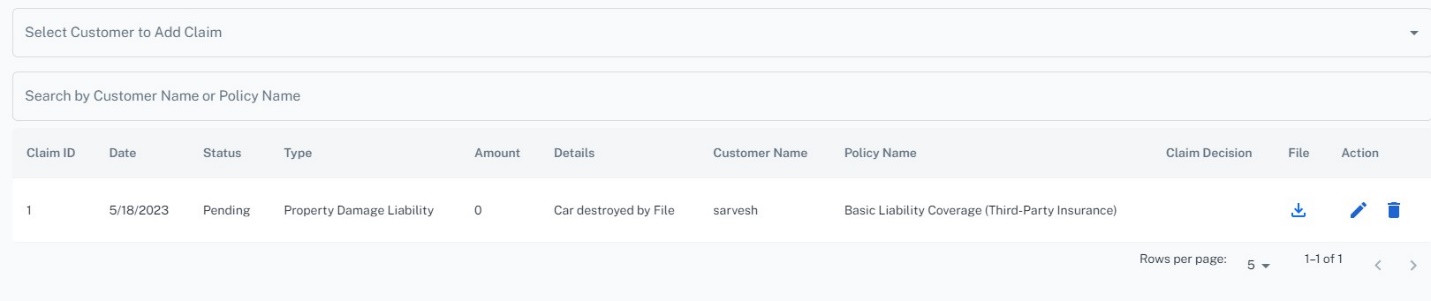
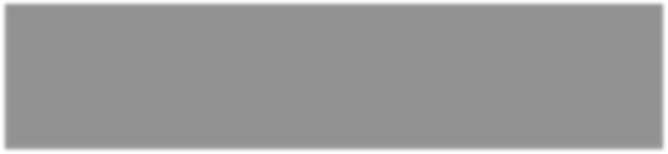


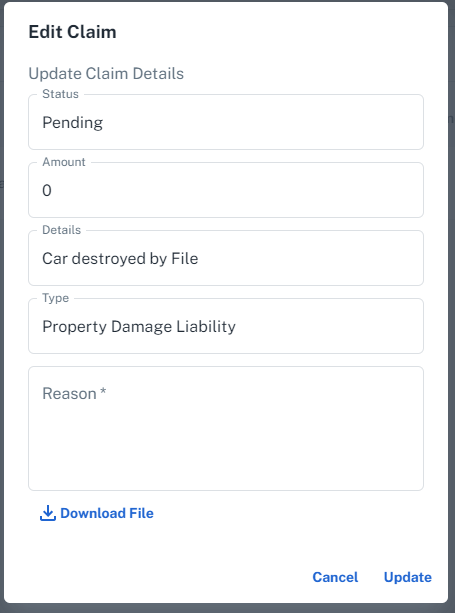
Add Claim

After Claim is filed

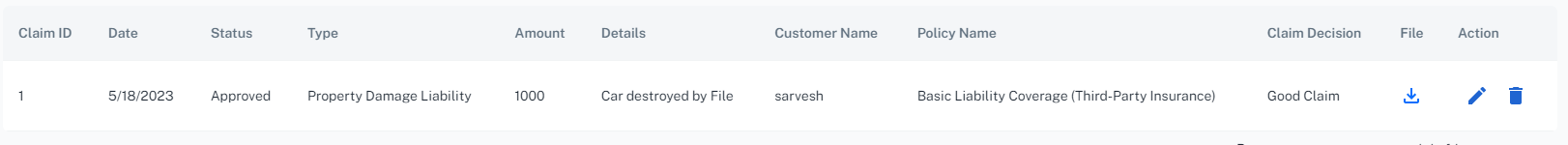
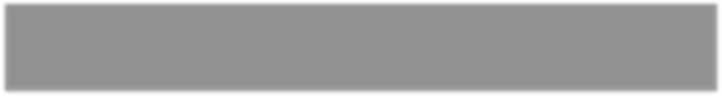
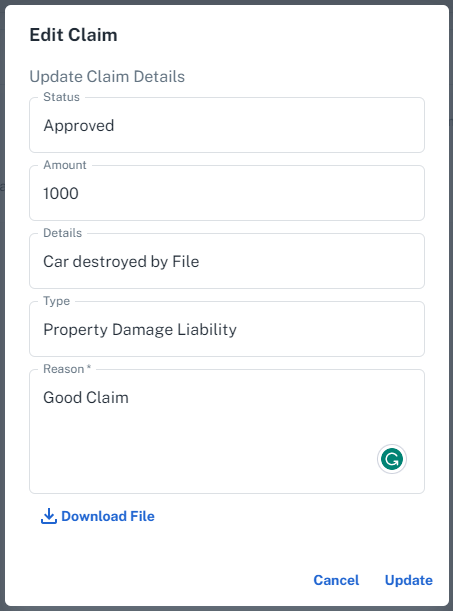
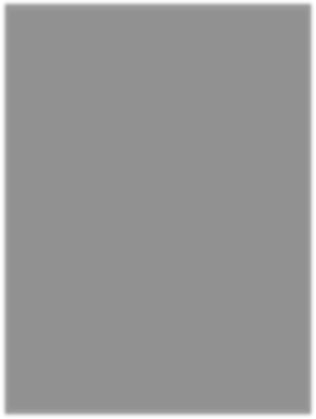


Insurer View

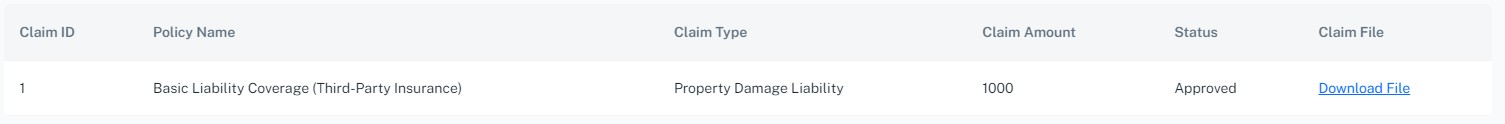
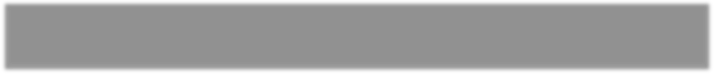


To View Claim Insurer will click on edit claim.

Insurer can update claim status.



The customer is also able to view the updated status.



## Cancellation and Renewal

### Function Requirement

Policy Renewal:

* 1. The system should be able to identify policies due for renewal based on policy end dates.
  2. The system should provide a functionality for the Insurer to renew policies for customers.
  3. Upon renewal, the system should update the policy status, renewal date, and any other relevant details in the customer reports and database.
  4. The system should send a confirmation notification to the customer and the insurer stating the successful renewal of the policy.

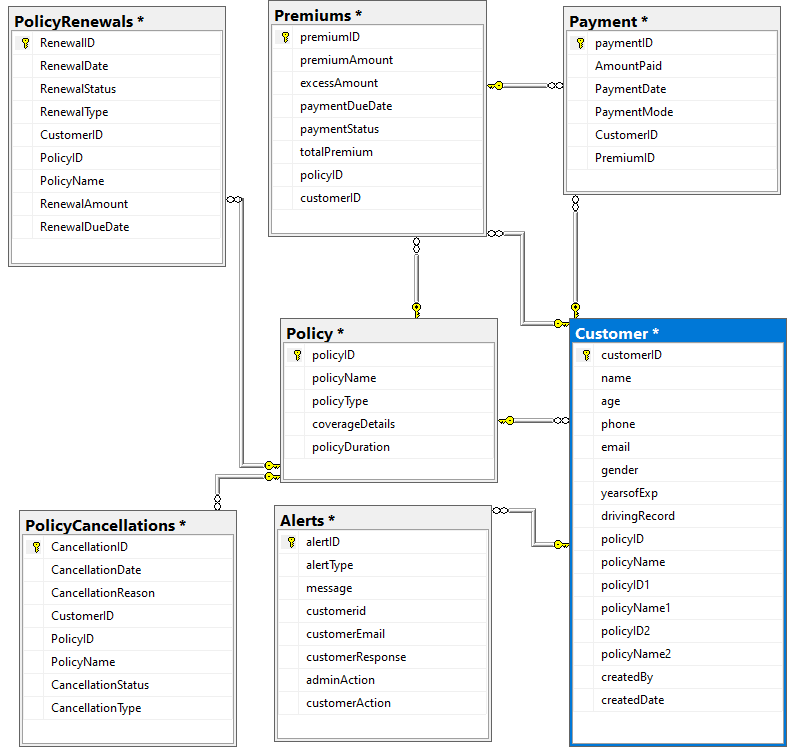
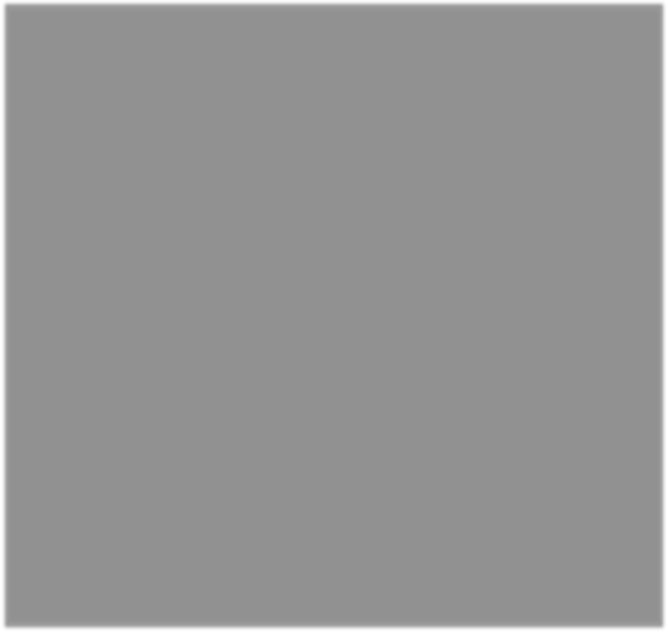
Policy Cancellation Due to Non-Payment of Premiums:

* 1. The system should be able to identify if a customer hasn't paid the premiums for an existing policy.
  2. If the premium for a policy is not paid by the due date, the system should change the status of the policy to 'Cancelled due to non-payment'.
  3. The system should update this cancellation status in the customer reports and database.
  4. The system should automatically generate and send alerts to the customer regarding the policy cancellation. These alerts can be sent through email, SMS, or other communication channels integrated with the system.

Policy Cancellation Upon Customer Request:

* 1. The system should allow customers to request policy cancellations even if premiums have been paid for the current term.
  2. Upon receiving the cancellation request, the system should update the policy status to 'Cancelled upon request' in the customer reports and database.
  3. The system should send a confirmation notification to the customer and the insurer indicating the cancellation of the policy.

### Entity Relationship Diagram



### API End Points

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API**  **Function** | **Method** | **API Call** | **Input**  **Parameters** | **Output** | **Error Handling** |
| Get all cancellations | GET | api/PolicyCancellation | None | A list of  cancellation objects | If no cancellations are found, return an empty list |
| Create a new cancellation | POST | api/PolicyCancellation | Cancellation object (JSON  format) | The created cancellation object with a  new ID | If input data is invalid or incomplete, return a 400 Bad Request status with an  error message |
| Get a specific cancellation by ID | GET | api/PolicyCancellation/{id} | Cancellation ID (integer) | The cancellation object with the specified  ID | If the cancellation ID is not found, return a 404 Not Found status with an error message |
| Update a cancellation by ID | PUT | api/PolicyCancellation/{id} | Cancellation ID (integer), updated cancellation object (JSON  format) | The updated cancellation object | If the cancellation ID is not found, return a 404 Not Found status with an error message. If input data is invalid or incomplete, return a 400 Bad Request status with an  error message |
| Delete a cancellation by ID | DELETE | api/PolicyCancellation/{id} | Cancellation ID (integer) | A  confirmation message indicating successful  deletion | If the cancellation ID is not found, return a 404 Not Found status with an error message |

#### Policy Renewal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **API**  **Function** | **Method** | **API Call** | **Input**  **Parameters** | **Output** | **Error Handling** |
| Get all renewals | GET | api/PolicyRenewal | None | A list of renewal objects | If no renewals are found, return an empty list |
| Create a new renewal | POST | api/PolicyRenewal | Renewal object (JSON format) | The created renewal object with a new ID | If input data is invalid or incomplete, return a 400 Bad Request status with an error message |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Get a specific renewal by ID | GET | api/PolicyRenewal/{id} | Renewal ID (integer) | The renewal object with the specified ID | If the renewal ID is not found, return a 404 Not Found status with an error message |
| Update a renewal by ID | PUT | api/PolicyRenewal/{id} | Renewal ID (integer), updated renewal object (JSON format) | The updated renewal object | If the renewal ID is not found, return a 404 Not Found status with an error message. If input data is invalid or incomplete, return a 400 Bad Request status with an error message |
| Delete a renewal by ID | DELETE | api/PolicyRenewal/{id} | Renewal ID (integer) | A confirmation message indicating successful deletion | If the renewal ID is not found, return a 404 Not Found status with an error message |

### Work Flow

#### Policy Renewal:

* 1. The insurer initiates the renewal process for a specific customer's policy.
  2. The system identifies the policy details and calculates the new premium for the renewal period.
  3. The insurer confirms the renewal and the system updates the policy details including the status, renewal date, and premium.
  4. The system generates a customer report reflecting the updated policy details.
  5. The system sends a notification to the customer informing them about the policy renewal.

#### Policy Cancellation due to Non-Payment of Premiums:

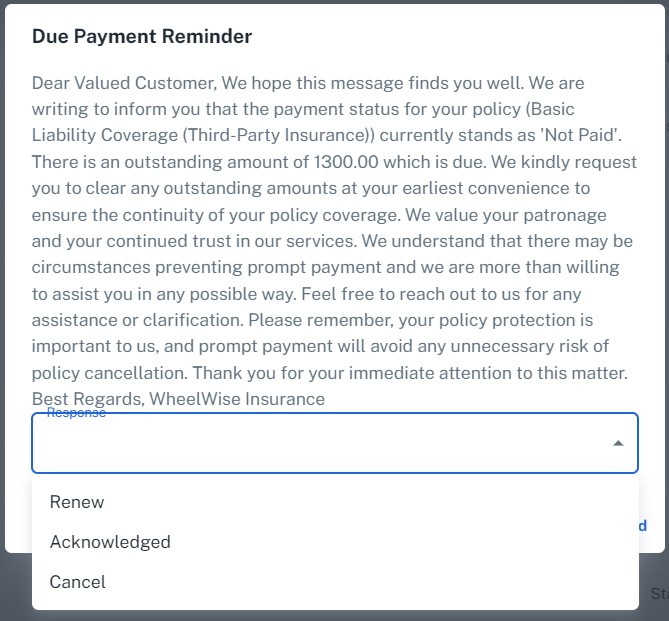
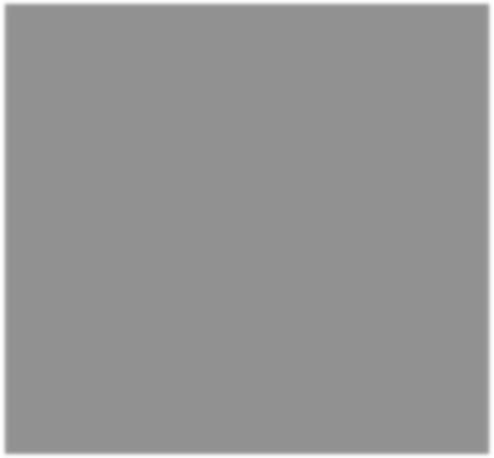
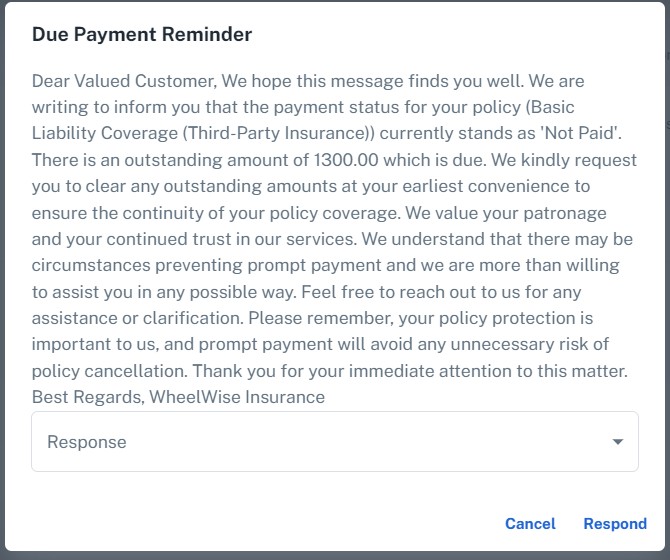
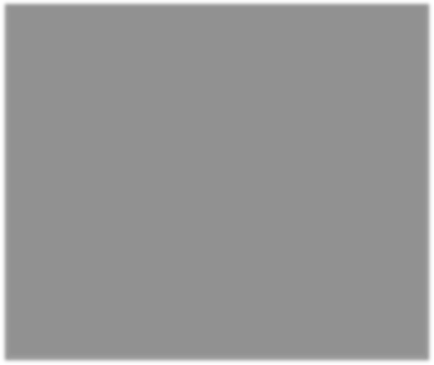
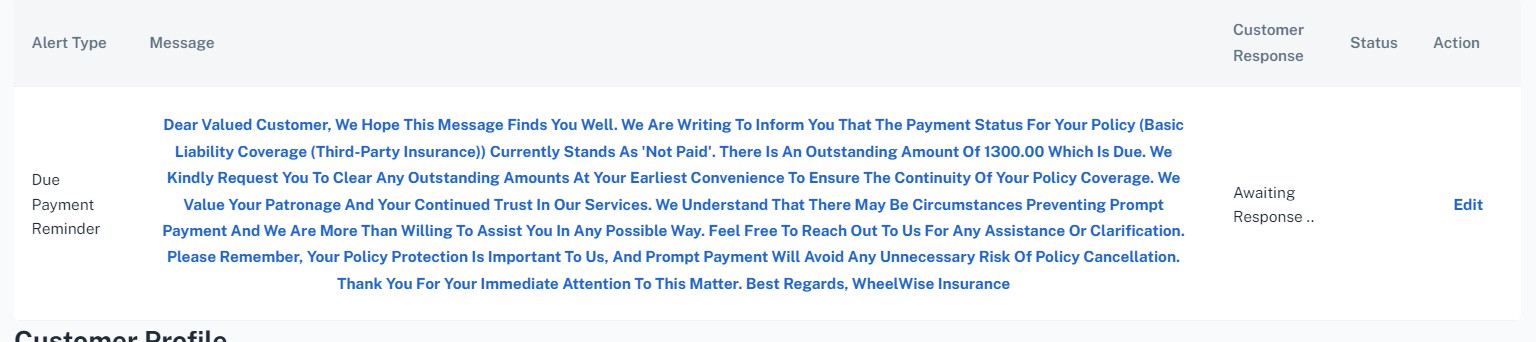
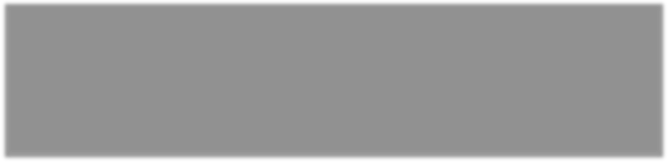
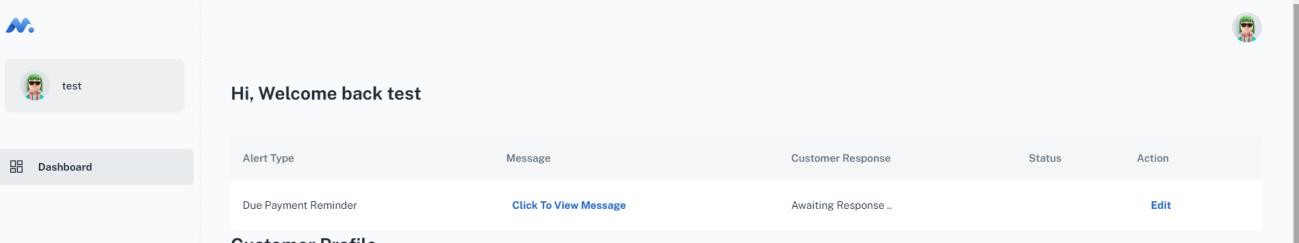
* 1. The system identifies policies with unpaid premiums after a specified due date.
  2. For each identified policy, the system changes the policy status to "Cancelled" and updates this information in the customer reports.
  3. The system generates and sends an alert to the respective customers informing them about the policy cancellation due to non-payment of premiums.

#### Policy Cancellation Upon Customer Request:

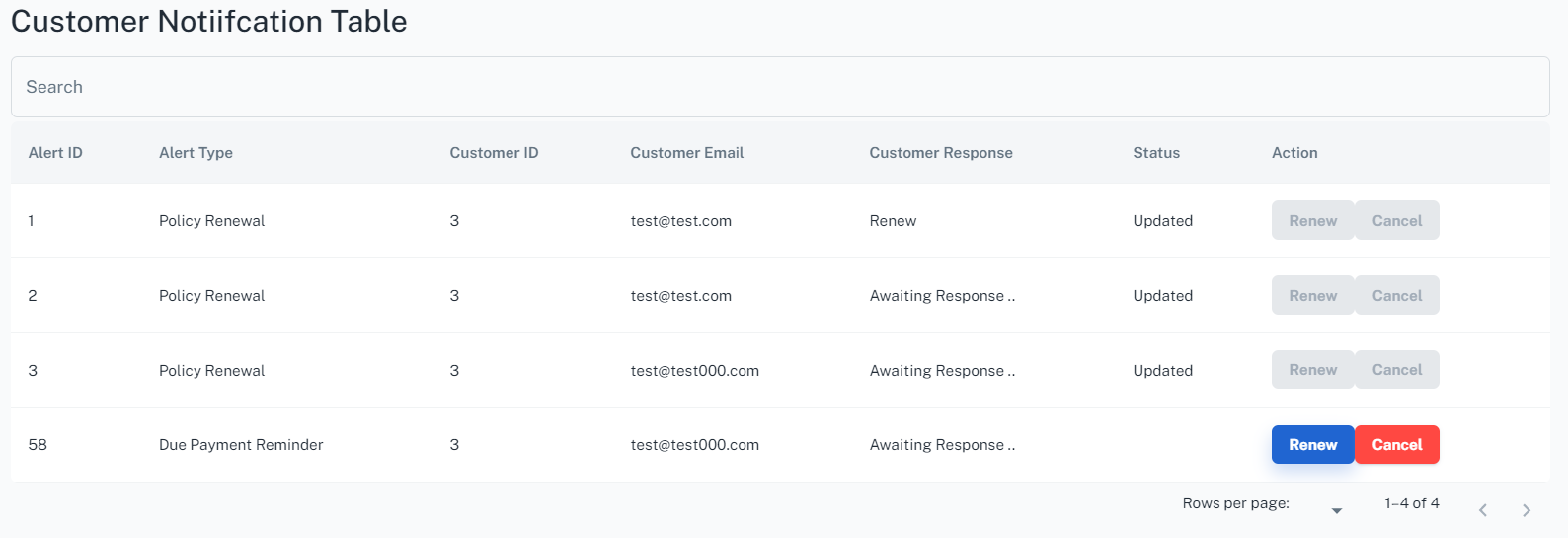
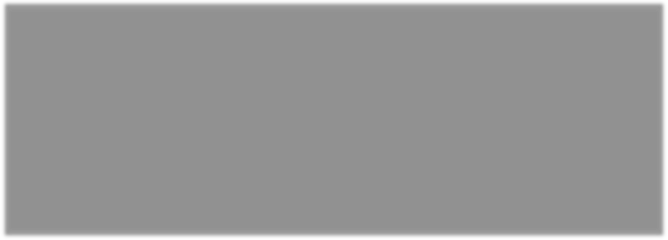
* 1. The customer submits a request for policy cancellation through a defined channel (could be an online form, email, etc.).
  2. The system verifies the customer request and identifies the policy to be cancelled.
  3. Upon confirmation, the system updates the policy status to "Cancelled", updates the customer report with this change, and processes any necessary refunds.
  4. The system sends a notification to the customer informing them about the policy cancellation and any subsequent actions (like refunds).

### User Interface

#### Customer Portal



#### Admin Portal



# System Functionalities

## Premium Calculation

The `calculatePremium` function helps determines the premium amount for an insurance policy based on a number of factors. Here's how it works:

* + 1. Base Premium: The base premium is initially set at 200.
    2. Age Factor: If the age of the customer is less than 25, an additional 300 is added to the premium. If the customer is 25 or older, an additional 200 is added.
    3. Experience Factor: If the customer has less than 5 years of driving experience, the premium is increased by 250.
    4. Driving Record Factor: Depending on the driving record of the customer, an additional amount is added to the premium. If the record is "fine", 400 is added. If the record shows an "accident", 200 is added. If the record is neither "fine" nor "accident", no amount is added.
    5. Vehicle Value Factor: If the value of the vehicle is less than 30,000, an additional 200 is added to the premium. If the vehicle's value is 30,000 or more, an additional 400 is added.
    6. Mileage Factor: If the mileage of the vehicle is less than 50,000, an additional 200 is added to the premium. If the mileage is 50,000 or more, an additional 400 is added.
    7. Vehicle Make Factor: If the vehicle make is one of the specified luxury brands ("BMW", "Mercedes-Benz", "Audi", "Lexus", "Porsche", "Tesla"), an additional 500 is added to the premium.

After all the factors are considered and the premium is calculated, the function uses the setPremiumAmount function to store the premium amount, and the setexcessAmount function to set the excess amount (which is 20% of the calculated premium).

## Claim Management Process

#### Load a Claim on Behalf of the Customer:

The Insurance Management System (IMS) should be equipped with a user-friendly interface for claims staff. This interface includes options to input necessary information regarding the claim. It also include an option to upload files, functioning as proof of loss, through a file picker mechanism.

When a claim is loaded, the system should automatically assign a status of "Pending" to the claim and save all information related to the claim in the database. The system also ensures that the uploaded files are securely stored and easily accessible for review.

#### Update the Existing Claim Status:

The system allows claims staff to update the status of a claim, with options including "Pending", "Approved", or "Rejected". Upon updating the status, the system should automatically save the change in the database.

A claim is approved or rejected, by the Claims Manager, a Insurer with that job role.

#### Customer Dashboard:

Customers has the ability to create claims directly from their dashboard. The dashboard includes a user- friendly interface where customers can input necessary information and upload files as proof of loss.

When a customer submits a claim, the system automatically assigns a status of "Pending" to the claim, save all information related to the claim in the database.

The system also ensures customers are kept informed about the status of their claim.

## Handling Customer Request

The system functionality for handling customer request can be described as follows:

#### Upgrade Request:

The system provides an option for customers to request an upgrade of their existing policies. This could be achieved through a button or a link labelled "Request Upgrade" on the customer's dashboard or in the policy details section.

#### Upgrade Options:

After selecting the "Request Upgrade" option, customers are presented with different upgrade options. These options will depend on the specific policy the customer currently holds and the available upgrade paths.

#### Scenario 1 - Cover Plan Upgrade:

Customers is be able to upgrade from a Basic cover plan to an Advanced cover plan. The system should update the policy details to reflect the new cover plan and adjust the premium according to the rates defined for the Advanced cover plan.

#### Scenario 2 - Adjust Sum Insured and Excess Amount:

Customers has the option to increase or decrease the sum insured and the excess amount. Upon making these changes, the Insurer will recalculate the premium, reflecting the updated sum insured and excess amount.

#### Scenario 3 - Multiple Policies with Group Discount:

The system allows customers to adopt multiple policies and get a group discount on premiums. When a customer adds a new policy, the system will check if the customer is eligible for a group discount (based on the number of policies or specific policy combinations) and, if so, automatically apply the discount to the premium calculations.

#### Policy Update:

After the customer selects their desired upgrade option(s) and confirms their choice, the system will send a request to update the policy details in the database. The customer will then receive a notification confirming the successful upgrade of their policy.

#### Insurer Approval:

Upgrades require approval from the insurer.

Renewals Process

#### Initiate Renewal

The system should allow insurers to initiate the renewal process for any customer's policy that is due for renewal. This could be through a user interface or automated scripts that identify policies nearing their expiry date.

#### Calculate Premium

The system should use the customer's policy details and any applicable renewal rules to calculate the premium for the renewal period. This could involve a risk assessment module, historical data, or actuarial algorithms, depending on the insurer's business rules.

#### Update Policy

Once the insurer confirms the renewal, the system should update the policy's status, renewal date, and premium. This would typically involve updating the policy record in the database and any in- memory data structures or caches.

#### Notify Customer

The system should inform the customer about the renewal. This could be through email, SMS, push notification, or any other communication channel supported by the system.

Cancellation Process

#### Identify Policies for Cancellation

The system should identify policies that are candidates for cancellation. This could be due to non- payment of premiums or customer requests. The identification process could involve database queries or automated scripts that run at specified intervals.

#### Update Policy Status

For each policy that is to be cancelled, the system should update the policy status to "Cancelled". This involves updating the policy record in the database and any in-memory data structures or caches.

#### Notify Customer

The system should alert the customer about the policy cancellation. This could be through an email, SMS, push notification, or any other communication channel supported by the system. The notification should clearly communicate the reason for cancellation and any subsequent actions, such as refunds if applicable.

# 8.Software Metrics

The following software metric is used to evaluate the response time for each feature main component that was recorded during the testing phase.

|  |  |
| --- | --- |
| **Feature: Main Function** | **Response Time** |
| **Policy Management:**  Policy Loading Time: Time taken to fetch and display policy details from the database. | 0.1 seconds |
| **Premium Management:**  Premium Calculation Time: Time taken to calculate the premium for a specific policy. | 0.2 seconds |
| **Customer Management:**  Customer Profile Loading Time: Time taken to fetch and display a customer's profile or information. | 0.3 seconds |
| **Reporting:**  Report Generation Time: Time taken to generate and display a report. | 1 seconds |
| **Payment Management:**  Payment Processing Time: Time taken from initiating a payment transaction to the confirmation of the payment. | 0.5 seconds |
| **Customer Requests Upgrade**:  Upgrade Request Processing Time: Time taken from the customer's upgrade request submission to the confirmation of the upgrade. | 0.5 seconds |
| **Claim Management:**  Claim Loading Time: Time taken to fetch and display claim details from the database. | 0.5 seconds |
| **Cancellation and Renewal**:  Cancellation/Renewal Processing Time: Time taken from the initiation of a cancellation/renewal to its confirmation. | 1. seconds |

# 9.Security

The Wheel Wise Insurance Management System implements various security measures to ensure the confidentiality, integrity, and availability of data within the system. These measures include data encryption, authentication, authorization, and adherence to specific compliance requirements.

## Data Encryption

Data encryption is employed to protect sensitive information, such as customer personal details, policy data, and payment information, both at rest and in transit.

For data in transit, the system uses HTTPS with SSL/TLS encryption to secure communication between the client and the server.

For data at rest, encryption techniques are used to secure sensitive information stored in the database, such as passwords and payment details.

## Authentication

The system requires users to provide valid login credentials, such as a username and password, to access the application.

Passwords are stored securely using a combination of hashing and salting techniques to protect against unauthorized access and potential attacks.

## Authorization

Role-based access control (RBAC) is used to manage user permissions, ensuring that users can only access and perform actions within the system according to their assigned roles.

Roles are defined based on user responsibilities, such as insurers, claims staff, and customers, with specific permissions granted to each role to control access to system resources.

# 10.Testing

## Functional Testing

The Web API’s were tested using a tool called Swagger and the results of the test have been displayed below.

|  |  |  |
| --- | --- | --- |
| User API End Point | | View Image |
| GET /api/Users | Passed | [Click to view](https://drive.google.com/file/d/1BQNpBog-_MfrNyLi-fB3UXVrx-9Z7cHO/view?usp=share_link) |
| POST /api/Users | Passed | [Click to view](https://drive.google.com/file/d/1X7QqnfZb9AciS5hM0BuYCXpcrOrbGLGl/view?usp=share_link) |
| GET /api/Users/{id} | Passed | [Click to view](https://drive.google.com/file/d/1BQNpBog-_MfrNyLi-fB3UXVrx-9Z7cHO/view?usp=share_link) |
| PUT /api/Users/{id} | Passed | [Click to view](https://drive.google.com/file/d/135Y6hD16a5MBYsIbLMK8ktvYTDZof6tS/view?usp=sharing) |
| DELETE /api/Users/{id} | Passed | [Click to view](https://drive.google.com/file/d/1RYHcAnqpAOQlhnqsoI905CNWW8XHxU_x/view?usp=share_link) |
| POST /api/Users/login | Passed | [Click to view](https://drive.google.com/file/d/1BQNpBog-_MfrNyLi-fB3UXVrx-9Z7cHO/view?usp=share_link) |

|  |  |  |
| --- | --- | --- |
| Policy API End Point | | View Image |
| GET /api/Policy | Passed | [Click to view](https://drive.google.com/file/d/1zGks-Yab7dhlrfgWhwVorhxSIeHQBS9y/view?usp=share_link) |
| POST /api/Policy | Passed | [Click to view](https://drive.google.com/file/d/157PygvyhaSqtdQUBlXYLAZzL8PN1ZmQM/view?usp=share_link) |
| GET /api/ Policy/{id} | Passed | [Click to view](https://drive.google.com/file/d/1b4mE8cGIkHm-FlOUdYCJSOChUoPDFwN3/view?usp=share_link) |
| PUT /api/Policy /{id} | Passed | [Click to view](https://drive.google.com/file/d/1L6soluZOrBe3TvFpt2lTSJi-k4a5aw-O/view?usp=share_link) |
| DELETE /api/Users/{id} | Passed | [Click to view](https://drive.google.com/file/d/1MJdYtqAt1ABU8nz_t85rNtZKi1bMSAX4/view?usp=share_link) |

|  |  |  |
| --- | --- | --- |
| Customer API End Point | | View Image |
| GET /api/Customer | Passed | [Click to view](https://drive.google.com/file/d/1aYhIGcd3dfp2umUThmdAjc9sF4jfCRaA/view?usp=share_link) |
| POST /api/Customer | Passed | [Click to view](https://drive.google.com/file/d/17-Yu1_yfpHa1X6b0ebwARI2-AFWQCLo9/view?usp=share_link) |
| GET /api/Customer/{id} | Passed | [Click to view](https://drive.google.com/file/d/1F60-pwdTNExuBzVY066IrmUmaRxtLL4L/view?usp=share_link) |
| PUT /api/Customer /{id} | Passed | [Click to view](https://drive.google.com/file/d/1zmNWStzrU4Rk2nP36Fu9-niC7EHki7FK/view?usp=share_link) |
| DELETE /api/Customer /{id} | Passed | [Click to view](https://drive.google.com/file/d/1QDn6XNouJQbnfbxO5ZPNNXOgCogMHHk2/view?usp=share_link) |

|  |  |  |
| --- | --- | --- |
| Vehicle API End Point | | View Image |
| GET /api/ Vehicle | Passed | [Click to view](https://drive.google.com/file/d/1czdm5bxiHgfmIFtAmAl9YBVShtqPHmJo/view?usp=share_link) |
| POST /api/ Vehicle | Passed | [Click to view](https://drive.google.com/file/d/1Y4ld25ZJtrE1Whv2EhmeN_uRnEKpaZRd/view?usp=share_link) |
| GET /api/ Vehicle/{id} | Passed | [Click to view](https://drive.google.com/file/d/1EN0LXaGKFLRay3Hkl7hab2QyCp6yR7QH/view?usp=share_link) |
| PUT /api/ Vehicle /{id} | Passed | [Click to view](https://drive.google.com/file/d/1JrYOd1kosrsJNlDNK-5Ww_KhnWJU4aQw/view?usp=share_link) |
| DELETE /api/ Vehicle /{id} | Passed | [Click to view](https://drive.google.com/file/d/1EN0LXaGKFLRay3Hkl7hab2QyCp6yR7QH/view?usp=share_link) |

|  |  |  |
| --- | --- | --- |
| Premiums API End Point | | View Image |
| GET /api/Premiums | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| POST /api/Premiums | Passed | [Click to view](https://drive.google.com/file/d/1UXwXwibfrvTnqvauCYNA7CMv25EzFjA8/view?usp=share_link) |
| GET /api/Premiums/{id} | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| PUT /api/Premiums /{id} | Passed | [Click to view](https://drive.google.com/file/d/1U8Xj3bP2rQ-HsOwNtGxb1RHfgdUf0xUE/view?usp=share_link) |
| DELETE /api/Premiums /{id} | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |

|  |  |  |
| --- | --- | --- |
| Claims API End Point | | View Image |
| GET /api/ Claims | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| POST /api/ Claims | Passed | [Click to view](https://drive.google.com/file/d/1UXwXwibfrvTnqvauCYNA7CMv25EzFjA8/view?usp=share_link) |
| GET /api/ Claims /{id} | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| PUT /api/ Claims /{id} | Passed | [Click to view](https://drive.google.com/file/d/1U8Xj3bP2rQ-HsOwNtGxb1RHfgdUf0xUE/view?usp=share_link) |
| DELETE /api/ Claims /{id} | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PolicyUpgrade API End Point | | | | | View Image |
| GET /api/PolicyUpgrade | | | | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| POST /api/PolicyUpgrade | | | | Passed | [Click to view](https://drive.google.com/file/d/1UXwXwibfrvTnqvauCYNA7CMv25EzFjA8/view?usp=share_link) |
| GET /api/PolicyUpgrade /{id} | | | | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| PUT /api/PolicyUpgrade /{id} | | | | Passed | [Click to view](https://drive.google.com/file/d/1U8Xj3bP2rQ-HsOwNtGxb1RHfgdUf0xUE/view?usp=share_link) |
| DELETE | /api/ PolicyUpgrade | /{id} |  | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Payment API End Point | | | | | | | View Image |
| GET /api/Payment | | | | | | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| POST /api/Payment | | | | | | Passed | [Click to view](https://drive.google.com/file/d/1UXwXwibfrvTnqvauCYNA7CMv25EzFjA8/view?usp=share_link) |
| GET /api/Payment /{id} | | | | | | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |
| PUT | /api/Payment | | /{id} |  | | Passed | [Click to view](https://drive.google.com/file/d/1U8Xj3bP2rQ-HsOwNtGxb1RHfgdUf0xUE/view?usp=share_link) |
| DELETE | | /api/ Payment | | /{id} |  | Passed | [Click to view](https://drive.google.com/file/d/1TtNz0FkBD7_3oTXOPKu_wEGawx_cObyL/view?usp=share_link) |

React UI Test case were tested manually. The table lists the cases that were tested and their results.

|  |  |  |
| --- | --- | --- |
| * Test case 1 | Create policy | Passed |
| * Test case 2 | View/Search policy | Passed |
| * Test case 3 | Update policy | Passed |
| * Test case 4 | Delete policy | Passed |
| * Test case 5 | Create claim | Passed |
| * Test case 6 | View/Search claim | Passed |
| * Test case 7 | Update claim | Passed |
| * Test case 8 | Delete claim | Passed |
| * Test case 9 | View premium | Passed |
| * Test case 10 | Update premium | Passed |
| * Test case 11 | Receive payment | Passed |
| * Test case 12 | Refund payment | Passed |
| * Test case 13 | Create user | Passed |
| * Test case 14 | Update user | Passed |
| * Test case 15 | Delete user | Passed |

# 11. Deployment

## System Requirements

### Hardware Requirements

The following hardware requirements are recommended for optimal performance of the Wheel Wise Insurance Management System:

* Server: Quad-core processor, 16 GB RAM, 1 TB HDD or SSD storage
* Client: Modern computer or mobile device with a compatible web browser

### Software Requirements

The software requirements for the IMS are as follows:

* Server: Windows Server
* Web server: Microsoft's Internet Information Services (IIS)
* Database: Microsoft SQL Server
* Backend: .NET Framework
* Frontend: HTML, CSS, JavaScript, and a modern web browser (e.g., Google Chrome, Mozilla Firefox, Microsoft Edge), node.js

## Installation and Configuration

To set up the Wheel Wise Insurance Management System, follow these steps:

1. Install the required server operating system, web server, database, and backend language on the server.
2. Download and extract the IMS source code to the appropriate directory on the server.
   * [Download Web API code.](https://github.com/SarveshChand12/WheelWise_API)
   * [Download React UI code.](https://github.com/SarveshChand12/IMS_UI)
   * User Manual
3. Configure the web server and the database according to the User Manual.
4. Create the required database tables and populate them with initial data, as specified in the IMS User Manual.
5. Configure the application's settings, such as database connection details, payment gateway integration, and email server settings.
6. Perform a test run of the IMS to ensure proper installation and configuration.

# Training

To ensure a smooth adoption of the Wheel Wise Insurance Management System, a comprehensive training program and documentation will be provided to users. Training sessions will be conducted for each user role, focusing on the specific functionalities and responsibilities associated with their role. In addition, detailed user guides, help documentation, and technical support will be available to assist users in navigating and utilizing the system effectively.

# Maintenance and Support

The Wheel Wise Insurance Management System will be regularly maintained and updated to address any issues, incorporate new features, and ensure compatibility with the latest technology standards. Technical support will be available to assist users with any system-related issues or concerns, providing prompt and efficient resolution to maintain the optimal performance of the IMS.

# 12. Change Management and Requests

## Change request template.

|  |  |
| --- | --- |
| Requestor's Name: |  |
| Date: |  |
| Description: |  |
| Rationale: |  |
| Priority: | High / Medium / Low |
| Potential Impact: |  |
| Supporting |  |
| Documentation: |  |
| Approval |  |
| Requirements: |  |

# 13. Conclusion

In conclusion, the Wheel Wise Insurance Management System (IMS) provides a robust, comprehensive, and user-friendly solution for managing and administering car insurance policies. Through its advanced features, such as policy management, claim management, premium management, payment management, customer management, and reporting capabilities, the IMS streamlines the insurance process, reducing operational costs and increasing efficiency.

The web-based nature of the system ensures that it is accessible from any location with an internet connection, catering to the needs of a modern, connected workforce. The non-functional requirements, including performance, security, and compatibility, have been meticulously addressed to provide a reliable, secure, and accessible system for its users.

The reporting capabilities offer valuable insights into the company's performance, enabling stakeholders to identify areas for improvement and implement strategic changes. By providing a centralized platform for managing all aspects of car insurance policies, the Wheel Wise IMS not only simplifies the day-to-day tasks of insurers but also enhances the customer experience, leading to increased customer satisfaction and retention.

This technical report has provided an in-depth analysis of the Wheel Wise IMS, its features, design, and benefits, serving as a comprehensive reference for stakeholders involved in the project. The successful implementation of the Wheel Wise IMS will undoubtedly contribute to the growth and success of the company, positioning it as a leader in the car insurance industry.

# 14. Reference

1. React JS
   * Facebook, 2014. React: A JavaScript library for building user interfaces. [online] Available at: https://reactjs.org [Accessed 20 April 2023].
2. ASP.NET Web APIs
   * Microsoft, 2012. ASP.NET Web API: A framework for building HTTP services. [online] Available at: https://docs.microsoft.com/en-us/aspnet/web-api/ [Accessed 20 April 2023].
3. Vehicle Insurance Management
   * Harrington, S.E. and Niehaus, G.R., 2013. Risk Management and Insurance. 3rd ed. Boston, MA: McGraw-Hill/Irwin.

1. Agile Software Development
   * Schwaber, K. and Sutherland, J., 2017. The Scrum Guide: The Definitive Guide to Scrum. [online] Available at: https://[www.scrumguides.org/scrum-guide.html](http://www.scrumguides.org/scrum-guide.html) [Accessed 20 April 2023].
   * Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D., 2001. Manifesto for Agile Software Development. [online] Available at: https://agilemanifesto.org/ [Accessed 20 April 2023].
2. Change Request Process
   * Project Management Institute, 2017. A Guide to the Project Management Body of Knowledge (PMBOK Guide), 6th ed. Newtown Square, PA: Project Management Institute.