|  |  |
| --- | --- |
|  | **DOKUZ EYLÜL UNIVERSITY**  **ENGINEERING FACULTY**  **DEPT. OF COMPUTER ENGINEERING** |

BMO Alumni Member System

# CME 3201 Database Management Systems

# Term Project Report

Phase Final

2019-2020 FALL

2016510015 Berkay Coşkuner

2017510069 İlker Soner

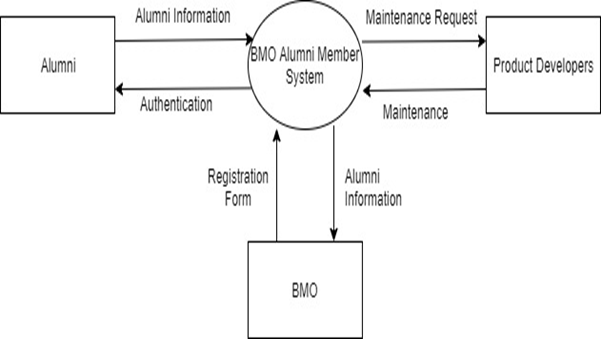
## Introduction

This project includes the graduate member system module of the chamber of computer engineers. The aim of the project is to enable the graduates of computer engineering to register in the system. The project was proposed as a solution to the loss of time due to the slowness caused by the previous system. The target audience of the project is those who have graduated from the department of computer engineering and want to register to the chamber of computer engineers. The name of the project is BMO Alumni Member System.

This report provides detailed information about the progress of the project and its future plans. Information about screen displays, diagrams and detailed system design related to the project will be shared later in the report.

.

## Overview



As seen above, this system determines how the graduates of the computer engineering department can participate in a structure in which 3 partners interact.

Information traffic between partners is as shown above.

Stakeholders:

1) Alumni

People who have graduated from computer engineering and want to join BMO.

2) Product Developers

The team that solves the problems in the existing system and adds new features to the application.

3) BMO

Legally authorized engineers to execute their profession within the borders of Turkey, engineer, architect, architects collects within the Turkish Engineers and Architects Chambers located and has legal personality of the Union is one of 24 rooms.

## Assumptions/Constraints/Risks

### Assumptions

Our dependencies are based on the standards required by BMO. Integrated development environments are as follows; as Framework .NET, as database PostgreSQL. Reports and diagrams are written in draw.io and codes are written in C # language in Visual Studio.

The project is installed on the Microsoft Windows operating system.

### Constraints

1) BMO Standarts

2) Deadline

3) Programming Language (C#)

### Risks

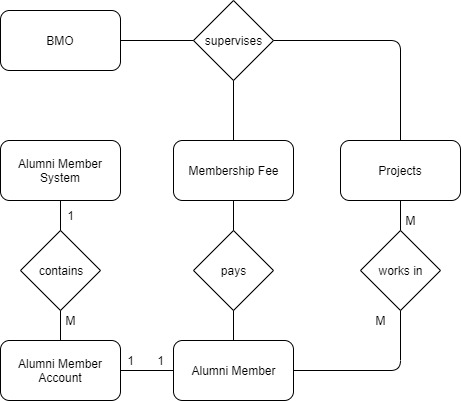
There is no predicted risk at the moment.

## Software Architecture

In the project we provided database and project connection using nugget packages and necessary commands. For example, we had to make arrangements in the startup.cs and database context classes. In addition we had to use console commands to keep the software up-to-date with our database. We used asp.net MVC architecture in our software design. We handled everything in our website using models, views and controllers. When connecting PostgreSQL with Asp.net it automatically created our models and made our job easier. We created controllers for necessary tables in our database, and using layout html page we added the functions we want in our website. We managed to build our login page in the account section of our MVC. In account’s model view and controller we had to add some html, c# codes to keep the login build altogether and to make it work effectively.

## Detailed System Design

### Entity-Relationship Diagram



### Relational Algebra Expressions

1-**Get Alumni Names**

Πname (Alumni)

SELECT name FROM Alumni

2-**Finding Names of Alumni Who Lives in İzmir**

πname(σcity=İzmir(Address) ⨝ Alumni)

SELECT name FROM Alumni

WHERE account\_id IN(SELECT account\_id FROM Address WHERE city=”İzmir”);

3-**Find the Names of All Alumni Who Have an Account And Paid Their Membership Fees**

πname(alumni) - πname(σfee=FALSE(Account) ⨝ Alumni)

SELECT name FROM Alumni

WHERE account\_id IN(SELECT account\_id FROM Account WHERE fee=TRUE);

4-**Select Rows That Satisfy Selection Condition**

σage<25(Alumni)

SELECT \* FROM Alumni

WHERE age<25

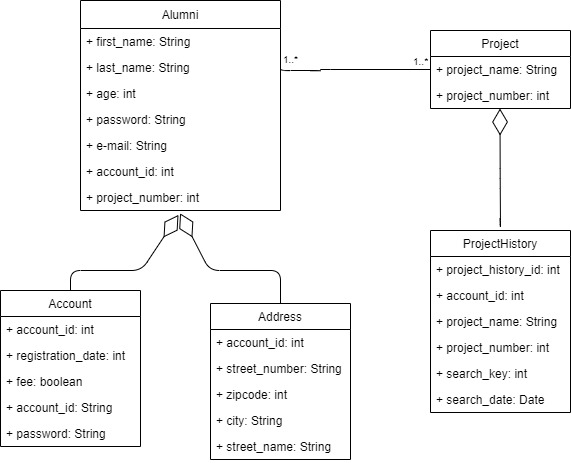
5-**Find the Surnames of Alumni Whose Account Id is Smaller Than 100**

Πsurname(σaccount\_id<100(Alumni))

SELECT surname FROM Alumni

WHERE account\_id<100

### Class Diagram



### CRUD Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Relations\Operations** | **Create** | **Read** | **Update** | **Delete** |
| Create\_Account | X | X |  |  |
| Delete\_Account |  | X | X | x |
| Edit\_Account |  | X | X |  |
| Account\_Details |  | X |  |  |
| Login |  | X |  |  |

**Appendix A: SQL Statements**

**CREATE TABLE public.account (**

**account\_id integer NOT NULL,**

**registration\_date integer NOT NULL,**

**fee boolean NOT NULL,**

**email text NOT NULL,**

**password text NOT NULL,**

**CONSTRAINT account\_email\_check CHECK ((email ~\* '^\w+@\w+[.]\w+$'::text)),**

**CONSTRAINT account\_password\_check CHECK ((char\_length(password) >= 8))**

**);**

**CREATE TABLE public.alumni (**

**account\_id integer NOT NULL,**

**first\_name text NOT NULL,**

**last\_name text NOT NULL,**

**age integer NOT NULL,**

**email text NOT NULL,**

**password text NOT NULL,**

**project\_number integer NOT NULL,**

**CONSTRAINT alumni\_check CHECK (((first\_name !~ '\s'::text) AND (last\_name !~ '\s'::text))),**

**CONSTRAINT alumni\_email\_check CHECK ((email ~\* '^\w+@\w+[.]\w+$'::text)),**

**CONSTRAINT alumni\_password\_check CHECK ((char\_length(password) >= 8))**

**);**

**CREATE TABLE public.address (**

**account\_id integer NOT NULL,**

**street\_name text NOT NULL,**

**street\_number text NOT NULL,**

**city text NOT NULL,**

**zipcode integer**

**);**

**CREATE TABLE public.alumni\_audits (**

**account\_id integer NOT NULL,**

**first\_name text NOT NULL,**

**last\_name text NOT NULL,**

**age integer NOT NULL,**

**email text NOT NULL,**

**password text NOT NULL,**

**project\_number integer NOT NULL,**

**changed\_on timestamp(6) without time zone NOT NULL**

**);**

**CREATE TABLE public.logs (**

**id integer,**

**table\_name character varying(255),**

**old\_value character varying(255),**

**new\_value character varying(255),**

**changed\_on timestamp(6) without time zone**

**);**

**CREATE TABLE public.project (**

**project\_number integer NOT NULL,**

**project\_name text NOT NULL**

**);**

**CREATE TABLE public.project\_history (**

**project\_history\_id bigint NOT NULL,**

**account\_id integer NOT NULL,**

**project\_number integer NOT NULL,**

**project\_name text NOT NULL,**

**search\_key text NOT NULL,**

**search\_date date NOT NULL**

**);**

**CREATE VIEW public.accountsofalumni AS**

**SELECT alumni.first\_name,**

**alumni.last\_name,**

**account.email,**

**account.fee**

**FROM (public.alumni**

**JOIN public.account ON ((account.account\_id = alumni.account\_id)));**

**CREATE FUNCTION public.log\_account\_id\_changes\_project\_history() RETURNS trigger**

**LANGUAGE plpgsql**

**AS $$**

**BEGIN**

**IF NEW.account\_id <> OLD.account\_id THEN**

**INSERT INTO logs(id,table\_name,old\_value,new\_value,changed\_on) VALUES(project\_number,'project\_history\_table',old.account\_id,new.account\_id,now());**

**END IF;**

**RETURN NEW;**

**END;**

**$$;**

**CREATE FUNCTION public.log\_age\_changes\_alumni() RETURNS trigger**

**LANGUAGE plpgsql**

**AS $$**

**BEGIN**

**IF NEW.age <> OLD.age THEN**

**INSERT INTO logs(id,table\_name,old\_value,new\_value,changed\_on)**

**VALUES(account\_id,'alumni\_table',old.age,new.age,now());**

**END IF;**

**RETURN NEW;**

**END;**

**$$;**

**CREATE FUNCTION public.log\_city\_changes\_address() RETURNS trigger**

**LANGUAGE plpgsql**

**AS $$**

**BEGIN**

**IF NEW.city <> OLD.city THEN**

**INSERT INTO logs(id,table\_name,old\_value,new\_value,changed\_on)**

**VALUES(account\_id,'address\_table',old.city,new.city,now());**

**END IF;**

**RETURN NEW;**

**END;**

**$$;**

**CREATE TRIGGER account\_id\_changes BEFORE UPDATE ON public.project\_history FOR EACH ROW EXECUTE FUNCTION public.log\_account\_id\_changes\_project\_history();**

**CREATE TRIGGER age\_changes BEFORE UPDATE ON public.alumni FOR EACH ROW EXECUTE FUNCTION public.log\_age\_changes\_alumni();**

**Appendix B: Screenshots**

