

BILKENT UNIVERSITY

CS353 GROUP 1 - OH No

SUIT UP

---

# Project Design Report

---

*Group Members:*

Ergün Batuhan KAYNAK	21501178
Beyza KALKANLI	21600944
Kerem YILMAZ	21601223
Muhammed Salih ALTUN	21602314

April 5, 2019



**Bilkent University**

# Contents

<b>1</b>	<b>Final Entity/Relationship Diagram</b>	<b>3</b>
1.1	Revisions . . . . .	3
1.2	Revised Entity/Relationship Diagram . . . . .	4
<b>2</b>	<b>Relational Schemas</b>	<b>5</b>
2.1	Account . . . . .	5
2.2	Office_Account . . . . .	6
2.3	Editor . . . . .	7
2.4	Admin . . . . .	8
2.5	Super_Admin . . . . .	9
2.6	Evaluate . . . . .	10
2.7	Add_Editor . . . . .	11
2.8	Remove_Editor . . . . .	12
2.9	Add_Admin . . . . .	13
2.10	Remove_Admin . . . . .	14
2.11	Report . . . . .	15
2.12	Review . . . . .	16
2.13	Employment_Review . . . . .	18
2.14	Interview_Review . . . . .	19
2.15	Pros_Cons . . . . .	20
2.16	Review_Proc_Cons . . . . .	21
2.17	Office_Sub . . . . .	22
2.18	Job . . . . .	23
2.19	Employment . . . . .	24
2.20	Company . . . . .	25
2.21	Office . . . . .	26
2.22	Job_Offers . . . . .	27
2.23	User . . . . .	28
2.24	Event . . . . .	29
2.25	User_Apply . . . . .	30

2.26	User_Event . . . . .	31
2.27	Company_Sub . . . . .	32
2.28	Location_Sub . . . . .	33
<b>3</b>	<b>Normalization of Tables</b>	<b>34</b>
<b>4</b>	<b>Functional Components</b>	<b>35</b>
4.1	Use Case Diagram . . . . .	35
4.2	Scenarios . . . . .	36
4.3	Algorithms and Data Structures . . . . .	55
<b>5</b>	<b>User Interface Design and Corresponding SQL Statements</b>	<b>56</b>
<b>6</b>	<b>Advanced Database Components</b>	<b>87</b>
6.1	Views . . . . .	87
6.2	Reports . . . . .	88
6.3	Triggers . . . . .	90
6.4	Constraints . . . . .	91
6.5	Stored Procedure . . . . .	91
<b>7</b>	<b>Implementation Plan</b>	<b>92</b>

# 1 Final Entity/Relationship Diagram

## 1.1 Revisions

We have revised our E/R diagram according to the feedback that we received from our TA, Mustafa Çavdar. The following changes were made:

- Changed primary keys of all entities to *id*.
- Removed the wrong use of weak entities in *post*, *submit*, *offers*, *is\_type*, *is\_position*, *user\_address*, *editor\_address*, *office\_address*, *company\_office*, *represent*, and *is\_about* relations.
- Removed the *Address* entity and added address information to *User*, *Editor*, and *Office* entities as an attribute. String matching will be used for location based search.
- Divided the *add\_remove\_editor* and *add\_remove\_admin* relations into two to separate add and remove functionalities. Since each editor will be added by an admin or a super admin and each admin will be added by a super admin, they totally participate in *add\_editor* and *add\_admin* relations.
- Removed *avg\_rating* attribute from *Company* entity.
- Changed the cardinality constraints of *has* relation to total participation on both sides.
- Changed the relation of *Review* and *User* into the relation of *Review* with the ternary relationship of *User*, *Job*, and *Office* using aggregation since *Review* requires information from *Job* and *Office* also.
- Generalized the *Review* entity into *EmploymentReview* and *InterviewReview* entities.
- Added *interview* descriptive attribute to *employment* relation in order to avoid having two aggregate relations for *EmploymentReview* and *InterviewReview* entities.
- Added *creation\_date* attribute to *Account*, *Review*, *Report*, and *Event* entities in order to be able to sort by timestamp.
- Added *description* attribute to *User*, *Event*, and *Report* entities.
- Added *progress* attribute to *apply* relation representing different stages of a job application as rejected without interview, accepted for interview, hired, and rejected after interview.

## 1.2 Revised Entity/Relationship Diagram

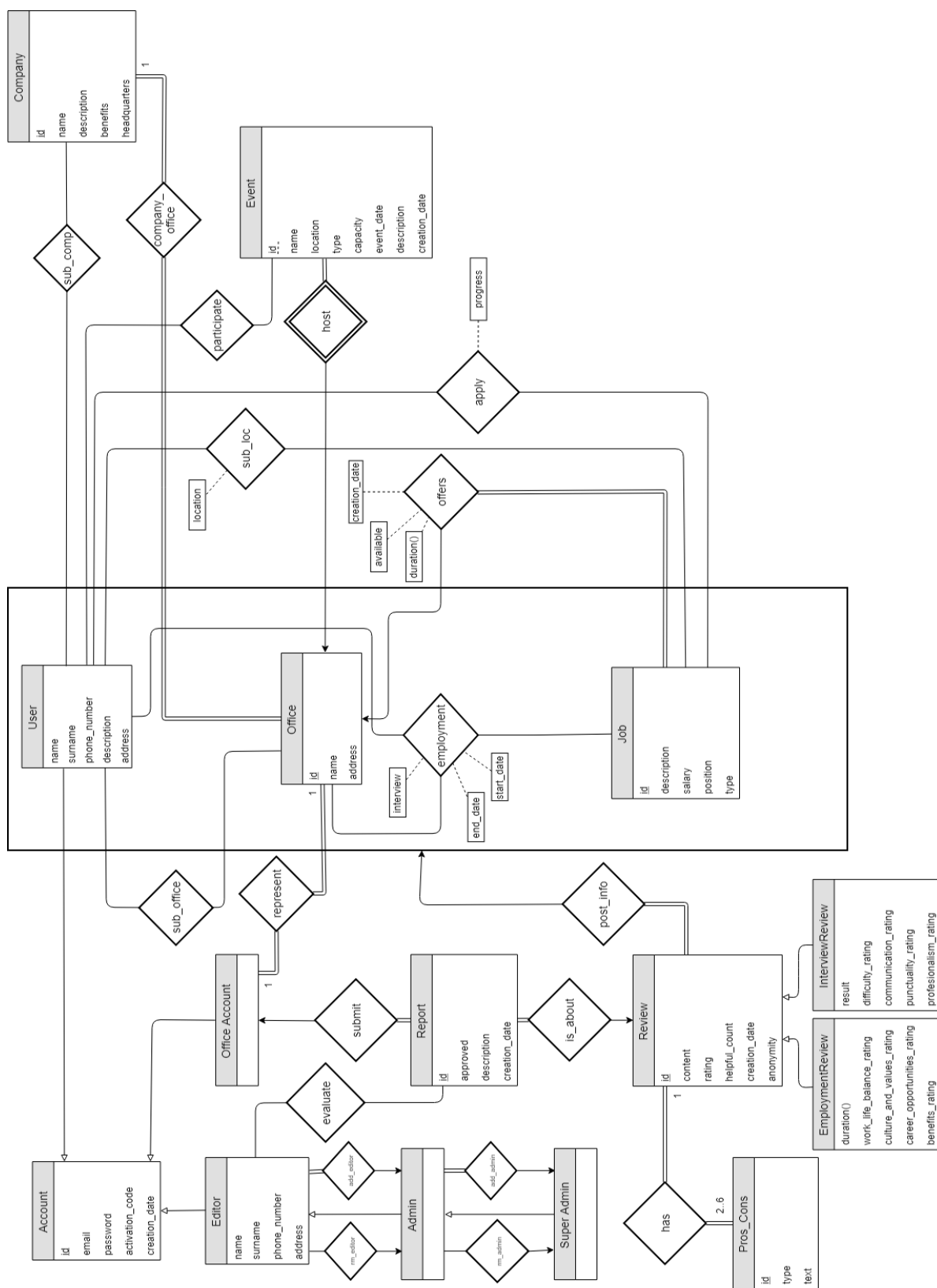


Figure 1: Entity Relationship diagram for Suit Up [1][2]

## 2 Relational Schemas

### 2.1 Account

#### Relational Model

Account(id, email, password, activation\_code, creation\_date)

#### Functional Dependencies

$\text{id} \rightarrow \text{email}, \text{password}, \text{activation\_code}, \text{creation\_date}$

$\text{email} \rightarrow \text{id}$

#### Candidate Keys

$\{(\text{id}), (\text{email})\}$

#### Normal Form

BCNF

#### Table Definition

```
CREATE TABLE Account(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    email VARCHAR(64) NOT NULL UNIQUE,  
    password VARCHAR(32) NOT NULL,  
    activation_code varchar(50) DEFAULT '',  
    creation_date TIMESTAMP DEFAULT now()  
) ENGINE = INNODB;
```

## 2.2 Office\_Account

### Relational Model

Office\_Account(id)

### Functional Dependencies

None

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Office_Account(  
    id INT PRIMARY KEY,  
    FOREIGN KEY (id) REFERENCES Account(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.3 Editor

### Relational Model

Editor(id, name, surname, phone\_number, address)

### Functional Dependencies

$\text{id} \rightarrow \text{name, surname, phone\_number, address}$

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Editor(  
    id INT PRIMARY KEY,  
    name VARCHAR(32) ,  
    surname VARCHAR(32) ,  
    phone_number CHAR(16) UNIQUE,  
    address VARCHAR(255) ,  
    FOREIGN KEY (id) REFERENCES Account(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```



## 2.4 Admin

### Relational Model

Admin(id)

### Functional Dependencies

None

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Admin(  
    id INT PRIMARY KEY,  
    FOREIGN KEY (id) REFERENCES Editor(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.5 Super\_Admin

### Relational Model

Super\_Admin(id)

### Functional Dependencies

None

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Super_Admin(  
    id INT PRIMARY KEY,  
    FOREIGN KEY (id) REFERENCES Admin(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.6 Evaluate

### Relational Model

Evaluate(editor\_id, report\_id)

### Functional Dependencies

None

### Candidate Keys

{(editor\_id, report\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Evaluate(  
    editor_id INT,  
    report_id INT,  
    PRIMARY KEY (editor_id, report_id),  
    FOREIGN KEY (editor_id) REFERENCES Editor(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (report_id) REFERENCES Report(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.7 Add\_Editor

### Relational Model

Add\_Editor(editor\_id, admin\_id)

### Functional Dependencies

editor\_id  $\rightarrow$  admin\_id

### Candidate Keys

{(editor\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Add_Editor(  
    editor_id INT PRIMARY KEY,  
    admin_id INT,  
    FOREIGN KEY (editor_id) REFERENCES Editor(id)  
        ON DELETE RESTRICT  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (admin_id) REFERENCES Admin(id)  
        ON DELETE SET NULL  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

## 2.8 Remove\_Editor

### Relational Model

Remove\_Editor(editor\_id, admin\_id)

### Functional Dependencies

editor\_id  $\rightarrow$  admin\_id

### Candidate Keys

{(editor\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Remove_Editor(  
    editor_id INT PRIMARY KEY,  
    admin_id INT,  
    FOREIGN KEY (editor_id) REFERENCES Editor(id)  
        ON DELETE RESTRICT  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (admin_id) REFERENCES Admin(id)  
        ON DELETE SET NULL  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.9 Add\_Admin

### Relational Model

Add\_Admin(admin\_id, super\_admin\_id )

### Functional Dependencies

admin\_id  $\rightarrow$  super\_admin\_id

### Candidate Keys

{(admin\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Add_Admin(  
    admin_id INT PRIMARY KEY,  
    super_admin_id INT,  
    FOREIGN KEY (admin_id) REFERENCES Admin(id)  
        ON DELETE RESTRICT  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (super_admin_id) REFERENCES Super_Admin(id)  
        ON DELETE SET NULL  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.10 Remove\_Admin

### Relational Model

Remove\_Admin(admin\_id, super\_admin\_id )

### Functional Dependencies

admin\_id  $\rightarrow$  super\_admin\_id

### Candidate Keys

{(admin\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Remove_Admin(  
    admin_id INT PRIMARY KEY,  
    super_admin_id INT,  
    FOREIGN KEY (admin_id) REFERENCES Admin(id)  
        ON DELETE RESTRICT  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (super_admin_id) REFERENCES Super_Admin(id)  
        ON DELETE SET NULL  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.11 Report

### Relational Model

Report(id, approved, description, creation\_date, office\_account\_id, review\_id)

### Functional Dependencies

$id \rightarrow \text{approved, description, creation\_date, office\_account\_id, review\_id}$

$\text{office\_account\_id, review\_id} \rightarrow id$

### Candidate Keys

$\{(id), (\text{office\_account\_id, review\_id})\}$

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Report(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    approved BOOLEAN NOT NULL,  
    description VARCHAR(255),  
    creation_date TIMESTAMP DEFAULT now(),  
    office_account_id INT,  
    review_id INT,  
    FOREIGN KEY (office_account_id) REFERENCES Office_Account(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (review_id) REFERENCES Review(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```



## 2.12 Review

### Relational Model

Review(id, content, rating, helpful\_count, creation\_date, anonymity, user\_id, job\_id, office\_id, interview)

### Functional Dependencies

$id \rightarrow \text{content, rating, helpful\_count, creation\_date, anonymity, user\_id, job\_id, office\_id, interview}$

$\text{user\_id, job\_id, office\_id, interview} \rightarrow id$

$\text{job\_id} \rightarrow \text{office\_id}$

### Candidate Keys

$\{(id), (\text{user\_id, job\_id, office\_id, interview})\}$

### Normal Form

3NF

### Table Definition

```
CREATE TABLE Review(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    content VARCHAR(512) NOT NULL,  
    rating INT NOT NULL,  
    helpful_count INT NOT NULL,  
    creation_date TIMESTAMP DEFAULT now() ,  
    anonymity BOOLEAN NOT NULL,  
    user_id INT,  
    job_id INT,  
    office_id INT,  
    interview BOOLEAN NOT NULL,  
    FOREIGN KEY (user_id , job_id , office_id , interview) REFERENCES Employment(  
        user_id , job_id , office_id , interview)
```

```
        ON DELETE CASCADE
        ON UPDATE RESTRICT
    ) ENGINE = INNODB;
```

## 2.13 Employment\_Review

### Relational Model

Employment\_Review(id, work\_life\_balance\_rating, benefits\_rating, culture\_and\_values\_rating, career\_opportunities\_rating)

### Functional Dependencies

$id \rightarrow \text{work\_life\_balance\_rating, culture\_and\_values\_rating, career\_opportunities\_rating, benefits\_rating}$

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Employment_Review(  
    id INT PRIMARY KEY,  
    work_life_balance_rating INT NOT NULL,  
    culture_and_values_rating INT NOT NULL,  
    career_opportunities_rating INT NOT NULL,  
    benefits_rating INT NOT NULL,  
    FOREIGN KEY (id) REFERENCES Review(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.14 Interview\_Review

### Relational Model

Interview\_Review(id, result, difficulty\_rating, communication\_rating, punctuality\_rating, professionalism\_rating)

### Functional Dependencies

$id \rightarrow result, difficulty\_rating, communication\_rating, punctuality\_rating, professionalism\_rating$

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Interview_Review(  
    id INT PRIMARY KEY,  
    result INT NOT NULL,  
    difficulty_rating INT NOT NULL,  
    communication_rating INT NOT NULL,  
    punctuality_rating INT NOT NULL,  
    professionalism_rating INT NOT NULL,  
    FOREIGN KEY (id) REFERENCES Review(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.15 Pros\_Cons

### Relational Model

Pros\_Cons(id, type, text)

### Functional Dependencies

$\text{id} \rightarrow \text{type}, \text{text}$

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Pros_Cons(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    type BOOLEAN NOT NULL,  
    text VARCHAR(64) NOT NULL  
) ENGINE = INNODB;
```

## 2.16 Review\_Proc\_Cons

### Relational Model

Review\_Proc\_Cons(review\_id, proc\_cons\_id)

### Functional Dependencies

None

### Candidate Keys

{(review\_id, proc\_cons\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Review_Proc_Cons(  
    review_id INT,  
    proc_cons_id INT,  
    PRIMARY KEY(review_id , proc_cons_id) ,  
    FOREIGN KEY (review_id) REFERENCES Review(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (proc_cons_id) REFERENCES Proc_Cons(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.17 Office\_Sub

### Relational Model

Office\_Sub(office\_id, user\_id)

### Functional Dependencies

None

### Candidate Keys

{(office\_id, user\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Office_Sub(  
    office_id INT,  
    user_id INT,  
    PRIMARY KEY (office_id , user_id),  
    FOREIGN KEY (office_id) REFERENCES Office(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (user_id) REFERENCES User(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.18 Job

### Relational Model

Job(id, description, salary, job\_position, type)

### Functional Dependencies

$\text{id} \rightarrow \text{description, salary, job\_position, type}$

### Candidate Keys

$\{(\text{id})\}$

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Job(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    description VARCHAR(512) NOT NULL,  
    salary INT NOT NULL,  
    job_position VARCHAR(255) NOT NULL,  
    type VARCHAR(16) NOT NULL  
) ENGINE = INNODB;
```



## 2.19 Employment

### Relational Model

Employment(office\_id, user\_id, job\_id, interview, start\_date, end\_date)

### Functional Dependencies

office\_id, user\_id, job\_id, interview  $\rightarrow$  start\_date, end\_date

job\_id  $\rightarrow$  office\_id

### Candidate Keys

{(office\_id, user\_id, job\_id, interview)}

### Normal Form

3NF

### Table Definition

```
CREATE TABLE Employment(  
    office_id INT,  
    user_id INT,  
    job_id INT,  
    interview BOOLEAN NOT NULL,  
    start_date DATE NOT NULL,  
    end_date DATE,  
    PRIMARY KEY(office_id, user_id, job_id, interview),  
    FOREIGN KEY(office_id) REFERENCES Office(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY(user_id) REFERENCES User(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY(job_id) REFERENCES Job(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.20 Company

### Relational Model

Company(id, name, description, benefits, headquarters)

### Functional Dependencies

$\text{id} \rightarrow \text{name, description, benefits, headquarters}$

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Company(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(64) NOT NULL UNIQUE,  
    description VARCHAR(512) NOT NULL,  
    benefits VARCHAR(255) ,  
    headquarters VARCHAR(255) NOT NULL  
) ENGINE = INNODB;
```

## 2.21 Office

### Relational Model

Office(id, name, address, company\_id, acc\_id)

### Functional Dependencies

$id \rightarrow \text{name, address, company\_id, acc\_id}$

$\text{acc\_id} \rightarrow id$

### Candidate Keys

$\{(id), (\text{acc\_id})\}$

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Office(  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(32) NOT NULL,  
    address VARCHAR(255) NOT NULL,  
    acc_id INT,  
    company_id INT,  
    FOREIGN KEY (acc_id) REFERENCES Office_Account(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (company_id) REFERENCES Company(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

## 2.22 Job\_Offers

### Relational Model

Job\_Offers(job\_id, office\_id, creation\_date, available)

### Functional Dependencies

$\text{job\_id} \rightarrow \text{office\_id}, \text{creation\_date}, \text{available}$

### Candidate Keys

$\{(\text{job\_id})\}$

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Job_Offers(  
    job_id INT PRIMARY KEY,  
    office_id INT,  
    creation_date TIMESTAMP DEFAULT now() ,  
    available BOOLEAN NOT NULL,  
    FOREIGN KEY (office_id) REFERENCES Office(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (job_id) REFERENCES Job(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

## 2.23 User

### Relational Model

User(id, name, surname, phone\_number, description, address)

### Functional Dependencies

$\text{id} \rightarrow \text{name, surname, phone\_number, description, address}$

### Candidate Keys

{(id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE User(  
    id INT PRIMARY KEY,  
    name VARCHAR(32) NOT NULL,  
    surname VARCHAR(32) NOT NULL,  
    phone_number CHAR(16) UNIQUE,  
    description VARCHAR(512) ,  
    address VARCHAR(255) ,  
    FOREIGN KEY (id) REFERENCES Account(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

## 2.24 Event

### Relational Model

Event(id, office\_id, name, location, type, capacity, event\_date, description, creation\_date)

### Functional Dependencies

$id \rightarrow office\_id, name, location, type, capacity, , event\_date, description, creation\_date$

$office\_id, name, location, type, capacity, event\_date \rightarrow id$

### Candidate Keys

$\{(id), (office\_id, name, location, type, capacity, event\_date)\}$

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Event(  
    event_id INT PRIMARY KEY AUTO_INCREMENT,  
    office_id INT,  
    name VARCHAR(64) NOT NULL,  
    location VARCHAR(255) NOT NULL,  
    type VARCHAR(32) NOT NULL,  
    capacity INT NOT NULL,  
    event_date DATE NOT NULL,  
    description VARCHAR(512),  
    creation_date TIMESTAMP DEFAULT now(),  
    FOREIGN KEY (office_id) REFERENCES Office(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE = INNODB;
```

## 2.25 User\_Apply

### Relational Model

User\_Apply(user\_id, job\_id, progress)

### Functional Dependencies

user\_id, job\_id  $\rightarrow$  progress

### Candidate Keys

{(user\_id, job\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE User_Apply(  
    user_id INT,  
    job_id INT,  
    progress TINYINT NOT NULL,  
    PRIMARY KEY(user_id, job_id),  
    FOREIGN KEY (user_id) REFERENCES User(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (job_id) REFERENCES Job(id)  
        ON DELETE RESTRICT  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

## 2.26 User\_Event

### Relational Model

User\_Event(user\_id, event\_id)

### Functional Dependencies

None

### Candidate Keys

{(user\_id, event\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE User_Event(  
    user_id INT,  
    event_id INT,  
    PRIMARY KEY(user_id, event_id),  
    FOREIGN KEY (user_id) REFERENCES User(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (event_id) REFERENCES Event(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```



## 2.27 Company\_Sub

### Relational Model

Company\_Sub(company\_id, user\_id)

### Functional Dependencies

None

### Candidate Keys

{(company\_id, user\_id)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Company_Sub(  
    company_id INT,  
    user_id INT,  
    PRIMARY KEY(company_id, user_id),  
    FOREIGN KEY (company_id) REFERENCES Company(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (user_id) REFERENCES User(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

## 2.28 Location\_Sub

### Relational Model

Location\_Sub(job\_id, user\_id, location)

### Functional Dependencies

None

### Candidate Keys

{(job\_id, user\_id, location)}

### Normal Form

BCNF

### Table Definition

```
CREATE TABLE Location_Sub(  
    job_id INT,  
    user_id INT,  
    location VARCHAR(64) ,  
    PRIMARY KEY(job_id , user_id , location) ,  
    FOREIGN KEY (job_id) REFERENCES Job(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT,  
    FOREIGN KEY (user_id) REFERENCES User(id)  
        ON DELETE CASCADE  
        ON UPDATE RESTRICT  
) ENGINE=INNODB;
```

### 3 Normalization of Tables

The tables are all in at least 3NF, thus no normalization is required.

## 4 Functional Components

### 4.1 Use Case Diagram

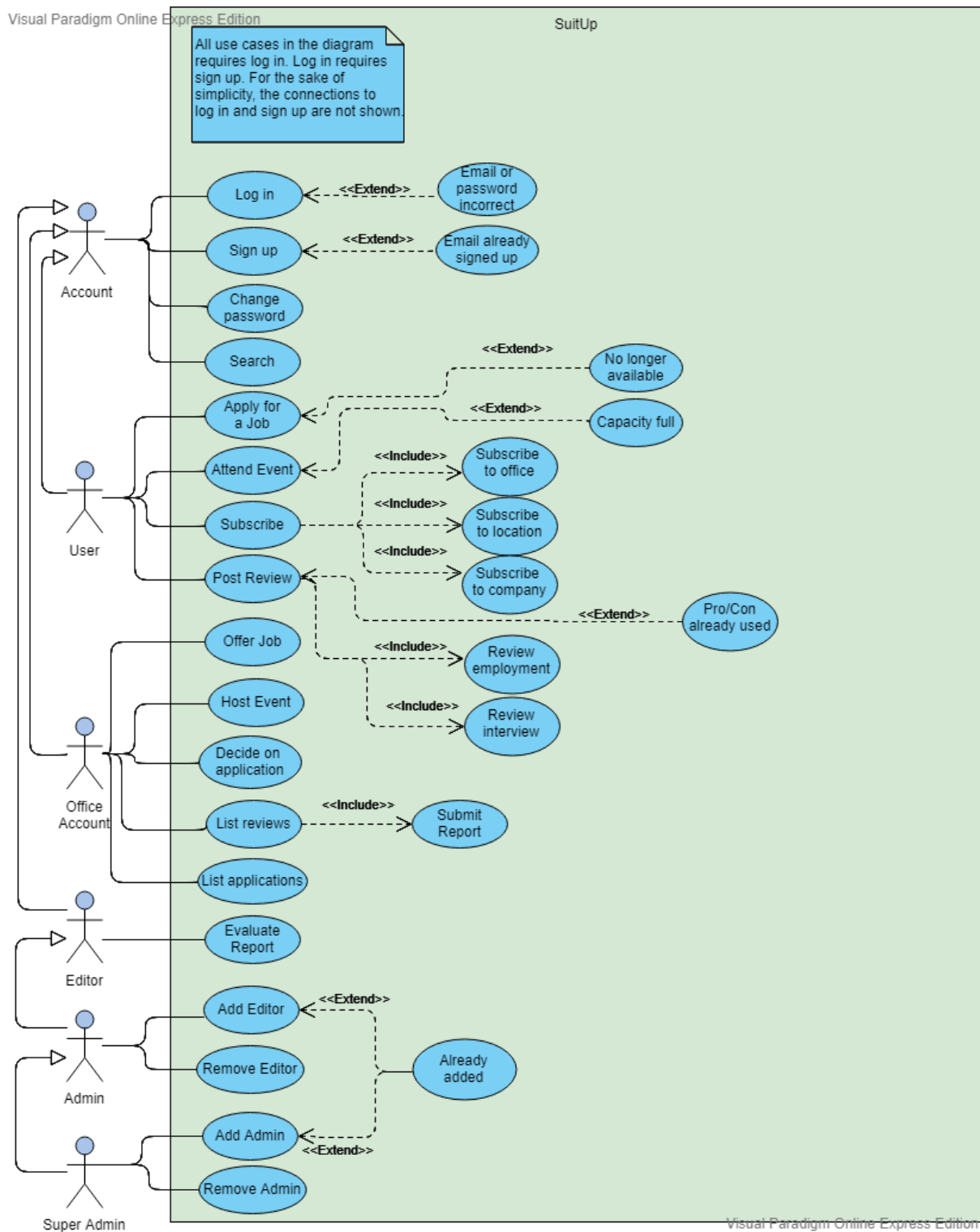


Figure 2: Entity Relationship diagram for Suit Up [1][2]

## 4.2 Scenarios

### Log In

Use Case Name:	Log in
Participating Actor:	Account
Entry Condition:	<ul style="list-style-type: none"><li>• Account owner is in the log in screen.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Account owner has logged in to the system.</li></ul>
Flow of Events:	
1.Basic Flow:	
1.1. Account owner inputs the email address for their account.	
1.2. Account owner inputs their password..	
1.3. System checks the account's credentials.	
1.4. Account logs in to the system.	
1.5. The system displays the corresponding screen based on Account type.	
2. Exceptional Cases:	
2.1. User types an email address that doesn't exist in the system or a password that doesn't match with the password in the system.	
2.1.1. System displays an error indicating the input email or password was incorrect.	
Special/Quality Requirements:	<ul style="list-style-type: none"><li>• Extends the "Email or password incorrect" use case which corresponds to exceptional cases of this use case.</li></ul>

Figure 3: Textual use case for log in

## Sign Up

Use Case Name:	Sign Up
Participating Actor:	Account
Entry Condition:	<ul style="list-style-type: none"> <li>Account owner is in the sign up screen.</li> </ul>
Exit Condition:	<ul style="list-style-type: none"> <li>Account owner has signed up to the system.</li> </ul>
Flow of Events:	
1. Basic Flow:	
1.1. Account owner fills the related fields in the sign up screen.	
	1.2. System checks the input information for validation.
	1.3. System sends the account owner's email address their account validation information.
1.4. Account owner inputs the validation information they received via email.	
	1.5. System checks the account's validation information.
	1.6. System redirects the account to their page.
2. Exceptional Cases:	
2.1. Account owner inputs invalid information in some field in the signup screen.	
	2.1.1. System displays an error indicating that information in one of the fields was invalid.
2.2. Account owner inputs invalid validation information.	
	2.1.1. System rejects the validation information and alerts the user.
Special/Quality Requirements:	<ul style="list-style-type: none"> <li>Extends the "Email already signed up" use case which is an exceptional case.</li> </ul>

Figure 4: Textual use case for sign up

## Change Password

Use Case Name:	Change Password
Participating Actor:	Account
Entry Condition:	<ul style="list-style-type: none"><li>Account owner is in their respective profile page based on the account type.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>Account owner has successfully changed their password.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Account owner chooses the change password option in their edit profile page.	
	1.2. System displays the change password screen.
1.3. Account owner inputs their new password, and re-enters it.	
	1.5. System checks the equality of passwords and validity of the password.
	1.6. System displays an alert stating that the password change has been successfully completed.
2. Exceptional Cases:	
2.1. Account owner inputs two different passwords.	
	2.1.1. System displays an error telling the account owner that the passwords do not match.

Figure 5: Textual use case for change password

## Apply for a Job

Use Case Name:	Apply for a Job
Participating Actor:	User
Entry Condition:	<ul style="list-style-type: none"><li>• User is searching for a job using the search function in home page or browsing recent jobs from a company's page.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• User has successfully applied for a job.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. User clicks on one of the listed jobs after a search with their desired parameters.	
	1.2. System displays the job post screen that displays more information specific to that job, and also has the apply button.
1.3. User clicks the apply button.	
	1.4. System registers the application.
2. Alternative Flow:	
2.1. User clicks on one of the listed jobs from the recent jobs in a company's page.	
	2.2. System displays the job post screen that displays more information specific to that job, and also has the apply button.
2.3. User clicks the apply button.	
	2.4. System registers the application.
3. Exceptional Cases:	
3.1. The job is no longer available.	
	3.2. System displays an error telling the account owner that the application period for that specific job is over.

Figure 6: Textual use case for applying for a job



## Search for a Job

Use Case Name:	Search for a Job
Participating Actor:	User
Entry Condition:	<ul style="list-style-type: none"><li>• User is in the home page.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• User has received a list of jobs fitting their search.</li></ul>
Flow of Events:	
1.Basic Flow:	
1.1. User inputs a search query in the search bar.	
	1.2. System displays the jobs that fit the search query.

Figure 7: Textual use case for searching for a job

## Attend Event

Use Case Name:	Attend Event
Participating Actor:	User
Entry Condition:	<ul style="list-style-type: none"><li>• User is browsing events.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• User has signed up to participate for an event.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. User clicks on an event's name.	
	1.2. System displays the event post screen that displays more information specific to that event, and also has the button to participate in that event.
1.3. User clicks the participate button.	
	1.4. System registers the user's participation.
2. Exceptional Cases:	
2.1. The event has no remaining capacity left.	
	2.2. System displays an error telling the user that the event is already full in capacity.

Figure 8: Textual use case for attending an event

## Subscribe

Use Case Name:	Subscribe
Participating Actor:	User
Entry Condition:	<ul style="list-style-type: none"><li>• User is browsing in home page, an office page or a company page.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• User has subscribed to a location, office page or company.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. User clicks the subscribe button in the home page.	
	1.2. System displays a pop-up screen asking for location and job type information to subscribe to.
1.3. User specifies the location and type of jobs they want to subscribe to.	
	1.4. System registers the user's subscription.
2. Alternative Flow:	
2.1. User clicks the subscribe button in a company page.	
	2.2. System registers the user's subscription.
3. Alternative Flow 2:	
3.1. User clicks the subscribe button in an office page.	
	3.2. System registers the user's subscription.

Figure 9: Textual use case for subscribing to jobs based on location, company or office.

## Post Review

Use Case Name:	Post Review
Participating Actor:	User
Entry Condition:	<ul style="list-style-type: none"><li>• User is in an office page.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• User has posted a review about an employment or interview experience.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. User clicks the post review button and chooses the option to do an interview review.	
	1.2. System displays the fields to rate on a scale of 1-5 stars, an empty text field for content of review, interview result and possibly offered salary.
1.3. User fills in all the required fields.	
	1.4. System accepts the review and displays it.
2. Alternative Flow:	
2.1. User clicks the post review button and chooses the option to do an employment review.	
	2.2. System displays the fields to rate on a scale of 1-5 stars, an empty text field for content of review, job position and type.
3. Exceptional Cases:	
3.1. The user chooses a previously chosen pro or con for this particular review.	
	2.2. System displays an error telling the user that the same pro or con can't be used twice.

Figure 10: Textual use case for posting reviews

## Offer Job


Use Case Name:	Offer Job
Participating Actor:	Office Account
Entry Condition:	• Office Account owner is in their Office Panel.
Exit Condition:	• Office Account owner has posted a job offering.
Flow of Events:	
1. Basic Flow:	
1.1. Office Account owner presses the "Post New Job" button.	
	1.2. System displays a pop-up screen in which the Office Account owner can enter the specifics for the job and the requirements for the candidate that they are looking for as description.
1.3. Office Account owner fills in all the required fields.	
	1.4. System accepts the job offering and displays it when searched for.
2. Exceptional Cases:	
2.1. Office Account owner makes an error while filling out the form.	
	2.2. System displays an error telling the user that they entered illegal information 

Figure 11: Textual use case for offering a job

## Host Event

Use Case Name:	Host Event
Participating Actor:	Office Account
Entry Condition:	• Office Account owner is in their Office Panel.
Exit Condition:	• Office Account owner has posted an event.
Flow of Events:	
1. Basic Flow:	
1.1. Office Account owner presses the "Host new Event" button.	
	1.2. System displays a pop-up screen in which the Office Account owner can enter the information for the event, such as a description, location and capacity.
1.3. Office Account owner fills in all the fields.	
	1.4. System accepts the event and displays it when searched for.
2. Exceptional Cases:	
2.1. Office Account owner makes an error while filling out the form.	
	2.2. System displays an error telling the user that they entered illegal information.

Figure 12: Textual use case for hosting an event

## Decide on Application

Use Case Name:	Decide on Application
Participating Actor:	Office Account
Entry Condition:	<ul style="list-style-type: none"><li>• Office Account owner is in their Office Panel looking for candidates that responded to a job offering.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Office Account owner has made a decision on an application.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Office Account owner clicks the "See Applicants" button for a particular job that they have offered.	
	1.2. System displays all the applicants that have applied for this job.
1.3. Office Account owner chooses "Interview" option for a particular applicant.	
	1.4. System alerts the applicant that the company has called them for an interview for that particular job offering.
2. Alternative Flow:	
2.1. Office Account owner clicks the "See Applicants" button for a particular job that they have offered.	
	2.2. System displays all the applicants that have applied for this job.
2.3. Office Account owner chooses "Reject" option for a particular applicant.	
	2.4. System alerts the applicant that their application has been rejected.

Figure 13: Textual use case for deciding on an application

## List Reviews

Use Case Name:	List Reviews
Participating Actor:	Office Account
Entry Condition:	<ul style="list-style-type: none"><li>• Office Account owner is in their Office Panel.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Office Account owner has listed the reviews made about their office.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Office Account owner clicks the "List Reviews" button in their Office Panel.	
	1.2. System displays all the reviews that have been made about their office, sorted by time posted.

Figure 14: Textual use case for listing reviews



## Submit Report

Use Case Name:	Submit Report
Participating Actor:	Office Account
Entry Condition:	<ul style="list-style-type: none"><li>• Office Account owner is in their Office Panel and they have listed the reviews made about their office.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Office Account owner has reported a review made about their company.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Office Account owner clicks the "Report" button next to a review.	
	1.2. System displays a pop-up screen requesting a brief description about why the Office Account owner wants to report the review.
1.3. Office Account owner fills in the description and reports the review.	
	1.4. System accepts the report.

Figure 15: Textual use case for submitting report

## List Applications

Use Case Name:	List Applications
Participating Actor:	Office Account
Entry Condition:	<ul style="list-style-type: none"><li>• Office Account owner is in their Office Panel and they are looking at their jobs posted.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Office Account owner has received a list of applicants for a particular job.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Office Account owner clicks the "List Applicants" button next to a job offering of theirs.	
	1.2. System displays the applicants that have applied for that particular job.

Figure 16: Textual use case for listing applications for a job

## Evaluate

Use Case Name:	Evaluate Report
Participating Actor:	Editor
Entry Condition:	<ul style="list-style-type: none"><li>• Editor is in the Editor Panel looking at reports.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Editor has evaluated a report.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Editor reads the contents of a report and if necessary, he approves the report by clicking "Approve Removal" button.	
	1.2. System records that the editor has chosen to approve the removal.

Figure 17: Textual use case for evaluating a report

## Add Editor

Use Case Name:	Add Editor
Participating Actor:	Admin
Entry Condition:	<ul style="list-style-type: none"><li>• Admin is in the admin panel.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Admin has added an editor.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Admin presses the "Add Editor" button.	
	1.2. System asks for user information of editor.
1.3. Admin provides the information about the user they want to give editor privileges to.	
	1.4. System adds the editor.
2. Exceptional Cases:	
2.1. Admin provides invalid credentials for the user.	
	2.1.1. System warns the admin about the wrong credentials.
2.2. Editor is already added.	
	2.2.1. System warns the admin about editor already being added.

Figure 18: Textual use case for adding an editor

## Remove Editor

Use Case Name:	Remove Editor
Participating Actor:	Admin
Entry Condition:	<ul style="list-style-type: none"><li>• Admin is in the admin panel.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Admin has removed an editor.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Admin presses the "View Editors" button.	
	1.2. System displays all editors.
1.3. Admin presses the "Remove Editor" button next to an editor.	
	1.4. System removes the editor.

Figure 19: Textual use case for removing an editor

## Add Admin

Use Case Name:	Add Admin
Participating Actor:	Super Admin
Entry Condition:	<ul style="list-style-type: none"><li>• Super Admin is in the super admin panel.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Super Admin has added an admin.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Super Admin presses the "Add Admin" button.	
1.2. System asks for user information of admin.	
1.3. Super Admin provides the information about the user they want to give admin privileges to.	
1.4. System adds the admin.	
2. Exceptional Cases:	
2.1. Super Admin provides invalid credentials for the user.	
2.1.1. System warns the super admin about the wrong credentials.	
2.2. Admin is already added.	
2.2.1. System warns the super admin about admin already being added.	

Figure 20: Textual use case for adding an admin

## Remove Admin

Use Case Name:	Remove Admin
Participating Actor:	Super Admin
Entry Condition:	<ul style="list-style-type: none"><li>• Super Admin is in the admin panel.</li></ul>
Exit Condition:	<ul style="list-style-type: none"><li>• Super Admin has removed an admin.</li></ul>
Flow of Events:	
1. Basic Flow:	
1.1. Super Admin presses the "View Admins" button.	
1.2. System displays all admins.	
1.3. Super Admin presses the "Remove Admin" button next to an admin.	
1.4. System removes the admin.	

Figure 21: Textual use case for removing an admin

## 4.3 Algorithms and Data Structures

### Password hashing

For the sake of the security of the accounts, we are directly hashing the passwords and storing them in the database as hashed. Hence, we never transfer the user's password data outside their local machines. The password hashing function uses hash maps. We did not use any other specific data structures during the implementation for the time being.

### Account Activation

While signing up a user, we are creating a unique activation code which is assigned as an attribute to the tuple. Momentarily, we are sending an activation email to the user which contains a link to our webpage, where s/he can type in the activation code given in the email. If the activation codes match, the attribute is set to be "**activated**" in the corresponding tuple.

### Company Average Ratings

We will calculate the overall average and per specific rating average in order to show the averages in the company specific page.



## 5 User Interface Design and Corresponding SQL Statements

### Log in

The page below is the UI design for the log in page for our system.

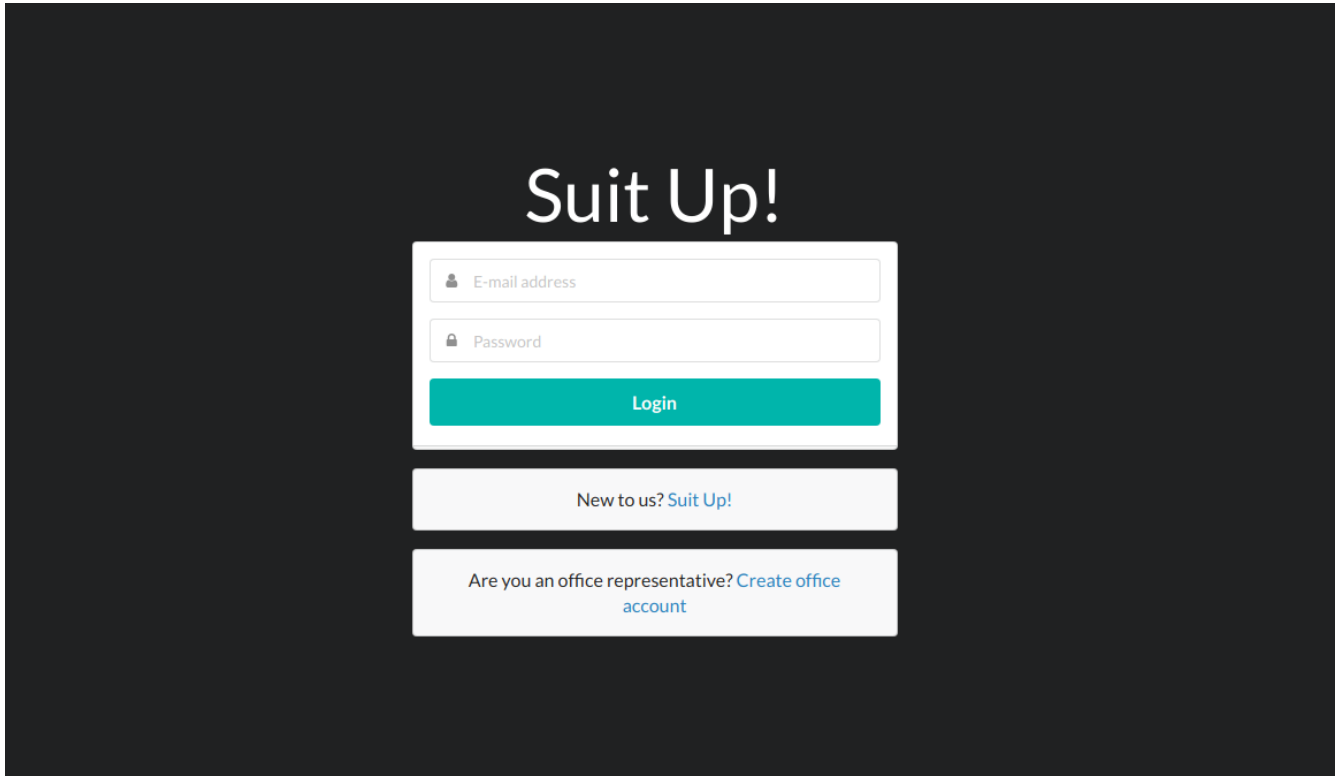


Figure 22: UI design of log in

The corresponding SQL statements are as follows:

- **Login validation:** We need to obtain the hashed password for the entered email address. Password matching will be done using PHP. If query below returns empty set, it means no user with the entered email exists in the system. We will also hold the ID of the user as *@user\_id* to check the account type.

```
SELECT id , password
FROM Account
WHERE email = @email;
```

We will use sessions to navigate and limit the visitors according to account types. The type-checking is as follows:

**User:**

```
SELECT id  
FROM User  
WHERE id = @user_id
```

**Super\_Admin:**

```
SELECT id  
FROM Super_Admin  
WHERE id = @user_id
```

**Admin:**

```
SELECT id  
FROM Admin  
WHERE id = @user_id
```

**Editor:**

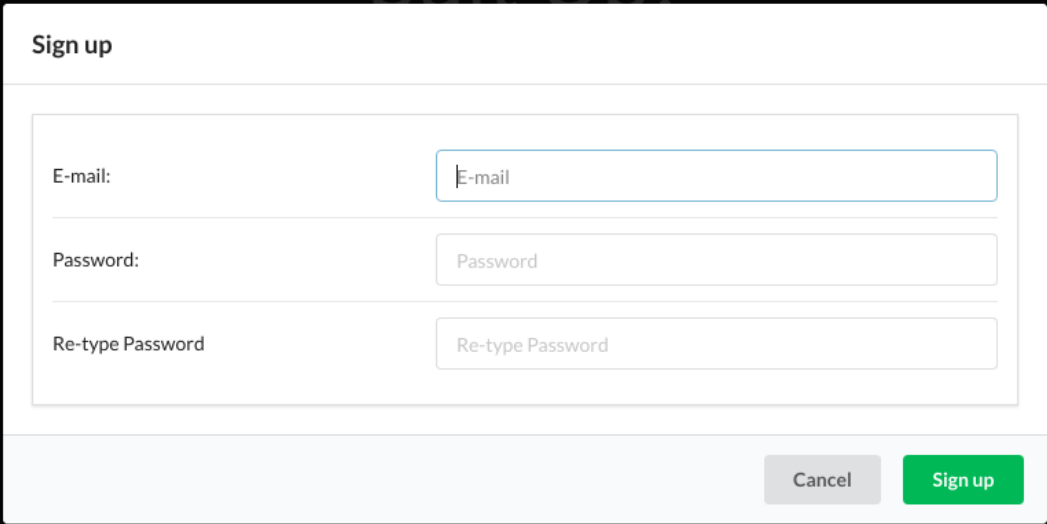
```
SELECT id  
FROM Editor  
WHERE id = @user_id
```

**Office\_Account:**

```
SELECT id  
FROM Office_Account  
WHERE id = @user_id
```

## Sign up pop up

The page below is the sign up page for the users of our system.



The image shows a 'Sign up' pop-up window. It has a title bar with the text 'Sign up' and a close button (X) in the top right corner. The main content area contains three input fields: 'E-mail:', 'Password:', and 'Re-type Password'. Each input field has a placeholder text that matches the label. Below the input fields are two buttons: 'Cancel' and 'Sign up'. The 'Sign up' button is green, while the 'Cancel' button is grey.

Figure 23: UI design of sign up pop up

The corresponding SQL statements are as follows:

- We need to check if another account with the same email address exists in our system.

```
SELECT email
FROM Account
WHERE email = @email;
```

- If the above query returns an empty set, it means the email is valid for signing up. We will first insert into Account table since user IDs should be determined according to auto incrementation of the ID in Account table. Since we will also be implementing email activation, we will generate a verification code and initially it will be set as the activation code attribute. The activation code will be stored as *@activation\_code*. We will also hash the functions prior to adding them to the database. The hashed password will be stored as *@hashed\_password*.

```
INSERT INTO Account (email, password, activation_code)
VALUES (@email, @hashed_password, @activation_code );
```

- After adding the new account, now we need to add it to the corresponding account class table. First, we need to obtain the assigned ID of the account from the Account table for the sake of referential integrity between child tables and the parent table. We will store the obtained ID as *@user\_id*.

```
SELECT id
FROM Account
WHERE email = @email;
```

If **user**, we will execute the query below.

```
INSERT INTO User (id, name, surname, phone_number, description, address)
VALUES (@user_id, @name, @surname, @phone_number, @description, @address);
```

If **editor**, we will execute the query below. However, this operation is only allowed for admins and super admins to conduct. Even though the corresponding UI is in the scope of our additional features, we are including the SQL for insertion to Editor table.

```
INSERT INTO Editor (id, name, surname, phone_number, address)
VALUES (@user_id, @name, @surname, @phone_number, @address );
```

If **office account**, we will execute the query below.

```
INSERT INTO Office_Account (id)
VALUES (@user_id);
```

- To check whether the entered activation code, *@activation\_code*, is valid, we expect the below query to return a valid tuple.

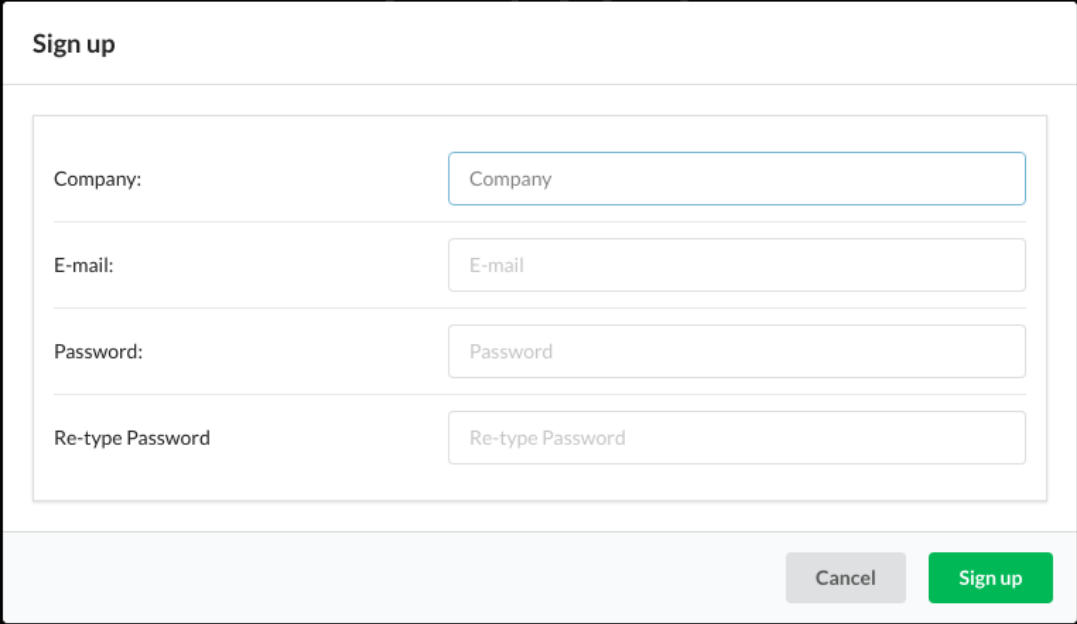
```
SELECT *
FROM Account
WHERE (activation_code = @activation_code
      AND email = @email);
```

If this query returns a non-empty set, we will classify it as an activated account.

```
UPDATE Account
SET activation_code = 'activated'
WHERE (email = @email
      AND activation_code = @activation_code);
```

## Sign up for an office account

The page below is the sign up page for the office accounts of our system.



The image shows a 'Sign up' dialog box with a title bar containing a close button (X). The dialog has a white background and is set against a dark background. It contains four input fields with labels to their left: 'Company:', 'E-mail:', 'Password:', and 'Re-type Password'. Each input field has a light blue border and contains placeholder text matching its label. At the bottom right of the dialog, there are two buttons: a grey 'Cancel' button and a green 'Sign up' button.

Figure 24: UI design of sign up for an office account

The required SQL statements are included in the previous section.

## Main page

The home page of our website is as follows.

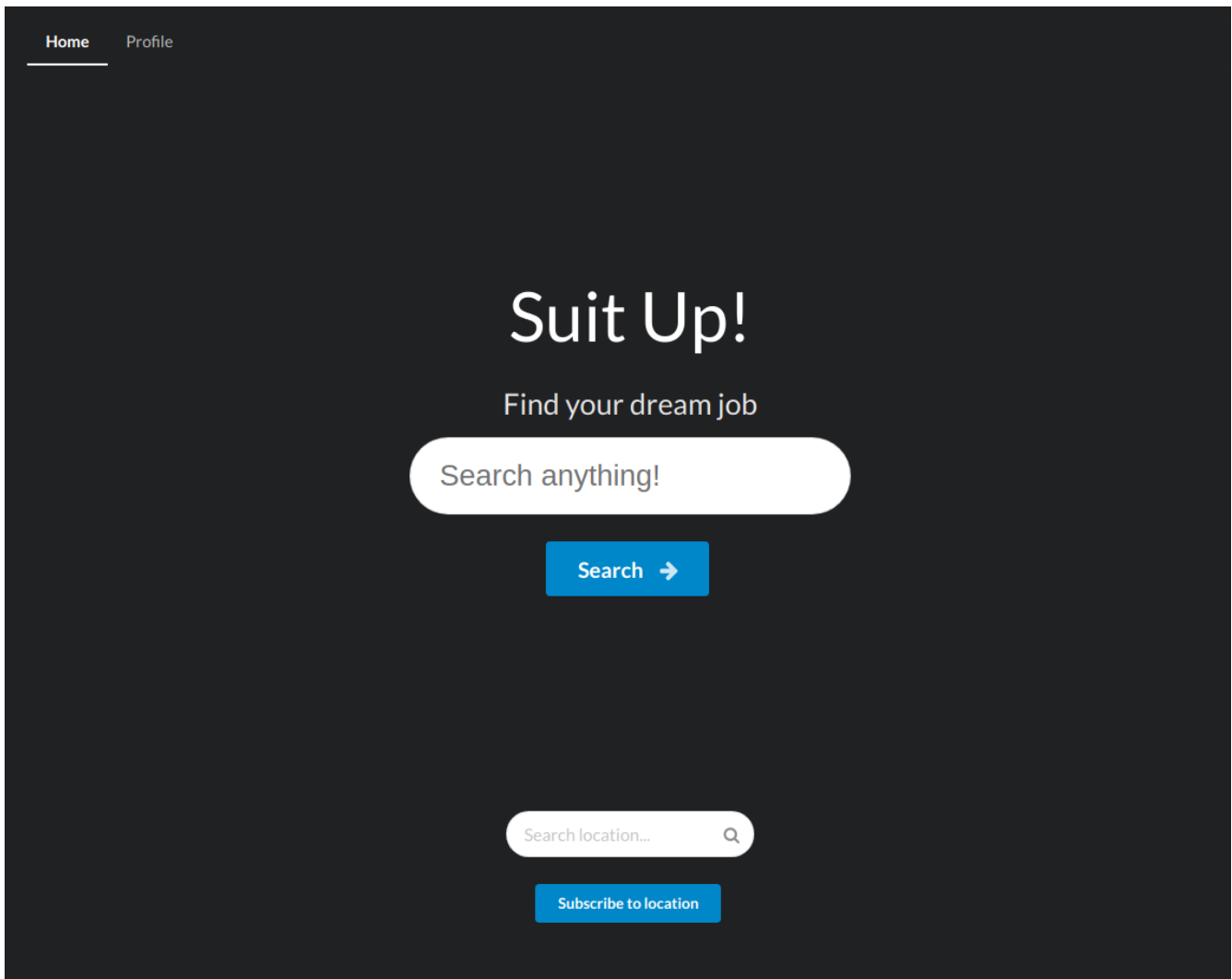


Figure 25: UI design of main page

This page contains a search bar and a location subscription option at the below. We are considering a full-text search for the search bar but it is not of concern at this stage. Location subscription query is as follows:

```
INSERT INTO Location_Sub (user_id, job_id, location)
SELECT DISTINCT @user_id, job_id, @location
FROM (Location_Sub AS LS
      NATURAL JOIN
      (SELECT id AS office_id, address AS addr FROM Office) AS O)
WHERE O.addr LIKE "%@location%";
```

## Event search result

The page below is a sample result page for an event search with the given query.

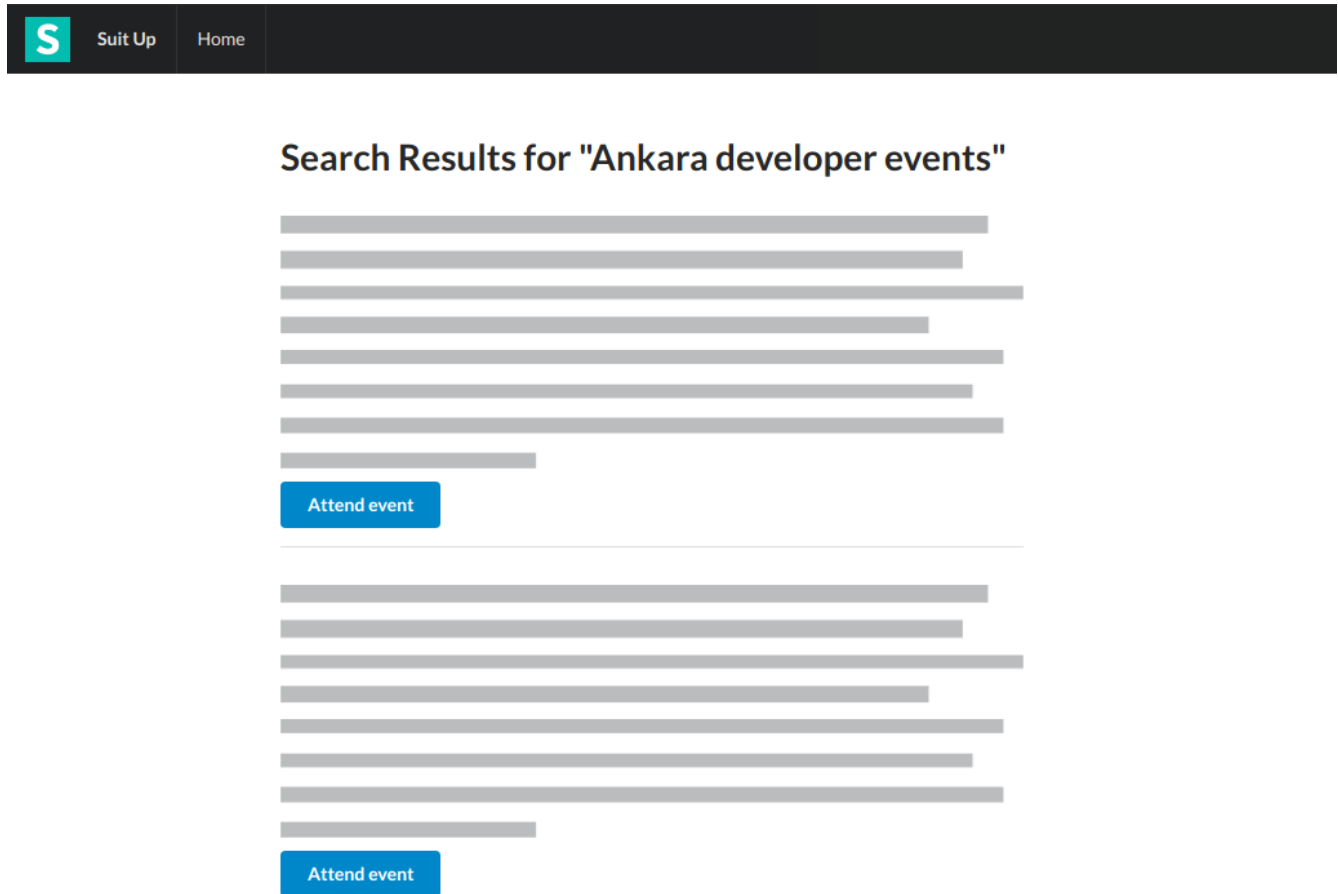


Figure 26: UI design of event search result

The event participation SQL query is given below.

```
INSERT INTO User_Event(user_id , event_id) VALUES (@user_id , @event_id);
```

## Office account panel

The office account's panel is as follows.

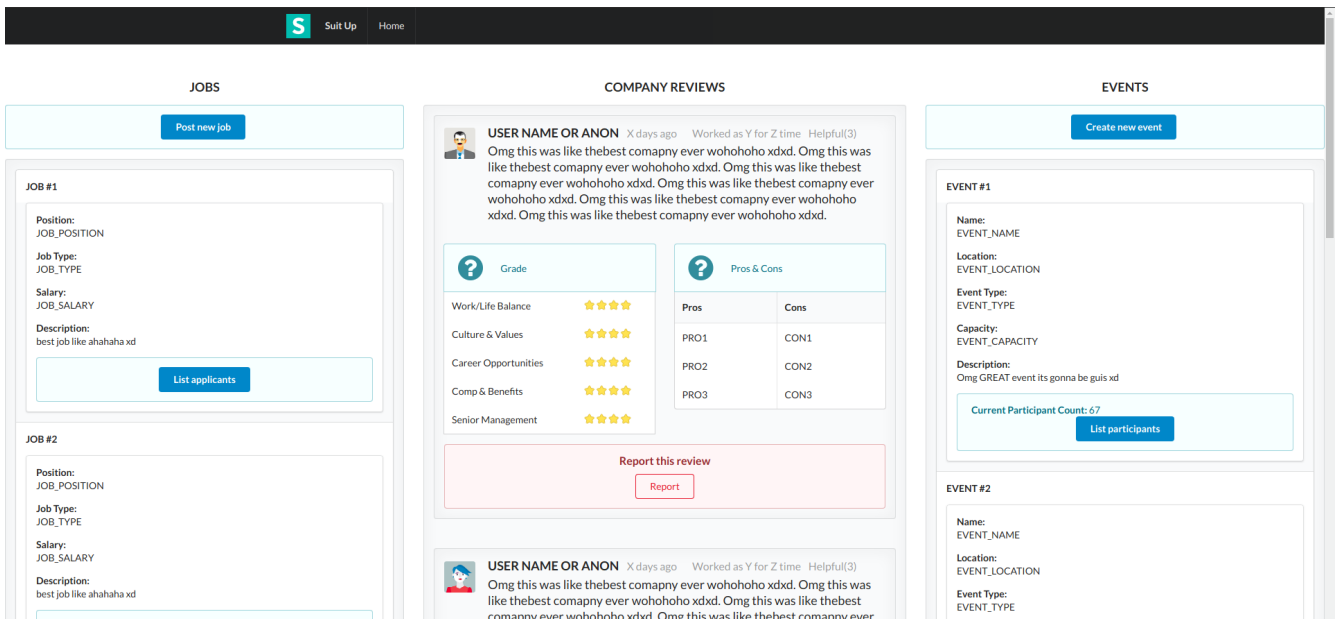


Figure 27: UI design of office account panel

In this page, office account can post jobs, view company reviews and report them, create new events, list participants of a particular event, and list applicants for a job offer which the account is responsible for. The SQL statements are as follows.

- Posting a job is a two-step execution. First, we insert the job into Job table, then we insert it into the Job\_Offers table. The queries are as follows. UI will be given later. ????

— insert job

```
INSERT INTO Job (description , salary , job_position , type)
VALUES (@description , @salary , @position , @type);
```

— insert job into offers table

```
INSERT INTO Job_Offer (job_id , office_id , creation_date , available)
VALUES (
  (SELECT MAX(id)
   FROM Job
   WHERE (description = @description
        AND salary = @salary
        AND position = @position
        AND type = @type) ) ,
  @office_id , NOW() , 1);
```



- Listing applicants for a specific job, UI will be given later

```
SELECT *
FROM ((SELECT user_id AS id , job_id as jid , progress AS prg FROM User_Apply)
      AS UA
      NATURAL JOIN
      User AS U)
WHERE UA.jid = @job_id
ORDER BY UA.prg DESC
```

- Listing all non-removed reviews for an office

```
SELECT (*)
FROM Review AS R
WHERE R.id IN
      (SELECT REV.rid
       FROM (id AS rid , office_id AS oid FROM Review) AS REV
       WHERE REV.oid = @office_id)
EXCEPT
      (SELECT REP.rid
       FROM (id as id , approved AS approved , office_account_id AS oaid , review_id
            AS rid FROM Report) AS REP
            NATURAL JOIN
            (id AS oid , acc_id , oaid FROM Office) AS O
       WHERE (0.oid = @office_id AND REP.approved = 2)
      )
);
```

- If the office account reports a review, they are given a pop-up to write a reasoning. The UI will be shown later, however the query is as below assuming the given description is stored as *@description*.

```
INSERT INTO Report(approved , description , creation_date , office_account_id ,
                  review_id)
VALUES (0 , @description , NOW() , @office_account_id , @review_id);
```

- Posting an event, UI will be given later

```
INSERT INTO Event(office_id , name , description , location , type , capacity ,
                 event_date , creation_date) VALUES (@office_id , @name , @description ,
                 @location , @type , @capacity , @event_date , NOW());
```

- Listing attendees for an event, UI will be given later

```
SELECT U.id , U.name, U.surname
FROM User AS U
      NATURAL JOIN
      (SELECT user_id AS id , event_id AS eid FROM User_Event) AS UE
WHERE UE.eid = @event_id;
```

- See number of current participants to an event

```
SELECT COUNT(*)
FROM User AS U
      NATURAL JOIN
      (SELECT user_id AS id , event_id AS eid FROM User_Event) AS UE
WHERE UE.eid = @event_id;
```

## List applicants in office account panel

When list of applicants for a job is requested, below pop up is shown.

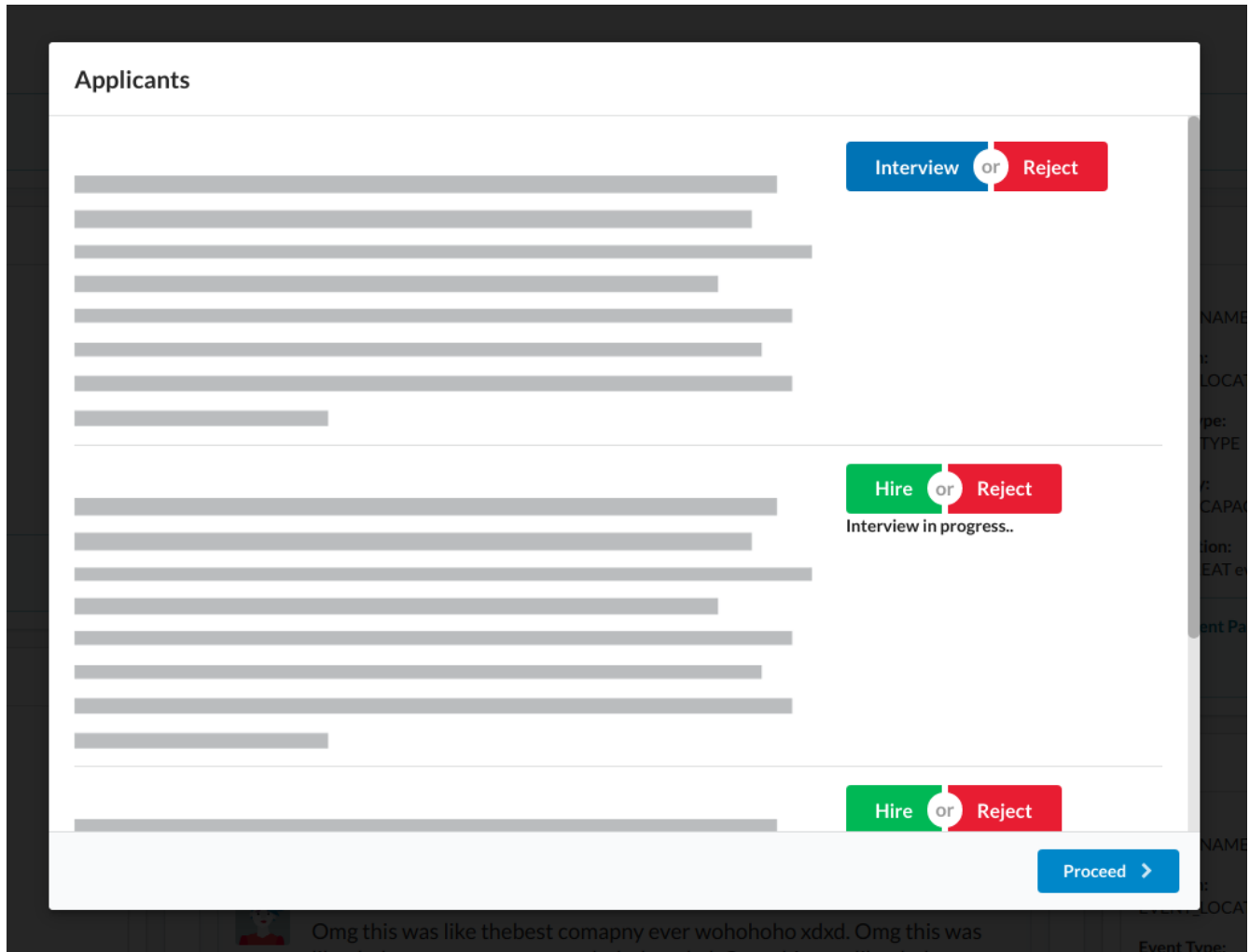


Figure 28: UI design of list applicants in office account panel

The query for listing applicants is given previously. When an application is submitted, we first give the options to interview and directly reject without interviewing. If interview is selected, new options become hiring and rejecting. The set of queries are given below.

```
-- decline application directly with no interview
-- inputs: @job_id, @user_id
UPDATE User_Apply
SET progress = -1
WHERE (job_id = @job_id
      AND user_id = @user_id );
```

```

— accept application for interview
— inputs: @job_id, @user_id
UPDATE User_Apply
SET progress = 1
WHERE (job_id = @job_id
      AND user_id = @user_id );

INSERT INTO Employment (office_id, user_id, job_id, interview, start_date, end_date
)
VALUES (@office_id, @user_id, @job_id, 1, CURRENT_DATE(), NULL);

— decline after interview
— inputs: @office_id, @job_id, @user_id
UPDATE Employment
SET end_date = CURRENT_DATE()
WHERE (office_id = @office_id AND
      job_id = @job_id AND
      user_id = @user_id AND
      interview = 1);

UPDATE User_Apply
SET progress = -2
WHERE (job_id = @job_id
      AND user_id = @user_id );

— Hire the employee
— inputs: @office_id, @job_id, @user_id
UPDATE User_Apply
SET progress = 2
WHERE (job_id = @job_id
      AND user_id = @user_id );

UPDATE Employment
SET end_date = CURRENT_DATE()
WHERE (office_id = @office_id AND
      job_id = @job_id AND
      user_id = @user_id AND

```

```
interview = 1);
```

```
INSERT INTO Employment (office_id, user_id, job_id, start_date, end_date, interview  
)  
VALUES (@office_id, @user_id, @job_id, CURRENT_DATE(), NULL, 0)
```

## List participants in office account panel

The pop-up for listing the participants for a specific event is given below.

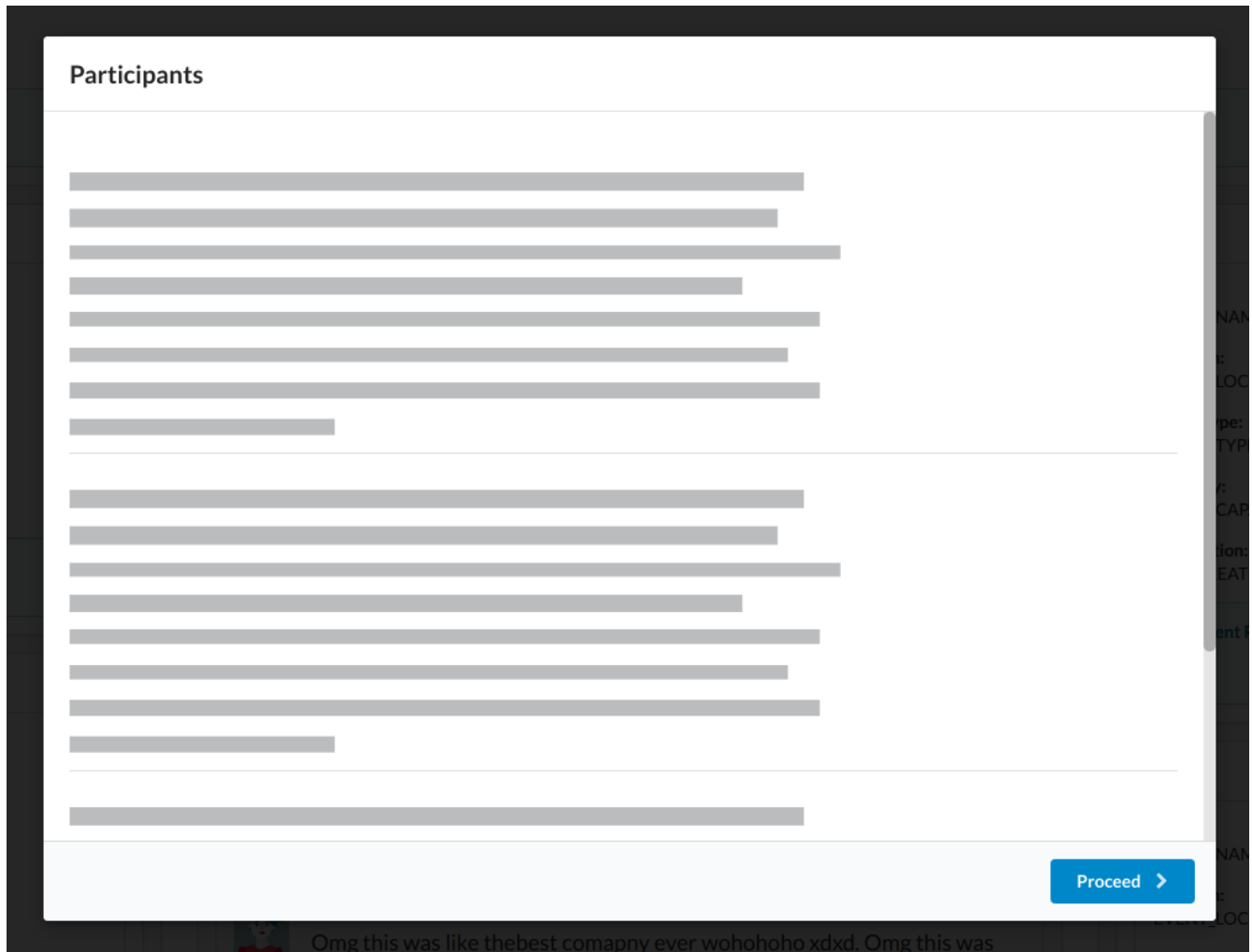
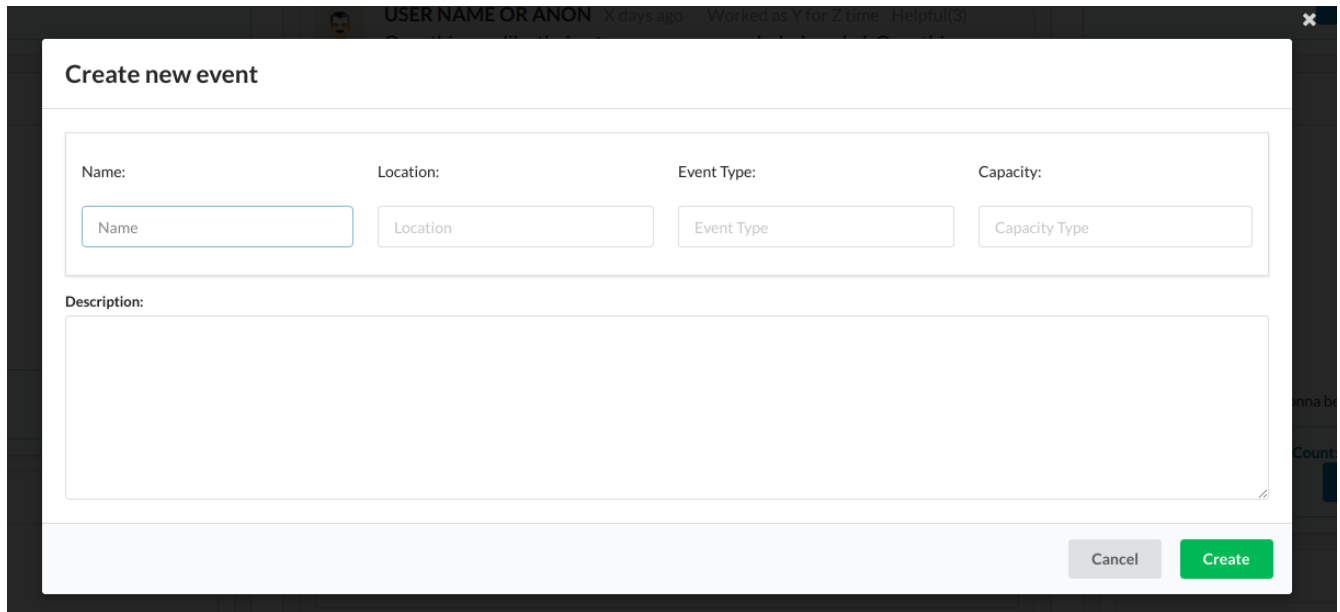


Figure 29: UI design of list participants in office account panel

The corresponding SQL statements are previously given.

## Create event in office account panel

The pop-up for posting an event is given below.



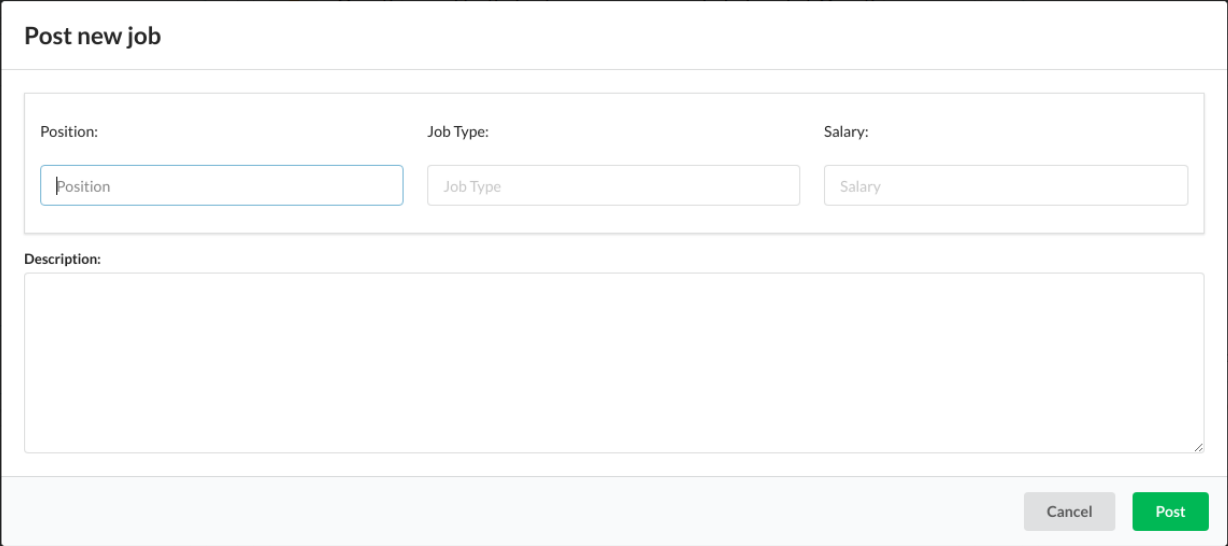
The image shows a web application interface for creating a new event. At the top, there is a dark header bar with a user profile icon, the text "USER NAME OR ANON", and a dropdown menu showing "x days ago", "Worked as Y for Z time", and "Helpful". Below the header, the main content area is titled "Create new event". It contains four input fields: "Name:", "Location:", "Event Type:", and "Capacity:". Each field has a placeholder text: "Name", "Location", "Event Type", and "Capacity Type" respectively. Below these fields is a large text area for "Description:". At the bottom right of the form, there are two buttons: "Cancel" and "Create".

Figure 30: UI design of create event in office account panel

The corresponding SQL statements are previously given.

## Post job in office account

The pop-up for posting a job offer is given below.



The image shows a dark-themed user interface with a white pop-up window titled "Post new job". At the top of the window, there is a header bar with a close button (X) on the right. Below the header, the form contains three input fields: "Position:", "Job Type:", and "Salary:". Each field has a placeholder text: "Position", "Job Type", and "Salary" respectively. Below these fields is a large text area labeled "Description:". At the bottom right of the form, there are two buttons: a grey "Cancel" button and a green "Post" button. The background of the application shows a user profile section with a placeholder for a profile picture, the text "USER NAME OR ANON", and some statistics like "X days ago", "Worked as Y for Z time", and "Helpful(3)".

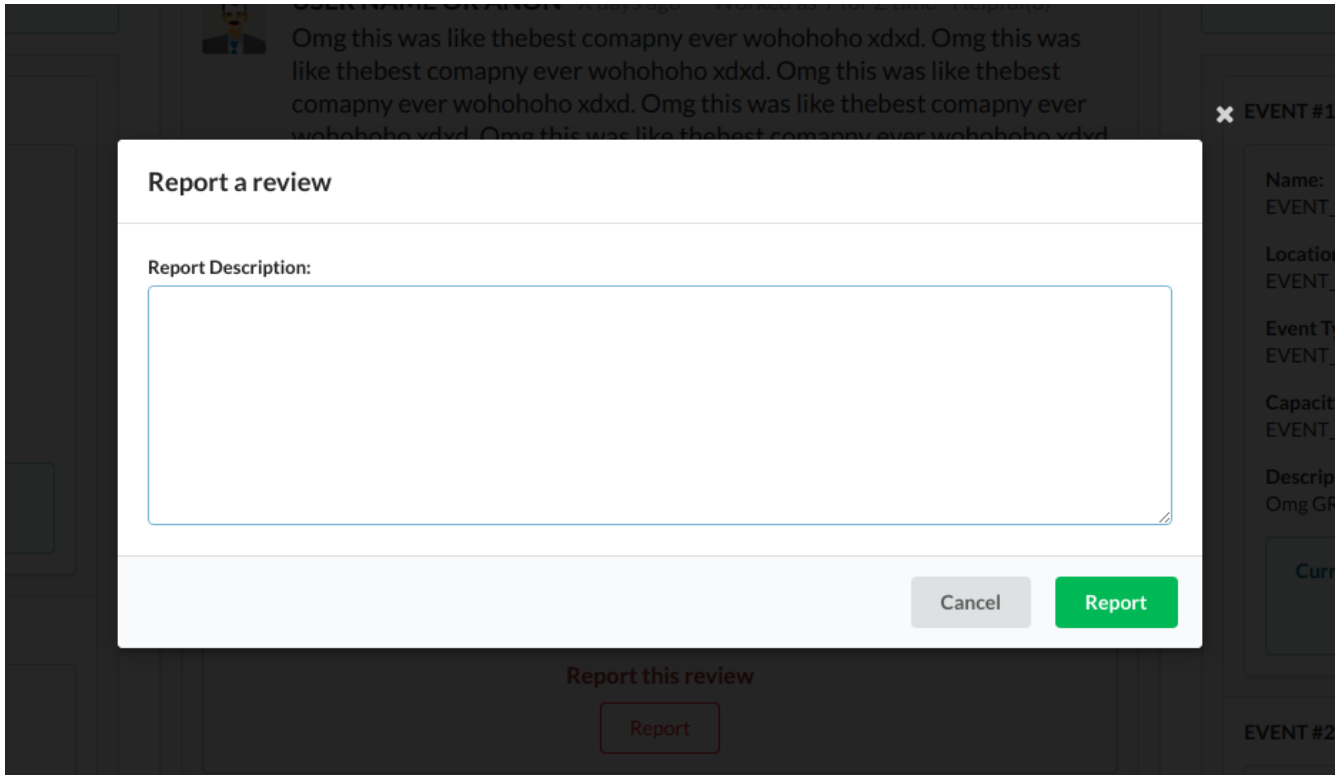
Figure 31: UI design of post job in office account

The corresponding SQL statements are previously given.



## Post report in office account

The pop-up for submitting a report is given below.



The image shows a dark-themed application interface with a white pop-up dialog titled "Report a review". The dialog has a "Report Description:" label above a large, empty text input area. At the bottom right of the dialog are two buttons: a grey "Cancel" button and a green "Report" button. In the background, a review is visible with the text "Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx." Below the review, there is a "Report this review" button and a "Report" button. On the right side of the background, there is a sidebar with event details for "EVENT #1" and "EVENT #2", including fields for Name, Location, Event Type, Capacity, and Description.

Figure 32: UI design of post report in office account

The corresponding SQL statements are previously given.

## Office page with job posts

The office specific page with the job posts tab on, is given below.

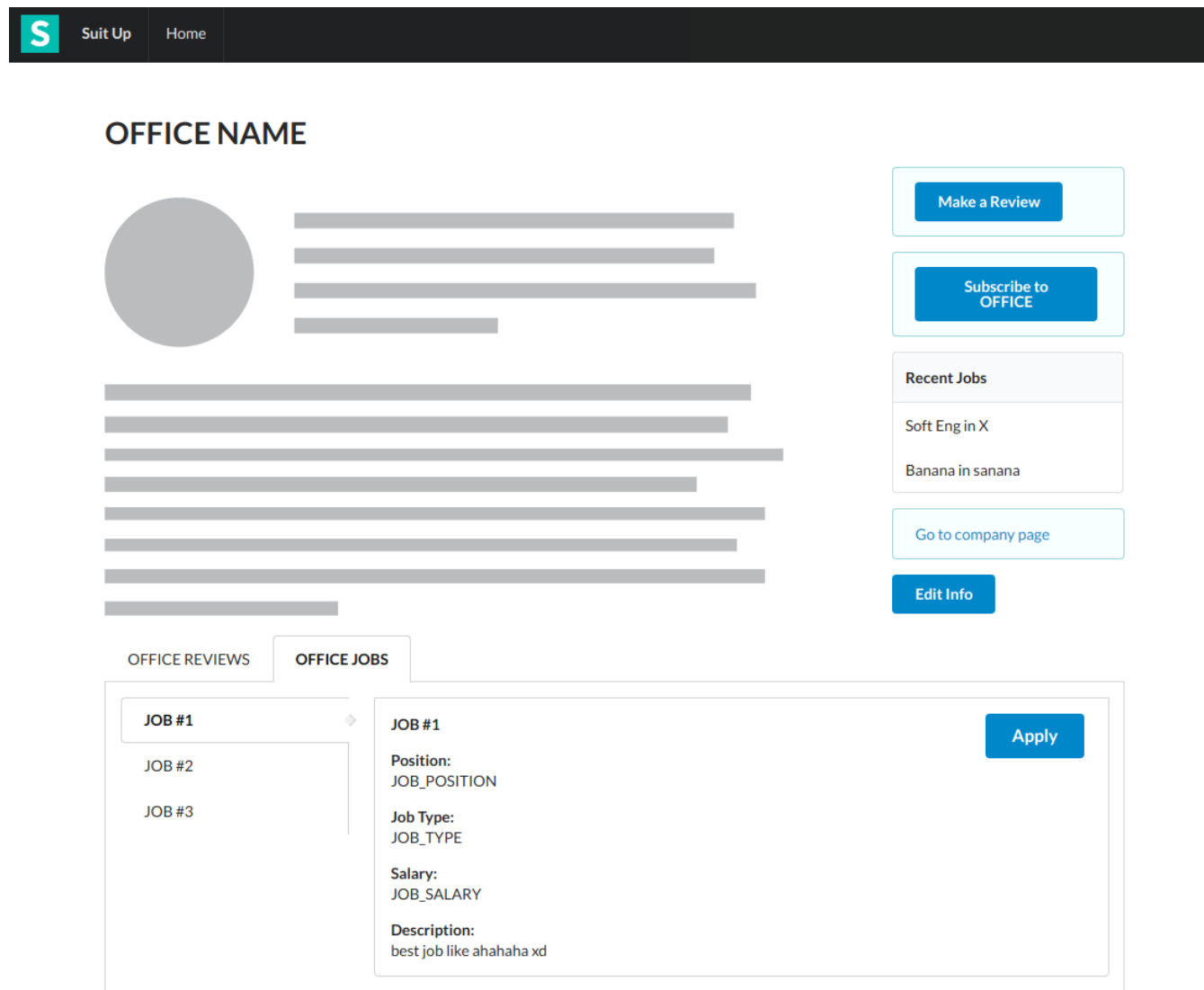


Figure 33: UI design of office page with job posts

The corresponding SQL statements for listing jobs are as follows.

— list jobs for a given office

```
SELECT JA.id
FROM Job_Applications AS JA
WHERE JA.office_id = @office_id;
```

## Office page with review and information

The office specific page with the reviews tab on, is given below.

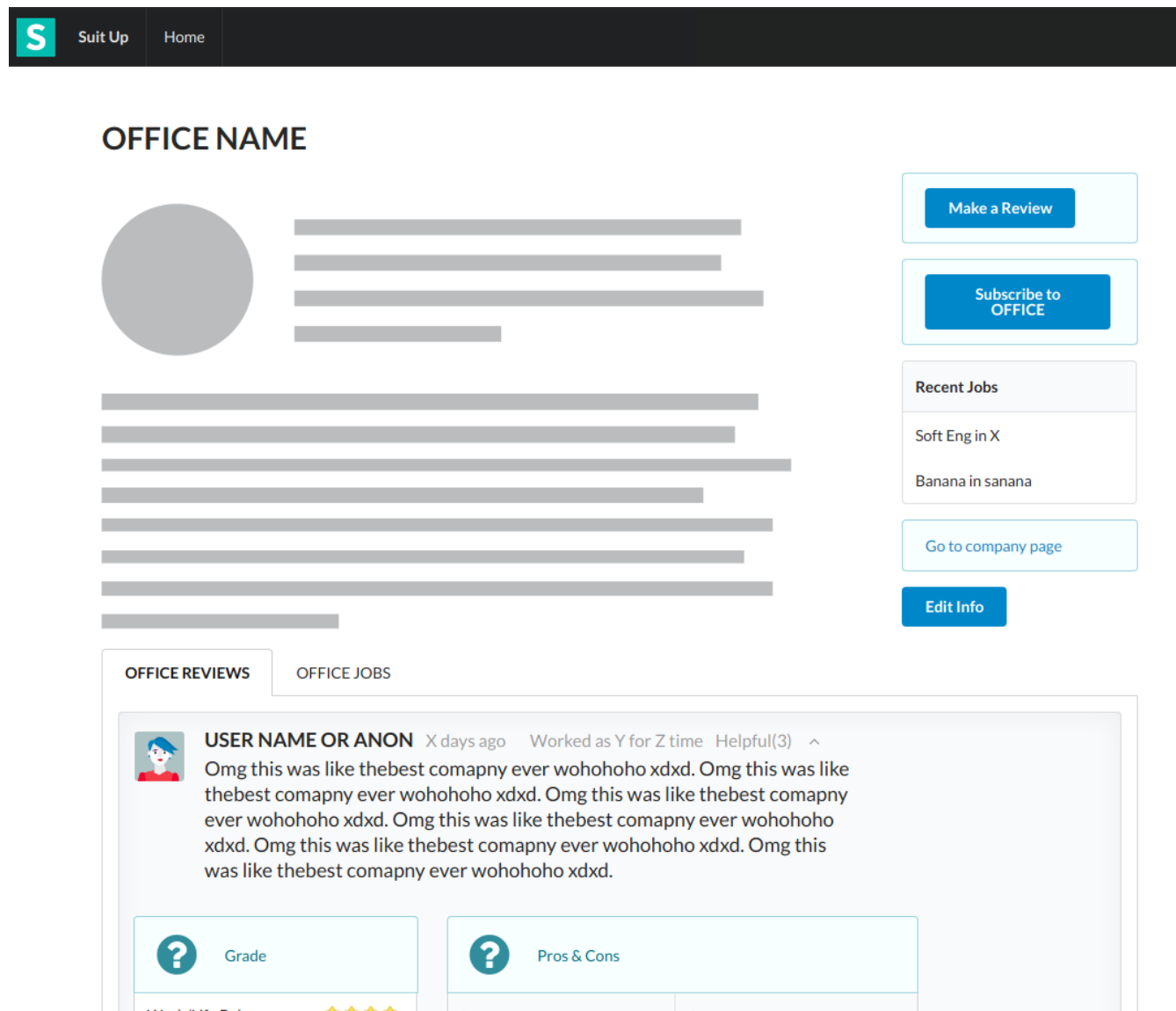


Figure 34: UI design of office page with review and information

We will only be giving the SQL to post a review since it is the only functionality in the scope of this page in this report.

The pop-up shown after pressing **Make a Review** is shown below.

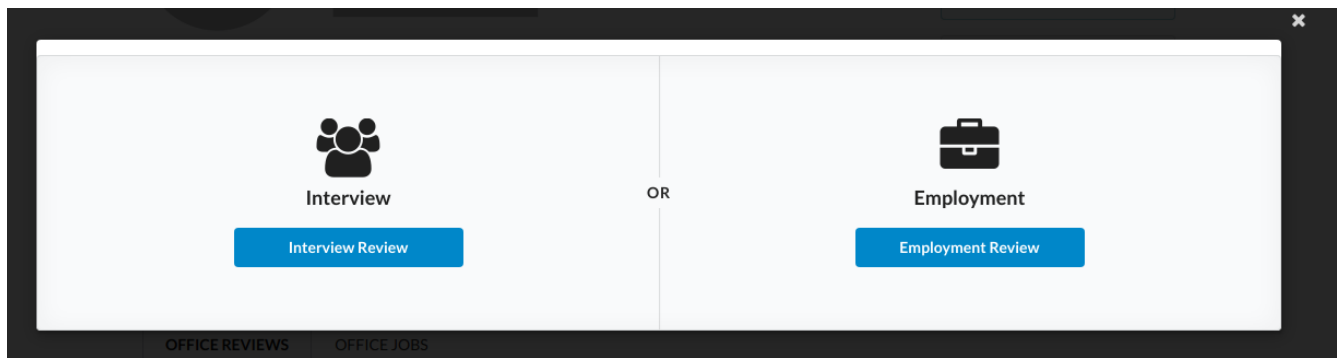


Figure 35: UI design of office page with review and information

The pop-up after pressing **Employment Review** is given below.

### Post employment review

Position:

Job Type:

Salary:

Position

Job Type

Salary

Review:

☐ Post anonymously

?

Grade

Work/Life Balance	★★★★★
Culture & Values	★★★★★
Career Opportunities	★★★★★
Comp & Benefits	★★★★★
Senior Management	★★★★★

?

Pros & Cons

Pros	Cons
PRO1 ▼	CON1 ▼
PRO1 ▼	PRO1 ▼
PRO1 ▼	PRO1 ▼

Cancel

Create

Figure 36: UI design of office page with review and information

The corresponding SQL statement is as follows. First, we need to insert the review into Review as we did with the Account table while registering.

```
INSERT INTO Review (user_id, job_id, office_id, interview, content, rating,
    helpful_count, creation_date, anonymity)
VALUES (@user_id, @job_id, @office_id, @interview, @content, @rating,
    @helpful_count, @creation_date, @anonymity);
```

Then, depending on the user's choice on the review type we will insert the review with the necessary information to the one of the child tables.

```
INSERT INTO Employee_Review
VALUES (@review_id, @rating1, @rating2, @rating3, @rating4);
```

```
INSERT INTO Interview_Review
VALUES (@review_id, @result, @rating1, @rating2, @rating3, @rating4);
```

The pop-up after pressing **Interview Review** is given below.

Post interview review

Result:

Accepted Offer

Position:

Position

Offered Salary:

Amount

Review:

☐ Post anonymously

Grade

Work/Life Balance

★★★★★

Culture & Values

★★★★★

Career Opportunities

★★★★★

Comp & Benefits

★★★★★

Senior Management

★★★★★

Pros & Cons

Pros	Cons
PRO1	CON1
PRO1	PRO1
PRO1	PRO1

Cancel

Create

Figure 37: UI design of office page with review and information

The corresponding SQL statement is given above.

# Company page information

The company specific page is given below.

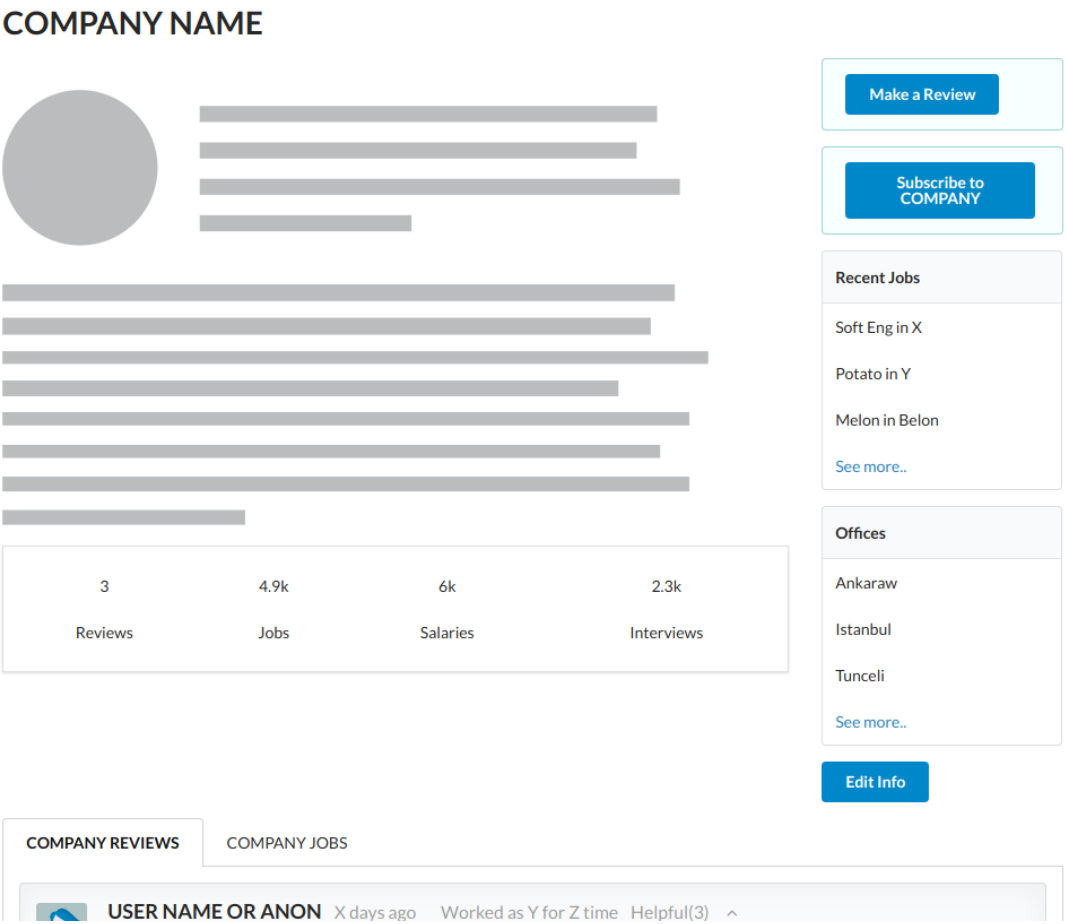


Figure 38: UI design of company page information

The queries for the functionality are not in the scope of this document.

## Reviews in company page

The UI for the reviews of a given company is shown below.

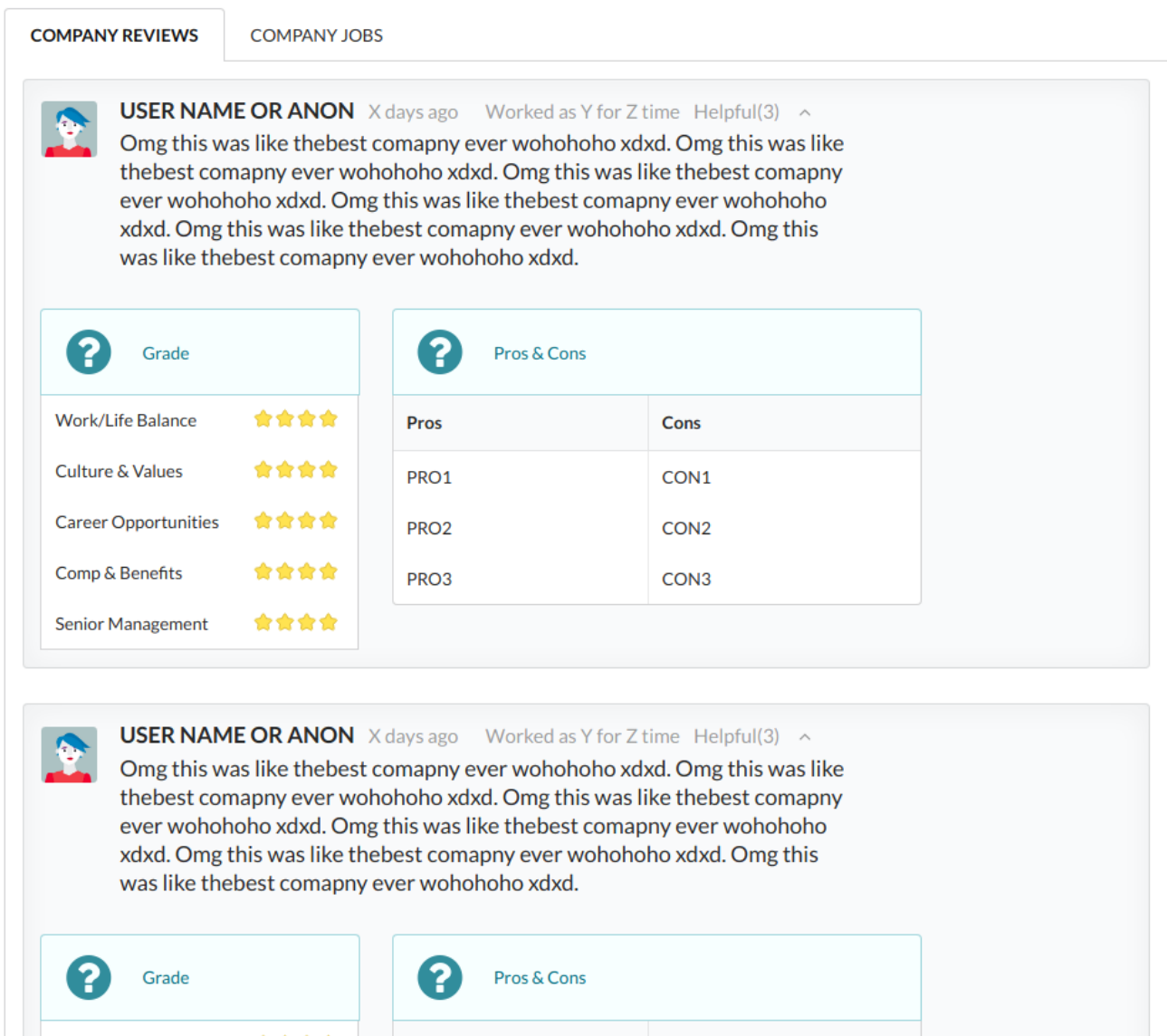


Figure 39: UI design of reviews in company page

The SQL statement is as follows.

```
SELECT (*)
FROM Review AS R
WHERE R.id IN (
  (
    SELECT REV.rid
    FROM (id AS oid, company_id AS cid FROM Office) AS O1
```



```

        NATURAL JOIN
        (id AS rid , office_id AS oid FROM Review) AS REV
WHERE O1.cid = @company_id
)
EXCEPT
(
SELECT REP.review_id
FROM (id as id , approved AS approved , office_account_id AS oaid , review_id AS
rid FROM Report) AS REP
        NATURAL JOIN
        (id AS oid , company_id AS cid , acc_id , oaid FROM Office) AS O2
WHERE (O2.cid = @company_id AND REP.approved = 2)
)
);

```

## Job posts in company page

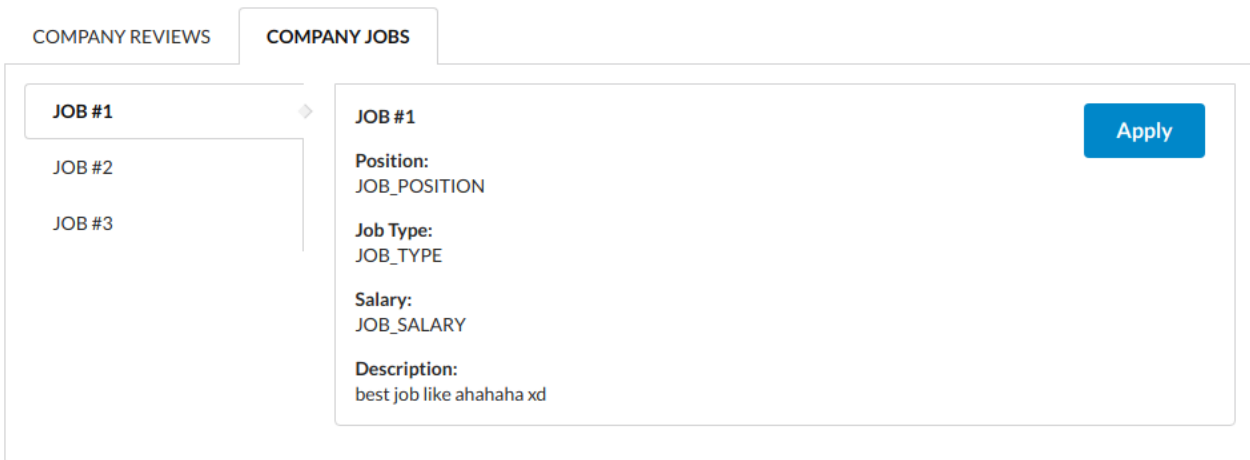


Figure 40: UI design of job posts in company page

The corresponding query for this functionality is given as follows.

— list jobs for a company

```
SELECT JA.id
FROM Job_Applications AS JA
WHERE JA.office_id IN
  (SELECT O.id
   FROM Office AS O
   WHERE O.company_id = @company_id);
```

## Editor panel


The editor panel is shown below.

S

Suit Up

Home

REPORTED REVIEWS



**USER NAME OR ANON** X days ago Worked as Y for Z time Helpful(3)

Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx. Omg this was like thebest comapny ever wohohoho xdx.

?

Grade

Work/Life Balance

★★★★★

Culture & Values

★★★★★

Career Opportunities

★★★★★

Comp & Benefits

★★★★★

Senior Management

★★★★★

?

Pros & Cons

Pros	Cons
PRO1	CON1
PRO2	CON2
PRO3	CON3

Report Description:

WOWOWWO this reviw is like sooooo offensive? omg\_? WOWOWWO this reviw is like sooooo offensive? omg\_? WOWOWWO this reviw is like sooooo offensive? omg\_? WOWOWWO this reviw is like sooooo offensive? omg\_?

Approve Removal

Decline Removal

Figure 41: UI design of editor panel

The query for listing the reviews for an editor is as follows.

```
SELECT (*)
FROM REPORT AS R
NATURAL JOIN
(editor_id AS eid, report_id AS id FROM Evaluate) AS E
WHERE (R.approved < 2 AND E.eid != @editor_id);
```

The query following the approval is as follows:

```
UPDATE Report
SET approved = CALL get_approved_count(@review_id) + 1
WHERE (review_id = @review_id);
```

```
INSERT INTO Evaluate (editor_id , report_id)
VALUES(@editor_id , @report_id);
```


The query following the denial is as follows:

```
UPDATE Report
SET approved = CALL get_approved_count(@review_id) - 1
WHERE (review_id = @review_id);
```

```
INSERT INTO Evaluate (editor_id , report_id)
VALUES(@editor_id , @report_id);
```

# User profile

Web page for the user profile of a specific user is shown below.



Recent Employments

Soft Eng in X

Banana in sanana

User Details

Name:	NAME_OF_USER
Surname:	SURNAME_OF_USER
E-mail:	EMAIL_OF_USER
Phone Number:	PHONENUMBER_OF USER
Address:	ADDRESS_OF_USER
Description:	DESCRIPTION_OF_USER

Edit Credentials

Figure 42: UI design of user profile

The queries are not given since the functionality is not in the scope of this document.

The past application part of the profile is given below. Edit profile button will only be shown to the owner of the account.

**User Details**

Name:	NAME_OF_USER
Surname:	SURNAME_OF_USER
E-mail:	EMAIL_OF_USER
Phone Number:	PHONENUMBER_OF USER
Address:	ADDRESS_OF_USER
Description:	DESCRIPTION_OF_USER

Edit Credentials

**Past Applications**

JOB #1:	In progress
JOB #2:	Hired
JOB #3:	Rejected

Figure 43: UI design of user profile

The corresponding SQL statements to list the past applications of a user along with the current status.

```

SELECT C.name, O.name, UA.job_id, UA.user_id, UA.progress
FROM (
  ((User_Apply AS UA NATURAL JOIN Job_Offers AS JO)
    NATURAL JOIN
    (SELECT id AS office_id, name AS name, company_id as company_id FROM Office)
  as O)
  JOIN
  ((SELECT id AS company_id, name AS name FROM Company) as C)
  ON C.company_id = O.company_id)
WHERE UA.user_id = @user_id;

```

### Edit profile pop up in user

User is shown the below pop-up if s/he clicks the edit profile button in the page above.



## 6 Advanced Database Components

### 6.1 Views

#### Get Non-Removed Reviews to List

```
CREATE VIEW reviews_to_list AS
SELECT *
FROM ((
    SELECT Rev.id
    FROM Review AS Rev
    EXCEPT(
        SELECT Rep.review_id
        FROM Report AS Rep
        WHERE (
            Rep.review_id = Rev.id
            AND
            Rep.approved = 2
        ))
    NATURAL JOIN Review Rev2);
```

#### Get Available Job Offers to List

```
CREATE VIEW available_jobs AS
SELECT *
FROM Job_Offers AS J
WHERE J.available = 1;
```

#### Get Available Events to List

```
CREATE VIEW available_events AS
SELECT *
FROM((
    SELECT U.event_id, COUNT(*) as cnt
    FROM User_Event as U
    GROUP BY U.event_id
    ) NATURAL JOIN EVENT E)
WHERE((
    E.capacity > cnt
    AND
```



```

        E.event_date >= CURRENT_DATE()
    ));

```

## 6.2 Reports

**Retrieve the ids of the users who applied the jobs posted in Ankara in last week for Software Engineer**

```

WITH Jobs(job_id) as ((
    SELECT job_id
    FROM Job_Offers AS JO NATURAL JOIN (
        SELECT id AS job_id, creation_date
        FROM Job AS J
    )
    WHERE JO.creation_date > CURRENT_DATE() - 7
    EXCEPT(
        SELECT job_id
        FROM Job_Offers AS JO1 NATURAL JOIN (
            SELECT id AS office_id, address
            FROM Office AS O
        )
        WHERE O.address NOT LIKE "%ANKARA%"))

SELECT user_id
FROM User_Apply AS UA
WHERE NOT EXISTS((
    SELECT J.job_id
    FROM Jobs as J
    EXCEPT(
        SELECT UA1.job_id
        FROM User_Apply as UA1
        WHERE(
            UA.user_id = UA1.user_id
            AND
            UA1.job_id = J.job_id
        ))));

```

**Retrieve the number of reviews posted about the office which has the biggest difference between the number of reports for anonymous and non-anonymous reviews**

```
WITH anonymous_count(office_account_id, cnt) as((
    SELECT office_account_id, COUNT(*) as cnt
    FROM Report AS Rep NATURAL JOIN Review AS Rev
    WHERE Rev.anonymity = 1
    GROUP BY Rep.office_account_id
))
```

```
WITH non_anonymous_count(office_account_id, cnt2) as((
    SELECT office_account_id, COUNT(*) as cnt2
    FROM Report AS Rep NATURAL JOIN Review AS Rev
    WHERE Rev.anonymity = 0
    GROUP BY Rep.office_account_id
))
```

```
WITH max_difference_office(office_account_id) as(
    SELECT office_account_id, MAX(ABS(anonymous_count.cnt - non_anonymous_count.
cnt2))
    FROM anonymous_count NATURAL JOIN non_anonymous_count
)
```

```
SELECT count(*) as max_count
FROM Review AS R
WHERE R.office_account_id = max_difference_office;
```

## 6.3 Triggers

Checking that an added job has type "Internship", "Full-time", or "Part-time" on insert

```
DELIMITER //
```

```
CREATE TRIGGER job_type_check AFTER INSERT ON Job
```

```
REFERENCING NEW ROW AS nrow
```

```
FOR EACH ROW
```

```
WHEN(nrow.type not in {'Internship', 'Full-time', 'Part-time'})
```

```
BEGIN
```

```
    ROLLBACK
```

```
END //
```

```
DELIMITER ;
```

If an office that is registered as headquarters is deleted then set the oldest added office as headquarters

```
DELIMITER //
```

```
CREATE TRIGGER update_headquarters AFTER DELETE ON Office
```

```
REFERENCING OLD ROW AS orow
```

```
FOR EACH ROW
```

```
WHEN orow.address = (SELECT C.headquarters
```

```
                     FROM Company AS C
```

```
                     WHERE orow.company_id = C.id)
```

```
BEGIN ATOMIC
```

```
    WITH oldest_office(office_id, created, company_id) AS(
```

```
        SELECT id AS office_id, min(creation_date) AS created
```

```
        FROM Office
```

```
    )
```

```
    UPDATE Company
```

```
    SET headquarters = (SELECT headquarters
```

```
                       FROM oldest_office
```

```
                       WHERE Company.id = oldest_office.office_id)
```

```
END //
```

```
DELIMITER ;
```

## 6.4 Constraints

Integrity constraints during the creation of table are stated in the following formats.

- not null
- primary key(A1, ..., An )
- foreign key (Am, ..., An ) references r

Therefore, there was no need for renaming.

## 6.5 Stored Procedure

```
DELIMITER //
```

```
CREATE PROCEDURE get_approved_count(IN review_id INT, OUT approved_count INT)
```

```
    BEGIN
```

```
        SELECT R.approved INTO approved_count
```

```
        FROM Review AS R
```

```
        WHERE R.id = review_id;
```

```
    END //
```

```
DELIMITER ;
```

```
DELIMITER //
```

```
CREATE PROCEDURE get_avg_rating_interview(IN review_id INT, OUT avg_rating NUMERIC
```

```
    (1,1))
```

```
    BEGIN
```

```
        SELECT AVG(R) INTO avg_rating
```

```
        FROM ((SELECT difficulty_rating , communication_rating , punctuality_rating ,
```

```
professionalism_rating FROM Interview_Review) AS R
```

```
        NATURAL JOIN
```

```
        Interview_Review AS IR)
```

```
        WHERE IR.id = review_id;
```

```
    END //
```

```
DELIMITER ;
```

## 7 Implementation Plan

We used Semantic UI Framework which is based on NodeJS and bootstrap in order to create the user interface of our web page. We will use JavaScript and PHP to provide the flow of events and animations in our web page. Our web site will be hosted on a Google Cloud Compute Engine with Ubuntu 18.04 and we will be using MySQL version 14.14 and distribution 5.7.25.

## References

- [1] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database system concepts*, 6th ed. New York: McGraw-Hill, 2010. [Online]. Available: <http://www.db-book.com/>
- [2] R. Elmasri and S. Navathe, *Fundamentals of database systems*. Addison-Wesley Reading, Mass, 2000.