www.ijecs.in
International Journal Of Engineering And Computer Science
Volume 13 Issue 04 April 2024, Page No. 26068-26114
ISSN: 2319-7242 DOI: 10.18535/ijecs/v13i04.4808

Software Engineering Implementation Model on a Tool Rental System

Khaled Ainieh, Anas Muslemani, Ftoon Kedwan

College of Computer and Cyber Sciences, University of Prince Mugrin, Medina, Saudi Arabia

Abstract
This work presents the software development lifecycle model applied on a rental product store system development. What motivates the idea of this work is the growing demand for rental stores that offer tools and other products, and the need for a system to manage and facilitate the rental process. This work aims to showcase an ideal software engineering implementation example. It also aims to meet the needs of the rental market and provide a technological solution to support and organize the operations of rental stores and customers. The study begins by defining and analyzing existing systems and comparing them to the proposed system using a comparison table. The proposed system includes functional and non-functional requirements and use case diagrams with descriptions and activity diagrams for each use case. The design phase includes an overall architecture, component diagram, and class diagram. The candidate solution pattern plan is used to compare various alternative patterns in order to come up with the most effective solution.

Keywords: Software Engineering Implementation, Tool Rental System, software development lifecycle

Introduction
Background of the Project
For this work and other similar software engineering projects (Kedwan F. , 2024), there is a need for an ideal example of a software development lifecycle (SDLC) model to follow. This work invests in the design and analysis of each phase of the SDLC to assure a correct and sufficient project implementation within allocated resources. As an SDLC model showcase, a Tool Rental System (TRS) will be developed and each phase of the development will be thoroughly designed and analyzed, supported with diagrams and tables where needed.

In today's rapidly evolving world where businesses are expanding and technology is playing an increasingly significant role, there is a growing need for advanced systems to support various industries. However, the rental market field has been relatively underserved in terms of technological advancements. Recognizing this gap, the TRS project aims to revolutionize the tool rental industry by providing a state-of-the-art solution that addresses the challenges and limitations faced by rental shop owners.

The developed TRS leverages cutting-edge technologies and industry best practices to offer a comprehensive and efficient tool rental management system. It introduces a new era of organization, convenience, and safety in the rental process, enabling shop owners to seamlessly handle their operations. By implementing the TRS, owners can expect enhanced security measures, simplified rental procedures, streamlined management capabilities, and real-time monitoring of their inventory.

To develop the TRS, extensive research and analysis of the state-of-the-art models of similar systems in the industry were conducted. Lessons learned from existing TRSs, as well as advancements in related fields, were carefully examined to incorporate the latest features and functionalities into the TRS. A major focus was put on understanding the pain points faced by rental shop owners and customers, and how emerging technologies could address those issues.

By utilizing the latest advancements in user experience design, database management, security protocols, and reporting and analytics, the TRS sets a new benchmark in TRS standards. It provides a user-friendly interface for both rental shop owners and customers, ensuring a seamless and intuitive experience. Furthermore, the system's robust security measures, such as data encryption and access controls, guarantee the protection of sensitive information, including customer data and financial records.

The TRS not only serves as a tool rental management system but also acts as a catalyst for business growth. Its comprehensive reporting and analytics capabilities enable owners to gain valuable insights into their rental operations, customer preferences, and overall business performance. Armed with this knowledge, owners can make informed decisions, optimize their inventory, and identify opportunities for expansion and improvement.

In conclusion, the TRS project addresses the technological gap in the tool rental market by offering a modern, secure, and efficient system for rental shop owners. By leveraging state-of-the-art models and incorporating cutting-edge technologies. The TRS empowers owners to overcome challenges, streamline their operations, and provide an exceptional rental experience for their customers. It is a testament to the potential of technology to transform traditional industries and create new opportunities for growth and success.

Objective of the Project

The main objective of this work is to develop an ideal software engineering project as a user-friendly system with intuitive interfaces that ensure an enjoyable and easy user experience. The system aims to address the following key objectives:

- Develop User-Friendly Interfaces: The project focuses on designing and implementing interfaces that are intuitive, visually appealing, and easy to navigate. By prioritizing user-friendliness, we aim to enhance user satisfaction and improve overall usability.
- Rental Process Follow-up Mechanism: The system will incorporate a mechanism to track and manage the rental process effectively. This includes features such as monitoring inventory, managing reservations, tracking rental durations, and handling returns seamlessly.
- Automatic Communication Mechanism: The project aims to establish a streamlined communication channel between the store and the customers. By implementing an automatic communication mechanism, users will receive timely notifications, updates, and reminders regarding their rentals, reservations, and any other relevant information.
- Periodic Income Reports: The system will generate periodic reports that provide insights into the income generated by the store. These reports will help the management in analyzing revenue trends, identifying popular rental items, and making informed business decisions.

Scope of the Project

The project will cover the major aspects of a desktop system for almost any rental shops that offer tools, equipment, products which includes the main following functionalities:

1. Store Tool Panel: give the user full control over the tools in the store (add, update, delete, status of the product, type, price).
2. Searching Tool: the user can search for specific tools. User can also search for availability, number of tools and what is the soonest possible time to get the tool if it is unavailable.
3. Customer Management: The system provides customer's information, then system can generate a form that is printable as a contract.
4. Periodic Report generation: Weekly and monthly periodic reports are printed to report the number and value of the leased tools.
5. Notification Management: a reminder message is automatically sent to customers shortly before the end of the lease term to remind them to return the tools. In addition, the system should alert the user (the owner) in case the customer is exceeded the agreed upon date to communicate with him.

6. Tools Follow-Up Management: All tools currently rented and the information of customers who rented them are shown with the return date.

Deliverable

The expected outcomes for this rental shop desktop system are as follows:

- ✓ Desktop system development using Java Programing language.
- ✓ A relational database, which is developed by MySQL.
- ✓ Graphical User Interface (GUI), which is developed by JavaFX.
- ✓ All architectures and diagrams related to the system are as follows:
    - Use case diagrams.
    - Class diagram.
    - Sequence diagrams.
    - Activity diagrams.
    - Component diagram.
    - Architectural diagram.

Literature Review

After the spread of shops for renting tools and various products, there is a report by Verdon (Verdon, 2019) found that 57.3% of all consumers are willing to rent, rather than buy, if the products are "well-made and trendy." For millennial and Generation Z consumers, the percentage was over 70% (Verdon, 2019).

The automation of the basic processes for managing these stores began to spread as well, and several products and systems appeared that served this goal. In this literature review, we will try to contrast some of the previous systems of similar stores. Then, some useful services provided will be chosen, and the previous mistakes of those systems will be avoided. The goal is to build an efficient and a better system than already built ones.

Two systems that serve the type of stores with interest in renting products were chosen, Headmaster and Sharefox. Headmaster is a program that supports equipment rental companies and stores, and it is available in Arabic language. This program is made by the Arab Software Company (Headmaster, 2010).

The Sharefox rental software comprises an online store that you can use as part of a website or as a standalone web shop with help pages and a payment solution. This program is optimized for online bookings (Sharefox, 2021).

Other similar systems include RentMaster and EZRentOut. RentMaster is a comprehensive equipment rental software designed to streamline rental operations for businesses. It offers a range of features and functionalities to effectively manage customer relationships, track inventory, create rental contracts, generate invoices, and provide reporting capabilities. (SOLUTIONS, A RentMaster, 2023)

EZRentOut is a cloud-based equipment rental software designed to simplify and streamline rental operations for businesses. It offers a range of features and functionalities to efficiently manage customers, track inventory, create rental contracts, generate invoices, and provide reporting capabilities. (EZRentOut, 2021)

This proposed TRS system aims to organize, facilitate, and manage this process in a safe and effortless way. Using this system by the owners of tools rental stores, the rental process becomes safer, easier, more streamlined, manageable, and monitorable.

Table of Comparison

Tables 1 and 2 in the appendix (appendices 1 and 2 consecutively) contain a comparison between the aforementioned four systems compared to the proposed TRS system. The table shows the similar features and the differences between them, and the unique features in each one. It also includes some justifications of missed features in the TRS system and future plans.

The Proposed TRS System

System Specification

The TRS system allows the admin user (owner) to add, modify and delete the tools they want to rent within the system database. In addition, the system allows following up with the renters on the leased tools. The following up involves the names of customers and other information. The follow up tracks the dates of tools' retrieval, sends a reminder message to customers, and alerts the owner when the deadline is exceeded as a unique service that the TRS system provides. It also organizes the entire rental process and provides the ability to search for tools and their condition. The following section discusses the TRS system functional and nonfunctional requirements in Tables 3-7.

Functional Requirements

Admin(manager) needs

Table 3: Functional admin requirements table

Identifier	*Requirements*
FR-1	*The admin shall be able to manage products (add, delete, adjust).*
FR-2	*The admin shall be able to manage customers information (add, delete, adjust).*
FR-3	*The admin shall be able to generate a contract with the customer.*
FR-4	*The admin shall be able to track and check leased products (product name, return date, customer info).*
FR-5	*The admin shall be able to manage users (add, delete, adjust).*
FR-6	*The admin shall be able to generate weekly / monthly report.*
FR-7	*The admin should be able to search for a specific product.*

User(employee) needs

Table 4: Functional user requirements table

Identifier	Type	*Requirements*
NFR-1	*Usability*	The user should be able to use the system in their job after 7 days of training.
NFR-2	*Usability*	The friendly GUI of the system should get more than 70% of the users satisfied with the system using the survey before deployment.
NFR-3	*Security*	The system shall ensure that sensitive data, such as passwords, are encrypted when stored in the database.
NFR-4	*Security*	Users access is authenticated using log in form.

Customer needs

Table 5: Functional customer requirements table

Identifier	*Requirements*
FR-1	*The customer shall be able to get a notification message (SMS) as reminder of retrieve date.*
FR-2	*The customer shall be able to get a generated detailed hard copy contract.*

System needs

Table 6: Functional admin requirements table

Identifier	*Requirements*
FR-1	*The system shall be able to send notification messages for users and customers using SMS.*
FR-2	*System shall be connected to a database.*

Non-Functional Requirements

Identifier	*Requirements*
FR-1	*The user shall be able to manage customers information (add, delete, modify).*

FR-2	*The user shall be able to generate a printed contract to the customer.*
FR-3	*The user shall be able to track and check* leased *products (product name, return date, customer info).*
FR-4	*The user should be able to search for a specific product.*
FR-5	*The user shall be able to manage products (add, delete, modify).*

Table 7: Non-functional requirements of the system table

System Analysis

This section demonstrates a use case diagram which describes the high-level functionalities and scope of the system. The use case description shows how users will perform tasks and how the system behaves as it responds to a request (Usability.gov, 2015). Figure 1 (Appendix 3) illustrates the interactions between the system and its actors which can be individuals, other systems, or external entities. The use case diagram provides an overview of the system's behavior and the different ways in which users can interact with it to achieve specific goals *Table 7: Non-functional requirements of the system table* or perform certain tasks.

Tables 8-14 (Appendices 4-10) describe each use case. They provide a description of the steps and conditions required to perform every specific function in the system, as well as how the system interacts with the user.

Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram that shows the flow of activities or actions within a system. It is used to represent the behavior of a system by modeling the flow of control from one activity to another and to refer to the steps involved in the execution of each use case (simmytarika5, 2022). Each use case is represented by an activity diagram where it demonstrates the logic of an algorithm, and describes the steps performed in a UML use case and illustrates a business process or workflow between users and the system.

The activity diagrams in Figures 2-8 (Appendices 11-17) provide a visual representation of the various activities and processes within the system. They outline the steps involved in activities such as adding, modifying, and deleting products, user, customer and creating contracts, generating reports, and sending notifications. These diagrams illustrate the flow of activities, decision points, and any conditions or loops that may occur during each process. They serve as a valuable tool for understanding the sequence of actions and the interactions between different components, allowing for effective analysis and communication of the system's functionalities.

Sequence Diagram

A sequence diagram is a type of UML (Unified Modeling Language) diagram that shows the interactions between objects or components in a system over time. Sequence diagrams in Figures 9-12 (Appendices 18-21) represent the behavior of the TRS system by modeling the flow of messages between the system objects. The sequence diagrams show the order in which these messages are sent and received, along with the lifecycle of objects and their interactions. Sequence diagrams are useful for modeling the interactions between objects in a system and for understanding the flow of control.

System Design

One of the foundations of software engineering and design is to find the best design and pattern that serves the requirements of the program. This step saves time, effort, and the cost of modifications and improvements to the project in its last stages. It also makes way for communications between all the stakeholders (user-side, customer, management, etc.) Also, this step draws the outline of the implementation stage for developers and prevents developers from making incorrect decisions. Therefore, an appropriate design is built in Figure 13 considering the problem's requirements. Software design is developed and improved step by step to reach the optimal design.

Architecture Patterns Table

Table 15 presents a comparison of various design patterns commonly used in software development. Each design pattern has its own distinct characteristics and benefits, making them suitable choices for different types of projects. By analyzing and evaluating these patterns, developers can determine which ones are suitable candidates for their specific software development needs.

Design patterns serve as proven solutions to recurring problems in software design. They provide a blueprint or template that guides developers in structuring their code and organizing their system architecture.

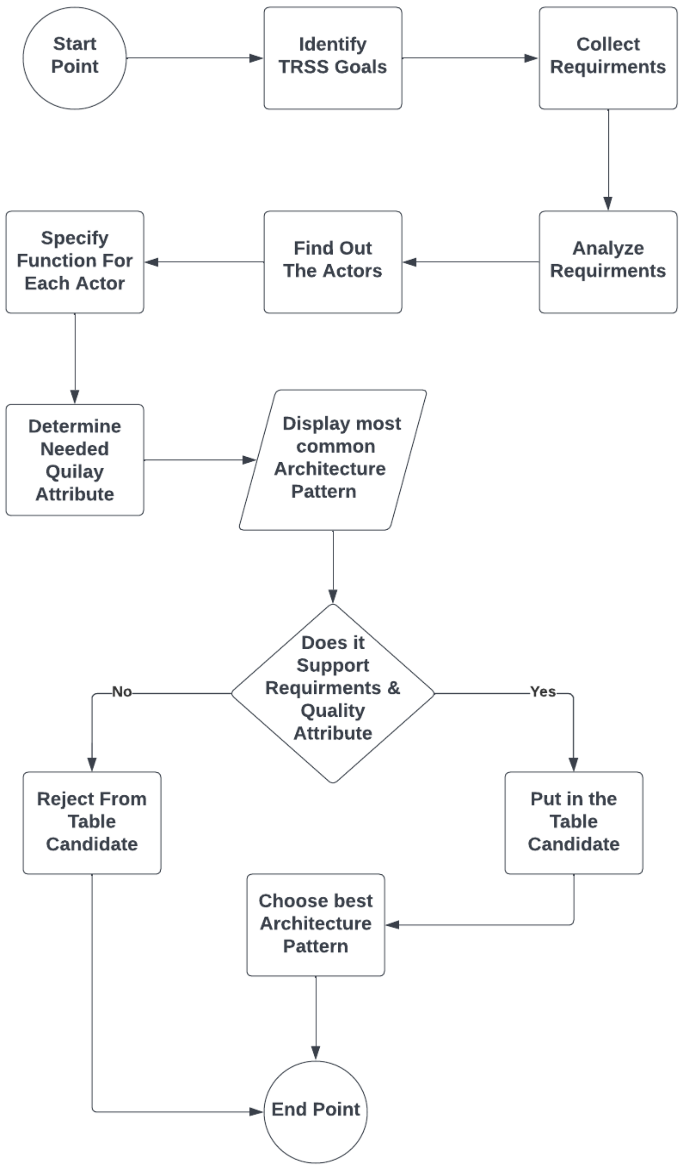

Start Point
Identify TRSS Goals
Collect Requirments
Analyze Requirments
Find Out The Actors
Specify Function For Each Actor
Determine Needed Quilay Attribute
Display most common Architecture Pattern
Does it Support Requirments & Quality Attribute
No
Yes
Reject From Table Candidate
Put in the Table Candidate
Choose best Architecture Pattern
End Point

Figure 1: Steps for finding the best architecture pattern

The table represents a comparison of different design patterns, including Client-Server, Multi-tiers, Peer-to-peer, Model-View-Controller (MVC), Blackboard, Layered, Pipe-and-filter, and Repository. Through this comparison, some design patterns may emerge as candidates for adoption, while others may be deemed less suitable for the specific software development project, so they are rejected.

OVERALL ARCHITECTURE DESIGN

After the identification of the candidate and possible architecture designs (Multi-tier, Layered, MVC) for the application of this system, we excluded and rejected the least worthy to reach the most appropriate design.

Table 15 Candidate/Reject Most Common Architecture Patterns Table

	Architecture Pattern	*Decision*	*Justification*
1	***Client-Server***	***Reject***	It requires connection with any web server since it's a desktop application.
2	***Multi-tiers***	***Candidate***	It increases system's improved scalability, security, and data integrity.
3	***Peer-to-peer***	***Reject***	Since the system will not initiate interaction with customers
4	***MVC***	***Candidate***	It promotes separation of concerns by dividing the application logic into three distinct components: Model, View, and Controller. This separation enhances maintainability and code readability.
5	***Blackboard***	***Reject***	The system does not require artificial intelligence approach which handles complex problems.
6	***Layered***	***Candidate***	Layers enable logical separation of kinds of components and provide structure for communication between components of systems.
7	***Pipe-and-filter***	***Reject***	The system is not completely automated and needs a lot of decisions and manual work.
8	***Repository***	***Reject***	TRS system does not share data (like order and configurations) between different customers, but it needs the separation between each order. Hence, the system does not ask for central shared database for all components.

After that, the candidate patterns were investigated. The multi-tier and layered architecture were then excluded, and the MVC architecture was preferred. The MVC supports the logical separation of functionality and enables code reusability and flexibility, allowing for easy adaptation to different user interfaces and future changes. While layered architecture could be deployed on the same physical device or unit (IBM_Cloud_Education, 2022) which suites the current project.

Since tier architecture is all about the physical deployment where system spread in different machines so finally, the MVC architecture was selected for the overall architecture. The chosen architecture shows that the MVC pattern is one of the most used patterns in all of software engineering projects (Len Bass, Paul Clements, Rick Kazman, 2012).

The MVC design pattern depicted in Figure 14 is a suitable choice for the RTS system project. MVC is widely used in web applications and desktop applications and provides a structured and organized approach to designing and implementing user interfaces.

1. Separation of Concerns: MVC promotes a clear separation of concerns between different components of the system. The Model represents the data and business logic, the View handles the presentation and user interface, and the Controller manages the interaction between the Model and the View. This separation allows for better code organization, maintainability, and scalability.
2. Code Reusability: MVC facilitates code reusability by allowing different components to be developed independently. The Model can be reused across multiple views, and views can be created or modified without affecting the underlying data and logic. This promotes modularity and makes it easier to maintain and extend the application in the future.
3. User Interface Flexibility: With MVC, the View component is responsible for rendering the user interface. This allows for flexibility in terms of supporting different presentation layers and user experiences. For example, you can have multiple views for different user roles or device types, all interacting with the same underlying Model and Controller.

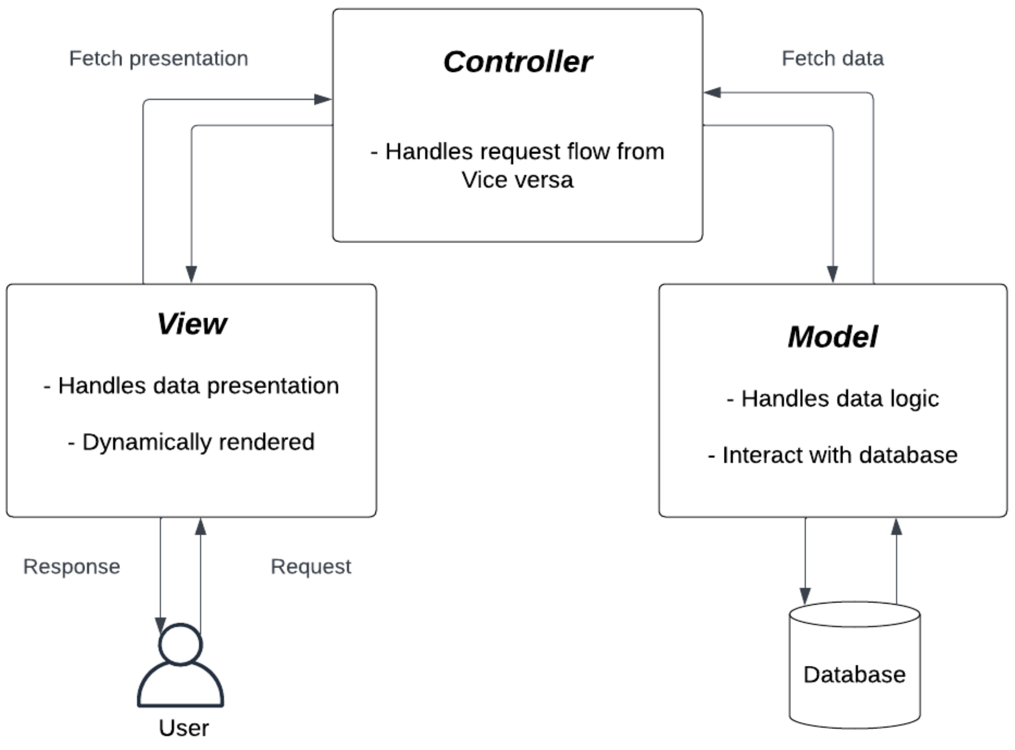


Figure 2: MVC architecture pattern of the system

Component Diagram

The component diagram's main purpose is to show the structural relationships between the components of a system (Bell, 2004). Component diagram provides a visual representation of the components that put together up to build a system, as well as the relationships between those components. This can be useful for understanding the architecture of a system. The component diagram in Figure 15 was generated by drawing a manager for each use case in the use case diagram. After reading and analyzing the requirements, the appropriate relationships between the components were generated indicating and which component requests data (consumer) and which one provides data(provider) using an interface.

Class diagram

Class diagram is used to represent classes, interfaces, inheritance, collaboration, and association relationships between the classes of the system. In addition to capturing classes, interfaces, inheritance, collaboration, and association relationships, the class diagram serves as a visual tool for understanding the overall architecture and structure of the system. It helps in identifying the key entities and their interactions, enabling developers to comprehend the system's functionality at a higher level. The class diagram also aids in identifying potential design flaws or missing components, allowing for timely adjustments and improvements. Moreover, it facilitates communication and collaboration among project stakeholders, as it provides a common language and reference point for discussing system design and implementation. Overall, the class diagram plays a crucial role in the software development process, providing a foundation for building robust and well-structured systems.

The class diagram in Figure 16 represents the final and approved version of the system's design. It has undergone thorough review and refinement to ensure its accuracy and alignment with the project requirements. The diagram depicts the main classes and their relationships, capturing the essential components and interactions within the system. The attributes and methods included in each class have been carefully selected and validated to support the system's functionality effectively. This class diagram serves as a reliable blueprint for the implementation phase, guiding developers in building the system according to the specified design specifications.

Since the design patterns are typical solutions to commonly occurring problems in software design (Refactoring.Guru., n.d.), so patterns were used as a solution for the current system.

There are some differences in the association signs from a book to another and between different websites (Nishadha, 2022), so the focus was specifically on the relation composition and the aggregation so following the standards in the adopted "creately" website. This website provides a very common Visual Workspace for UML style.

The website differentiates between aggregation, where the contained classes are not strongly dependent on the lifecycle of the container, and the composition which is contained class that will be destroyed when the container class is destroyed (Nishadha, 2022).

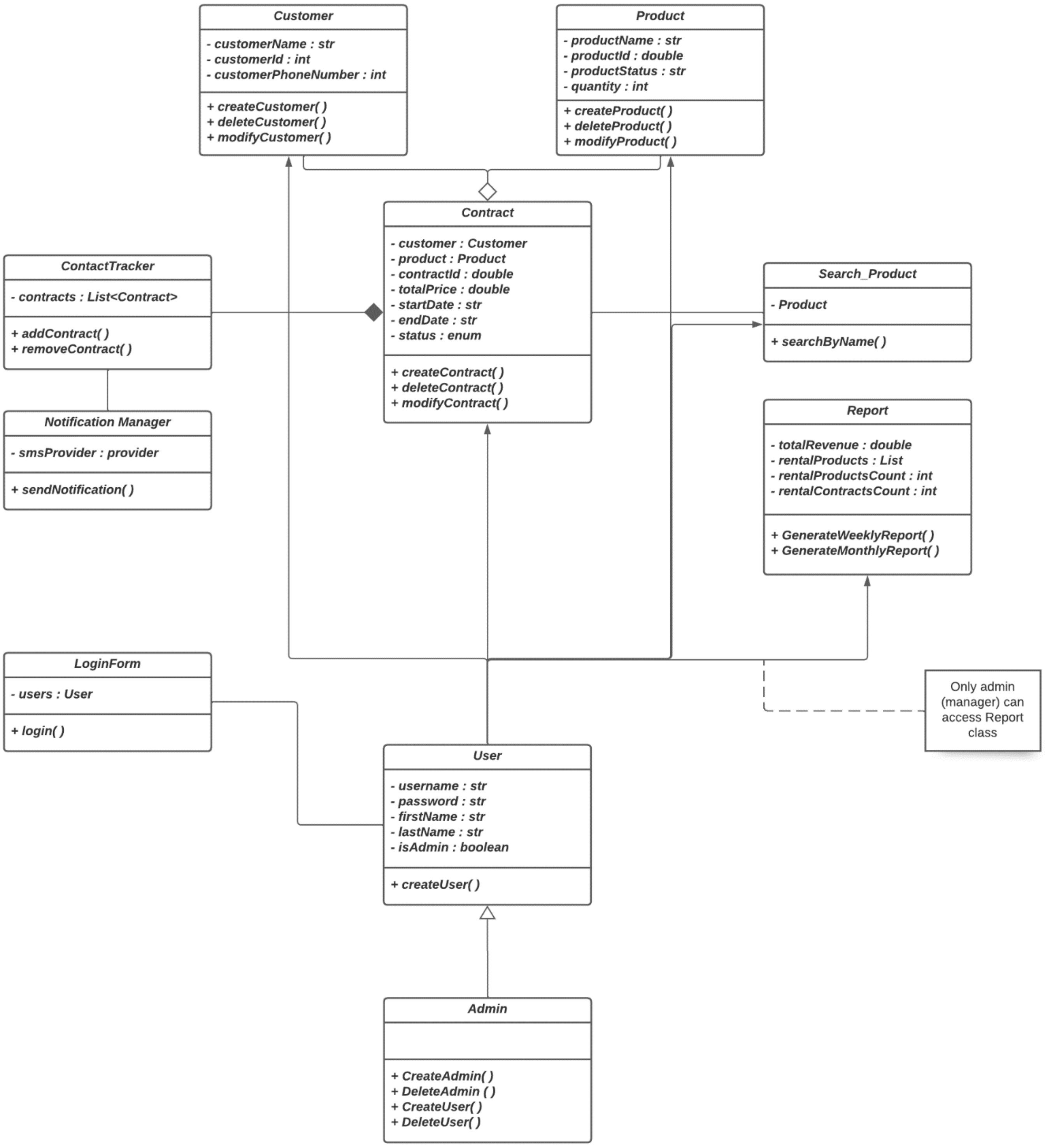


Figure 4: TRS Class diagram

Project Implementation

Project Implementation Overview

The implementation phase of the RTS system project marks a significant milestone in the development process, bringing the system's functionalities to life. During this phase, various windows and interfaces have been designed and developed to provide users with a seamless and intuitive experience. These windows encompass essential modules such as login, home, product management, customer management, contract creation, report generation, and user management. Each window serves a specific purpose, allowing users to perform tasks related to their roles and responsibilities within the RTS system. The system incorporates features like data retrieval from the database, input validation, and user-friendly interfaces to ensure efficiency, accuracy, and user satisfaction. This section will provide an in-depth exploration of each window, including screenshots, detailed descriptions, and functionality highlights, to offer a comprehensive understanding of the implemented RTS system.

Technology Stack:

The RTS system has been implemented using the following technology stack:

- Java Programming Language: Java was chosen as the primary programming language for its robustness, platform independence, and extensive ecosystem of libraries and frameworks.
- JavaFX Framework: JavaFX was utilized as the framework for building the graphical user interfaces (GUIs) of the system. It provides a rich set of UI components, event handling mechanisms, and layout management capabilities.
- MySQL Database Management System (via phpMyAdmin): MySQL, a popular relational database management system, was employed to store and manage system data. phpMyAdmin, a web-based administration tool, was used to facilitate database management tasks.
- XAMPP Development Environment: XAMPP (Cross-Platform, Apache, MySQL, PHP, and Perl) was utilized as a development environment. It combines the Apache web server, MySQL database server, and PHP interpreter, providing a comprehensive platform for web application development and testing. XAMPP was chosen from many alternative software development platforms (Kedwan F. H., 2019).
- Scene Builder UI Design Tool: Scene Builder, a visual layout tool, was used to design and create graphical user interfaces for the RTS system. It offers a drag-and-drop interface, simplifying the process of building complex UI layouts.
- CSS (Cascading Style Sheets) Styling: CSS was employed for styling the user interfaces, allowing for customization of colors, fonts, layouts, and other visual aspects of the system.
- Libraries:
  - javax.mail: The javax.mail library was used for email functionality, enabling the system to send email notifications and communications to users.
  - MySQL Connector/J: The MySQL Connector/J library provided the necessary JDBC driver for establishing the connection between the Java application and the MySQL database.

By leveraging Java as the programming language, JavaFX as the GUI framework, MySQL via phpMyAdmin for database management, XAMPP as the development environment, Scene Builder for UI design, and incorporating CSS for styling, along with the javax.mail and MySQL Connector/J libraries, the RTS system implementation benefited from a comprehensive technology stack.

Database Schema:

Tables 16-20 (Appendices 22-26) represent the RTS system database schema. The relationships between the tables in the RTS system are explained hereafter:

- One-to-Many Relationship: users table to contract table. The users table has a primary key (id). The contract table has a foreign key (user_id) referencing the users table's primary key. This relationship indicates that one user can have multiple contracts, but each contract belongs to only one user.
- One-to-Many Relationship: customer table to contract table. The customer table has a primary key (id). The contract table has a foreign key (customer_id) referencing the customer table's primary key. This relationship signifies that one customer can have multiple contracts, but each contract is associated with only one customer.

- Many-to-Many Relationship: contract table to product table. The contract table has a primary key (id). The contract product table serves as a junction or associative table. The contract product table has foreign keys (contract_id and product_id) referencing the contract table and product table's primary keys, respectively. This relationship allows multiple products to be rented in a contract, and a product can be rented in multiple contracts.

In summary, the users table and customer table have one-to-many relationships with the contract table, while the contract table has a many-to-many relationship with the product table through the contract product table.

System Windows:
TRS is a comprehensive software solution designed specifically for tool rental businesses. It provides a user-friendly interface and a range of features to streamline operations and enhance efficiency. Here is an overview of the main windows/screens in the TRS system:

RTS system Login page Description:
The login page, Figure 17 (Appendix 27) serves as the entry point for authorized users to access the RTS system. The login page showcases a visually appealing and professional design. At the top-left corner, a distinctive and tastefully designed logo/icon representing the RTS system prominently displayed. The logo/icon adds a touch of branding and enhances the system's visual identity.

As shown in Figure 17, the login page presents a clean and intuitive user interface, designed to provide a seamless login experience. The page features a prominently placed login form, consisting of fields for username and password input. The simple yet elegant design of the login page sets the tone for the user-friendly nature of the RTS system.

The main methods in the Log in component (class) are as follows:
1. **isLogin(Admin ad)**: Checks whether the provided **Admin** object's username and password match an entry in the "user" table in the database. It executes an SQL query to retrieve the user with a matching username and password and returns **true** if a result is found, indicating a successful login.
2. **getUserRole(String username)**: Retrieves the role of a user from the "user" table in the database based on the provided username. It executes an SQL query to retrieve the user's role and returns the role as a **String**. If the user role is not found, it returns **null** (which can be handled accordingly).

These methods handle database operations related to user login, authentication, and role retrieval.

RTS system Home page Description:
Figure 18 (Appendix 28) showcases the home page of the RTS system, offering a centralized hub for managing various aspects of the rental process. The page design demonstrates a thoughtful arrangement of buttons and informative elements, contributing to an efficient user experience.

At the center of the page, attention is drawn to a visually appealing display that highlights the system's advantages. Captivating shapes and concise descriptive text emphasize the system's key benefits, such as efficiency, quickness, and productivity. This feature aims to convey the system's value proposition and capture users' attention.

Moving towards the navigation section, the home page prominently features several buttons that enable users to perform essential actions. The "Product" button grants access to tools and resources related to rental items. The "User" button facilitates management of user accounts and permissions. The "Customer" button supports efficient handling of customer information. The "Contract" button serves as a gateway for creating and managing rental agreements and associated documentation of the rental process. The "Notifier" button enables users to set up notifications and reminders. The "Report" button provides a comprehensive view of monthly activities and enables users to explore valuable insights and information. Lastly, the "Log Out" button ensures secure logout from the system, maintaining data confidentiality and user privacy.

The Home window serves as the main dashboard for regular users within the RTS system, providing them with convenient access to key functionalities. This window allows regular users to manage products, customers, contracts, notifications, and perform the necessary actions within their assigned responsibilities.

RTS system Product Management Window Description:
Figure 19 (Appendix 29) illustrates the Product Management Window of the RTS system, offering a comprehensive view of the stored products and efficient tools for managing them.
The central focus of the window is a well-organized table that displays essential product information. The table consists of columns such as "Product ID," "Name," "Category" (or model), "Price," and "Status." This tabular representation enables users to quickly grasp details about the products stored within the system. The "Product ID" column provides a unique identifier for each item, while the "Name" column specifies the product's name or title. The "Category" column identifies the product's model or category, facilitating easy categorization and retrieval. The "Price" column displays the cost associated with each product, aiding in pricing and financial management. Lastly, the "Status" column indicates the availability of the product, distinguishing between "Available," "Rental," or "Faulty" states.

In the lower-left corner of the window, three buttons— "Add," "Modify," and "Delete" — provide convenient options for managing the product inventory. The "Add" button enables users to input new product details into the system, facilitating the addition of new items. The "Modify" button allows users to update and edit existing product information, ensuring accurate and up-to-date records. The "Delete" button grants the ability to remove products that are no longer in use or have become obsolete.

At the top of the window, a prominently positioned search field allows users to search for specific products by their names. This search functionality streamlines the process of finding products, enhancing productivity, and enabling quick access to the desired items.
The window also includes a button in the top-left corner, providing easy navigation back to the home page. This convenient button ensures smooth transitions between different modules or functionalities of the system, promoting a user-friendly experience.

Here are the main methods used in the product component(class):
1. **insert(ModelProduct product)**: Inserts a new product into the "product" table in the database. It prepares an SQL INSERT statement with the product details (name, category, price, and status) and executes it to add the product to the database.
2. **modify(ModelProduct product)**: Modifies an existing product in the "product" table based on the selected Id value. It prepares an SQL UPDATE statement with the updated product details and executes it to update the corresponding entry in the database.
3. **delete(int id)**: Deletes a product from the "product" table based on the selectedId value. It prepares an SQL DELETE statement and executes it to remove the product from the database.
4. **getAllProduct()**: Retrieves all products from the "product" table. It executes an SQL SELECT statement and retrieves the product data from the result set. Then, it creates **ModelProduct** objects and populates them with the retrieved data. The objects are added to an **ObservableList** and returned.
5. **searchProduct(String nameS)**: Searches for products in the "product" table based on a provided name pattern. It prepares an SQL SELECT statement with a LIKE condition to match the product names containing the specified pattern. It retrieves the matching products from the result set, creates **ModelProduct** objects, and adds them to an **ObservableList**. The list of matching products is returned.

These methods handle the database operations related to managing products, such as inserting, modifying, deleting, retrieving all products, and searching for products based on name.

RTS system Customer Management Window Description:
Figure 20 (Appendix 30) showcases the Customer Management Window of the RTS system, providing a user-friendly interface for efficient management of customer information. The window prominently displays a range of customer details, allowing for comprehensive tracking and administration. The customer information fields include the automatically generated "Customer ID," which serves as a unique identifier

for each customer. The "Customer Name" field captures the name of the customer, facilitating easy identification. The "Customer NIC" (National Identity Card) field stores the national identification number associated with the customer, ensuring accurate record-keeping. The "Customer Address" field captures the email address of the customer, facilitating communication. Finally, the "Customer Phone" field stores the contact number of the customer, enabling efficient customer outreach.

Similar to previous windows, a search field is positioned at the top of the window, enabling users to search for specific customers by their names or other relevant criteria. This search functionality simplifies the process of locating and accessing customer records, enhancing productivity.

The window also features a button in the top-left corner, providing easy navigation back to the home page, allowing users to switch between different modules or functionalities effortlessly.
Located in the lower-left corner of the window, intuitive buttons— "Add," "Modify," and "Delete" — offer convenient options for managing customer information. The "Add" button facilitates the addition of new customer details into the system, ensuring seamless integration of new customer records. The "Modify" button allows users to update and edit existing customer information, ensuring accurate and up-to-date records. The "Delete" button enables the removal of customer records when necessary.

Here are the main methods in the customer component(class):

1. **insert(ModelCustomer customer)**: Inserts a new customer into the "customer" table in the database, using the provided customer model object.
2. **modify(ModelCustomer customer)**: Modifies an existing customer in the "customer" table in the database, using the provided customer model object and the selected customer ID.
3. **delete(int id)**: Deletes a customer from the "customer" table in the database, based on the provided customer ID.
4. **getAllCustomers()**: Retrieves all customers from the "customer" table in the database and returns them as an observable list of **ModelCustomer** objects.
5. **searchCustomer(String nameS)**: Searches for customers in the "customer" table in the database based on a provided name string and returns the matching customers as an observable list of **ModelCustomer** objects.

These methods handle various database operations related to customers, such as inserting, modifying, deleting, retrieving all customers, and searching for customers based on their names.

RTS system User Management Window Description:
Figure 21 (Appendix 31) showcases the User Management Window of the RTS system, providing an interface for managing user accounts and their associated details. The window features a table that displays user information in an organized manner. The columns in the table include "ID," "Username," "First Name," "Last Name," "Email," and "Role." The "ID" column serves as a unique identifier for each user, facilitating easy identification and reference. The "Username" column displays the username chosen by each user during account creation. The "First Name" and "Last Name" columns present the respective first and last names of the users. The "Email" column shows the email address associated with each user account. Finally, the "Role" column indicates whether a user has an "Admin" or "Regular User" role, distinguishing between different levels of access and permissions within the system.

Here is the main method of the user component (class):

1. **insert(ModelUser user)**: Inserts a new user into the database. It prepares an SQL INSERT statement to add the user's information (username, first name, last name, email, and role) to the "user" table.
2. **modify(ModelUser user)**: Updates an existing user in the database. It prepares an SQL UPDATE statement to modify the user's information based on the provided **ModelUser** object. The update is performed based on the user's ID (**selectedId**).
3. **delete(int id)**: Deletes a user from the database. It prepares an SQL DELETE statement to remove the user from the "user" table based on the provided user ID.

4. **getAllUser()**: Retrieves all users from the database. It executes an SQL SELECT statement to fetch all rows from the "user" table. The retrieved data is used to populate **ModelUser** objects, which are added to an observable list (**userList**). The observable list is then returned.
5. **searchProduct(String nameS)**: Searches for users in the database based on a provided username pattern. It executes an SQL SELECT statement with a LIKE condition to find users whose username matches the pattern. The retrieved data is used to populate **ModelUser** objects, which are added to an observable list (**userList**). The observable list is then returned.

These methods provide functionality to insert, update, delete, retrieve, and search for user information in the database. It allows you to perform CRUD (Create, Read, Update, Delete) operations on user data within your application.

RTS system Report Window Description:
Figure 22 (Appendix 32) showcases the Report Window of the RTS system, providing a user-friendly interface for generating comprehensive reports. The window includes a field where users can enter the desired month number to retrieve information related to that specific month. Once the month number is entered, users can click on the "Refresh" button located at the top-right corner of the window. This action triggers the retrieval of data from the database, populating various fields with the relevant information.

Upon refreshing, the window displays important data based on the specified month. Fields such as "Total Contracts for this Month" show the total count of contracts created during the selected month, providing an overview of rental activity. The field "New Customers Added this Month" indicates the number of new customers who have been added to the system within the specified month, facilitating tracking of customer acquisition. The "Total Revenue for the Month" field presents the cumulative revenue generated from all contracts during the selected month, offering a snapshot of financial performance. The "Total Products Rented this Month" field showcases the total count of products rented during the selected month, highlighting the utilization of rental inventory.

Furthermore, the window provides insights into the "Top 5 Most Popular Products Rented this Month," identifying the products that have been rented most frequently during the specified month. This information aids in understanding customer preferences and assists in making informed decisions regarding inventory management. For easy navigation, a "Back Home" button is conveniently placed in the top-left corner of the window, allowing users to return to the home page with a single click.

Here are the main methods used in report component(class):
1. **connectToDatabase()**: Establishes a connection to the database using the **ConnectionDB** class.
2. **getContractCountForMonth(String month)**: Retrieves the count of contracts for a given month. It executes an SQL query that counts the number of rows in the "contract" table where the month of the date matches the specified month. The count is returned as an integer.
3. **getCustomerCountForMonth(String month)**: Retrieves the count of customers for a given month. It executes an SQL query that counts the number of rows in the "customer" table where the month of the date matches the specified month. The count is returned as an integer.
4. **getTotalRevenueForMonth(String month)**: Retrieves the total revenue for a given month. It executes an SQL query that calculates the sum of the "total_cost" column in the "contract" table where the month of the date matches the specified month. The total revenue is returned as a double.
5. **getProductCountForMonth(String month)**: Retrieves the count of products rented for a given month. It executes an SQL query that counts the number of rows in the "contractproduct" table where the month of the rental_date matches the specified month. The count is returned as an integer.
6. **getTopRentedProductsForMonth(String month)**: Retrieves the top 3 rented products for a given month. It executes an SQL query that joins the "contractproduct" and "product" tables, groups the results by product name.

These methods provide functionality to generate reports based on contract counts, customer counts, total revenue, product counts, and top rented products for a specific month.

RTS system Contract Management Window Description:
Figure 23 (Appendix 33) displays the Contract Management Window of the RTS system, providing a comprehensive interface for creating rental contracts and managing associated details. The top part of the window consists of input fields to capture essential contract information. Users can input the desired date for the contract, while the "Customer Name" dropdown allows for easy selection of the customer associated with the contract. The "Contract ID" field is automatically generated by the system, serving as a unique identifier for each contract.

The center part of the window focuses on organizing the product details and facilitating the rental process. Users can select the product name from a dropdown list, which dynamically populates the corresponding "Product ID." The "Product Name" field displays the selected product name for reference. The "Cost per Hour" field is automatically retrieved from the database based on the selected product name, ensuring accurate pricing. The "Period in Hours" field allows users to specify the rental duration in hours for the selected product. Additionally, the "Total per Item" field displays the calculated total cost for the rental of each item, based on the cost per hour and rental period.

Towards the bottom of the center part, three fields provide crucial information for financial management. The "Total" field calculates the total cost for all items rented, considering the individual totals per item. The "Deposit" field allows users to input the deposit amount made by the customer. Finally, the "Balance" field calculates the outstanding balance by subtracting the deposit amount from the total cost.

At the top-right corner of the window, a prominently placed "Save" button allows users to save the contract information entered and finalize the rental agreement. This convenient button ensures that all contract details Additionally, in the top-left corner of the window, a .are securely stored and accessible for future reference "Back Home" button provides easy navigation back to the home page. This button enables users to switch between different modules or functionalities seamlessly, enhancing user experience and workflow efficiency

Here are the main methods of this component(class):
1. **getAllCustomerNames()**: Retrieves a list of all customer names from the "customer" table in the database.
2. **getNextContractID()**: Retrieves the next contract ID by finding the maximum ID from the "contract" table in the database and incrementing it by 1.
3. **getAllProductNames()**: Retrieves a list of all product names from the "product" table in the database.
4. **getAllProductNamesWithCategory()**: Retrieves a list of all product names along with their categories from the "product" table in the database.
5. **getProductIdByName(String productName)**: Retrieves the product ID for a given product name from the "product" table in the database.
6. **getProductCostByName(String productName)**: Retrieves the cost of a product for a given product name from the "product" table in the database.
7. **getCustomerIdByName(String customerName)**: Retrieves the customer ID for a given customer name from the "customer" table in the database.
8. **saveContractProduct(int productId, String productName, int period, int contractId)**: Saves a contract product by inserting the provided product ID, product name, period, and contract ID into the "contractproduct" table in the database.

These methods perform various database operations such as retrieving data, inserting data, and processing query results related to contracts, customers, and products

Security Measures:
The RTS system incorporates several security measures to ensure the confidentiality, integrity, and availability of the system. These measures have been implemented to protect user data, prevent unauthorized access, and mitigate potential vulnerabilities.

Firstly, the system enforces a robust authentication mechanism through the login page. Users are required to provide their username and password to access the system. This helps verify the identity of users and restricts access to authorized individuals only. Additionally, the system differentiates between administrators and regular users, providing specific privileges and access rights based on user roles.

This code snippet is used in LoginControl class:

```
public boolean isLogin(Admin ad) throws SQLException {
   st = ConnectionDB.OpenConnection().createStatement();
   ResultSet res = st.executeQuery("SELECT * FROM user WHERE username ='" +
ad.getUsername() + "' AND password ='" + ad.getPassword() + "'");
   return res.next();}
```

- User Authentication: The **isLogin** method takes an **Admin** object as a parameter, which contains the username and password provided by the user attempting to log in. The method then establishes a database connection using **ConnectionDB.OpenConnection()** and creates a statement (**st**) to execute the SQL query.
- SQL Query Execution: The code executes the SQL query **select * from user where username ='"+ad.getUsername()+"' and password ='"+ad.getPassword()+"'** to retrieve user data matching the provided username and password. This query checks the "user" table in the database for a row where the username and password match the given values.
- Result Evaluation: The **ResultSet** object (**res**) stores the results of the query execution. The code checks **res.next()** to determine if any rows were returned. If the result set has at least one row, it indicates that a user with the provided username and password exists in the database, and the login is considered successful.
- Security Implications: The login process, requiring a valid username and password combination, adds a layer of security to the system. It ensures that only users with valid credentials can access the system's protected resources. By verifying the provided username and password against the stored values in the "user" table, the code prevents unauthorized users from gaining access to sensitive information or performing restricted actions.

Secondly, the implementation of data encryption for sensitive information, such as user passwords, further enhances security within the RTS system. While the database itself may not be encrypted, user passwords are securely stored using strong encryption algorithms. This ensures that even in the event of a data breach, unauthorized individuals cannot access and decipher the stored passwords. By encrypting passwords, the system adds an extra layer of protection, reducing the risk of compromising user accounts and enhancing overall security.

Moreover, the system employs prepared statements in the Java application to prevent SQL injection attacks. Prepared statements ensure that user input is treated as data and not executable code, mitigating the risk of malicious SQL injections. This practice significantly enhances the security of the system and protects against one of the most common attack vectors in web applications.

Example from the ProdcutContorl class

```
PreparedStatement statement = ConnectionDB.OpenConnection().prepareStatement("INSERT INTO
product (name, category, price, status) VALUES (?, ?, ?, ?)");
```

- Prepared Statement: The use of a prepared statement helps protect against SQL injection attacks. By using placeholders (?) for the values and binding them later with specific data, the statement ensures that user input is treated as data rather than executable code. This prevents malicious users from manipulating the SQL query and helps maintain the integrity and security of the database.

Lastly, the integration of Gmail for email functionality adds an additional layer of security. By activating two-step authentication for the Gmail account used by the system, an extra level of verification is required to send emails. This helps prevent unauthorized use of the email feature, ensuring that only authorized users can send emails through the system. Furthermore, the use of two-step authentication reinforces the security of the Gmail account, reducing the risk of email-related vulnerabilities.

By implementing these security measures, the RTS system establishes a secure environment for managing rental tools, customer information, contracts, and user accounts. These measures work in conjunction to safeguard data, protect against unauthorized access, and mitigate common security risks, ensuring the integrity and confidentiality of the system.

TESTING STAGE

In this section, we will discuss the testing stage of the project. The main objective of the testing stage is to ensure that the implemented functionality meets the specified requirements and performs as expected. This stage involves various types of testing to validate the system's behavior and identify any issues or bugs that need to be addressed.

Unit Testing:

Unit testing is performed to verify the individual components or units of the system in isolation. In our project, we conducted unit testing on the **isLogin** method of the **Admin** class. The purpose of this test was to ensure that the login functionality works correctly.

Example unit test for the **isLogin** method:

```
@Test
public void testIsLogin() throws SQLException {
   // Prepare test data
   Admin admin = new Admin("testUser", "testPassword");
   // Call the method under test
   boolean result = isLogin(admin);
   // Assert the expected result
   assertTrue(result); }
```

This example creates a test case that verifies the **isLogin** method's behavior with a sample username and password. The test asserts the result should be **true** if the login is successful.

Integration Testing:

Integration testing focuses on testing the interactions between different components or modules of the system. In our project, we performed integration testing to validate the connection between the **isLogin** method and the **ConnectionDB** class.

An example is the integration test for the **isLogin** method, we tested the integration between the **isLogin** method and the **ConnectionDB** class, ensuring that the login functionality works correctly in conjunction with the database connection.

System Testing:

System testing evaluates the complete system's behavior and functionality as a whole. It aims to validate that all the integrated components work together seamlessly. In our project, we conducted system testing to ensure that the login functionality, including the user interface and database interaction, functions correctly.

Example system test scenario for the login feature:

1. Launch the application and navigate to the login page.
2. Enter valid credentials (username:"testUser", password:"testPassword") in the login form.
3. Click the login button and verify that the system successfully logs in the user.
4. Repeat step 2 with invalid credentials and ensure that the system displays an appropriate error message.

During system testing, we executed various test scenarios and verified that the system behaves as expected in each case. Qualitative and quantitative analysis can also be done here to evaluate end user acceptance testing (Kedwan F. H., 2017).

Conclusions And Recommendations

Conclusion

In the current project, a TRS desktop software design is built, which is a tool rental store system for renting products. The benefits of this system are to manage the renting process. The waterfall methodology is used. Hence, the project building started with the planning, requirements gathering, analyzing, design, implementation, and testing.

For the existing systems, software programs have been reviewed (Headmaster, Sharefox) as similar rental products systems. Their characteristics, and the limitations of each system have been compared with the current proposed system. Only relevant and useful functionalities from the services provided by the previous two systems have been adopted in the proposed system to serve the interests of the proposed system functionality, especially the basics functionalities of any rental systems. This is in addition to the presence of some of the features of the proposed system that distinguish it from the previous systems. However, there are some limitations and restrictions in the current system due to the requirements imposed by the stakeholders and the nature of business domain of the RTS system.

It is recommended that the proposed system be further improved by developing and implementing a full web-based solution with online payment options to make it easier and faster for users and customers to access.

Future Work (Recommendations)

Website Application

Websites are common these days. The objective of designing a website that is to bring in dynamism and interaction between the website and its users. The data presented to the website users have to be updated frequently without manual interventions. The data is displayed from the database automatically. Such automated process is less prone to errors (Akpji, 2015). Making a web application is one of our future plans to enhance the accessibility and the global orientation so the customers can request services from anywhere in the world (Thakur, 2021).

Online Ordering

Online ordering feature saves time and effort for the customer and gives him an opportunity to browse through the website for all the tools they need and then reserve them in the shopping cart as an order that needs confirmation (payment) in order to become a confirmed order. Ordering via the Internet enhances the concept of ease of purchase and increases the number of customers because of the ease of access and creates a kind of loyalty to the organization and perhaps lower prices than traditional purchasing methods.

Online Payment

Online payment has become almost mandatory with every website these days in the field of e-commercial stores of all kinds. This is because it facilitates and speeds up the payment process, which leads to smoothness in all the purchase processes and the subsequent operations. Also, there are some security problems in traditional payment systems that help the payments system to bring out a huge revolution (Momin Mukherjee , Sahadev Roy, 2017).

Difficulties

Throughout the development journey, several challenges were encountered in various aspects of the project. When working with JavaFX, there were difficulties in dynamically updating UI components based on user interactions. For example, synchronizing data displayed in different views and handling complex layouts required careful consideration and thorough testing. In the realm of database connectivity, we struggled with optimizing SQL queries to retrieve data efficiently, especially when dealing with large datasets. Additionally, managing database transactions and handling exceptions related to database operations posed challenges that demanded meticulous error handling and debugging.

Integrating Scene Builder into the development workflow brought its own set of obstacles. Ensuring proper alignment between the UI components in Scene Builder and the associated code became a meticulous task, especially when complex event handling and data bindings were involved. Troubleshooting compatibility issues between different versions of Scene Builder and JavaFX libraries added to the complexity.

The Java language itself presented challenges, particularly in managing object-oriented concepts effectively. This involved designing robust class hierarchies, implementing inheritance and polymorphism, and resolving issues related to memory management and resource optimization. Debugging intricate code structures, identifying, and resolving performance bottlenecks, and maintaining code readability were constant considerations throughout the development process.

Furthermore, configuring and managing the XAMPP server required a deep understanding of server architecture and network configurations. There were difficulties in setting up the server environment, ensuring compatibility with the MySQL database, and addressing server-related errors that impacted the system's functionality.

References

1. Verdon, J. (2019, 11 30). *The Rental Economy Takes Flight*. Retrieved from www.uschamber.com.
2. Headmaster. (2010, 1 26). *Headmaster for accounts and stores*. Retrieved from http://www.headmasteraccounts.com/.
3. Sharefox. (2021, 2 5). *Fully-customizable online rental store*. Retrieved from https://sharefox.com.
4. Molino, T. (2021, 8 24). *Top 10 Most Spoken Languages in The Business World*. Retrieved from www.gmsmobility.com.
5. Samanta, S. (2021, 6 21). *Why is an Ecommerce Website Important?* Retrieved from www.opengrowth.com.
6. Gundaniya, N. (2021, 1 8). *7 Benefits of Electronic Payments*. Retrieved from customerthink.com.
7. Akpji, K. (2015). *A Web-Based Rental System (A Case-Study of Unibet Transport Services).* Benin : ACADEMIA.
8. Thakur, A. (2021). Car Rental System. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, 412.
9. Momin Mukherjee , Sahadev Roy. (2017). E-Commerce and Online Payment in the Modern Era. *International Journal of Advanced Research in,* 5.
10. Usability.gov. (2015, 4 5). *Use Cases*. Retrieved from www.usability.gov: https://www.usability.gov/how-to-and-tools/methods/use-cases.html
11. lucidchart. (n.d.). *What is a use case diagram?* Retrieved from www.lucidchart.com: https://www.lucidchart.com/pages/uml-use-case-diagram
12. simmytarika5. (2022, 5 2). *Unified Modeling Language (UML) | Activity Diagrams*. Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/
13. Martin, M. (2022, 2 19). *N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE*. Retrieved from guru99: https://www.guru99.com/n-tier-architecture-system-concepts-tips.html
14. IBM_Cloud_Education. (2022, 10 28). *Three-Tier Architecture*. Retrieved from www.ibm.com: https://www.ibm.com/cloud/learn/three-tier-architecture
15. Daeldung. (2021, 11 11). *Layered Architecture*. Retrieved from www.baeldung.com: https://www.baeldung.com/cs/layered-architecture
16. Len Bass, Paul Clements, Rick Kazman. (2012). *Software Architecture in Practice.* Westford: Addison-Wesley.
17. Bell, D. (2004, 12 15). UML basics: The component diagram. *IBM Corporation*.
18. Refactoring.Guru. (n.d.). *What's a design pattern?* Retrieved from Refactoring.Guru.: https://refactoring.guru/design-patterns/what-is-pattern

19. Nishadha. (2022, 11 4). *UML Class Diagram Relationships Explained with Examples*. Retrieved from creately: https://creately.com/blog/diagrams/class-diagram-relationships/
20. SOLUTIONS, R. R. (2023, 2 15). *A RentMaster*. Retrieved from rentmaster.net: https://www.rentmaster.net/products/rentmaster-er-version
21. EZRentOut. (2021, 3 21). *A EZRentOut*. Retrieved from EZRentOut.net: https://www.g2.com/
22. SOLUTIONS, R. R. (2023, 2 15). *A RentMaster*. Retrieved from rentmaster.net: https://www.rentmaster.net/products/rentmaster-er-version
23. Kedwan, F. H. (2017). *Patients Online Registration System: Feasibility and Perceptions.* Medina: Annals of Medical and Health Sciences Research.
24. Kedwan, F. H. (2019). *Model-Driven Software Development Platforms Reviews* (Vols. ISBN: 973-93-80900-26-6). International Journal of Computer Applications.
25. Kedwan, F. (2024). *A Software Engineering Approach on Developing a Real Time Radar Target Generator for Airborne Targets* (Vol. 3(2)). Medina, Saudi Arabia: Applied Science and Engineering Journal for Advanced Research.

Appendix

Appendix 1

Table 1: Compasion between Headmaster and Sharefox against TRS system

Features	***Headmaster***	***Sharefox***	***TRS***
Add, modify, delete Customers	✓	✓	✓
Add, modify, delete Users	✓	✓	✓
Add, modify, delete Products	✓	✓	✓
Create Reports	✓	✓	✓
Create Lease Contracts	✓	✓	✓
Product Search Tool	✓	✓	✓
Friendly & Modern GUI	☒	✓	✓
Suppliers Management	✓	✓	☒ Our program does not support this feature because it does not require the entry of purchases through companies and it not one of the requirements.
Rental invoice in Arabic	✓	☒	☒ Because it is not part of the requirements and English is the most prominent language in business (Molino, 2021).
Rental invoice in English	☒	✓	✓
Desktop Application	✓	☒	✓
Web Application	☒	✓	Future plan We strive for our system to support the web to be more

			usable, accessible, and increase customer reach (Samanta, 2021).
Social Media and Multimedia Integrations	☒	✓	☒ It is not among the requirements, and most clients in the field of this work are not social media users.
Online Payment Tool	☒	✓	Future plan This step helps speed up and organize the rental process and is necessary when the web is available and it provide higher security form the traditional payment (Gundaniya, 2021).
Keep Track of Contracts and Leased Products	☒	☒	✓ This feature is specific to our system, as it tracks and monitors lease expiration dates in an organized manner.
Send Reminders to The Customer and The Manager	☒	☒	✓ This is another unique feature of our system based on sending reminders to both the customer and the manager.

Appendix 2

Features	***Rent Master***	***EZRentout***	***TRS***
Add, modify, delete Customers	✓	✓	✓
Add, modify, delete Users	✓	✓	✓
Add, modify, delete Products	✓	✓	✓
Create Reports	✓	✓	✓
Create Lease Contracts	✓	✓	✓
Product Search Tool	✓	✓	✓
Friendly & Modern GUI	☒	✓	✓
Suppliers Management	✓	✓	☒ Our program does not support this feature because it does not require the entry of purchases through companies and it not one of the requirements.
Rental invoice in Arabic	✓	☒	☒ Because it is not part of the requirements and English is the most prominent language in business (Molino, 2021).
Rental invoice in English	☒	✓	✓
Desktop Application	✓	☒	✓
Web Application	☒	✓	Future plan We strive for our system to support the web to be more usable, accessible, and increase customer reach (Samanta, 2021).
Social Media and	☒	✓	☒

Multimedia Integrations			It is not among the requirements, and most clients in the field of this work are not social media users.
Online Payment Tool	☒	✓	Future plan This step helps speed up and organize the rental process and is necessary when the web is available and it provide higher security form the traditional payment (Gundaniya, 2021).
Keep Track of Contracts and Leased Products	☒	☒	✓ This feature is specific to our system, as it tracks and monitors lease expiration dates in an organized manner.
Send Reminders to The Customer and The Manager	☒	☒	✓ This is another unique feature of our system based on sending reminders to both the customer and the manager.

Table 2: Compasion between Rent Master and EZRentout against TRS system

Appendix 3

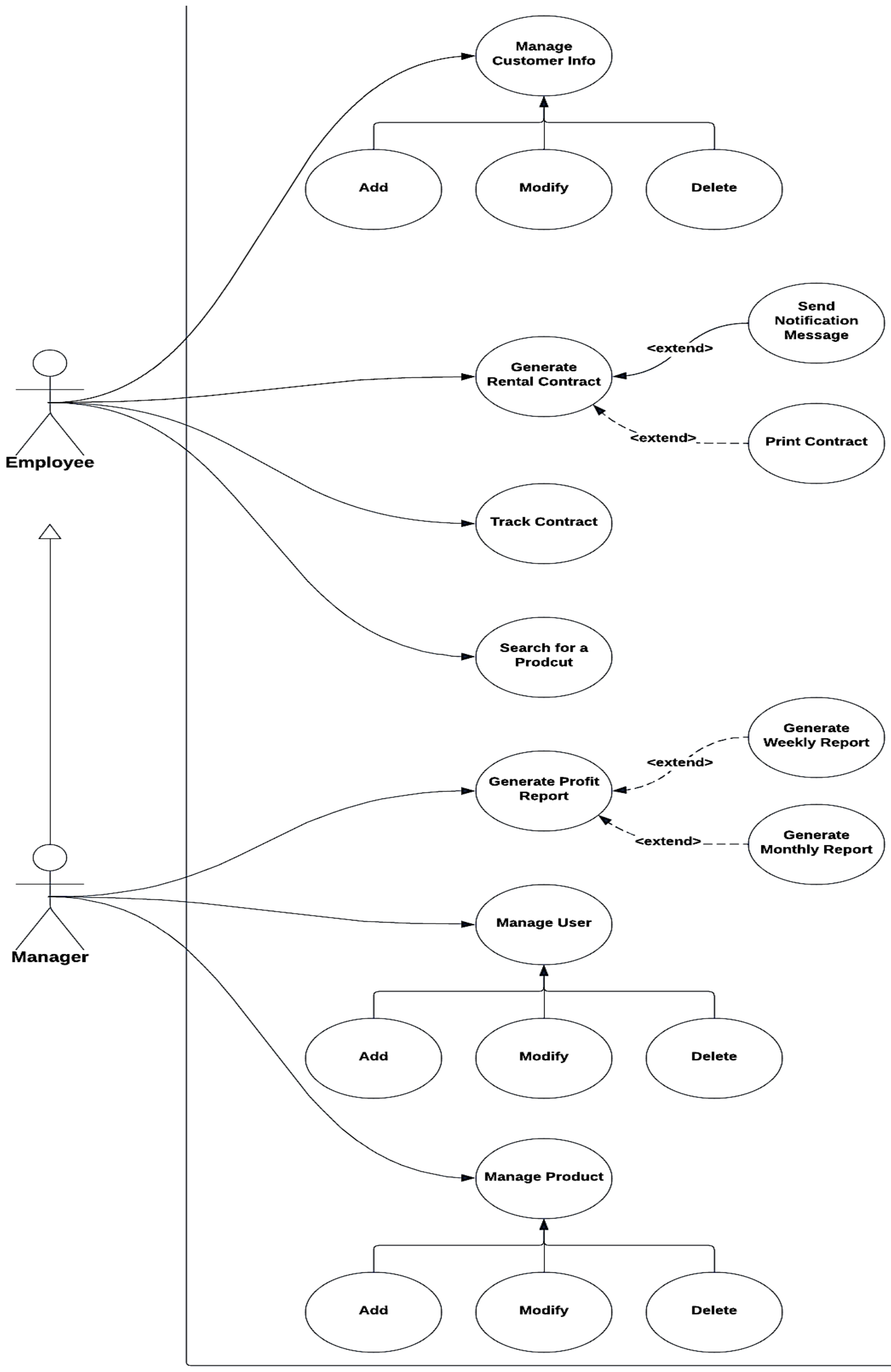
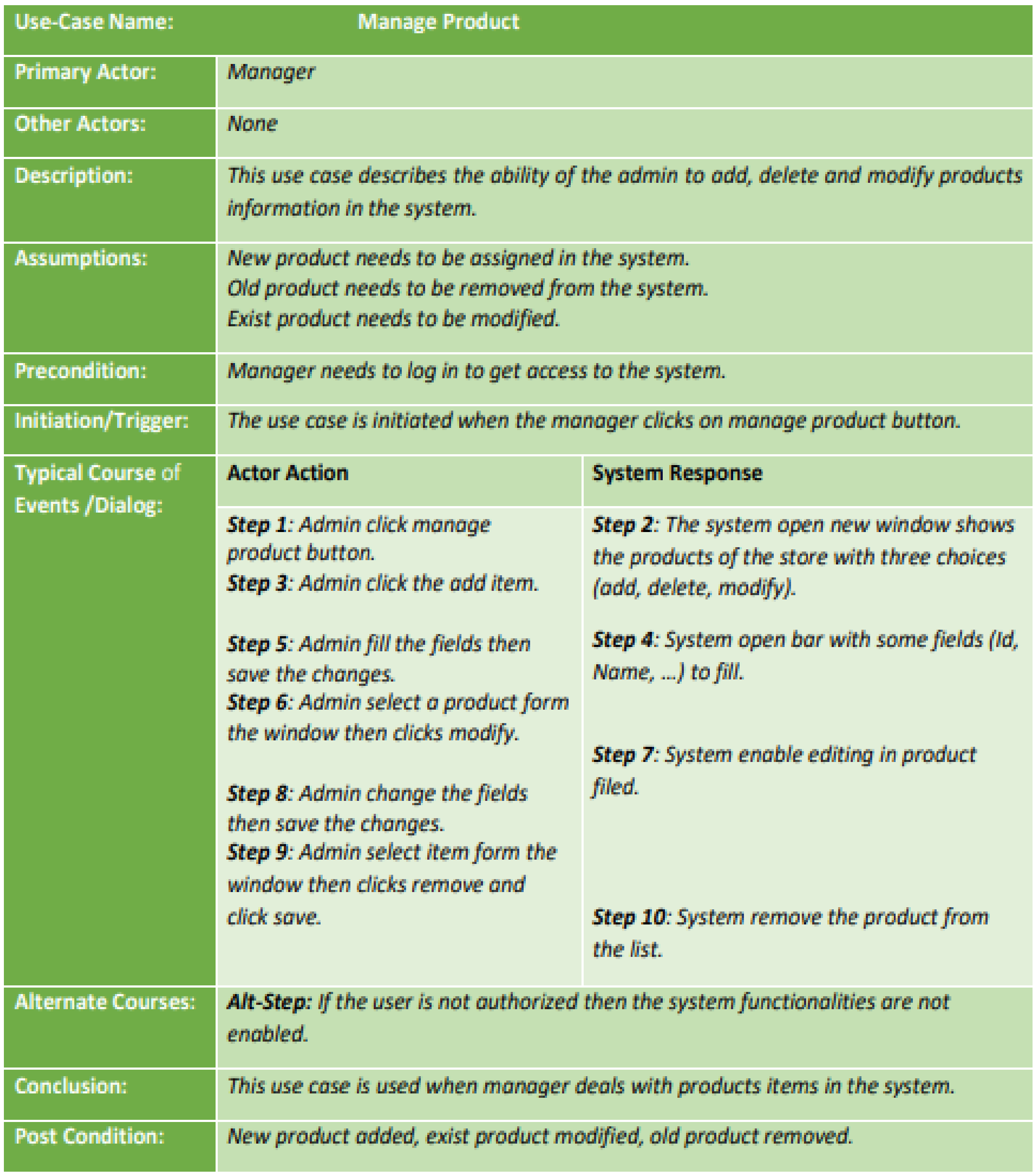


Figure 1: Use case diagram demonstration of the actors and system relationship

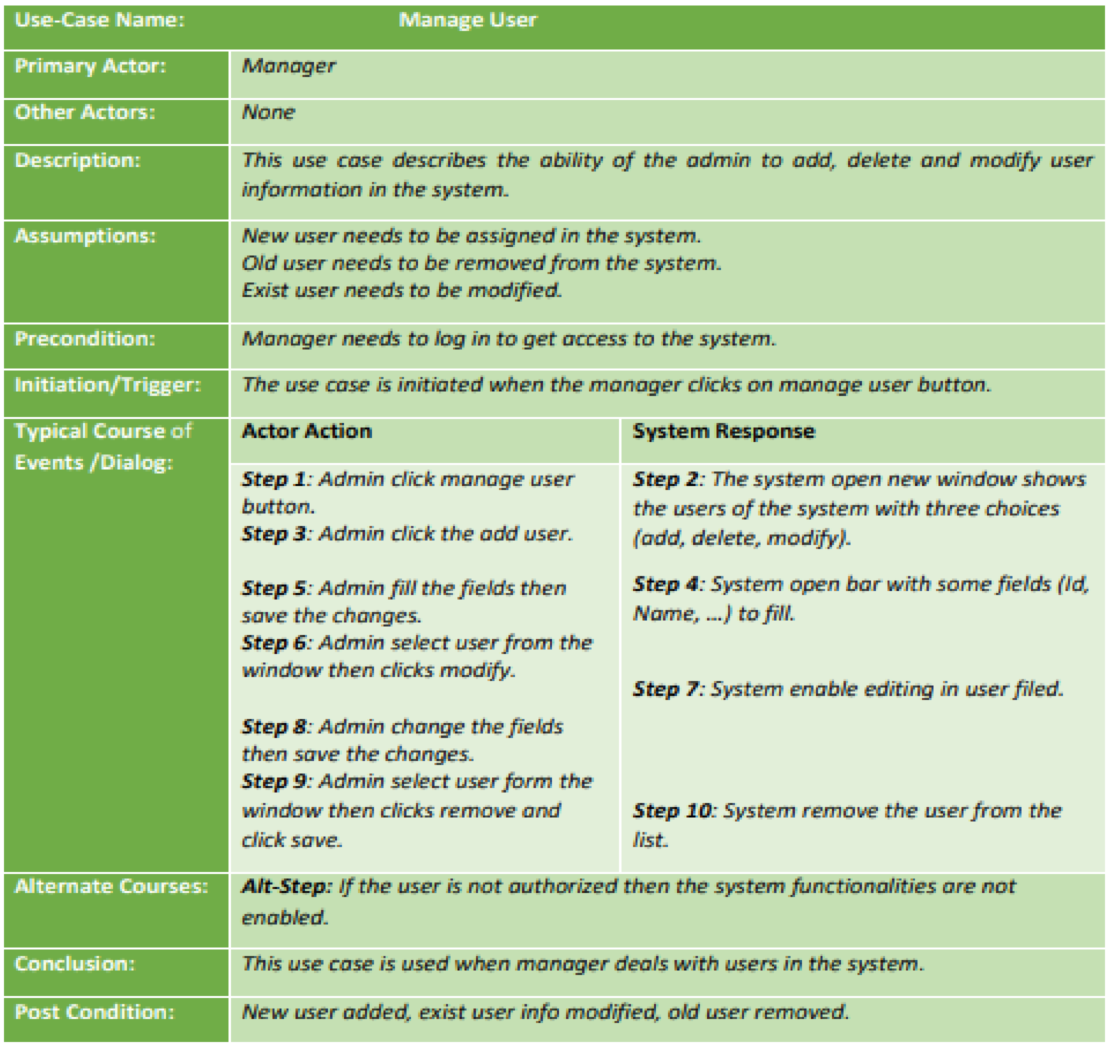
Appendix 4



Use-Case Name:	Manage Product	
Primary Actor:	*Manager*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the admin to add, delete and modify products information in the system.*	
Assumptions:	*New product needs to be assigned in the system.* *Old product needs to be removed from the system.* *Exist product needs to be modified.*	
Precondition:	*Manager needs to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager clicks on manage product button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	Step 1*: Admin click manage product button.* ***Step 3****: Admin click the add item.*	***Step 2****: The system open new window shows the products of the store with three choices (add, delete, modify).*
	Step 5*: Admin fill the fields then save the changes.* ***Step 6****: Admin select a product form the window then clicks modify.*	***Step 4****: System open bar with some fields (Id, Name, …) to fill.*
	Step 8*: Admin change the fields then save the changes.* ***Step 9****: Admin select item form the window then clicks remove and click save.*	***Step 7****: System enable editing in product filed.*
		Step 10*: System remove the product from the list.*
Alternate Courses:	***Alt-Step:*** *If the user is not authorized then the system functionalities are not enabled.*	
Conclusion:	*This use case is used when manager deals with products items in the system.*	
Post Condition:	*New product added, exist product modified, old product removed.*	

Table 8: Use case discription for Manage Product use case

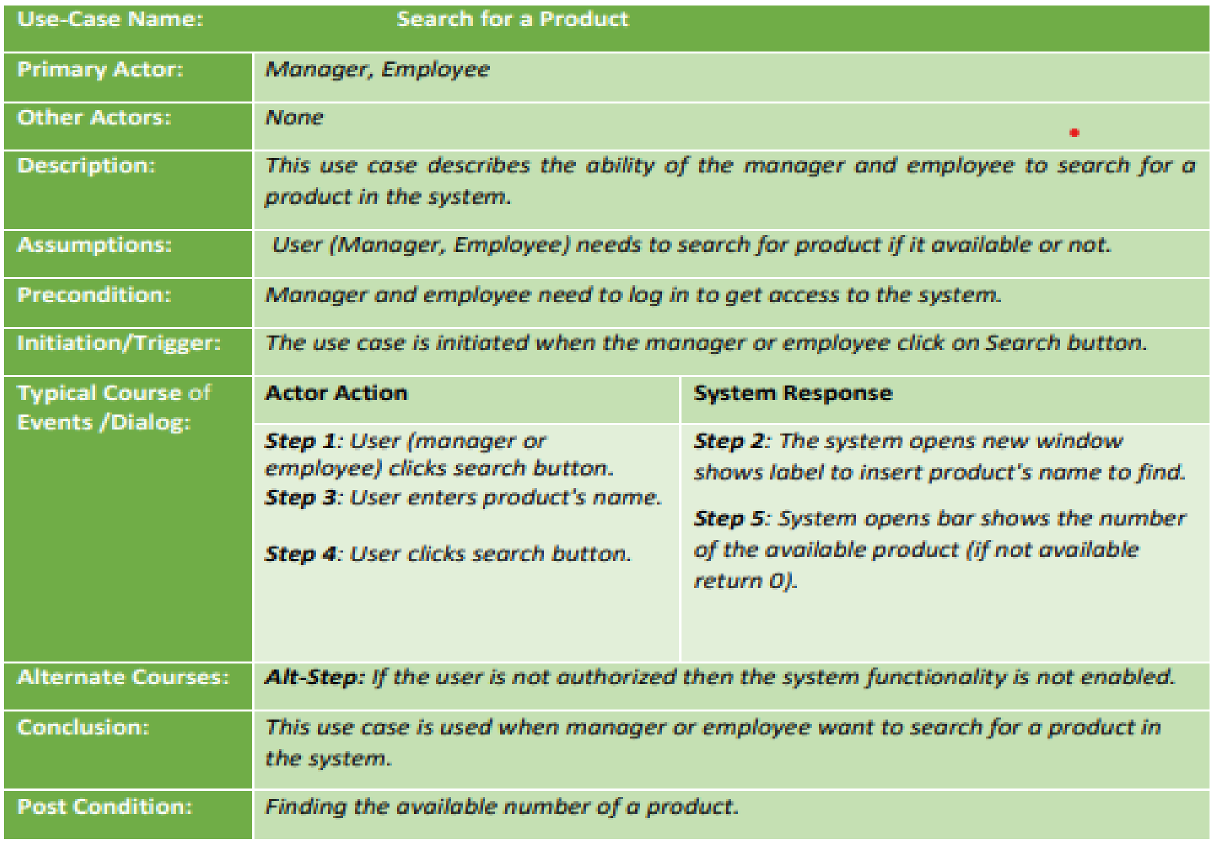
Appendix 5



Use-Case Name:	**Manage User**	
Primary Actor:	*Manager*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the admin to add, delete and modify user information in the system.*	
Assumptions:	*New user needs to be assigned in the system.* *Old user needs to be removed from the system.* *Exist user needs to be modified.*	
Precondition:	*Manager needs to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager clicks on manage user button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	***Step 1**: Admin click manage user button.* ***Step 3**: Admin click the add user.*	***Step 2**: The system open new window shows the users of the system with three choices (add, delete, modify).*
	***Step 5**: Admin fill the fields then save the changes.* ***Step 6**: Admin select user from the window then clicks modify.*	***Step 4**: System open bar with some fields (Id, Name, …) to fill.*
	***Step 8**: Admin change the fields then save the changes.* ***Step 9**: Admin select user form the window then clicks remove and click save.*	***Step 7**: System enable editing in user filed.* ***Step 10**: System remove the user from the list.*
Alternate Courses:	***Alt-Step:** If the user is not authorized then the system functionalities are not enabled.*	
Conclusion:	*This use case is used when manager deals with users in the system.*	
Post Condition:	*New user added, exist user info modified, old user removed.*	

Table 9: Use case discription for Manage User use case

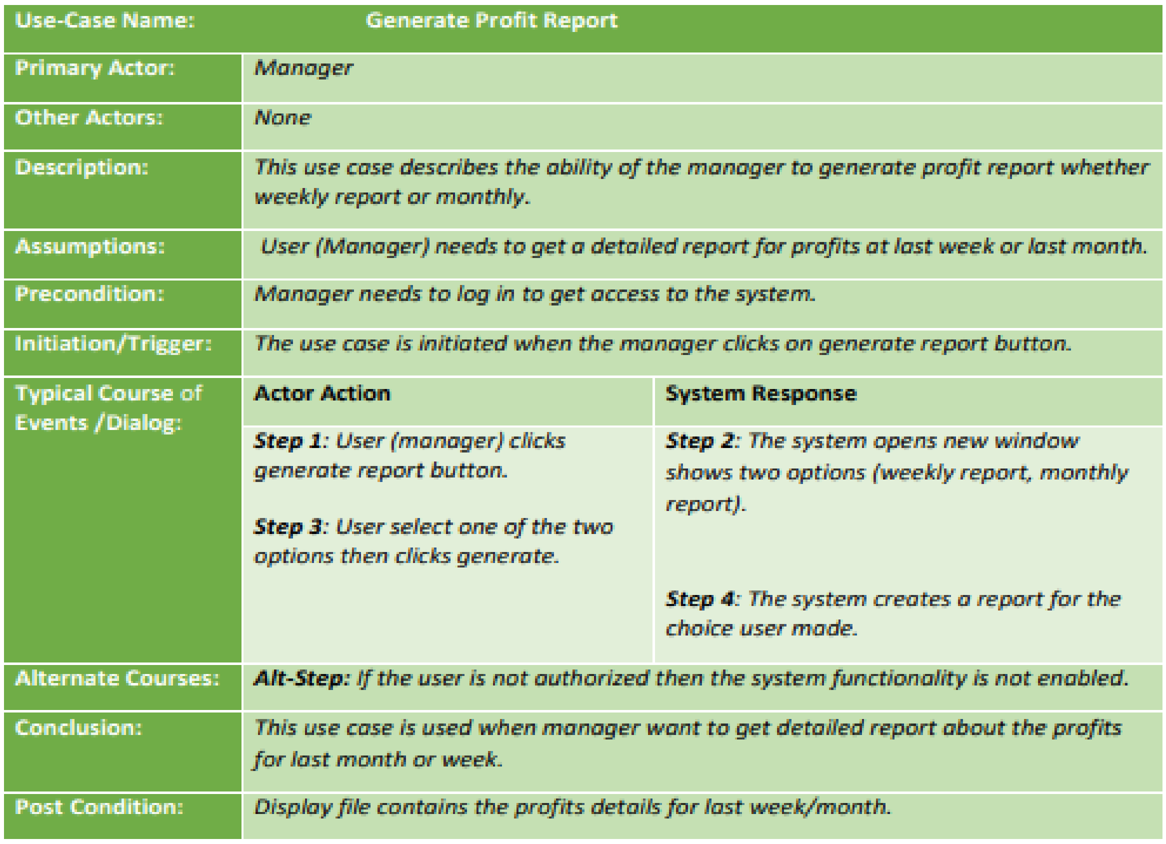
Appendix 6



Use-Case Name:	Search for a Product	
Primary Actor:	*Manager, Employee*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the manager and employee to search for a product in the system.*	
Assumptions:	*User (Manager, Employee) needs to search for product if it available or not.*	
Precondition:	*Manager and employee need to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager or employee click on Search button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	***Step 1**: User (manager or employee) clicks search button.* ***Step 3**: User enters product's name.* ***Step 4**: User clicks search button.*	***Step 2**: The system opens new window shows label to insert product's name to find.* ***Step 5**: System opens bar shows the number of the available product (if not available return 0).*
Alternate Courses:	***Alt-Step:** If the user is not authorized then the system functionality is not enabled.*	
Conclusion:	*This use case is used when manager or employee want to search for a product in the system.*	
Post Condition:	*Finding the available number of a product.*	

Table 10: Use case discription for Search for a Product use case

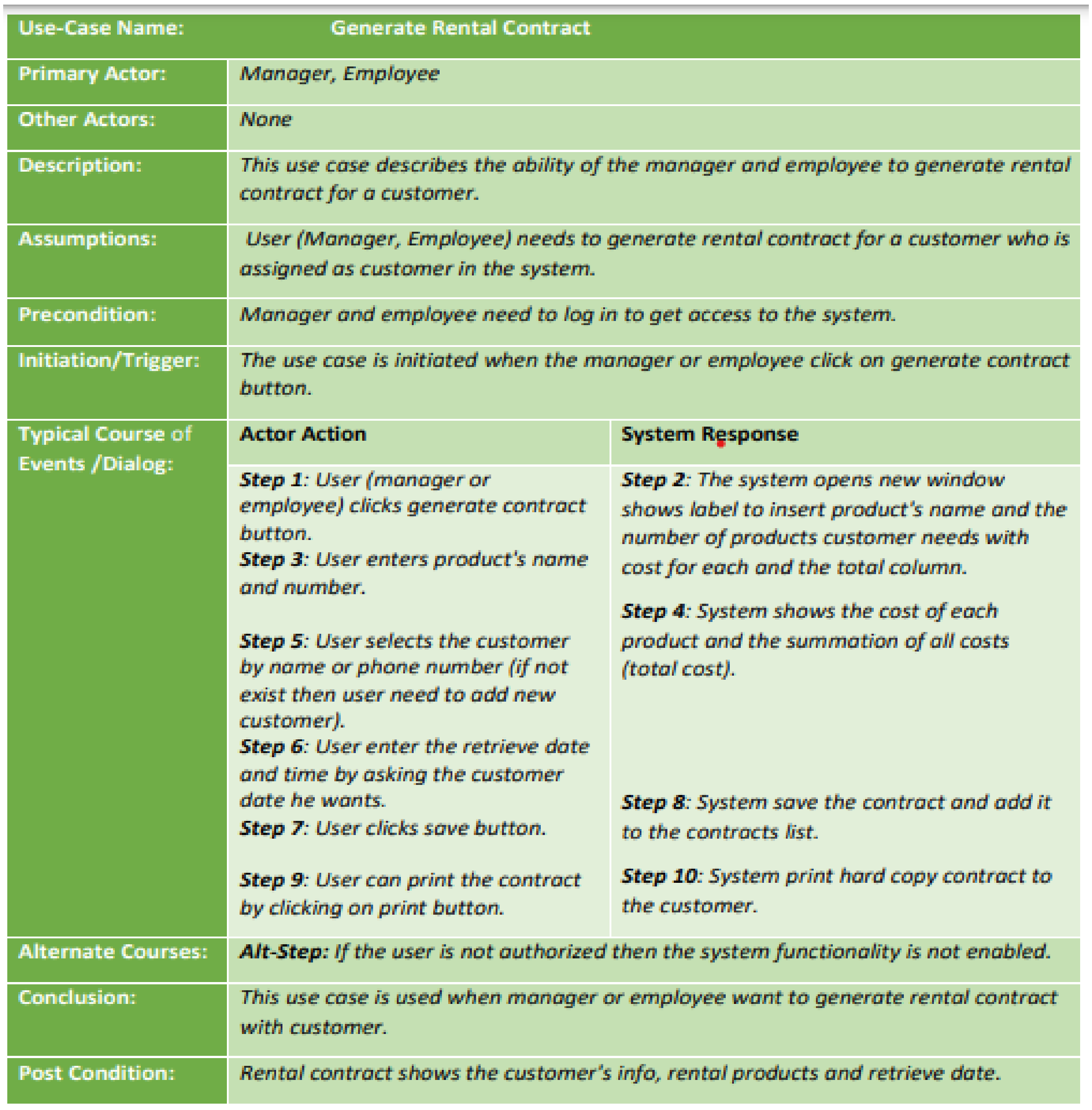
Appendix 7



Use-Case Name:	Generate Profit Report	
Primary Actor:	*Manager*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the manager to generate profit report whether weekly report or monthly.*	
Assumptions:	*User (Manager) needs to get a detailed report for profits at last week or last month.*	
Precondition:	*Manager needs to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager clicks on generate report button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	***Step 1**: User (manager) clicks generate report button.* ***Step 3**: User select one of the two options then clicks generate.*	***Step 2**: The system opens new window shows two options (weekly report, monthly report).* ***Step 4**: The system creates a report for the choice user made.*
Alternate Courses:	***Alt-Step:** If the user is not authorized then the system functionality is not enabled.*	
Conclusion:	*This use case is used when manager want to get detailed report about the profits for last month or week.*	
Post Condition:	*Display file contains the profits details for last week/month.*	

Table 11: Use case discription for Generate Profit Report use cas

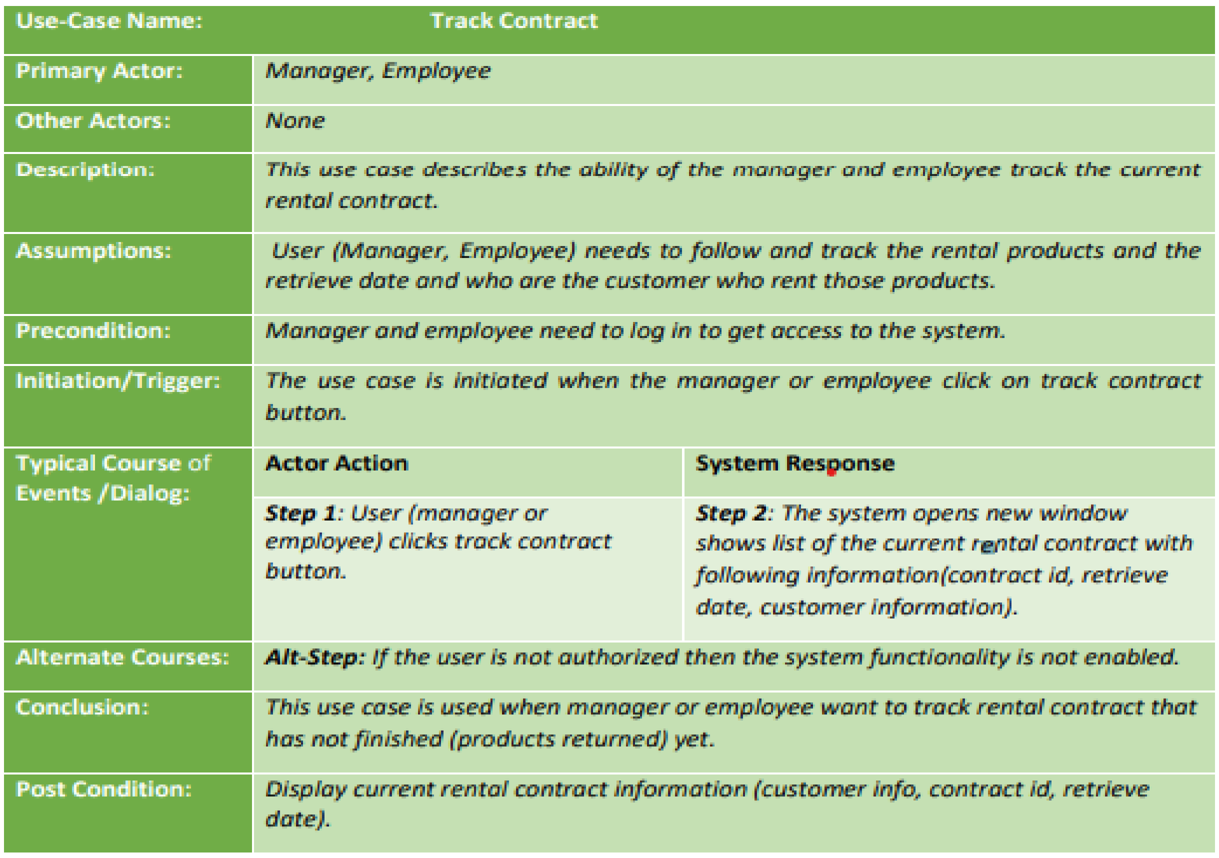
Appendix 8



Use-Case Name:	Generate Rental Contract	
Primary Actor:	*Manager, Employee*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the manager and employee to generate rental contract for a customer.*	
Assumptions:	*User (Manager, Employee) needs to generate rental contract for a customer who is assigned as customer in the system.*	
Precondition:	*Manager and employee need to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager or employee click on generate contract button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	***Step 1**: User (manager or employee) clicks generate contract button.*	***Step 2**: The system opens new window shows label to insert product's name and the number of products customer needs with cost for each and the total column.*
	***Step 3**: User enters product's name and number.*	***Step 4**: System shows the cost of each product and the summation of all costs (total cost).*
	***Step 5**: User selects the customer by name or phone number (if not exist then user need to add new customer).*	
	***Step 6**: User enter the retrieve date and time by asking the customer date he wants.*	
	***Step 7**: User clicks save button.*	***Step 8**: System save the contract and add it to the contracts list.*
	***Step 9**: User can print the contract by clicking on print button.*	***Step 10**: System print hard copy contract to the customer.*
Alternate Courses:	***Alt-Step:** If the user is not authorized then the system functionality is not enabled.*	
Conclusion:	*This use case is used when manager or employee want to generate rental contract with customer.*	
Post Condition:	*Rental contract shows the customer's info, rental products and retrieve date.*	

Table12: Use case discription for Generate Rental Contract use case

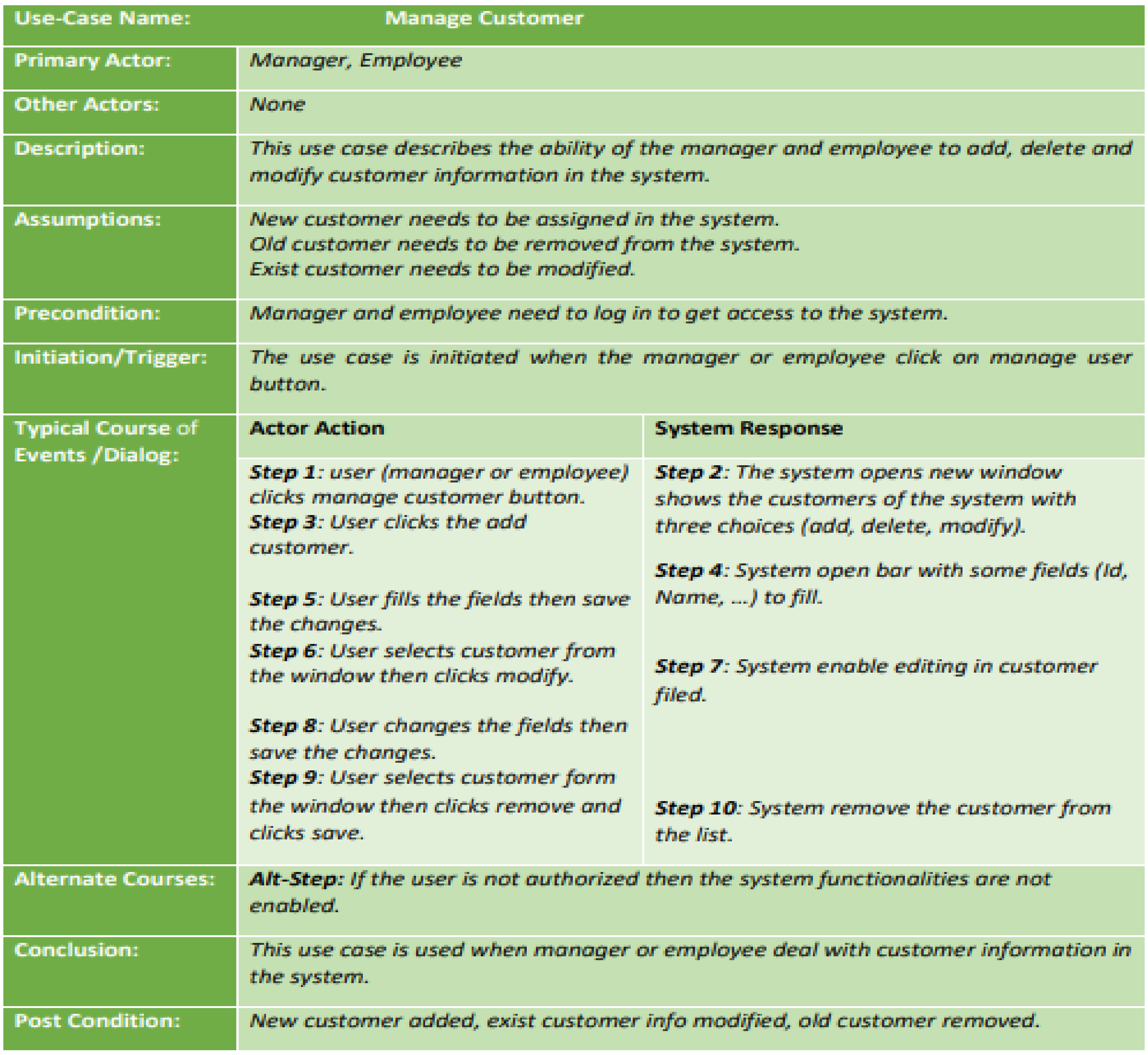
Appendix 9



Use-Case Name:	Track Contract	
Primary Actor:	*Manager, Employee*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the manager and employee track the current rental contract.*	
Assumptions:	*User (Manager, Employee) needs to follow and track the rental products and the retrieve date and who are the customer who rent those products.*	
Precondition:	*Manager and employee need to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager or employee click on track contract button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	***Step 1**: User (manager or employee) clicks track contract button.*	***Step 2**: The system opens new window shows list of the current rental contract with following information(contract id, retrieve date, customer information).*
Alternate Courses:	***Alt-Step:** If the user is not authorized then the system functionality is not enabled.*	
Conclusion:	*This use case is used when manager or employee want to track rental contract that has not finished (products returned) yet.*	
Post Condition:	*Display current rental contract information (customer info, contract id, retrieve date).*	

Table13: Use case discription for Track Contract use case

Appendix 10



Use-Case Name:	Manage Customer	
Primary Actor:	*Manager, Employee*	
Other Actors:	*None*	
Description:	*This use case describes the ability of the manager and employee to add, delete and modify customer information in the system.*	
Assumptions:	*New customer needs to be assigned in the system.* *Old customer needs to be removed from the system.* *Exist customer needs to be modified.*	
Precondition:	*Manager and employee need to log in to get access to the system.*	
Initiation/Trigger:	*The use case is initiated when the manager or employee click on manage user button.*	
Typical Course of Events /Dialog:	**Actor Action**	**System Response**
	Step 1*: user (manager or employee) clicks manage customer button.* ***Step 3****: User clicks the add customer.* ***Step 5****: User fills the fields then save the changes.* ***Step 6****: User selects customer from the window then clicks modify.* ***Step 8****: User changes the fields then save the changes.* ***Step 9****: User selects customer form the window then clicks remove and clicks save.*	***Step 2****: The system opens new window shows the customers of the system with three choices (add, delete, modify).* ***Step 4****: System open bar with some fields (Id, Name, …) to fill.* ***Step 7****: System enable editing in customer filed.* ***Step 10****: System remove the customer from the list.*
Alternate Courses:	***Alt-Step:*** *If the user is not authorized then the system functionalities are not enabled.*	
Conclusion:	*This use case is used when manager or employee deal with customer information in the system.*	
Post Condition:	*New customer added, exist customer info modified, old customer removed.*	

Table14: Use case discription for Manage Customer use case

Appendix 11

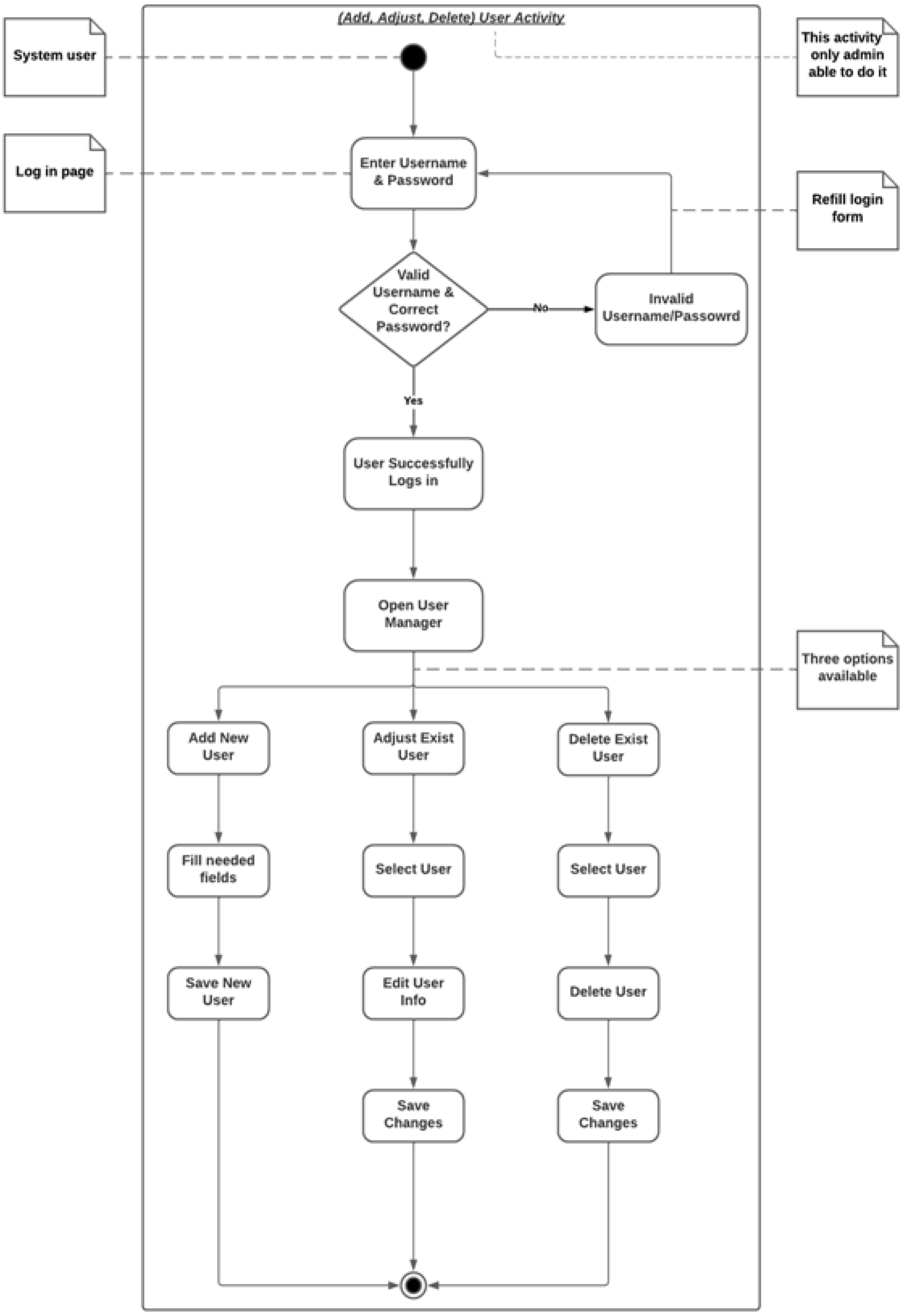


Figure 1: Activity diagram for Manage User use case

Appendix 12

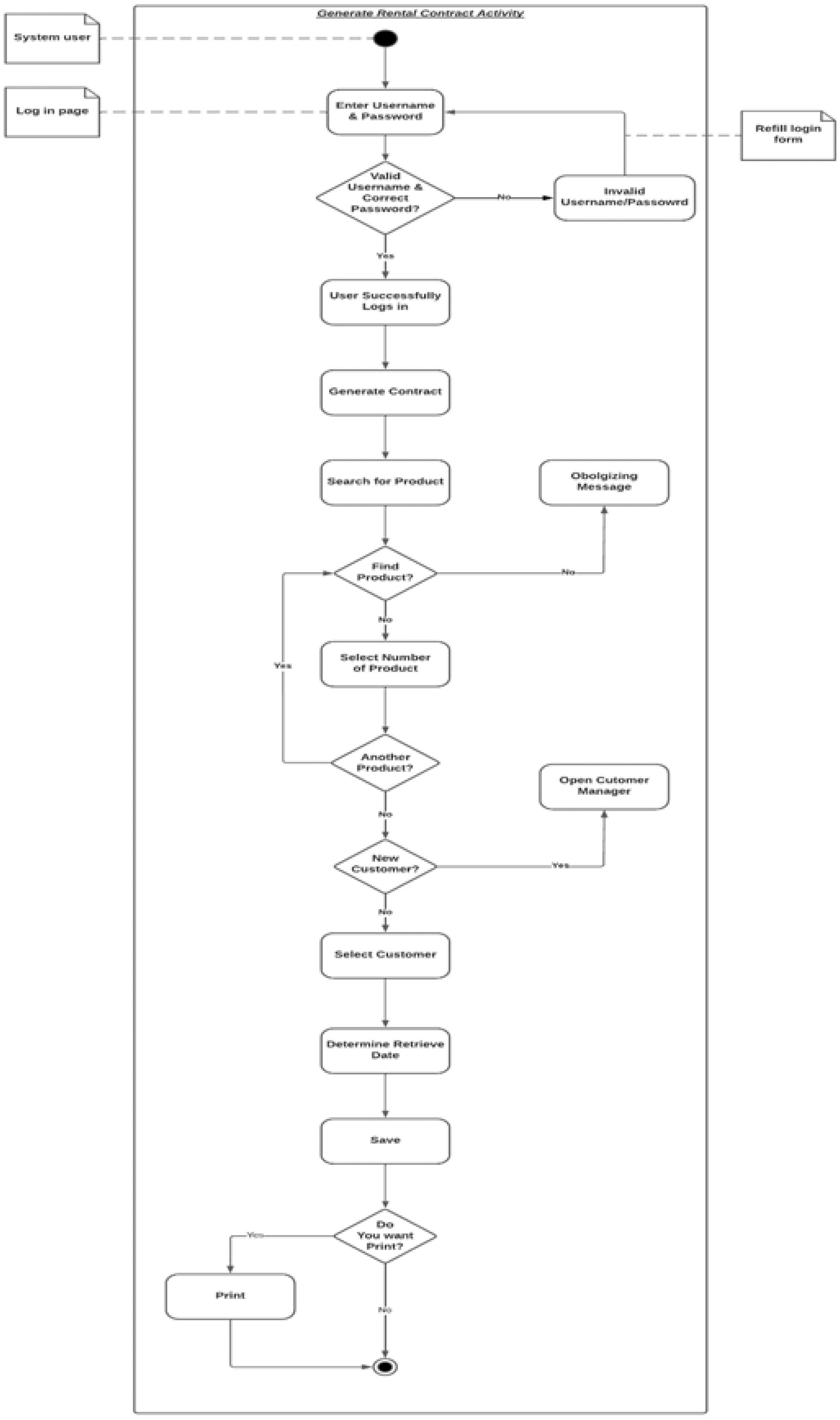


Figure 1: Activity diagram for Manage Contract use case

Appendix 13

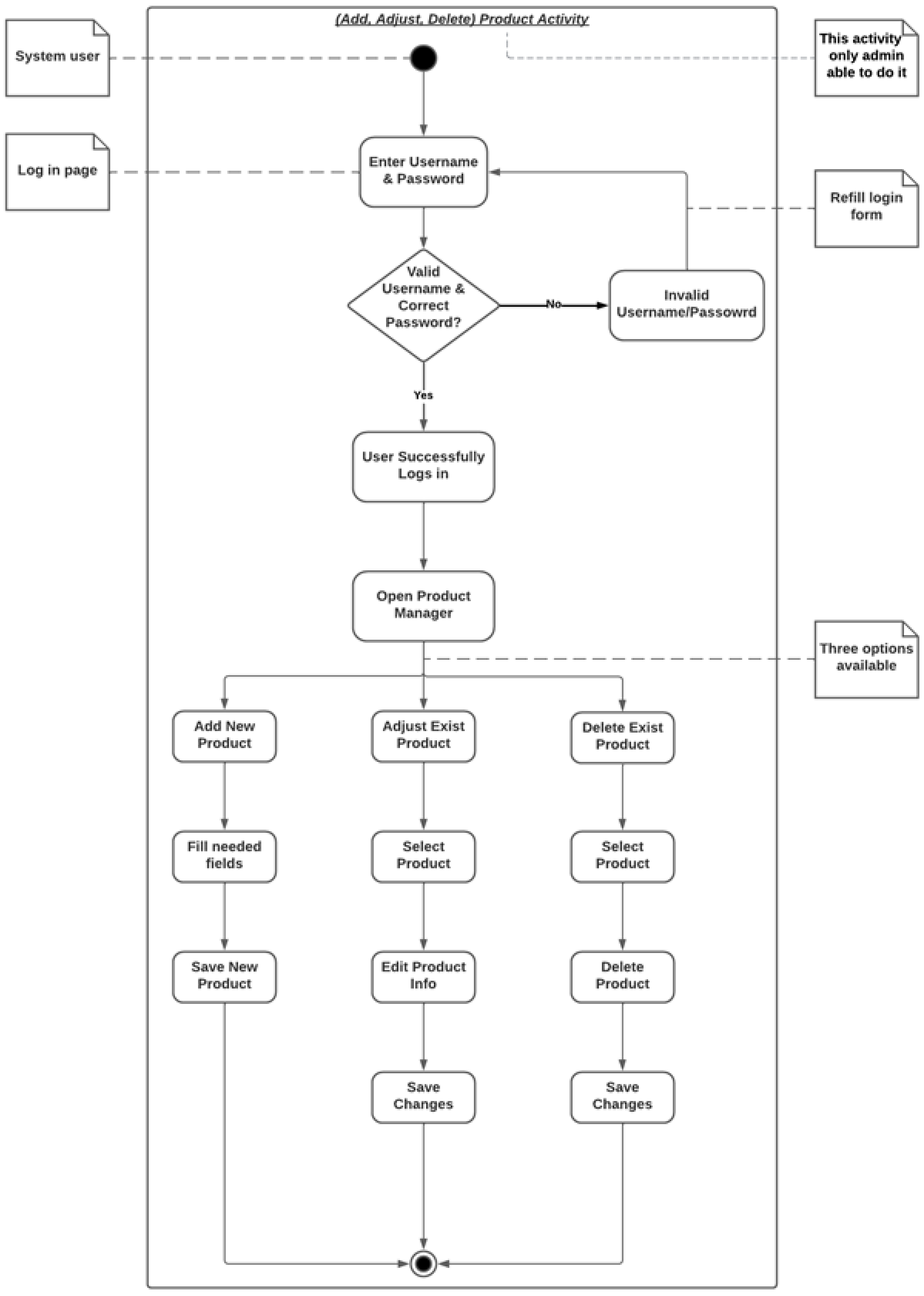


Figure 1: Activity diagram for Manage Product use case

Appendix 14

Appendix 15

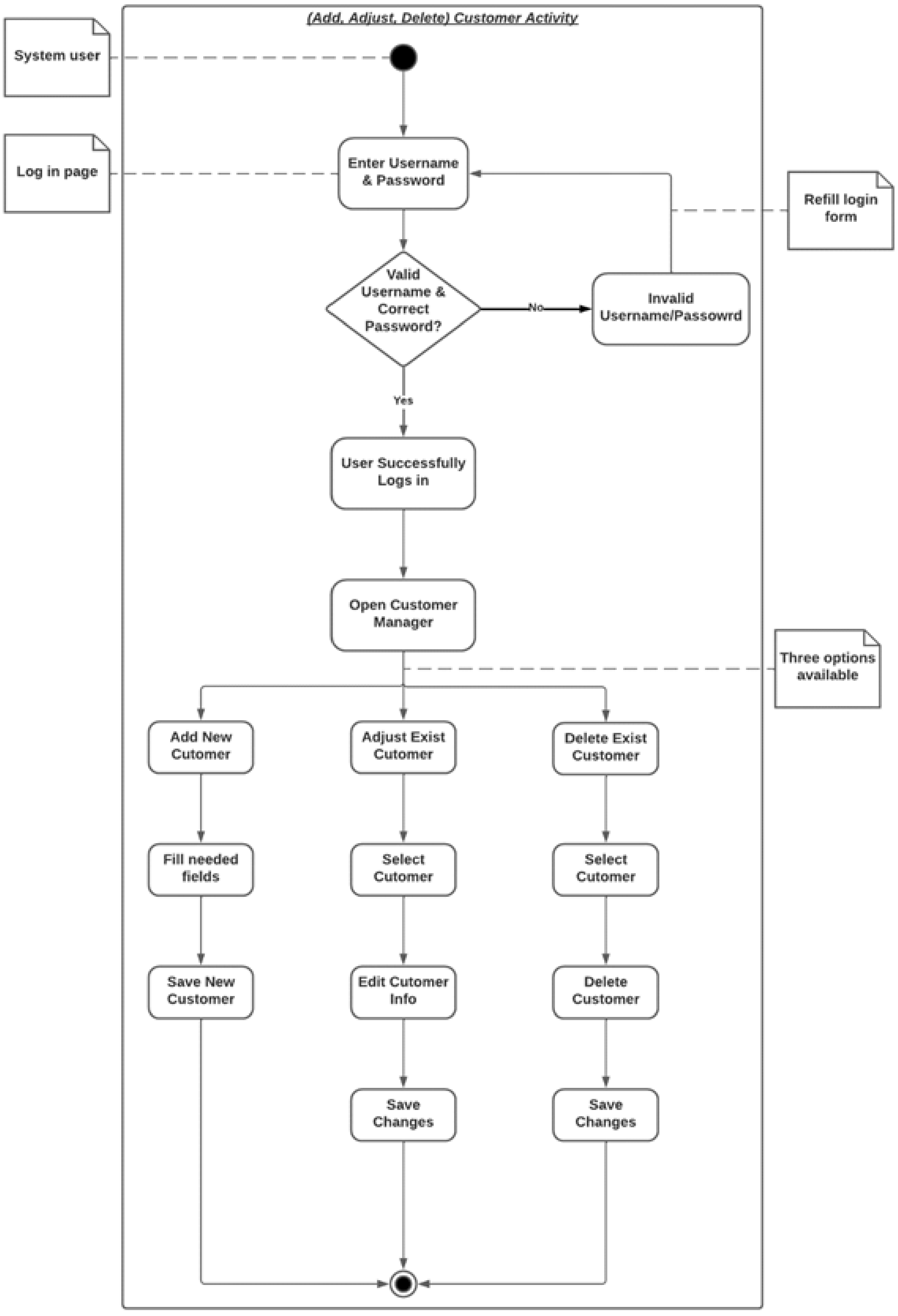


Figure 1: Activity diagram for Manage Customer use case

Appendix 15

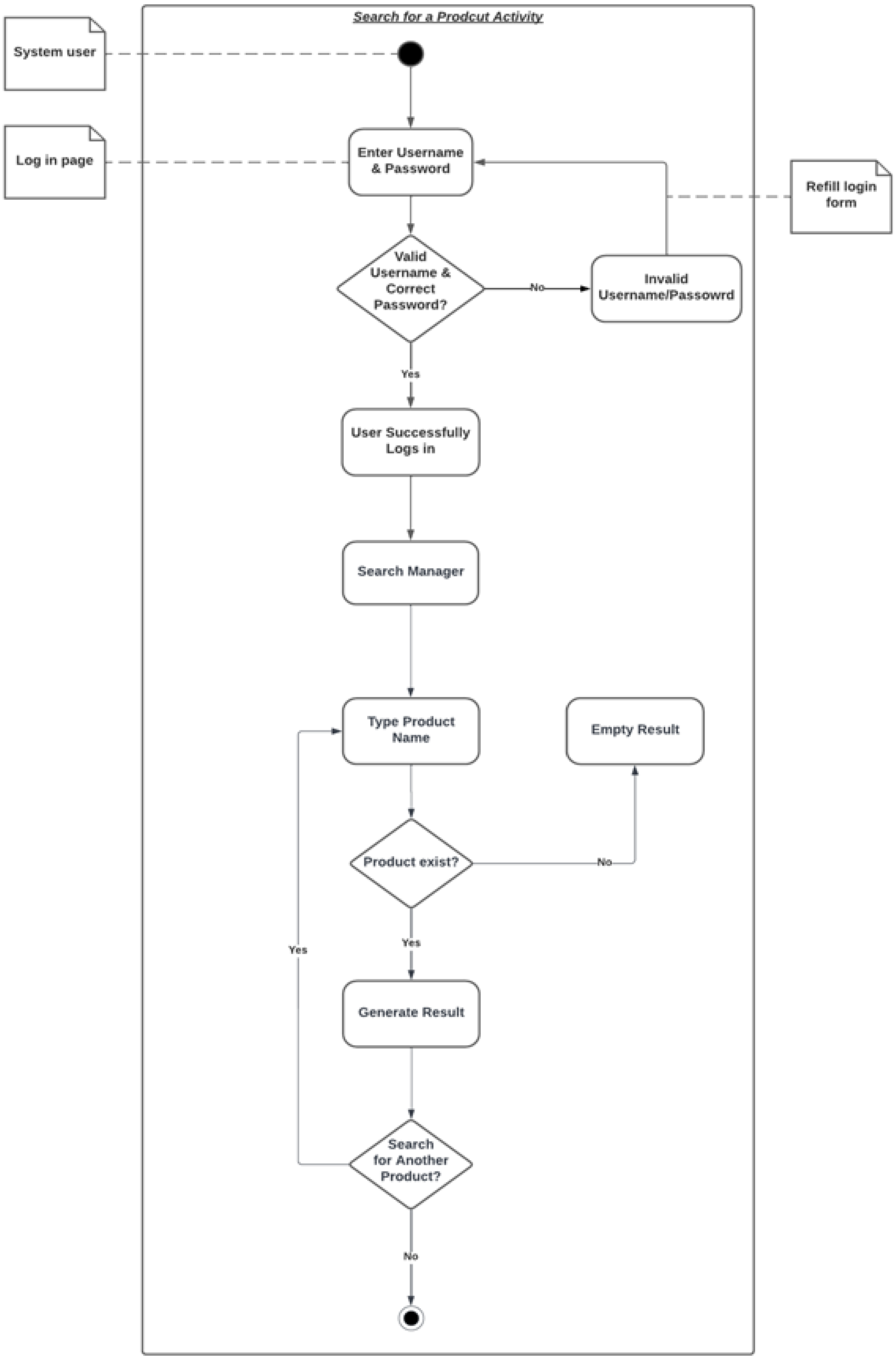
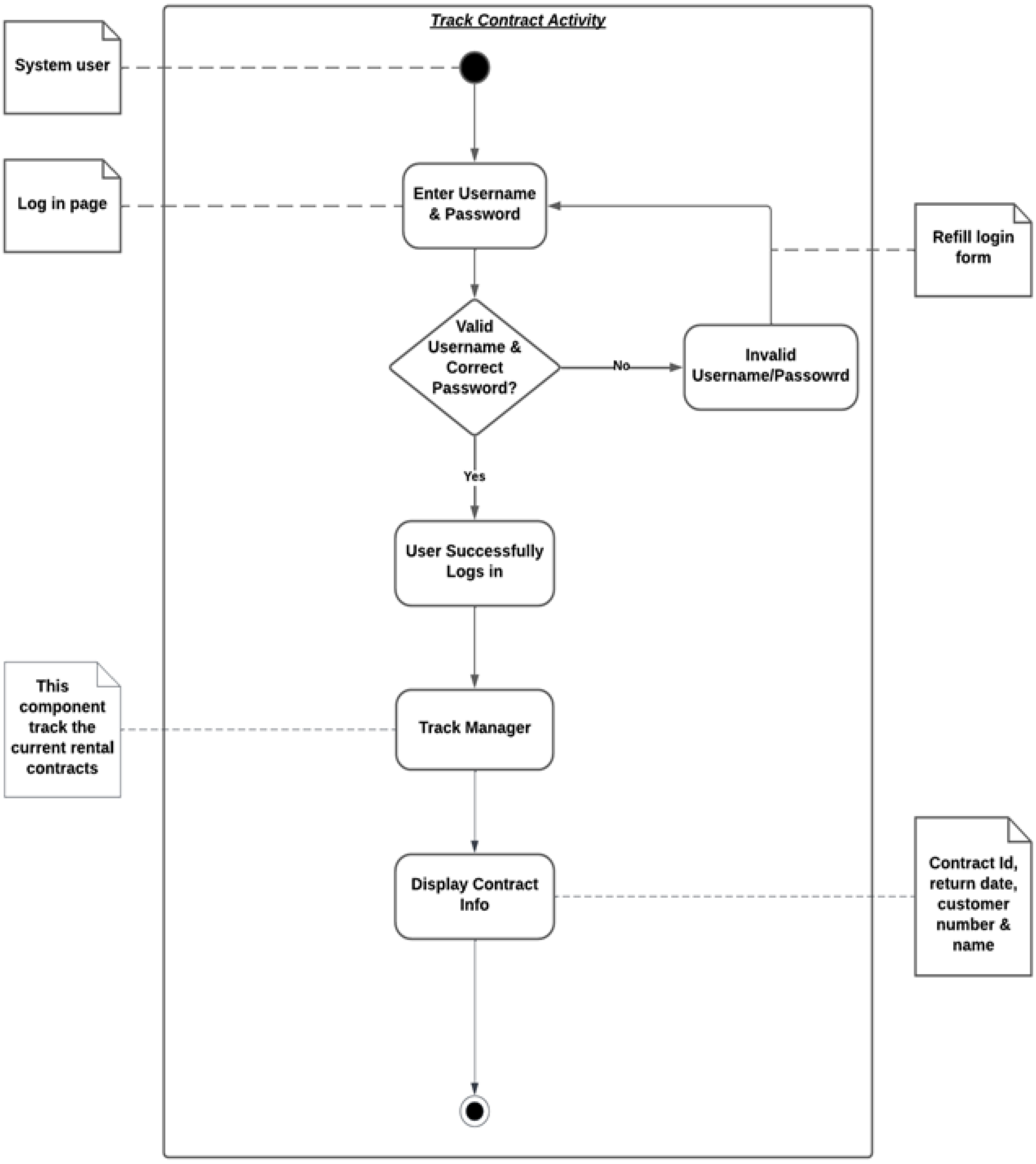

Search for a Prodcut Activity
System user
Log in page
Refill login form
Enter Username & Password
Valid Username & Correct Password?
No
Invalid Username/Passowrd
Yes
User Successfully Logs in
Search Manager
Type Product Name
Empty Result
Product exist?
No
Yes
Generate Result
Yes
Search for Another Product?
No

Figure 1: Activity diagram for Search for a Product use case

Appendix 16



Track Contract Activity
System user
Log in page
Enter Username & Password
Refill login form
Valid Username & Correct Password?
No
Invalid Username/Passowrd
Yes
User Successfully Logs in
This component track the current rental contracts
Track Manager
Display Contract Info
Contract Id, return date, customer number & name

Figure 1: Activity diagram for Track Contract use case

Appendix 17

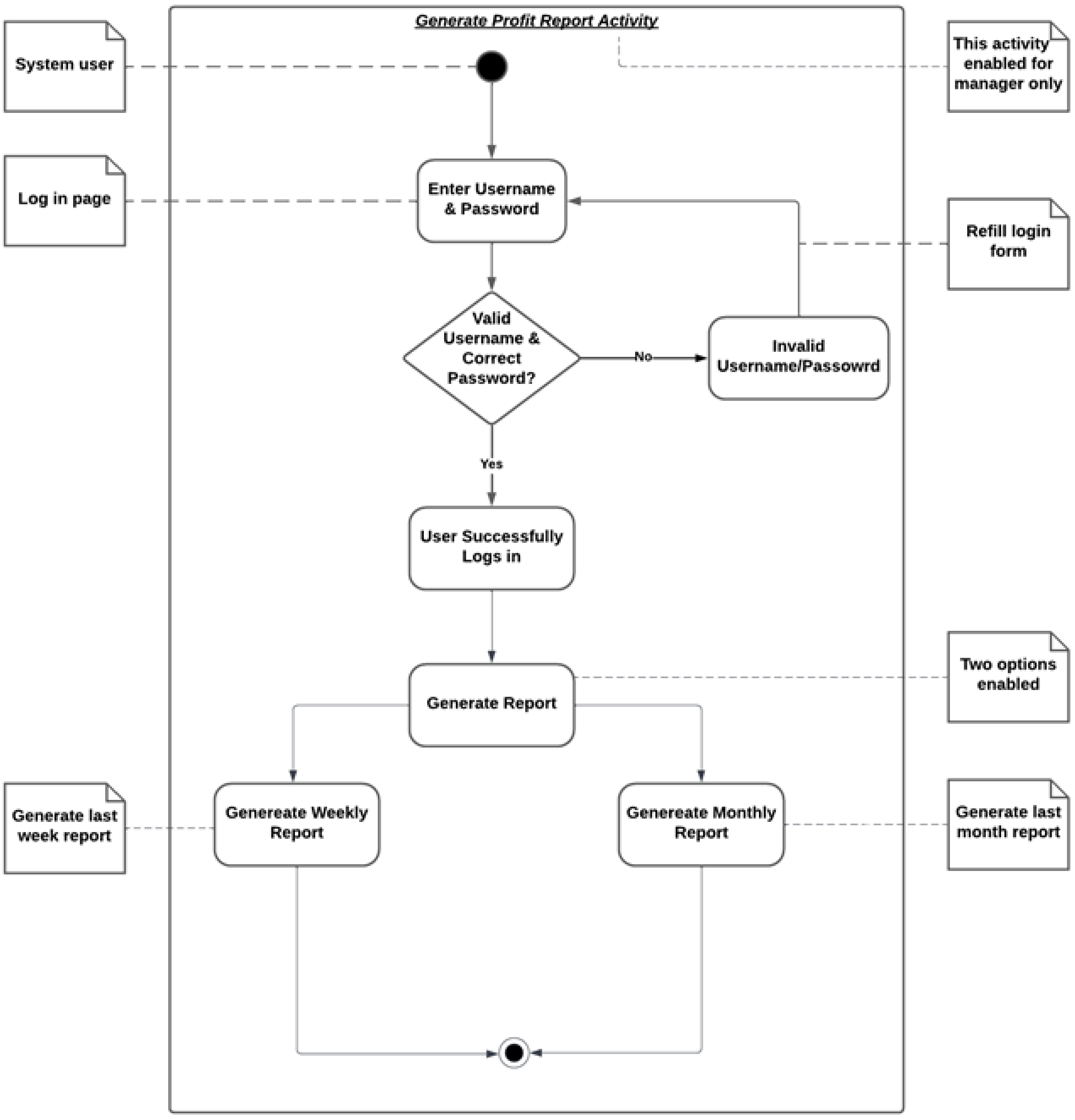

Generate Profit Report Activity
System user
This activity enabled for manager only
Log in page
Enter Username & Password
Refill login form
Valid Username & Correct Password?
No
Invalid Username/Passowrd
Yes
User Successfully Logs in
Generate Report
Two options enabled
Generate last week report
Genereate Weekly Report
Genereate Monthly Report
Generate last month report

Figure 1: Activity diagram for Generate Profit Report use case

Appendix 18

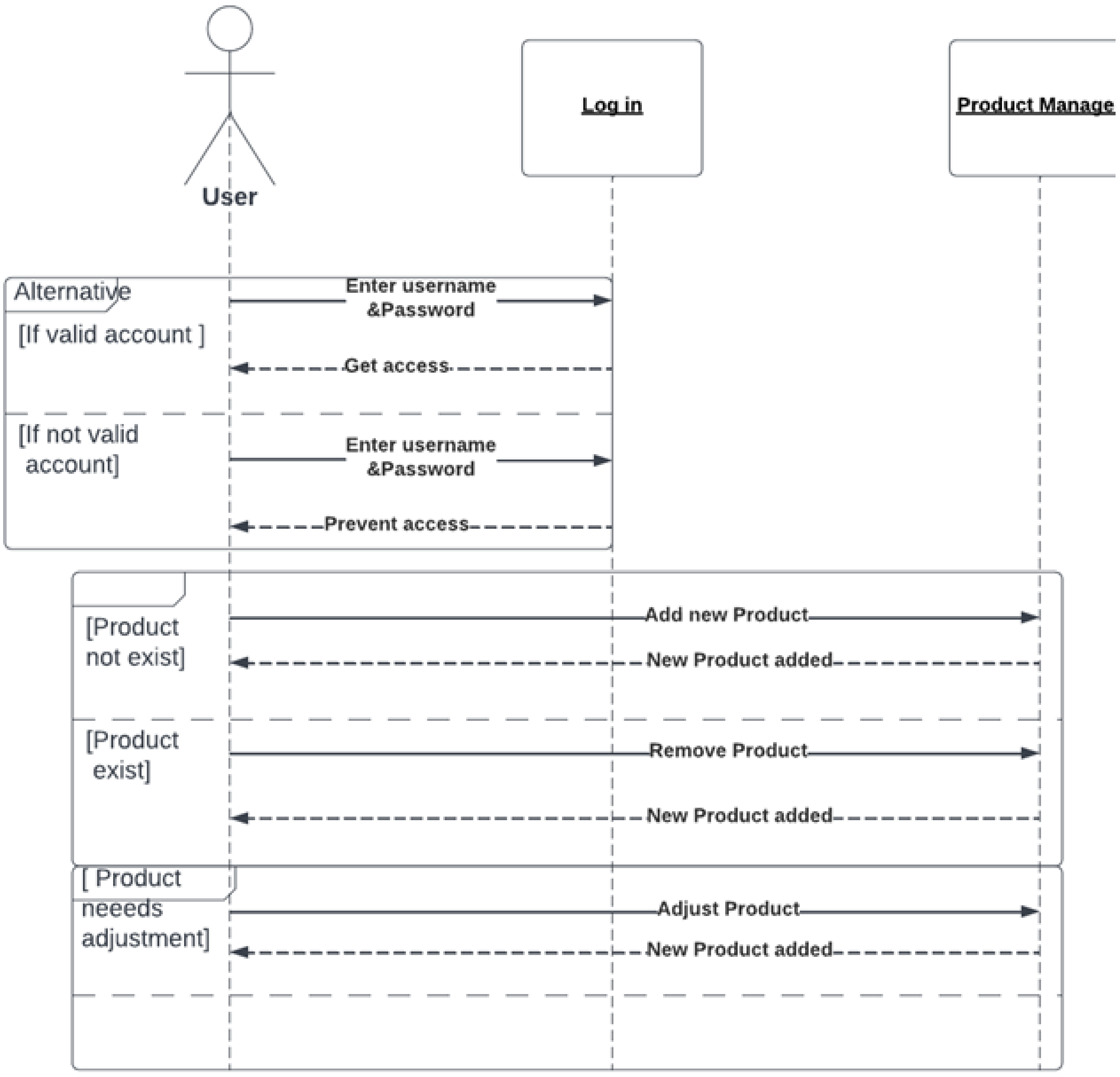


Figure 1: Sequence diagram for manage product use case

Appendix 19

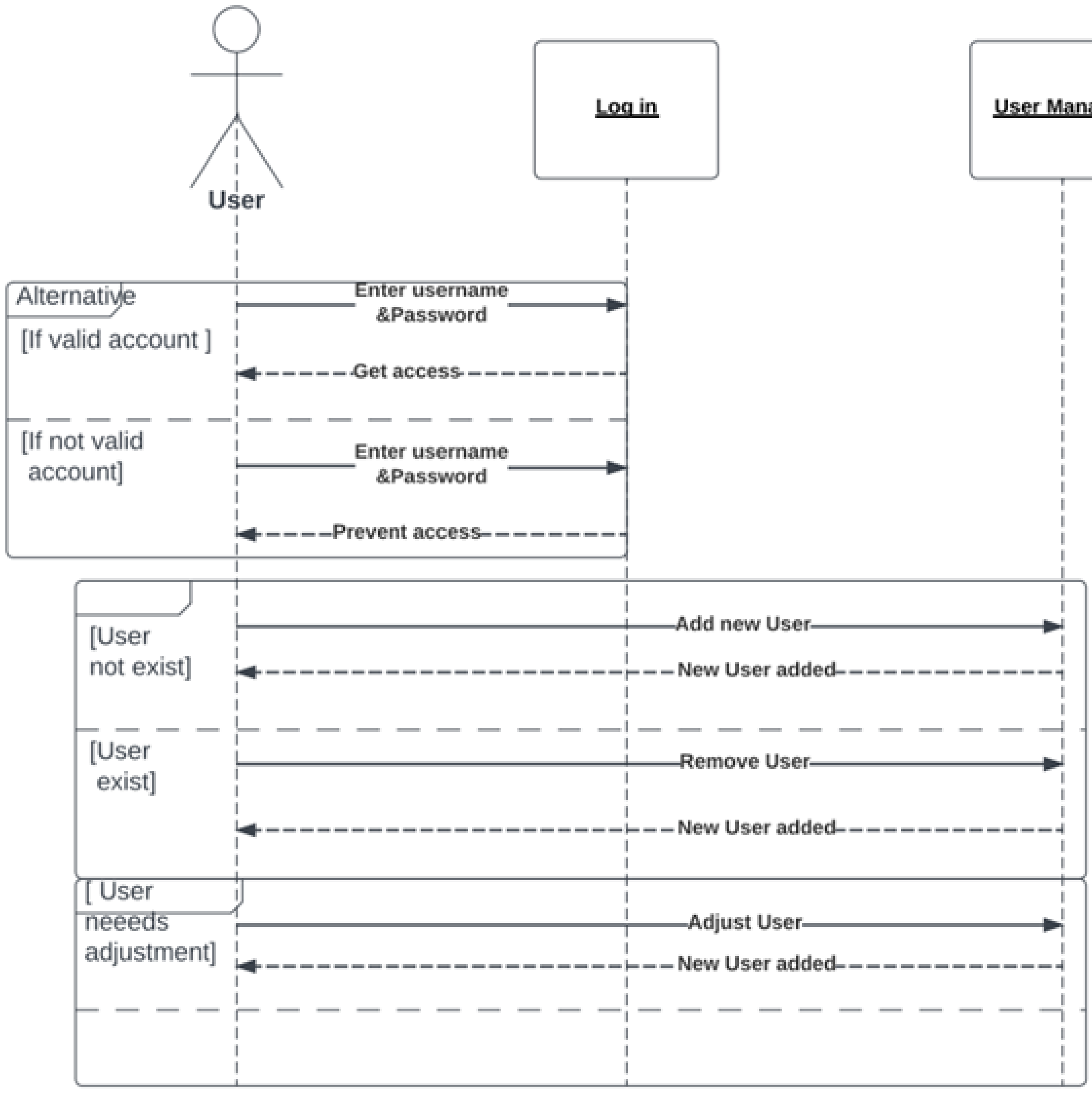


Figure 1: Sequence diagram for manage user use case

Appendix 20

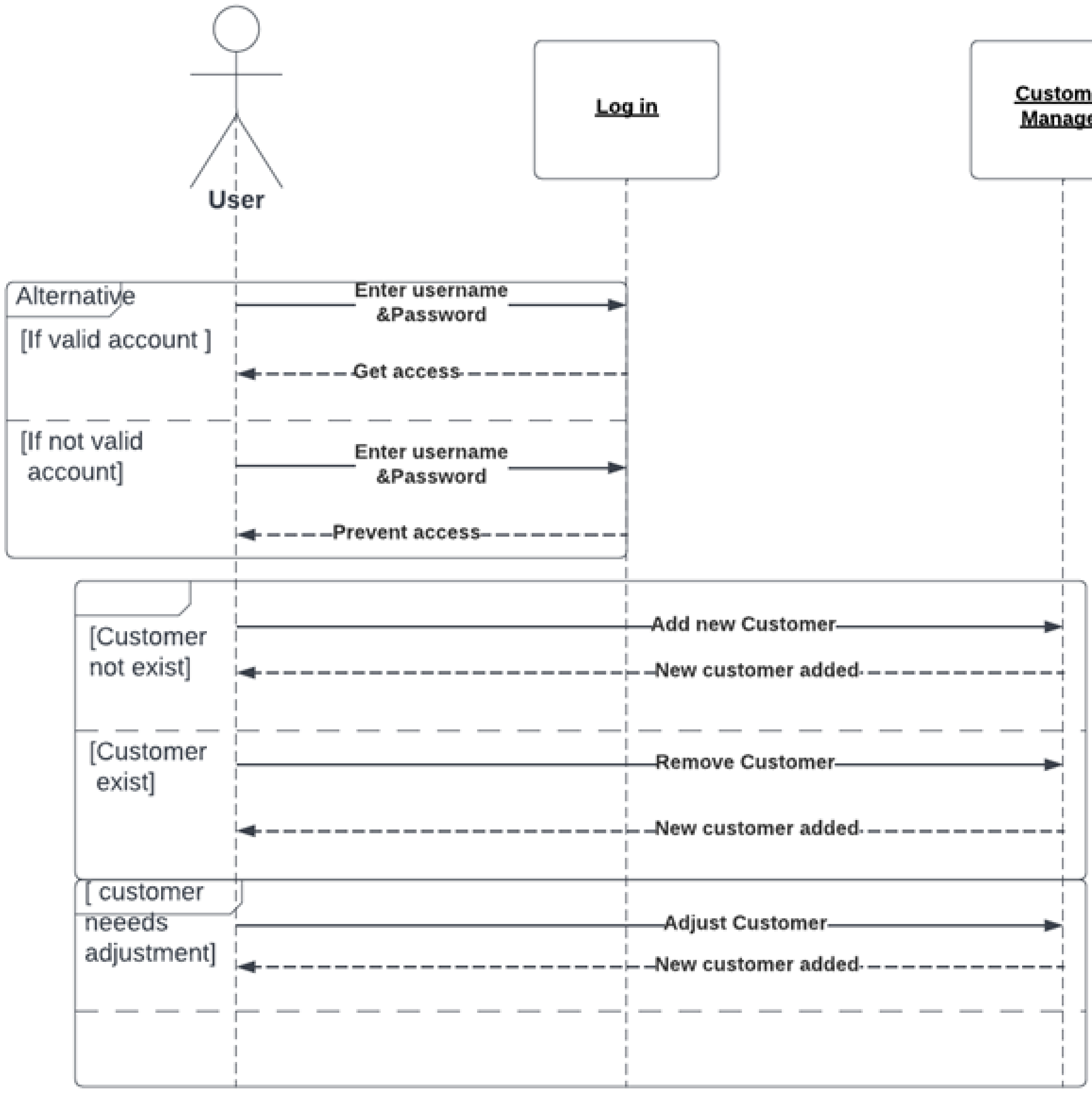


Figure 1: Sequence diagram manage customer use case

Appendix 21

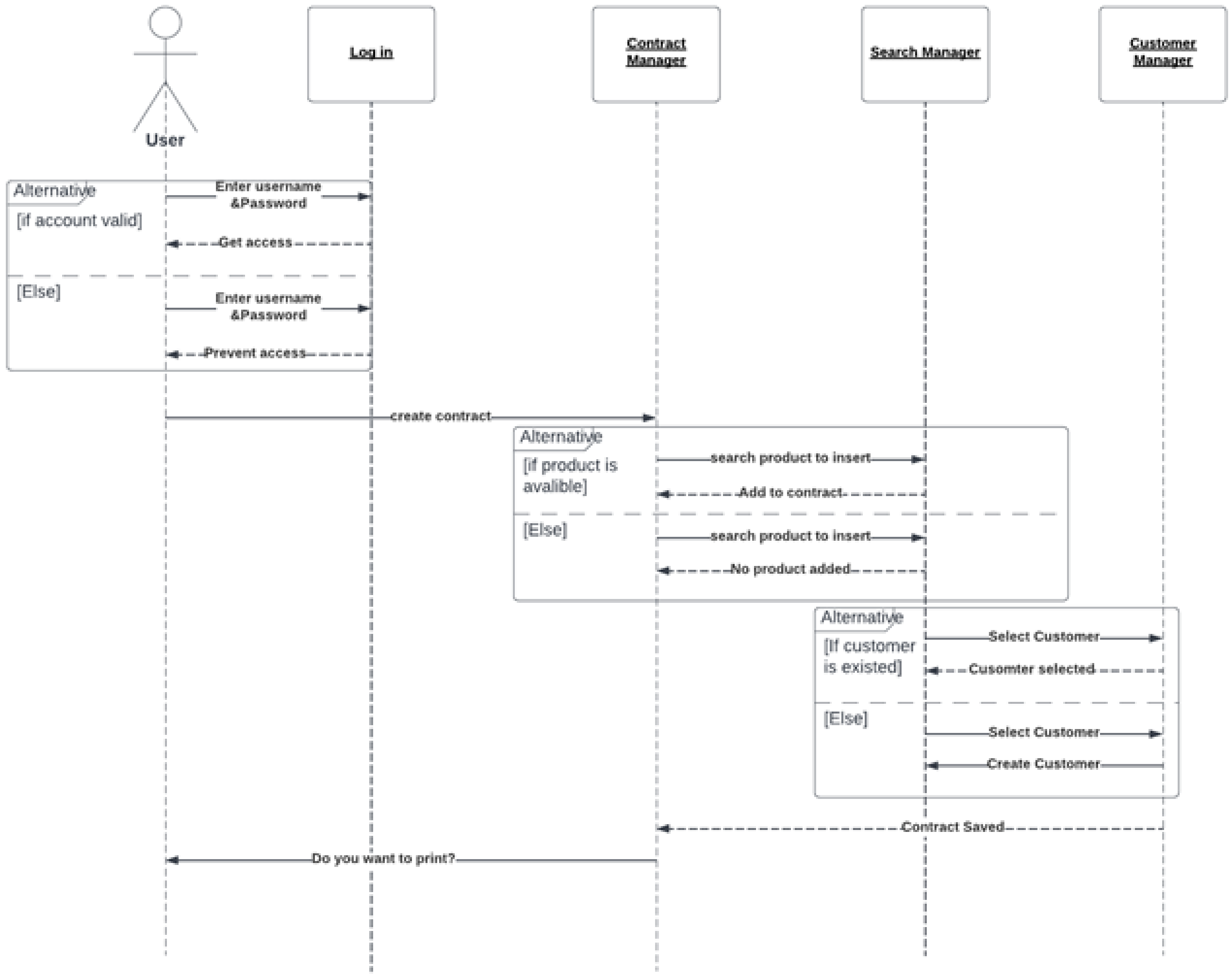
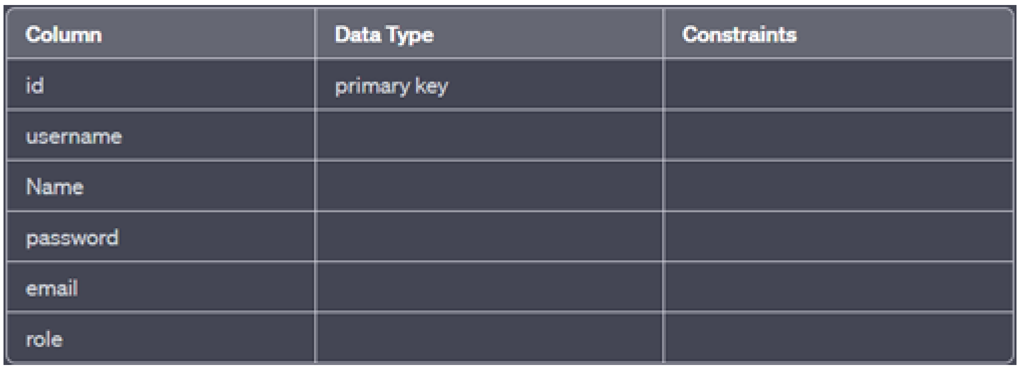


Figure 1: Sequence diagram for generate contract use case

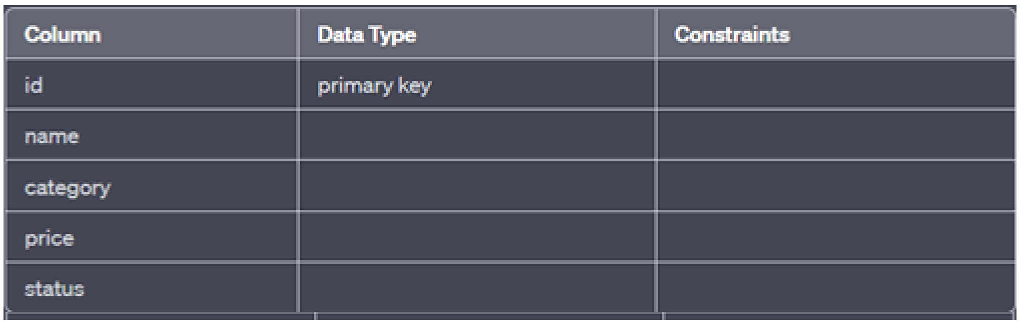
Appendix 22



Column	Data Type	Constraints
id	primary key	
username		
Name		
password		
email		
role		

Table 16: User table Schema

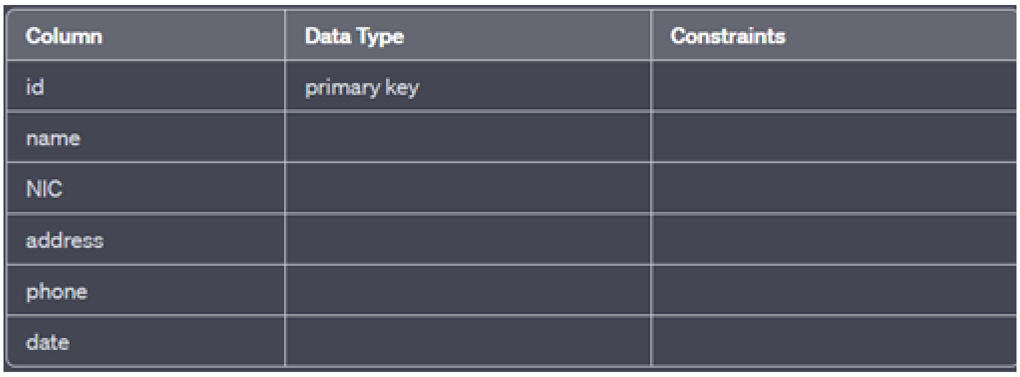
Appendix 23



Column	Data Type	Constraints
id	primary key	
name		
category		
price		
status		

Table 17: Product table Schema

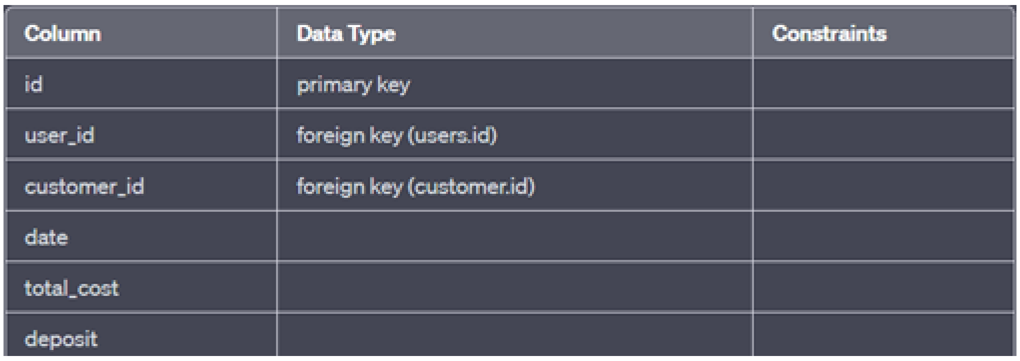
Appendix 24



Column	Data Type	Constraints
id	primary key	
name		
NIC		
address		
phone		
date		

Table 18: Customer table Schema

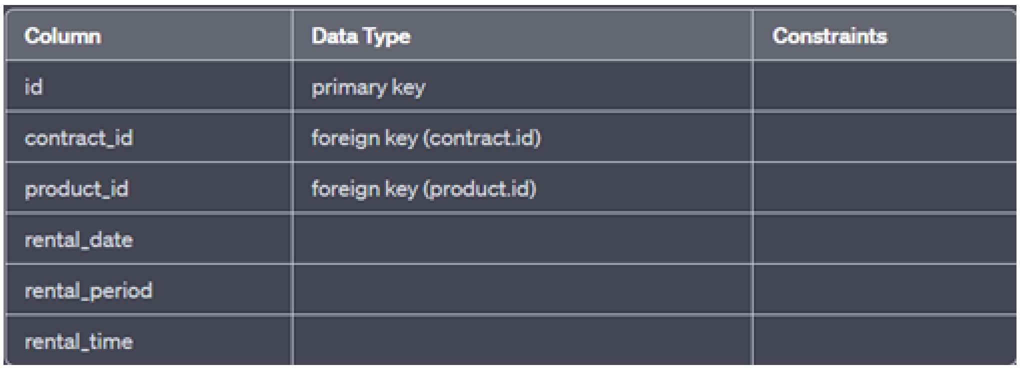
Appendix 25



Column	Data Type	Constraints
id	primary key	
user_id	foreign key (users.id)	
customer_id	foreign key (customer.id)	
date		
total_cost		
deposit		

Table 19: Contract table Schema

Appendix 26



Column	Data Type	Constraints
id	primary key	
contract_id	foreign key (contract.id)	
product_id	foreign key (product.id)	
rental_date		
rental_period		
rental_time		

Table 20: contract product table Schema

Appendix 27

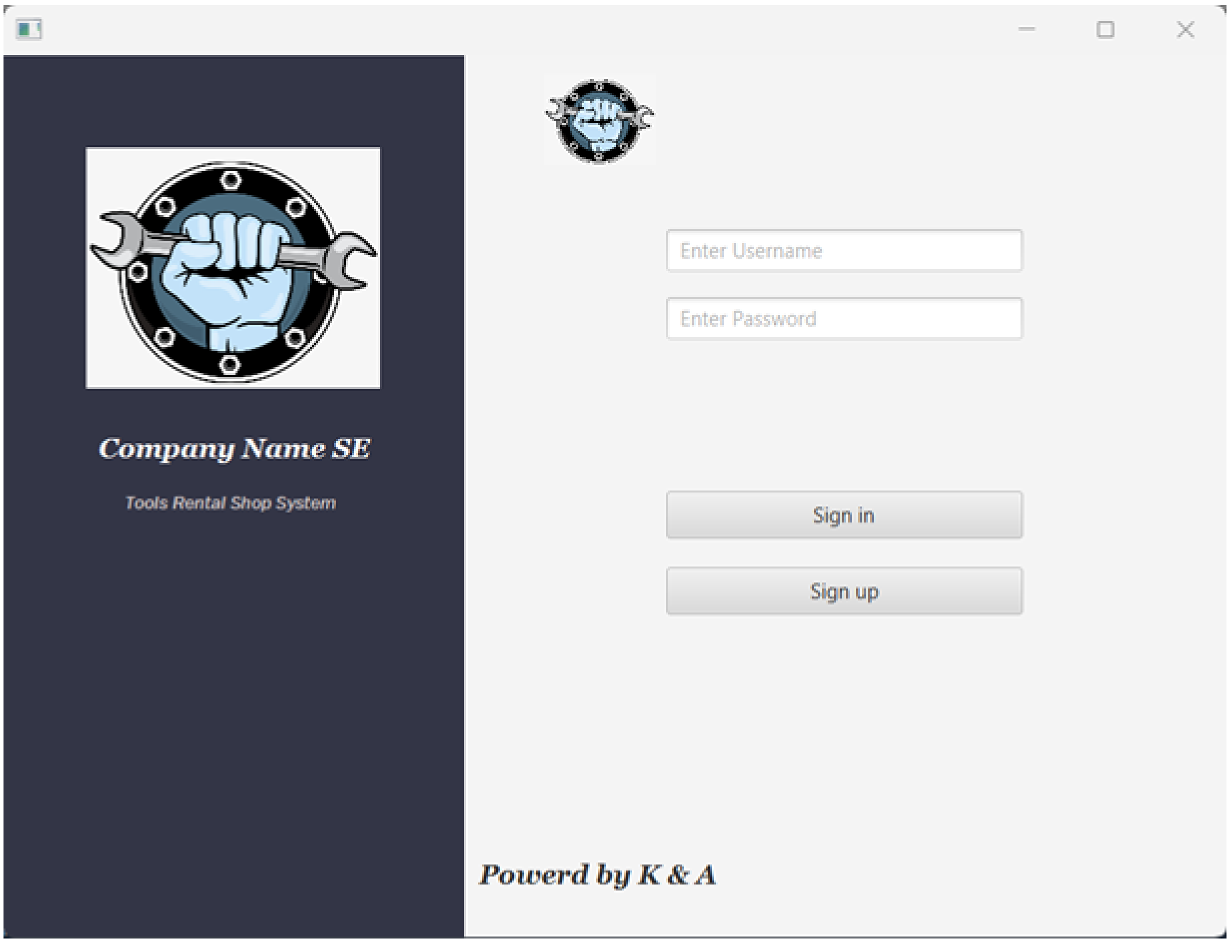


Figure 17:Login page of the RTS system.

Appendix 28

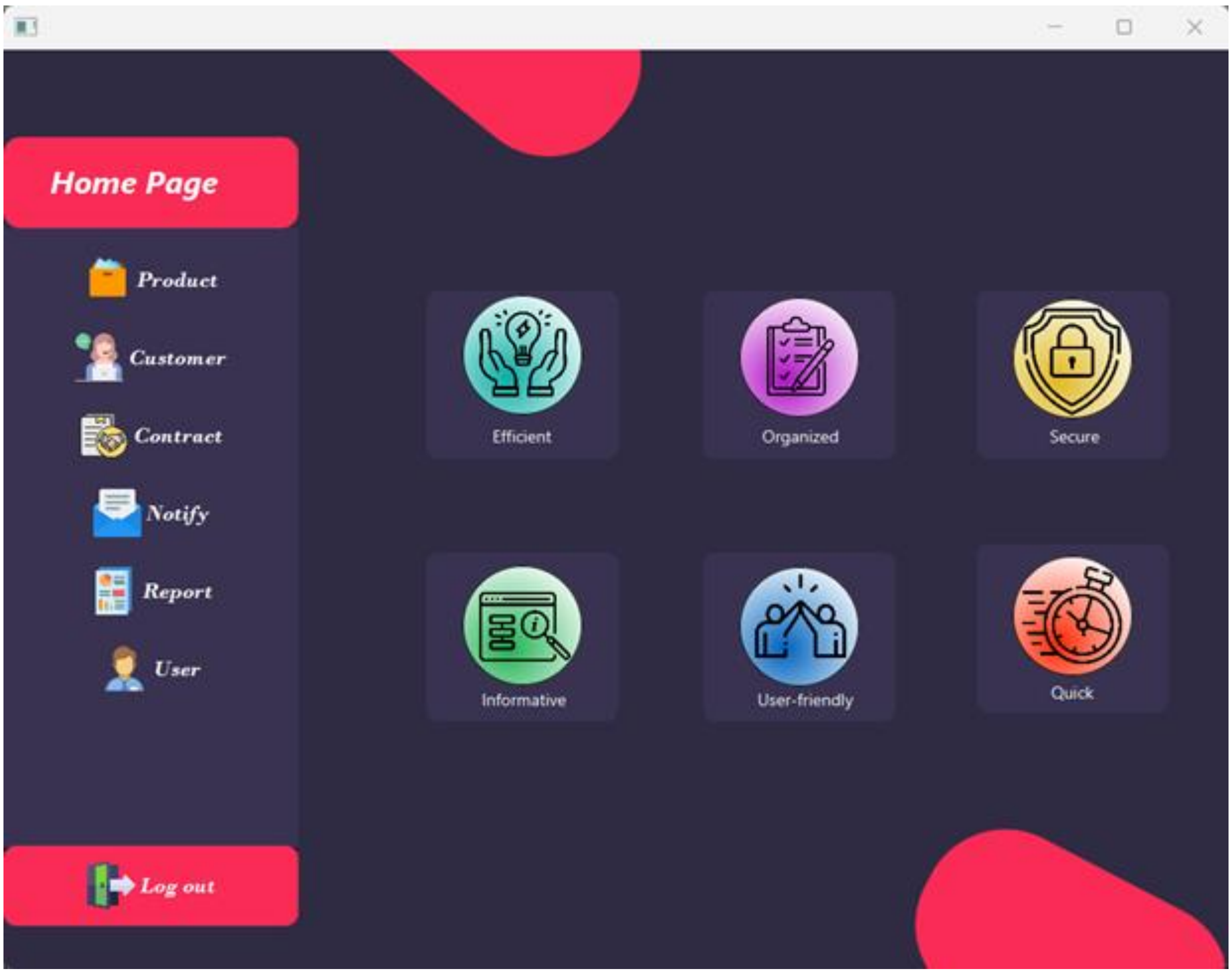
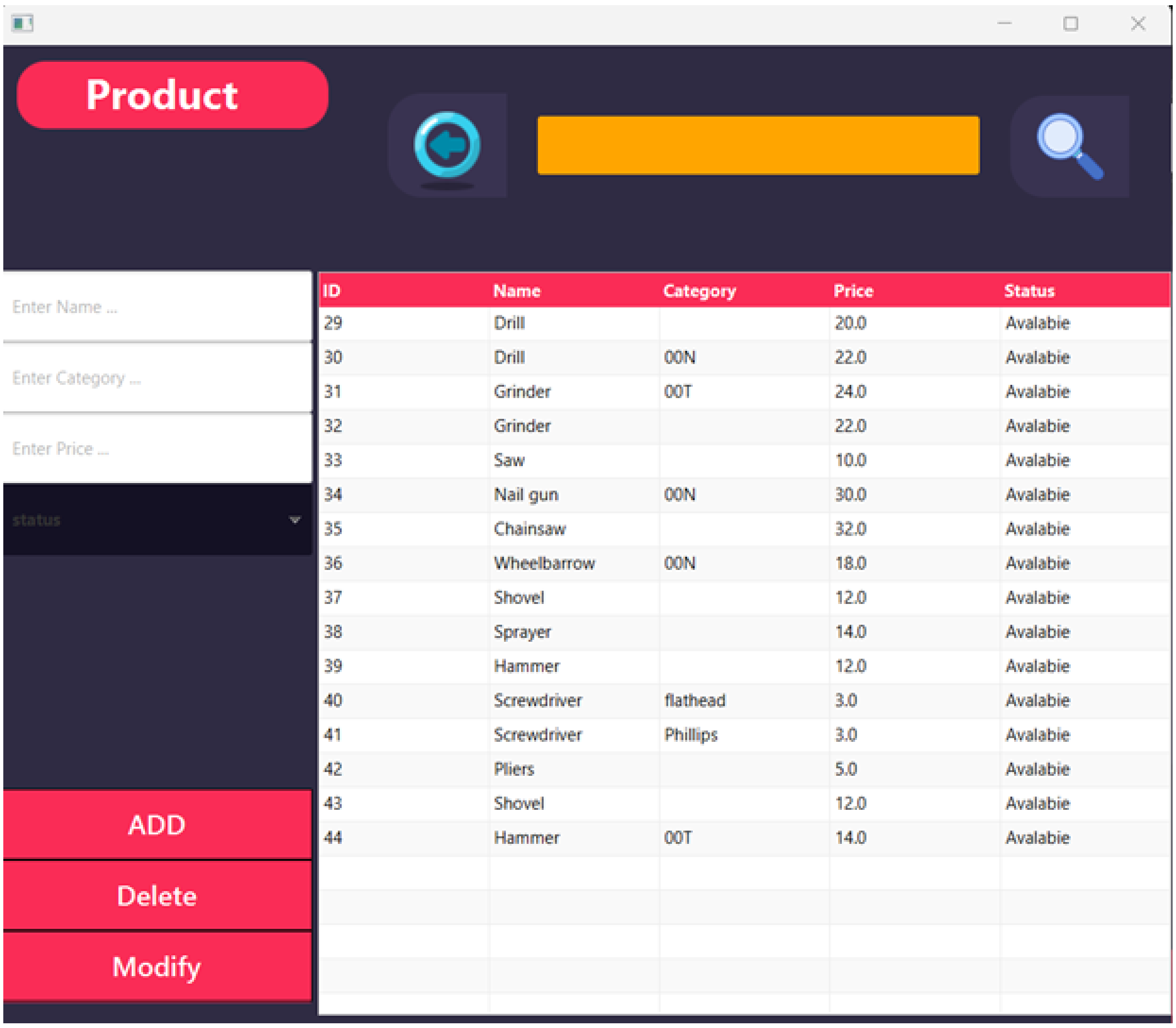


Figure 18: RTS system Home page

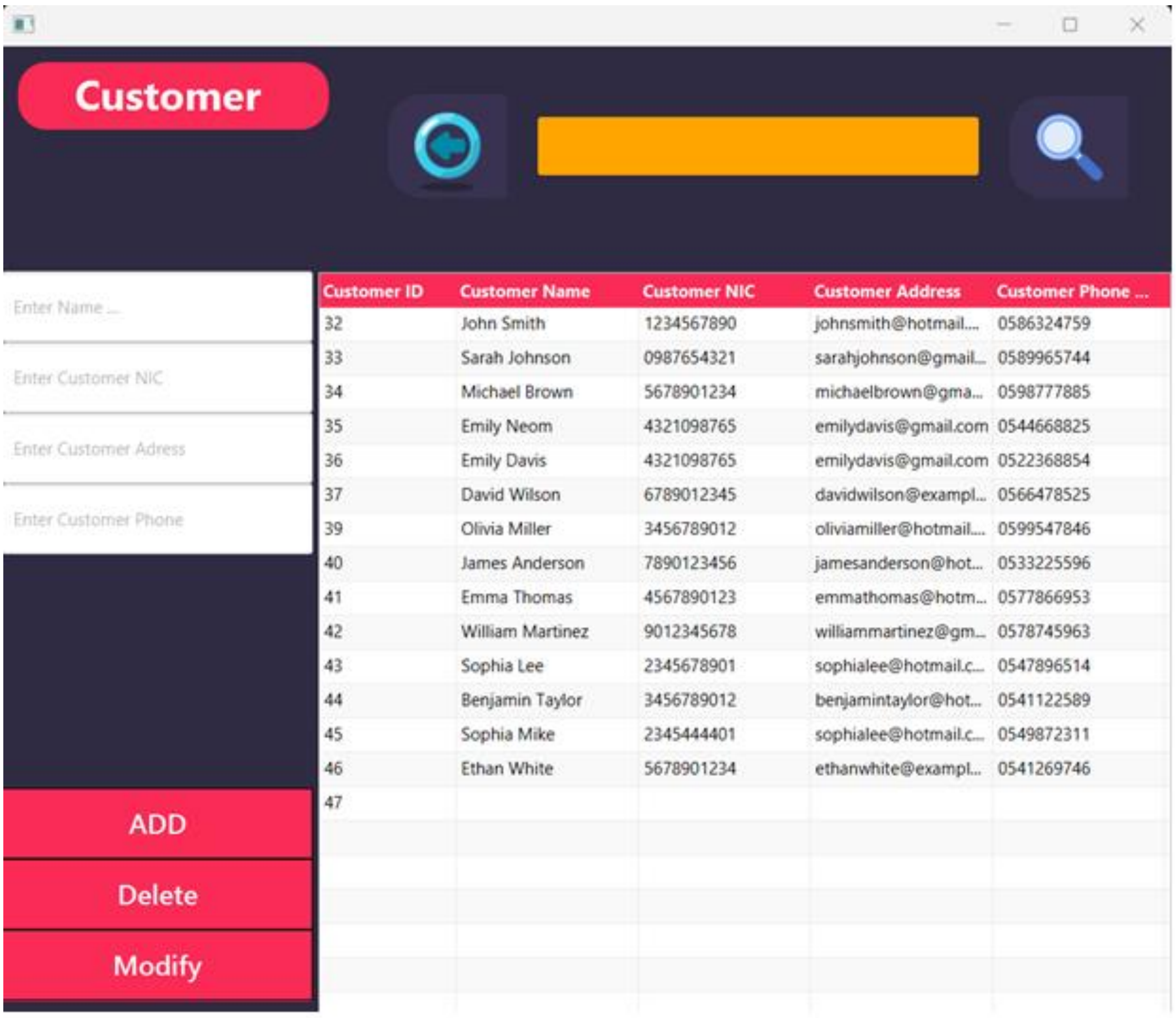
Appendix 29



ID	Name	Category	Price	Status
29	Drill		20.0	Avalabie
30	Drill	00N	22.0	Avalabie
31	Grinder	00T	24.0	Avalabie
32	Grinder		22.0	Avalabie
33	Saw		10.0	Avalabie
34	Nail gun	00N	30.0	Avalabie
35	Chainsaw		32.0	Avalabie
36	Wheelbarrow	00N	18.0	Avalabie
37	Shovel		12.0	Avalabie
38	Sprayer		14.0	Avalabie
39	Hammer		12.0	Avalabie
40	Screwdriver	flathead	3.0	Avalabie
41	Screwdriver	Phillips	3.0	Avalabie
42	Pliers		5.0	Avalabie
43	Shovel		12.0	Avalabie
44	Hammer	00T	14.0	Avalabie

Figure 19: RTS system Product Management Window

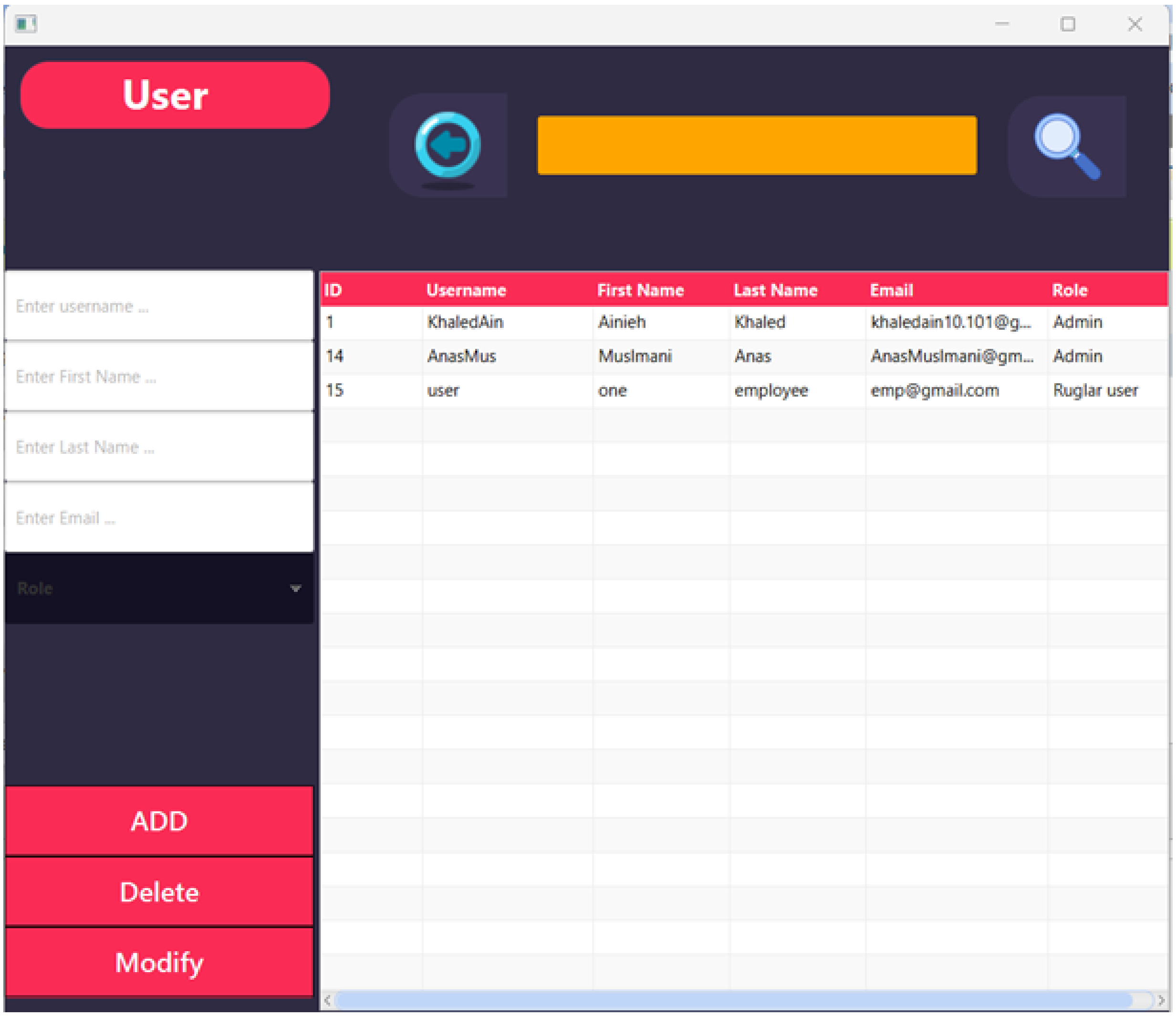
Appendix 30



Customer ID	Customer Name	Customer NIC	Customer Address	Customer Phone ...
32	John Smith	1234567890	johnsmith@hotmail....	0586324759
33	Sarah Johnson	0987654321	sarahjohnson@gmail...	0589965744
34	Michael Brown	5678901234	michaelbrown@gma...	0598777885
35	Emily Neom	4321098765	emilydavis@gmail.com	0544668825
36	Emily Davis	4321098765	emilydavis@gmail.com	0522368854
37	David Wilson	6789012345	davidwilson@exampl...	0566478525
39	Olivia Miller	3456789012	oliviamiller@hotmail....	0599547846
40	James Anderson	7890123456	jamesanderson@hot...	0533225596
41	Emma Thomas	4567890123	emmathomas@hotm...	0577866953
42	William Martinez	9012345678	williammartinez@gm...	0578745963
43	Sophia Lee	2345678901	sophialee@hotmail.c...	0547896514
44	Benjamin Taylor	3456789012	benjamintaylor@hot...	0541122589
45	Sophia Mike	2345444401	sophialee@hotmail.c...	0549872311
46	Ethan White	5678901234	ethanwhite@exampl...	0541269746
47				

Figure 20: RTS system Customer Management Window

Appendix 31



ID	Username	First Name	Last Name	Email	Role
1	KhaledAin	Ainieh	Khaled	khaledain10.101@g...	Admin
14	AnasMus	Muslmani	Anas	AnasMuslmani@gm...	Admin
15	user	one	employee	emp@gmail.com	Ruglar user

Figure 21: RTS system User Management Window

Appendix 32

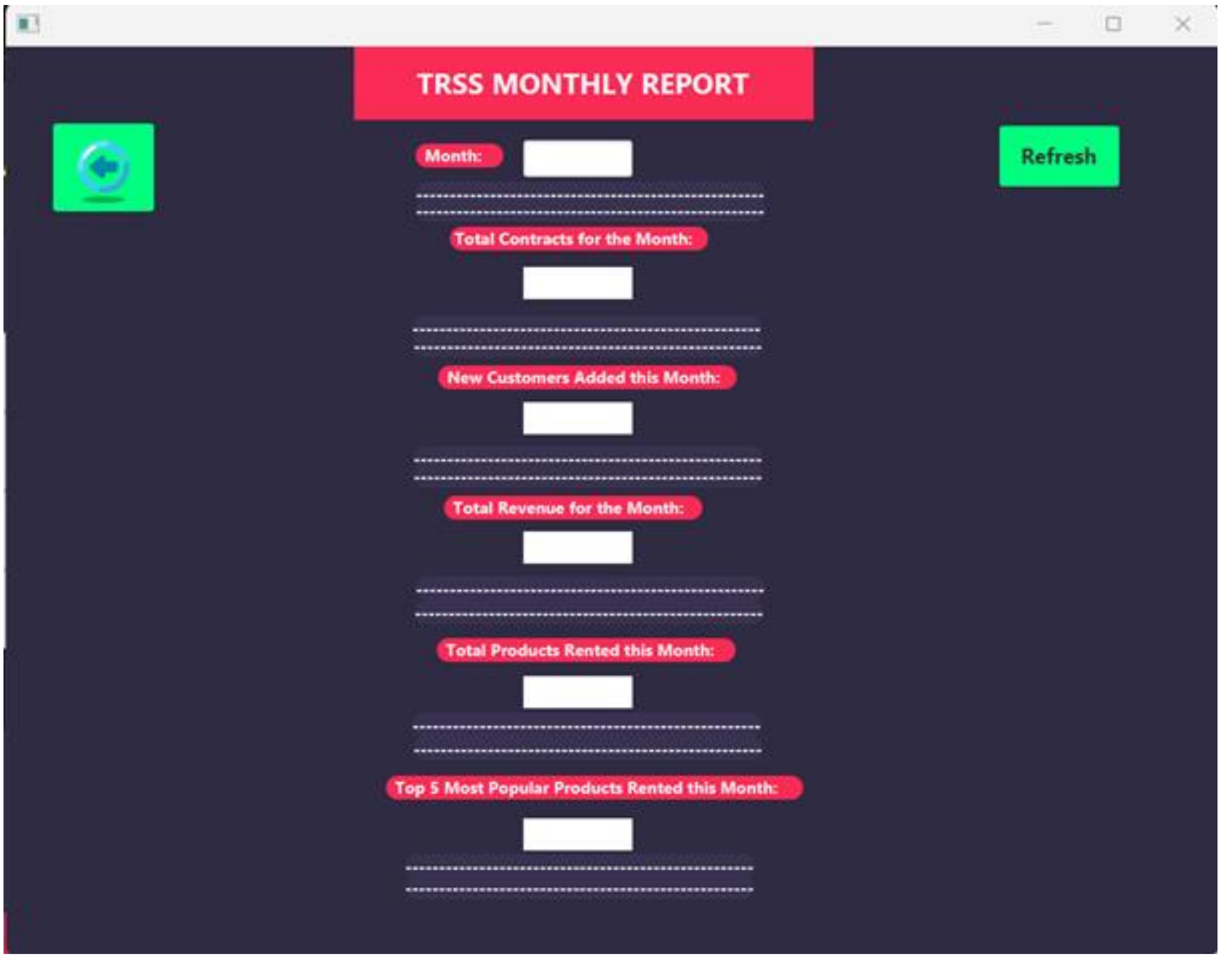
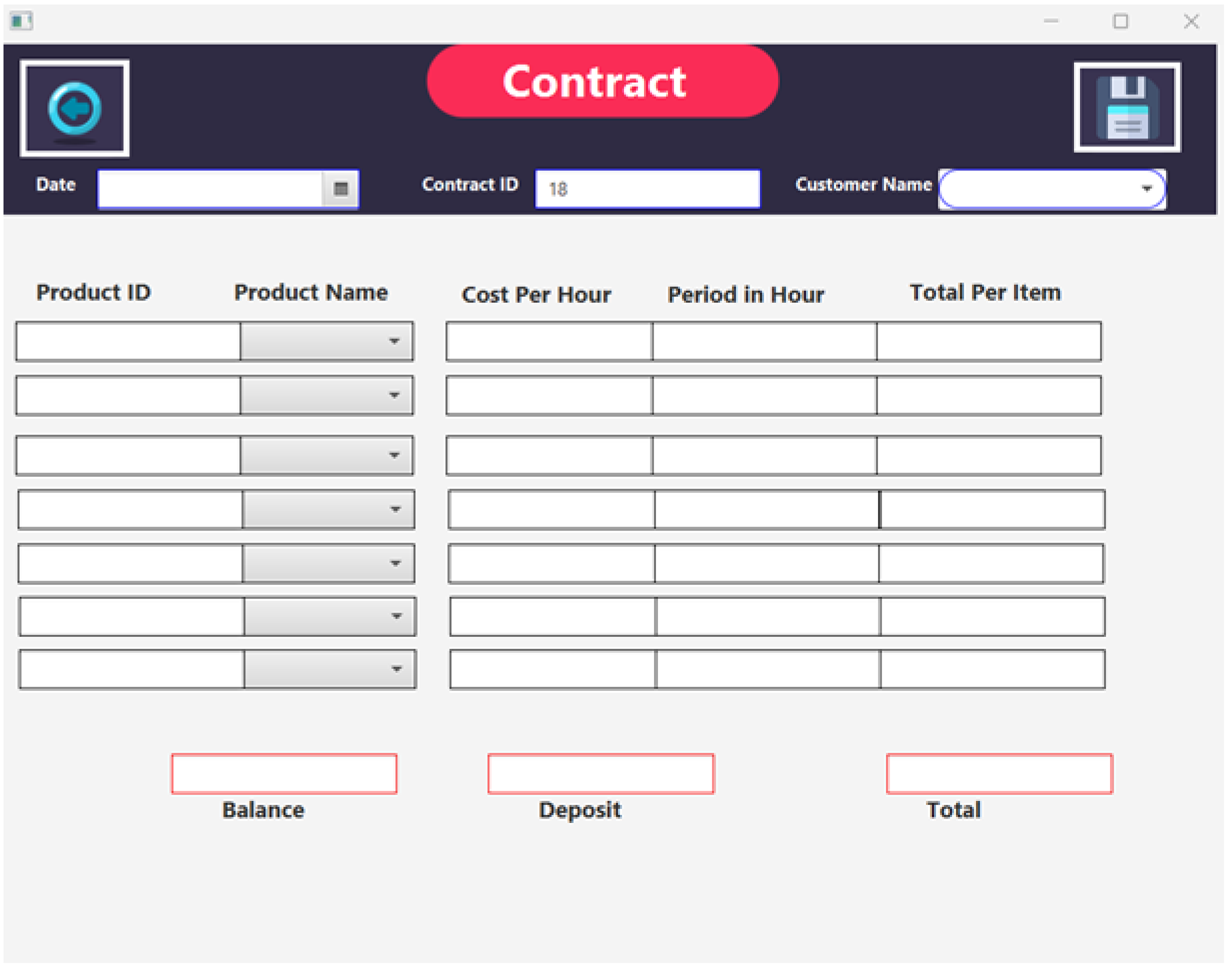

TRSS MONTHLY REPORT
Refresh
Month:
Total Contracts for the Month:
New Customers Added this Month:
Total Revenue for the Month:
Total Products Rented this Month:
Top 5 Most Popular Products Rented this Month:

Figure 22: RTS system Report Window

Appendix 33\



Date		Contract ID	18	Customer Name	

Product ID	Product Name	Cost Per Hour	Period in Hour	Total Per Item

Figure 23: RTS system Contract Management Window