

Software Engineering: Requirements Document for ARP Solutions: ARP-IC1 Insurance System

Dr. Ibrahim El Bitar

Prepared by

Anas Albaqeri

Robil Sabek

IC1 and all related software, documentation, and materials are the exclusive property of ARP Solutions and are protected by copyright laws. Unauthorized use, reproduction, or distribution of IC1 or any of its components is strictly prohibited and may result in severe civil and criminal penalties.

ARP Solutions retains all ownership rights, including all intellectual property rights, to IC1 and any modifications, updates, or derivative works thereof. The use of IC1 is governed by the terms and conditions of the license agreement between ARP Solutions and its authorized users.

Any questions regarding the use or licensing of IC1 should be directed to ARP Solutions at [anas.albaqeri@lau.edu, robil.sabek@lau.edu].

ARP Solutions reserves the right to modify, update, or discontinue IC1 at any time without prior notice.

Contents

Glossary:.....	5
0.1 Acronyms:	5
0.2 Definitions:	6
1- Introduction	7
1.1 About ARP Solutions:	7
1.2 ARP-IC1:.....	7
2- Purpose of the System	8
2.1 Objectives:.....	9
2.2 Constraints:	9
2.2.1 General Constraints:.....	9
2.2.2 Functional Constraints:	11
3- Policies:	12
4- Domain Requirements	14
5- User requirements	15
5.1 Non-Functional Requirements:	15
5.2 General User Requirements:.....	16
5.3 Functional User Requirements:.....	17
6- System requirements:	19
7- Conclusion:.....	22
8- Design and Architecture.....	23
8.1 Introduction:	23
8.2 Use Case Diagram	24
8.2.1 Managers:	24
8.2.2 Accountants:	26
8.2.3 Underwriters:	28
8.2 Activity Diagram	30
8.3 Sequence Diagram	32
8.4 Class Diagram:	34
9 Conclusion:	38
Implementation	39

Introduction: 39

Login as Underwriter..... 39

Underwriter functionalities..... 45

Policy Manager Functions: 51

Conclusion:..... 54

Glossary:

0.1 Acronyms:

- **ARP:** Anas & Robil Pacific Solutions LTD.
- **IC1:** is the first version of the insurance software developed exclusively for United Insurance co.
- **SSL** stands for Secure Socket Layer. It is the industry standard method for maintaining an internet connection secure and protecting any sensitive data being communicated between two computers. It does this by preventing hackers from accessing and changing any transferred data, including potentially personal information.
- **API** stands for Application Programming Interphase. An API is a software intermediary that allows two applications to talk to each other.
- **KPI** stands for Key Performance Indicators. KPIs specifically help determine a company's strategic, financial, and operational achievements, especially compared to those of other businesses within the same sector.
- **GDPR** stands for General Data Protection Legislation. GDPR governs the way in which we can use, process, and store personal data.
- **HIPAA** stands for the Health Insurance Portability and Accountability Act of 1996. HIPAA is a federal law that requires the creation of national standards to protect sensitive patient health information from being disclosed without the patient's consent or knowledge.
- **PCI DSS** stands for Payment Card Industry Data Security Standard PCI DSS protects cardholder data and sensitive authentication data wherever it is processed, stored, or transmitted.
- **NAICS** stands for the North American Industry Classification System. NAICS is the standard used by Federal statistical agencies in classifying business establishments for the purpose of collecting, analyzing, and publishing statistical data related to the U.S. business economy.

0.2 Definitions:

- **User requirements** are declarations of the functionality and limitations the system must adhere to.
- **System Requirements:** Explanations that go into greater detail on how the system will function.
- **Domain Requirements:** Requirements pertaining to the software's domain. Requirements in this situation pertaining to education and learning.
- **Functional Needs:** Requirements pertaining to the system's functionalities. They outline in general terms what the system ought to or ought not to accomplish.
- **Non-functional requirements** are those that describe the system holistically rather than being connected to a single behavior or functionality. The majority of the time, they deal with things like software interactions, system performance, security, and accessibility.

1- Introduction

1.1 About ARP Solutions:

Our software company has been providing innovative software solutions to the banking and insurance industries since 2005. Our team of experienced developers and engineers is dedicated to developing software that is customized to meet the unique needs of our clients. We have built a strong reputation for delivering reliable, scalable, and customer-centric software solutions that help our clients streamline their operations, enhance their customer experiences, and improve their bottom line.

We believe in using technology to simplify complex processes, and our software solutions are designed to give our clients a competitive edge in the marketplace. We understand the highly regulated nature of the banking and insurance industries and ensure that our software solutions meet all regulatory requirements and industry standards. Our software development life cycle follows best practices to ensure the highest quality of our software solutions.

Our software solutions are trusted by some of the world's largest banks and insurance companies, reflecting our commitment to innovation, quality, and customer satisfaction. We take pride in the fact that our software has helped our clients achieve their goals and continue to lead the way in the banking and insurance industries.

1.2 ARP-IC1:

United Insurance Company is a leading provider of insurance products for individuals and businesses. In recent years, United has experienced significant growth, resulting in an increase

in the volume of policies, claims, and customers. However, this growth has also led to an increase in operational complexities and challenges, such as the need to process a large volume of claims, manage multiple policies, and maintain regulatory compliance.

To address these challenges, United has decided to invest in a new insurance software system that will enable them to streamline their operations, enhance their customer experience, and improve their compliance and security capabilities. The new system will be implemented as a web-based application and will cater to the needs of insurance companies.

The system will provide United with the ability to manage their policies, claims, and customers more efficiently and effectively. It will include features such as policy management, claims management, underwriting, billing and payment processing, customer relationship management, document management, compliance management, reporting and analytics, and third-party integration.

To better understand what our clients want and need, we have used different techniques for gathering the different requirements for the software ranging from closed interviews with the company's executives, IT-department personnel, and other stakeholders. Moreover, we have conducted thorough research and observation on the workplace environment and created an analysis for a better and clearer understanding of the company's goals and domain of work.

2- Purpose of the System

The purpose of this software engineering document is to provide a comprehensive description of the design, development, and implementation of the ARP-IC1 insurance system. The system is designed to automate and streamline insurance processes, enhancing the efficiency and accuracy of data management and policy administration. This software system aims to provide

users with a user-friendly interface that facilitates the easy management of insurance policies, claims, and related processes. The software solution also integrates robust security measures to ensure the protection of sensitive customer data, conforming to industry regulations and standards. Through this document, stakeholders and developers will have a clear understanding of the software system's features, architecture, and functionality, enabling effective collaboration and successful project delivery.

2.1 Objectives:

- Develop an efficient and scalable system architecture to ensure high performance and scalability, while also maintaining data integrity and security.
- Implement modern web development technologies and frameworks to create an intuitive and responsive user interface.
- Utilize industry-standard security protocols such as SSL, encryption, and multi-factor authentication to ensure data security and compliance with regulations.
- Integrate with third-party APIs and services such as payment gateways, fraud detection software, and external data sources to provide users with additional capabilities and insights.
- Leverage data analytics and reporting tools to provide actionable insights into policy performance, claims trends, and other key performance indicators (KPIs).

2.2 Constraints:

2.2.1 General Constraints:

- **Security Constraints:** The software must implement robust security measures to ensure the confidentiality, integrity, and availability of sensitive customer data. The system must conform to industry regulations and standards such as GDPR, HIPAA, and PCI-DSS.

- **User Authentication and Authorization:** The software should implement an effective user authentication and authorization mechanism to prevent unauthorized access to the system's resources. The system must enforce strong password policies, session timeouts, and access control lists.
- **Policy Management Constraints:** The software should enable insurance agents to easily manage insurance policies, including the creation, modification, and cancellation of policies. The system must also support policy endorsements, renewals, and underwriting rules.
- **Claims Management Constraints:** The software should allow for the efficient management of insurance claims, including the submission, processing, and settlement of claims. The system must also support the investigation of claims, fraud detection, and reporting.
- **Integration Constraints:** The software should be able to integrate with other systems, such as underwriting systems, accounting systems, and customer relationship management systems. The system must also be capable of interfacing with third-party service providers, such as medical providers and auto repair shops.
- **Reporting Constraints:** The software should provide comprehensive reporting capabilities, including the generation of reports on policy sales, claims processing, and financial performance. The system must also support the creation of custom reports and dashboards.
- **Performance Constraints:** The software should be able to handle a large number of transactions efficiently and effectively. The system must be scalable, reliable, and able to handle peak loads without compromising system performance.

2.2.2 Functional Constraints:

- **Premium Calculation Constraints:** The software should be able to calculate insurance premiums based on predefined business rules and rate tables. The system must also support the application of discounts, surcharges, and endorsements based on the policyholder's risk profile.
- **Policy Quoting Constraints:** The software should enable insurance agents to generate policy quotes quickly and accurately based on the policyholder's needs and preferences. The system must also support the generation of proposals and illustrations.
- **Policy Administration Constraints:** The software should provide insurance agents with the ability to manage policies throughout their lifecycle, including renewals, cancellations, endorsements, and claims. The system must also allow for policyholder self-service, such as online policy access and payments.
- **Underwriting Constraints:** The software should enable underwriters to make informed decisions based on policyholder risk assessments and underwriting guidelines. The system must also support the issuance of underwriting decisions and provide visibility into the underwriting process.
- **Claims Processing Constraints:** The software should facilitate the efficient processing of insurance claims, including the ability to file claims, assign claims to adjusters, track claims status, and issue claim payments. The system must also support the resolution of complex claims and appeals.
- **Customer Service Constraints:** The software should enable insurance agents and customer service representatives to provide prompt and effective customer service, including the ability to answer policyholder inquiries, resolve complaints, and provide policy information.

- **Compliance Constraints:** The software should be able to monitor and enforce compliance with regulatory requirements and industry standards, such as insurance laws and regulations, privacy laws, and data protection laws. The system must also support the audit and review of compliance activities.
 - **Billing Constraints:** The software should enable insurance agents to generate accurate invoices and track policyholder payments. The system must also support the collection of premiums and fees and the management of billing disputes.
-

3- Policies:

After consulting with United Insurance stakeholders, we were provided the following main policies that the software should abide by and follow:

1. **Underwriting policy:** This is the policy that determines the conditions under which an insurance policy will be issued. According to The Balance, underwriting policies typically take into account factors such as the age, health, occupation, and lifestyle of the policyholder.

(Source: <https://www.thebalance.com/what-is-insurance-underwriting-2645778>)

2. **Claims policy:** This policy outlines the procedures and requirements for making a claim under an insurance policy. According to Investopedia, a claims policy may include requirements for providing proof of loss, filing deadlines, and other information related to the claims process.

(Source: <https://www.investopedia.com/terms/c/claims-policy.asp>)

3. **Pricing policy:** Insurance companies use a pricing policy to determine the premiums that policyholders will be charged. According to the National Association of Insurance

Commissioners (NAIC), pricing policies may take into account factors such as the risk level of the policyholder, the type of insurance policy, and other relevant factors.

(Source: <https://content.naic.org/article/what-factors-determine-price-my-auto-insurance-policy>)

4. Renewal policy: This policy outlines the conditions under which an insurance policy will be renewed. According to the NAIC, renewal policies may include requirements for maintaining certain coverage levels, paying premiums on time, and other relevant factors.

(Source: <https://www.iii.org/article/renewing-your-auto-insurance-policy>)

5. Exclusion policy: This policy lists the conditions under which an insurance policy will not provide coverage. According to The Balance, an exclusion policy may include conditions such as pre-existing conditions or intentional acts.

(Source: <https://www.thebalance.com/exclusions-in-an-insurance-policy-2645783>)

6. Termination policy: This policy outlines the conditions under which an insurance policy may be terminated. According to the NAIC, termination policies may include provisions for non-payment of premiums, fraudulent claims, or other violations of the policy terms.

(Source: https://www.naic.org/documents/consumer_alert_understanding_insurance.pdf)

7. Policyholder service policy: This policy outlines the level of customer service that policyholders can expect to receive from the insurance company. According to the NAIC, a policyholder service policy may include provisions for handling inquiries, providing information, and resolving disputes.

4- Domain Requirements

Based on given policies, and our extensive work with our development and assessment department we have identified the following domain requirements based on our final ethnographic report:

1. Underwriting policy: This is the policy that determines the conditions under which an insurance policy will be issued. Underwriting policies usually take into account factors such as the age, health, occupation, and lifestyle of the policyholder.
2. Claims policy: This policy outlines the procedures and requirements for making a claim under an insurance policy. It may include requirements for providing proof of loss, filing deadlines, and other information related to the claims process.
3. Pricing policy: Insurance companies use a pricing policy to determine the premiums that policyholders will be charged. This policy may take into account factors such as the risk level of the policyholder, the type of insurance policy, and other relevant factors.
4. Renewal policy: This policy outlines the conditions under which an insurance policy will be renewed. It may include requirements for maintaining certain coverage levels, paying premiums on time, and other relevant factors.
5. Exclusion policy: This policy lists the conditions under which an insurance policy will not provide coverage. For example, a health insurance policy may exclude coverage for pre-existing conditions.

6. Termination policy: This policy outlines the conditions under which an insurance policy may be terminated. It may include provisions for non-payment of premiums, fraudulent claims, or other violations of the policy terms.
 7. Policyholder service policy: This policy outlines the level of customer service that policyholders can expect to receive from the insurance company. It may include provisions for handling inquiries, providing information, and resolving disputes
-

5- User requirements

5.1 Non-Functional Requirements:

After our project management executives conducted open interviews with United Insurance different stakeholders, and after passing by their analysis to the analysis team then our development team in ARP Solutions, we have finalized the following main non-functional requirements:

- 1) Performance: The system must be able to handle a high volume of transactions without compromising speed, with a response time of 2 seconds or less.
- 2) Scalability: The system should be able to accommodate future growth without any disruption in service and should be able to scale up or down as required.
- 3) Availability: The system should be available 24/7 with minimal downtime and should be designed to minimize the risk of system failure.
- 4) Security: The system must employ multiple layers of security to ensure the confidentiality, integrity, and availability of sensitive data, and should be compliant with industry regulations and standards.

- 5) Usability: The system should be easy to use and accessible on different devices and platforms, with clear instructions and error messages to facilitate efficient use.

5.2 General User Requirements:

- 1) Through our analysis and different interviews with the stakeholders and especially the company's employees, we have reached the following general user requirements that our system must achieve:
- 2) User Interface: The software shall have a user-friendly interface that is easy to navigate, with intuitive menus, dashboards, and reporting tools.
- 3) Compliance: The software shall be designed to comply with all relevant industry regulations and standards, such as GDPR, HIPAA, and PCI DSS.
- 4) Scalability: The software shall be scalable to accommodate a growing number of customers, policies, claims, and other data as the insurance company expands.
- 5) Security: The software shall have robust security features to ensure the confidentiality, integrity, and availability of sensitive customer and company data. This includes secure user authentication, data encryption, and access controls.
- 6) Integration: The software shall be able to integrate with other systems and applications such as CRM, accounting, and claims management software to streamline workflows and increase efficiency.
- 7) Customization: The software shall allow for customization to meet the unique needs and requirements of the insurance company, such as policy types, underwriting rules, pricing models, and reporting.

- 8) Performance: The software shall be able to handle high volumes of data and transactions, with fast response times and minimal downtime.
- 9) Reporting and Analytics: The software shall provide robust reporting and analytics capabilities to help insurance companies analyze and interpret data, identify trends, and make informed business decisions.
- 10) Mobile Compatibility: The software shall be compatible with mobile devices to allow users to access it on the go.
- 11) Support and Maintenance: The software shall come with adequate support and maintenance services to ensure that any issues or bugs are promptly addressed, and that the software is kept up to date.

5.3 Functional User Requirements:

As for the functional requirement of the software, our team has gone through the different successive steps of formulating the appropriate functional requirements for ARP-IC1 which included but were not limited to requirements elicitation with stakeholders, open and closed interviews with the company's employees, situation modeling and analysis, reference models analysis including Oracle OIPA and HubSpot CRM IMS (Insurance Management System). The following requirements were found to be the main functional requirements needed by our client United Insurance.

- 1) User login: The software shall allow users to create an account and log in to access their account information and perform various actions based on their user role.

- 2) Policy Management: The software shall allow authorized employees to create, manage, and modify insurance policies, including policy documents, coverages, premiums, and deductibles.
- 3) Claims Management: The software shall allow authorized employees to manage the entire claims process, from initial reporting to claim settlement, including claim tracking, investigation, and payment processing.
- 4) Underwriting: The software shall provide authorized employees with the ability to underwrite policies, assess risks, calculate premiums, and determine policy eligibility.
- 5) Billing and Payment Processing: The software shall allow authorized employees to generate invoices, accept payments, and process refunds, including online payment options and automated payment reminders.
- 6) Customer Relationship Management: The software shall provide authorized employees with a full view of the customer, including their policy information, claims history, and communication history.
- 7) Document Management: The software shall allow authorized employees to manage all policy documents and forms, including electronic signature capabilities.
- 8) Compliance Management: The software shall enable authorized employees to manage compliance with regulatory requirements and standards, including automated compliance checks and reports.

- 9) Reporting and Analytics: The software shall provide authorized employees with reports and analytics on policy performance, claims trends, and other key performance indicators (KPIs).
 - 10) Communication Management: The software shall provide authorized employees with tools to communicate with customers, agents, and other stakeholders, including email, SMS, and chat functionalities.
 - 11) Third-Party Integration: The software shall integrate with third-party tools and services, such as payment gateways, fraud detection software, and external data sources.
-

6- System requirements:

1. User login:

- 1.1 User Authentication: The software shall provide a secure user authentication mechanism to ensure that only authorized users can access their accounts.
- 1.2 User Account Management: The software shall allow users to create and manage their accounts, including resetting passwords and updating personal information.
- 1.3 Role-Based Access Control: The software shall implement role-based access control to restrict user access to certain functionalities based on their assigned role.
- 1.4 Session Management: The software shall manage user sessions to ensure that users remain logged in only for a specific period and automatically log them out after a defined period of inactivity.
- 1.5 Password Policies: The software shall enforce password policies, such as minimum length, complexity, and expiration, to ensure user account security.
- 1.6 Multi-Factor Authentication: The software shall support multi-factor authentication, such as SMS verification or token-based authentication, to provide an additional layer of security for user accounts.
- 1.7 Audit Logging: The software shall log all user login and logout attempts, including the user's IP address, date, and time, for auditing and security purposes.

2. Policy Management:

- 2.1 System shall have a simple descriptive interface for creating, managing, and modifying insurance policies.
- 2.2 System shall have the ability to create and store policy documents, including electronic signature capabilities.
- 2.3 System shall have options to customize coverages, premiums, and deductibles based on policy type.

- 2.4 System shall provide validation checks to ensure that policies meet regulatory requirements and standards
 - 2.5 System shall provide automated notifications for policy renewals, cancellations, and expirations
3. Claims Management:
- 3.1 System shall provide user interface for managing the entire claims process, from initial reporting to claim settlement.
 - 3.2 System shall provide the ability to track and manage claims throughout the process.
 - 3.3 System shall provide automated investigation and payment processing to ensure timely claim settlements.
 - 3.4 System shall provide integration with third-party tools for fraud detection and investigation.
 - 3.5 System shall provide the ability to generate reports and analytics on claims trends and performance
4. Underwriting:
- 4.1 System shall provide user interface for assessing risks, calculating premiums, and determining policy eligibility.
 - 4.2 System shall provide customizable underwriting rules and guidelines based on policy type.
 - 4.3 System shall provide the ability to generate quotes and proposals for potential policyholders.
 - 4.4 System shall provide integration with external data sources for risk assessment and analysis.
 - 4.5 System shall provide automated underwriting and policy approval process to ensure efficient and accurate underwriting decisions
5. Billing and Payment Processing:
- 5.1 System shall provide user interface for generating invoices and accepting payments.
 - 5.2 System shall provide ability to process refunds and cancellations.
 - 5.3 System shall provide integration with third-party payment gateways for online payment options.
 - 5.4 System shall provide automated payment reminders and notifications for late payments.
 - 5.5 System shall provide customizable billing and payment rules based on policy type and customer preferences.
6. Customer Relationship Management:
- 6.1 System shall provide user interface for providing a full view of the customer, including their policy information, claims history, and communication history.
 - 6.2 System shall provide the ability to manage customer accounts, including policy changes and updates.
 - 6.3 System shall provide automated customer service and support, including chat and email functionalities.
 - 6.4 System shall provide integration with external communication tools for seamless communication with customers and agents.
7. Document Management:
- 7.1 System shall provide user interface for managing all policy documents and forms, including electronic signature capabilities.
 - 7.2 System shall provide the ability to generate and store policy documents, including endorsements and riders.
 - 7.3 System shall provide automated document management and storage, including version control and access control.

- 7.4 System shall provide integration with external document management systems for seamless document processing and management.

8. Compliance Management:

- 8.1 System shall provide user interface for managing compliance with regulatory requirements and standards.
- 8.2 System shall provide automated compliance checks to ensure that policies and claims meet regulatory requirements.
- 8.3 System shall provide integration with external compliance management tools for seamless compliance processing and management.
- 8.4 System shall provide customizable compliance rules and guidelines based on policy type and regulatory requirements.

9. Reporting and Analytics:

- 9.1 System shall provide user interface for providing reports and analytics on policy performance, claims trends, and other KPIs.
- 9.2 System shall provide customizable reporting options, including charting and graphing functionalities.
- 9.3 System shall provide integration with external analytics tools for advanced data analysis and processing.
- 9.4 System shall provide automated reporting and analytics based on user-defined KPIs and metrics.

10. Communication Management:

- 10.1 System shall provide user interface for providing tools to communicate with customers, agents, and other stakeholders.
- 10.2 System shall provide integration with external communication tools for seamless communication with customers and agents
- 10.3 System shall provide automated communication and notification functionalities, including email, SMS, and chat functionalities
- 10.4 System shall provide customizable communication rules and guidelines based on policy type and customer preferences

11. Third-Party Integration:

- 11.1 Software shall provide user interface for integrating with third-party tools and services, such as payment gateways, fraud detection software, and external data sources.
- 11.2 Software shall provide Integration with external APIs for seamless data processing and management.
- 11.3 System shall provide customizable integration rules and guidelines based on policy type and user preferences.
- 11.4 System shall provide automated integration and data processing to ensure seamless data flow between the software and external systems.

7- Conclusion:

This document should serve as a complete requirement specification document being stage one of our software developing cycle. Various techniques were used to formulate the final requirement report meeting all client's requirements and needs and having met the standard qualities and policies. After conducting the evaluation and validation for the requirements document as part of the incremental approach we are following for this software development, we will next draw a clear picture of the flow of control between the different users and show in detail the different relational diagrams linking the different users with the system.

8- Design and Architecture

8.1 Introduction:

As ARP Solutions, we are proud to introduce our latest project: ARP-IC1, an insurance system designed to provide comprehensive coverage to our clients. In the phase of Design and Architecture, we have carefully crafted the system's architecture to ensure it meets the requirements of our clients while ensuring optimal functionality.

To achieve this, we have taken a holistic approach to the design and architecture of ARP-IC1. We have developed a series of diagrams that provide a comprehensive view of the flow of control between different users of the system. These diagrams highlight the main architecture of the software, including the most vital and expected use cases. While we did not take every possible use case into consideration, we have ensured that the base model we provided covers all aspects of the system and will be able to handle any additional requirements that may arise in the future.

Our approach was to focus on the most inclusive flow of controls between functions and users, ensuring that we cover all aspects of the system. This strategy was deemed the best for the time limit given, and we believe it will provide the most efficient and effective solution for our clients.

In developing the design and architecture of ARP-IC1, we conducted extensive interviews and ethnography reports to gain insight into the workflow and requirements of our clients. This analysis helped us to identify areas that needed improvement and allowed us to develop a more sufficient product.

Moving forward with the project, we are excited to present the various diagrams that we have developed in the phase of Design and Architecture. These diagrams provide a comprehensive view of ARP-IC1's system architecture, highlighting the flow of control between different users and functions.

First, the use case diagram will provide a visual representation of the system's functional requirements, illustrating the interactions between users and the system.

Second, the activity diagram will provide a detailed view of the system's business processes, illustrating the flow of control between different activities and how they relate to each other.

Third, the sequence diagram will provide a visual representation of the interaction between different objects in the system, illustrating how messages are exchanged and how different objects collaborate to achieve a specific task.

Lastly, the class diagram will provide a structural view of the system, illustrating the different classes, their attributes, and their relationships. It will provide an understanding of the system's data model and how it is structured.

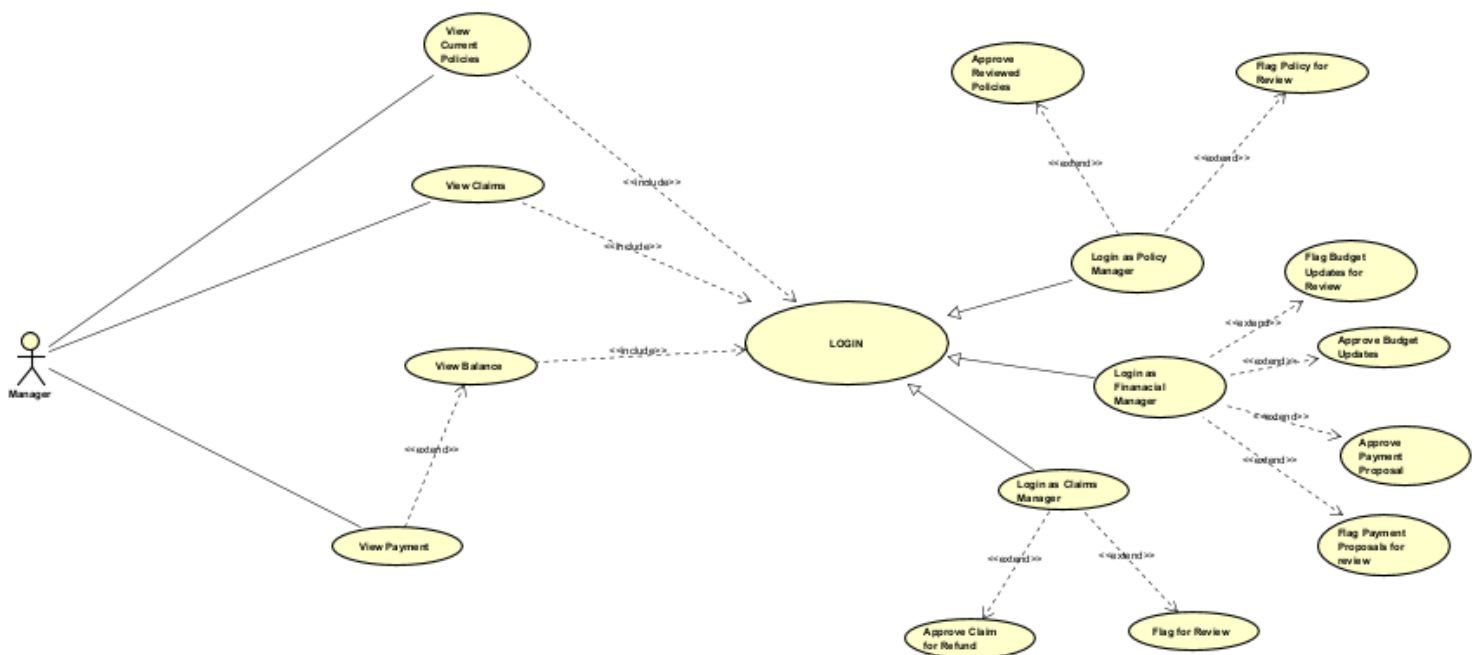
Overall, these diagrams will provide a comprehensive view of ARP-IC1's system architecture, enabling a better understanding of the system's design and functionality. With these diagrams, we are confident that we have developed a solution that will revolutionize the insurance industry and exceed our clients' expectation.

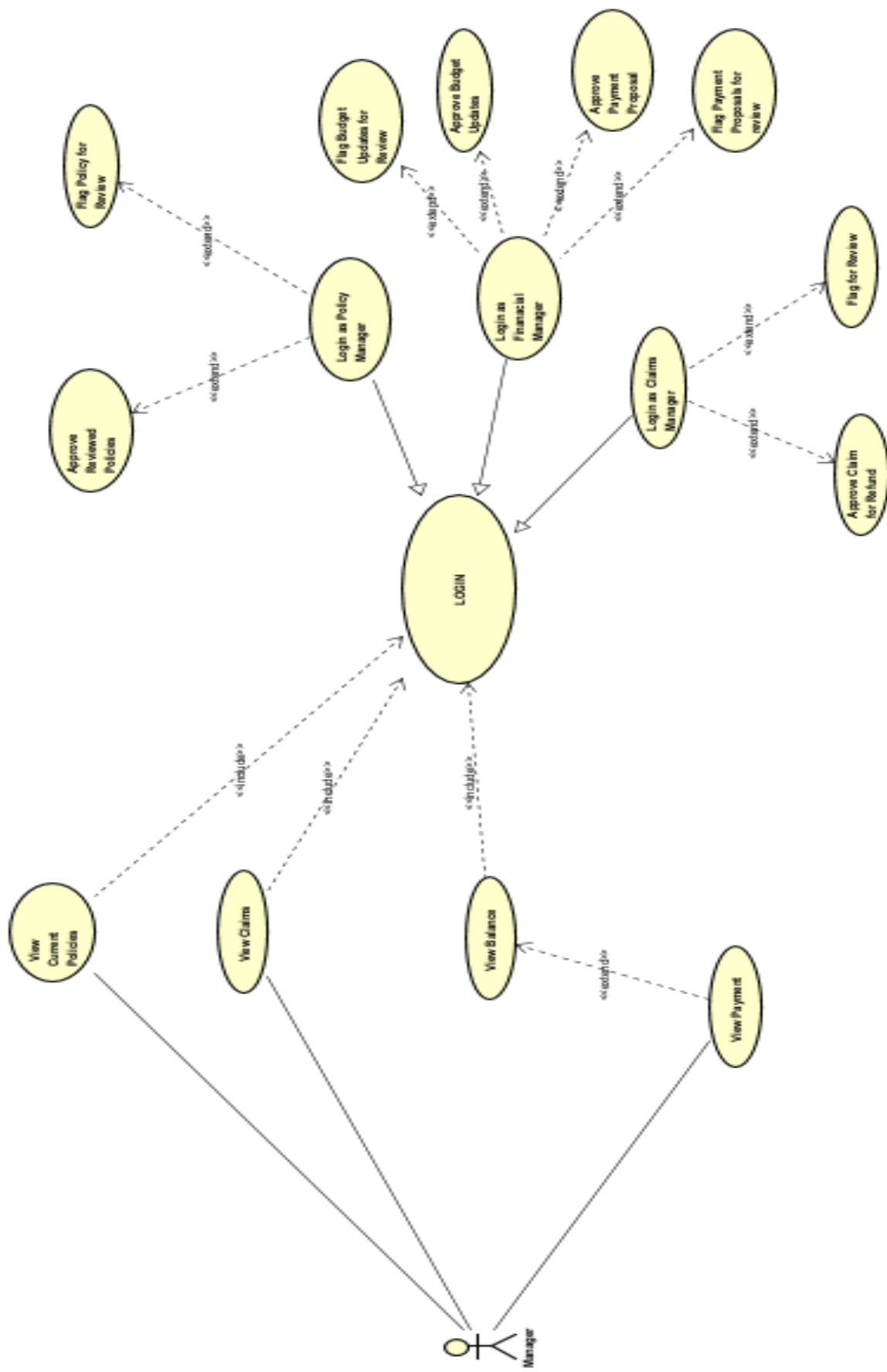
8.2 Use Case Diagram

In the following section, we will be taking a thorough look on the flow of work of the system, represented by the continuous cycle of functions and intercommunications between the different layers of management within the company.

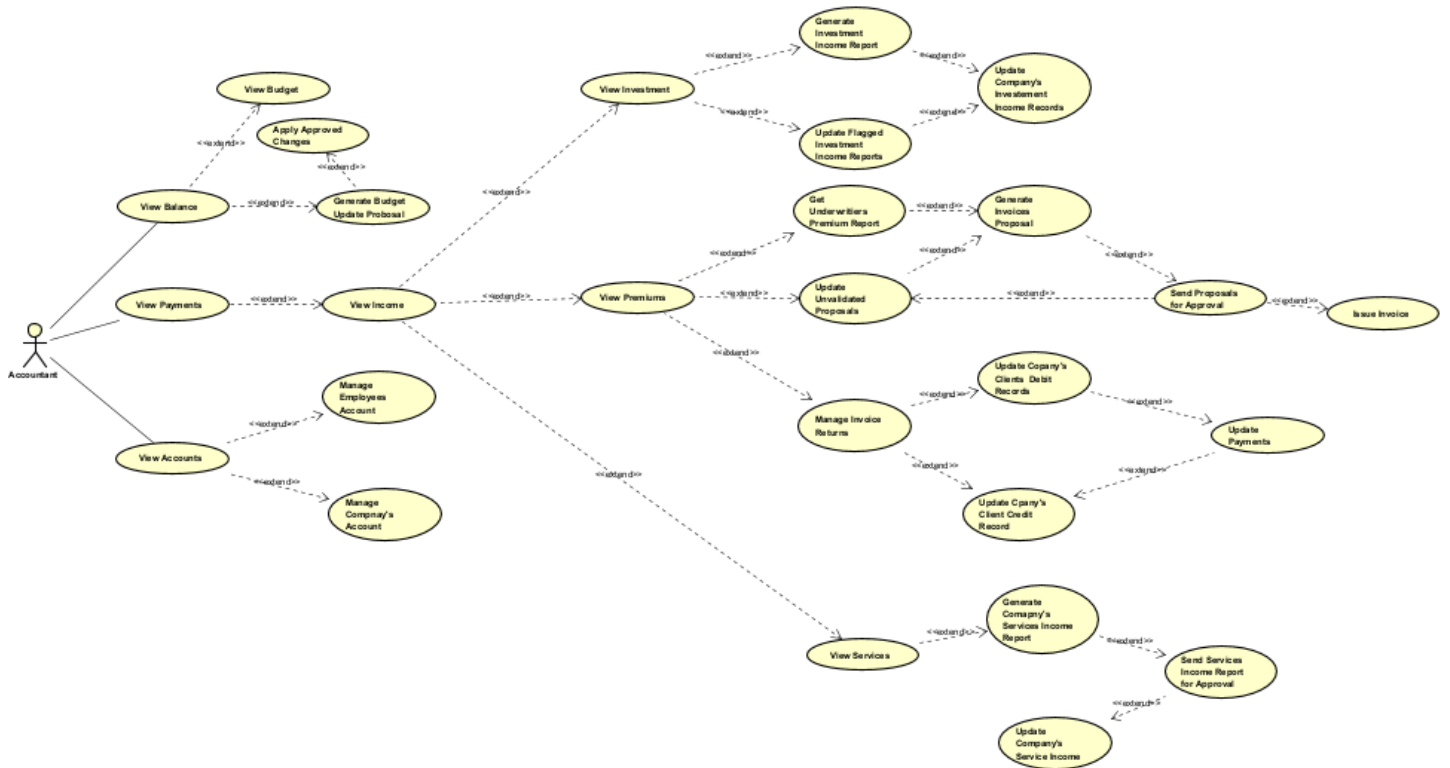
United Insurance, one of the biggest Insurance Corporations in the world, have helped us identify the main flow of work in the company, and so we were able to focus on the main scenarios that represent everyday transactions and flow between employees from different managerial layers. We have chosen the most common use cases to represent here as they cover almost all the cases that will be confronted in any of the company's different departments.

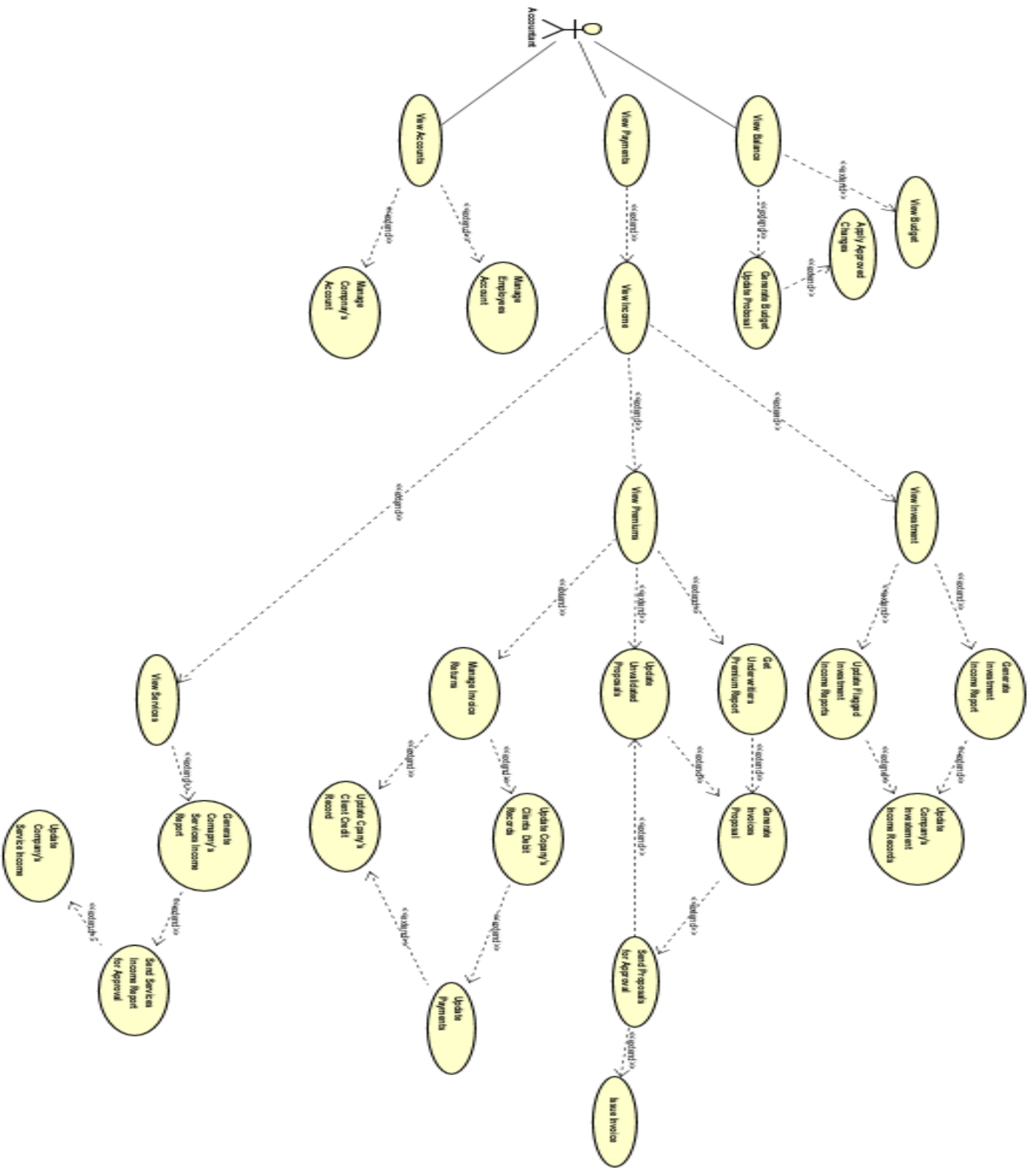
8.2.1 Managers:



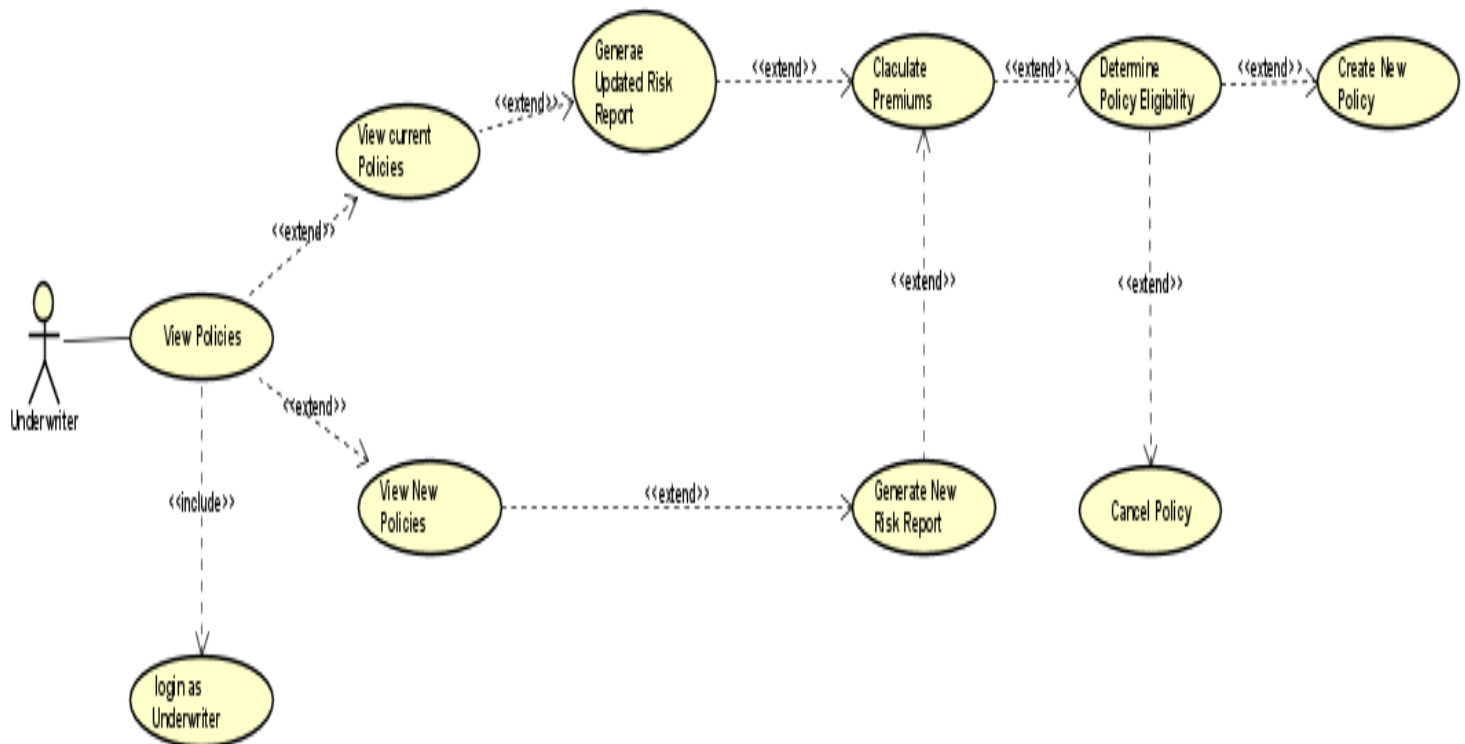


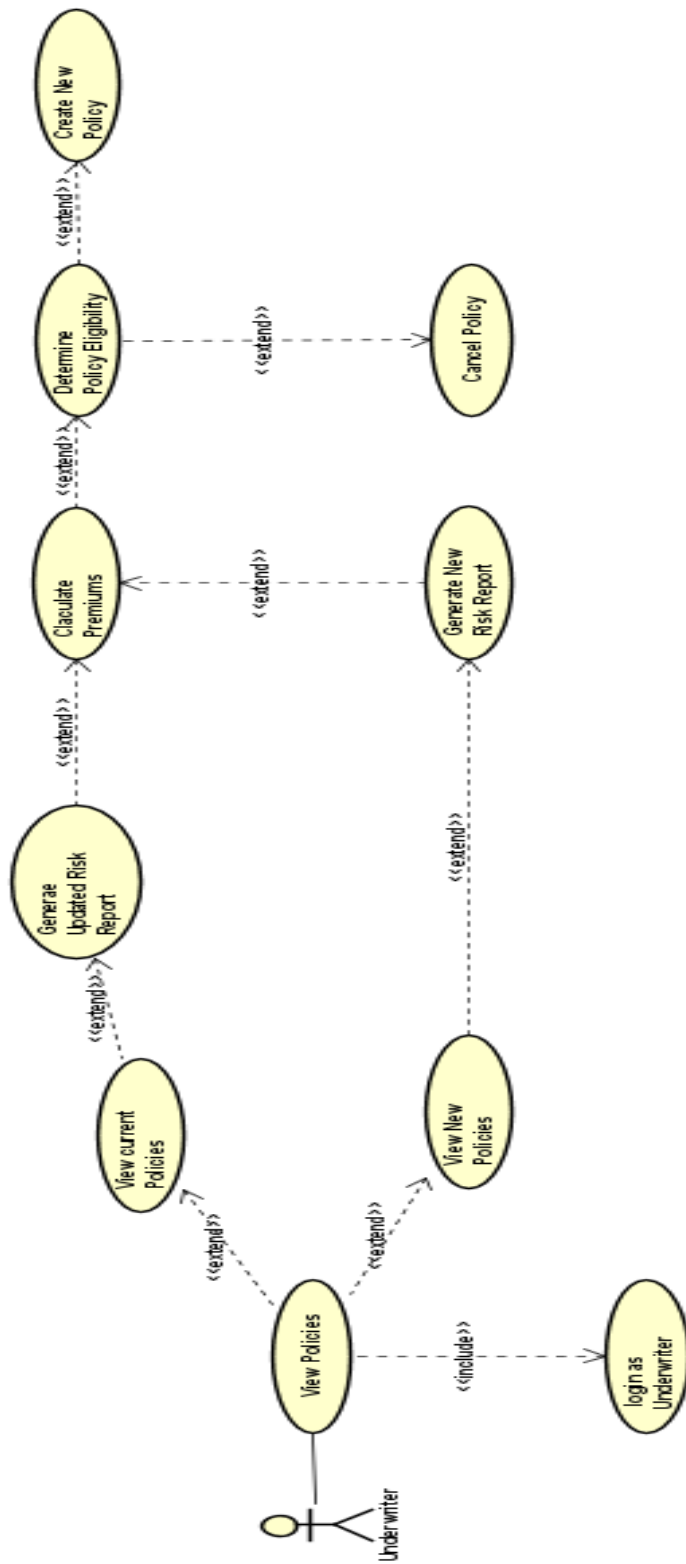
8.2.2 Accountants:





8.2.3 Underwriters:





In the previous pages, we have shown three different use cases for different employees involving different responsibilities in the company.

We notice that the managers all share the ability to view the vital status of the company, whether it is the budget, records or policies. However, under each of the specialized managers have the control over the vital functions in each department such as approving payments, issuing new policies or updating budget.

As move down the managerial hierarchy, the number of functions increase, and employees have the most portion of functions that add up to fulfill the company's requirements. These functions that need special approvals are always communicated within the system to the management personnel in charge.

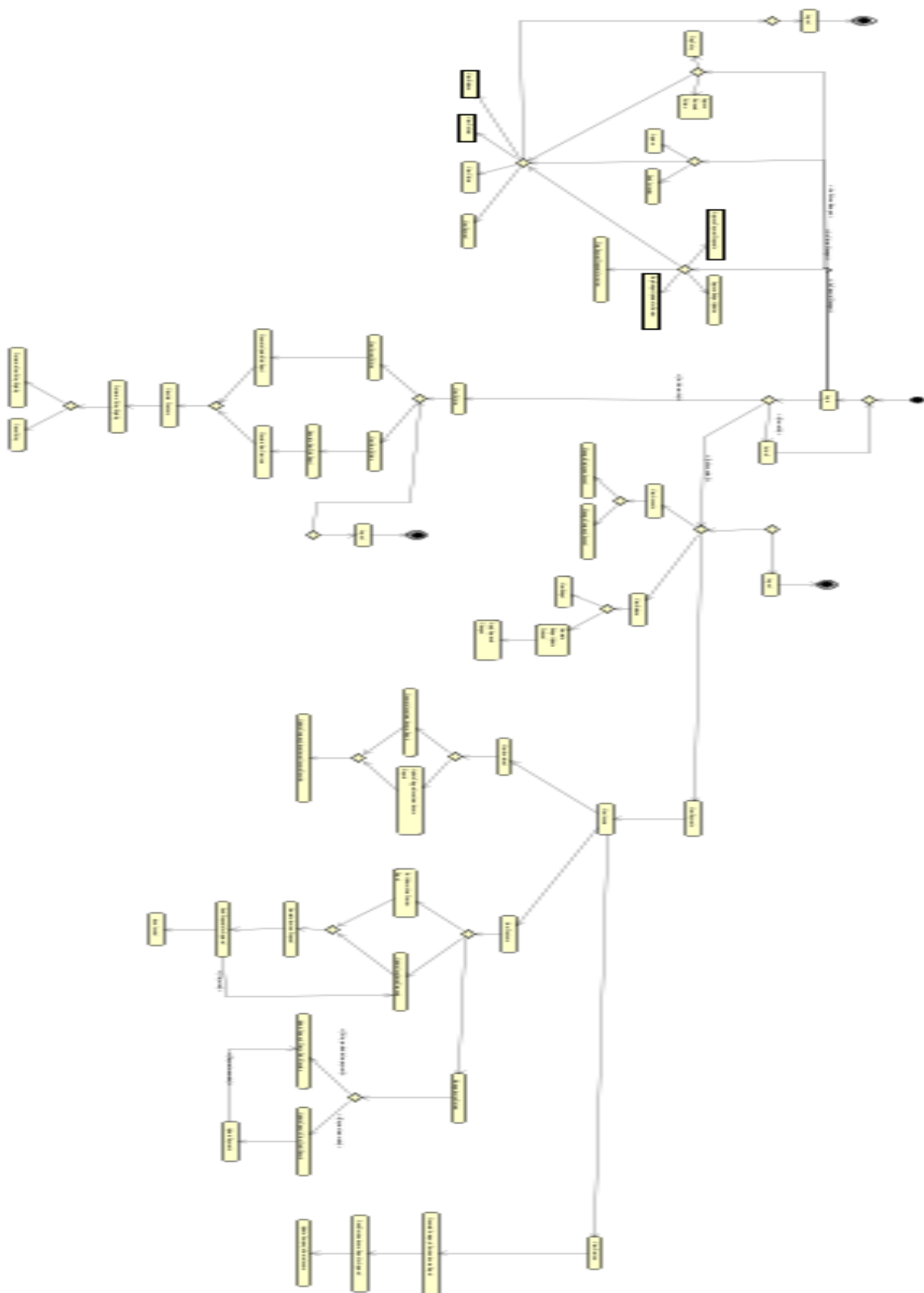
It is noticed throughout the diagrams that the flow of work goes in a circular way, whether starting from high in the managerial hierarchy with claims manager noticing a claim that is outside the policy coverage and flagging the transaction back to the employees in the department to review and advise the situation, or the other way around with employees sending payments forms to be approved and advised by managers. Both scenarios demonstrate the flexibility and systematic approach that the system provides for the most efficient performance.

8.2 Activity Diagram

In the following section, we will be looking in depth in the different activities flow and chains. the activity diagram can be used to model processes such as policy creation, claims processing, underwriting, and customer management. Each of these processes can be broken down into a set of smaller activities, which can then be represented using nodes and edges in the activity diagram.

Overall, an activity diagram can be a powerful tool for modeling complex processes in a software system and can help to identify potential bottlenecks or areas for optimization. By providing a visual representation of the system's behavior, it can also help to facilitate communication and collaboration between different stakeholders involved in the software development process.

(note: all diagrams will be submitted along the zip file, the following submitted ones are the best we were able to compress such big diagram)



8.3 Sequence Diagram

We will be tackling a sequence diagram for an underwriter employee, analyzing the steps one by one from the moment the employee logged to his account and up till he policy was issued.

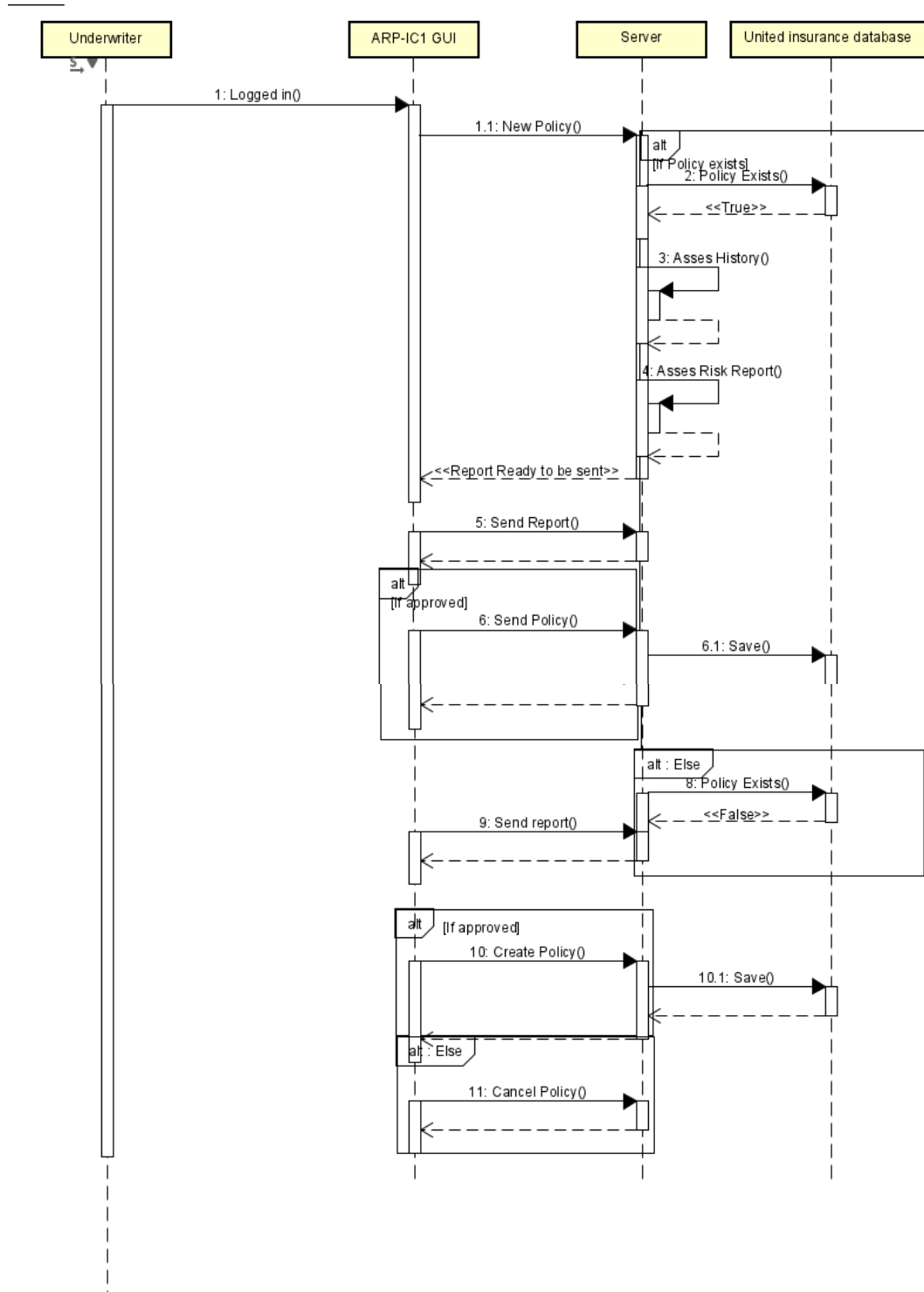
The various objects or components involved in the process would normally be represented by a set of lifelines in a sequence diagram for an underwriting scenario. They could be the applicant, the underwriter, the team in charge of administering the policy, and any other pertinent parties.

The diagram would also show a sequence of communications or interactions between these lifelines, which would stand in for the different phases of the underwriting procedure. For instance, if the applicant applies form, the underwriter receives a notification to review the application. The applicant can then receive a second communication from the underwriter requesting more information from them. The underwriter will decide on the policy and notify the policy administration team once all the essential data has been gathered.

The Process starts with our employee logged in to the system, on the GUI of the system, the employee choose to create a new policy, the server then sends a query with the policy details to the database to check if the policy exist or if it is a new policy. Have the policy showed to be already in the database, the server flags the policy as an old one and an update process takes place instead. First, a history assessment on the policy is made as it is the most determinantal variable in updating policies. With the result of the history of the policy given and taken into consideration, a general risks assessment is ran using the appropriate actuarial equations satisfying the type of policy offered.

The server then, return reports results and the next step is calculating feasibility for services. This step is still not automated in most of insurance companies if not all today and instead traditional bargaining usually takes place between client and company, however, this Is a really powerful tool that could help make feasibility negotiations much more affective had it been used to serve that purpose

If both the company's and software efforts were unable to find middle grounds between the client and company, the policy is canceled and deleted form requests history.



8.4 Class Diagram:

ALL MANAGERS
- Manager ID: int - Name: string - Email: string -password: string
+viewCurrentPolicies(): List<Policy> +viewClaims(): List<Claim> +viewBalance(): double +viewPayment(): List<Payment>

PAYMENT
-paymentID: int -policyID: int paymentAmount: double -paymentDate: date

POLICY MANAGER
- Manager ID: int - Name: string - Email: string -password: string
+approveReviewedPolicy(policyID: int): bool +flagPolicyForReview(policyID: int): bool

The above diagram represents the class PolicyManager, which has two methods to approve reviewed policies and flag policies for review. The class Policy represents the policy information, including the policy ID, policy holder, policy type, policy start and end dates, and a Boolean indicating whether the policy has been reviewed. The attributes employeeID, name, email, and password are private and can only be accessed through getter methods or within the class. The return types of the methods are specified in the table with data types.

FINANCIAL MANAGER
- Manager ID: int - Name: string - Email: string -password: string
+flagBudgetUpdatesForReview(updateID: int): bool +approveBudgetUpdates(updateID: int): bool +approvePaymentProposal(proposalID: int): bool +flagPaymentProposalsForReview(proposalID: int): bool

BudgetUpdate
- updateID: int -updateAmount: double -reviewed: bool

PaymentProposal
proposalID: int proposalAmount: reviewed: bool

The above diagram represents the class FinancialManager, which has four methods to flag budget updates for review, approve budget updates, approve payment proposals, and flag payment proposals for review. The classes BudgetUpdate and PaymentProposal represent the budget and payment proposal information, respectively, including the proposal ID, proposal amount, and a boolean indicating whether the proposal has been reviewed. The attributes employeeID, name, email, and password are private and can only be accessed through getter methods or within the class. The return types of the methods are specified in the table with data types.

CLAIMS MANAGER
- Manager ID: int - Name: string - Email: string -password: string
+flagClaimForReview(claimID: int): bool +approveClaimForRefund(claimID: int): bool

CLAIM
-claimID: int policyID: int claimAmount: double status: string claimDate: date

The above diagram represents the class ClaimsManager, which has two methods to flag claims for review and approve claims for refund. The class Claim represents the claim information, including the claim ID, policy ID, claim amount, and a boolean indicating whether the claim has been reviewed. The attributes employeeID, name, email, and password are private and can only be accessed through getter methods or within the class. The return types of the methods are specified in the table with data types.

UNDERWRITER
- User ID: int - Name: string - Role: string - Email: string
+ viewPolicies(): List<Policy> + generateRiskReport(policy: Policy): RiskReport + calculatePremium(policy: Policy): double + determinePolicyEligibility(policy: Policy): boolean + issuePolicy(policy: Policy): boolean + cancelPolicy(policy: Policy): boolean

Accountant
-employeeID: int -name: string -email: string -password: string
+viewBalance() +viewBudget() +generateBudgetProposal() +applyApprovedChanges() +viewPayments() +viewIncome() +viewInvestment() +generateInvestmentIncomeReport() +updateFlaggedInvestmentIncomeReports() +updateCompanyInvestmentIncomeRecords() +viewPremiums() +getUnderwritersPremiumReport() +updateUnvalidatedProposals() +generateInvoiceProposals() +sendProposalsForApproval() +issueInvoice() +manageInvoiceReturns() +updateCompanyClientsDebitRecords() +updateCompanyClientsCreditRecord() +updatePayments() +viewServices() +generateCompanyServicesIncomeReport() +updateCompanyServiceIncome() +sendServicesIncomeReportForApproval() +viewAccounts() +manageEmployeesAccount() +manageCompanyAccount()

The above diagram represents the class Accountant, which has various methods related to the accountant's job functions in an insurance company. The attributes employeeID, name, email, and password are private and can only be accessed through getter methods or within the class.

POLICY
-policyID: int -policyHolder: string -policyType: string -policyStartDate: date policyEndDate: date -reviewed: bool

9 Conclusion:

In conclusion, the use case diagram, activity diagram, sequence diagram, and class diagram provide a comprehensive overview of the ARP-IC1 insurance system. These diagrams help to illustrate the different aspects of the system, including the functionality, business processes, and system architecture.

The use case diagram provides a high-level view of the system's functionality, identifying the various actors and use cases that the system supports. It also helps to identify the interactions between the actors and the system, and the main goals of the system.

The activity diagram provides a detailed representation of the different processes and workflows involved in the insurance system. It helps to illustrate the sequence of activities and events that occur during the various processes, including policy creation, claims processing, underwriting, and customer management.

The sequence diagram provides a detailed representation of the interactions between different components of the system during the underwriting process. It helps to illustrate the flow of events and the dependencies between different components, such as the applicant, the underwriter, and the policy administration team.

Finally, the class diagram provides a detailed representation of the system's architecture, illustrating the different classes, attributes, and relationships between them. It helps to identify the main components of the system and their roles, and how they interact with each other.

Overall, these diagrams provide a clear and concise overview of the ARP-IC1 insurance system, and help to facilitate communication and collaboration between different stakeholders involved in the software development process. By providing a visual representation of the system's behavior, they can help to identify potential issues or inefficiencies, and improve the overall design and functionality of the system.

Implementation

Introduction:

The implementation section of this report will provide a comprehensive overview of the development process of the IC1 web application insurance system. This section will detail the technology and tools used to build the system, as well as the different stages of development, including planning, design, implementation, and testing.

The IC1 insurance system is a web-based application designed to provide insurance companies with a platform to manage their policies, claims, and customers. The system is designed to be highly secure, scalable, and user-friendly, allowing insurers to easily access, update, and manage policy information.

The system was developed using the latest web development technologies, including PHP, HTML, CSS, and JavaScript. The database used to store the insurance data is MySQL, which provides fast and secure data storage and retrieval.

In the following sections, we will provide a detailed overview of the development process, including the tools and technologies used, as well as the different stages of development. We will also provide a detailed analysis of the system's features and functionality, as well as the challenges faced during development and the solutions used to overcome them.

In this document, we will be showing four cases of use:

- 1- Login as an Underwriter
- 2- Login as a Policy Manager
- 3- Adding policies as an Underwriter
- 4- Displaying policies both as Underwriter and Policy Manager
- 5- Approving / Denying Pending Policies

Login as Underwriter

For the login functionality, we used a combination of front and backend functions. login page that sends data to a PHP script on the server to authenticate the user. The PHP script connects to a MySQL database and checks if the provided credentials are valid. If the credentials are valid, it sets session variables and returns a JSON response indicating a successful login along with the user's position within the organization.

The login page also contains JavaScript that is executed when the user clicks the

```
<?php
header('Content-Type: application/json');

//start our session
session_start();
session_destroy();
session_start();

//get the username and password from the form
$un = $_POST['username'];
$pass = sha1($_POST['password']);

//connect to our database
$conn = mysqli_connect('localhost', 'root', 'root', 'ic1');

//check to see if connection was successful
if (!$conn) {
    die('Connection failed: ' . mysqli_connect_error());
}

//check to see if username and password match what is in the database
$sql = "SELECT * FROM credentials WHERE Username = '$un' AND Password = '$pass'";

//run our query and get the results
$result = mysqli_query($conn, $sql);
```

"Login" button. The JavaScript gets the values of the username and password fields, and sends an XMLHttpRequest to the PHP script to authenticate the user. If the login is successful, the JavaScript redirects the user to the appropriate page based on their position within the organization.

The login page also has a "Cancel" button that clears the input fields.

The php file that handles the login submission, retrieving data from the database and redirecting the employee to their particular page.

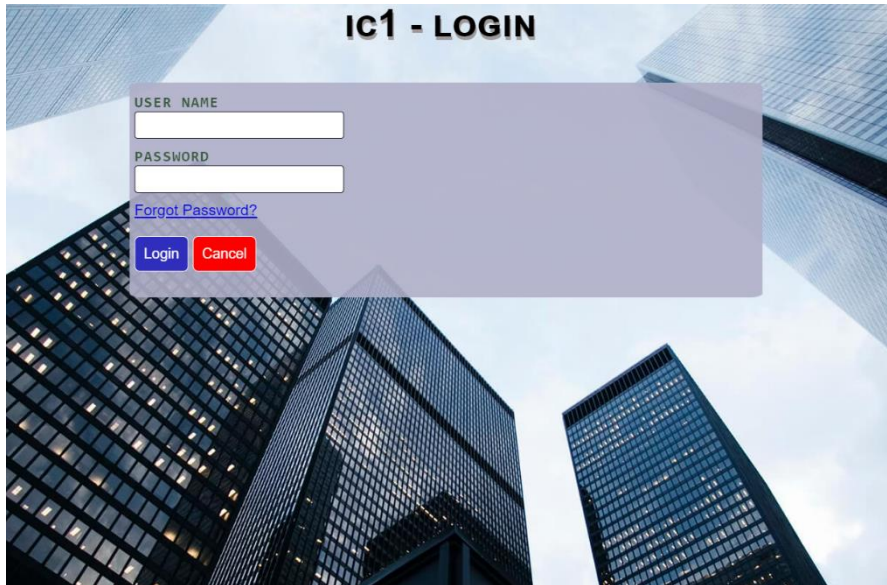
```
//check to see if we got any results
if (mysqli_num_rows($result) == 1) {
    //get the row from the result
    $row = mysqli_fetch_assoc($result);
    //check which department is user and which position is the user
    $department = $row['Dept'];
    //check if user is a manager or employee
    $position = $row['PositionID'];
    //set the session variable with the username
    $_SESSION['username'] = $un;
    $_SESSION['user_id'] = $row['ID']; // set session variable with user ID

    //check which department the user is in
    if ($department == 0) {
        $response = array('success' => true, 'position' => 'GeneralManager', 'message' => 'Login successful');
        echo json_encode($response);
        exit();
    }
    //financial department
    if ($department == 1) {
        if ($position == 0) {
            $response = array('success' => true, 'position' => 'FinancialManager', 'message' => 'Login successful');
            echo json_encode($response);
            exit();
        }
        if ($position == 1) {
            $response = array('success' => true, 'position' => 'Accountant', 'message' => 'Login successful');
            echo json_encode($response);
            exit();
        }
    }
    //policy department
    if ($department == 2) {
        if ($position == 0) {
            $response = array('success' => true, 'position' => 'PolicyManager', 'message' => 'Login successful');
            echo json_encode($response);
            exit();
        }
        //policy department
        if ($department == 2) {
            if ($position == 0) {
                $response = array('success' => true, 'position' => 'PolicyManager', 'message' => 'Login successful');
                echo json_encode($response);
                exit();
            }
            if ($position == 1) {
                $response = array('success' => true, 'position' => 'Underwriter', 'message' => 'Login successful');
                echo json_encode($response);
                exit();
            }
        }
    }
    //claims department
    if ($department == 3) {
        if ($position == 0) {
            $response = array('success' => true, 'position' => 'ClaimsManager', 'message' => 'Login successful');
            echo json_encode($response);
            exit();
        }
    }
} else {
    $response = array('success' => false, 'message' => 'Invalid username or password');
    echo json_encode($response);
    exit();
}

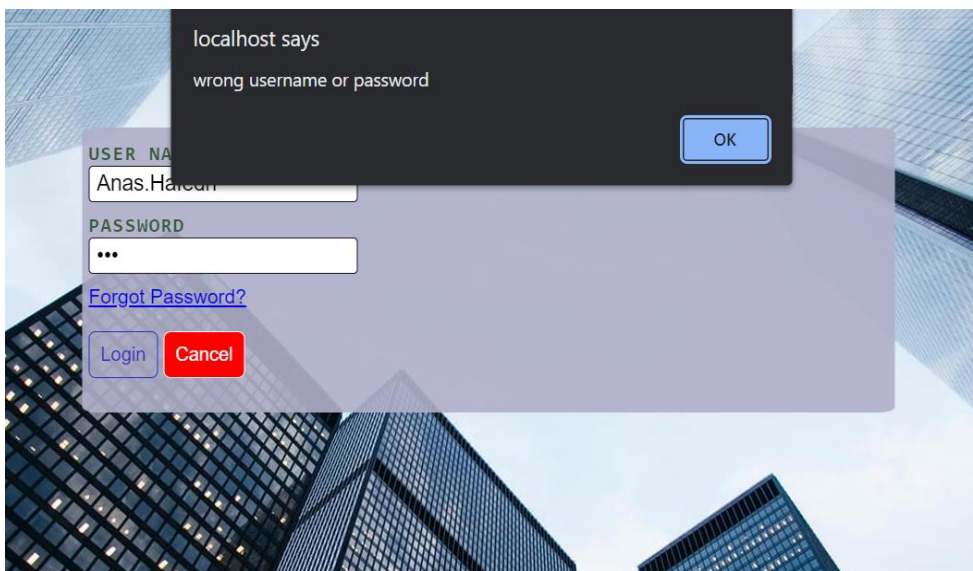
mysqli_close($conn);

?>
```

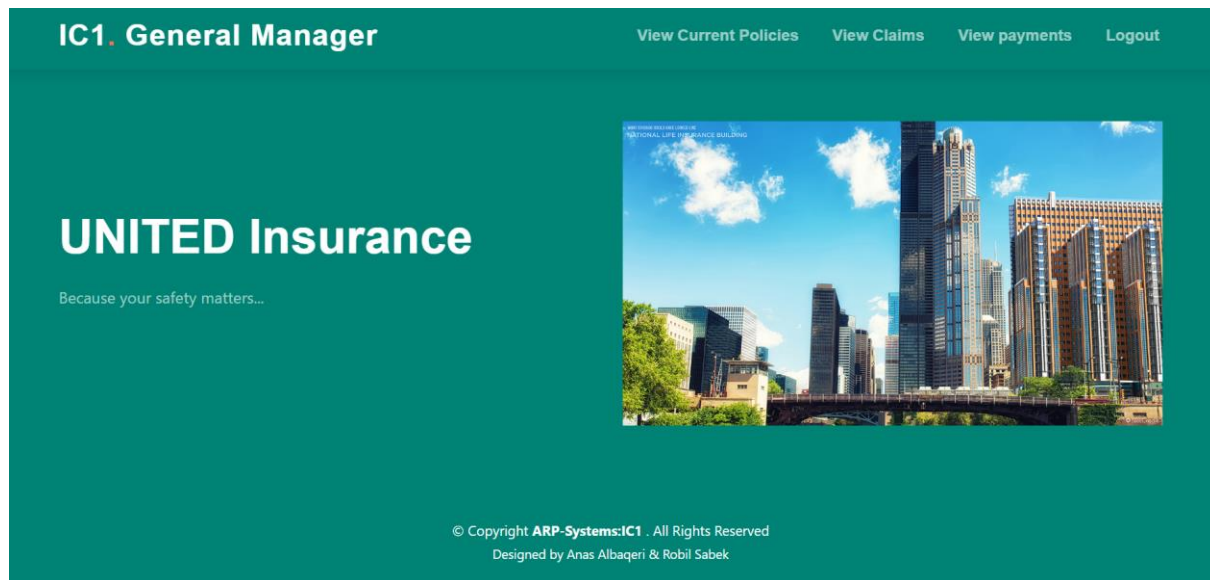
Front End:



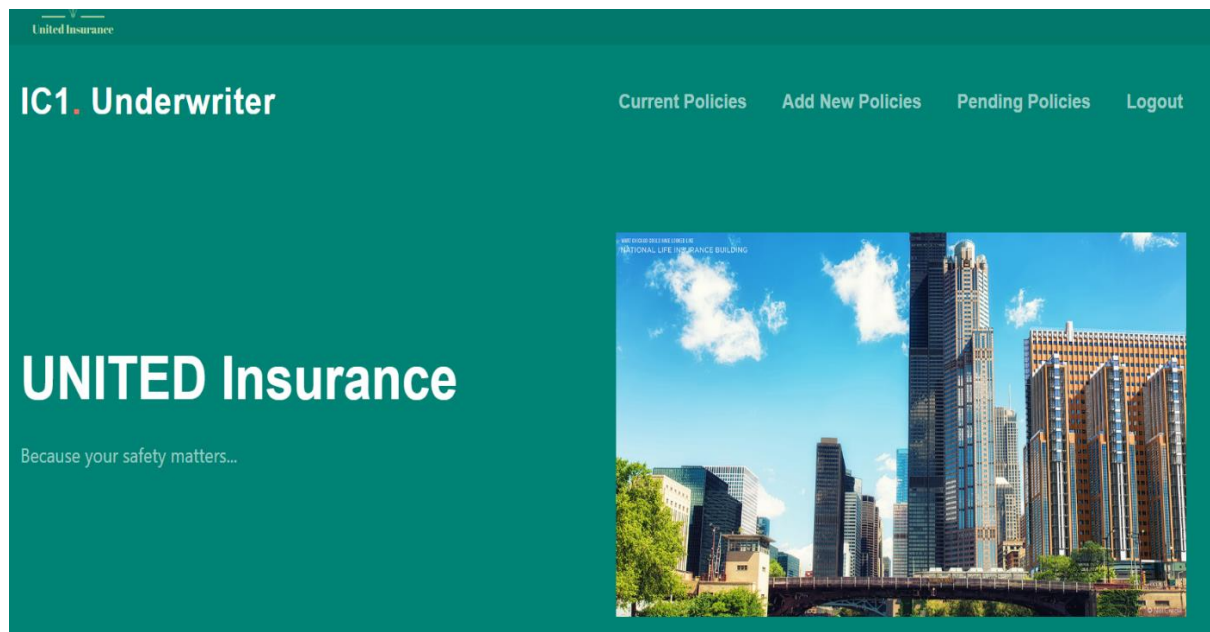
In case of wrong password or username, ajax requests handles the alerts.



In case of correct login, it directs to the main page which looks like this:



For our case, the Underwriter have this interface:



Backend:

```
<?php
//start the session
session_start();
//check to see if the user is logged in, if not redirect to login page
if (!isset($_SESSION['username'])) {
    header('Location:../Index.php');
}
?>

<!DOCTYPE html>
<html lang="en">

<head>

    <title>IC1 - Underwriter Home Page</title>

    <!-- Vendor CSS Files -->
    <link href="../assets/css-2/css-3/css/css4-a.css" rel="stylesheet">
    <link href="../assets/css-2/ic1-icons/ic1-icons.css" rel="stylesheet">

    <!-- Template Main CSS File -->
    <link href="../assets/css/main.css" rel="stylesheet">

</head>

<body>

    <!-- ===== Header ===== -->
    <section id="topbar" class="topbar d-flex align-items-center">
        <div class="container d-flex justify-content-center justify-content-md-between">
            
        </div>
    </section><!-- End Top Bar -->

    <header id="header" class="header d-flex align-items-center">

        <div class="container-fluid container-xl d-flex align-items-center justify-content-between">
            <a class="logo d-flex align-items-center">
                <!-- Uncomment the line below if you also wish to use an image logo -->
                <!--  -->
                <h1>IC1<span>.</span> Underwriter </h1>
            </a>
            <nav id="navbar" class="navbar">
```

```

<nav id="navbar" class="navbar">
  <ul>
    <li><a href="../BE/policies/allpolicies_U.php">Current Policies</a></li>
    <li><a href="../BE/policies/addpolicy.php">Add New Policies</a></li>
    <li><a href="../Be/policies/pendingpolicies_U.php">Pending Policies</a></li>
    <li><a href= "../index.php"> Logout</a></li>
  </ul>
</nav><!-- .navbar -->

<i class="mobile-nav-toggle mobile-nav-show bi bi-list"></i>
<i class="mobile-nav-toggle mobile-nav-hide d-none bi bi-x"></i>

</div>
</header><!-- End Header -->
<!-- End Header -->

<!-- ===== Hero Section ===== -->
<section id="hero" class="hero">
  <div class="container position-relative">
    <div class="row gy-5" data-aos="fade-in">
      <div class="col-lg-6 order-2 order-lg-1 d-flex flex-column justify-content-center text-center text-lg-start">
        <h2>UNITED Insurance </h2>
        <p>Because your safety matters...</p>
      </div>
      <div class="col-lg-6 order-1 order-lg-2">
        
      </div>
    </div>
  </div>
</section>
<!-- End Hero Section -->

<main id="main">

</main><!-- End #main -->

<!-- ===== Footer ===== -->
<footer id="footer" class="footer">

```

Underwriter functionalities

Add a policy

The Underwriter have three main functions, adding a policy , view all policies and pending policies.


Adding policy


IC1. Add Policy

[Go Back](#)

Policy Number:

Holder:

Policy Effective Date: 

Policy Expire Date: 

Total Amount:

This calls the php file that handles send the request and updating the table of policies. All added policies are approved by the manager, so all of new policies are defaulted to pending state.

In the back end, the form is sent to the addpolicy.php which in turn takes the submitted form variables and call addpolicy() function which handles the update of the table.

IC1. Pending Policies

[Go Back](#)

Policy Number	Holder	Effective Date	Expire Date	Total Amount	Status
6	RDF Equipment	2023-05-02	2023-05-02	8000000	Pending

addpolicy.php

```
<?php
session_start();

if (!isset($_SESSION['username'])) {
    header('Location: ../../index.php');
    exit();
}
?>

<html lang="en">

<head>

    <title>IC1 - Add Policy Page</title>

    <link href="../../styles/ic1form.css" rel="stylesheet">

    <!-- Vendor CSS Files -->
    <link href="../../assets/css-2/css-3/css/css4-a.css" rel="stylesheet">

    <!-- Template Main CSS File -->
    <link href="../../assets/css/main.css" rel="stylesheet">
    <link rel="stylesheet" href="../../styles/table.css">

</head>

<body style="background: #008374">

    <!-- ===== Header ===== -->
    <section id="topbar" class="topbar d-flex align-items-center">
        <div class="container d-flex justify-content-center justify-content-md-between">
            
        </div>
    </section><!-- End Top Bar -->

    <header id="header" class="header d-flex align-items-center">

        <div class="container-fluid container-xl d-flex align-items-center justify-content-between">
            <a class="logo d-flex align-items-center">
                <!-- Uncomment the line below if you also wish to use an image logo -->
                <!--  -->
            </a>
        </div>
    </header>

    <div class="table">
```

```

</div>
</header><!-- End Header -->

<div class= "ic1formstyle">
<form method="POST" action="addpolicy.php" ID="login-form">
  <label for="policy_number">Policy Number:</label>
  <input type="text" name="policy_number" required class = "ictextfield">
  <br>
  <label for="holder">Holder:</label>
  <input type="text" name="holder" required class = "ictextfield">
  <br>
  <label for="effective_date">Policy Effective Date:</label>
  <input type="date" name="effective_date" required class = "ictextfield">
  <br>
  <label for="expire_date">Policy Expire Date:</label>
  <input type="date" name="expire_date" required class = "ictextfield">
  <br>
  <label for="total_amount">Total Amount:</label>
  <input type="number" name="total_amount" min="0" step="0.01" required class = "ictextfield">
  <br>

  <button type="submit" class= "mbutton">Add Policy</button>
</form>

</div>
<?php
include '../..../functions/functions.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  // Get form data
  $policy_number = $_POST['policy_number'];
  $holder = $_POST['holder'];
  $effective_date = $_POST['effective_date'];
  $expire_date = $_POST['expire_date'];
  $total_amount = $_POST['total_amount'];
  // Insert new policy into database
  $result = addpolicy($policy_number, $holder, $effective_date, $expire_date, $total_amount);

  if ($result) {
    // Redirect user to policies table to view new policy
    echo '<script>alert("Policy added successfully")</script>';
  } else {
    // Display error message
    echo "<script>alert('Error adding policy')</script>";
  }
}

```

addpolicy(\$policy_number, \$holder, \$effective_date, \$expire_date, \$total_amount);

takes as input the variables that are posted from the added policy that got sent and updates the table accordingly.

```
1 reference
function addpolicy($policyNumber, $holder, $policyEffectiveDate, $policyExpireDate, $totalAmount){
    // Connect to the database
    $conn = mysqli_connect("localhost", "root", "root", "ic1");

    // Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }

    // Prepare the SQL statement
    $sql = "INSERT INTO policies (PolicyNumber, Holder, PolicyEffectiveDate, PolicyExpireDate, TotalAmount)
    VALUES ('$policyNumber', '$holder', '$policyEffectiveDate', '$policyExpireDate', '$totalAmount')";

    // Execute the SQL statement
    if (mysqli_query($conn, $sql)) {
        $result = true;
    } else {
        $result = false;
    }

    // Close the database connection
    mysqli_close($conn);

    return $result;
}
```

Display Company's policies:

When underwriter clicks on current policies



The user gets redirected to the allpolicies.php page that handles the displaying of all policies

allpolicies.php

```
<?php
//start session
session_start();
//check if user is logged in
if (!isset($_SESSION['username'])) {
    header("Location: ../Index.php");
    exit;
}
?>

<!DOCTYPE html>
<html lang="en">
<head>

    <title>IC1 - All claims</title>

    <!-- Vendor CSS Files -->
    <link href="../../assets/css-2/css-3/css/css4-a.css" rel="stylesheet">

    <!-- Template Main CSS File -->
    <link href="../../assets/css/main.css" rel="stylesheet">
    <link rel="stylesheet" href="../../styles/table.css">

</head>
<body style="background: #008374">

    <!-- ===== Header ===== -->
    <section id="topbar" class="topbar d-flex align-items-center">
        <div class="container d-flex justify-content-center justify-content-md-between">
            
        </div>
    </section><!-- End Top Bar -->

    <header id="header" class="header d-flex align-items-center">

        <div class="container-fluid container-xl d-flex align-items-center justify-content-between">
            <a class="logo d-flex align-items-center">
                <!-- Uncomment the line below if you also wish to use an image logo -->
                <!-- 
                <h1>IC1<span>.</span> Current Policies</h1>
            </a>
            <nav id="navbar" class="navbar">
                <ul>
                    <li><a href="../../pages/policymanager.php"> Go Back</a></li>
                </ul>
            </nav><!-- .navbar -->

            <i class="mobile-nav-toggle mobile-nav-show bi bi-list"></i>
            <i class="mobile-nav-toggle mobile-nav-hide d-none bi bi-x"></i>
        </div>
    </header><!-- End Header -->
    <!-- ===== Hero Section ===== -->
    <section id="hero" class="hero">
        <?php
            include '../../functions/functions.php';
            displaypolicies();
        ?>
    </section>

</body>
</html>
```

Inside allpolicies.php main page the displaypolicies() function, which retrieves all policies from the database and displays them in the page as a table.

displaypolicies()

```
1 reference
function addpolicy($policyNumber, $holder, $policyEffectiveDate, $policyExpireDate, $totalAmount){
    // Connect to the database
    $conn = mysqli_connect("localhost", "root", "root", "ic1");

    // Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }

    // Prepare the SQL statement
    $sql = "INSERT INTO policies (PolicyNumber, Holder, PolicyEffectiveDate, PolicyExpireDate, TotalAmount)
    VALUES ('$policyNumber', '$holder', '$policyEffectiveDate', '$policyExpireDate', '$totalAmount')";

    // Execute the SQL statement
    if (mysqli_query($conn, $sql)) {
        $result = true;
    } else {
        $result = false;
    }

    // Close the database connection
    mysqli_close($conn);

    return $result;
}
```

Output:

Note that the new policy that we added are set to pending

IC1. Current Policies					Go Back
Policy Number	Holder	Effective Date	Expire Date	Total Amount	Status
1	BMW Group	2022-05-01	2022-05-01	5000000	Active
2	LAU Health Center	2020-01-01	2023-01-01	12000000	Active
3	Nike Corporation	2019-03-01	2023-03-01	3000000	Expired
4	Airbus	2023-05-01	2023-06-01	3000000	Denied
5	Boeing	2023-05-01	2023-05-03	1200000	Active
6	RDF Equipment	2023-05-02	2023-05-02	8000000	Pending

The underwriter can also view the pending policies:

IC1. Pending Policies						Go Back
Policy Number	Holder	Effective Date	Expire Date	Total Amount	Status	
6	RDF Equipment	2023-05-02	2023-05-02	8000000	Pending	

Policy Manager Functions:

While almost having the access to the same functions of the Underwriter, the policy manager have the ability to approve the pending policies coming form the underwriters of the company

View pending policies as a policy manager:

In addition to the normal table the manager have option to approve pending policies

IC1. Pending Policies							Go Back
Policy Number	Holder	Effective Date	Expire Date	Total Amount	Status	Action	
6	RDF Equipment	2023-05-02	2023-05-02	8000000	Pending	<input type="button" value="Approve"/> <input type="button" value="Submit"/>	

The function that handles the requests backend:

[approve_deny.php](#)

```
<?php
    include '../functions/functions.php';
    if ($_SERVER['REQUEST_METHOD'] === 'POST'){
        $status = $_POST['approve_deny'];
        $policy_number = $_POST['policy_number'];

        if ($status == "Approve"){
            $result = approve($policy_number);
            if ($result == true ){
                echo "<script>alert('Policy approved
successfully!')</script>";
            }
        }
        else{
```

```

        echo "<script>alert('Policy approval failed!')</script>";
    }
}
else{
    $result = deny($policy_number);
    if ($result == true){
        echo "<script>alert('Policy denied successfully!')</script>";
        header("Location: ../../policies/pendingpolicies.php");
    }
    else{
        echo "<script>alert('Policy denial failed!')</script>";
        header("Location: ../../policies/pendingpolicies.php");
    }
}
}
header("Location: pendingpolicies.php");
?>

```

And the php functions are approve and deny defined as follow

Approve(\$policy)

```

function approve($policy_number){
    $conn = mysqli_connect("localhost", "root", "root", "ic1");

    // Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
    // Prepare the SQL statement
    $sql = "UPDATE policies SET Status = 'Active' WHERE PolicyNumber = '$policy_number'";
    // Execute the SQL statement
    if (mysqli_query($conn, $sql)) {
        $result = true;
    } else {
        $result = false;
    }
    // Close the database connection
    mysqli_close($conn);
    return $result;
}

```

Deny(\$policy)

```
function deny($policy_number){
    $conn = mysqli_connect("localhost", "root", "root", "ic1");

    // Check connection
    if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
    }
    // Prepare the SQL statement
    $sql = "UPDATE policies SET Status = 'Denied' WHERE PolicyNumber = '$policy_number'";
    // Execute the SQL statement
    if (mysqli_query($conn, $sql)) {
        $result = true;
    } else {
        $result = false;
    }
    // Close the database connection
    mysqli_close($conn);
    return $result;
}
```

Future Recommendations and Updates:

the insurance industry has undergone significant changes over the years, with advancements in technology and data analytics leading the way. With the adoption of AI and machine learning, insurers can now better predict risk and offer more personalized coverage to their clients. Furthermore, the introduction of a deep learning machine to estimate new policies before approval, taking into account all the risks involved and the insurer's history, will further enhance the industry's ability to deliver effective coverage to customers. As the industry continues to evolve, it is essential that insurers embrace technology and leverage it to create value for their clients while maintaining profitability.

In response to these changing trends, our company is committed to developing a cutting-edge tool that meets the evolving needs of the insurance industry. This tool will utilize AI and machine learning to provide accurate risk assessments, enabling us to offer more tailored coverage to our clients. With this approach, we aim to remain at the forefront of the insurance industry, delivering innovative and effective solutions to our customers. We are excited to continue driving change in this space, and we look forward to the opportunities that lie ahead.

Conclusion:

Overall, the implementation of the IC1 web application insurance system has been successful, with the final product being a functional and user-friendly system that meets the needs of insurance companies. The system was developed using the latest web development technologies, including PHP, HTML, CSS, and JavaScript, and the database used to store the insurance data is MySQL, which provides fast and secure data storage and retrieval.

The login functionality was implemented using a combination of front-end and back-end functions, with the PHP script on the server authenticating the user and checking if the provided credentials are valid. If the login is successful, the system sets session variables and returns a JSON response indicating a successful login along with the user's position within the organization.

The system's main features, such as adding policies, viewing all policies, viewing pending policies, and approving/denying pending policies, were all implemented successfully using PHP functions that interact with the MySQL database to store, retrieve, and update data.

During development, some challenges were faced, such as data validation and security, but these were overcome by implementing appropriate measures, such as data sanitization and input validation, to ensure the system's security and reliability.

Overall, the IC1 web application insurance system is a successful implementation that meets the needs of insurance companies, providing a secure, scalable, and user-friendly platform for managing policies, claims, and customers.