Bilkent University

CS353
DATABASE SYSTEMS

<u>Project Design Report</u>
Company Interview and Employment Review Platform Database System

<u>GROUP 5</u>

Mehmet Sanisoğlu

Mehmet Selim Özcan

Ayça Begüm Taşçıoğlu

Erdem Ege Maraşlı

1. Description of MESA

MESA is an online review company and interview review platform for the participants in various sectors to contribute and share their experiences. In the platform users can give reviews related to companies, jobs, and interview processes. In the platform, users can share information regarding their employment status, qualifications, resume, and their desired jobs. The reviews related to a company and its employment reviews can be seen by all the other users of the platform, and thus provide an environment of connectedness.

Companies can post about the currently available job offerings within their ranks with the pre-requirements they are looking for. The users can list the available jobs from the platform, filter the results according to their desires and sort them in the way they are interested. If they come across a job offering that they are suitable for they can send an application for the position. The companies, by again using the platform, can approve or decline the incoming application.

Companies can develop projects, by providing a set of required descriptive information about the project. ----

2. Separation of Labour

- Mehmet Sanisoğlu

  -Proposal,design and final report.

  -Worked on the transition between screens.

  -Worked on the user mode identification and visual improvements

- Mehmet Selim Özcan

  -Proposal,design and final report.

  -Worked on several pages design and implementation

  -Worked on database design and implementation.

- Ayça Begüm Taşçıoğlu

  -Proposal,design and final report.

  -Worked on reviews and admin pages.

  -Worked on appearance of GUI.

- Erdem Ege Maraşlı

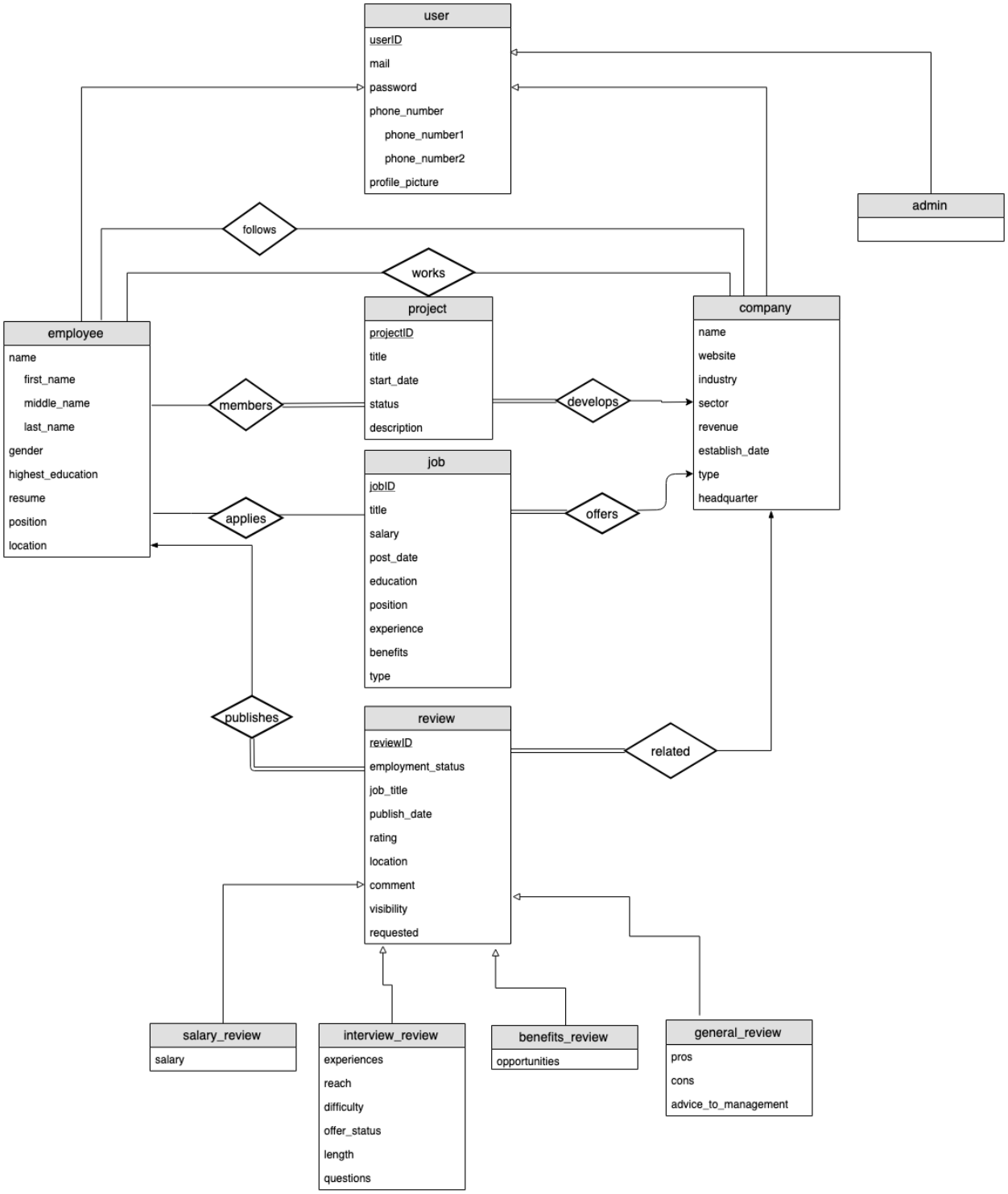
  -Proposal,design and final report.

  -Worked on database design and implementation.

  -Worked on list structures such as project list, job list etc.

  -Worked on several pages design and implementation.

2.Final E/R Diagram



3. Final List of Table Schemas

4.1 User

Relational Model:

user(<u>userID</u>, mail, password, phone_number1, phone_number2, profile_picture)

Functional Dependencies:

userID -> mail, password, phone_number1, phone_number2, profile_picture

mail -> password, phone_number1, phone_number2, profile_picture

Candidate Keys:

{ (userID), (mail) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user(
        userID          varchar(20) PRIMARY KEY,
        mail            varchar(40) NOT NULL,
        password        varchar(20) NOT NULL,
        phone_number1       varchar(20),
        phone_number2       varchar(20),
        profile_picturevarchar(200) ) Engine=InnoDB;
```

4.2 Employee

Relational Model:

employee(employeeID, first_name, middle_name, last_name, gender, highest_education, resume, position, Location)

Functional Dependencies:

employeeID -> first_name, middle_name, last_name, gender, highest_education, resume, position, Location

Candidate Keys:

{(employeeID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE employee(
        employeeID          varchar(20) PRIMARY KEY,
        first_name          varchar(40) NOT NULL,
        middle_name         varchar(40),
        last_name           varchar(40) NOT NULL,
        gender              varchar(20),
        highest_education   varchar(40),
        resume          varchar(40) NOT NULL,
        position            varchar(40),
        Location            varchar(40),
        FOREIGN KEY(employeeID) REFERENCES user(userID) ON UPDATE CASCADE
ON DELETE CASCADE) Engine=InnoDB;
```

4.3 Company

Relational Model:

company(<u>companyID</u>, name, website, industry, sector, revenue, establish_date, type, headquarter)

Functional Dependencies:

companyID -> name, website, industry, sector, revenue, establish_date, type, headquarter

Candidate Keys:

{ (companyID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE company(
        companyID     varchar(20) PRIMARY KEY,
        name          varchar(20) NOT NULL,
        website       varchar(50),
        industry      varchar(10) NOT NULL,
        sector        varchar(10) NOT NULL,
        revenue       double,
        establish_data date NOT NULL,
        type          varchar(10) NOT NULL,
        headquarter   varchar(10) NOT NULL,
        FOREIGN KEY(companyID) REFERENCES user(userID)ON UPDATE CASCADE
ON DELETE CASCADE ) Engine=InnoDB;
```

4.4 Follows

Relational Model:

follows(<u>employeeID</u>, <u>companyID</u>)

Functional Dependencies:

None

Candidate Keys:

{ (employeeID, companyID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE follows(
        employeeID          varchar(20),
        companyID           varchar(20),
        PRIMARY KEY(employeeID, companyID),
        FOREIGN KEY(employeeID) REFERENCES employee(employeeID)ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY(companyID) REFERENCES company(companyID)ON UPDATE CASCADE ON DELETE CASCADE ) Engine=InnoDB;
```

4.5 Works

Relational Model:

works(<u>employeeID</u>, <u>companyID</u>)

Functional Dependencies:

None

Candidate Keys:

{ (employeeID, companyID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE works(
        employeeID          varchar(20),
        companyID           varchar(20),
        PRIMARY KEY(employeeID, companyID),
        FOREIGN KEY(employeeID) REFERENCES employee(employeeID)ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY(companyID) REFERENCES company(companyID)ON UPDATE CASCADE ON DELETE CASCADE ) Engine=InnoDB;
```

4.6 Job

Relational Model:

job(jobID, title, salary, post_date, education, position, experience, benefits, type)

Functional Dependencies:

(jobID) -> title,salary,post_date,education, position, experience, benefits, type

Candidate Keys:

{(jobID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE job(
        jobID           varchar(20),
        title           varchar(40),
        salary          double,
        post_date       date,
        education       varchar(40),
        position        varchar(20) NOT NULL,
        experience      varchar(40),
        benefits        varchar(40),
        type            varchar(40) NOT NULL,
        PRIMARY KEY(jobID)) Engine=InnoDB;
```

4.7 Project

Relational Model:

project(projectID ,title, start_date, status, description)

Functional Dependencies:

(projectID) -> title,start_date,status, description

Candidate Keys:

{(projectID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE project(
        projectID           varchar(20),
        title               varchar(20),
        start_date          date,
        status              varchar(40),
        description         varchar(40),
        PRIMARY KEY(projectID)) Engine=InnoDB;
```

4.8 Applies

Relational Model:

applies(employeeID, jobID)

Functional Dependencies:

None

Candidate Keys:

{ (employeeID,jobID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE applies(
        employeeID     varchar(20),
        jobID          varchar(20),
        PRIMARY KEY ( employeeID, jobID ),
        FOREIGN KEY ( employeeID) REFERENCES employee(employeeID) ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY ( jobID) REFERENCES job(jobID) ON UPDATE CASCADE ON DELETE CASCADE) Engine=InnoDB;
```

4.9 Review

Relational Model:

review(reviewID, employment_status, job_title, date, rating, location, comment, visibility)

Functional Dependencies:

reviewID -> employment_status, job_title, date, rating, location, comment, visibility

Candidate Keys:

{ (reviewID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE review(
        reviewID            int PRIMARY KEY,
        Employment_status   bit NOT NULL,
        job_title           varchar(40) NOT NULL,
        publish_date        date NOT NULL,
        rating              double NOT NULL,
        location            varchar(40) NOT NULL,
        comment             varchar(500) NOT NULL,
        visibility          bit NOT NULL )
        requested           bit NOT NULL) Engine=InnoDB;
```

4.10 Publishes

Relational Model:

publishes(reviewID, employeeID)

Functional Dependencies:

None

Candidate Keys:

{ (reviewID, employeeID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE publishes(
        reviewID                int,
        employeeID              varchar(20),
        PRIMARY KEY (reviewID),
        FOREIGN KEY (reviewID) references review(reviewID) ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY (employeeID) references employee(employeeID) ON UPDATE CASCADE ON DELETE CASCADE) Engine=InnoDB;
```

4.11 Related

Relational Model:

related(reviewID, companyID)

Functional Dependencies:

None

Candidate Keys:

{ (reviewID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE related(
        reviewID                int,
        companyID               varchar(20),
        PRIMARY KEY ( reviewID),
        FOREIGN KEY ( reviewID) references review(reviewID)ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY ( companyID) references company(companyID) ON UPDATE CASCADE ON DELETE CASCADE) Engine=InnoDB;
```

4.12 Admin

Relational Model:

admin(adminID)

Functional Dependencies:

None

Candidate Keys:

{ (adminID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE admin(
        adminID                         varchar(20) PRIMARY KEY,
        FOREIGN KEY(adminID) REFERENCES user(userID) ON UPDATE CASCADE ON DELETE CASCADE) Engine=InnoDB;
```

4.13 Salary_Review

Relational Model:

salary_review(reviewID, salary)

Functional Dependencies:

reviewID -> salary

Candidate Keys:

{ (reviewID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE salary_review(
        reviewID                int PRIMARY KEY,
        salary                  double NOT NULL,
        FOREIGN KEY(reviewID) REFERENCES review(reviewID) ON UPDATE CASCADE
ON DELETE CASCADE) Engine=InnoDB;
```

4.14 Benefits_Review

Relational Model:

benefits_review(reviewID, opportunities)

Functional Dependencies:

reviewID -> opportunities

Candidate Keys:

{ (reviewID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE benefits_review(
        reviewID                int PRIMARY KEY,
        opportunities           varchar(100) NOT NULL,
        FOREIGN KEY(reviewID) REFERENCES review(reviewID) ON UPDATE CASCADE
ON DELETE CASCADE) Engine=InnoDB;
```

4.15 General_Review

Relational Model:

general_review(reviewID, pros, cons, advice_to_management)

Functional Dependencies:

reviewID -> pros, cons, advice_to_management

Candidate Keys:

{ (reviewID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE general_review(
        reviewID                    int PRIMARY KEY,
        pros                        varchar(100) NOT NULL,
        cons                        varchar(100) NOT NULL,
        advice_to_management        varchar(200),
        FOREIGN KEY(reviewID) REFERENCES review(reviewID) ON UPDATE CASCADE
ON DELETE CASCADE) Engine=InnoDB;
```

4.16 Interview_Review

Relational Model:

interview_review(reviewID, experiences, reach, difficulty, offer_status, length, questions)

Functional Dependencies:

reviewID -> experiences, reach, difficulty, offer_status, length, questions

Candidate Keys:

{ (reviewID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE interview_review(
        reviewID                int PRIMARY KEY,
        experiences             varchar(200) NOT NULL,
        reach                   varchar(20) NOT NULL,
        difficulty              int NOT NULL,
        offer_status            bit NOT NULL,
        length                  int NOT NULL,
        Questions               varchar(200) NOT NULL,
        FOREIGN KEY(reviewID) REFERENCES review(reviewID) ON UPDATE CASCADE
ON DELETE CASCADE ) Engine=InnoDB;
```

4.17 Members

Relational Model:

members(employeeID, projectID)

Functional Dependencies:

None

Candidate Keys:

{(employeeID,projectID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE members (
        employeeID          varchar(20),
        projectID           varchar(10),
        PRIMARY KEY ( employeeID, projectID),
        FOREIGN KEY ( employeeID ) references employee(employeeID)ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY ( projectID ) references project(projectID)ON UPDATE CASCADE ON DELETE CASCADE ) Engine=InnoDB;
```

4.18 Develops

Relational Model:

develops(projectID, companyID)

Functional Dependencies:

None

Candidate Keys:

{ (projectID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE related(
        projectID           varchar(10),
        companyID           varchar(20),
        PRIMARY KEY ( projectID),
        FOREIGN KEY ( projectID) references project(projectID) ON UPDATE CASCADE ON DELETE CASCADE,
        FOREIGN KEY ( companyID) references company(companyID) ON UPDATE CASCADE ON DELETE CASCADE) Engine=InnoDB;
```

4.19 Offers

Relational Model:

offers(jobID, companyID)

Functional Dependencies:

None

Candidate Keys:

{ (jobID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE related(
        jobID                   varchar(10),
        companyID               varchar(20),
        PRIMARY KEY ( jobID),
        FOREIGN KEY ( jobID) references job(jobID),
        FOREIGN KEY ( companyID) references company(companyID) ) Engine=InnoDB;
```

4. Implementation Details

We have approached the project from three main branches: database; for storing the models and data, the frontend; for the view of the platform and the backend, for managing the interactions in between these two. We used the MariaDB server on the Dijkstra platform to build our database. In order to establish the connection between our local machines and the Dijkstra server we have used XAMPP. Our platform is based heavily on PHP and HTML, as the backbone of our backend is a combination of simple and complex HTML manipulations by the PHP scripts. The database is updated according to these manipulations and provides another a different view. We used a template from "Landed by HTML5up" as a starting point and changed it according to our needs in each step as we progressed.

- **Technical problems that we faced**

GitHub Integration

In order to increase our development speed and utilize each member's potential we have worked separately, but kept up with each other's progress via GitHub. The initial stages of integration and concurrent usage proved to be chaotic and inhibited our progress via faulty merges and connection failures.

Dijkstra Server

We needed to see the results of our backend manipulations, however the frontend of our system has not always been capable of visually displaying our progress. In this aspect, we have been using the Putty to directly access our database and seeing the contents of our tables. Direct access was essential to overcoming the obstacles on the way. However, the Putty system only allowed one terminal to work at a time, so all the other members of the group were forced out of the system, which caused a burden on our overall development speed.

5. Advanced Database Features

- **Views**

CREATE VIEW exceptAdmin **AS** (**SELECT * FROM** user **WHERE** userID > 9);

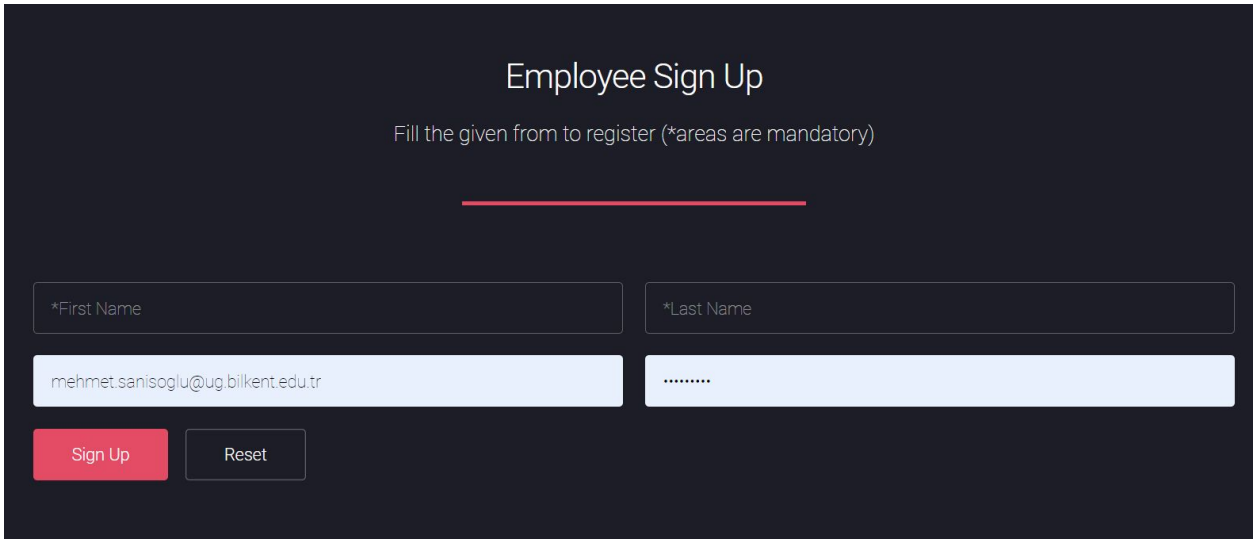
We use exceptAdmin view in admin_userList.page that admins can view user except themselves and remove them if they want. This view provides protection that admins cannot remove their accounts.

- **Triggers**

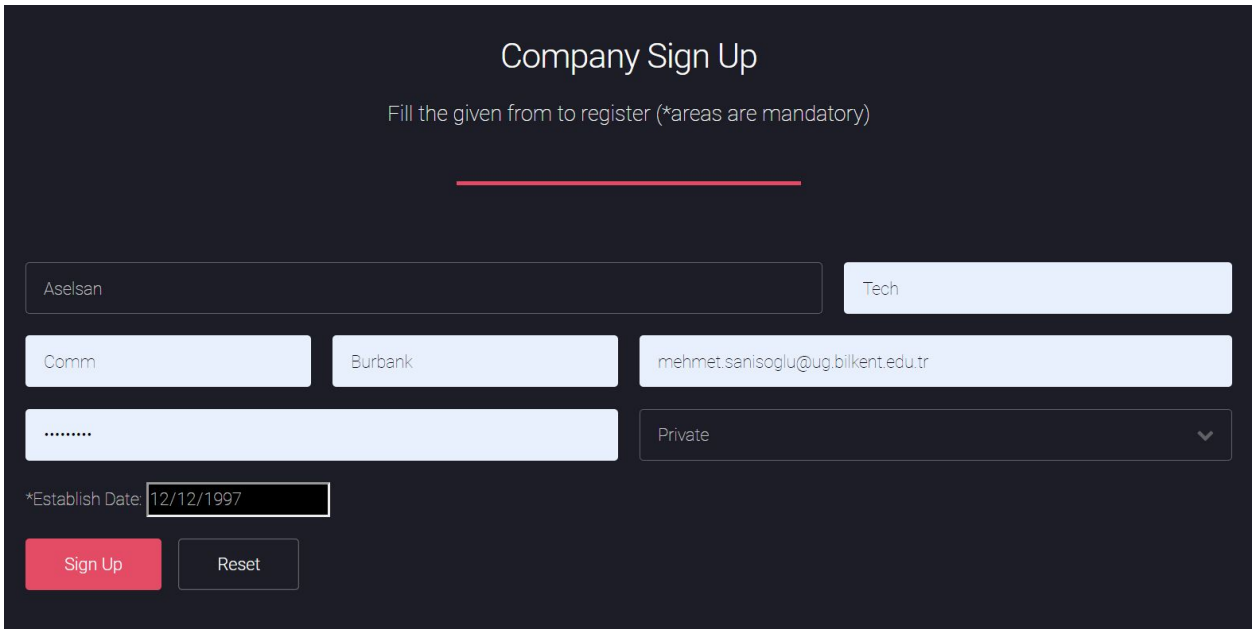
```
CREATE TRIGGER triger_review_after_delete
AFTER DELETE
    ON company
FOR EACH ROW
BEGIN
    DELETE FROM review WHERE reviewID in ( SELECT reviewID
    FROM related
    WHERE companyID = OLD.companyID)
END;
```

6. User Manual

7.1 Register Page

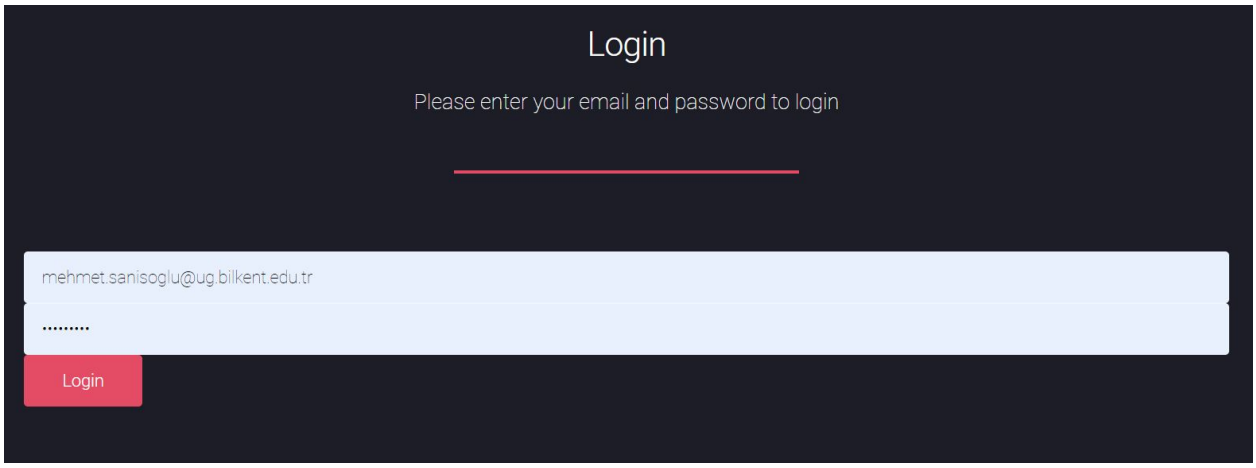


The register pages differ according to the user type. There are two possible user types: employee and company. The system allows the employees to register to the system by providing minimal information such as: name, surname, a valid email address and password. On the other hand a company register requires more detailed information about the company's specifications.



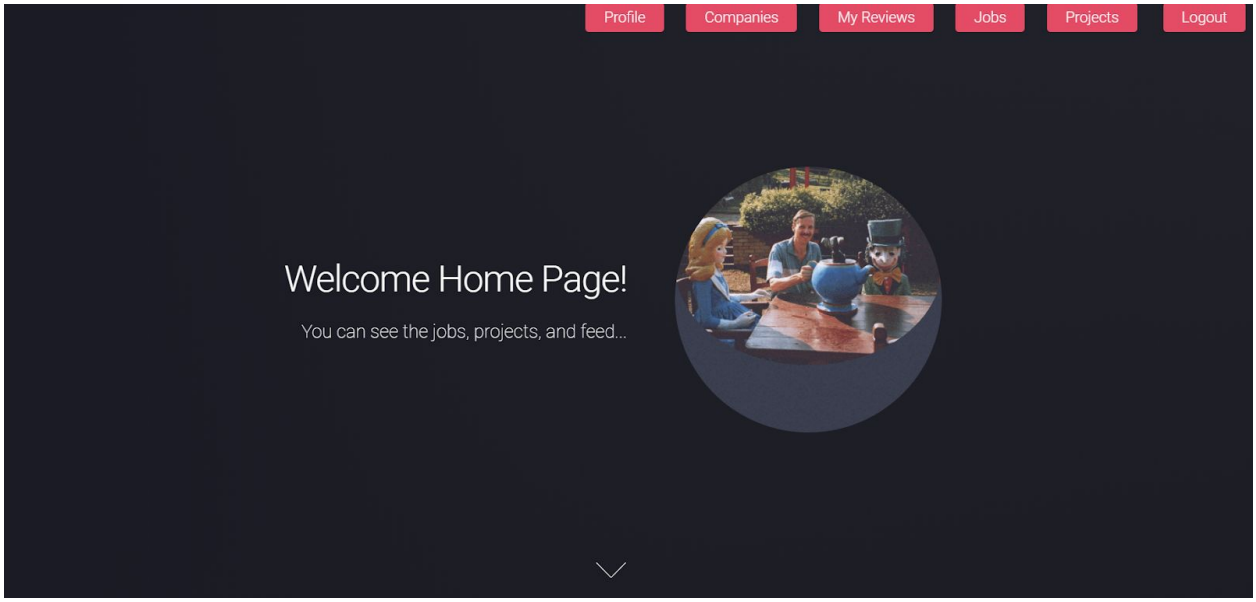
7.2 Login Page

After acquiring an account the user can enter their email and password to enter the system. The system checks and determines the user type from the provided email address, which is unique for every user.



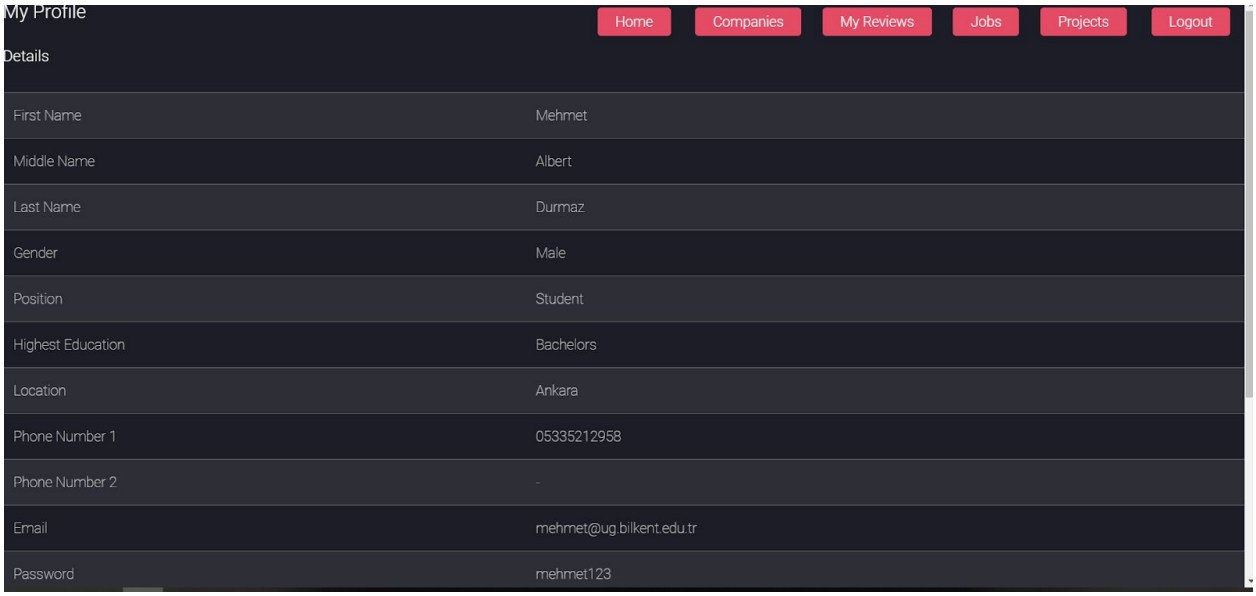
7.3 Home Page

The home page acts as an anchor point for the user to come back to and access the full navigation that the site offers. The home page itself includes news, and feeds for the user.



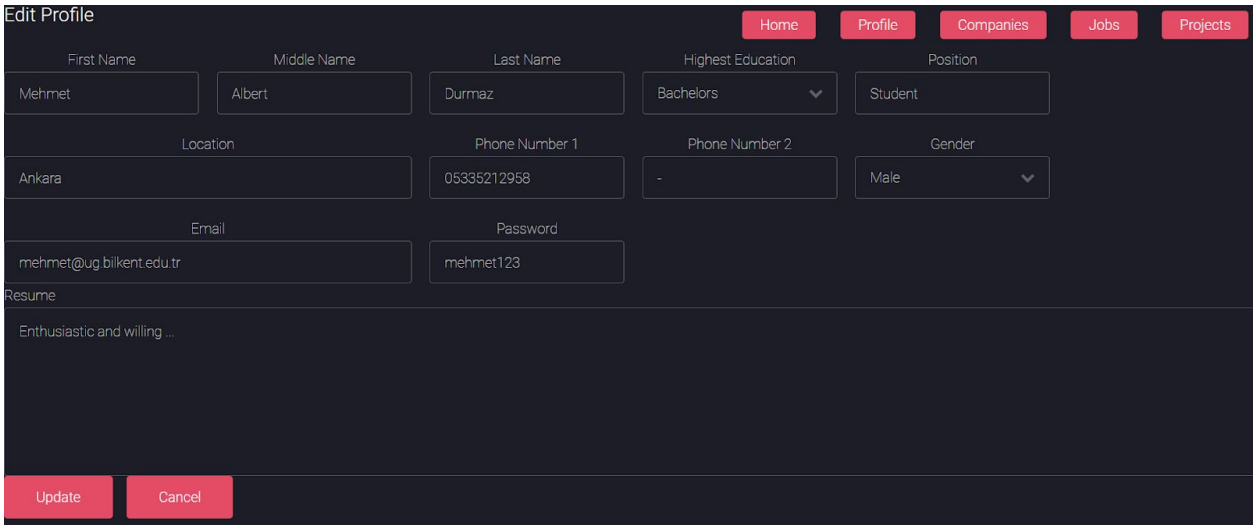
7.4 Profile Page

The profile page differs according to the user type. These screens display the related information about their profiles.

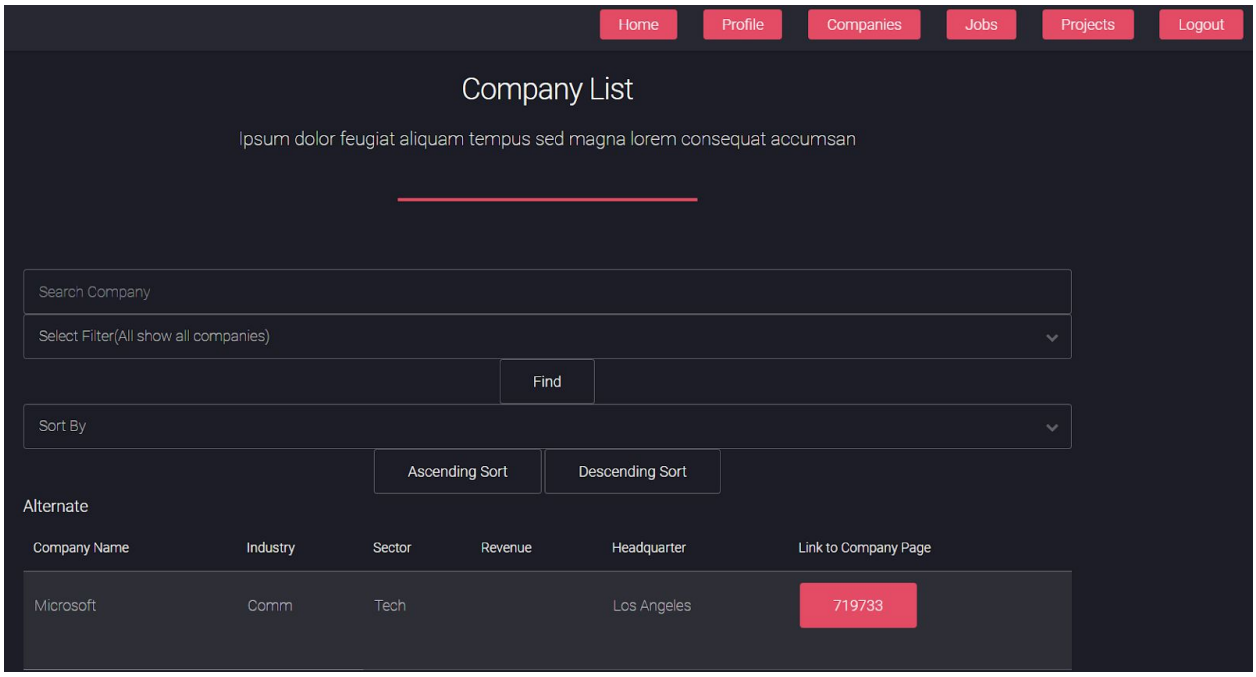


7.5 Edit Profile Page

Each user can enter or update the information regarding their accounts in this screen. After their editing is done, they are transferred back to their profile page.



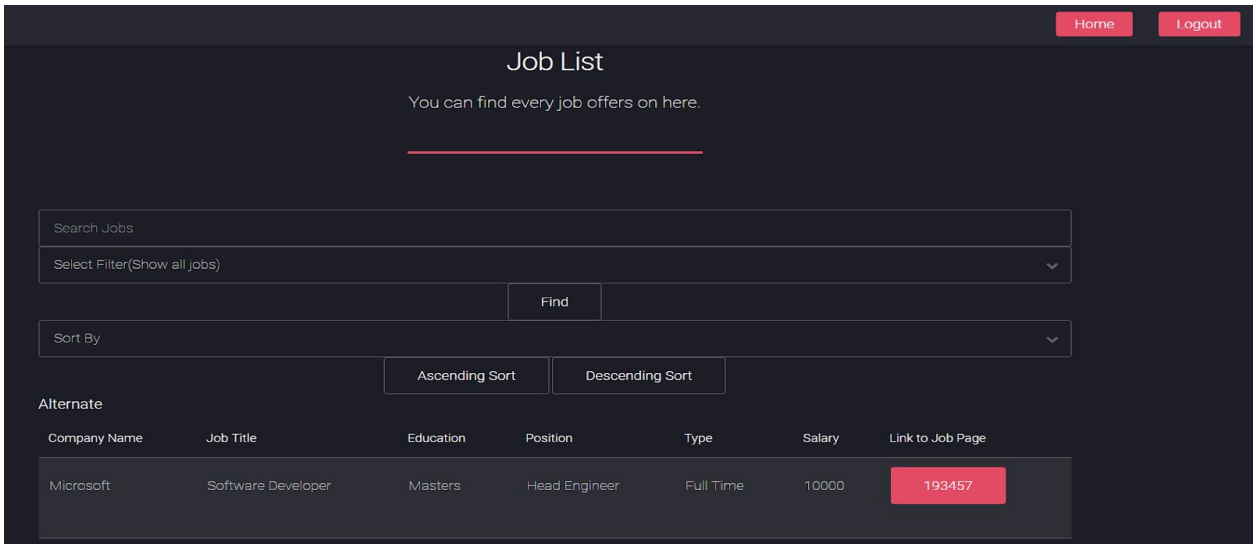
7.6 Company List Page



The user can get a list view of the available job offers and go to their related pages. In order to ease the search process, the user can use the filter and sort features to examine the list easily.

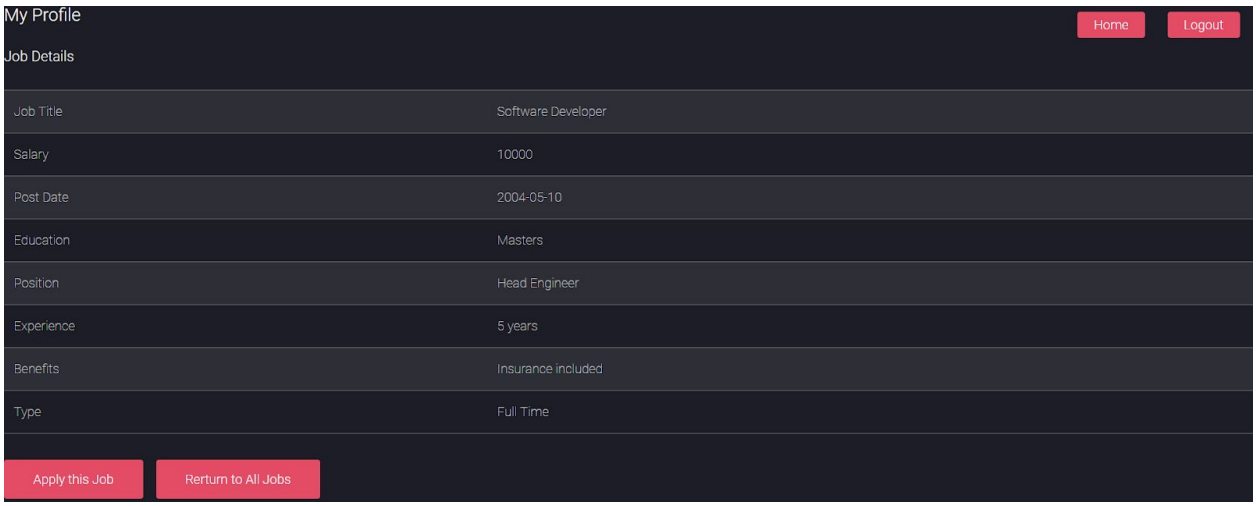
7.7 Job List Page

The job list displays all the available jobs offers from different companies to the user. The user in turn, can filter and sort the offers and select one of them to get more information. The user is directed to the selected job's page.



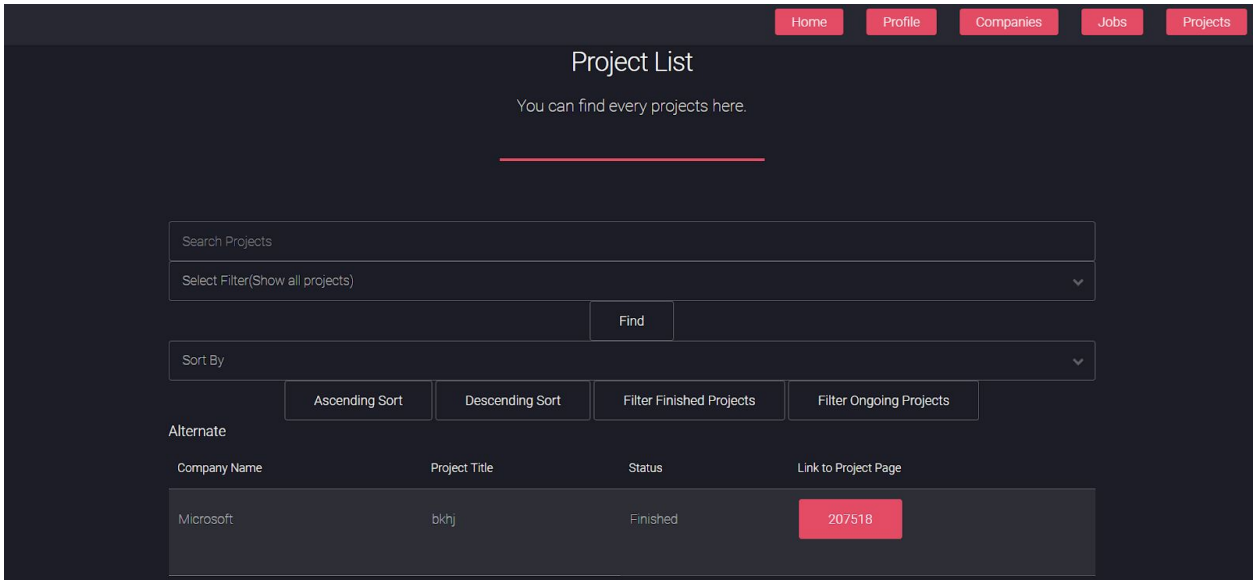
7.8 Job Page

Here the user can get more detailed information about the job they have selected.If they are content with the job specifications they can apply for the position, or they can choose to go back to jobs list.



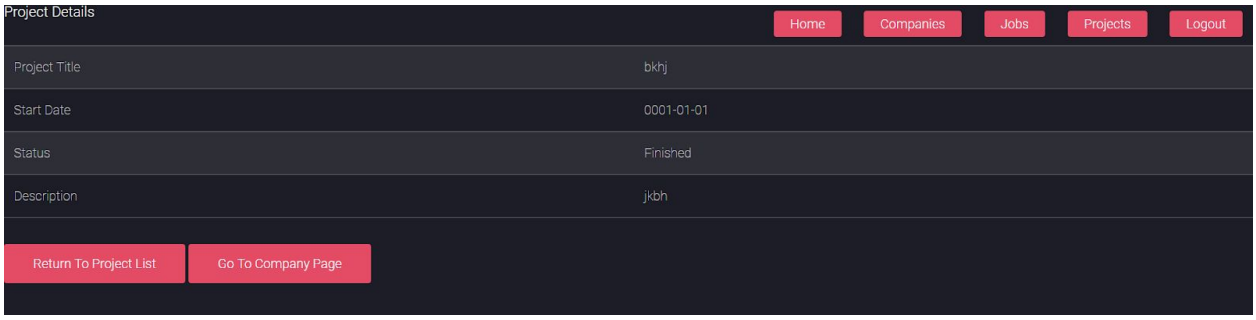
7.9 Project List Page

From the home page the user can access a list of projects that are developed by various companies and keep up with the ongoing trends in different sectors.



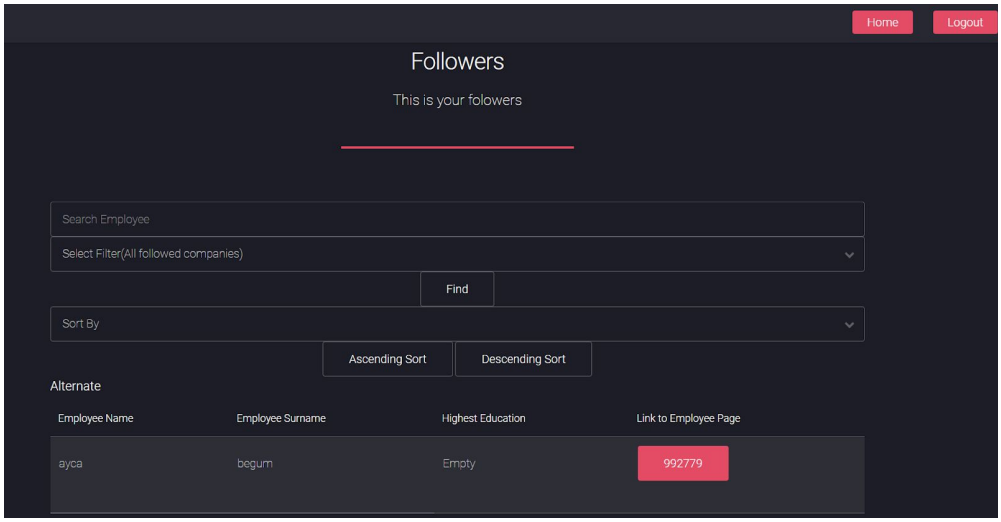
7.10 Project Page

The project page can inform the user about the details of the project they have selected.



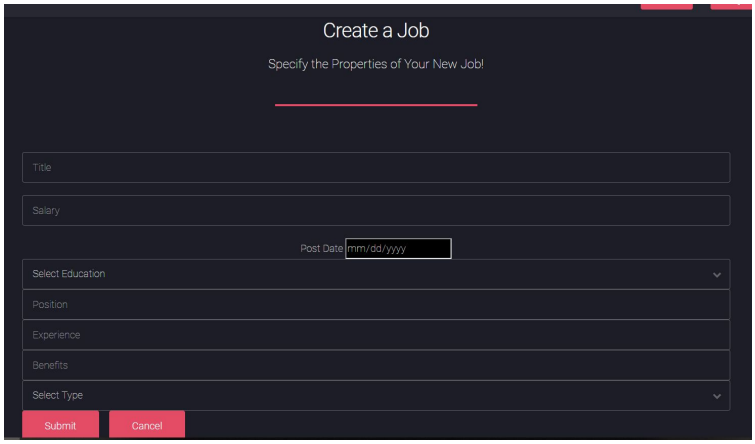
7.11 Followers Page

A company can see its followers in the Followers Page. Followers can be sorted or filtered by their first names, surnames and highest educations. Company can search among all followers by using search bar.



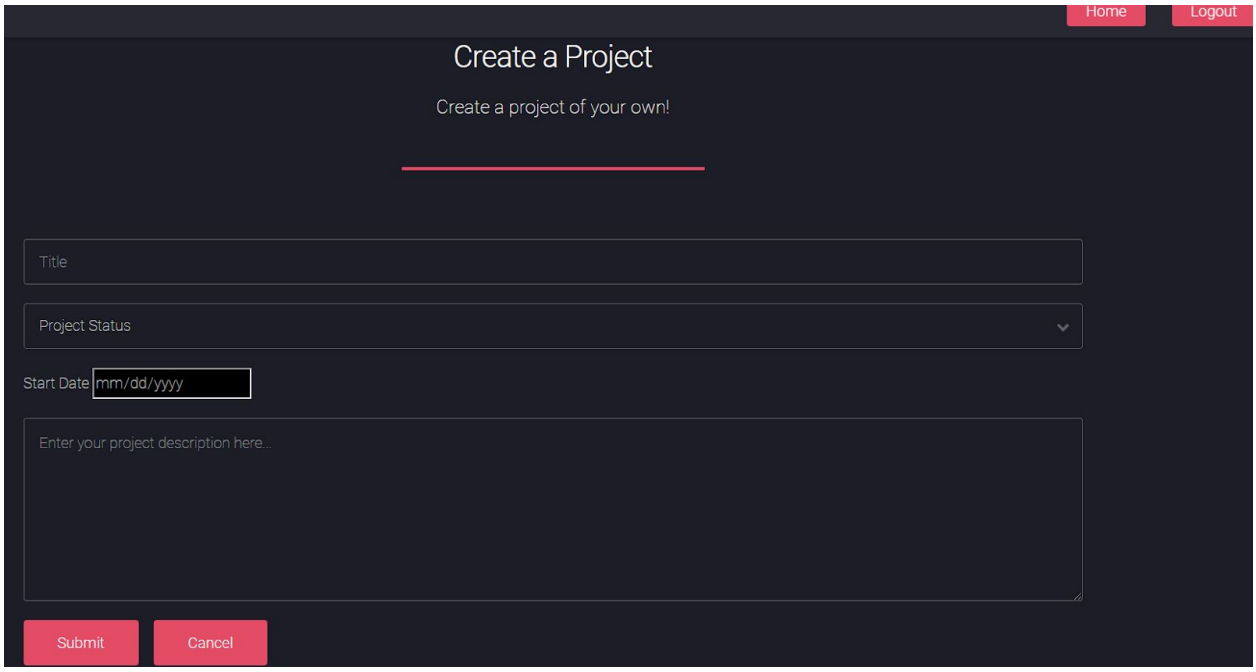
7.12 Create a Job Page

By specifying the related information, the company user can create a new job offer for other users to search and apply to.



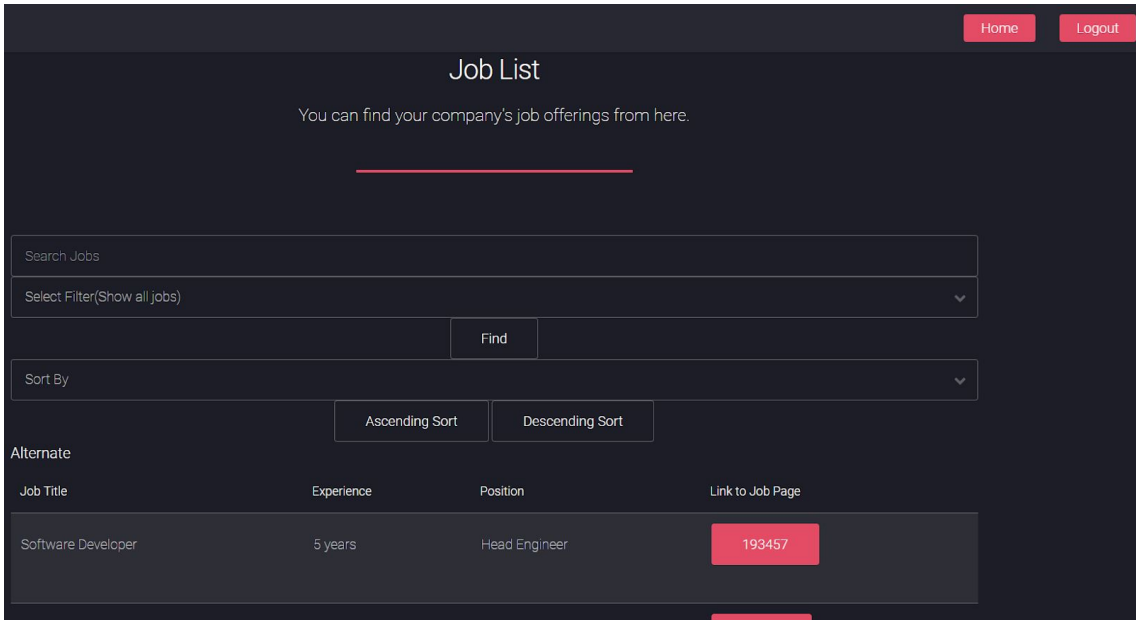
7.13 Create a Project Page

A company can create a project by specifying its title, project status, start date and project description.



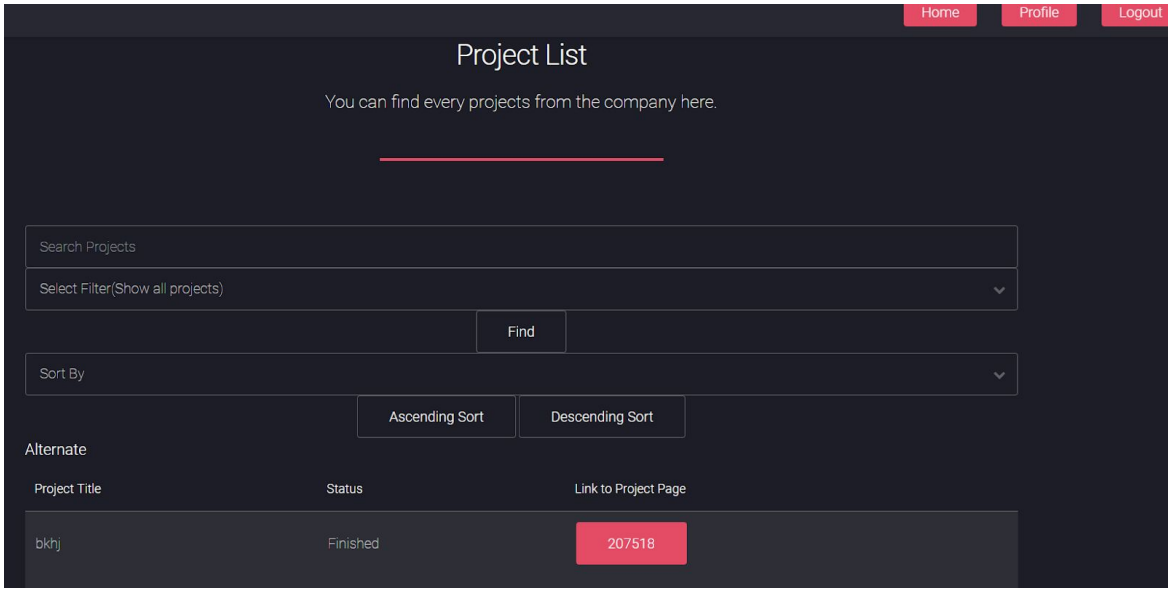
7.14 Job List Page

Here the company user can see all of the job offers that they have put up and go to their detailed information pages.



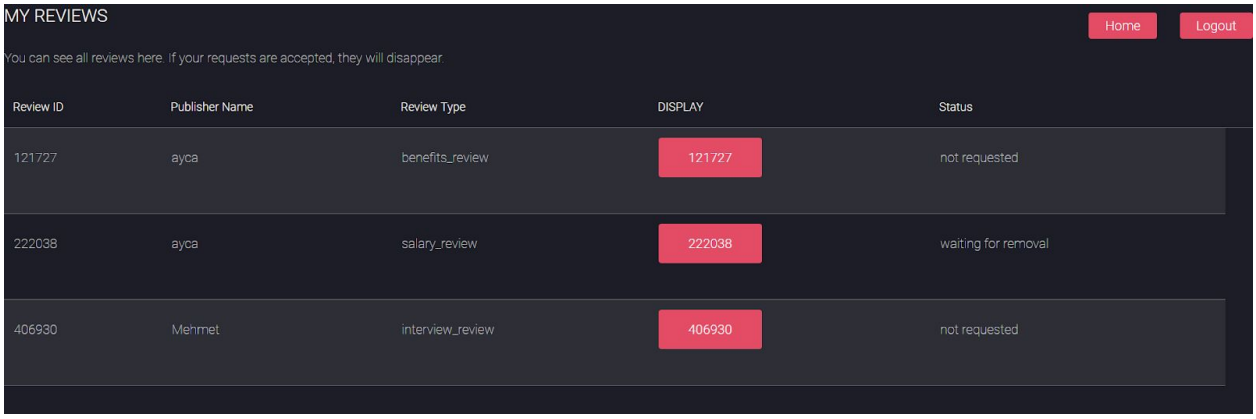
7.15 Project List

With the help of Project List view, the company can keep track of the projects they have been developing with a list view. They can select a project for more detailed information.



7.16 My Reviews Page

A company can see its reviews by their reviewIDs, Publisher Names, Review Types and Status. If a company want to request from admins to delete a review, the company should press DISPLAY button and redirect to the Display Review Page, and then request that review to be deleted. If the requested review is deleted, deleted review will be disappeared.



7.17 Add a Worker Page

The company can use this screen to add their companies' rooster.

Search Employee

Select Filter(All employees)

Find

Alternate

Employee Name	Employee Surname	Highest Education	Link to Employee Page
erdem	ege		190223
Mehmet	Durmaz	Bachelors	222646
ayca	begum	Empty	992779
ugur	gudukbay		995408

7.18 Show Workers

The company user can get a list view of the employees that work for them and go to their profile pages for additional information.

Workers

These are your workers

Search Employee

Select Filter(All workers)

Find

Sort By

Ascending Sort | Descending Sort

Alternate

Employee Name	Employee Surname	Highest Education	Link to Employee Page
ayca	begum	Empty	992779

8. Website

Our project directory with reports

Project directory:

https://github.com/aeyc/Company-Interview-and-Employment-Review-Platform-Database-System.git