Assignment 3 – Design & Implementation

Phone Repair Store

Software Engineering & Testing

Burair B00138007

Benjamin B00156009

Samuel B00150540

A

Project Definition

The aim of the project is to help our clients set up an online platform through which they're able to offer a variety of repair services to phones, directly to consumers. The main problem in hand being tackled is to bridge the gap between, but also provide a seamless straightforward experience for the customer. The phone store aims to make fixing phones easier and better for both customers and store employees by addressing issues such as long waiting times, which may lead to poor customer service. By improving this issue, the store anticipates an increase in customers and revenue.

Functional Specifications

Appointment Scheduling: Users should be easily be able choose a service they need and book it

Customer Database : Maintain records of repairs and customer details to go along with it, allowing it to be more personalised and structured. Holds key information such as contact details

Product Viewing : Users are able to easily browse and view available repair services, along with each of its own associated descriptions and pricing

Repair Details Selection: Allow users to seamlessly specify the details of the repair they are choosing ie: the phone model

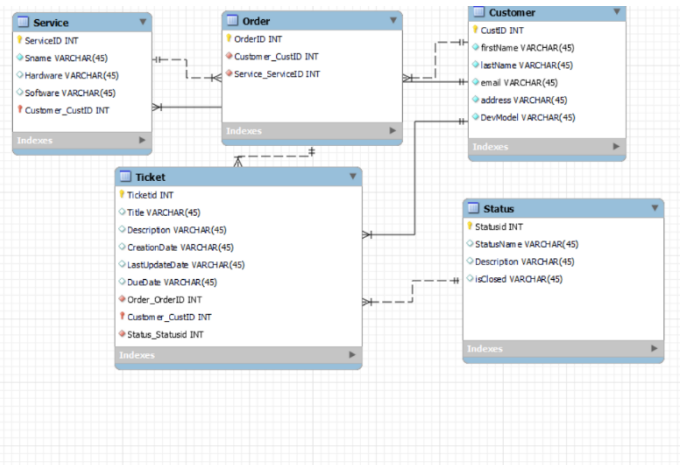
Registration / Login: Allow users to create their own account to allow for a easier method of order tracking and seeing the current status

Methodology

In order to ensure a clear and concise delivery of the Phone Repair task, designs and certain methodologies have been planned out. Doing so allows for us to make realistic well timed progress.

System Models

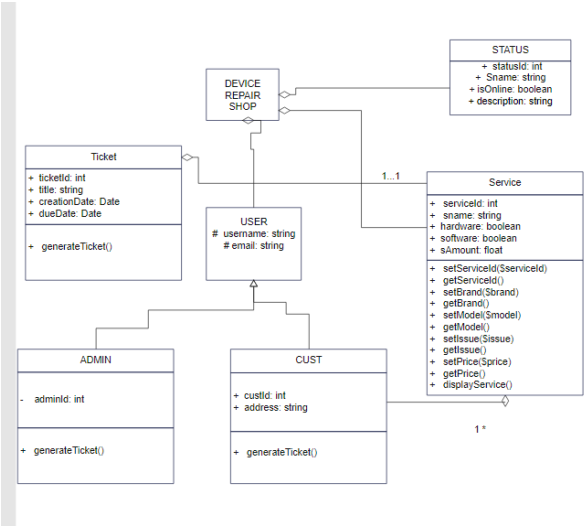
Entity Relationship Diagram : the ERD was utilized to create a model of the database structure to be used to help set up the entire backend for the project. Its main purpose was to help establish the various entities, attributes and relationships behind storing customer information, repair records, service details and the current status of tickets.



ERD Relationships

- Customer-Ticket Relationship: Associates each customer with their respective service tickets, allowing for personalized tracking of service history and requests. This relationship enables efficient customer support and targeted service recommendations.
- Service-Order Relationship: Links repair services with customer orders, facilitating seamless order processing and inventory management. This connection ensures accurate service delivery and optimized resource allocation.
- Ticket-Status Relationship: Connects service tickets to their current status, providing real-time updates on repair progress. By tracking ticket statuses, the system ensures transparency and timely communication with customers.

Class Diagrams : The class diagram allowed for visual representation to be made of the structure that the project is based on. Helping make it easier to understand how the different components fit together, what we needed to add, different interactions within and allow for OOAD to be applied. Overall, it allows for all components to be modularized and simply maintained throughout the software development process.



Decisions based on Class Diagram

Relationships and Multiplicities : The connection between ticket and service is presented by a diamond symbol on the Ticket side, indicating that each Ticket may be associated with one or more Services.

Associations: An association exists between the Service class and the Ticket class to capture the relationship where a service can be associated with tickets. This association is crucial for tracking service requests and their corresponding tickets.

Generalizations (Inheritance): The generalization relationship between the Admin class and the User class signifies inheritance, indicating that an admin is a specialized type of user. This design decision promotes code reuse and maintains a clear hierarchy in our system.

Aggregations (Compositions): Within the Ticket class, the aggregation relationship with the Service class represents the composition of a service within a ticket. This aggregation ensures that service details are encapsulated within each ticket, allowing for effective management and tracking.

Usage of OOAD

Ensuring OOAD the implementation was essential throughout the design of the Phone Repair Store system. Doing so helped segregate and break it down into much more manageable parts. Which directly helped us be more organized in the code and its implementation and avoid unorganized code. Encapsulating it into parts ensured modularized, easier for maintenance and future changes.

Static VS Dynamic Case Diagrams

Dynamic case diagrams highlight the sequence of actions that are expected to take place while the software is being run, helping understand the interactions taking place at different stages by the actors and components ie. Customer in our case. By highlighting the various expected paths, it makes clear what the systems behaviour should be and help make the implementation and communication much more fluent between all members throughout the entire process. In our case, visual representation covered components such as confirming user details, searching for ticket details or viewing item progress allowing for guided decisions. This approach was much more ideal over static case diagrams due to the dynamic view of user interactions and systems responses, which benefits by providing a much more detailed/specific understanding of behaviours/requirements.

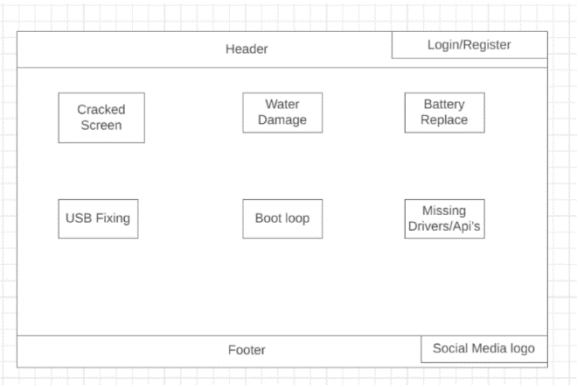
Purpose of Using Classes

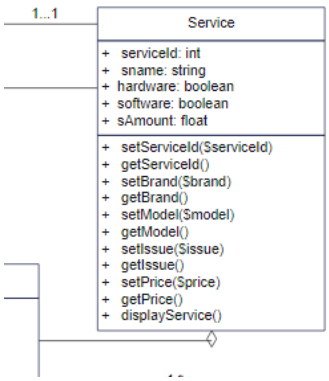
Making sure to use classes throughout our project is influenced mainly by the ability to further structure/organize and modularize all the code. Encapsulating into smaller much more easier to manage blocks also helps maintain and refer back later on. It also helps to apply opp concepts such as encapsulation, inheritance and polymorphism. In terms of the Repair Store project, classes were used to cover areas like customers, services, tickets and statuses.

User Interface

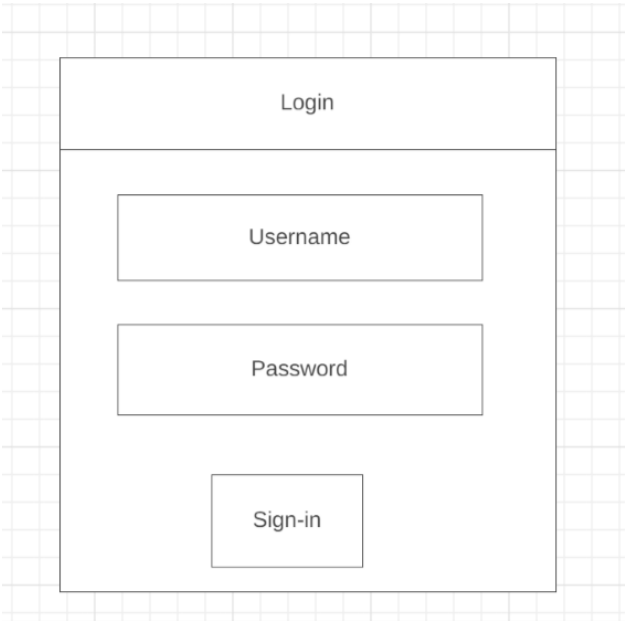
Having a user interface template created prior to starting anything has had a significant impact on the execution of the functional specifications along with almost every other key decision since its essentially a point of referral of what the ideal result is. While the front-end development is still in its early stages, having the design itself in mind has played a major role in the back end side of the project. Mentioned within the functional spec was product viewing or register/login as an example – pairing that with the interface draft we had decided on affected how or what was added within other diagrams such as the erd in addition to the actual code itself.

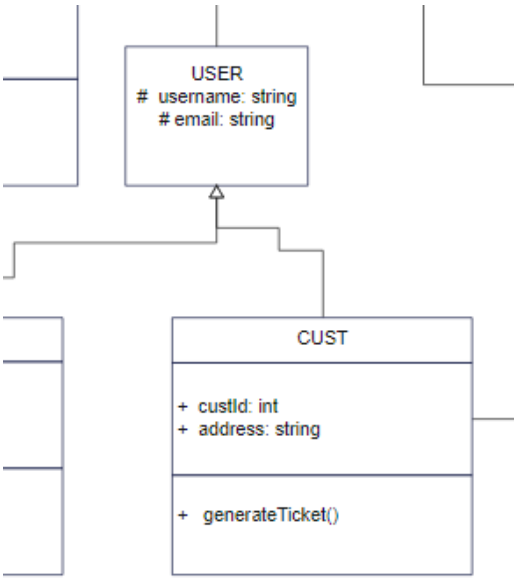
Services Page Template :





Login Page :





Non-functional specs also highlighted other key areas influenced by the templates themselves to ensure are executed such as over all performance – avoiding overloading the page for quick loading times, usability – easy enough for any new person who isn't too familiar to navigate, appearance – maintain as consistently as possible through the entire user experience or the adaptability of everything in differencnt sizes .

Requirements – Use Case Specifications

Login:

- Registration: The user registration process influenced the design of the User class, including attributes such as username, email, and password. Methods for user registration and validation were developed based on these requirements.
- Login: The login functionality drove the implementation of authentication methods to verify user credentials against stored data in the database.

Create a Ticket / Request

- Ticket Generation: The process of generating tickets for service requests influenced the design of the Ticket class and methods for generating ticket numbers and associating them with user orders.
- Database Integration: Database tables were designed to store ticket information, including ticket numbers, associated users, service details, and current status.

Order Service

- Service Selection: Use case specifications helped in choosing how to develop methods for selecting repair services and providing relevant phone details for service orders.
- Ticket Generation: Similar to the "Create a Ticket" use case, ticket generation functionality was integrated into the service ordering process to ensure proper tracking and management of service requests.

Track Ticket

- Ticket Search: Methods for searching and retrieving ticket information from the database were implemented based on the requirements specified in the use case.
- Progress Tracking: Functionality for tracking ticket progress and updating status information was developed to provide users with real-time updates on their service requests.

By aligning each use case specification with the corresponding classes, methods, and database tables, we're able to ensure that the system design as we continue to develop it further will reflect the functional requirements of the application that have been outlined at the start. Taking this approach helps our group to keep track of the different progression stages and avoid deviating from the main goal

Conclusions

Overall this project document has outlined the design and implementation process for the Phone Repair Store system that has been completed so far by the group as well what the plan to progress further looks like. Our requirement specifications, use case and functional specifications, have defined the end system's scope and expected functionality.

On the other hand, models like our class diagram and ERD assisted in visually nothing the architecture to follow.

In summary, this document serves as a guide for further developing the Phone Repair Store system with the expectation in the end being to fully satisfy goals set early on at the start of the project and client requirements/expectations