UNIVERSITY OF GREENWICH

SYSTEM DESIGN AND DEVELOPMENT (COMP1430)

COURSEWORK REPORT

MELEK IZMIRLI BICAKCI

001260581

MSC COMPUTING AND INFORMATION SYSTEM

CONTENT TABLE

LIST OF FIGURES

INTRODUCTION

Mariata Homes, a charity organization committed to addressing housing challenges, offers subsidized housing solutions to the broader public. Its primary focus is on providing short-term accommodations to individuals in need. Prospective residents at Mariata Homes are required to complete paper-based application forms to request assistance.

Recognizing the high volume of applications and the need for streamlined processes, Mariata Homes has embarked on an initiative to transition its assistance platform to an online system. This new system entails the registration of both clients (users) and administrative users. The application process involves the completion of digital forms, facilitating a more efficient and accessible means for individuals seeking assistance. Moreover, the online platform empowers administrative users with CRUD (Create, Read, Update, Delete) functionalities to manage registered users effectively.

This study is comprehensive, encompassing the identification of entities, the demonstration of process normalization up to the Third Normal Form (3NF), and the creation of an Entity-Relationship Diagram (ERD) based on the identified entities. Additionally, the study involves generating a Class Diagram and Use Case Diagrams to facilitate a nuanced understanding of the case study.

The investigation is twofold, dividing its focus between frontend and backend operations. The front-end operations concentrate on designing user interfaces. Conversely, the backend operations are centred around essential functionalities. This encompasses user registration, login mechanisms, the submission of applications for assistance, and the comprehensive management of users through CRUD operations.

PART1. SYSTEM DESIGN

A. NORMALISATION TO 3RD NORMAL FORM

Normalization is the process of simplifying and ensuring the reliability of complex data structures. (McFadden and Hoffer, 1994)

Designing an effective database involves the concept of normalization, aiming for a collection of interconnected tables that minimize redundant data and eliminate update, delete, or insert anomalies. (Kennedy, 2000)

Database design using normalization is a methodical "bottom-up" approach. The designer engages in user interviews and gathers documents, such as reports. (Rob and Coronel, 1997) The information within these reports is then organized and normalized to generate the necessary tables and attributes, adhering to the principles of First Normal Form (1NF), which requires atomic values and the absence of repeating groups, Second Normal Form (2NF), ensuring there are no partial dependencies, and Third Normal Form (3NF), eliminating transitive dependencies (McFadden & Hoffer, 1994).

According to the scenario given, three main and three helper entities have been identified as follows:

Entities	
1	User
2	Application Form
3	Recommending Source

The step-by-step normalization processes for these three entities are outlined below:

Due to the nature of the scenario, two distinct user roles, admin and user, are involved. To ensure a secure design, an additional field for roles has been assigned to the user table for defining roles either at the database or system level.

User				
name	**username**	**password**	**image**	**role**
Melek Izmirli	mizmirli	123456		user
Salih Bicakci	sbicakci	123		admin

In adherence to the first normal form, information must be atomic, and repetition of data should be avoided. Specifically, for the "name" field, it is necessary to split it into distinct columns for "name" and "surname."

User					
name	**surname**	**username**	**password**	**image**	**role**
Melek	Izmirli	mizmirli	123456		user
Salih	Bicakci	sbicakci	123		user

After the organization of atomic data into tables in the 1NF stage, the next step involves distinguishing and organizing partial dependencies into separate tables. These tables are then linked to each other using foreign keys. This stage is called the Second Normal Form(2NF).
In this context, it's evident that the role field shows partial dependency, needing its storage in a separate table along with its distinct attributes, as outlined below:

Roles	
roleId	**roleName**
1	User
2	Admin
3	System Admin

The final step to achieve 3NF is to establish relationships between tables by identifying foreign keys and creating fields in the tables. In this case, roleId serves as a foreign key (FK) in the user table for accessing relevant data from the roles table.

User						
id	**name**	**surname**	**username**	**password**	**photo**	**roleId**
1	Melek	Izmirli	mizmirli	123456		1
2	Salih	Bicakci	sbicakci	123		1

Users can be either customers seeking assistance from Mariata Homes or administrators who can access all users' information and make alterations to them. Application form, on the other hand, is used to submit a request. Lastly, the recommended sources play a role in identifying where the customers come from.

In order to provide smooth and secure registration, two different roles have been created for this system. During the registration, all users are identified as regular users and as “1”. After registration, a system admin assigns an admin role for any user, if necessary.

Roles	
roleId	**roleName**
1	User
2	Admin
3	System Admin

The user table serves dual purposes, accommodating both registration and CRUD (Create, Read, Update, Delete) operations. It has been intentionally designed with simplicity in mind, considering potential user challenges. This design allows users to sign up with minimal information, omitting details such as email and phone numbers.

The applicationForm entity created for seeking assistance has undergone 1NF and 2NF normalizations. To enhance usability and achieve optimal normalization, the partially dependent fields regarding address information have been separated into distinct tables and linked through foreign keys.

applicationForm								
formid	**userid**	**phone**	**illness**	**nextofkin**	**address**	**postcode**	**city**	**country**

applicationForm (continues)						
sourceId	**sAddress**	**sPostcode**	**cityId**	**countryId**	**DOB**	**gender**

To avoid repetitions of the names of cities and countries, there are also city and country tables holding city and country information seperately.

City	
cityId	**cityName**
1	London
2	Birmingham
3	Cambridge
4	Brighton
5	Bristol

Country	
countryId	**countryName**
1	UK

As a main entity of the assistance system, the source table is designated as a separate table having two fields sourceid and sourcetype.

source	
sourceid	**sourceType**
1	Hospital
2	Police Centre
3	Immigrant Service
4	Homeless
5	Other

B. PHYSICAL MODEL OF ENTITY RELATIONSHIP DIAGRAM(ERD)

The entity-relationship diagram is commonly used in structured analysis and conceptual modelling. (Song, Evans, and Park, 1995) This approach is easy to understand, effective for modelling real-world problems, and can be easily translated into a database schema. (Batini, Ceri, and Navathe, 1992) The ERD represents the real world as a collection of business entities, their relationships, and the attributes that describe them. In this representation model, rectangles are used to show entities that the system owns, and their attributes are the fields belonging to a database table. Relationships vary and represent an association among entities. (Lind, Song, and Park, 1995)

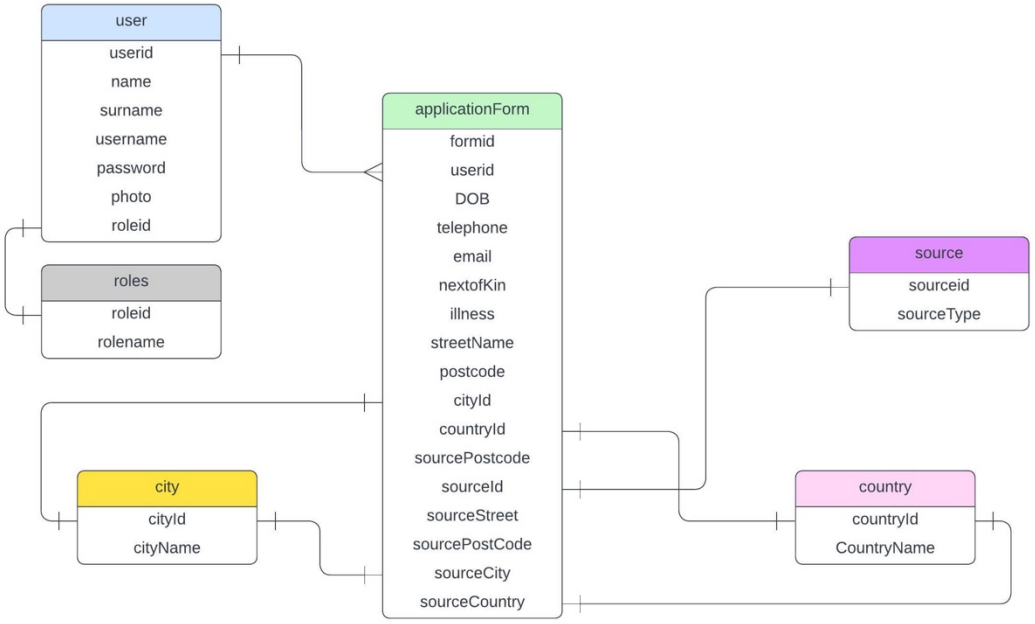

Figure 1.Entity Relationship Diagram (ERD)

C. UNIFIED MODELLING LANGUAGE(UML) CLASS DIAGRAM

A class diagram provides an overview of the system's object types and outlines the static relationships between them. It visually represents the properties and operations associated with each class, along with the constraints governing how objects interconnect. Within the Unified Modelling Language (UML), the term "feature" serves as a comprehensive term encompassing both the properties and operations of a class. (Fowler, 2003)

Based on the provided class diagram, we observe that each entity is depicted in a rectangular shape, divided into three distinct sections. The initial segment is allocated for the class name. The second segment encompasses properties, which refer to variables associated with the class. The final segment pertains to actions, specifically the methods attributed to the class. This organizational structure within the class diagram facilitates a clear representation of the essential components and relationships within each entity.

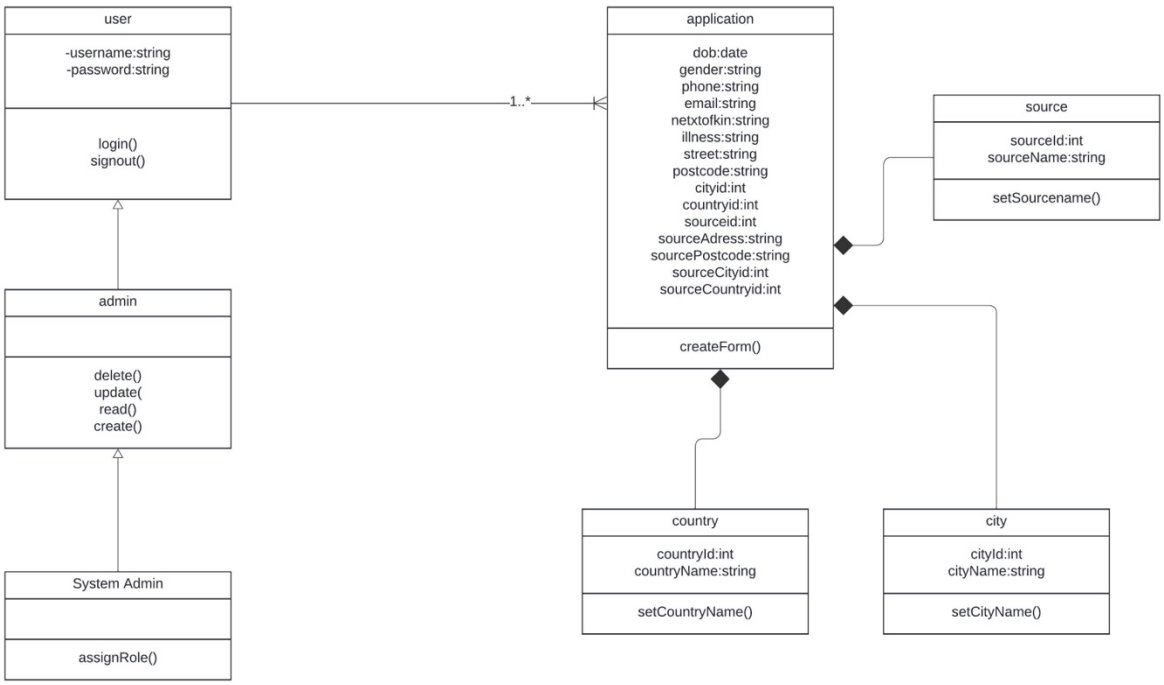

Figure 2.Class Diagram

D. USE CASE DIAGRAM

It belongs to the category of Unified Modelling Language (UML) diagrams, commonly applied in the field of software engineering. (Mule, Waykar, and Mahavidyalaya, 2015) Use case diagrams function as a valuable instrument for performing high-level requirement analysis within a system. (Kušek et al., 2001) In the course of scrutinizing system requirements, the functionalities are encapsulated into use cases, serving as systematically organized depictions of the system's operations. Conversely, actors constitute entities engaged in interactions with the system and may encompass human users, internal applications, or external applications.

When crafting a use case diagram, it is essential to identify the following components:

1. Functionalities, typically portrayed as oval shapes, are represented as **use cases.**
2. **Actors** are the external entities engaging with the system.
3. **Relationships** are illustrated by lines connecting use cases and actors. (Kulak and Guiney, 2012)

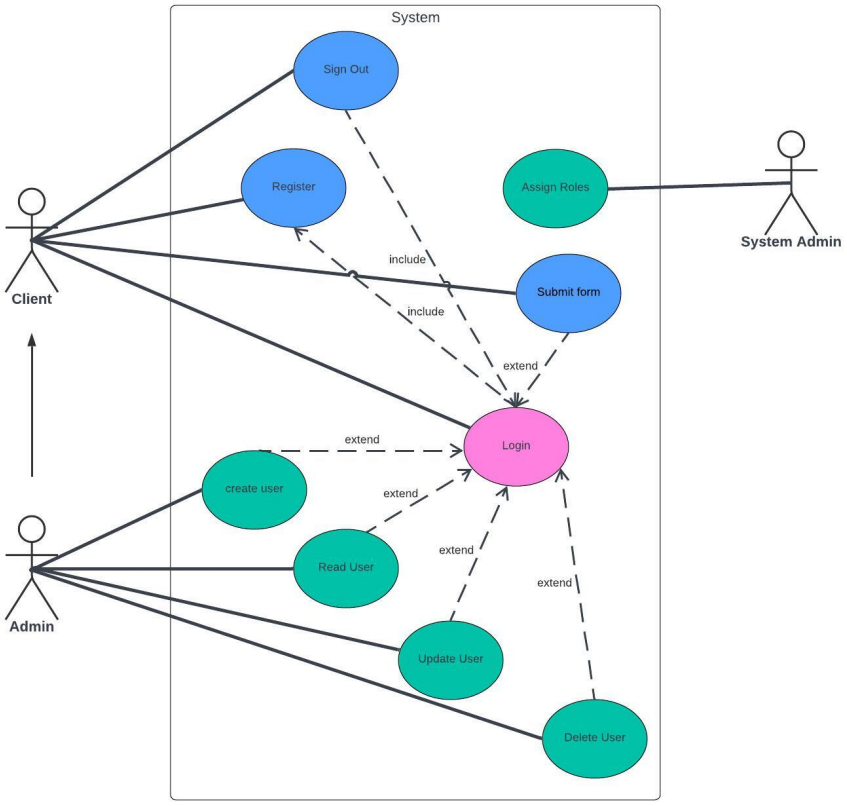

Figure 3.Use Case Diagram

The system is represented within a rectangular shape. Actors to the left of the system are referred to as primary actors, while those on the right are secondary actors. In this use case diagram, there are two primary actors, namely the user and admin, and one secondary actor, the system admin. According to this use case diagram, registering in the system is a prerequisite for making an application. Admin users, as they perform tasks similar to regular users, are symbolized as an inheritance. Admin users, if they are logged in, can perform CRUD operations on registered users in the database. The secondary actor, the system admin, is responsible for changing user roles within the system."

E. USER INTERFACES/FRONT-END FORMS

Prototyping serves as a developmental strategy aimed at enhancing the planning and implementation of software projects. This method involves creating executable software systems, known as prototypes, for experimental exploration. It proves highly effective for acquiring practical insights into novel application domains and helping in incremental or evolutionary software development processes.

Utilizing prototyping is a valuable approach for generating concepts on the design of a user interface and assessing the effectiveness of a solution in its initial phases. Consequently, the increasing application of user interface prototyping in numerous projects is attributed to its ability to foster idea generation and early-stage solution evaluation. (Baumer et al., 1996)

Rapid prototyping is the creation of a preliminary version of a system or its components to clarify requirements and address design considerations. This process allows engineers and users to test the software, ensuring alignment with user needs. Engineers may use prototyping to understand technical demands and system feasibility. Unlike the traditional "waterfall" development model, prototyping addresses weaknesses by clarifying essential requirements before full implementation.

Rapid prototyping often used interchangeably with "prototyping," includes two methodologies: throw-away (discarding the prototype) and evolutionary (retaining part or all in the final product). The iterative prototyping process concludes when sufficient experience is gained (throw-away) or when the final system is complete (evolutionary). The focus is solely on rapid prototyping, distinct from executable specifications. (Gordon and Bieman, 1991)

In the initial stage of the project, some wireframe designs were created with the help of the Lucidcart free version, based on the needs gathered during the requirement analysis process. These designs can be amended as necessary throughout the project life cycle. Following the Software Development Life Cycle (SDLC) philosophy, eliminations or updates will be added where deemed necessary.

In anticipation of the possibility that not every user may have information such as email and phone, these fields are positioned in the application form, instead of registration.

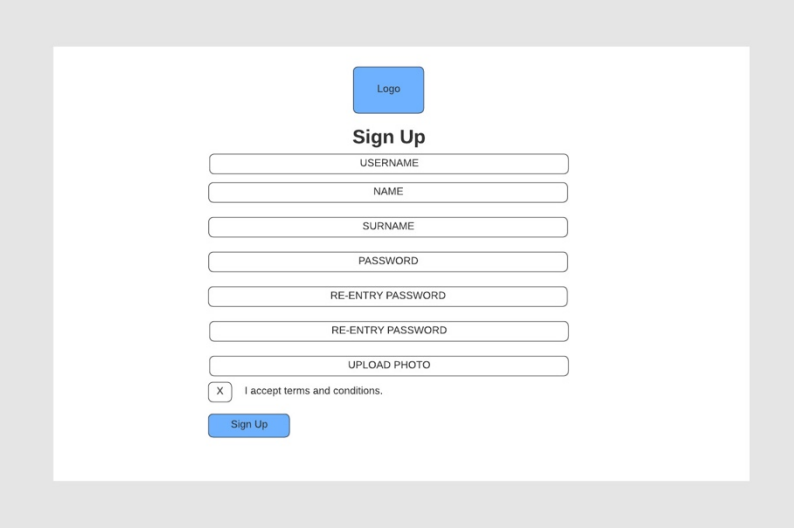

Figure 4.Sign Up Form Wireframe

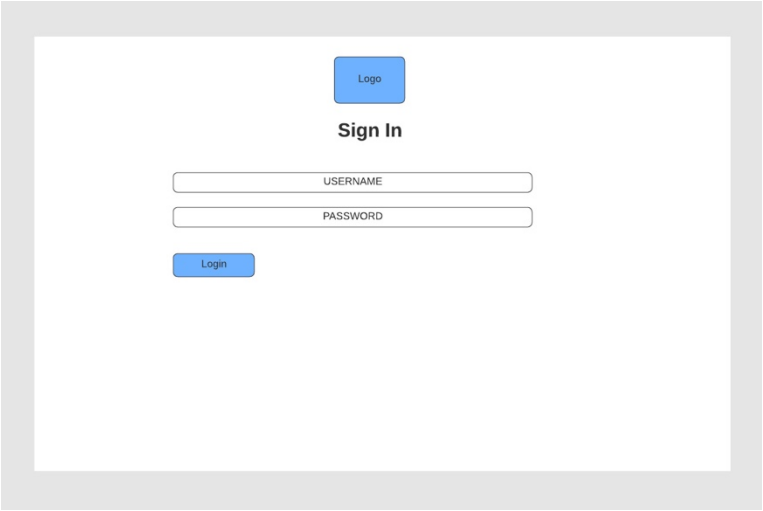

Figure 5.Sign In Form Wireframe

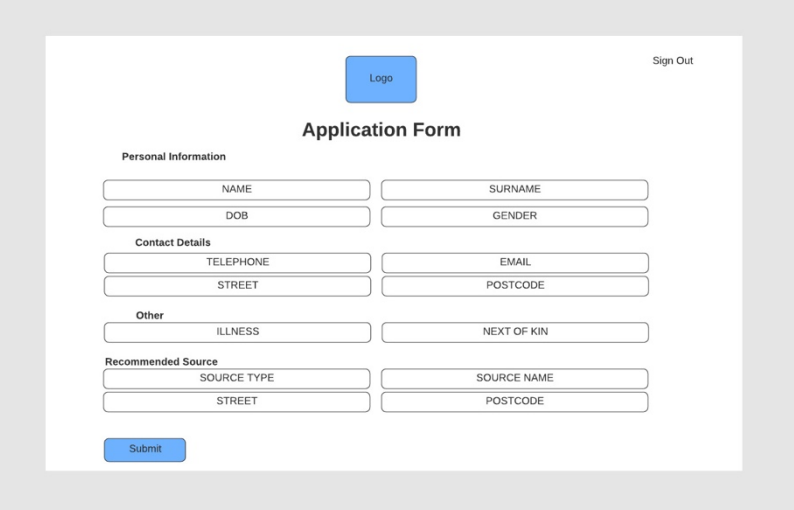

Figure 6.Application Form Wireframe

PART2. SYSTEM DEVELOPMENT

In this study, Visual Studio Code and MySQL Workbench were used as the development interface. Due to compatibility issues with MacBook Air, a dump file of the database has been added to the project zip file. The relevant query content can be accessed from the dump file.

A. REGISTRATION OF USER AND ADMIN

Mariata Homes, serving as a charitable organization, provides accommodation services for those in need. When considering the disadvantaged position of the users, it is essential to meet the user with a simple user interface, which is devoid of details but equally informative. Light colours and simple fonts have been selected with the aim of improving the experiences of different types of users. Both registration and login processes can be easily accessed through the homepage.

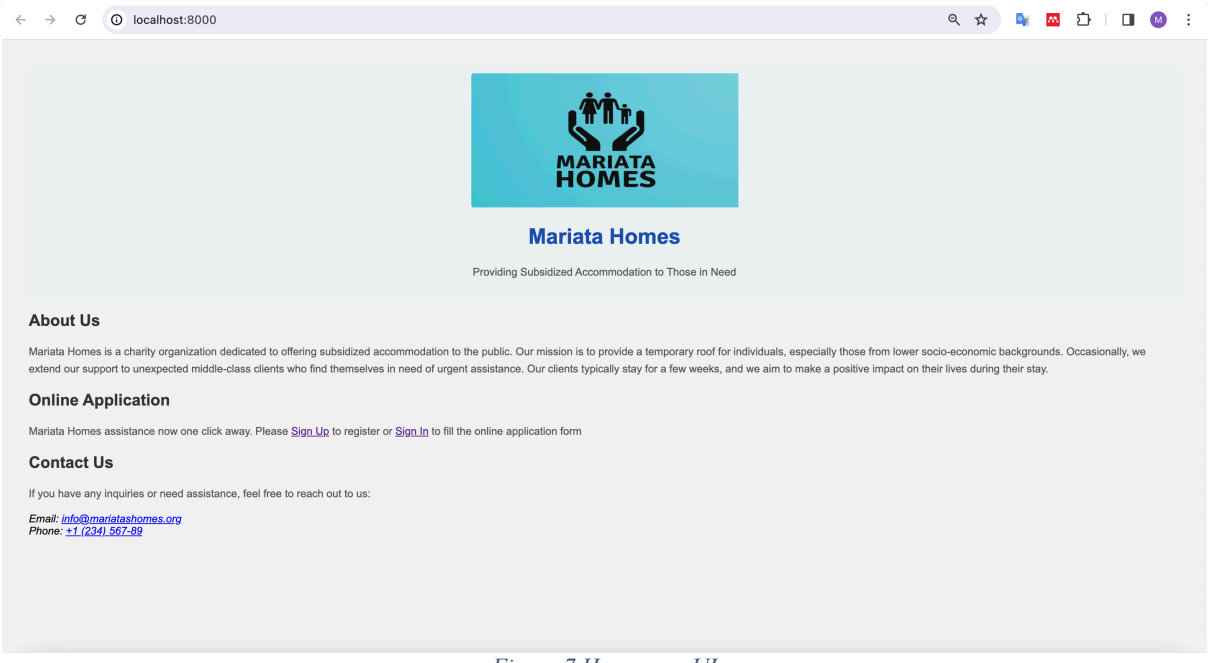

Figure 7.Homepage UI

The user registration process requires the input of name, surname, username, password, and photo information.

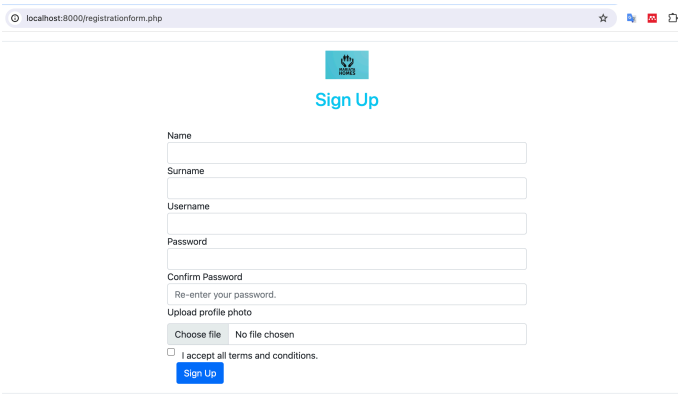

Figure 8.Sign Up UI

B. INPUT USER DETAILS AND INFORMATION

To ensure both a user-friendly experience and prevent applications on behalf of others, the registration process automatically and passively populates the name and surname fields in the application form with the information saved in the database for the registered user.

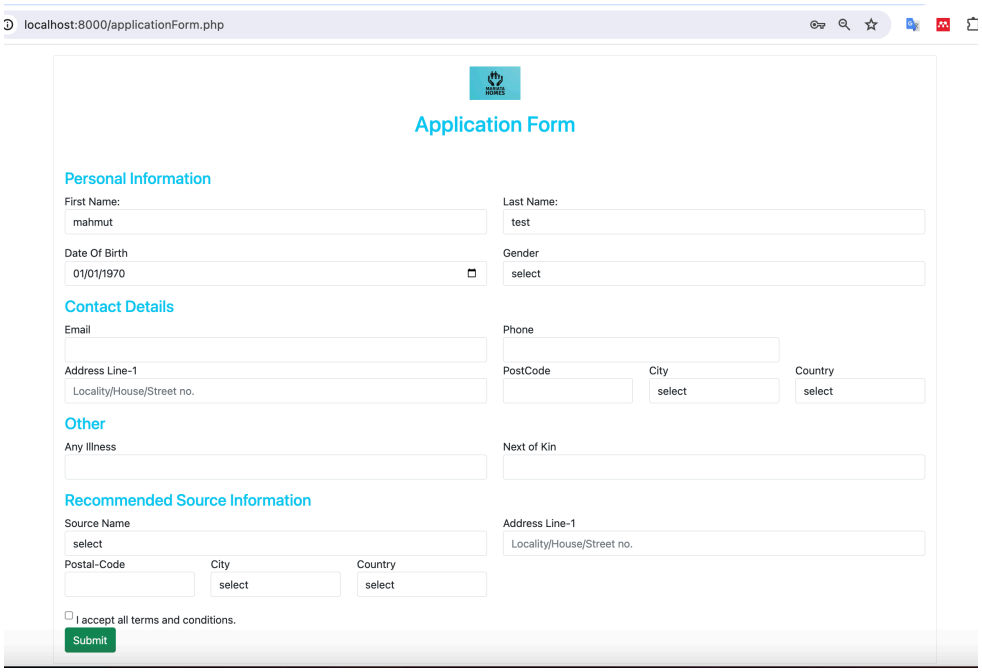

Figure 9.Application Form UI

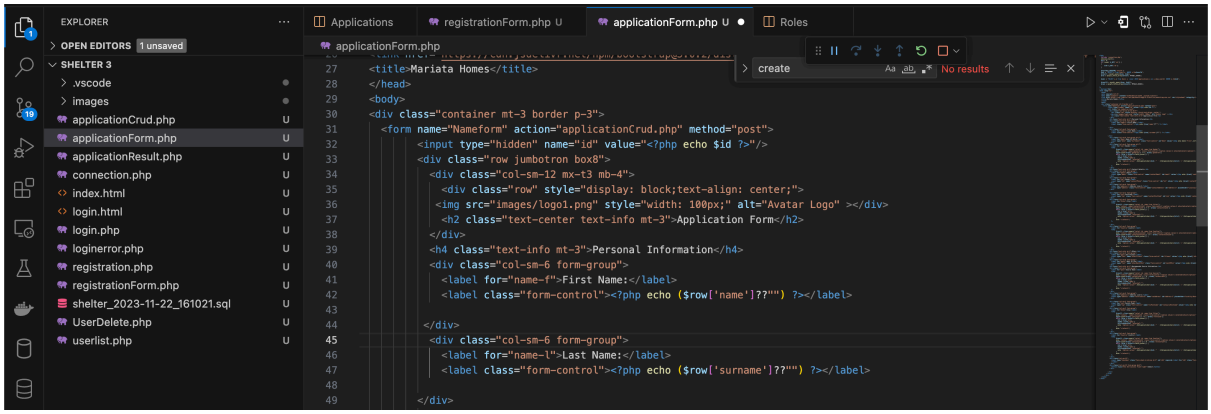

Figure 10.Name-Surname retrieving with PHP

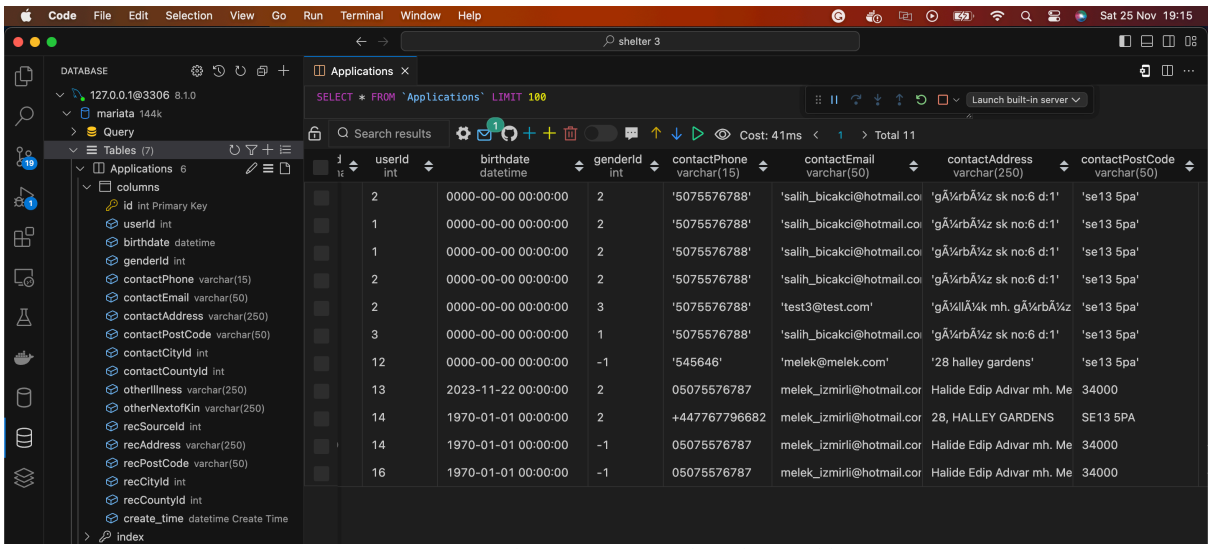

Figure 11.Application Form Back-End DB Table

To prevent unnecessary redundancies in the database and ensure normalization, data such as city, country, gender, and recommended source are stored in separate tables. The information from these tables is reflected in the relevant fields of the application form as dropdown lists through queries.

```php
<div class="col-sm-2 form-group">
  <label for="State">City</label>
  <?php
        $result = $con->query("select id, name from Cities");
        echo "<select   name='recCityId' class='form-control'><option value=-1 selected>select</option>";
        $gid=!isset($row1['recCityId']) ? 0 : $row1['recCityId'];
        while ($row = $result->fetch_assoc()) {
          $id = $row['id'];
          $name = $row['name'];
          $slct=$gid==$id? "selected":"";
          echo '<option value="'.htmlspecialchars($id).'"  '.htmlspecialchars($slct).'>'.htmlspecialchars($name).'</option>';
        }
        echo "</select>";
  ?>
</div>
<div class="col-sm-2 form-group">
  <label for="Country">Country</label>
  <?php
        $result = $con->query("select id, name from Counties");
        echo "<select  name='recCountyId' class='form-control'><option value=-1 selected>select</option>";
        $gid=!isset($row1['recCountyId']) ? 0 : $row1['recCountyId'];
        while ($row = $result->fetch_assoc()) {
          $id = $row['id'];
          $name = $row['name'];
          $slct=$gid==$id? "selected":"";
          echo '<option value="'.htmlspecialchars($id).'"  '.htmlspecialchars($slct).'>'.htmlspecialchars($name).'</option>';
        }
        echo "</select>";
  ?>
```

Figure 12.Filling Drop-down Lists by Query

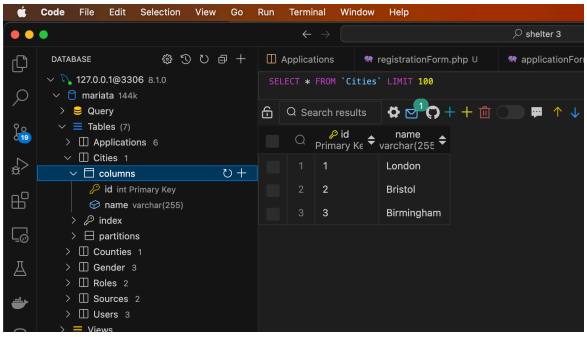

Figure 13.City Table for Drop-down Selection

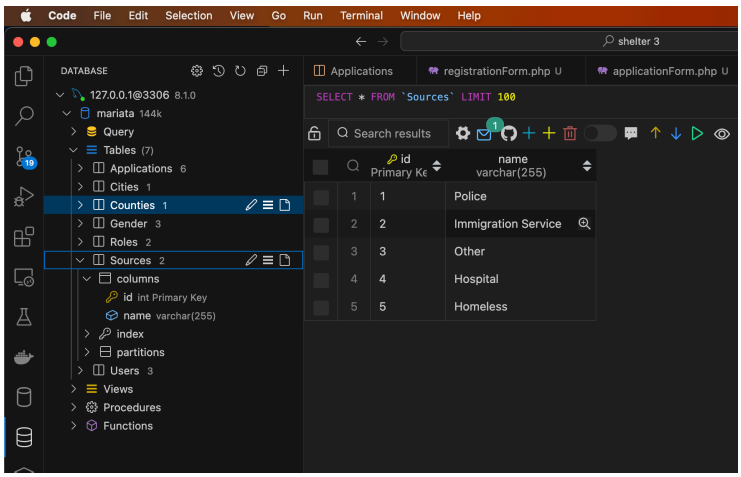

Figure 14.Country Table for Drop-down selection

C. USER AND ADMIN LOGIN

Users who have registered on the system can log in by using their credentials, which consist of a combination of a username and password.

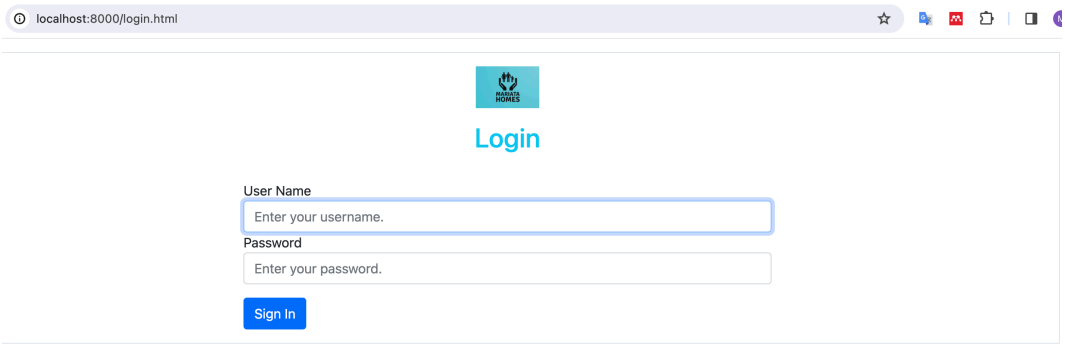

Figure 15.Login UI

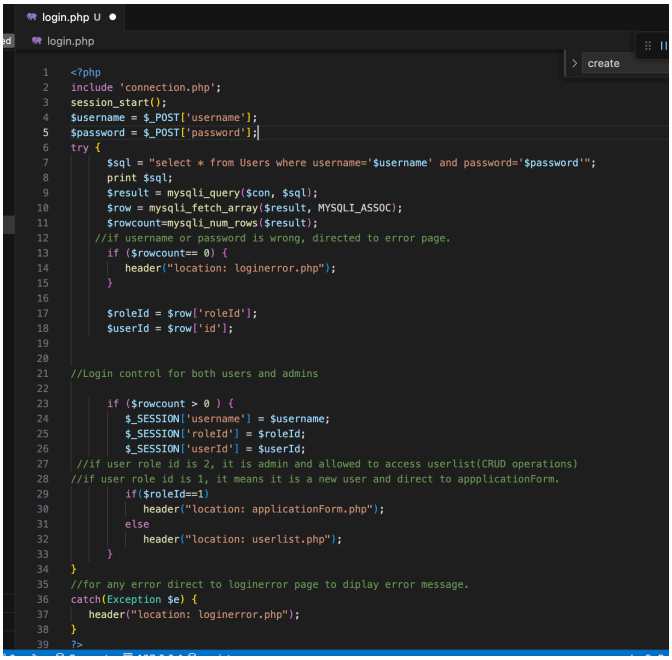

Figure 16.Role Based Decision at Signing In

There are three types of users in the system: System Admin, Admin and User. Clients registering for the first time are initially assigned the default user role. In cases where a user needs to have an admin role, the system admin changes the user's role to admin in the database. Below, the flow for both types of users on the site is illustrated:

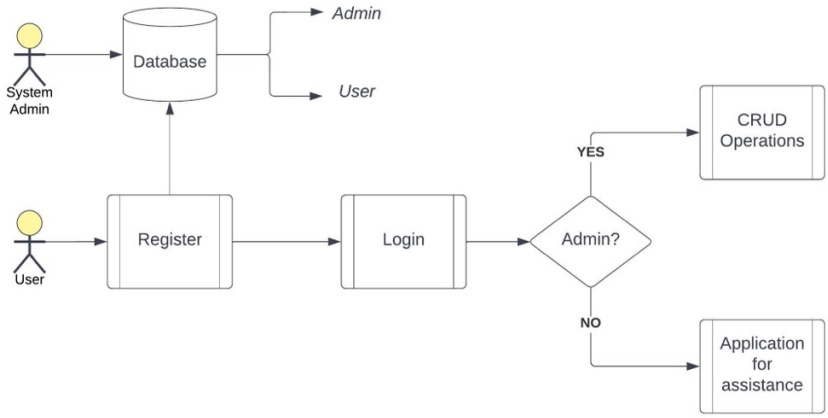

Figure 17.Role-Based Flowchart

A registered user with the 'user' role (on user table roles=1), upon logging into the system, can only access the application form for assistance. On the other hand, a user with the 'admin' role(user table roles=2), after logging in, can access the list form and perform CRUD operations.

D. CRUD OPERATIONS

The CRUD pattern encompasses a single use case known as CRUD Information or Manage Information, representing various operations that can be executed on a specific type of information. These operations include creating, reading, updating, and deleting the information. (Prashant, 2012)

In this study, users with admin privileges can access the list form after the login screen. Here, registered users in the system are displayed. The existing information of users can be modified through the form that opens by clicking the pen icon, unnecessary users can be removed by clicking the bin icon, or new users can be added through the registration form opened by clicking the 'Add User' button. It is also possible to intervene in the roles of existing users through role changes on this list form.

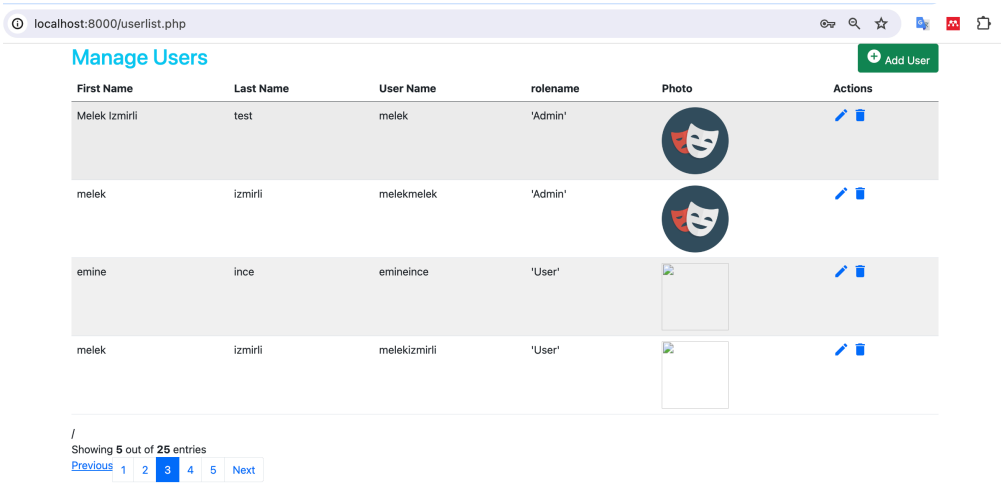

Figure 18.List Form for CRUD

Below are the PHP codes that retrieve all users from the ‘User’ table in the database for the list view. Initially, a session is started, and access to this page has been programmatically restricted if users do not have the ‘Admin’ role. However, if session owner is an admin, the query is executed, and the data obtained from the query results are transferred to the relevant fields in the list form.

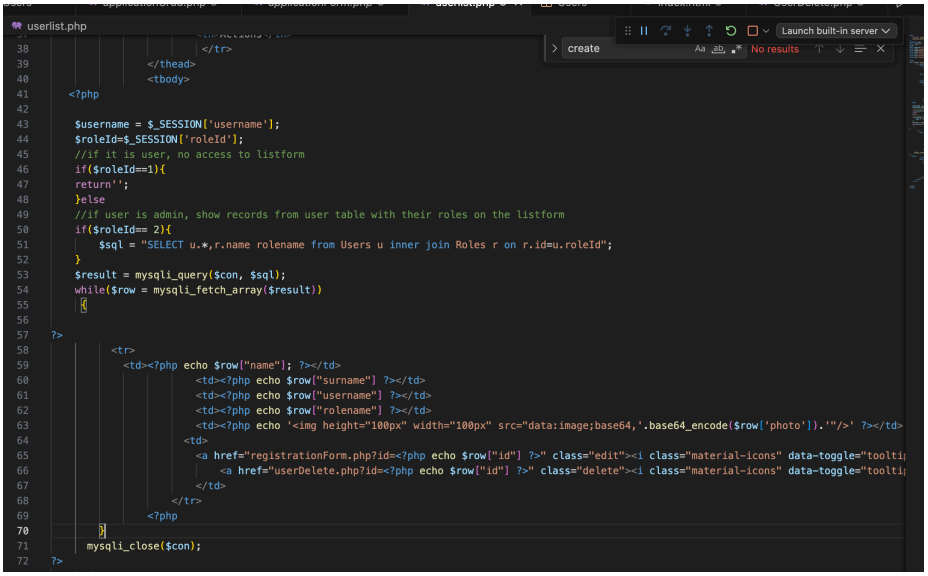

Figure 19.Session and Role Control PHP

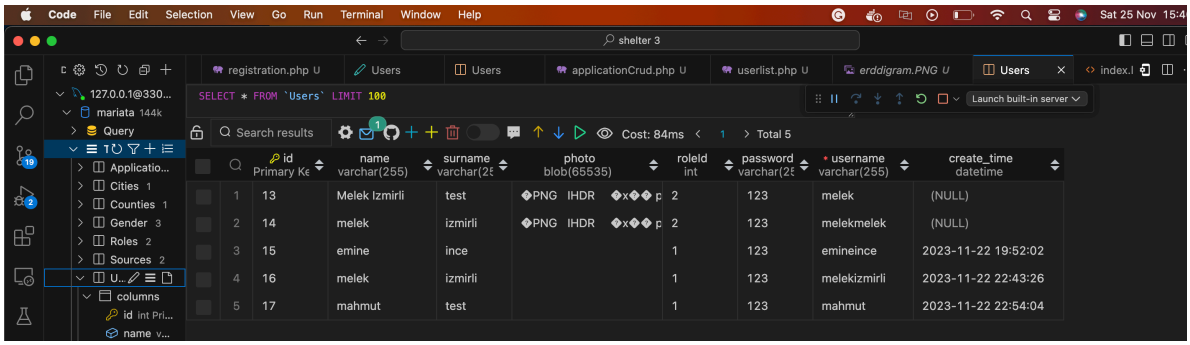

Figure 20.Users Table

The PHP code and SQL queries needed to delete selected rows in the list form are as follows:

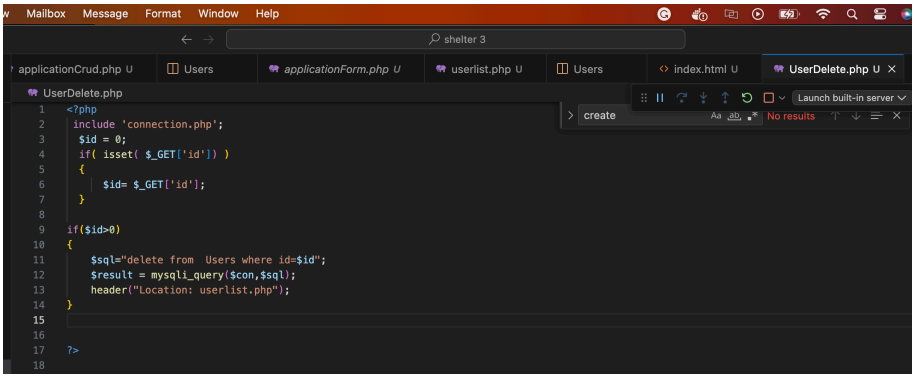

Figure 21.Delete Query with PHP

To update a user in the user table using PHP, the registrationForm.php is employed as a populated version of the registration.php originally utilized for sign-up. Furthermore, a dropdown menu has been included to modify roles for specific users.

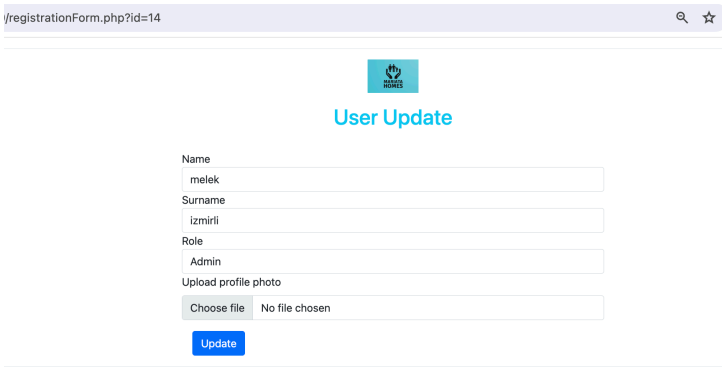

Figure 22.CRUD- Update Existing User

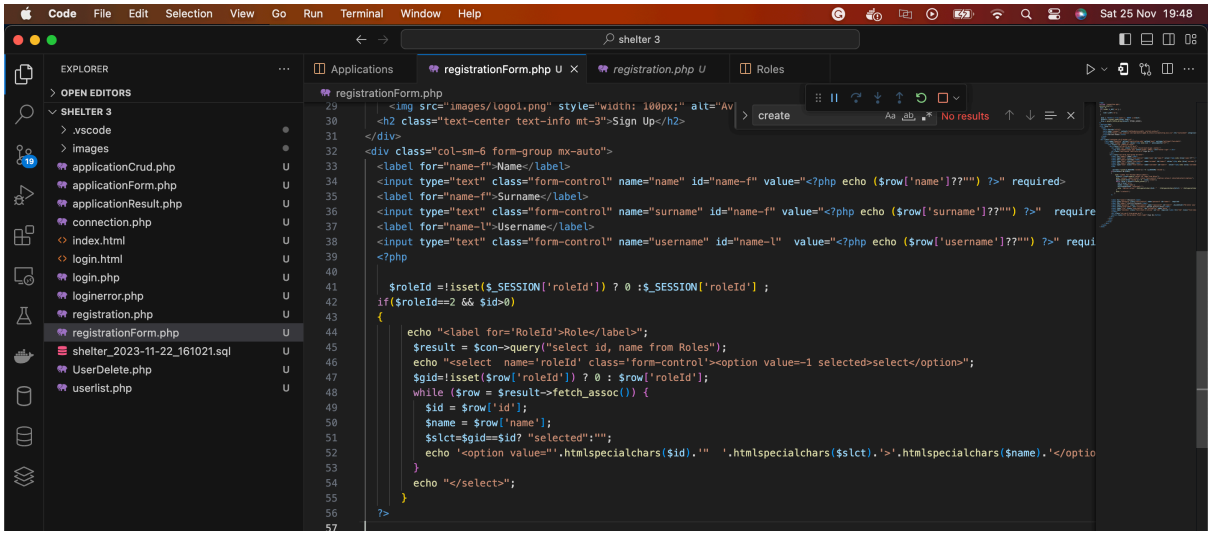

Figure 23.CRUD Update User- Adding Extra Dropdown for Assigning Roles

CONCLUSION

This document describes how this digital transformation is implemented. Many lessons have been learned in the project development process. One of the most crucial is understanding how versatile and dynamic the system development process is. Sometimes, regardless of how detailed the requirement analysis is conducted, producing a design that aligns with SDLC principles, conventions, or the target audience is not as straightforward as it seems. Occasionally, a small change made at the UI or database level can have a significant impact on all components of the product. Moreover, being overly tied to the scenario can sometimes complicate reaching the desired product output. Each stakeholder in the design and implementation process should sometimes learn to put themselves in the end-user's shoes while also having a grasp of technical possibilities. The best way to achieve this is through more research and examining similar examples.

In this study, it may be necessary to conduct a comprehensive usability test and various functionality tests. The project would have achieved a higher quality if there had been an extended deadline, allowing for more thorough testing and refinement.

Further studies may focus on enhancing UXD principles and software quality. Implementing a more responsive design can enable users to access and apply through various devices. Moreover, efforts to improve code quality in terms of safeguarding personal information will contribute to both the product's quality and security.

REFERENCES

Baumer, D., Bischofberger, W., Lichter, H., and Zullighoven, H. (1996, March) "User interface prototyping-concepts, tools, and experience." In: *Proceedings of IEEE 18th International Conference on Software Engineering,* pp. 532-541. IEEE.

Batini, C., Ceri, S., and Navathe, S. (1992) "Conceptual Database Design: an Entity-Relationship Approach." Benjamin/Cummings Publishing, Redwood City, CA.

Fowler, M. (2018) *UML Distilled: A Brief Guide to the Standard Object Modeling Language.* Addison-Wesley Professional.

Fowler, M. (2003) *UML Distilled: A Brief Guide to the Standard Object Modeling Language.* Addison-Wesley Professional.

Kennedy, D., 2000, July. "Database Design and the Reality of Normalisation." In *Proceedings of the 13th Annual NACCQ Conference, July.*

Kušek, M., Dešić, S., Gvozdanović, D. (2001) "UML Based Object-oriented Development: Experience with Inexperienced Developers." In: *6th International Conference on Telecommunications,* June 13–15, 2001, Zagreb, Croatia.

Kulak, D., and Guiney, E. (2012) *Use cases: requirements in context.* Addison-Wesley.

Lind, J., Song, I-Y., and Park, E.K. (1995) "Object-Oriented Analysis: A Study in Diagram Notations." *Journal of Computer and Software Engineering,* Vol. 3, No. 1 (Winter 1995), pp. 133-165.

McFadden, F. R. and Hoffer, J. A. (1994) "Modern Database Management". Fourth Edition, The Benjamin/Cummings Publishing Company.

Mule, S.S., Waykar, Y., and Mahavidyalaya, S.V. (2015) "Role of USE CASE diagram in s/w development." *International Journal of Management and Economics.*

Prashant, S.G. (2012) "Simplifying Use Case Models using CRUD Patterns." *International Journal of Soft Computing and Engineering (IJSCE)*, ISSN: 2231-2307, Volume-2 Issue-2, May 2012.

Rob, P. and Coronel, C. (1997) "Database Systems: Design, Implementation and Management", Course Technology.

Song, I.Y., Evans, M., and Park, E.K. (1995) "A comparative analysis of entity-relationship diagrams." *Journal of Computer and Software Engineering,* 3(4), pp. 427-459.