Pernet Club Platform

Comp204 Term Project

Final Report

Instructor

Dr. Samet Tonyalı

Group Members

Ahmed Demirezen

Ahmet Caner Sağır

Emre Kılıç (Contact)

İsmet Enes İnce

Mehtap Saatçı

Serkan Biçer

TABLE OF CONTENTS

TABLE OF CONTENTS 2

TABLE OF FIGURES 3

1. ABSTRACT 4

2. SUMMARY 4

3. GENERAL DESCRIPTION 4

4. REQUIREMENT ANALYSIS 5

5. SPECIFICATIONS 5

6. TOOLS/IDEs 6

7. USE-CASE DIAGRAM 7

8. ER DIAGRAM 8

9. NORMALIZATION 10

10. DATABASE SCHEMA 18

11. DESIGN PHILOSHOPY 25

12. FUNCTIONAL DEPENDENCIES 26

13. ER TO RELATIONAL MAPPING 29

14. HIGH-LEVEL DESIGN 30

15. SCRIPTS(DDL) 31

16. TABLE CONSTRAINT SCRIPTS(DDL) 35

17. SCRIPTS(DML) 37

18. SCRIPTS (JavaScript samples) 38

19. VIEWS 40

TABLE OF FIGURES

Figure 1. Use Case Diagram 7

Figure 2. Pernet Platform Website ER Diagram (Crow's Foot) 8

Figure 3. Pernet Platform Website ER Diagram (Chen's) 9

Figure 4. Deriving Badges from UserBadges 11

Figure 5. Deriving Languages from UserLanguages 12

Figure 6. High-Level Design 30

Figure 7. Login Page 40

Figure 8. Home Page 41

Figure 9. Portfolio Page 41

Figure 10. Edit user details such as user education 42

1. ABSTRACT

Nowadays, Database Management Systems are in a very important position in terms of keeping data and providing communication. This article contains information on the implementation of a database system for a student club. The project aims to provide a better service to the members of the club and to communicate more effectively. Thanks to the website that is created for this project, users will be able to easily access events, and career opportunities and expand their networks. Besides, club admins will be able to follow their users better. In addition, a comfortable communication environment is provided with the messaging feature. Finally, MySQL and Python are our tools to create our website.

2. SUMMARY

This report provides information about the database's structure, design, and content created to enable faster and more effective communication between Personal Network Club members and the club's admins. In this project, SQL will be used to keep the users' information and provide communication. Moreover, the website will be deployed to an Apache Web Server hosted in a Windows Server so that the users are assured to benefit from the website whenever they want. Also, Python was used for the back-end development of the project with the Flask framework, and Vue.js was used for the front-end development of the project. In the following, the report contains information about the general description of the project, requirements analysis, specifications, tools/IDEs, ER schema, philosophy, cardinalities, and some other detailed information.

3. GENERAL DESCRIPTION

With the website to be created within the scope of this project, it is aimed that the members of the Personal Networking Club can easily follow the events and career opportunities and create a network. In addition, the club will be able to follow its members easily and contribute to their career development better. First, users will be able to log in to the site with their e-mails and passwords. Second, users will be able to see the sent posts on the main page of the website and can reach the person they want with the message feature. Moreover, users will be allowed to create their portfolios and there will be options to add a CV and cover letter. In this project, the website was preferred in terms of ease of use and accessibility, and it is also possible to reach this site over

a mobile phone. Finally, the database will hold users' data, and posts. This article will explain how the database is designed, what methods are used and what it includes.

4. REQUIREMENT ANALYSIS

Pernet Club Platform invests in its members comprehending their needs and providing the best job opportunities, courses, and events. This articulates the presence of a mutual relationship between Personal Networking authorities and the club members. Therefore, the website fully requires the authority of the Personal Networking Club over the ownership and financial maintenance of the website.

First, the website is unconditionally bounded to the valid membership of prospective users. That is, only the members of the Personal Networking Club are going to be able to benefit from the services provided on the website. Thus, the website strictly requires a digital copy, desirably in Microsoft Excel format, of the member list provided by the Personal Networking Club. Thereby, the website will provide an option to set up a password for those on the member list.

Second, the website requires the users to input their professional information to properly continue to pursue its services. This information typically requires a user's first name, last name, AGU mail, contact details, professional skills, CV, Cover Letter, licenses & certifications, and experience details. Subsequently, the Personal Networking authorities will analyze the user data and provide the best internship and job opportunities, courses, and events via posts.

5. SPECIFICATIONS

The website will provide job opportunities, events, and courses to its users via the administrators. The administrators will analyze the information given on the website and manage the users accordingly. Therefore, there are certain differences between the administrators and users as the administrators are more privileged than the users.

First, both users and administrators can log in using their e-mail and password but only administrators will provide the emails that will be accepted into the system, and those emails are the ones of club members. This acceptance stage will be maintained through the "Forgot password" option. Valid users will be able to set their passwords with this option.

Second, the user interface of administrators differs from the users. Administrators will have additional user interface tools to give badges to the users. Also, administrators will be able to remove or edit all posts sent by the users, but this is not valid for the users. Users are only able to remove or edit their posts, not the others. Moreover, it is worth mentioning that the user interface of the website will be responsive so that both Android and IOS will be able to use the website, as well as desktop users.

6. TOOLS/IDEs

JavaScript: It is used for backend development of the project like making an API with the Nuxt.js framework.

Node.js: It is an infrastructure for the Vue.js framework to work, and it also provides us with access to the library and other frameworks required for the frontend.

Vue.js: The framework is used for the frontend development of the project.

Quasar: Quasar is an MIT licensed open-source Vue.js based framework. It is used for frontend design.

Visual Studio Code: It is used for editing source code, debugging, and running tasks.

MySQL Workbench: It is used for SQL development, relational data mapping, and database design.

Windows Server 2012: The operating system is used to store the project files and host the project.

FileZilla: It is used for transferring the file to the virtual server.

XAMPP: A program used in a virtual server by running Apache, MySQL and FileZilla servers from a single point.

Apache Web Server: It is used for managing HTTP requests, server-side programming, and deployment.

GitHub/Git: It is used for version control and building a codebase of the project.

Slack: It is used for agile development and establishes communication channels among the members of the development team.

7. USE-CASE DIAGRAM

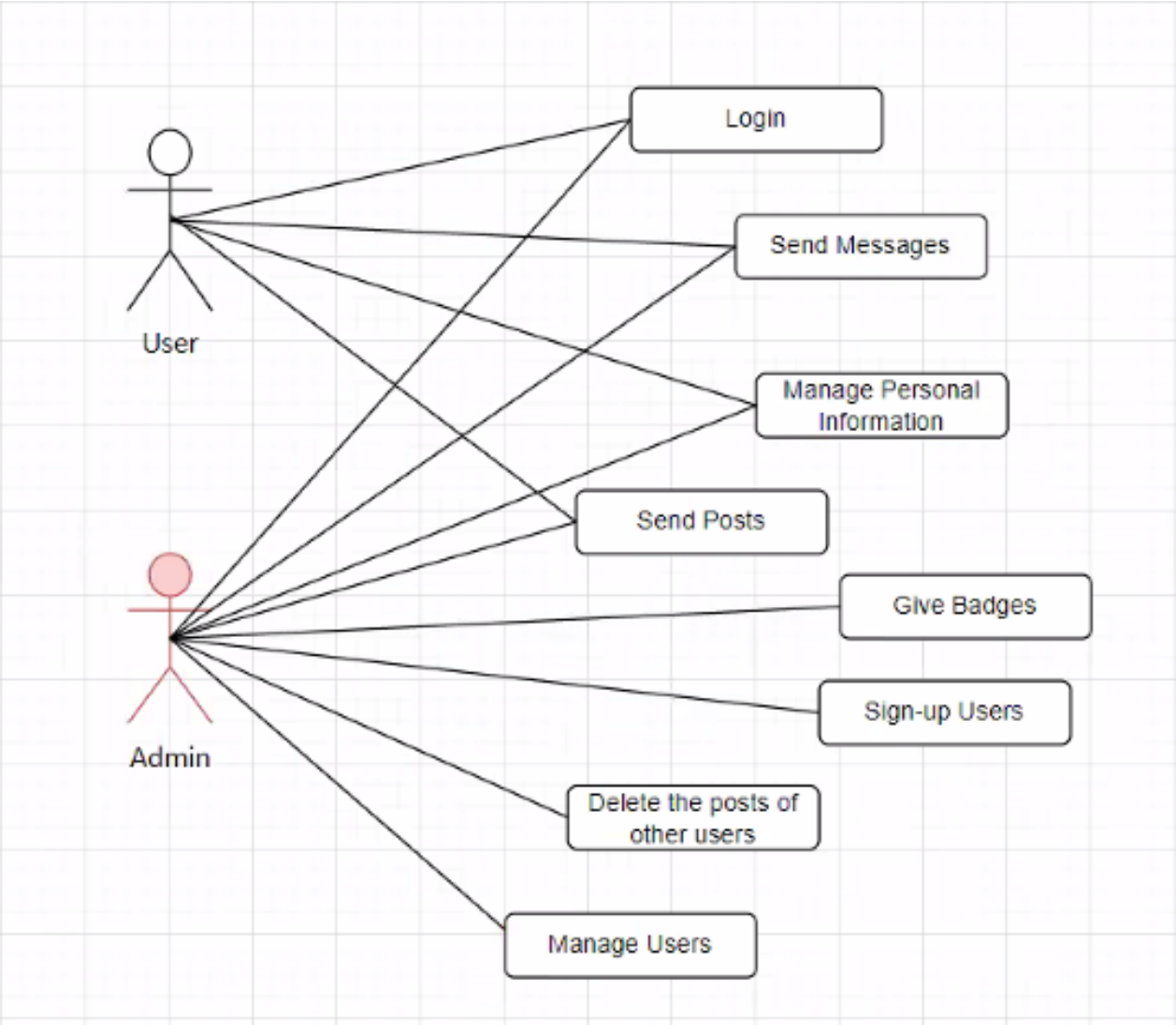

Figure 1. Use Case Diagram

Typical users and administrators share common properties. Both users and administrators can log in using their e-mail and password, manage personal information, and send posts visible to all users. However, only admins can sign-up users to the website, give badges for rewarding user activities, and manage the posts and users for the reliability and consistency of the website.

8. ER DIAGRAM

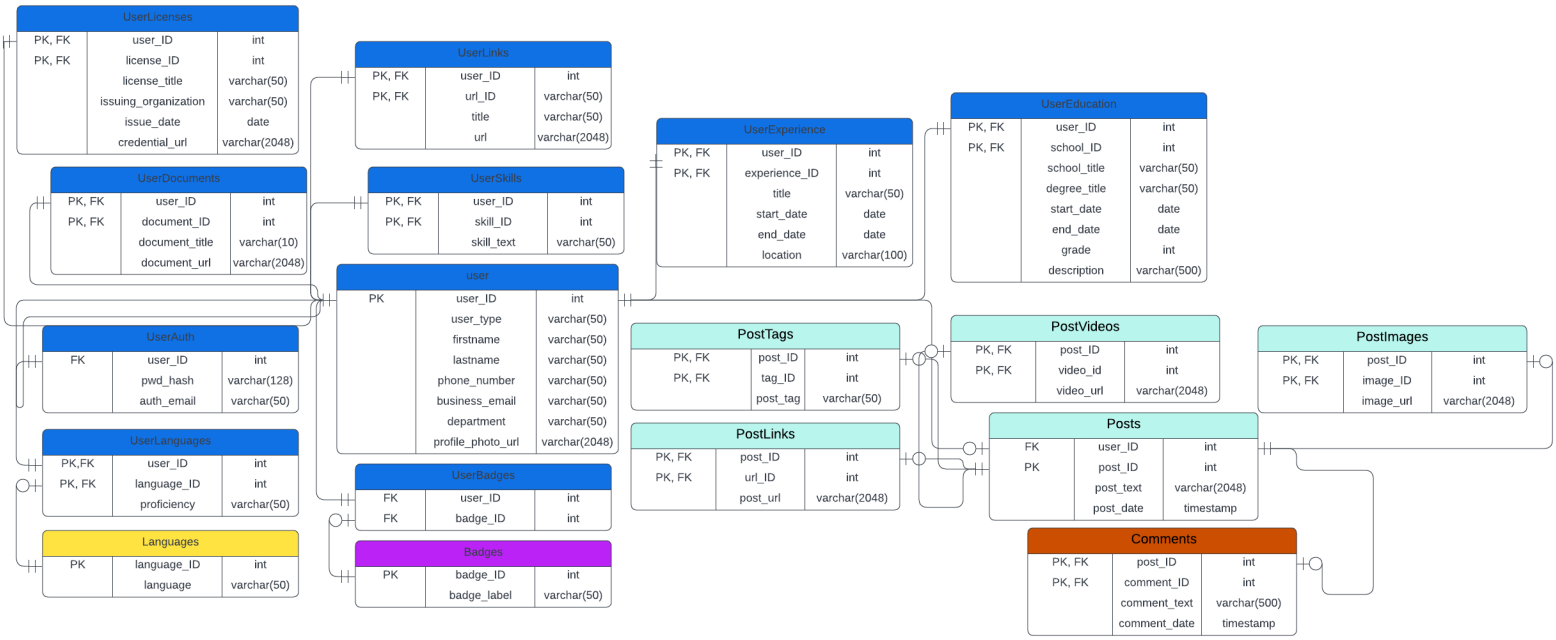

Figure 2. Pernet Platform Website ER Diagram (Crow's Foot)

Figure 3. Pernet Platform Website ER Diagram (Chen's)

9. NORMALIZATION

UserAuth table: Each column contains atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions of the first normal form(1NF). Next, all attributes are functionally dependent on *user_ID*. Also, there is no partial dependency because the table has no compound key. Therefore, this table satisfies the conditions of the second normal form(2NF). Next, this table does not have any transitive dependency since *pwd_hash* and *auth_email* are dependent on *user_ID, a primary key*. Therefore, this table satisfies the conditions of the third normal form(3NF). Subsequently, this table has a primary key to satisfy the conditions of Boyce-Codd normal form (BCNF), although *pwd_hash* forces every functional dependency to be unique. Therefore, *user_ID* must be the primary key of this table. Thereby, every functional dependency is unique by *user_ID* and this table satisfies the Boyce-Codd normal form (BCNF). Next, this table does not have multivalued attributes as it does not show any unnecessary data repetition. Therefore, this table satisfies the conditions of the fourth normal form(4NF). Finally, this table satisfies the fifth normal form(5NF) as you cannot obtain this table by joining two tables.

Users table: Each column contains atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form(1NF). Next, all attributes are functionally dependent on *user_ID*. Also, there is no partial dependency because the table has no compound key. Thus, this table satisfies the conditions of the second normal form(2NF). Next, this table does not have any transitive dependency since all attributes are dependent on *user_ID, a primary key*. Therefore, this table satisfies the conditions of the third normal form(3NF). Subsequently, this table satisfies the conditions of Boyce-Codd normal form (BCNF) since the super key is *user_ID* and allows functional dependencies to be uniquely identified. Next, this table does not have multivalued attributes since users are not allowed to save multiple values for the given attributes. For example, a user is not able to create multiple phone numbers, emails, and departments with no unnecessary data repetition. Therefore, this table satisfies the conditions of the fourth normal form(4NF). Finally, this table satisfies the

fifth normal form(5NF) as every attribute is dependent on *user_ID* so that no multiple tables join in this table.

UserBadges table: Each column contains atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form(1NF). Next, all attributes are functionally dependent on *user_ID* and *badge_ID*. However, *badge_text* should be only dependent on *badge_ID*. Therefore, there is a partial dependency, and we should derive another table as **Badges** so that there will be unique badges that can be assigned to the members by the website administrators, eliminating the case in which users create random badges. Thereby, this table satisfies the conditions of the second normal form(2NF) as no partial dependency is left in **UserBadges**.

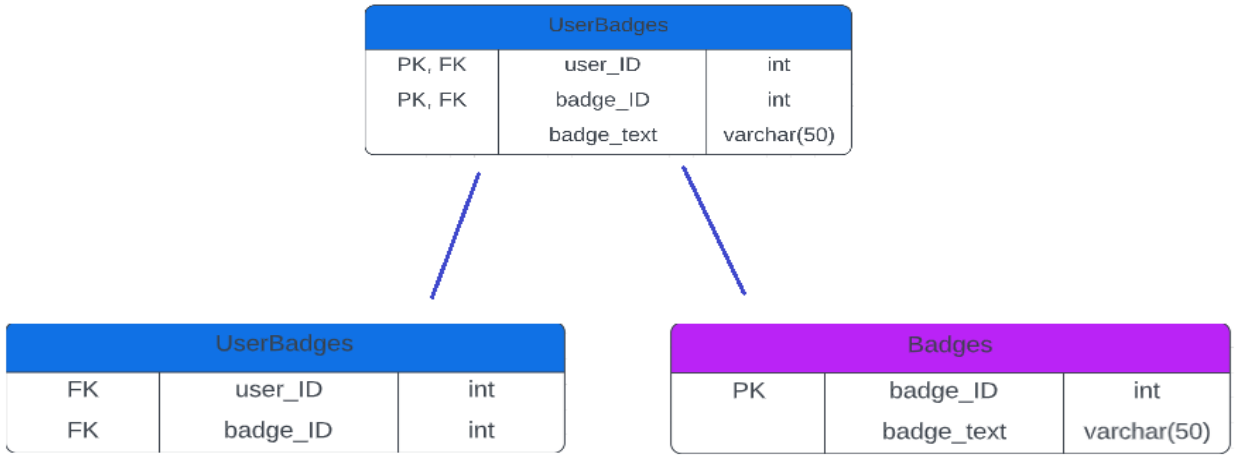

Figure 4. Deriving Badges from UserBadges

Next, this table does not have any transitive dependency since *badge_ID is* dependent on *user_ID, a primary key*. Therefore, **UserBadges** satisfies the conditions of the third normal form(3NF). Subsequently, this table satisfies the conditions of Boyce-Codd normal form (BCNF) since the super key is a combination of *{user_ID, badge_ID}*, allowing every functional dependency to be uniquely identified. Next, this table does not have multivalued attributes as it has two columns. Therefore, this table satisfies the conditions of the fourth normal form(4NF). Finally, having two columns, this table satisfies the fifth normal form(5NF) as you cannot obtain this table by joining two tables.

UserLanguages table, each column contains atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions to be in the first normal form(1NF). Next, all attributes are functionally dependent on *user_ID* and *language_ID*. However, *language* should not be dependent on *user_ID*. Therefore, there is a partial dependency, and we should derive another table as **Languages** so that there will unique and a finite number of languages. For instance, a user can select a language from an enumerated list and define his/her proficiencies in the selected language on behalf of herself/himself, so that the possibility of randomly defined languages is eliminated. Thereby, this table satisfies the conditions of the second normal form(2NF) as no partial dependency is left in **UserLanguages**.

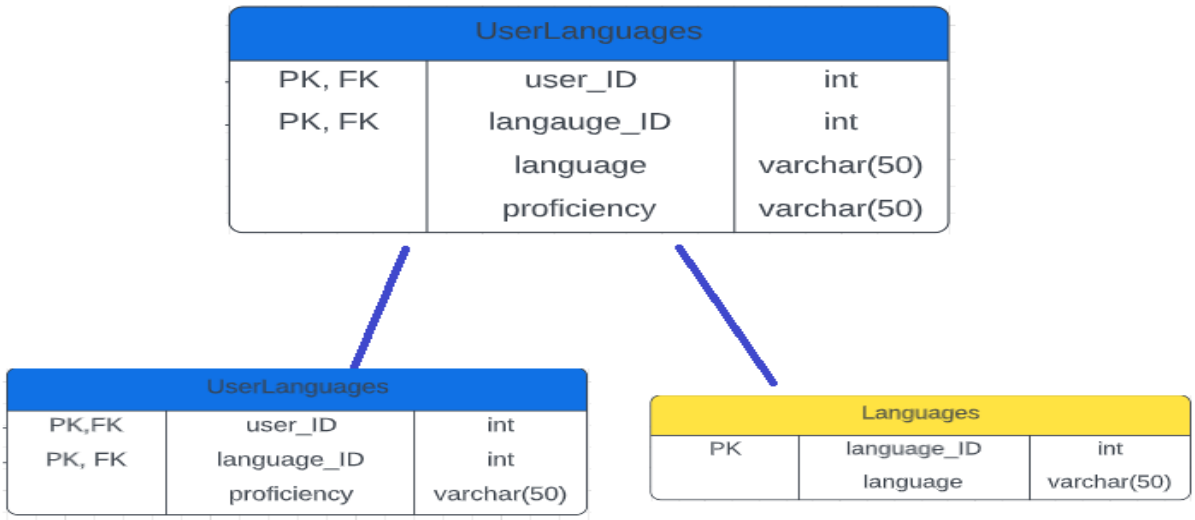

Figure 5. Deriving Languages from UserLanguages

Next, this table does not have any transitive dependency since *proficiency* is dependent on a part of the primary key. Therefore, **UserLanguages** satisfies the conditions of the third normal form(3NF). Next, this table does not have multivalued attributes since users are allowed to set just one proficiency for each language they know. Therefore, this table satisfies the conditions of the fourth normal form(4NF). Finally, this table satisfies the fifth normal form(5NF) as you cannot obtain this table by joining two tables.

UserDocuments table: Each column contains atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form(1NF). Next, all attributes are functionally dependent on *user_ID* and *document_ID*.

Also, there is no partial dependency since every document must be unique and has a unique user for better tracking of the documents and improving security. Finally, this table satisfies the conditions of the second normal form(2NF) as no partial dependency is present in **UserDocuments**. Next, this table does not have any transitive dependency since all attributes are dependent on a part of the primary key. Therefore, **UserDocuments** satisfies the conditions of the third normal form(3NF). Next, this table satisfies the conditions of Boyce-Codd normal form (BCNF) since the super key is a combination of *{user_ID, document_ID}*, allowing every functional dependency to be uniquely identified. Next, this table does not have multivalued attributes since *document_title and document_url* are related to *document_ID*. Therefore, this table satisfies the conditions of the fourth normal form(4NF). Finally, this table satisfies the fifth normal form(5NF) as you cannot obtain this table by joining two tables.

UserSkills table, each column contains atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions to be in the first normal form(1NF). Next, all attributes are functionally dependent on *user_ID* and skill_*ID*. Although there might be the same skills for different users, they are still specific to users as a compound key is required to define a skill. Therefore, there is no partial dependency. Finally, this table satisfies the conditions of the second normal form(2NF) as no partial dependency is present in **UserSkills**. Next, this table does not have any transitive dependency since *skill_text is* dependent on a part of the primary key. Therefore, **UserSkills** satisfies the conditions of the third normal form(3NF). Next, this table satisfies the conditions of Boyce-Codd normal form (BCNF) since the super key is a combination of *{user_ID, skill_ID}*, allowing every functional dependency to be uniquely identified. Next, this table does not have multivalued attributes since *skill_text* is related to *skill* so unnecessary data repetition is prevented. Therefore, this table satisfies the conditions of the fifth normal form(4NF). Finally, this table satisfies the fifth normal form(5NF) as you cannot obtain this table by joining two tables.

User License table: Each column has atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Every non-key attribute *(license_title, issuing_organization, issue_date, credential_url)* are dependent on compound key *(user_id, license_id)*. Also, there is no partial dependency since all licenses must be unique and have a unique user. Thus, this table satisfies the

conditions of the second normal form **(2NF).** Furthermore, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since any non-primary attribute does not derive the primary key. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

User Links table: Each column has atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Every non-key attribute *(url)* is dependent on the compound key *(user_id, url_id)*. Also, there is no partial dependency since all links must be unique and have a unique user. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since any non-primary attribute does not derive the primary key. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

User Experience table: Each column has atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Every non-key attribute *(title, start_date, end_date, location)* are dependent on compound key *(user_id, experience_id)*. Also, there is no partial dependency since all experiences must be unique and have a unique user. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since any non-primary attribute does not derive the primary key. Moreover, since the table has a functional dependency and non-primary attributes are related

to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

User Education table: Each column has atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Every non-key attribute *(school_title, degree_title, start_date, end_date, grade, description)* are dependent on compound key *(user_id, school_id)*. Also, there is no partial dependency since all educational information must be unique and has a unique user. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since any non-primary attribute does not derive the primary key. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

Posts table: Each column contains atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form**(1NF).** Next, all attributes are functionally dependent on *post_ID*. Also, there is no partial dependency because the table has no compound key. Thus, this table satisfies the conditions of the second normal form(**2NF**). Next, this table does not have any transitive dependency since all attributes are dependent on *post_ID, a primary key*. Therefore, this table satisfies the conditions of the third normal form(**3NF**). Subsequently, this table satisfies the conditions of Boyce-Codd normal form (**BCNF**) since the super key is *post_ID* allowing all functional dependencies to be uniquely identified. Next, this table does not have multivalued attributes since users are not allowed to save multiple values for the given attributes. For example, a post cannot be created by different dates and texts, assuring no unnecessary data repetition. Therefore, this table satisfies the conditions of the fourth normal form(**4NF**). Finally, this table satisfies the fifth normal form(5NF) as every attribute is dependent on *post_ID* so that no multiple tables join in this table.

Post Links table, each column has atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions to be in the first

normal form **(1NF).** Only the non-key attribute "*postUrl*" is dependent on the compound key *(url_id, post_id)*. Also, there is no partial dependency since all links must be unique and have a unique post. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since the super key is a combination of *{post_ID, url_ID}*, allowing every functional dependency to be uniquely identified. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

Post Videos table, each column has atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Only the non-key attribute "*video_url* "is dependent on the compound key *(video_id, post_id)*. Also, there is no partial dependency since all videos must be unique and have a unique post. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since the super key is a combination of *{post_ID, video_ID}*, allowing every functional dependency to be uniquely identified. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

Post Tags table, each column has atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Only the non-key attribute "*post_tag*" is dependent on the compound key *(tag_id, post_id)*. Also, there is no partial dependency since all links must be unique and have a unique post. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend

on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since the super key is a combination of *{post_ID, tag_ID}*, allowing every functional dependency to be uniquely identified. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

Post Images table, each column has atomic values, each column contains values of the same type, and each column has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Only the non-key attribute "*image_url* "is dependent on the compound key *(image_id, post_id)*. Also, there is no partial dependency since all images must be unique and have a unique post. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since any non-primary attribute does not derive the primary key. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

Comments table: Each column has atomic values, each column contains values of the same type, and each has a unique name. Thus, this table satisfies the conditions to be in the first normal form **(1NF).** Every non-key attribute (*comment_date, comment_text)* is dependent on compound key *(comment_id, post_id)*. Also, there is no partial dependency since all comments must be unique and have a unique post. Thus, this table satisfies the conditions of the second normal form **(2NF).** Moreover, this table does not have any transitive dependency since any non-primary attributes do not depend on other non-primary attributes. Therefore, this table satisfies the conditions of the third normal form **(3NF).** Subsequently, this table satisfies the conditions of Boyce-Codd normal form **(BCNF)** since the super key is a combination of *{comment_ID, post_ID}*, allowing every functional dependency to be uniquely identified. Moreover, since the table has a functional dependency and non-primary attributes are related to each other, this table satisfies the conditions

of the fourth normal form **(4NF).** Finally, since the table does not have join dependency, this table satisfies the fifth normal form **(5NF).**

10. DATABASE SCHEMA

- **Users**

Column	Type	Key
user_ID	INT	PK
user_type	VARCHAR(50)	
firstname	VARCHAR(50)	
lastname	VARCHAR(50)	
phone_number	VARCHAR(50)	
business_email	VARCHAR(50)	
department	VARCHAR(50)	
Profile_photo_url	VARCHAR(2048)	

1	**member**	**Robert**	**Langdon**	**+90555555555**	**r.langdon@mail.com**		**https://tr.web.img4.acsta.net/newsv7/20/01/30/08/17/1379969.jpg**

- **User Badges**

Column	Type	Key
user_ID	INT	FK
badge_ID	INT	FK

1	**5**

- **User Auth**

Column	Type	Key
pwd_hash	VARCHAR(128)	PK
user_ID	INT	FK
auth_email	VARCHAR(50)	

948892d435d05cfdab671fa59c02f96f8b9871d3e64920e26691c9e9ed4425d9	10	r.langdon@mail.com

- **User Languages**

Column	Type	Key
user_ID	INT	PK, FK
language_ID	INT	PK, FK
proficiency	VARCHAR(50)	

1	2	Advanced

- **User Skills**

Column	Type	Key
user_ID	INT	PK, FK
skill_ID	INT	PK, FK
skill_text	VARCHAR(50)	

1	3	Web Development

- **User Links**

Column	Type	Key
user_ID	INT	PK, FK
url_ID	VARCHAR(50)	PK, FK
url	VARCHAR(2048)	

1	2	https://www.linkedin.com/in/robert-langdon-a99b54135/

- **User Experience**

Column	Type	Key
user_ID	INT	PK, FK
experience_ID	VARCHAR(50)	PK, FK
title	VARCHAR(50)	
start_date	DATE	
end_date	DATE	
location	VARCHAR(100)	

6	2	Professor	01.02.2001	08.06.20013	Cambridge, Massachusetts

- **User Documents**

Column	Type	Key

user_ID	INT	PK, FK
document_ID	VARCHAR(50)	PK, FK
document_title	VARCHAR(50)	
document_url	VARCHAR(2048)	

58	5	CV	https://www.robertlangdon.com/resume/

- **User Licences**

Column	Type	Key
user_ID	INT	PK, FK
licence_ID	INT	PK, FK
licence_title	VARCHAR(50)	
issuing_organization	VARCHAR(50)	
issue_date	DATE	
credential_url	VARCHAR(2048)	

6	12	Google Crash Python Course	Coursera	08.06.1973	https://credentials.coursera.com/robert-langdon-1973

- **User Education**

Column	Type	Key
user_ID	INT	PK, FK
school_ID	INT	PK, FK
school_title	VARCHAR(50)	

degree_title	VARCHAR(50)	
start_date	DATE	
end_date	DATE	
grade	INT	
description	VARCHAR(500)	

5	AGU	Bachelor's of Science	17.10.2019	24.06.2024	2	Activities and societies: Personal Networking, IEEE,

- **Posts**

Column	Type	Key
user_ID	INT	PK, FK
post_ID	INT	PK, FK
post_text	VARCHAR(50)	
post_date	TIMESTAMP	

1	9	Hello, I would like to share an internship opportunity.	06/02/2022 08:52:48

- **Post Tags**

Column	Type	Key

post_ID	INT	PK, FK
tag_ID	INT	PK, FK
post_tag	VARCHAR(50)	

9	1	internship

- **Post Videos**

Column	Type	Key
post_ID	INT	PK, FK
video_ID	INT	PK, FK
video_url	VARCHAR(2048)	

9	1	https://www.youtube.com/watch?v=KynMm9ucC94

- **Post Links**

Column	Type	Key
post_ID	INT	PK, FK
tag_ID	INT	PK, FK
post_url	VARCHAR(2048)	

9	3	https://www.europelanguagejobs.com/blog/How-to-Find-Internship-in-Europe

- **Post Images**

Column	Type	Key
post_ID	INT	PK, FK
url_ID	INT	PK, FK
post_url	VARCHAR(2048)	

9	1	https://www.europelanguagejobs.com/utilities/resize-img/100?img=https://www.europelanguagejobs.com/uploads/posts/Internship.jpg

- **Comments**

Column	Type	Key
post_ID	INT	PK, FK
comment_ID	INT	PK, FK
comment_text	VARCHAR(500)	
comment_date	TIMESTAMP	

9	5	Sir thank you so much sir	06/02/2022 08:52:48

- **Badges**

Column	Type	Key
badge_ID	INT	PK
badge_text	VARCHAR(50)	

1	**Active Member**

- **Languages**

Column	Type	Key
language_ID	INT	PK
language	VARCHAR(50)	

1	**English**

11. DESIGN PHILOSHOPY

Entities

- Users
- UserDocuments
- UserSkills
- UserLicenses
- Languages
- UserLanguages
- UserExperience
- UserEducation
- Badges
- UserBadges
- UserAuth
- UserLinks
- Posts
- PostVideos
- PostImages
- PostTags
- PostLinks
- Comments

12. FUNCTIONAL DEPENDENCIES

- Users = {user_ID} --> {*user_type, firstname, lastname, phone_number, business_email, department, profile_photo_url}*
- UserDocuments = {*user_ID, document_ID} --> {document_title, document_url}*
- UserSkills = {*user_ID, skill_ID} -->{ skill_text}*
- UserLicenses = {*user_ID, license_ID} --> {license}*
- Languages = {*language_ID} --> {language}*
- UserLanguages = {*user_ID, language_ID} --> {proficiency}*
- UserExperience ={*user_ID, experience_ID} --> {title, start_date, end_date, location}*
- UserEducation = {*user_ID, school_ID} --> {school_title, degree_title, start_date, end_date, grade, description}*
- Badges = {*badge_ID} --> {badge_text}*
- UserBadges = {*user_ID, badge_ID*}
- UserAuth = {*user_ID} --> { pwd_hash, auth_email}*
- UserLinks = {*user_ID, url_ID} --> {title, url}*
- Posts = {*user_ID, post_ID} --> {post_text, post_date}*
- PostVideos = {*post_ID, video_id} --> {video_url}*
- PostImages = {*post_ID, image_ID} --> {image_url}*
- PostTags = {*post_ID, tag_ID} --> {post_tag}*
- PostLinks = {*post_ID, url_ID} --> {post_url}*
- Comments = {*post_ID, comment_ID} --> {comment_date, comment_text}*

Users is the main entity of the project, and it contains the general information of the user. It has weak relationships with UserDocuments, UserSkills, UserLicences, UserLanguages, UserExperiece, UserEducation, UserBadges, UserLinks, and UserAuth. The user_ID attribute is the primary key, thence it is a strong entity. In short, users sustain all the tables around itself.

UserDocuments is a weak entity dependent on Users. The general purpose of this entity is to save and manage the information of documents uploaded by the users. It has a foreign key that comes from the Users entity. The pair of user_ID and document_ID is a composite primary key in the entity.

UserSkills is a weak entity dependent on Users. The general purpose of this entity is to save and manage the skills of the users. It has a foreign key that comes from the Users entity. The pair of user_ID and skill_ID is a composite primary key in the entity.

UserLicences is a weak entity dependent on Users. The general purpose of this entity is to save and manage the licenses and certificates of the users. It has a foreign key that comes from the Users entity. The pair of user_ID and license_ID is a composite primary key in the entity. Therefore, it is a weak entity.

Languages is a strong entity. It keeps data on the languages spoken worldwide. language_ID is a primary key in the entity. **UserLanguages** is dependent on this table since every language an user has should be accordingly recorded with the user's proficiency.

UserLanguages is a weak entity dependent on Users. It keeps data on the languages spoken by the users with their proficiency level. It has a foreign key that comes from the Users entity. The pair of user_ID and language_ID is a composite primary key in the entity.

UserExperience is a weak entity dependent on Users. It keeps the data about the work experience of the users. It has a foreign key that comes from the Users entity. The pair of user_ID and experience_ID is a composite primary key in the entity.

UserEducation is a weak entity dependent on Users. It keeps the data on the educational backgrounds of the users. It has a foreign key that comes from Users entity. The pair of user_ID and school_ID is a composite primary key.

Badges is a strong entity. It keeps data of badges that will be assigned to successful members by the administrators. badge_ID is a primary key in the entity. **UserBadges** is dependent on this table since every badge an user has should be accordingly recorded.

UserBadges is a weak entity dependent on Users. It keeps the data of badges that are given by the authorities to the users. It has a foreign key that comes from Users entity. The pair of user_ID and badge_ID is a composite primary key.

UserAuth is a weak entity dependent on Users. This entity enables to secure the data by permitting only authenticated users to access their sources. It has a primary key user_ID.

UserLinks is a weak entity dependent on Users. This entity saves the users personal access to links to Github, LinkedIn, and other external links. It has a foreign key that comes from the Users entity. The pair of url_ID and user_ID is a composite primary key.

Posts is a weak entity to store posts shared by the users. The entity holds the text and the date of the post for each user. It has a foreign key that comes from the Posts entity. The pair of post_ID and user_ID is a composite primary key.

PostVideos is a weak entity dependent on Posts. It stores the URL of the videos contained in the posts. It has a foreign key that comes from the Posts entity. The pair of post_ID and video_ID is a composite primary key.

PostImages is a weak entity dependent on Posts. It stores the URL of images contained in the posts. It has a foreign key that comes from the Posts entity. The pair of post_ID and image_ID is a composite primary key.

PostTags is a weak entity dependent on Posts. It stores the information about the tags of a post. It has a foreign key that comes from the Posts entity. The pair of post_ID and tag_ID is a composite primary key.

PostLinks is a weak entity dependent on Posts. It stores the information about external links in the posts. It has a foreign key that comes from the Posts entity. The pair of post_ID and url_ID is a composite primary key.

Comments is a weak entity dependent on Posts. It stores every comment that is written in posts. The pair of post_ID and comment_ID is a composite primary key.

Cardinalities

ONE (AND ONLY ONE) to ONE (AND ONLY ONE)

- Users – UserExperience
- Users – UserBadges
- Users – UserEducation
- Users – UserLinks

- Users – UserSkills
- Users – UserDocuments
- Users – UserAuth
- Users – UserLicences
- Users – UserLanguages

ONE (AND ONLY ONE) to ZERO OR ONE

- Users – Posts
- Posts – PostImages
- Posts – PostVideos
- Posts – PostTags
- Posts – PostLinks
- Posts – Comments
- Badges – UserBadges
- Languages – UserLanguages

13. ER TO RELATIONAL MAPPING

- **Strong Entities**
    - Users (user_ID, user_type, firstname, lastname, phone_number, business_email, department, profile_photo_url)
    - Languages *(language_ID, language)*
    - Badges*(badge_ID, badge_text)*
    - Posts *(user_ID, post_ID, post_text, post_date)*
- **Weak Entities**
    - UserDocuments *(user_ID, document_ID, document_title, document_url)*
    - UserSkills *(user_ID, skill_ID, skill_text)*
    - UserLicenses *(user_ID, license_ID, license)*
    - UserLanguages *(user_ID, language_ID, proficiency)*
    - UserExperience *(user_ID, experience_ID, title, start_date, end_date, location)*
    - UserEducation *(user_ID, school_ID, school_title, degree_title, start_date, end_date, grade, description*)

- UserBadges *(user_ID, badge_ID)*
- UserAuth *(user_ID, auth_email, pwd_hash)*
- UserLinks *(user_ID, url_ID, title, url)*
- PostVideos *(post_ID, video_id, video_url)*
- PostImages *(post_ID, image_ID, image_url)*
- PostTags *(post_ID, tag_ID, post_tag)*
- PostLinks *(post_ID, url_ID, post_url)*
- Comments *(post_ID, comment_ID, comment_date, comment_text*

14. HIGH-LEVEL DESIGN

Figure 6. High-Level Design

The ecosystem of the project includes three main components: User, Website, and Database Server. First, the users will be able to log in to the website hosted by Apache Web Server. During the login process, the website will communicate with the MySQL database server to authenticate the users. Next, they will be able to navigate through pages on the website. Those pages will display the information by communicating with the MySQL database server.

15.SCRIPTS(DDL)

a. Users Table

```
CREATE TABLE `Users`(
   `user_ID`   int(11) NOT NULL AUTO_INCREMENT,
   `user_type` varchar(50) NOT NULL,
   `firstname` varchar(50) NOT NULL,
   `lastname`  varchar(50) NOT NULL,
   `phonenumber`   varchar(50),
   `business_email`    varchar(50),
   `department`    varchar(50),
   `profile_photo_url` varchar(2048),
   PRIMARY KEY (`user_ID`)
)ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

b. UserAuth Table

```
CREATE TABLE IF NOT EXISTS `UserAuth` (
   `user_ID` int(11),
   `pwd_hash` varchar(128) NOT NULL,
   `auth_email` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

c. UserLanguages Table

```
CREATE TABLE `UserLanguages`(
   `user_ID` int(11),
   `language_ID` int(11),
   `proficiency` varchar(50),
   PRIMARY KEY (`user_ID`, `language_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

d. Languages Table

```
CREATE TABLE `Languages`(
   `language_ID` int(11) NOT NULL AUTO_INCREMENT,
   `language` varchar(50) NOT NULL,
   PRIMARY KEY (`language_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

e. UserDocuments Table

```
CREATE TABLE `UserDocuments`(
   `user_ID` int(11),
   `document_ID` int(11) AUTO_INCREMENT,
   `document_title` varchar(50),
   `document_url` varchar(2048),
   PRIMARY KEY (`user_ID`, `document_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

f. UserLicenses Table

```
CREATE TABLE `UserLicenses`(
   `user_ID` int(11),
   `license_ID` int(11) AUTO_INCREMENT,
   `license_title` varchar(50),
   `issuing_organization` varchar(50),
   `issue_date` DATE,
   `credential_url` varchar(2048),
   PRIMARY KEY (`user_ID`, `license_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

g. UserLinks Table

```
CREATE TABLE `UserLinks`(
   `user_ID` int(11),
   `url_ID` int(11) AUTO_INCREMENT,
   `title` varchar(50),
   `url` varchar(2048),
   PRIMARY KEY (`user_ID`, `link_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

h. UserSkills Table

```
CREATE TABLE `UserSkills`(
   `user_ID` int(11),
   `skill_ID` int(11) AUTO_INCREMENT,
   `skill_text` varchar(50),
   PRIMARY KEY (`user_ID`, `skill_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

i. UserBadges Table

```
CREATE TABLE `UserBadges`(
   `user_ID` int(11),
   `badge_ID` int(11)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

j. Badges Table

```
CREATE TABLE `Badges`(
   `badge_ID` int(11) NOT NULL AUTO_INCREMENT,
   `badge_label` varchar(50) NOT NULL,
   PRIMARY KEY (`badge_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

k. UserExperience Table

```
CREATE TABLE `UserExperience`(
   `user_ID` int(11),
   `experience_ID` int(11) AUTO_INCREMENT,
   `title` varchar(50),
   `start_date` DATE,
   `end_date` DATE,
   `location` varchar(100),
   PRIMARY KEY (`user_ID`, `experience_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

l. UserEducation Table

```
CREATE TABLE `UserEducation`(
   `user_ID` int(11),
   `school_id` int(11) AUTO_INCREMENT,
   `school_title` varchar(50),
   `degree_title` varchar(50),
   `start_date` DATE,
   `end_date` DATE,
   `grade` int,
   `description` varchar(500),
   PRIMARY KEY (`user_ID`, `education_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

m. Posts Table

```
CREATE TABLE `Posts`(
   `post_ID` int(11) NOT NULL AUTO_INCREMENT,
   `user_ID` int(11),
   `post_text` varchar(2048),
   `post_date` TIMESTAMP,
   PRIMARY KEY (`post_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

n. PostsLinks Table

```
CREATE TABLE `PostLinks`(
  `post_ID` int(11),
  `url_ID` int(11),
  `post_url` varchar(2048),
  PRIMARY KEY (`post_ID`, `link_ID`)
ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

15. PostTags Table

```
CREATE TABLE `PostTags`(
   `post_ID` int(11),
   `tag_ID` int(11),
   `post_tag` varchar(50),
  PRIMARY KEY (`post_ID`, `tag_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

16. PostVideos Table

```
CREATE TABLE `PostVideos`(
   `post_ID` int(11),
   `video_ID` int(11),
   `video_url` varchar(2048),
  PRIMARY KEY (`post_ID`, `video_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

17. PostImages Table

```
CREATE TABLE `PostImages`(
   `post_ID` int(11),
   `image_ID` int(11),
   `image_url` varchar(2048),
  PRIMARY KEY (`post_ID`, `image_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

18. Comments Table

```
CREATE TABLE `Comments`(
   `post_ID` int(11) NOT NULL AUTO_INCREMENT,
   `comment_ID` int(11),
   `comment_text` varchar(2048),
   `comment_date` TIMESTAMP,
  PRIMARY KEY (`post_ID`, `comment_ID`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

16.TABLE CONSTRAINT SCRIPTS(DDL)

```
ALTER TABLE UserAuth
   ADD CONSTRAINT user_auth_id
   FOREIGN KEY (user_ID)
   REFERENCES Users(user_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserLanguages
   ADD CONSTRAINT user_language_id
   FOREIGN KEY (user_ID)
   REFERENCES Users(user_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserLanguages
   ADD CONSTRAINT languages_language_id
   FOREIGN KEY (user_ID)
   REFERENCES Languages(language_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserDocuments
   ADD CONSTRAINT user_document_id
   FOREIGN KEY (user_ID)
   REFERENCES Users(user_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserLicenses
   ADD CONSTRAINT user_license_id
   FOREIGN KEY (user_ID)
   REFERENCES Users(user_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserLinks
   ADD CONSTRAINT user_link_id
   FOREIGN KEY (user_ID)
   REFERENCES Users(user_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserSkills
   ADD CONSTRAINT user_document_id
   FOREIGN KEY (user_ID)
   REFERENCES Users(user_ID)
   ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserBadges
```

```
  ADD CONSTRAINT user_badge_id
  FOREIGN KEY (user_ID)
  REFERENCES Users(user_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserBadge
  ADD CONSTRAINT badges_badge_id
  FOREIGN KEY (badge_ID)
  REFERENCES Badges(badge_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE UserExperience
  ADD CONSTRAINT user_experience_id
  FOREIGN KEY (user_ID)
  REFERENCES Users(user_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;


ALTER TABLE UserEducation
  ADD CONSTRAINT user_education_id
  FOREIGN KEY (user_ID)
  REFERENCES Users(user_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE Posts
  ADD CONSTRAINT user_post_id
  FOREIGN KEY (user_ID)
  REFERENCES Users(user_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE PostLinks
  ADD CONSTRAINT post_link_id
  FOREIGN KEY (post_ID)
  REFERENCES Posts(post_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE PostTags
  ADD CONSTRAINT post_tag_id
  FOREIGN KEY (post_ID)
  REFERENCES Posts(post_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;


ALTER TABLE PostVideos
  ADD CONSTRAINT post_video_id
```

```
  FOREIGN KEY (post_ID)
  REFERENCES Posts(post_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE PostImages
  ADD CONSTRAINT post_img_id
  FOREIGN KEY (post_ID)
  REFERENCES Posts(post_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;

ALTER TABLE Comments
  ADD CONSTRAINT post_comment_id
  FOREIGN KEY (post_ID)
  REFERENCES Posts(post_ID)
  ON DELETE RESTRICT ON UPDATE RESTRICT;
```

17.SCRIPTS(DML)

```
# USER SKILL
INSERT INTO `userskills` (`user_ID`, `skill_ID`, `skill_text`) VALUES ('1', NULL, 'c#');

# USER EDUCATION
INSERT INTO `usereducation` (`user_ID`, `school_ID`, `school_title`, `degree_title`, `start_date`, `end_date`,
`grade`, `description`) VALUES ('3', NULL, 'Abdullah Gül University', 'Computer Engineering', '2019-06-01',
'2022-06-01', '2', 'Interests: Web Development');
UPDATE `usereducation` SET  `school_title` =  'Abdullah Gül University', `degree_title` =  'Computer
Engineering', `start_date` =  '2019-08-01', `end_date` =  '2024-08-01', `grade` =  '2', `description` =  'Interests:
Blockchain' WHERE  `user_ID` = '1' AND `school_ID` = '$1';

# USER EXPERIENCE
INSERT INTO `userexperience` (`user_ID`, `experience_ID`, `title`, `start_date`, `end_date`, `location`) VALUES
('2', NULL, 'Software Developer ', '2021-06-01', '2022-06-01', 'Silicon Valley, San Francisco');

# USER POST
# post_ID auto-increment
INSERT INTO `posts` (`post_ID`, `user_ID`, `post_text`, `post_date`) VALUES (NULL, '1', '10',
'CURRENT_TIMESTAMP');
# Edit post
UPDATE `posts` SET `post_text` = 'Hello, you can find the internship opportunities of Google here' WHERE
`user_ID` = '1' AND `post_ID` = '10';
# video_ID auto-increment
INSERT INTO `postvideos` (`post_ID`, `video_ID`, `video_url`) VALUES ('10', NULL,
'https://www.youtube.com/watch?v=dQw4w9WgXcQ');
```

```
UPDATE `postvideos` SET `video_url` = 'https://www.youtube.com/watch?v=dQw4w9WgXcQ' WHERE
`user_ID` = '1' AND `video_ID` = '2';
# image_ID auto-increment
INSERT INTO `postimages` (`post_ID`, `image_ID`, `image_url`) VALUES ('10', NULL,
'https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png');
UPDATE `postimages` SET `post_url` =
'https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png' WHERE `user_ID` = '1'
AND `image_ID` = '1';
# tag_ID auto-increment
INSERT INTO `posttags` (`post_ID`, `tag_ID`, `post_tag`) VALUES ('10', NULL, 'internship');
UPDATE `posttags` SET `post_tag` = 'urgent' WHERE `user_ID` = '1' AND `tag_ID` = '1';

# USER COMMENT
# comment_ID auto-increment
INSERT INTO `comments` (`post_ID`, `comment_ID`, `comment_text`, `comment_date`) VALUES ('10', NULL,
'Thats amazing', 'CURRENT_TIMESTAMP');

UPDATE `comments` SET `comment_text` = 'Thanks Emre for the opportunity' WHERE `post_ID` = '10' AND
`post_ID` = '1';
# USER LINK
INSERT INTO `userlinks` (`user_ID`, `url_ID`, `title`, `url`) VALUES ('1', NULL, 'Github',
'https://github.com/Quelich');

UPDATE `userlinks` SET `title` = 'linkedin', `url` = 'www.linkedin.com' WHERE `user_ID` = '1' AND `url_ID` =
'1';

# USER LICENSE
INSERT INTO `userlicenses` (`user_ID`, `license_ID`, `license_title`, `issuing_organization`, `issue_date`,
`credential_url`) VALUES ('1', NULL, 'Google IT Automation with Python', 'Coursera', '2021-07-31',
'https://www.coursera.org/account/accomplishments/specialization/certificate/5APCKUHPKMXX');

UPDATE `userlicenses` SET `license_title` = 'Google IT Automation with Python', `issuing_organization` =
'Google', `issue_date` = '2022-05-06', `credential_url` =
'https://www.coursera.org/account/accomplishments/specialization/certificate/5APCKUHPKMXX}' WHERE
`user_ID` = '1' AND `license_ID` = '2';
```

18. SCRIPTS (JavaScript samples)

a. Initializing the database instance

```
const conn = mysql.createConnection({
```

```
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'pn_platform'
});
```

b. Authentication

```
function Auth(email, password) {
  return new Promise((resolve) => {
    conn.query(`SELECT * FROM userauth WHERE email = '${email}' AND pwd_hash = '${password}'`,
function (err, result) {
      if (err)
        resolve(false)

      if (result) {
        resolve(true)
      } else {
        resolve(false)
      }
    })
  })
}
```

c. Updating user data on the Profile page

```
 if (Auth(email, pwd_hash))
    conn.query(`UPDATE users SET firstname = '${firstname}', lastname = '${lastname}', phone_number =
'${phone_number}', business_email = '${business_email}', department = '${department}', profile_photo_url =
'${profile_photo_url}' WHERE user_ID = ${user_ID}`, (err, result) => {
      if (err)
        res.send({
          success: false,
          message: err
        })
      if (result) {
        res.send({
          success: true,
          message: 'User data updated successfully'
        })
      }
    })
```

d. Sharing Posts

```
if (Auth(email, pwd_hash)) {
   conn.query(`INSERT INTO posts (user_ID, post_text) VALUES ('${user_ID}', '${post_text}')`, (err, result)
=> {
     if (err) {
       return res.send({
         success: false,
         message: err,
       });
     }
     if (result) {
       const post_ID = result.insertId;
       conn.query(`INSERT INTO postimages (post_ID, image_url) VALUES ('${post_ID}', '${image_url}')`);
       conn.query(`INSERT INTO postvideos (post_ID, video_url) VALUES ('${post_ID}', '${video_url}')`);
       conn.query(`INSERT INTO posttags (post_ID, post_tag) VALUES ('${post_ID}', '${post_tag}')`);
     }
   })
  }
```

19. VIEWS

On the login page only club members with @agu.edu.tr mails will be able to login. Therefore, the admins of the website will arrange the register by creating users on the database, which means that the users will only login nor register. Users will only arrange their passwords. Also, we kept SHA512 hash codes of the passwords on the database for security reasons.

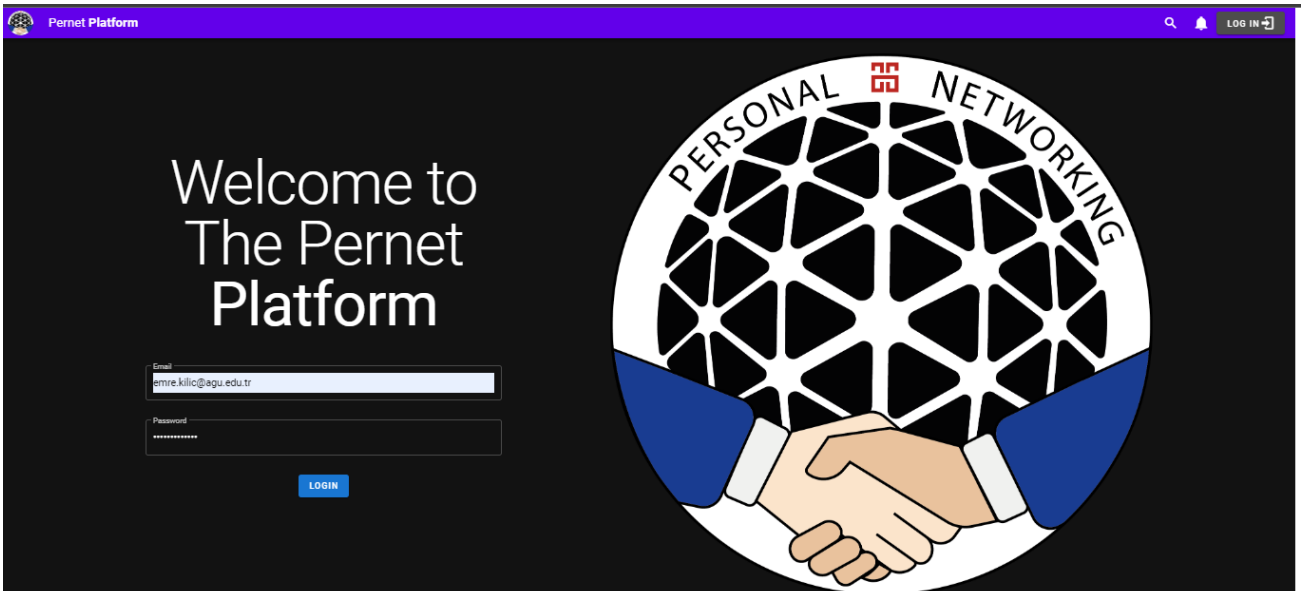

Figure 7. Login Page

On the home hope users are able to share posts with image links and tags. Also all the posts are fetched from the database after a post sharing so you need to refresh after that.

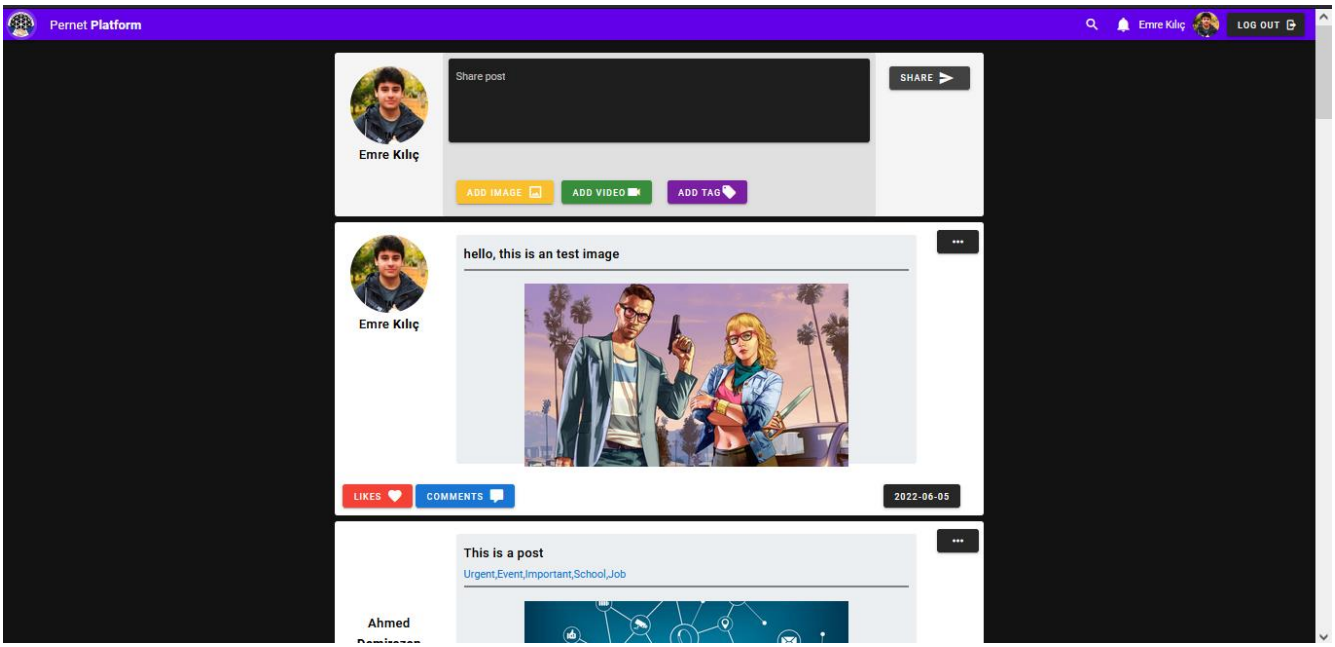

Figure 8. Home Page

On the profile page, users can edit their user details dynamically. Also the badges given by the admins is shown below the user profile photo.

Figure 9. Portfolio Page

That is, we use cache to dynamically change the text on the user interface and after pushing the save button.

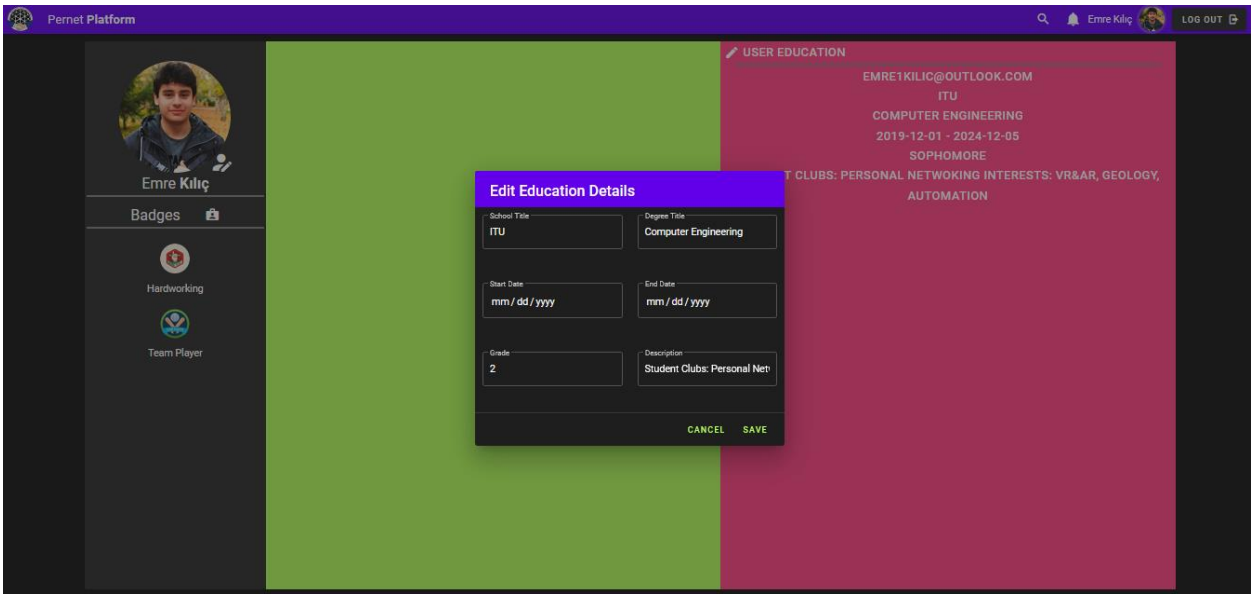

Figure 10 Edit user details such as user education