Software Engineering - Iteration 4

QuickLocalFix Repair Services

Team Name: Phoenix

Team Leader: Srinivas Makkena

Team Members:

Divya Podila

Srinivas Makkena

Venkata Sai Prakash Boddu

01

Architectural Design Objectives

1. **Modularity:** Design the system with modular components to facilitate easy maintenance, updates, and scalability.

2. **Security:** Implement robust security measures to protect user data, payment transactions, and sensitive information from unauthorized access or breaches.

3. **Performance:** Optimize system performance to ensure fast response times, smooth user experience, and minimal downtime.

4. **User Experience:** Prioritize user-centric design principles to provide an intuitive and seamless experience for both customers and service providers.

5. **Compliance:** Ensure compliance with relevant regulations and standards related to data privacy, security, and online transactions.

02

Architectural Design Considerations

1. **Component-Based Architecture:** Utilize ReactJS's component-based architecture to build modular, reusable UI components. This approach promotes code reusability, maintainability, and scalability.

2. **Client-Server Communication:** Use RESTful APIs for client-server communication. RESTful APIs provide a standardized, stateless communication protocol.

3. **Database Design:** Design the PostgreSQL database schema efficiently to store and retrieve data effectively. Normalize the database schema to minimize redundancy and maintain data integrity, while also considering denormalization for performance optimization when necessary.

4. **Error Handling and Logging:** Implement robust error handling mechanisms to gracefully handle exceptions and provide meaningful error messages to users.

5. **Authentication and Authorization:** Implement secure authentication and authorization mechanisms using industry-standard protocols like OAuth 2.0 or JWT (JSON Web Tokens). Ensure proper validation and sanitization of user input to prevent security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks.

03

Identifying the System

- **Explain the type of system that you are implementing?**

QuickLocalFix website can be classified as a Service Management System.

- **Why did you classify it as such system?**

- Users can request services such as repairs, maintenance, or other assistance, while professionals can offer their services to fulfill these requests.

- The system acts as a platform for connecting users with service providers, managing service bookings, and facilitating communication between both parties.

- It involves the organization and delivery of services to meet the needs of users efficiently.

04

Subsystems

a.) List of all subsystems

1. User Management Subsystem
2. Service Management Subsystem
3. Professional Management Subsystem
4. Rating and Review Subsystem
5. Payment and Invoicing Subsystem
6. Product and Cart Subsystem

1. User Management Subsystem:

- Type: Interactive
- Explanation: This subsystem allows users to register, log in, and manage their accounts.

  It involves a two-way communication between users and the system.
- Layers:

- Presentation layer (UI for registration and login)
- Application layer (business logic for user authentication and account management)
- Data layer (database for storing user information)

2. Service Management Subsystem:

- Type: Interactive
- Explanation: Service management involves continuous interaction between users and the system. Users request services and report issues, which triggers responses and actions from the system. This interactive nature is central to managing services effectively.
- Components: Service Request Handling, Service Level Management, Service booking functionality.

3. Professional Management Subsystem:

- Type: Interactive

- Explanation: Enables professionals to register, verify their credentials, and manage their profiles. It involves interactions between professionals and the system for profile setup and updates.

- Layers: Presentation layer (UI for professional registration), Application layer (verification process and profile management), Data layer (database for storing professional information).

4. Rating and Review Subsystem:

- Type: Interactive
- Explanation: Allows users to provide feedback, ratings, and reviews for services and professionals. It facilitates two-way communication between users and the system for feedback submission and review display.
- Layers: Presentation layer (UI for feedback submission), Application layer (processing and displaying feedback), Data layer (database for storing feedback and reviews).

5. Payment and Invoicing Subsystem:

Type: Interactive

Explanation: Users may initiate payments, view invoices, or request billing information through interfaces provided by the system. This interactive nature, where users actively engage with the system to perform tasks and access information, aligns with the characteristics of an interactive subsystem.

Components: Payment gateway integration, invoicing system.

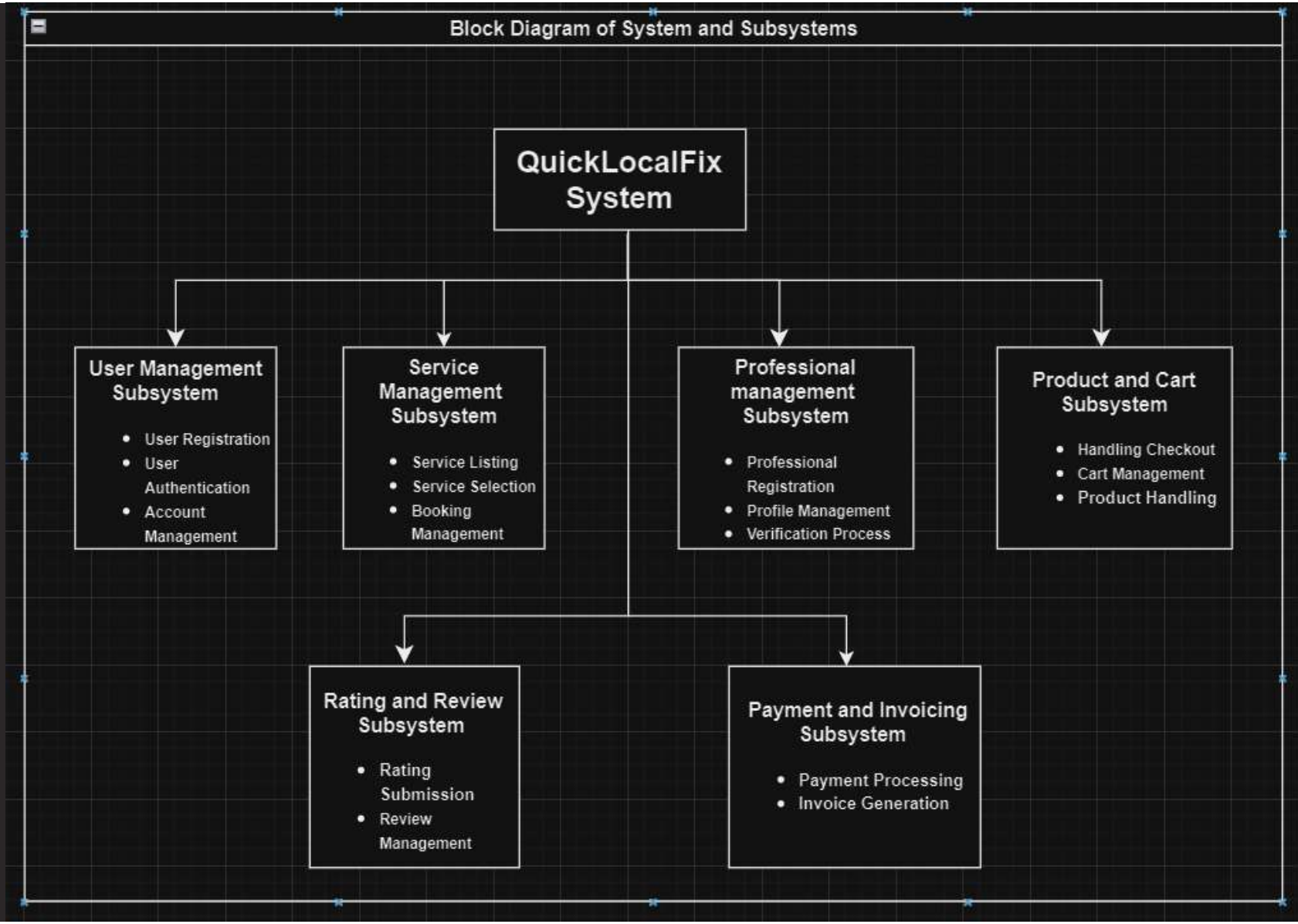
6. Product and Cart Subsystem:

Type:

Explanation:

Components:

Block Diagram of System and it's Subsystems



Identify the architecture that rules your system.

- N-Tier Architecture
- **Why?**
- Divided into multiple tiers or layers such as presentation layer, application layer and data layer
- Presentation Layer: user interaction and interface design with web pages, forms, and UI elements.
- Application Layer: business logic and application processing logic. It handles user requests, performs data processing, and coordinates interactions between various components.
- Data Layer: Manages data storage and retrieval. It includes the database management system (DBMS) and data storage mechanisms.

Snapshots of code showing the design principles:

Django Data base connectivity **abstracted** by using models from Django ORM:

```python
from django.db import models
# Defining a base User model which is abstract
class User(models.Model):
    user_name = models.CharField(max_length=50, unique=True, null=False)
    password = models.CharField()
    email = models.EmailField()
    phone_number = models.CharField(max_length=20)
    class Meta:
        abstract = True  # Making User an abstract class
# Customer model inheriting from User
class Customer(User):
    def __str__(self):
        return "User :" + self.user_name
# RepairPerson model inheriting from User
class RepairPerson(User):
    categories_of_repairs = models.ManyToManyField(Category, blank=True)  # A repair person can have many categories of
repair
    price_per_hour = models.DecimalField(max_digits=10, decimal_places=2)
    zip_location = models.CharField(max_length=20)
    def __str__(self):
        return "Repair Person: " + self.user_name
```

File paths:
QuickLocalFix\api\models.py
& QuickLocalFix\api\views.py

```python
    @csrf_exempt
    def register_user(request):
        if request.method == "POST":
            . . . . . .
            customer = Customer(user_name=user_name, password=make_password(password), email=email, phone_number=phone_number)
            customer.save()
            return JsonResponse({"success": "Registered user successfully.", "status": "200"})

        return HttpResponse(request)
```

separation of concerns by isolating components for different functionalities

Importing all components to main app:

```
import Layout from "./Layout";
import './App.css';
import React, {useState, useEffect} from 'react';
import "./styles.css";
import Login from "./components/Login";
import Home from "./components/Home";
import Location from "./components/Location";
import Services from "./components/Services";
import {Routes, Route} from 'react-router-dom';
import Register from "./components/Register";
import ProfessionalLogin from "./components/ProfessionalLogin";
import ProfessionalRegister from "./components/ProfessionalRegister";
import Account from "./components/Account";
import Products from "./components/Products";
import Unauthorized from "./components/Unauthorized";
import Professinals from "./components/Professinals";
import Dashboard from "./components/Dashboard";
import CartPage from "./components/Cart";
import AddressSelectionPage from "./components/AddressSelectionPage ";
import View from "./components/view";
import BookingCalander from "./components/BookingCalender";
import { ToastContainer } from 'react-toastify';
```

File path: src\App.js

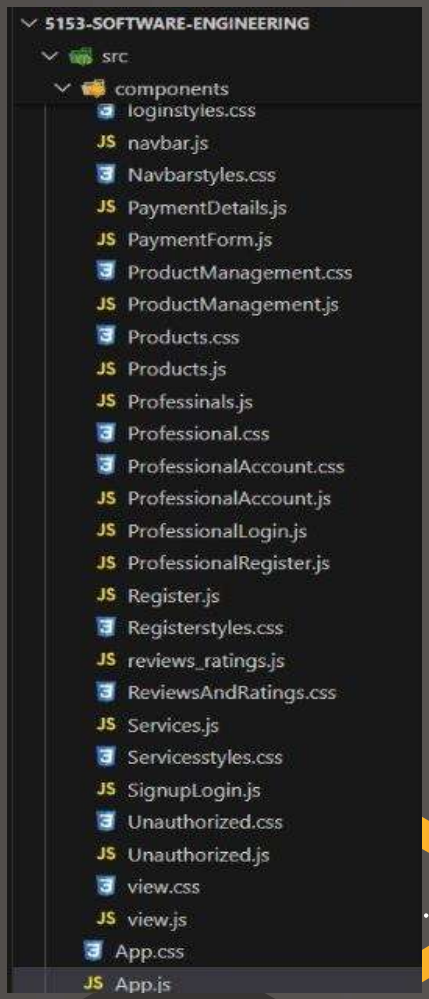


Fig. Component Tree

Loose **Coupling** of components:

Passing the required arguments only to reduce the requirement of one component on other like cart on payment component & products on cart component without making them as global variables.

File path: src\App.js

```
<Routes>
          <Route index element={<Home/>}></Route>
          <Route path="Login" element={<Login setIsLoggedIn={setIsLoggedIn} customer={customer}
setCustomer={setCustomer} cartItems = {cartItems} setCartItems = {setCartItems} />}/>
          <Route path="Services" element={isLoggedIn ?<Services location = {location}/> :
<Unauthorized/>}></Route>  <Route path="Location" element={isLoggedIn ?<Location/> : <Unauthorized/>}></Route>
          <Route path="Products" element={isLoggedIn ?<Products customer={customer} ProUser={ProUser}  cartItems =
{cartItems} setCartItems={setCartItems}/> : <Unauthorized/>}></Route>
          <Route path="ShoppingCart" element={isLoggedIn ?<CartPage customer={customer}
ProUser={ProUser}    setglobalCartItems = {setCartItems}/> : <Unauthorized/>}></Route>
          <Route path="Register" element={<Register/>}></Route>
          <Route path="Dashboard" element={<Dashboard  ProUser={ProUser} setProUser={setProUser}/>}></Route>
          <Route path="ProfessionalLogin" element={<ProfessionalLogin setIsLoggedIn={setIsLoggedIn}
setProUser={setProUser}/>}></Route>
          <Route path="ProfessionalRegister" element={<ProfessionalRegister />}></Route>
          <Route path="AddressSelection" element={<AddressSelectionPage
customer={customer}  setCustomer={setCustomer} setCartItems={setCartItems}/>} />
          <Route path="Account" element={isLoggedIn ? <Account customer={customer} setCustomer={setCustomer}
ProUser={ProUser}/> : <Unauthorized/>}></Route>
          <Route path="professional/:id" element={<View customer={customer}/>} />
          <Route path="booking/:id" element={<BookingCalander customer={customer} />} />
          <Route path="categories/:categoryId" element={<Professinals location={location} />} />
        </Route>
      </Routes>
```

Cohesion

Reusable components based on selection of tabs in dashboard.

File path: src\components\Dashboard.js

```
import React, { useState, useEffect } from "react";
import "./Dashboard.css";
import Chat from "./Chat";
import dashboardimg from "./assets/dashboard.gif";
import ProfessionalAccount from "./ProfessionalAccount";
import ProductManagement from "./ProductManagement";
import { toast } from 'react-toastify';
.
.
.
{selectedOption === "Chats" && <Chat repairRequests= {repairRequests} ProUser ={ProUser}/>}
{selectedOption === "Products" && (
    <ProductManagement professionalData={professionalData}/>)}
{selectedOption === "Account" && (
    <ProfessionalAccount ProUser={ProUser} setProUser={setProUser} />)}
.
.
```

8. Identify all your use cases. Provide a list of all your abstract use cases.

1. Register User Account
2. Login to User Account
3. Logout of User Account
4. Request Service
5. Accept Service Request
6. Track Repair Status
7. Search for Products
8. Add Products to Cart
9. Checkout Cart
10. Track Orders
11. Manage Repair Status
12. Submit Ratings and Reviews

Abstract and High-Level Use Case 1

Abstract Use Case : Register User Account

High Level Use Case:

Use Case: Register User Account
TUCBW : User clicks on "Register" link to create a new account on the platform.
TUCEW : User receives “Registered Successfully” notification and views Login page.

Abstract and High-Level Use Case 2

Abstract Use Case: Login to User Account

High Level Use Case:

Use Case: Login to User Account
TUCBW: User clicks on "Login" link to log in to their existing account.
TUCEW: User views homepage with their user account logged in.

Abstract and High-Level Use Case 3

Abstract Use Case: Logout of User Account

High Level Use Case:

Use Case: Logout of User Account
TUCBW: User clicks on log out of their account.
TUCEW: User is successfully logged out and session is terminated.

Abstract and High-Level Use Case 4

Abstract Use Case: Request Service

High Level Use Case:

Use Case: Request Service
TUCBW: Customer clicks on “Book Appointment” link in the Services Page.
TUCEW: Customer successfully submits a service request and receives confirmation notification.

Abstract and High-Level Use Case 5

Abstract Use Case: Accept Service Request

High-Level Use Case:

Use Case: Accept Service Request
TUCBW: Professional receives a service request from a customer.
TUCEW: Professional accepts the service request and confirms availability.

Abstract and High-Level Use Case 6

Abstract Use Case: Track Repair Status

High-Level Use Case:

Use Case: Track Repair Status
TUCBW: Customer clicks on "Repair Services" tab to know the status of their repair/service request.
TUCEW: User successfully views the repair status.

Abstract and High-Level Use Case 7

Abstract Use Case: Search for Products

High-Level Use Case:

Use Case: Search for Products
TUCBW: Customer clicks on "Products" link on Navbar.
TUCEW: Customer views the list of Products satisfying the search criteria.

Abstract and High-Level Use Case 8

Abstract Use Case: Add Products to Cart

High-Level Use Case:

Use case: Add Products to Cart
TUCBW: Customer clicks on "Add to Cart" button on products page.
TUCEW: Customer views the product added to the cart.

Abstract and High-Level Use Case 9

Abstract Use Case: Checkout Cart

High-Level Use Case:

Use Case: Checkout Cart
TUCBW: Customer clicks on Cart icon on Navbar.
TUCEW: Customer acknowledges the "Order Confirmed" notification..

Abstract and High-Level Use Case 10

Abstract Use Case: Track Orders

High-Level Use Case:

TUCBW: Customer clicks on "Orders" tab to track the order status.
TUCEW: Customer views the status of all the orders.

Abstract and High-Level Use Case 11

Abstract Use Case: Manage Repair Status

High-Level Use Case:

Use Case: Manage Repair Status
TUCBW: Professional clicks on "Repairs" tab from professional dashboard .
TUCEW: Professional views the "Updated Successfully" notification.

Abstract and High-Level Use Case 12

Abstract Use Case: Submit Ratings and Reviews

High-Level Use Case:

Use Case: Submit Ratings and Reviews
TUCBW: Customer clicks on Professional name to give ratings and reviews.
TUCEW: Customer clicks on "Add Review" button.

Use Cases based on Role Based Partition:

QuickLocalFix / Customer,
Actor: Customer

UC1. Register User Account
UC2. Login to User Account
UC3: Logout of User Account
UC4. Request Service
UC6. Track Repair Status
UC7. Search for Products
UC8. Add Products to Cart
UC9. Checkout Cart
UC10. Track Orders
UC12. Submit Ratings and Reviews

QuickLocalFix / Professional,
Actor: Professional

UC1. Register User Account
UC2. Login to User Account
UC3: Logout of User Account
UC5. Accept Service Requests
UC7. Search for Products
UC8. Add Products to Cart
UC10. Track Orders
UC11. Manage Repair Status.
UC12. Submit Ratings and Reviews

Expanded Use Case for UC1: Register User Account

Actor	System
	0) System displays the Homepage.
1) TUCBW User clicks on “Register” Link to create a new Account on the Platform.	2) The system displays the Register Page.
3) User enters necessary details and clicks on “Register” Button.	4) The system displays “Registered Successfully” and redirects to Login Page.
5)TUCEW User views the Login Page.	

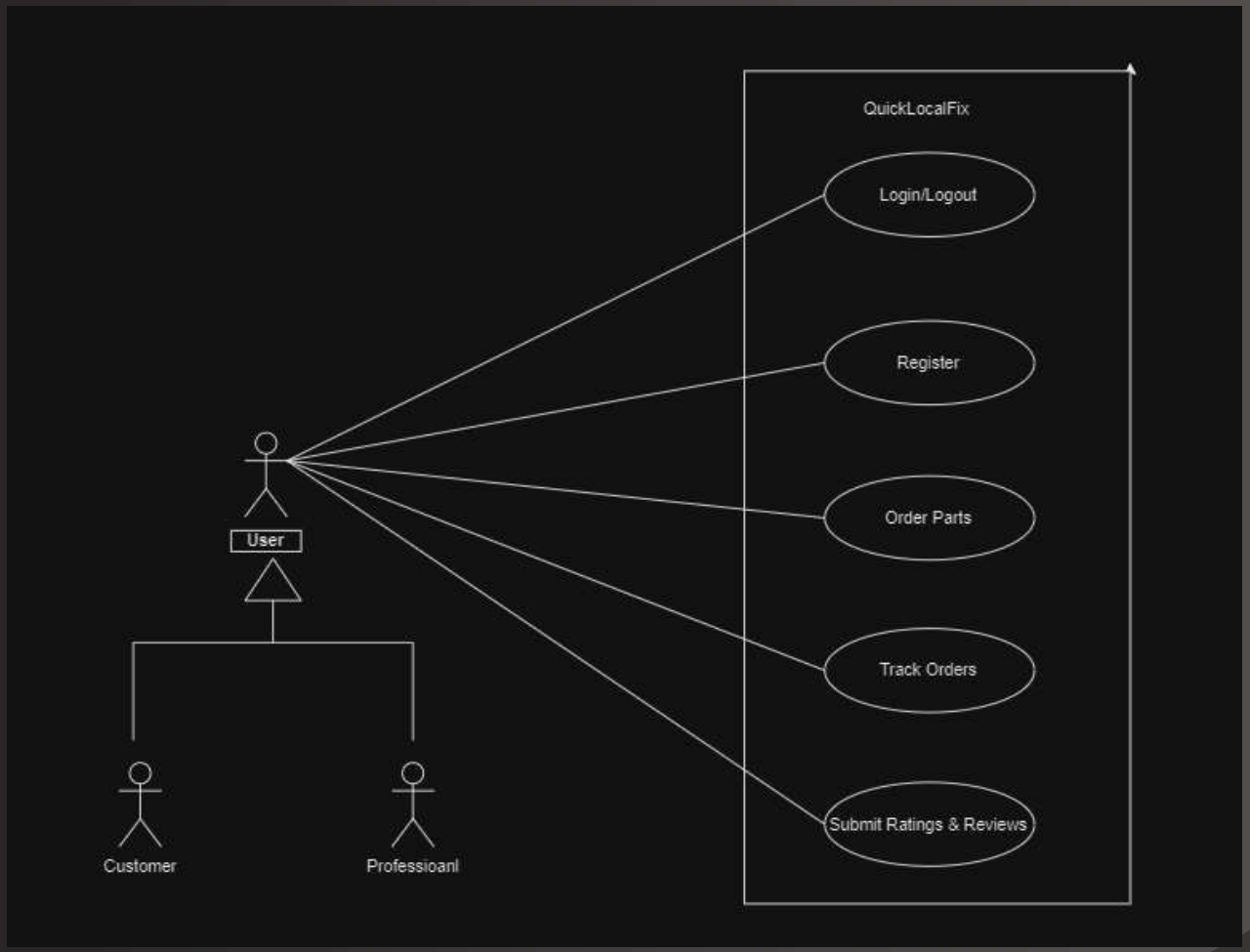
Expanded Use Case for UC7: Search for Products

Actor	System
	0) System displays the Home Page.
1) TUCBW User clicks on “Products” on Navbar.	2) The system displays the “Products” Page.
3) User enters required Product name and clicks on search button.	4) The system displays the Products list.
5)TUCEW User views the list of Products satisfying the search criteria.	

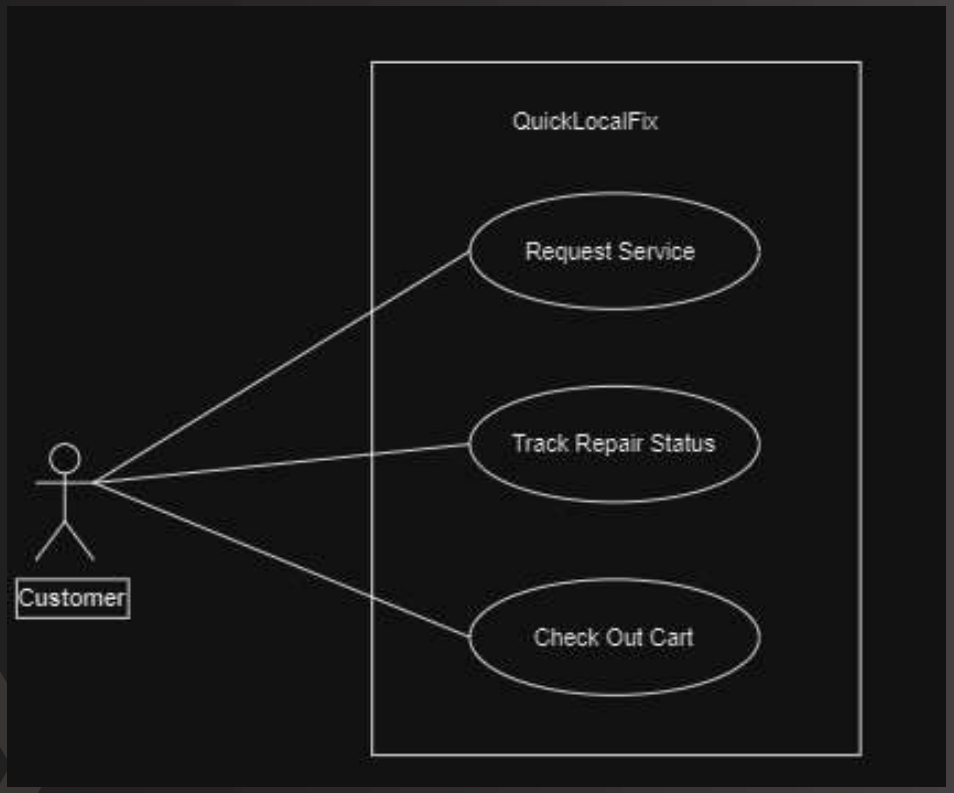
Expanded Use Case for UC9: Checkout Cart

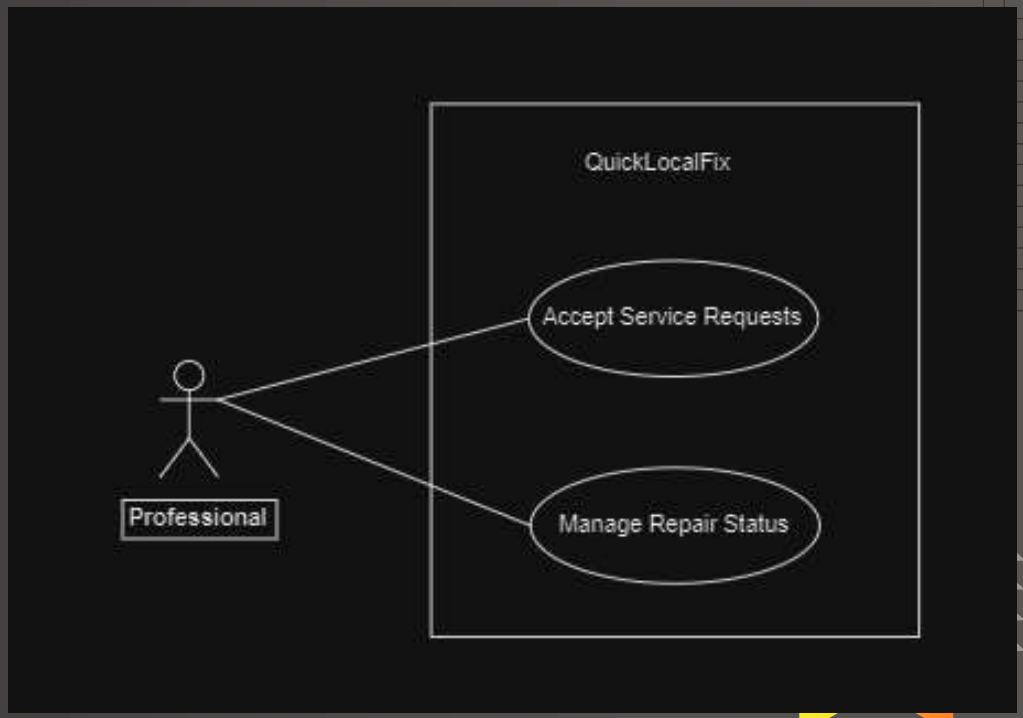
Actor	System
	0) System displays any of the QuickLocalFix Pages.
1) TUCBW Customer clicks on Cart icon on Navbar.	2) The system displays the Cart Page.
3) Customer verifies the Products and clicks on “Checkout” button.	4) The system displays the Cart Summary Page.
5) Customer selects the Address and Payment details and clicks on “Confirm & order” button.	6)System displays the “Order Confirmed” notification and redirects to Home Page.
7)TUCEW Customer acknowledges the notification.	

Use Case Diagram



Use Case Diagrams (Contd)





Application, Use case, Operation & Actions

Application	Use Case	Steps/Operations	Actions
QuickLocalFix	Register User Account	Open Registration Page	Click Register link from any of the pages of application
		Provide necessary details	Enters Username, Email, Password and Number
		Register	Click on Register Button

Application, Use case, Operation & Actions

Application	Use Case	Steps/Operations	Actions
QuickLocalFix	Search for Products	Open Products Page	Click Products Page link from Navbar
		Provide product name	Enters the product required for repair
		Search	Click on Search Button

Application, Use case, Operation & Actions

Application	Use Case	Steps/Operations	Actions
QuickLocalFIx	Check Out Cart	Open Cart Page	Click on Cart icon from Navbar
		Verify Products in the cart	Checks if all the products are added to cart and changes the quantities of each to required amount
		Provide Address and Payment details	Selects one of the available addresses and payment details
		Places Order	Clicks on Confirm & Order button and places the order

Scenario - Register User Account

1. User enters the registration details and clicks on Register button

2. Register GUI sends registration details to DBMgr to save the registration details

2.1. DBMgr checks for the username uniqueness

2.2. If the username is not unique

2.2.1. DBMgr writes error message "Username should be unique".

2.3 else,

2.3.1. DBMgr saves the registration details in the DB

2.3.2. DBMgr writes message "Registered Successfully"

2.3.3. Register GUI redirects to Login Page

2.2.4. Register GUI displays the message

3. User is shown the Login Page (or the error message)

Scenario - Search for Products

1.User enters the product name and clicks on search icon

2. Products GUI sends the product name to DBMgr

3. DBMgr sends the list of products matching the product name to Products GUI

4. Products GUI shows the list of products to User

Scenario - Check out Cart

1. Customer clicks on Cart icon from the navbar

2. Cart GUI sends the request to DBMgr to fetch the products added to the cart

3. DBMgr sends the List of products in the cart to Cart GUI

4. Cart GUI displays the products to Customer

Scenario Table - Register User Account

	Subject	Action of Subject	Other data/ Objects	Object Acted upon
2	Register GUI	sends	Registration details	DBMgr
2.1	DBMgr	Checks Uniqueness		Username
2.2	If Username is not unique			
2.2.1	DBMgr	creates	"Username should be Unique"	Error message
2.3	else			
2.3.1	DBMgr	Saves	Registration Details	DB
2.3.2	DBMgr	Creates	"Registered Successfully"	message
2.3.3	Register Component	Returns	Message/ Login Page	Register GUI
2.3.4	Register GUI	displays	message	
3	System	Shows	Login Page/ Error message	User

Scenario Table - Search for Products

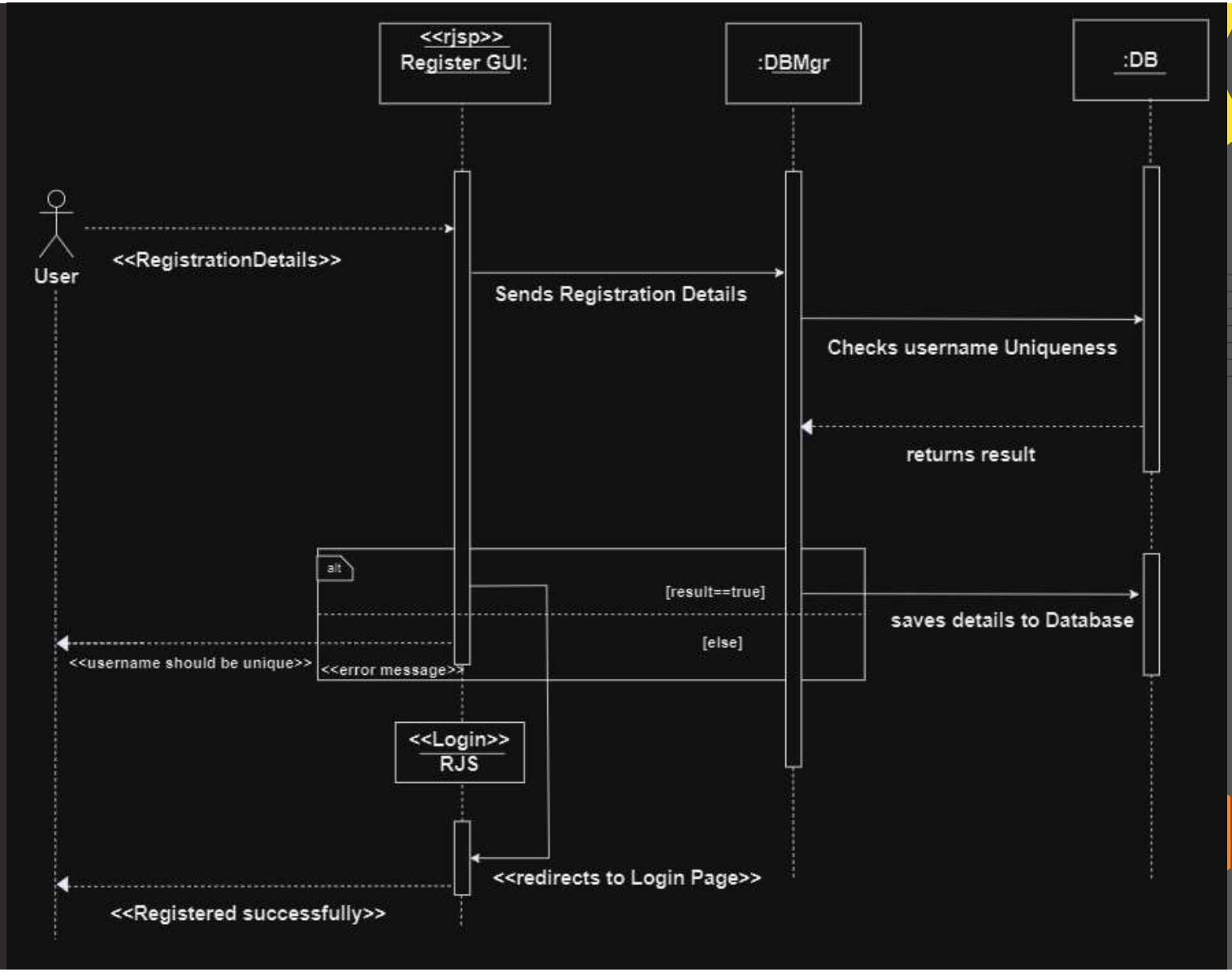
	Subject	Action of Subject	Other data/ Objects	Object Acted upon
2	Products GUI	Sends	Product Name	DBMgr
3	DBMgr	Sends	Products list	Products GUI
4	Products GUI	Displays	Products List	User

Scenario Table - Check out Cart

	Subject	Action of Subject	Other data/ Objects	Object Acted upon
2	Cart GUI	Sends request		DBMgr
3	DBMgr	Sends	Products	Cart GUI
4	Cart GUI	Displays	Products	Customer

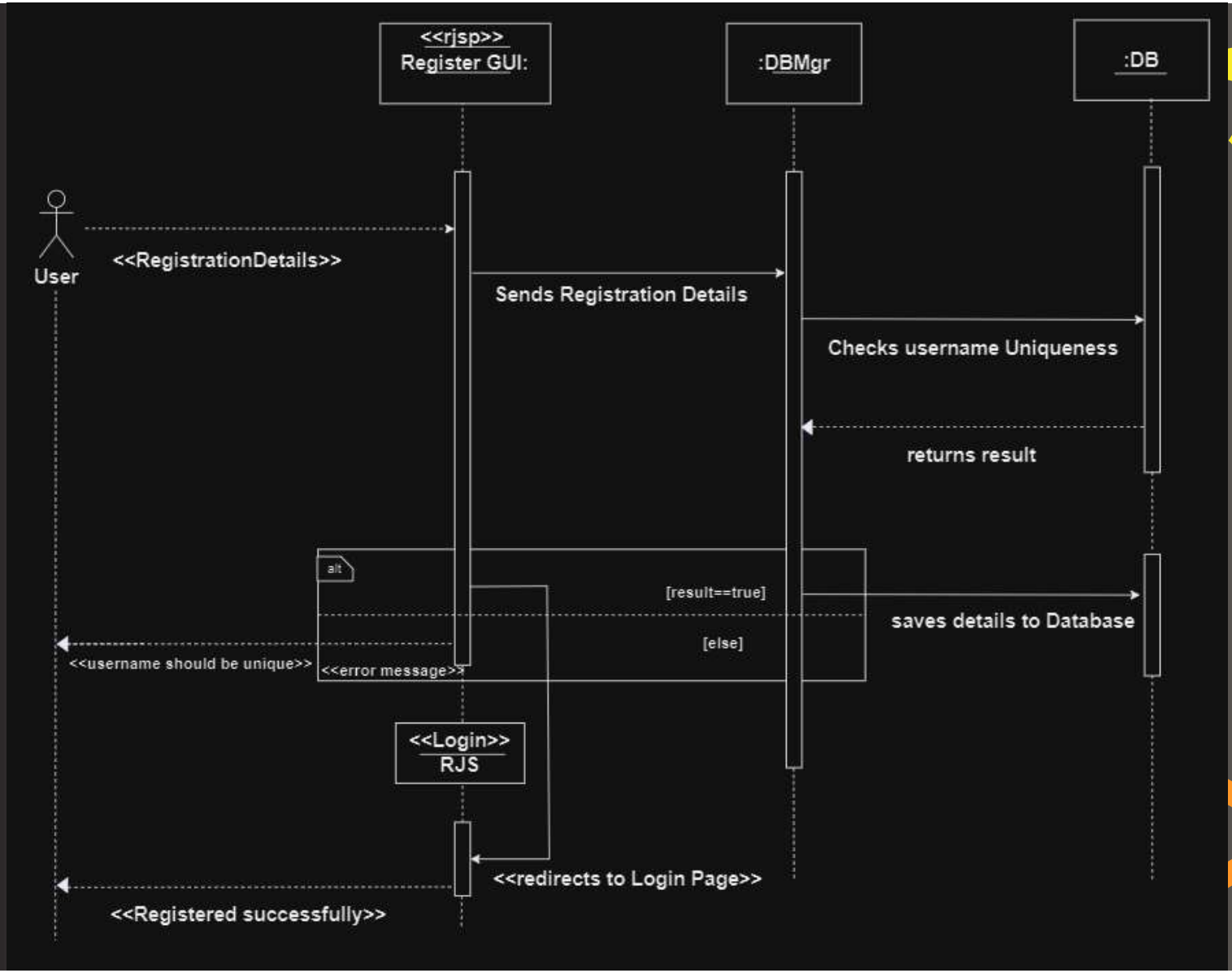
Informal sequence diagram :

Register User Account



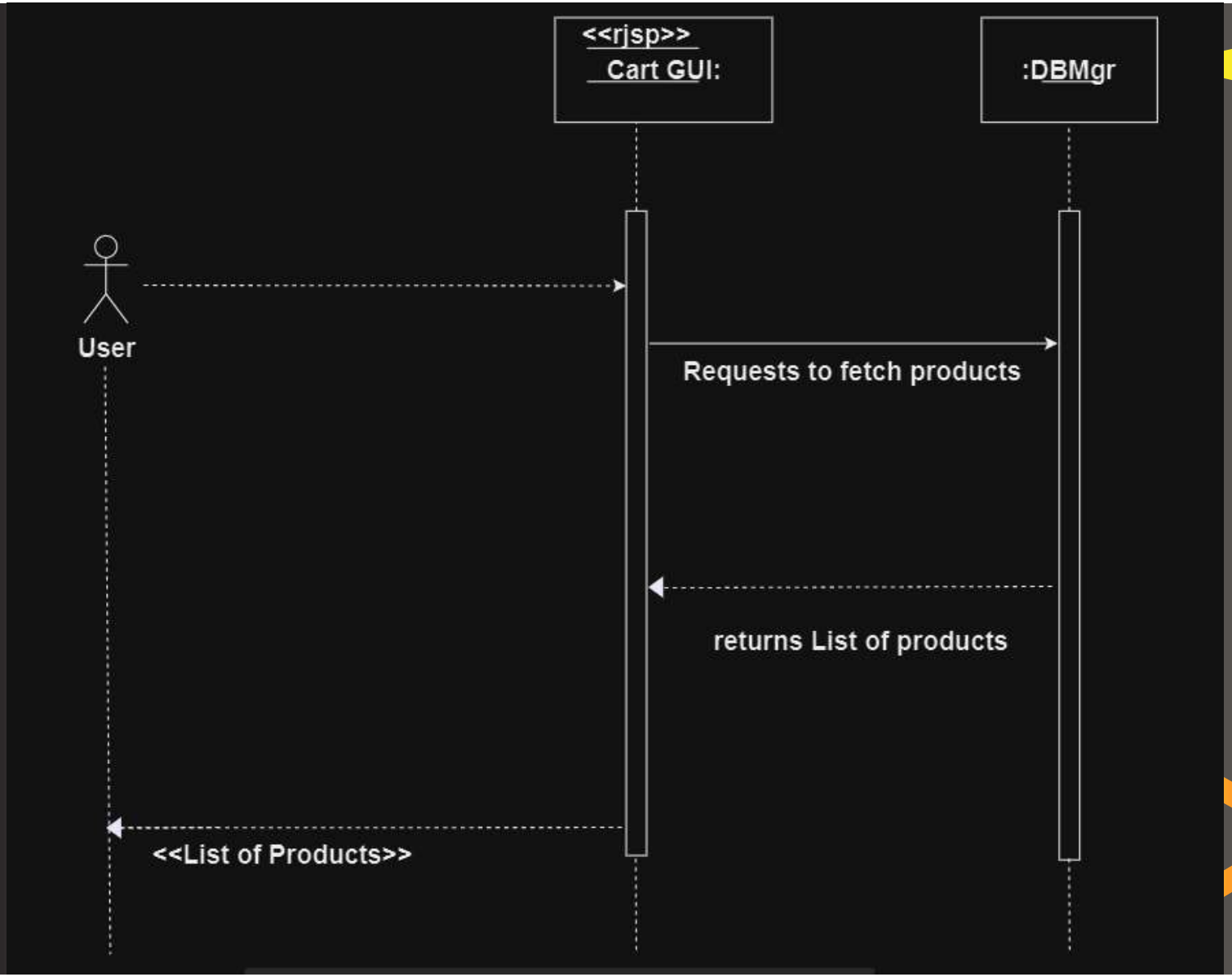
Informal sequence diagram :

Search for Products



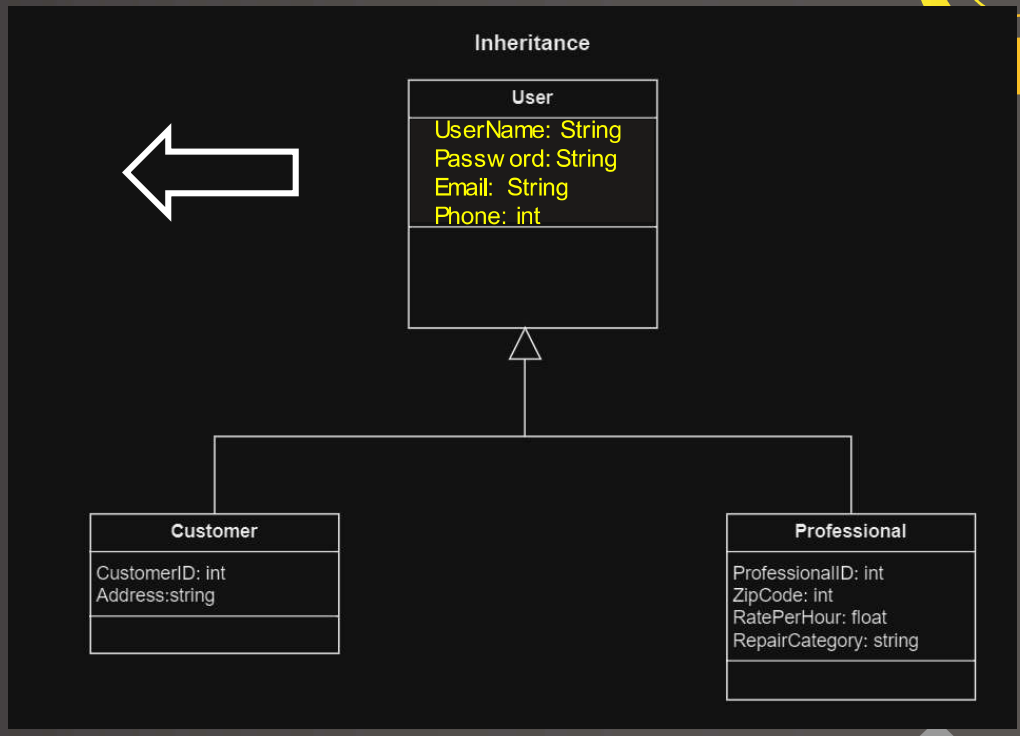
Informal sequence diagram :

Checkout Cart



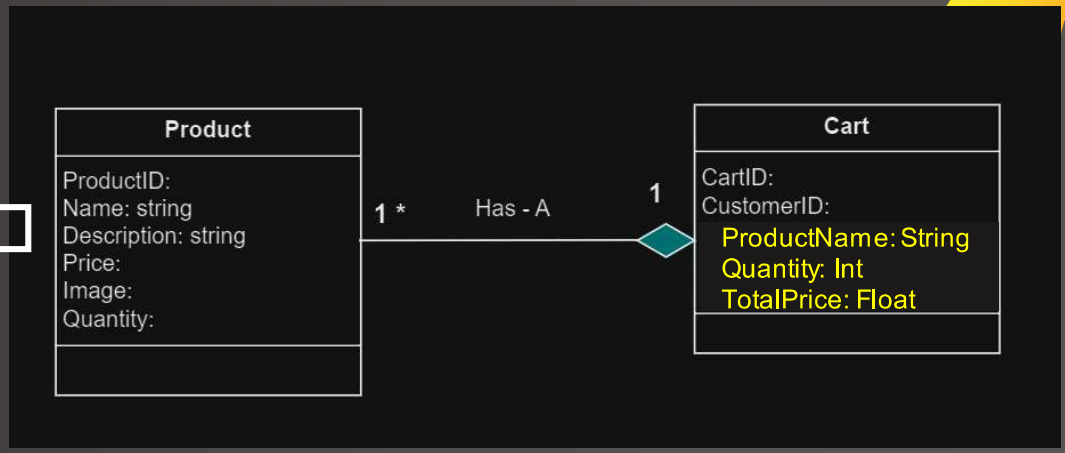
Domain Model for UC7: Register User Account

Actor	System
	0) System displays the Homepage.
1) TUCBW User clicks on “Register” Link to create a new Account on the Platform.	2) The system displays the Register Page.
3) User enters details such as **Username, Password, Email & Phone number** and clicks on “Register” Button.	4) The system displays “Registered Successfully” and redirects to Login Page.
5)TUCEW User views the Login Page.	



Domain Model for UC9:Checkout Cart

Actor	System
	0) System displays any of the QuickLocalFix Pages.
1) TUCBW Customer clicks on Cart icon on Navbar.	2) The system displays the Cart Page with **ProductName, Quantity,TotalPrice**
3) Customer verifies the Products and clicks on “Checkout” button.	4) The system displays the Cart Summary Page.
5) Customer selects the Address and Payment details and clicks on “Confirm & order” button.	6)System displays the “Order Confirmed” notification and redirects to Home Page.
7)TUCEW Customer acknowledges the notification.	



05

User Interface Prototypes


A picture is worth a thousand words

Showing UI Prototypes w/ Expanded Use Case 1: Register User Account

Actor	System
	0) System displays the Homepage.
1) TUCBW User clicks on “Register” Link to create a new Account on the Platform.	2) The system displays the Register Page.
3) User enters necessary details and clicks on “Register” Button.	4) The system displays “Registered Successfully” and redirects to Login Page.
5)TUCEW User views the Login Page.	

See Fig. A

See Fig. B & C

Fig. A

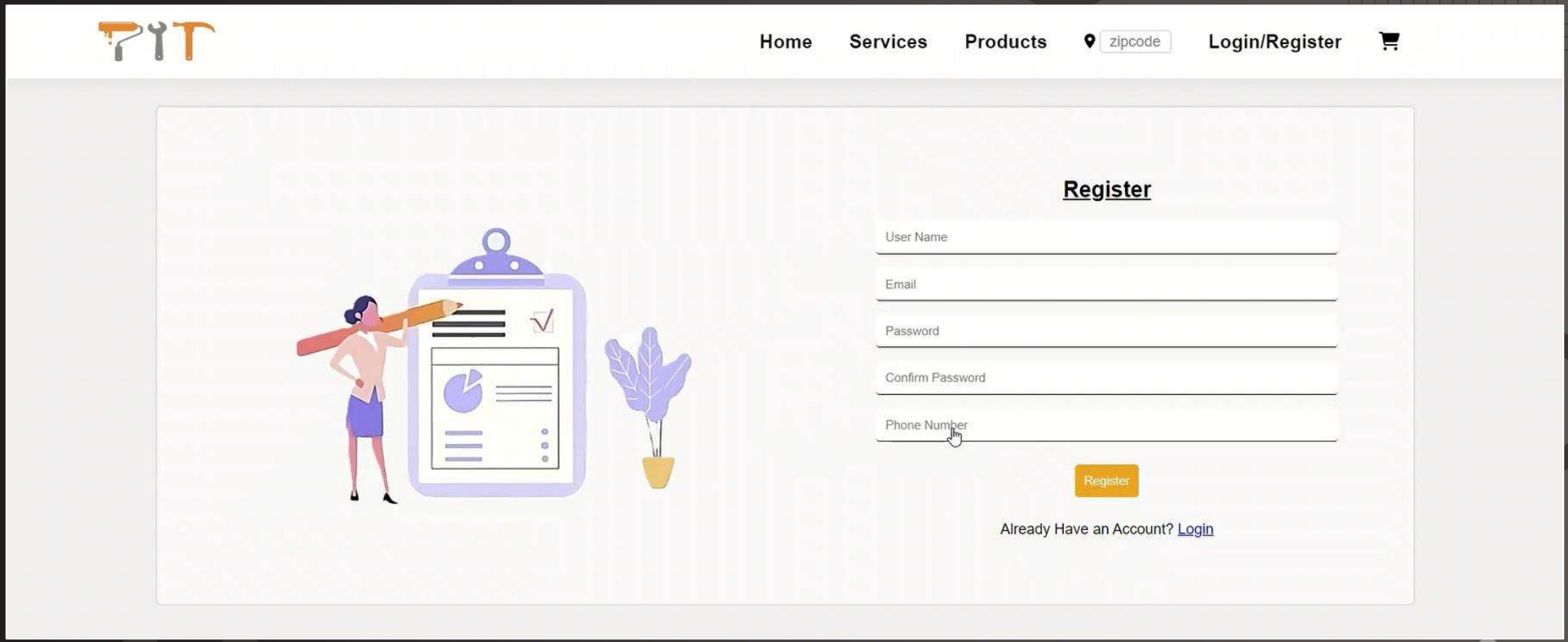


Fig. A: System Shows Register Page

Fig. B

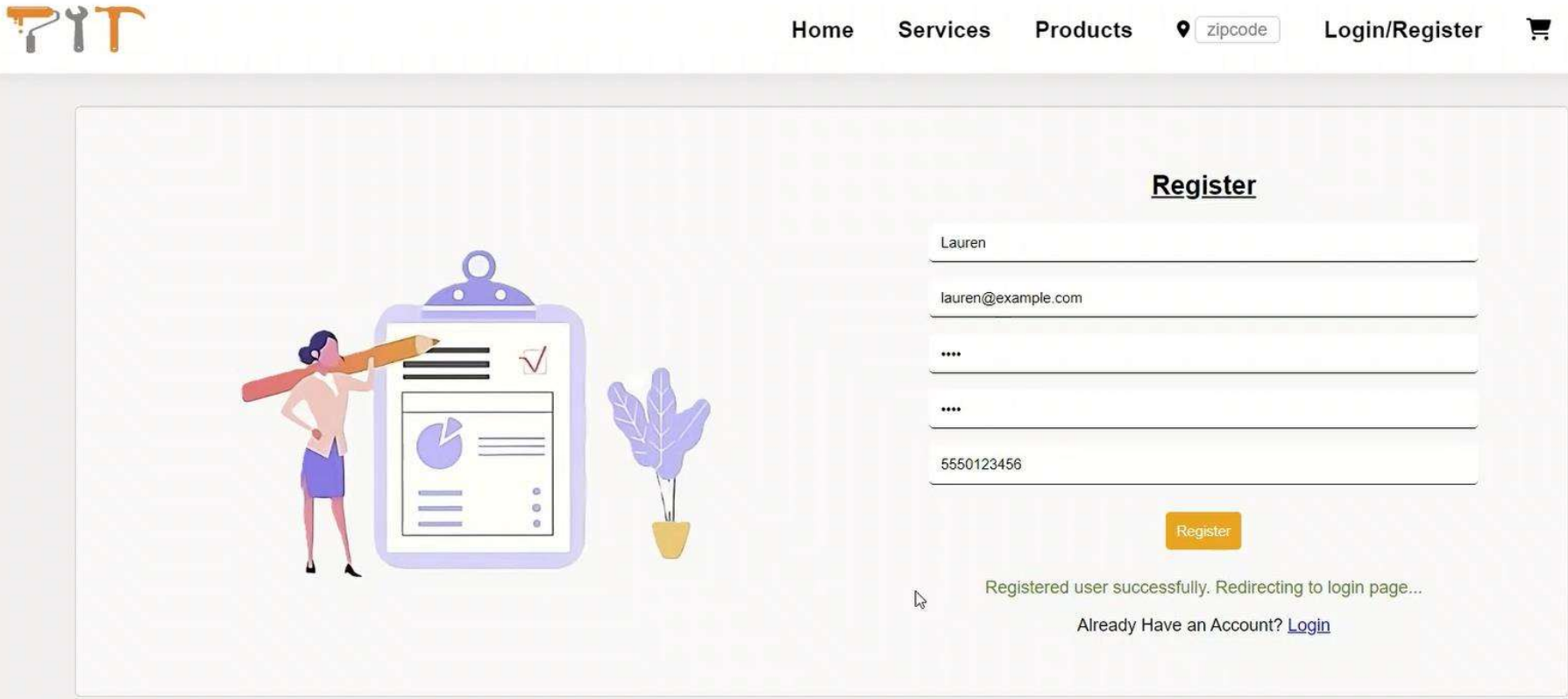


Fig. B: System Shows User Registered Successfully

Fig. C

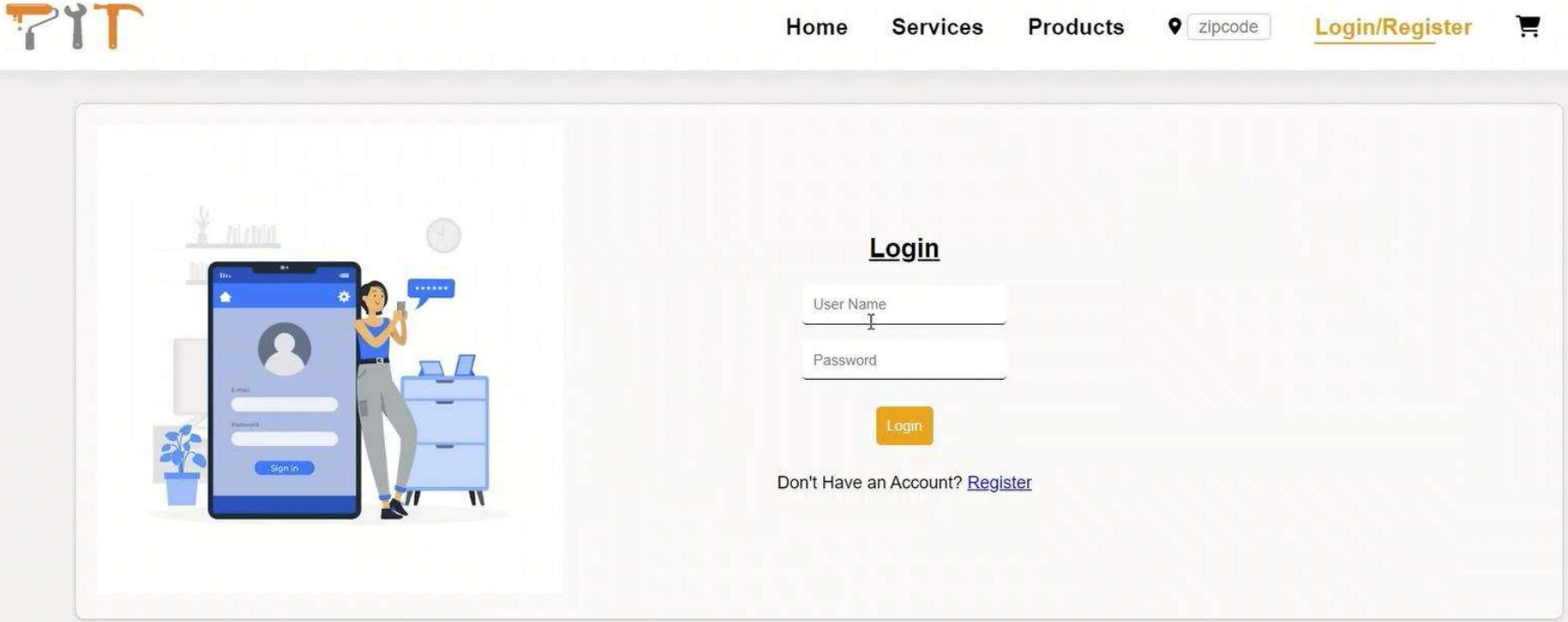


Fig. C: System redirects to Login Page

Showing UI Prototypes w/ Expanded Use Case 7: Search for Products

Actor	System
	0) System displays the Home Page.
1) TUCBW User clicks on “Products” on Navbar.	2) The system displays the “Products” Page.
3) User enters required Product name and clicks on search button.	4) The system displays the Products list.
5)TUCEW User views the list of Products satisfying the search criteria.	

See Fig. A

See Fig. B

Fig. A

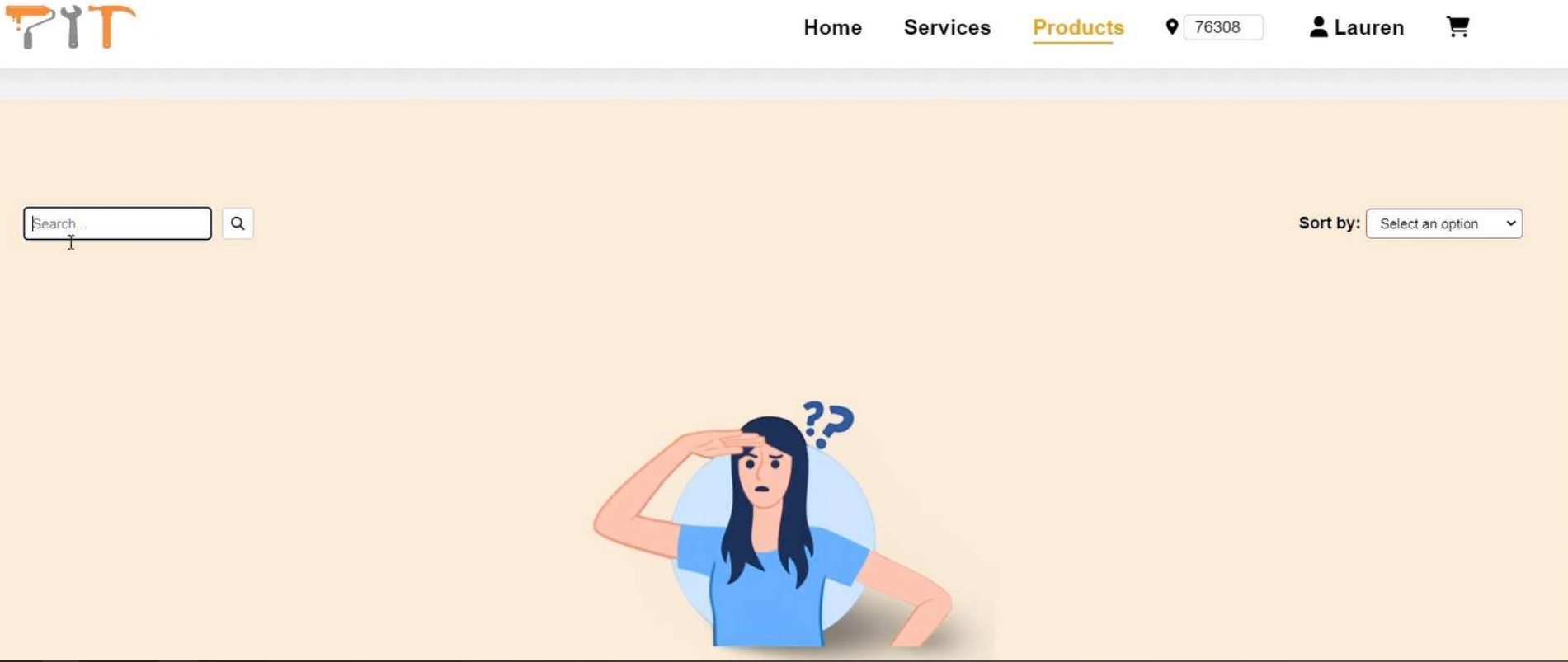


Fig. A: System Shows Products Page

Fig. B

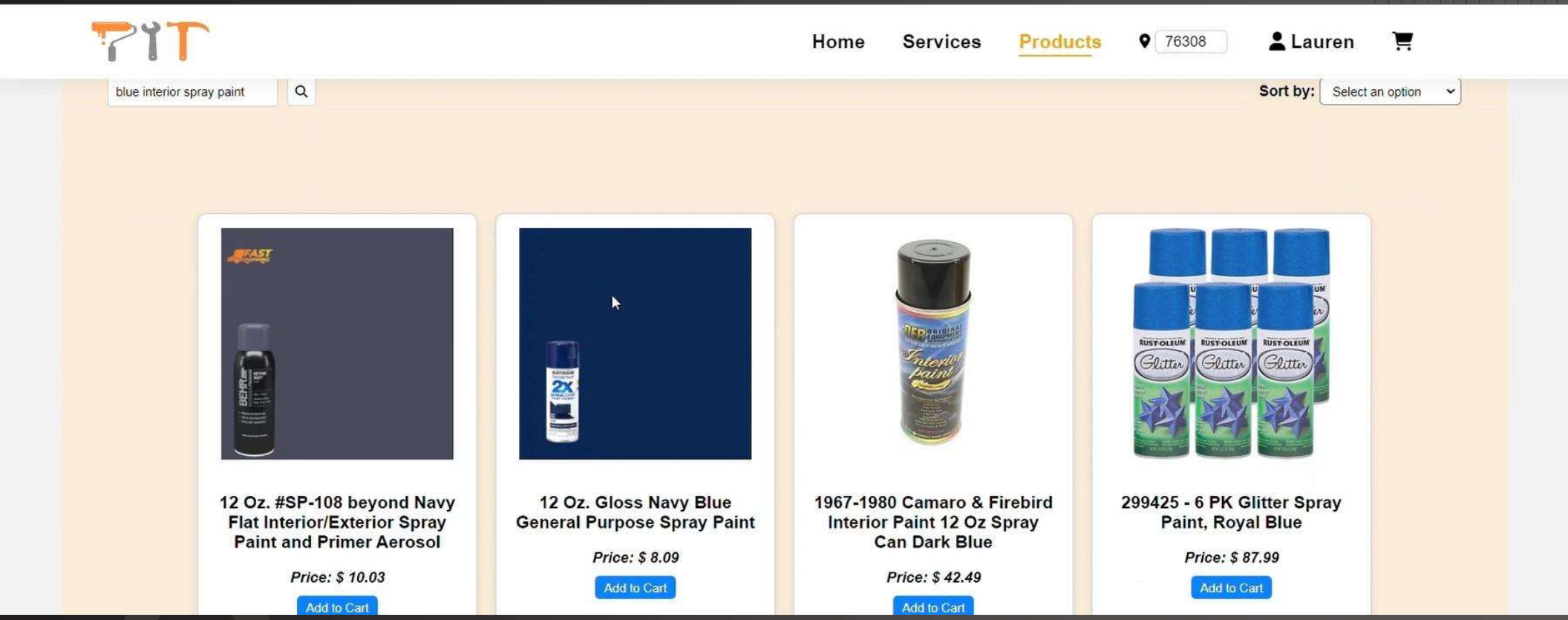


Fig. B: System Displays List of Products

Pre and Post Conditions for UC7: Search for Products

Pre-Condition: *This use case assumes that the user has logged into the system and is shown the Home page.*	
Actor	**System**
	0) System displays the Home Page.
1) TUCBW User clicks on “Products” on Navbar.	2) The system displays the “Products” Page.
3) User enters required Product name and clicks on search button.	4) The system displays the Products list.
5)TUCEW User views the list of Products satisfying the search criteria.	
Post-Condition: *List of Products are readily available to be added to the cart*	

Pre and Post Conditions for UC9: Checkout Cart

Pre-Condition: *This use case assumes that the user has logged into the system and has added required Products to Cart.*	
Actor	**System**
	0) System displays any of the QuickLocalFix Pages.
1) TUCBW Customer clicks on Cart icon on Navbar.	2) The system displays the Cart Page.
3) Customer verifies the Products and clicks on “Checkout” button.	4) The system displays the Cart Summary Page.
5) Customer selects the Address and Payment details and clicks on “Confirm & order” button.	6)System displays the “Order Confirmed” notification and redirects to Home Page.
Post-Condition: *Order is placed and delivered to the selected address.*	

Alternate Flow for UC7: Search for Products

Actor	System
	0) System displays the Home Page.
1) TUCBW the User clicks on “Products” on Navbar.	2) The system accordingly displays a.)The “Products” Page if user is logged in b.) Unauthorized Page if the user is not logged in.
3) The User a.)enters required Product name and clicks on search button. b.)clicks on Login link to Login.	4) The system displays a.) the Products list. b.) TUCCW Login to User Account.
5)TUCEW User views the list of Products satisfying the search criteria or Login Page.	

Alternate Flow for UC9: Checkout Cart

Actor	System
	0) System displays any of the QuickLocalFix Pages.
1) TUCBW Customer clicks on Cart icon on Navbar.	2) The system displays a.)The “Cart” Page if user is logged in b.) Unauthorized Page if the user is not logged in.
3) Customer a.)verifies the Products and clicks on “Checkout” button. b.)clicks on Login link to Login.	4) The system displays a.) the Cart Summary Page. b.) TUCCW Login to User Account.
5) Customer selects the Address and Payment details and clicks on “Confirm & order” button.	6)System displays the “Order Confirmed” notification and redirects to Home Page.
7)TUCEW Customer acknowledges the notification or views the Login Page.	

Incorrect Expanded Use Case: Register User Account

Actor	System
	0) System displays the Homepage.
1) TUCBW User clicks on “Register” Link to create a new Account on the Platform.	2) The system displays the Register Page.
3) User enters necessary details and clicks on “Register” Button.	**4) The Register GUI send the details to the DBMgr**
	5) DBMgr checks for username uniqueness and: **a) if the username is unique, it saves the details in the database** **b) else, it sends an error message.**
	5) The Register GUI displays “Registered Successfully” message and redirects to Login Page.
5)TUCEW User views the Login Page.	

Background Processing

Examples of incorrect Expanded Use Cases

Actor	System
	0) System displays the Homepage.
1) TUCBW User clicks on “Register” Link to create a new Account on the Platform.	2) The system displays the Register Page.
3) User enters necessary details and clicks on “Register” Button.	**4) The Register GUI send the details to the DBMgr**
	5) DBMgr checks for username uniqueness and: **a) if the username is unique, it saves the details in the database** **b) else, it sends an error message.**
	5) The Register GUI displays “Registered Successfully” message and redirects to Login Page.
5)TUCEW User views the Login Page.	

Background Processing

Team Member Work Accomplished

Divya Podila

Team Member Work Accomplished

Srinivas Makkena

Team Member Work Accomplished

Venkata Sai Prakash Boddu

Zip file of Code

Thank You!

Do you have any questions?