Bilkent University

Department of Computer Engineering

CareerBridge

Project short-name: CB (Group Number: 11)

Design Report

Berk Çakar - 22003021

Atak Talay Yücel - 21901636

Kutay Tire - 22001787

Yiğit Yalın - 22002178

Instructor: Özgür Ulusoy
Teaching Assistant: Zülal Bingöl

Design Report

April 19, 2023

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Database Systems course CS353.

Contents

Design Report

Project name: CareerBridge

1 Revised E/R Model

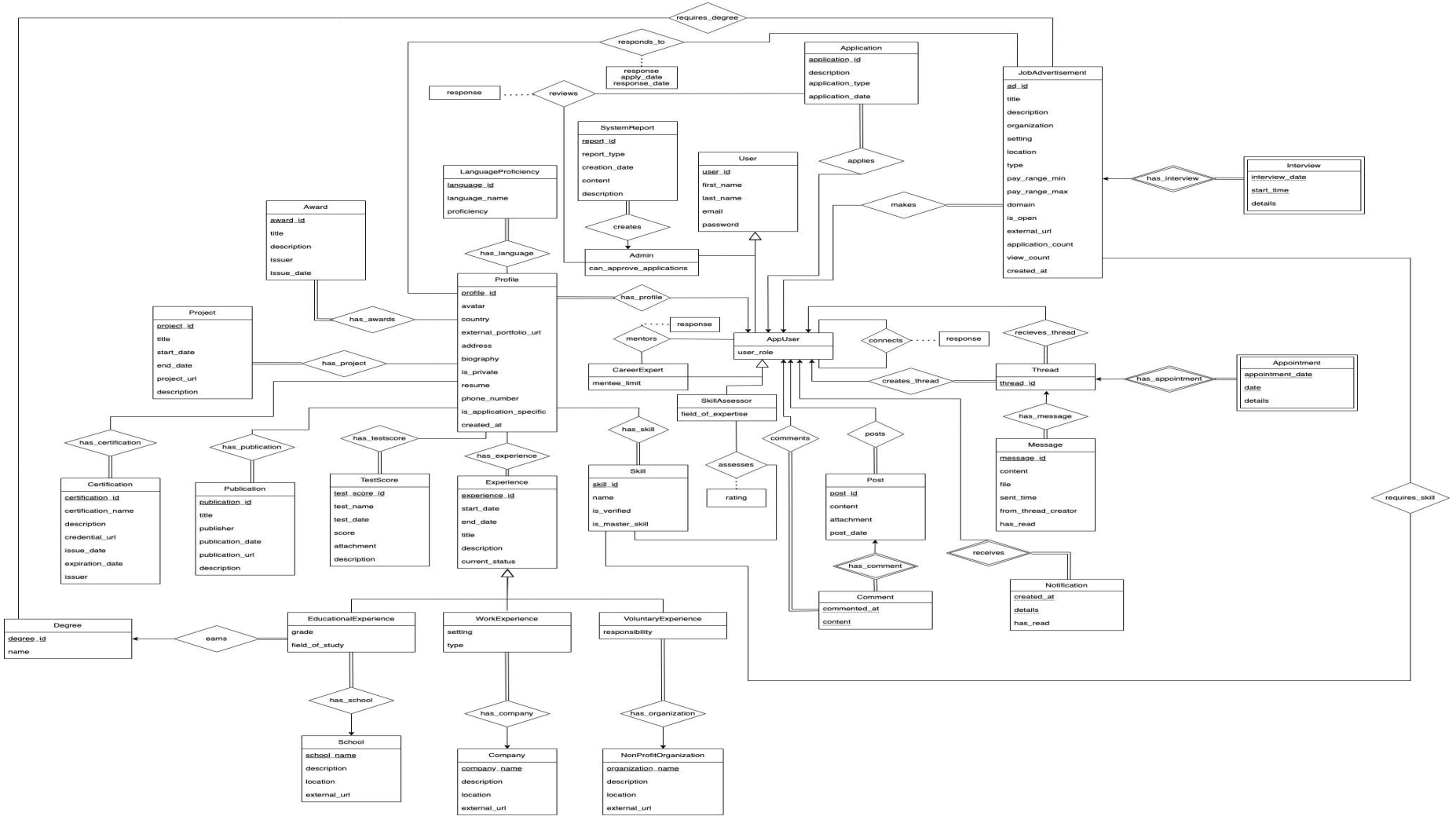

Figure 1: Revised Entity-Relationship Model of the CareerBridge Application

2 Relation Schemas

User
Relational Model: User(<u>user_id</u>, first_name, last_name, email, password)
Functional Dependencies: {(user_id → first_name, last_name, email, password)}
Candidate Keys: {(user_id)}
Primary Key: {(user_id)}
Foreign Keys: None
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS User (
user_id             INT NOT NULL AUTO_INCREMENT,
first_name          VARCHAR(50) NOT NULL,
last_name           VARCHAR(50) NOT NULL,
email               VARCHAR(50) NOT NULL,
password            VARCHAR(50) NOT NULL,
PRIMARY KEY (user_id),
UNIQUE (email)
);
```

AppUser
Relational Model: AppUser(<u>user_id</u>, user_role)
Functional Dependencies: {(user_id → user_role)}
Candidate Keys: {(user_id)}
Primary Key: {(user_id)}
Foreign Keys: user_id references User(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS AppUser(
user_id             INT NOT NULL,
user_role           VARCHAR(50) NOT NULL,
FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE
CASCADE,
PRIMARY KEY (user_id),
CHECK (user_role in ('Professional', 'Recruiter', 'Skill Assessor',
'Career Expert')
);
```

Admin
Relational Model: Admin(<u>user_id</u>, can_approve_applications)
Functional Dependencies: {(user_id → can_approve_applications)}
Candidate Keys: {(user_id)}
Primary Key: {(user_id)}
Foreign Keys: user_id references User(user_id)
Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Admin(
user_id                         INT NOT NULL
can_approve_applications        BOOLEAN NOT NULL,
FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE
CASCADE,
PRIMARY KEY (user_id)
);
```

SystemReport
Relational Model: SystemReport(report_id, report_type, creation_date, content, description, admin_id)
Functional Dependencies: {(report_id → report_type, creation_date, admin_id)}
Candidate Keys: {(report_id)}
Primary Key: {(report_id)}
Foreign Keys: admin_id references Admin(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS SystemReport(
report_id                       INT NOT NULL AUTO_INCREMENT,
report_type                     VARCHAR(25) NOT NULL,
content                         JSON NOT NULL,
description                     VARCHAR(250),
creation_date                   DATETIME NOT NULL,
admin_id                        INT NOT NULL,
FOREIGN KEY (admin_id) REFERENCES Admin(user_id),
PRIMARY KEY (report_id),
CHECK (report_type IN ('User Analytics', 'Ad Analytics');
```

Connection
Relational Model: Connection(sender_id, receiver_id, response)
Functional Dependencies: {(sender_id, receiver_id → response)}
Candidate Keys: {(sender_id, receiver_id)}
Primary Key: {(sender_id, receiver_id)}
Foreign Keys: sender_id references AppUser(user_id), receiver_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Connection(
sender_id                       INT NOT NULL,
receiver_id                     INT NOT NULL,
response                        BOOLEAN,
FOREIGN KEY (sender_id) REFERENCES AppUser(user_id) ON
DELETE CASCADE,
FOREIGN KEY (receiver_id) REFERENCES AppUser(user_id) ON
DELETE CASCADE
PRIMARY KEY (sender_id, receiver_id);
```

Application
Relational Model: Application(application_id, user_id, description, application_type, application_date)
Functional Dependencies: {(application_id → user_id, description, application_type, application_date)}
Candidate Keys: {(application_id)}
Primary Key: {(application_id)}
Foreign Keys: user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Application(
application_id          INT NOT NULL AUTO_INCREMENT,
user_id                 INT NOT NULL,
description             TEXT,
application_type        VARCHAR(25) NOT NULL,
application_date        DATETIME NOT NULL,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE
CASCADE,
PRIMARY KEY (application_id),
CHECK (application_type in ('Career Expert', 'Recruiter', 'Skill
Assessor'))
);
```

ApplicationReview
Relational Model: ApplicationReview(application_id, admin_id, response)
Functional Dependencies: {(application_id, admin_id → response)}
Candidate Keys: {(application_id, admin_id)}
Primary Key: {(application_id, admin_id)}
Foreign Keys: application_id references Application(application_id), admin_id references Admin(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS ApplicationReview(
application_id          INT NOT NULL,
admin_id                INT,
response                BOOLEAN,
FOREIGN KEY (application_id) REFERENCES
Application(application_id) ON DELETE CASCADE,
FOREIGN KEY (admin_id) REFERENCES Admin(user_id) ON DELETE
SET NULL,
PRIMARY KEY (application_id, admin_id)
);
```

Thread
Relational Model: Thread(thread_id, creator_id, receiver_id)
Functional Dependencies: {(thread_id → creator_id, receiver_id)}
Candidate Keys: {(thread_id)}
Primary Key: {(thread_id)}

Foreign Keys: creator_id references AppUser(user_id), receiver_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Thread(
thread_id                    INT NOT NULL AUTO_INCREMENT,
creator_id                   INT NOT NULL,
receiver_id                  INT NOT NULL,
FOREIGN KEY (creator_id) REFERENCES AppUser(user_id) ON
DELETE CASCADE,
FOREIGN KEY (receiver_id) REFERENCES AppUser(user_id) ON
DELETE CASCADE,
PRIMARY KEY (thread_id),
CONSTRAINT unique_participants UNIQUE(creator_id, receiver_id)
);
```

Appointment
Relational Model: Appointment(thread_id, date, details)
Functional Dependencies: {(thread_id, date→ details)}
Candidate Keys: {(thread_id, date)}
Primary Key: {(thread_id, date)}
Foreign Keys: thread_id references Thread(thread_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Appointment(
thread_id                    INT NOT NULL,
date                         DATETIME NOT NULL,
details                      TEXT,
FOREIGN KEY (thread_id) REFERENCES Thread(thread_id) ON
DELETE CASCADE,
PRIMARY KEY (thread_id, date)
);
```

Message
Relational Model: Message(message_id, thread_id, content, file, sent_time, from_thread_creator)
Functional Dependencies: {(message_id → thread_id, content, file, sent_time, from_thread_creator)}
Candidate Keys: {(message_id)}
Primary Key: {(message_id)}
Foreign Keys: thread_id references Thread(thread_id), sender_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Message(
message_id                   INT NOT NULL AUTO_INCREMENT,
thread_id                    INT NOT NULL,
```

```
content                 TEXT NOT NULL,
file                    BLOB,
sent_time               DATETIME NOT NULL,
has_read                BOOLEAN DEFAULT FALSE,
from_thread_creator     BOOLEAN NOT NULL,
FOREIGN KEY (thread_id) REFERENCES Thread(thread_id) ON
DELETE CASCADE,
PRIMARY KEY (message_id)
);
```

Notification
Relational Model: Notification(user_id, created_at, details)
Functional Dependencies: None
Candidate Keys: {(user_id, created_at, details)}
Primary Key: {(user_id, created_at, details)}
Foreign Keys: user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Notification(
user_id                 INT NOT NULL,
created_at              DATETIME NOT NULL,
details                 VARCHAR(1000) NOT NULL,
has_read                BOOLEAN DEFAULT FALSE,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE
CASCADE,
PRIMARY KEY (user_id, created_at, details)
);
```

Post
Relational Model: Post(post_id, user_id, content, attachment, post_date)
Functional Dependencies: {(post_id → user_id, content, attachment, post_date)}
Candidate Keys: {(post_id)}
Primary Key: {(post_id)}
Foreign Keys: user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Post(
post_id                 INT NOT NULL AUTO_INCREMENT,
user_id                 INT NOT NULL,
content                 VARCHAR(2048) NOT NULL,
attachment              BLOB,
post_date               DATETIME NOT NULL,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE
CASCADE,
PRIMARY KEY (post_id)
);
```

Comment
Relational Model: Comment(post_id, content, commented_at, user_id)
Functional Dependencies: {(post_id, content, commented_at → user_id)}
Candidate Keys: {(post_id, content, commented_at)}
Primary Key: {(post_id, content, commented_at)}
Foreign Keys: post_id references Post(post_id), user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Comment(
post_id                    INT NOT NULL,
user_id                    INT NOT NULL,
content                    VARCHAR(2048) NOT NULL,
commented_at               DATETIME NOT NULL,
FOREIGN KEY (post_id) REFERENCES Post(post_id) ON DELETE CASCADE,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE CASCADE,
PRIMARY KEY (post_id, content, commented_at)
);
```

JobAdvertisement
Relational Model: JobAdvertisement(ad_id, creator_id, title, description, organization, setting, location, type, pay_range_min, pay_range_max, domain, is_open, external_url, application_count, view_count, created_at)
Functional Dependencies: {(ad_id → creator_id, title, description, organization, setting, location, type, pay_range_min, pay_range_max, domain, is_open, external_url, application_count, view_count, created_at)}
Candidate Keys: {(ad_id)}
Primary Key: {(ad_id)}
Foreign Keys: creator_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS JobAdvertisement(
ad_id                      INT NOT NULL AUTO_INCREMENT,
creator_id                 INT NOT NULL,
title                      VARCHAR(50) NOT NULL,
description                TEXT NOT NULL,
organization               VARCHAR(100) NOT NULL,
setting                    VARCHAR(15) NOT NULL,
location                   VARCHAR(100) NOT NULL,
type                       VARCHAR(15) NOT NULL,
pay_range_min              INT NOT NULL,
pay_range_max              INT NOT NULL,
domain                     VARCHAR(50),
is_open                    BOOLEAN NOT NULL,
```

```
external_url            VARCHAR(1000),
application_count       INT NOT NULL DEFAULT 0,
view_count              INT NOT NULL DEFAULT 0,
created_at              DATETIME NOT NULL,
FOREIGN KEY (creator_id) REFERENCES AppUser(user_id) ON
DELETE CASCADE,
PRIMARY KEY (ad_id),
CHECK (setting IN ('On-site', 'Remote', 'Hybrid'),
CHECK (type IN ('Internship', 'Part-time Job', 'Job', 'Contract',
                'Project-based'))
);
```

JobAdvertisementResponse
Relational Model: JobAdvertisementResponse(profile_id, ad_id, response)
Functional Dependencies: {(user_id, ad_id → response)}
Candidate Keys: {(profile_id, ad_id)}
Primary Key: {(profile_id, ad_id)}
Foreign Keys: user_id references Profile(profile_id), ad_id references JobAdvertisement(ad_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS JobAdvertisementResponse(
profile_id              INT NOT NULL,
ad_id                   INT NOT NULL,
apply_date              DATETIME NOT NULL,
response_date           DATETIME,
response                VARCHAR(15),
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
FOREIGN KEY (ad_id) REFERENCES JobAdvertisement(ad_id) ON
DELETE CASCADE,
PRIMARY KEY (user_id, ad_id),
CHECK (response IN ('Waiting', 'Interview', 'Accepted', 'Rejected'))
);
```

Interview
Relational Model: Interview(ad_id, start_time, interview_date, details)
Functional Dependencies: {(ad_id, start_time, interview_date → details)}
Candidate Keys: {(ad_id, start_time, interview_date)}
Primary Key: {(ad_id, start_time, interview_date)}
Foreign Keys: ad_id references JobAdvertisement(ad_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Interview(
ad_id                   INT NOT NULL,
interview_date          DATETIME NOT NULL,
```

```
start_time                  DATETIME NOT NULL,
details                     TEXT,
FOREIGN KEY (ad_id) REFERENCES JobAdvertisement(ad_id) ON
DELETE SET NULL,
PRIMARY KEY (ad_id, start_time, interview_date)
);
```

Profile
Relational Model: Profile(profile_id, user_id, avatar, country, external_portfolio_url, address, biography, is_private, resume, phone_number, is_application_specific, created_at)
Functional Dependencies: {(profile_id → user_id, avatar, country, external_portfolio_url, address, biography, is_private, resume, phone_number, is_application_specific, created_at)}
Candidate Keys: {(profile_id)}
Primary Key: {(profile_id)}
Foreign Keys: user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Profile(
profile_id                  INT NOT NULL AUTO_INCREMENT,
user_id                     INT NOT NULL,
avatar                      BLOB,
country                     VARCHAR(50),
external_portfolio_url      VARCHAR(100),
address                     VARCHAR(300),
biography                   TEXT,
is_private                  BOOLEAN NOT NULL,
resume                      BLOB,
phone_number                VARCHAR(50),
is_application_specific     BOOLEAN NOT NULL,
created_at                  DATETIME NOT NULL,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE
CASCADE,
PRIMARY KEY (profile_id)
);
```

LanguageProficiency
Relational Model: LanguageProficiency(language_id, profile_id, language_name, proficiency)
Functional Dependencies: {(language_id → profile_id, language_name, proficiency)}
Candidate Keys: {(language_id)}
Primary Key: {(language_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS LanguageProficiency(
```

```
language_id                INT NOT NULL AUTO_INCREMENT,
profile_id                 INT NOT NULL,
language_name              VARCHAR(50) NOT NULL,
proficiency                VARCHAR(50),
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (language_id)
);
```

Award
Relational Model: Award(award_id, profile_id, title, description, issue_date, issuer)
Functional Dependencies: {(award_id → profile_id, title, description, issue_date, issuer)}
Candidate Keys: {(award_id)}
Primary Key: {(award_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Award(
award_id                   INT NOT NULL AUTO_INCREMENT,
profile_id                 INT NOT NULL,
title                      VARCHAR(50) NOT NULL,
description                VARCHAR(1000),
issue_date                 DATETIME,
issuer                     VARCHAR(100),
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (award_id)
);
```

Project
Relational Model: Project(project_id, profile_id, title, description, start_date, end_date, project_url)
Functional Dependencies: {(project_id → profile_id, title, description, start_date, end_date, project_url)}
Candidate Keys: {(project_id)}
Primary Key: {(project_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Project(
project_id                 INT NOT NULL AUTO_INCREMENT,
profile_id                 INT NOT NULL,
title                      VARCHAR(50) NOT NULL,
description                VARCHAR(1000),
start_date                 DATETIME,
end_date                   DATETIME,
```

```
project_url                    VARCHAR(100),
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (project_id)
);
```

Certification
Relational Model: Certification(certification_id, profile_id, certification_name, description, credential_url, issue_date, issuer, expiration_date)
Functional Dependencies: {(certification_id → profile_id, certification_name, description, credential_url, issue_date, issuer, expiration_date)}
Candidate Keys: {(certification_id)}
Primary Key: {(certification_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Certification(
certification_id          INT NOT NULL AUTO_INCREMENT,
profile_id                INT NOT NULL,
certification_name        VARCHAR(50) NOT NULL,
description               VARCHAR(1000),
credential_url            VARCHAR(100),
issue_date                DATETIME,
issuer                    VARCHAR(100),
expiration_date           DATETIME,
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (certification_id)
);
```

Publication
Relational Model: Publication(publication_id, profile_id, title, description, publication_date, publisher, publication_url)
Functional Dependencies: {(publication_id → profile_id, title, description, publication_date, publisher, publication_url)}
Candidate Keys: {(publication_id)}
Primary Key: {(publication_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Publication(
publication_id            INT NOT NULL AUTO_INCREMENT,
profile_id                INT NOT NULL,
title                     VARCHAR(50) NOT NULL,
description               VARCHAR(1000),
publication_date          DATETIME,
publisher                 VARCHAR(100),
```

```
publication_url                 VARCHAR(100),
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (publication_id)
);
```

TestScore
Relational Model: TestScore(test_score_id, profile_id, test_name, description, test_date, score, attachment)
Functional Dependencies: {(test_score_id → profile_id, test_name, description, test_date, score, attachment)}
Candidate Keys: {(test_score_id)}
Primary Key: {(test_score_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS TestScore(
test_score_id                   INT NOT NULL AUTO_INCREMENT,
profile_id                      INT NOT NULL,
test_name                       VARCHAR(50) NOT NULL,
description                     VARCHAR(1000),
test_date                       DATETIME,
score                           VARCHAR(50),
attachment                      BLOB,
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (test_score_id)
);
```

Skill
Relational Model: Skill(skill_id, profile_id, name, is_verified, is_master_skill)
Functional Dependencies: {(skill_id → profile_id, name, is_verified, is_master_skill)}
Candidate Keys: {(skill_id)}
Primary Key: {(skill_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Skill(
skill_id                        INT NOT NULL AUTO_INCREMENT,
profile_id                      INT NOT NULL,
name                            VARCHAR(50) NOT NULL,
is_verified                     BOOLEAN NOT NULL DEFAULT
FALSE,
is_master_skill                 BOOLEAN NOT NULL DEFAULT
FALSE,
```

```
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (skill_id)
);
```

SkillInJobAdvertisement
Relational Model: SkillInJobAdvertisement(skill_id, ad_id)
Functional Dependencies: None
Candidate Keys: {(skill_id, ad_id)}
Primary Key: {(skill_id, ad_id)}
Foreign Keys: skill_id references Skill(skill_id), ad_id references JobAdvertisement(ad_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS SkillInJobAdvertisement(
skill_id                    INT NOT NULL,
ad_id                       INT NOT NULL,
FOREIGN KEY (skill_id) REFERENCES Skill(skill_id) ON DELETE
CASCADE,
FOREIGN KEY (ad_id) REFERENCES JobAdvertisement(ad_id) ON
DELETE CASCADE,
PRIMARY KEY (skill_id, ad_id)
);
```

Experience
Relational Model: Experience(experience_id, profile_id, title, start_date, end_date, description, current_status)
Functional Dependencies: {(experience_id → profile_id, title, start_date, end_date, description, current_status)}
Candidate Keys: {(experience_id)}
Primary Key: {(experience_id)}
Foreign Keys: profile_id references Profile(profile_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Experience(
experience_id               INT NOT NULL AUTO_INCREMENT,
profile_id                  INT NOT NULL,
title                       VARCHAR(50) NOT NULL,
start_date                  DATETIME,
end_date                    DATETIME,
description                 VARCHAR(1000),
current_status              VARCHAR(25),
FOREIGN KEY (profile_id) REFERENCES Profile(profile_id) ON
DELETE CASCADE,
PRIMARY KEY (experience_id),
CHECK (current_status in ('Working', 'Past Working Experience'))
);
```

EducationalExperience
Relational Model: EducationalExperience(experience_id, grade, field_of_study, school_name, degree_id)
Functional Dependencies: {(experience_id → grade, field_of_study, school_name, degree_id)}
Candidate Keys: {(experience_id)}
Primary Key: {(experience_id)}
Foreign Keys: experience_id references Experience(experience_id), school_name references School(school_name), degree_id references Degree(degree_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS EducationalExperience(
experience_id                INT NOT NULL,
grade                        VARCHAR(50),
field_of_study               VARCHAR(50),
school_name                  VARCHAR(50) NOT NULL,
degree_id                    INT NOT NULL,
FOREIGN KEY (experience_id) REFERENCES
Experience(experience_id) ON DELETE CASCADE,
FOREIGN KEY (school_name) REFERENCES School(school_name)
ON DELETE CASCADE,
FOREIGN KEY (degree_id) REFERENCES Degree(degree_id) ON
DELETE CASCADE,
PRIMARY KEY (experience_id)
);
```

Degree
Relational Model: Degree(degree_id, name)
Functional Dependencies: {(degree_id → name)}
Candidate Keys: {(degree_id)}
Primary Key: {(degree_id)}
Foreign Keys: None
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Degree(
degree_id                    INT NOT NULL,
name                         VARCHAR(50),
PRIMARY KEY (degree_id)
);
```

DegreeInJobAdvertisement
Relational Model: DegreeInJobAdvertisement(degree_id, ad_id)
Functional Dependencies: None
Candidate Keys: {(degree_id, ad_id)}
Primary Key: {(degree_id, ad_id)}
Foreign Keys: degree_id references Degree(degree_id), ad_id references JobAdvertisement(ad_id)

Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS DegreeInJobAdvertisement(
degree_id                    INT NOT NULL,
ad_id                        INT NOT NULL,
FOREIGN KEY (degree_id) REFERENCES Degree(degree_id) ON
DELETE CASCADE,
FOREIGN KEY (ad_id) REFERENCES JobAdvertisement(ad_id) ON
DELETE CASCADE,
PRIMARY KEY (degree_id, ad_id)
);
```

School
Relational Model: School(school_name, description, location, external_url)
Functional Dependencies: {(school_name → description, location, external_url)}
Candidate Keys: {(school_name)}
Primary Key: {(school_name)}
Foreign Keys: None
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS School(
school_name                  VARCHAR(50) NOT NULL,
description                  TEXT,
location                     VARCHAR(100),
external_url                 VARCHAR(100),
PRIMARY KEY (school_name)
);
```

WorkExperience
Relational Model: WorkExperience(experience_id, setting, type, company_name)
Functional Dependencies: {(experience_id → setting, type, company_name)}
Candidate Keys: {(experience_id)}
Primary Key: {(experience_id)}
Foreign Keys: experience_id references Experience(experience_id), company_name references Company(company_name)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS WorkExperience (
experience_id                INT NOT NULL,
setting                      VARCHAR(25) NOT NULL,
type                         VARCHAR(15),
company_name                 VARCHAR(50) NOT NULL,
```

```
FOREIGN KEY (experience_id) REFERENCES
Experience(experience_id) ON DELETE CASCADE,
FOREIGN KEY (company_name) REFERENCES
Company(company_name) ON DELETE CASCADE,
PRIMARY KEY (experience_id),
CHECK (type IN ('Internship', 'Part-time Job', 'Job', 'Contract',
                'Project-based')),
CHECK (setting IN ('On-site', 'Remote', 'Hybrid')
);
```

Company
Relational Model: Company(company_name, description, location, external_url)
Functional Dependencies: {(company_name → description, location, external_url)}
Candidate Keys: {(company_name)}
Primary Key: {(company_name)}
Foreign Keys: None
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Company(
company_name            VARCHAR(50) NOT NULL,
description             VARCHAR(1000),
location                VARCHAR(100),
external_url            VARCHAR(100),
PRIMARY KEY (company_name)
);
```

VoluntaryExperience
Relational Model: VoluntaryExperience(experience_id, responsibility, organization_name)
Functional Dependencies: {(experience_id → responsibility, organization_name)}
Candidate Keys: {(experience_id)}
Primary Key: {(experience_id)}
Foreign Keys: experience_id references Experience(experience_id), organization_name references NonProfitOrganization(organization_name)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS VoluntaryExperience(
experience_id           INT NOT NULL,
responsibility          TEXT,
organization_name       VARCHAR(50) NOT NULL,
FOREIGN KEY (experience_id) REFERENCES
Experience(experience_id) ON DELETE CASCADE,
FOREIGN KEY (organization_name) REFERENCES
NonProfitOrganization(organization_name) ON DELETE CASCADE,
PRIMARY KEY (experience_id));
```

NonProfitOrganization
Relational Model: NonProfitOrganization(organization_name, description, location, external_url)
Functional Dependencies: {(organization_name → description, location, external_url)}
Candidate Keys: {(organization_name)}
Primary Key: {(organization_name)}
Foreign Keys: None
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS NonProfitOrganization(
organization_name          VARCHAR(50) NOT NULL,
description                VARCHAR(1000),
location                   VARCHAR(100),
external_url               VARCHAR(100),
PRIMARY KEY (organization_name)
);
```

SkillAssessor
Relational Model: SkillAssessor(user_id, field_of_expertise)
Functional Dependencies: {(user_id → field_of_expertise)}
Candidate Keys: {(user_id)}
Primary Key: {(user_id)}
Foreign Keys: user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS SkillAssessor(
user_id                    INT NOT NULL,
field_of_expertise         VARCHAR(50) NOT NULL,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE
CASCADE,
PRIMARY KEY (user_id)
);
```

SkillAssessment
Relational Model: SkillAssessment(skill_id, assessor_id, rating)
Functional Dependencies: {(skill_id, assessor_id → rating)}
Candidate Keys: {(skill_id, assessor_id)}
Primary Key: {(skill_id, assessor_id)}
Foreign Keys: skill_id references Skill(skill_id), assessor_id references SkillAssessor(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS SkillAssessment(
skill_id                   INT NOT NULL,
assessor_id                INT NOT NULL,
rating                     VARCHAR(50) NOT NULL,
```

```
FOREIGN KEY (skill_id) REFERENCES Skill(skill_id) ON DELETE
CASCADE,
FOREIGN KEY (assessor_id) REFERENCES SkillAssessor(user_id) ON
DELETE SET NULL,
PRIMARY KEY (skill_id, assessor_id)
);
```

CareerExpert
Relational Model: CareerExpert(user_id, mentee_limit)
Functional Dependencies: {(user_id → mentee_limit)}
Candidate Keys: {(user_id)}
Primary Key: {(user_id)}
Foreign Keys: user_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS CareerExpert(
user_id                    INT NOT NULL,
mentee_limit               INT,
FOREIGN KEY (user_id) REFERENCES AppUser(user_id) ON DELETE
CASCADE,
PRIMARY KEY (user_id)
);
```

Mentorship
Relational Model: Mentorship(mentor_id, mentee_id, response)
Functional Dependencies: {(mentor_id, mentee_id → response)}
Candidate Keys: {(mentor_id, mentee_id)}
Primary Key: {(mentor_id, mentee_id)}
Foreign Keys: mentor_id references CareerExpert(user_id), mentee_id references AppUser(user_id)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Mentorship(
mentor_id                  INT NOT NULL,
mentee_id                  INT NOT NULL,
response                   BOOLEAN,
FOREIGN KEY (mentor_id) REFERENCES CareerExpert(user_id) ON
DELETE CASCADE,
FOREIGN KEY (mentee_id) REFERENCES AppUser(user_id) ON
DELETE CASCADE,
PRIMARY KEY (mentor_id, mentee_id)
);
```

3 Advanced Database Components

3.1 Views

3.1.1 User Login View

```
CREATE VIEW UserLogin AS
SELECT u.user_id, u.first_name, u.last_name, u.email,
u.password
FROM User u
LEFT OUTER JOIN AppUser au ON u.user_id = au.user_id
LEFT OUTER JOIN Admin ad ON u.user_id = ad.user_id;
```

3.1.2 Get User Role View

```
CREATE VIEW GetUserRole AS
SELECT u.user_id, au.user_role,
       CASE WHEN ad.can_approve_applications IS NOT NULL THEN
1 ELSE 0 END AS is_admin
FROM User u
LEFT OUTER JOIN AppUser au ON u.user_id = au.user_id
LEFT OUTER JOIN Admin ad ON u.user_id = ad.user_id;
```

3.2 Triggers

3.2.1 Skill Assessment Notification Trigger

```
CREATE Trigger skillAssessmentNotificationTrigger
AFTER INSERT ON SkillAssesment
REFERENCING NEW ROW AS new_row
FOR EACH ROW
BEGIN
WITH user_skill AS (
       SELECT user_id, name
       FROM Skill
       WHERE skill_id = new_row.skill_id
)
INSERT INTO Notification (user_id, notification_date,
notification_content)
SELECT user_id, NOW(),
       CONCAT('You have been assessed a ',  new_row.rating, '
for ', name)
FROM user_skill;
END;
```

3.2.2 Comment Notification Trigger

```
CREATE Trigger commentNotificationTrigger
```

```
AFTER INSERT ON Comment
REFERENCING NEW ROW AS new_row
FOR EACH ROW
BEGIN
WITH post_temp AS (
	SELECT
		CONCAT(U.first_name, " ", U.last_name) AS
commentor_name,
		P.user_id as post_user_id
	FROM User as U, Post as P
	WHERE U.user_id = new_row.user_id
		AND P.post_id = new_row.post_id
)
INSERT INTO Notification (user_id, notification_date,
notification_content)
SELECT post_user_id, NOW(),
	CONCAT(commentor_name, ' commented on your post')
FROM post_temp
END;
```

3.2.3 Application Notification Trigger

```
CREATE TRIGGER applicationNotificationTrigger
AFTER INSERT ON ApplicationReview
REFERENCING NEW ROW AS new_row
FOR EACH ROW
BEGIN
WITH application_temp AS (
	SELECT A.user_id, A.application_type,
		CONCAT(U.first_name, " ", U.last_name) AS
admin_name
	FROM Application as A, User as U
	WHERE application_id = new_row.application_id
		AND U.user_id = new_row.admin_id
)
INSERT INTO Notification (user_id, notification_date,
notification_content)
SELECT user_id, NOW(),
	CASE WHEN new_row.response IS TRUE THEN CONCAT(
		'Your ', application_type, ' application has
been approved by ',
		admin_name
		)
	ELSE CONCAT(
		'Your ', application_type, ' application has
been rejected by ',
		admin_name
		)
	END
FROM application_temp;
```

```
END;
```

3.2.4 Connection Request Notification Trigger

```
CREATE TRIGGER connectionRequestNotificationTrigger
AFTER INSERT ON Connection
REFERENCING NEW ROW AS new_row
FOR EACH ROW
BEGIN
WITH connection_temp AS (
       SELECT
              CONCAT(U.first_name, " ", U.last_name) AS
requester_name,
       FROM
              User AS U
       WHERE
              U.user_id = new_row.sender_id
)
INSERT INTO Notification (user_id, notification_date,
notification_content)
SELECT new_row.receiver_id, NOW(),
       CONCAT(requester_name, ' sent you a connection
request')
FROM connection_temp;
END;
```

3.2.5 Connection Response Notification Trigger

```
CREATE TRIGGER connectionResponseNotificationTrigger
AFTER UPDATE ON Connection REFERENCING NEW ROW AS new_row
FOR EACH ROW
BEGIN
WITH connection_temp AS (
       SELECT
              CONCAT(U.first_name, " ", U.last_name) AS
respondent_name
       FROM User AS U
       WHERE U.user_id = new_row.receiver_id
)
INSERT INTO Notification (user_id, notification_date,
notification_content)
SELECT new_row.sender_id, NOW(),
       CASE WHEN new_row.response IS TRUE
              THEN CONCAT(requester_name,
                     ' accepted your connection request'
                     )
              ELSE CONCAT( requester_name,
                     ' rejected your connection request'
                     )
```

```
       END
FROM connection_temp;
END;
```

3.2.6 Add User to AppUser Trigger

```
CREATE TRIGGER add_user_to_appuser
AFTER INSERT ON User
FOR EACH ROW
BEGIN
    INSERT INTO AppUser (user_id, user_role)
    VALUES (NEW.user_id, 'Professional');
END;
```

3.3 Assertions

3.3.1 Check if Each User Has at Most One Master Skill

```
CREATE ASSERTION num_of_master_skill
CHECK (1 >= all(SELECT COUNT(*)
                FROM Skill S, Profile P
                WHERE S.profile_id = P.profile_id AND
                S.is_master_skill = 1);
```

3.3.2 Check if Number of Mentees of Each CareerExpert Does Not Exceed Mentee Limit

```
CREATE ASSERTION num_of_mentees CHECK (
  NOT EXISTS (
    SELECT
      mentor_id,
      COUNT(DISTINCT mentee_id)
    FROM
      Mentorship
    HAVING
      COUNT(DISTINCT mentee_id) > (
        SELECT
          mentee_limit
        FROM
          CareerExpert
        WHERE
          user_id = mentor_id
      )
    GROUP BY
      mentor_id
  );
```

3.3.3 Check if the Receiver and Creator of a Thread are not the Same Person

```
CREATE ASSERTION thread_distinct_person CHECK (
  NOT EXISTS(
    SELECT
      thread_id
    FROM
      Thread
    WHERE
      creator_id = receiver_id
  )
);
```

3.4 Procedures

3.4.1 Procedure for Increasing the Application Count of an Advertisement by One

```
CREATE PROCEDURE increase_application_count(IN ad_id_param
INT)
BEGIN
    UPDATE JobAdvertisement
    SET application_count = application_count + 1
    WHERE ad_id = ad_id_param;
END;
```

3.4.2 Procedure for Increasing the View Count of an Advertisement by One

```
CREATE PROCEDURE increase_view_count(IN ad_id_param INT)
BEGIN
    UPDATE JobAdvertisement
    SET view_count = view_count + 1
    WHERE ad_id = ad_id_param;
END;
```

4 User Interface Design and SQL Statements

4.1 Login Page

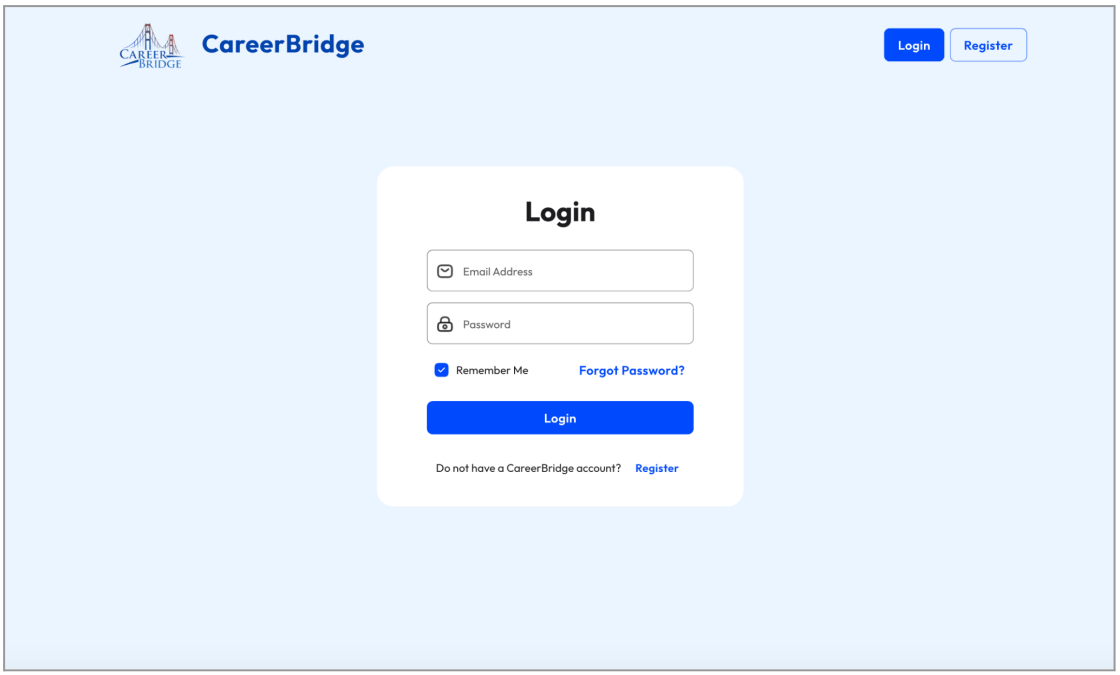

Figure 2

Inputs: @email, @password

On Login button pressed:

```
SELECT
  user_id,
  first_name,
  last_name,
  user_role,
  can_approve_applications,
  is_admin
FROM
  UserLogin
WHERE
  email = @email
  AND password = @password;
```

If the query above returns successfully, then the result of this query will be saved as a cookie to ease the future queries (i.e., get profile data using user_id). Otherwise, an error message will be displayed to the user.

```
SELECT user_role, is_admin
FROM GetUserRole
WHERE user_id = @user_id;
```

Also, we have another view called GetUserRole. The user_role and is_admin fields from this view will be gathered using the saved user_id and they will be used to display different pages for different types of users (for example the post job advertisement screen for recruiters or the admin panel for admins). Hence, we do not need separate login screens for different user roles. We do not save the role information as a cookie to prevent manipulation and increase the security when rendering the pages.

4.2 Register Page

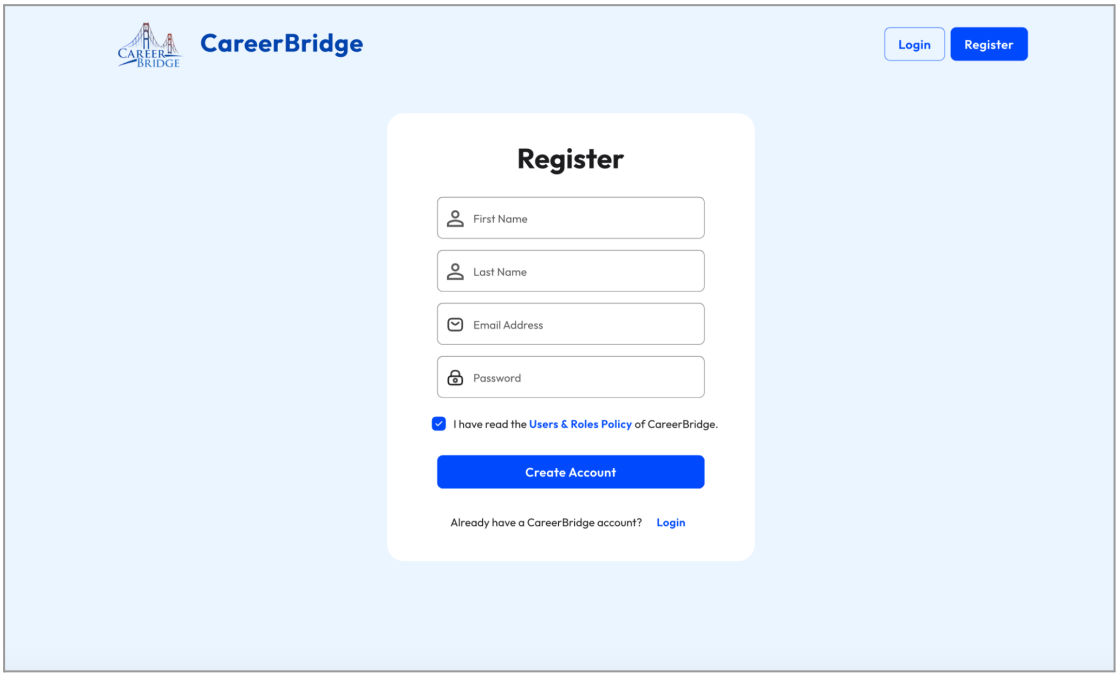

Figure 3

Inputs: @first_name, @last_name, @email, @password

On Create Account button pressed:

```
INSERT INTO User (
  first_name, last_name, email, password
)
VALUES (@first_name, @last_name, @email, @password);
```

If the inputs are valid, the user will be notified that the account is created and they will be redirected to the login page (Figure 2).

4.3 Policy Page

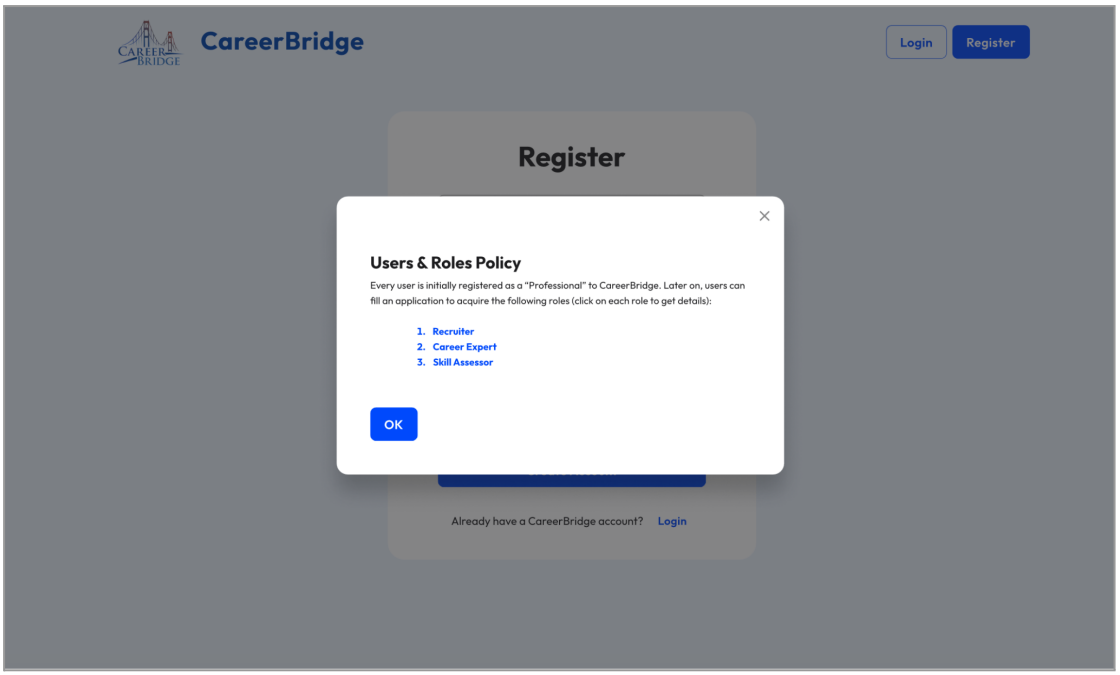

Figure 4

There are no related SQL queries for this page. It is embedded into HTML and the purpose is to notify the users that they will be signed up with the "Professional" role. We created a trigger to assign every user to the "Professional" role after registration.

4.4 Forgot Password Page

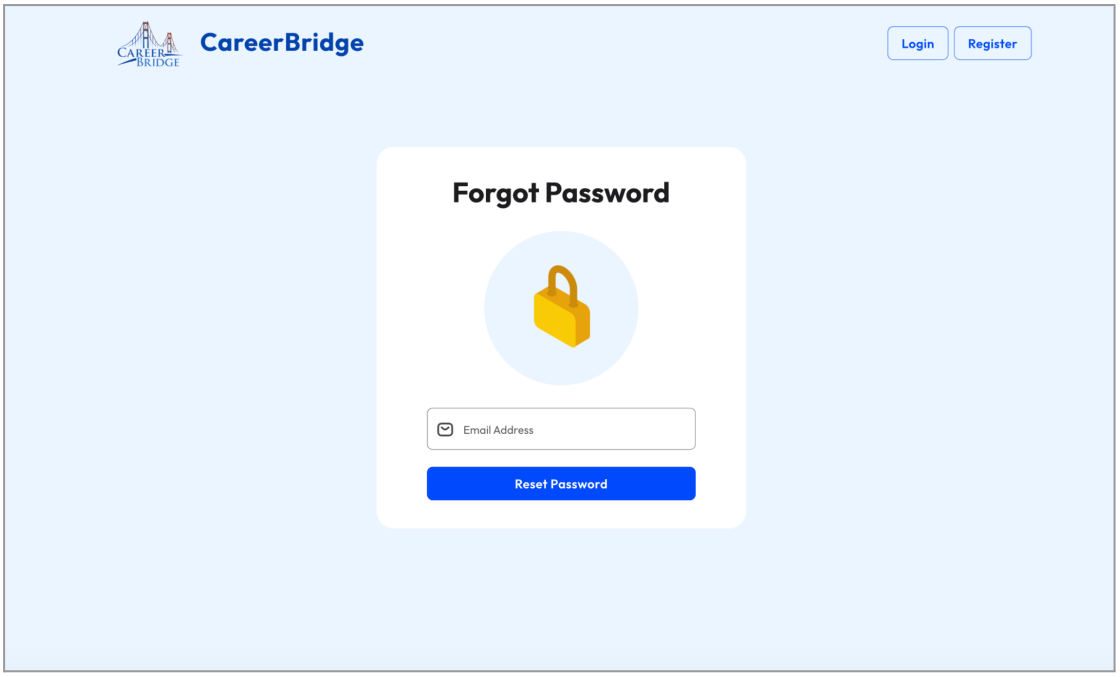

Figure 5

Inputs: @email

First of all, we check if the user is registered in the system.

```
SELECT EXISTS(SELECT * FROM User WHERE email = @email) AS
email_exists;
```

If this query returns 0, we abort and print an error message to screen, indicating that the email is not registered. Otherwise, we send a link to the user via email which will trigger:

```
SET @new_password = CONCAT(SUBSTRING(MD5(RAND()) FROM 1 FOR
8));

UPDATE User
SET password = @new_password
WHERE email = @email;
```

Then, we expose the @new_password to the user. They can change it afterwards from the user settings, if they wish.

```
SELECT @new_password AS new_password;
```

4.5 Browse Ads Page

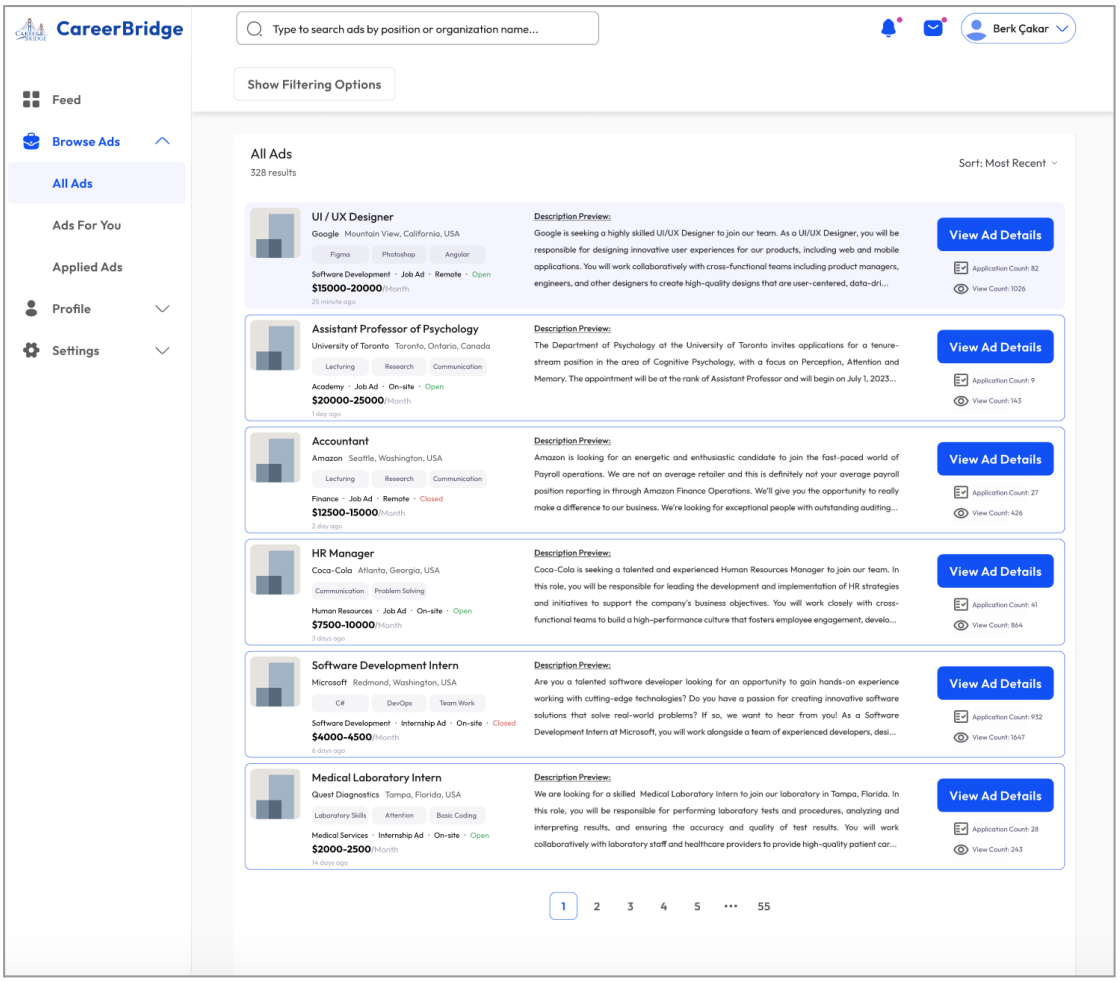

Figure 6

When a user enters to the All Ads page under Browse Ads tab, all available ads will be displayed with the following query:

```
SELECT * FROM JobAdvertisement;
```

If the "Sort Most Recent" or a similar sorting option is selected then the following query will be executed instead, considering the "Sort Most Recent" case.

```
SELECT *
FROM JobAdvertisement
ORDER BY created_at DESC;
```

4.6 Browse Ads Page with Filtering Options

Scenario 1.1:

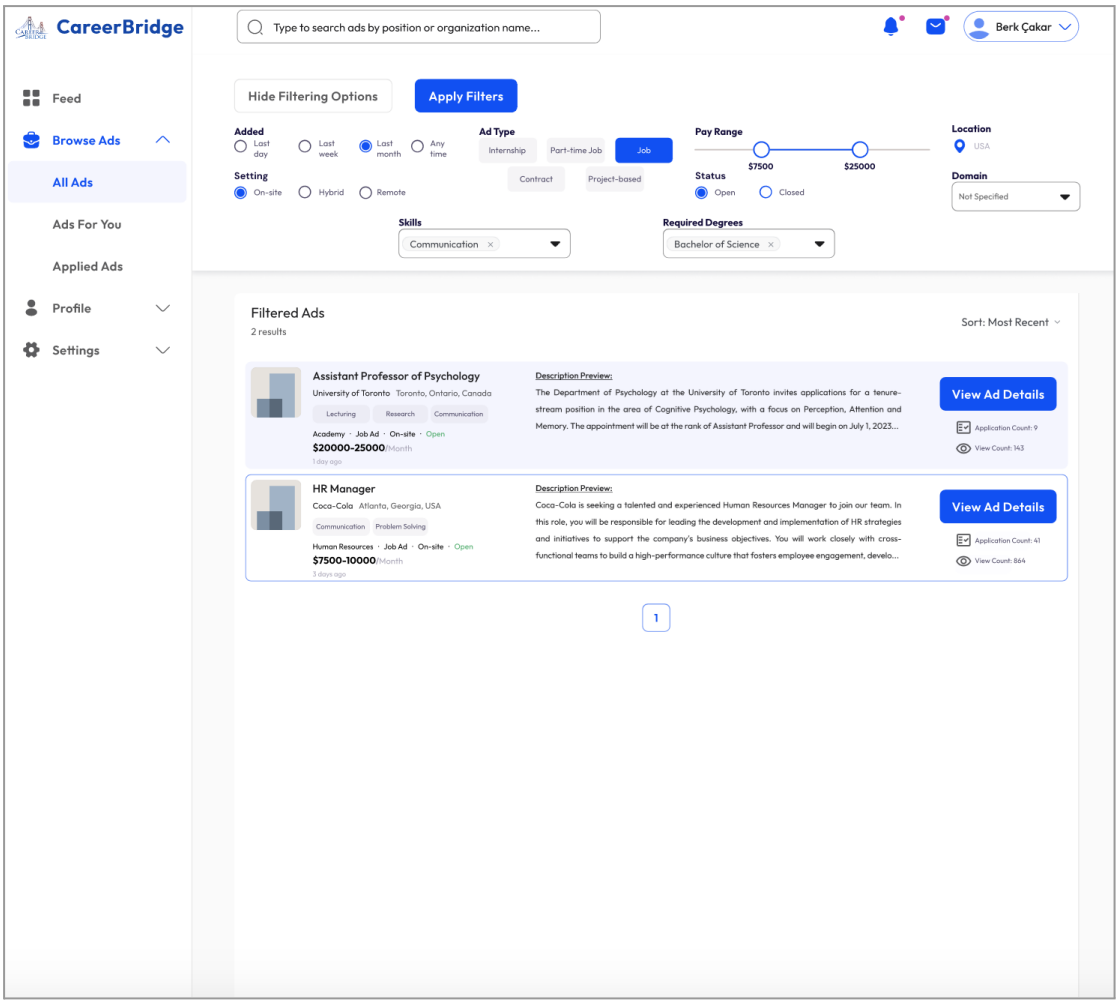

Figure 7

The ads will be filtered based on the following query:

```
SELECT *
FROM
  JobAdvertisement,
WHERE
  DATEDIFF(month,created_at,GETDATE()) = 1
  AND type = “Job”
  AND pay_range_min >= 7500
  AND pay_range_max <= 25000
  AND location LIKE “%USA%”
  AND setting = “On-site”
  AND status = “Open”
  AND ad_id IN (
    (SELECT
        ad_id
      FROM
```

```
            DegreeInJobAdvertisement NATURAL
            JOIN Degree
          WHERE
            name = “Bachelor of Science”)
        INTERSECT
          (SELECT
              ad_id
            FROM
              SkillInJobAdvertisement NATURAL JOIN Skill
            WHERE
              name = “Communication”));
```

If a sorting method is selected then a similar query with ORDER BY clause will be executed accordingly:

```
    SELECT *
    FROM
      JobAdvertisement,
    WHERE
      DATEDIFF(month,created_at,GETDATE()) = 1
      AND type = “Job”
      AND pay_range_min >= 7500
      AND pay_range_max <= 25000
      AND location LIKE “%USA%”
      AND setting = “On-site”
      AND status = “Open”
      AND ad_id IN (
        (SELECT
            ad_id
          FROM
            DegreeInJobAdvertisement NATURAL
            JOIN Degree
          WHERE
            name = “Bachelor of Science”)
        INTERSECT
          (SELECT
              ad_id
            FROM
              SkillInJobAdvertisement NATURAL JOIN Skill
            WHERE
              name = “Communication”))
ORDER BY created_at DESC;
```

Scenario 1.2 (When search is applied)

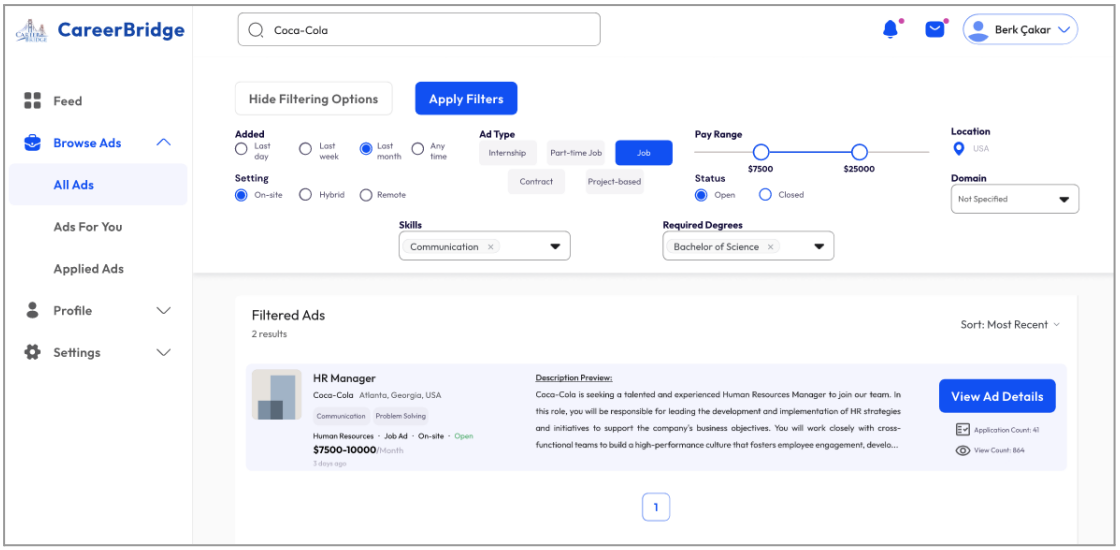

Figure 8

The following query will be used for search:

```
SELECT *
FROM
(SELECT *
    FROM JobAdvertisement
    WHERE
      title LIKE “%Coca-Cola%”
      OR organization LIKE “%Coca-Cola%”),
WHERE
  DATEDIFF(month,created_at, GETDATE()) = 1
  AND type = “Job”
  AND pay_range_min >= 7500
  AND pay_range_max <= 25000
  AND location LIKE “%USA%”
  AND setting = “On-site”
  AND status = “Open”
  AND ad_id IN (
    (SELECT ad_id
      FROM
        DegreeInJobAdvertisement NATURAL JOIN Degree
      WHERE
        name = “Bachelor of Science”)
    INTERSECT
      (SELECT ad_id
        FROM
          SkillInJobAdvertisement NATURAL JOIN Skill
        WHERE
          name = “Communication”)
  );
```

Scenario 2:

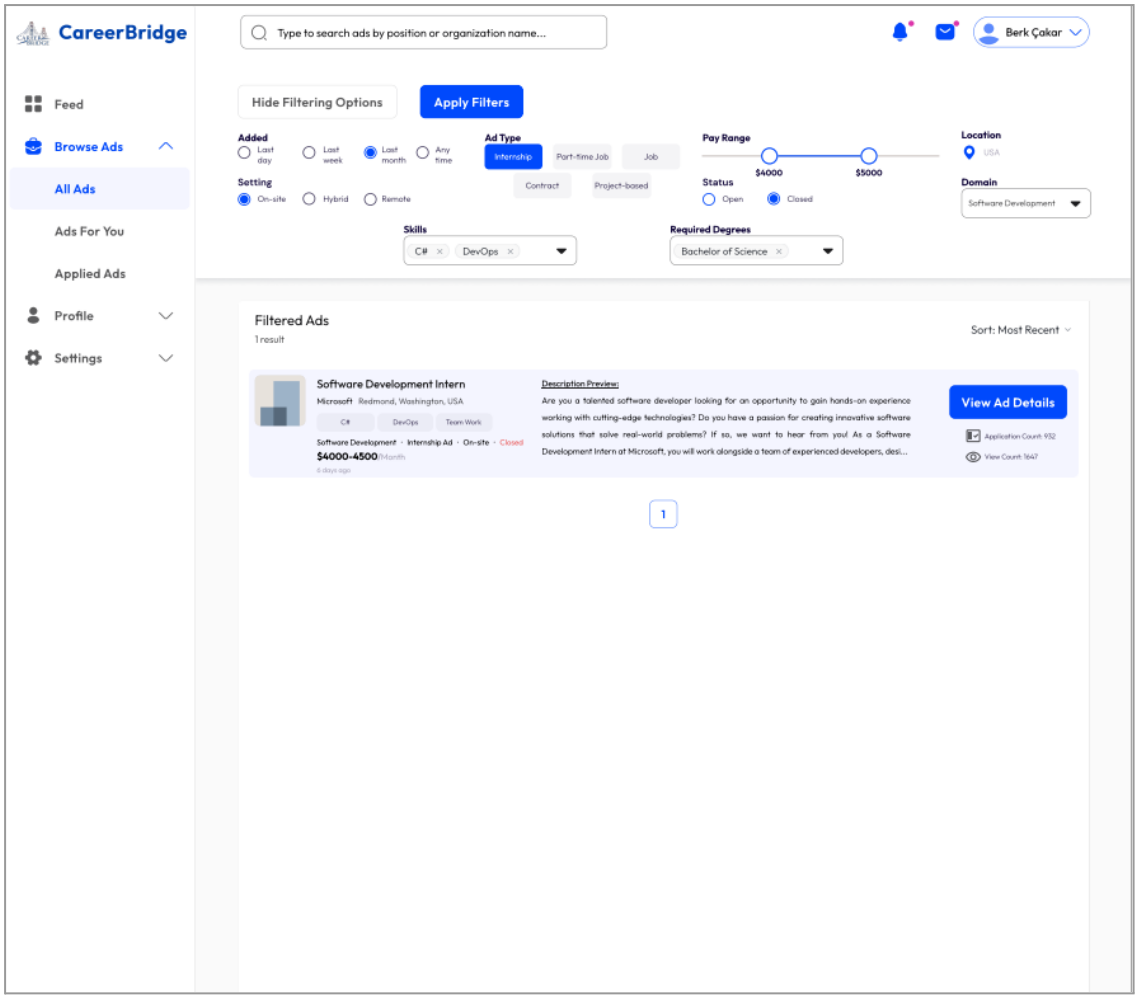

Figure 9

The ads will be filtered based on the following query:

```
SELECT *
FROM JobAdvertisement
WHERE
  DATEDIFF(month,created_at,GETDATE()) = 1
  AND type = “Internship”
  AND pay_range_min >= 4000
  AND pay_range_max <= 5000
  AND location LIKE “%USA%”
  AND setting = “On-site”
  AND status = “Closed”
  AND ad_id IN (
    (SELECT ad_id
      FROM
        DegreeInJobAdvertisement NATURAL JOIN Degree
      WHERE
```

```
        name = “Bachelor of Science”)
    INTERSECT
      (SELECT ad_id
        FROM
          SkillInJobAdvertisement SIJ,
          Skill S1,
          Skill S2
        WHERE
          SIJ.skill_id = S1.skil_id
          AND SIJ.skill_id = S2.skill_id
          AND S1.name = “C#” AND S2.name = “DevOps”));
```

Again, when a sorting method is selected then a similar query with ORDER BY clause will be executed accordingly:

```
SELECT *
FROM JobAdvertisement
WHERE
  DATEDIFF(month,created_at,GETDATE()) = 1
  AND type = “Internship”
  AND pay_range_min >= 4000
  AND pay_range_max <= 5000
  AND location LIKE “%USA%”
  AND setting = “On-site”
  AND status = “Closed”
  AND ad_id IN (
    (SELECT ad_id
      FROM
        DegreeInJobAdvertisement NATURAL JOIN Degree
      WHERE
        name = “Bachelor of Science”)
    INTERSECT
      (SELECT ad_id
        FROM
          SkillInJobAdvertisement SIJ,
          Skill S1,
          Skill S2
        WHERE
          SIJ.skill_id = S1.skil_id
          AND SIJ.skill_id = S2.skill_id
          AND S1.name = “C#” AND S2.name = “DevOps”))
ORDER BY created_at DESC;
```

However, as can be seen some filtering or searching options might be left empty depending on the user’s decision, or, for example, the user can supply more than one filtering constraint for skill. In that case, we will construct the queries depending on those selected options benefiting the if-else logic from the high level implementation language (i.e., Python). We have just demonstrated some example scenarios above.

4.7 Sample Ad Description

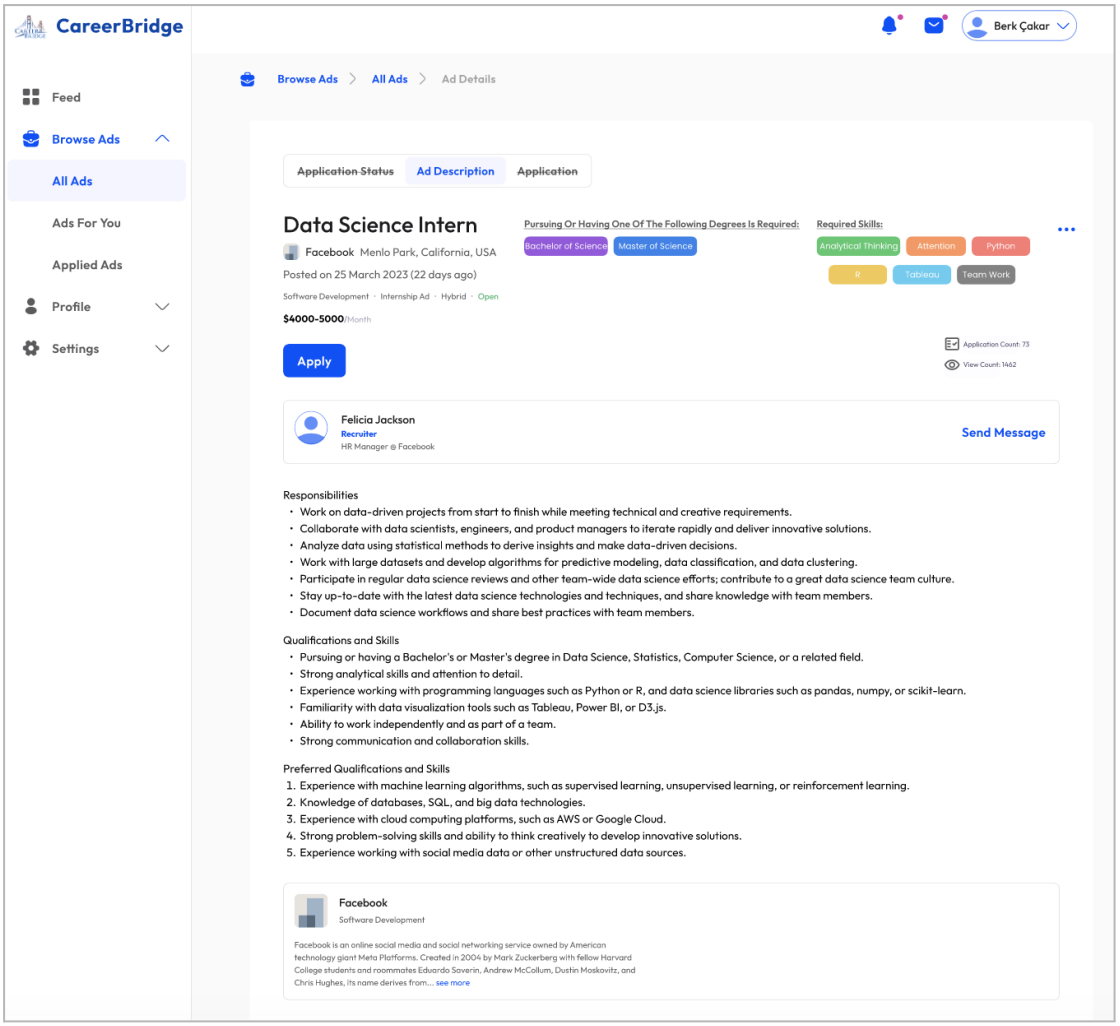

Figure 10

Inputs: @ad_id

The related attributes of an ad will be fetched based on the following query, these include the description, required skills, required degrees, application count, view count and more:

```
SELECT *
FROM JobAdvertisement
WHERE ad_id = @ad_id;
```

The related attributes of the recruiter will be fetched based on the following query:

```
SELECT
  User.first_name,
  User.last_name,
  AppUser.user_role,
```

```
  Experience.title,
  WorkExperience.company_name
FROM
  User
  INNER JOIN JobAdvertisement ON User.user_id =
JobAdvertisement.user_id
  INNER JOIN AppUser ON User.user_id = AppUser.user_id
  INNER JOIN Profile ON User.user_id = Profile.user_id
  INNER JOIN Experience ON Profile.profile_id =
Experience.profile_id
  LEFT OUTER JOIN WorkExperience ON Experience.experience_id =
WorkExperience.experience_id
WHERE
  Profile.is_application_specific IS FALSE
  AND JobAdvertisement.ad_id = @ad_id
  AND end_date IS NULL
  AND status = “Working”
ORDER BY
  start_date DESC
LIMIT 1;
```

The related attributes of the company giving the ad will be fetched based on the following query:

```
SELECT *
FROM
  Company C,
  JobAdvertisement J
WHERE
  J.ad_id = @ad_id
  AND C.company_name = J.organization;
```

Update the application view count when users enters to the ad description page:

```
CALL increase_view_count(@ad_id);
```

4.8 Ad Application Pages

4.8.1 Personal Information

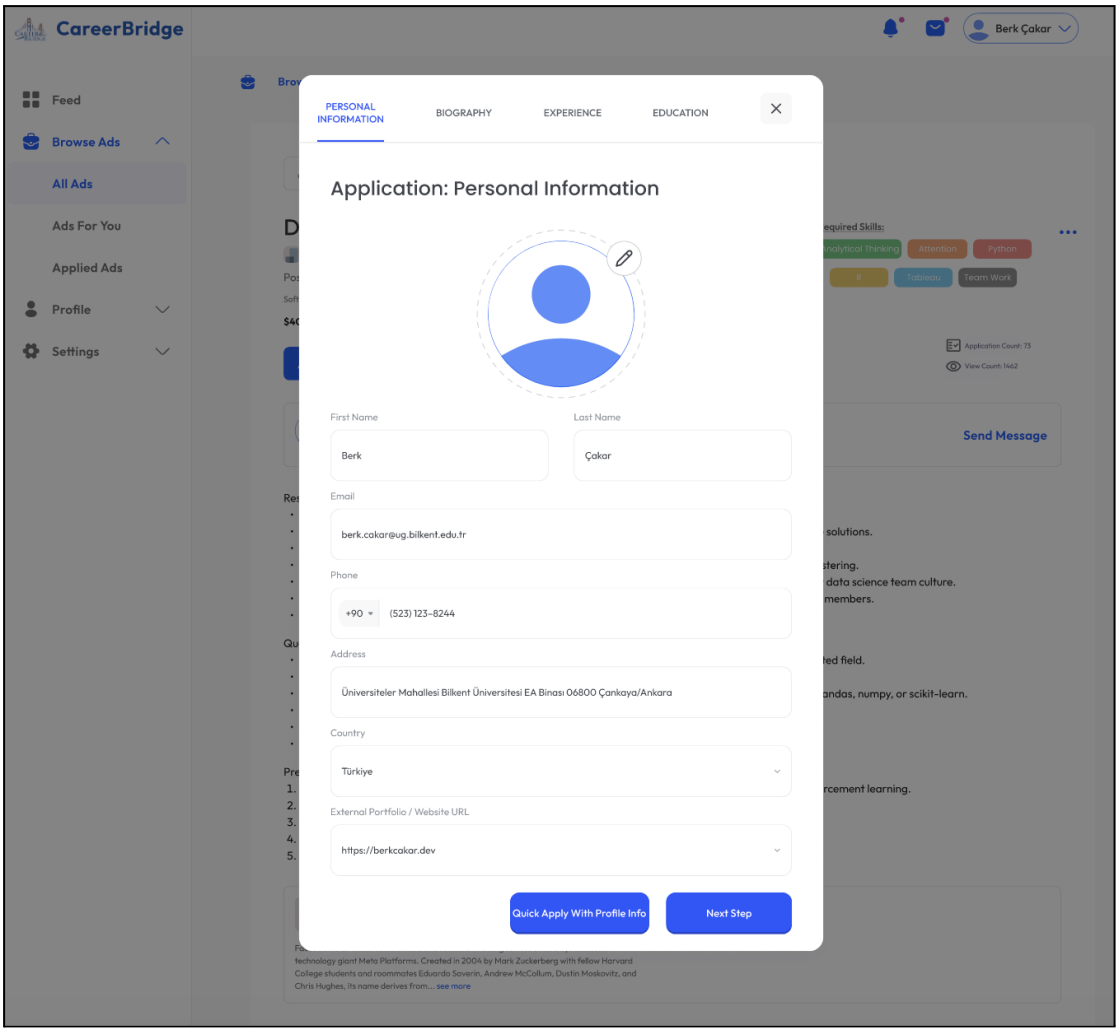

Figure 11

Inputs: @avatar, @first_name, @last_name, @phone_number, @email, @address, @country, @external_portfolio_url

4.8.2 Biography

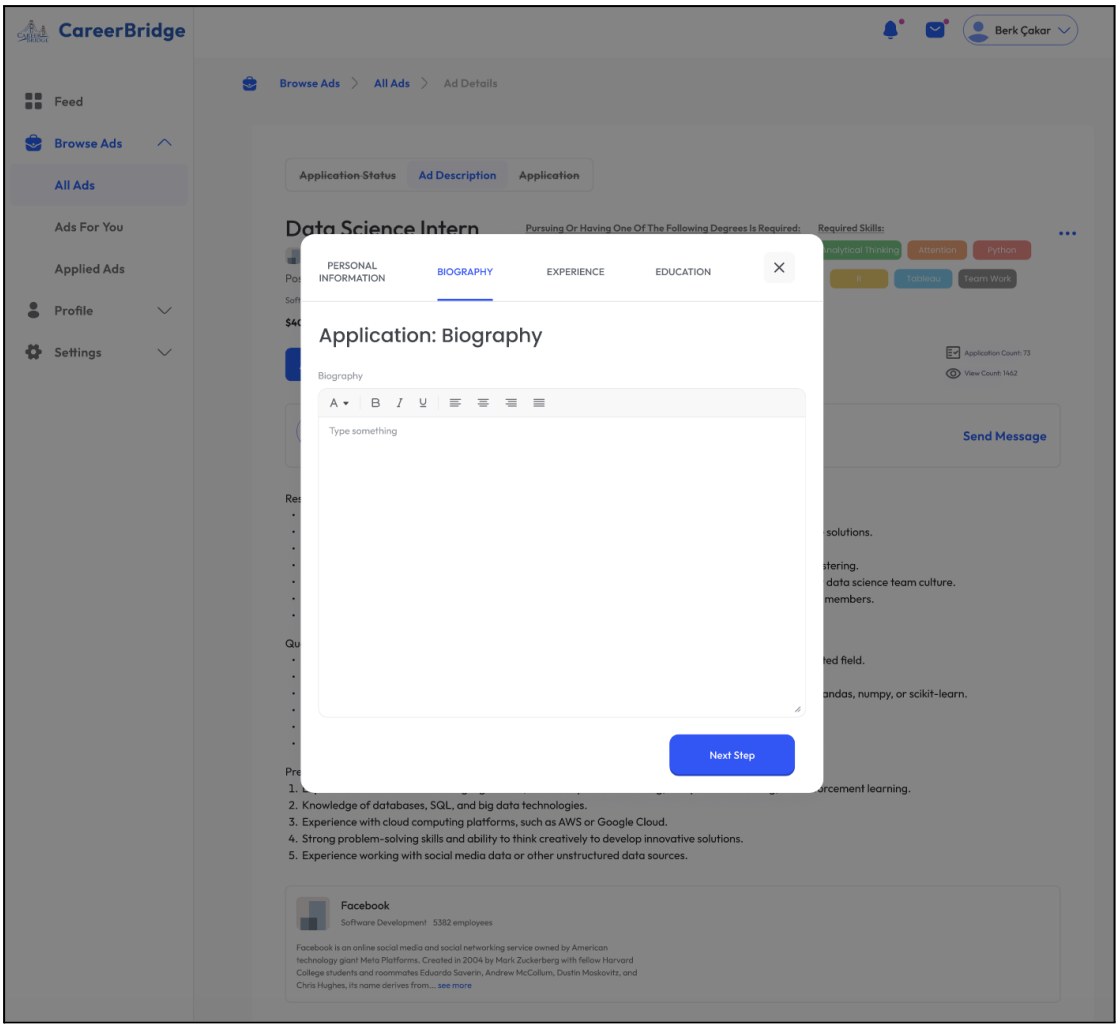

Figure 12

Inputs: @biography

4.8.3 Experience

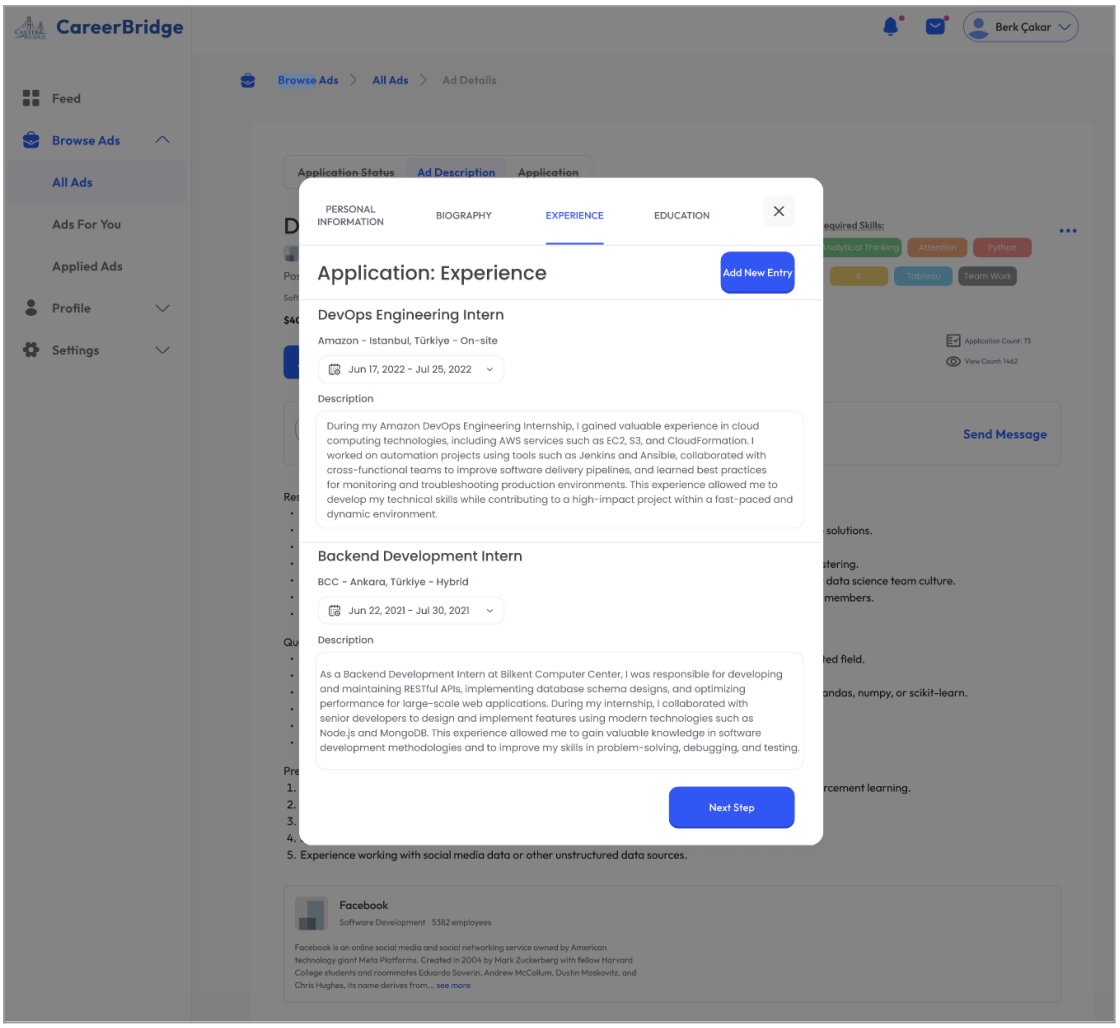

Figure 13

Inputs: @work_title1, @work_start_date1, @work_end_date1, @work_current_status1, @work_description1, @work_location1, @work_type1, @work_status1, @work_setting1, @work_company_name1 @work_title2, @work_start_date2, @work_end_date2, @current_status2, @work_description2, @work_location2, @work_type2, @work_status2, @work_setting2, @work_company_name2

4.8.4 Education

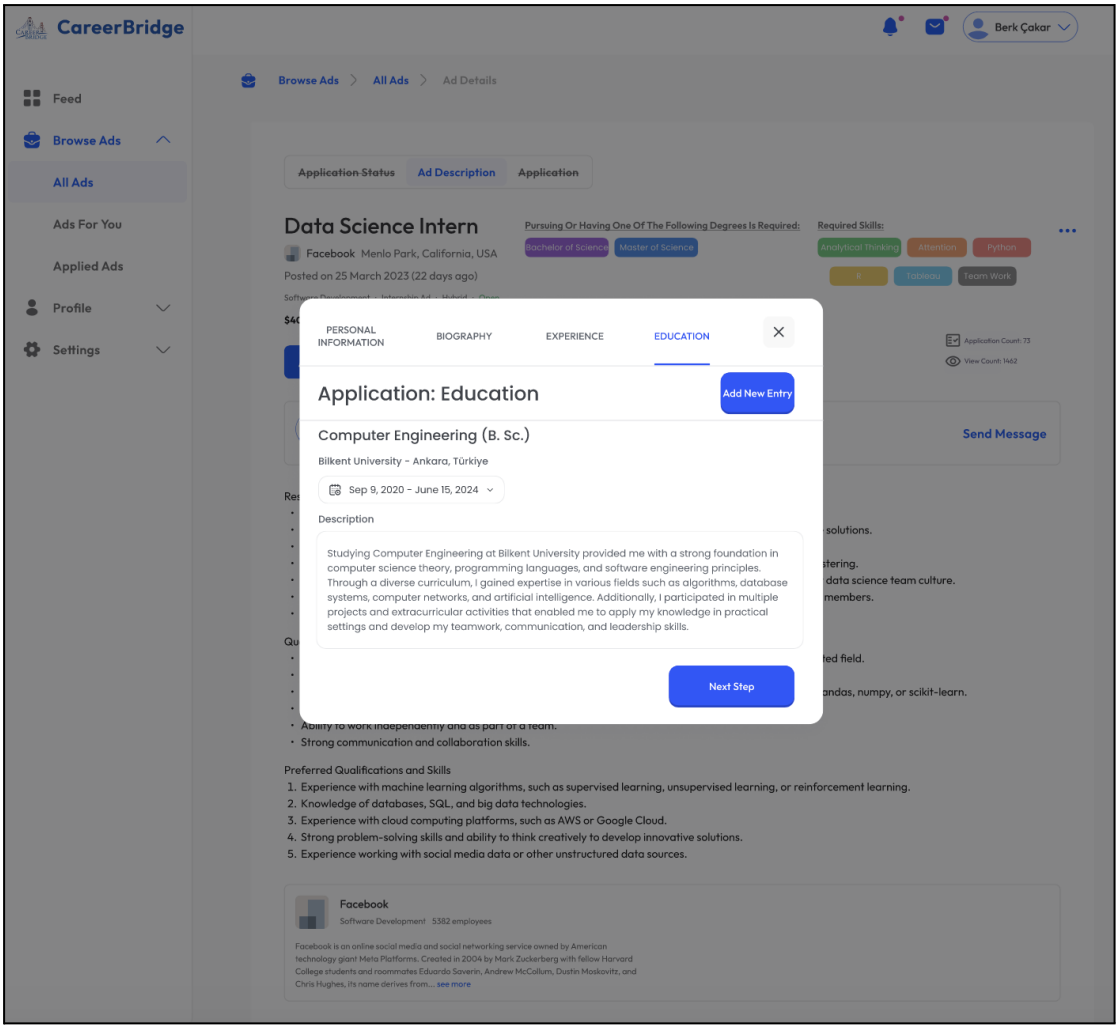

Figure 14

Inputs: @education_title, @education_start_date, @education_end_date, @education_current_status, @education_description, @education_degree_name, @education_location, @education_field_of_study, @education_school

4.8.5 Voluntary Work

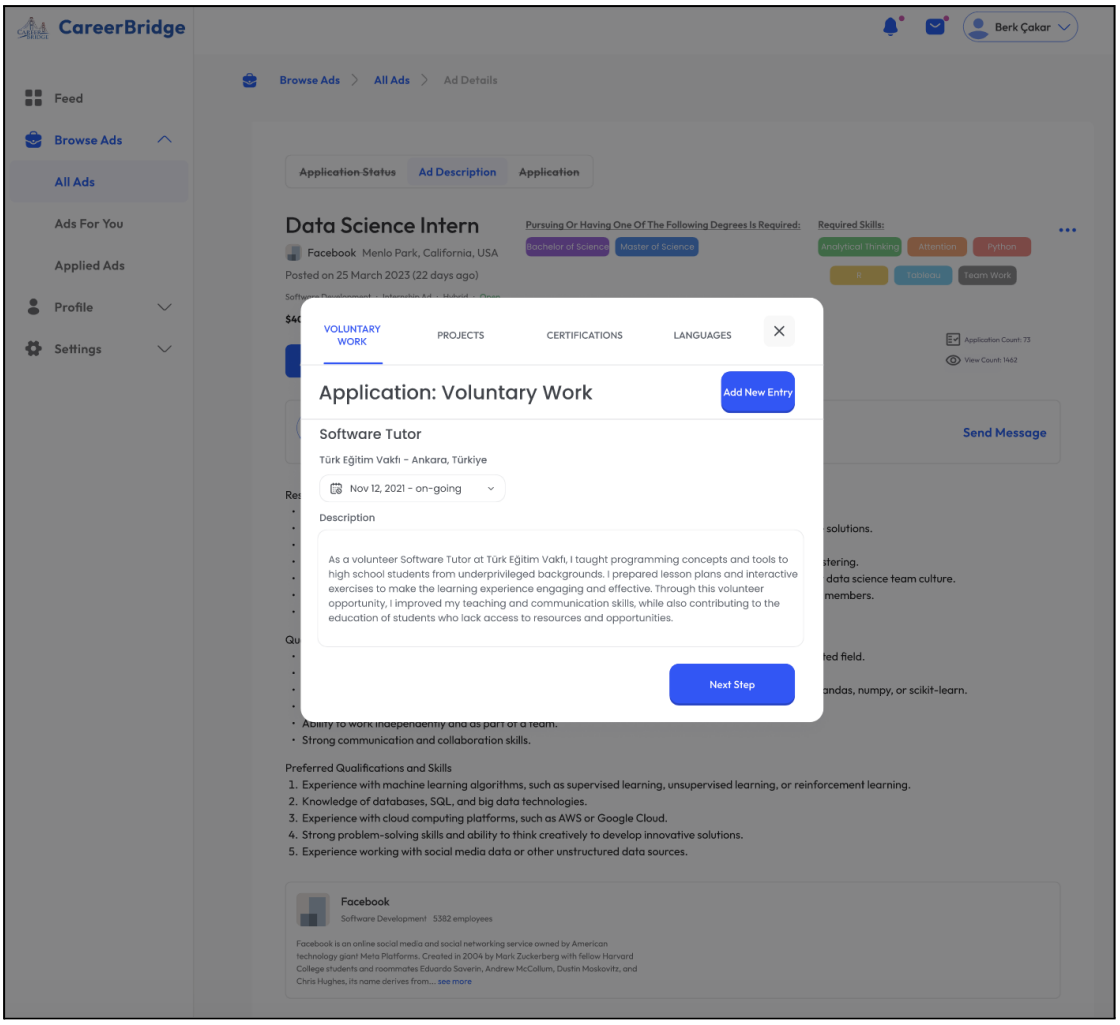

Figure 15

Inputs: @voluntary_title, @voluntary_start_date, @voluntary_end_date, @voluntary_current_status, @voluntary_description, @voluntary_location, @voluntary_organization_name, @voluntary_responsibility

4.8.6 Projects

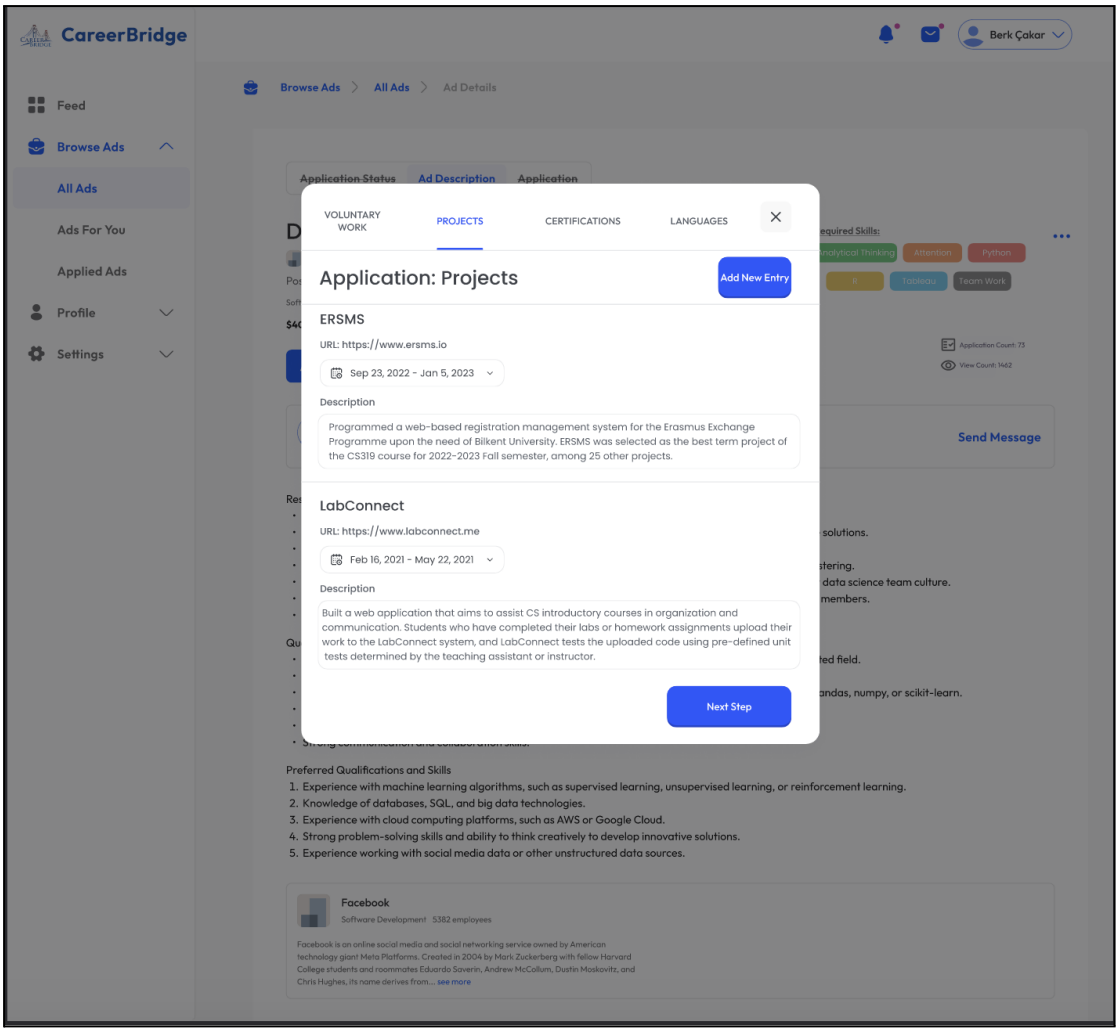

Figure 16

Inputs: @project_title1, @project_start_date1, @project_end_date1, @project_url1, @project_description1, @project_title2, @project_start_date2, @project_end_date2, @project_url2, @project_description2

4.8.7 Certifications

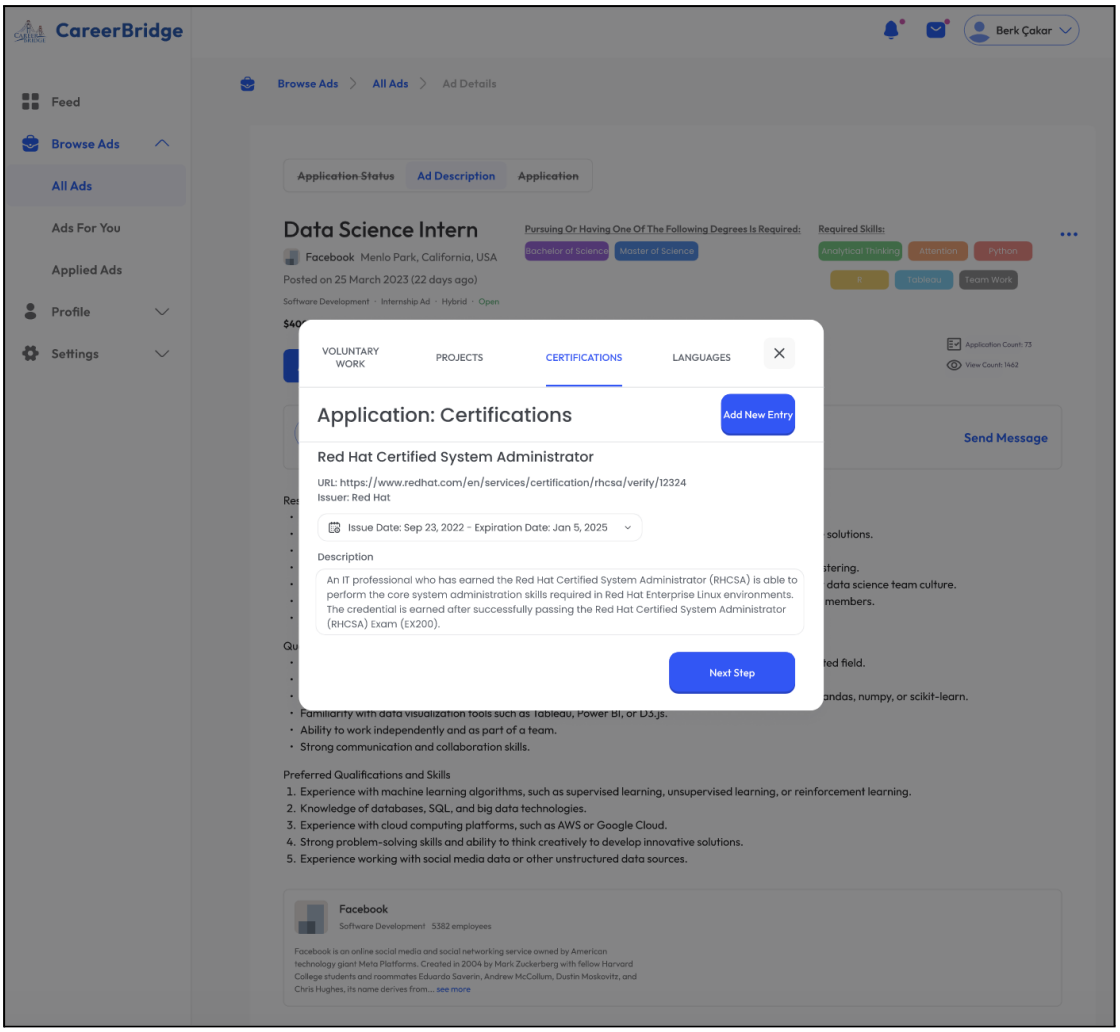

Figure 17

Inputs: @certification_name, @certification_description, @certification_credential_url, @certification_issue_date, @certification_issuer, @certification_expiration_date

4.8.8 Awards

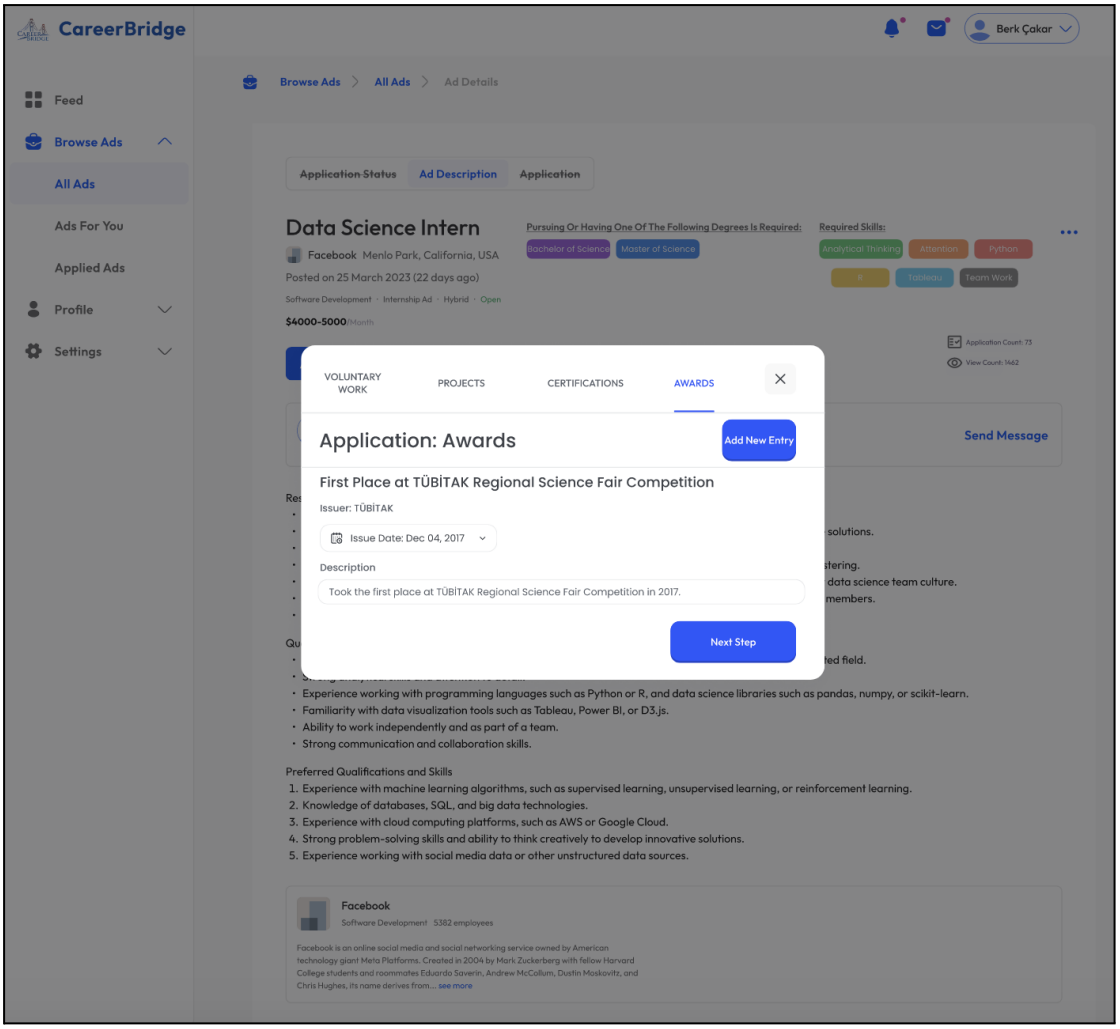

Figure 18

Inputs: @award_title, @award_description, @award_issuer, @award_issue_date

4.8.9 Test Scores

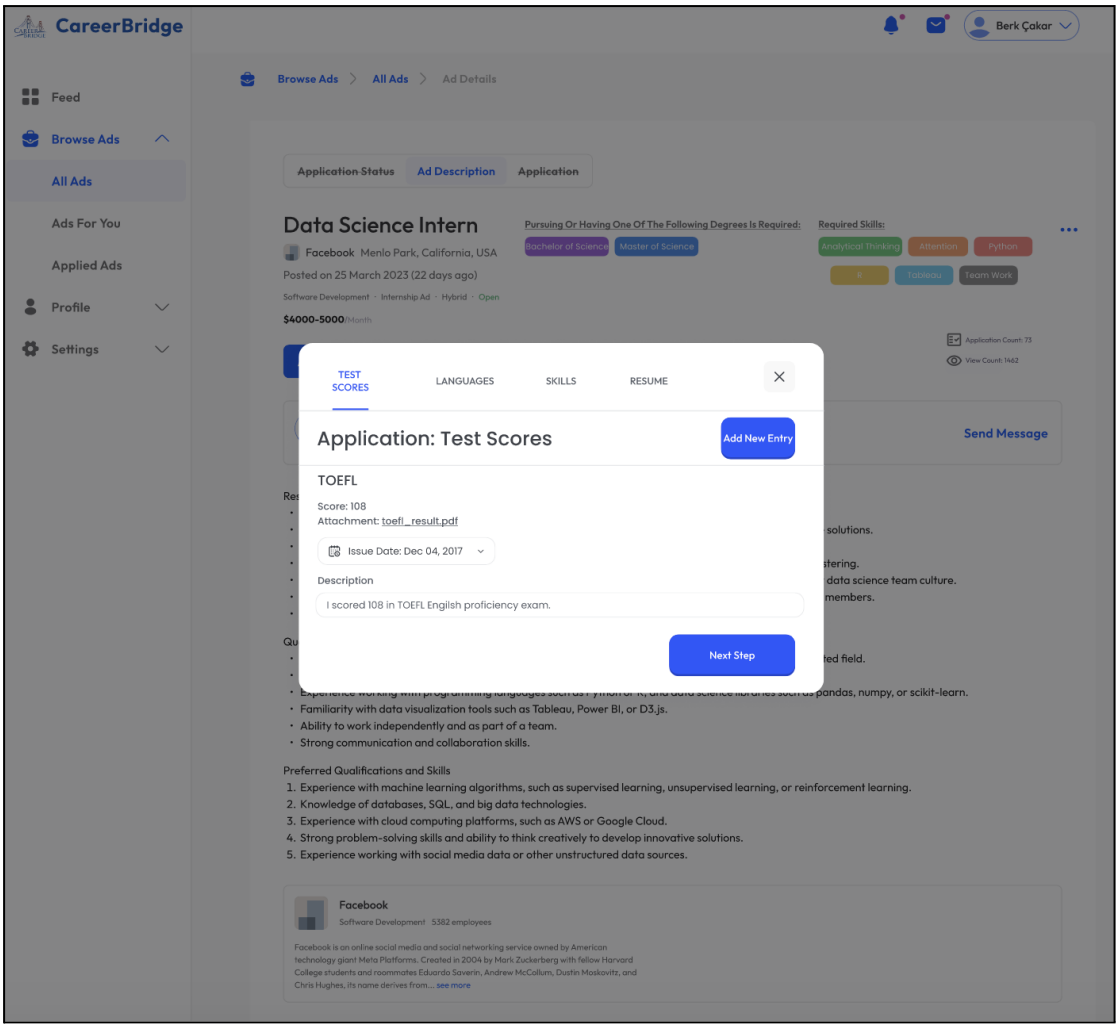

Figure 19

Inputs: @test_name, @test_date, @test_score, @test_attachment, @test_date, @test_description

4.8.10 Languages

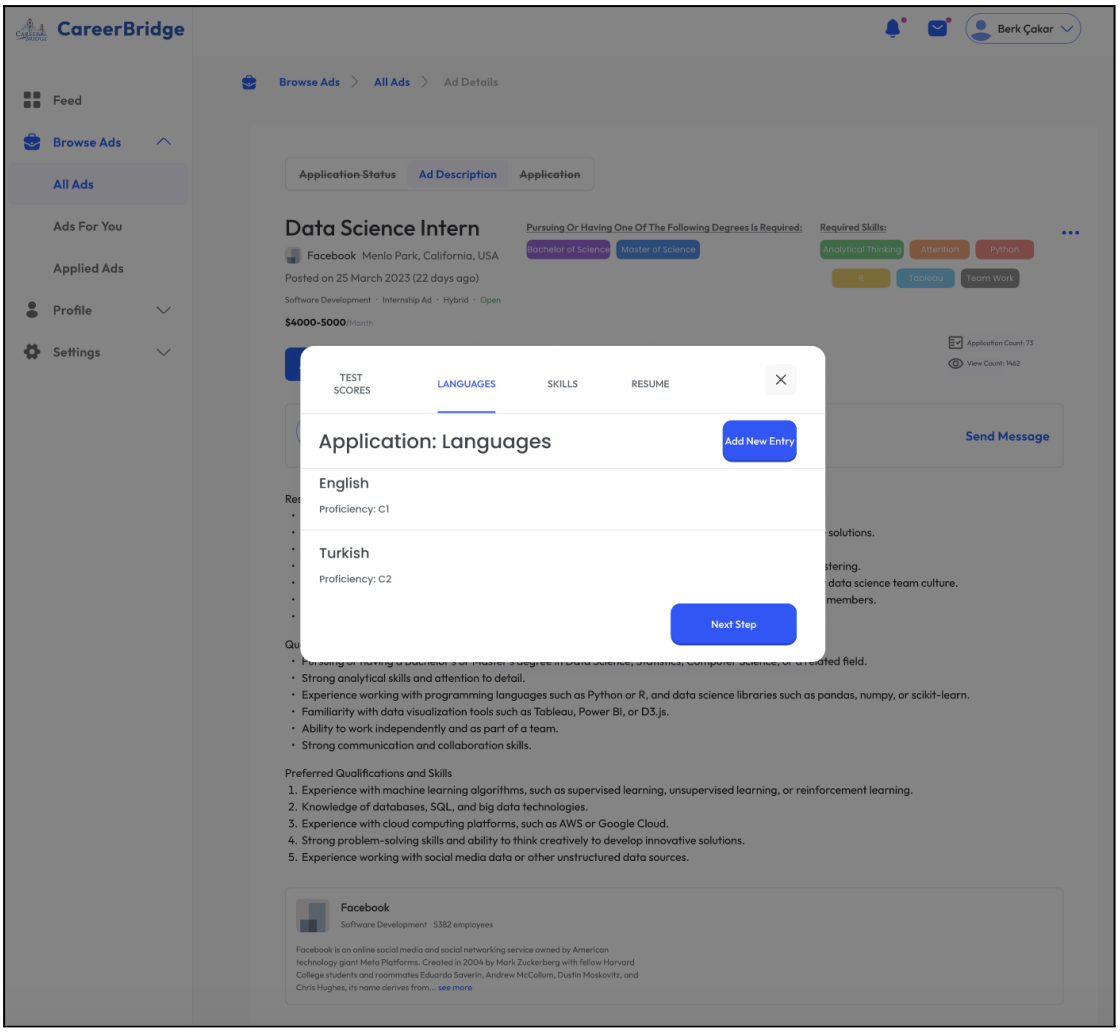

Figure 20

Inputs: @language_name1, @language_proficiency1, @language_name2, @language_proficiency2

4.8.11 Skills

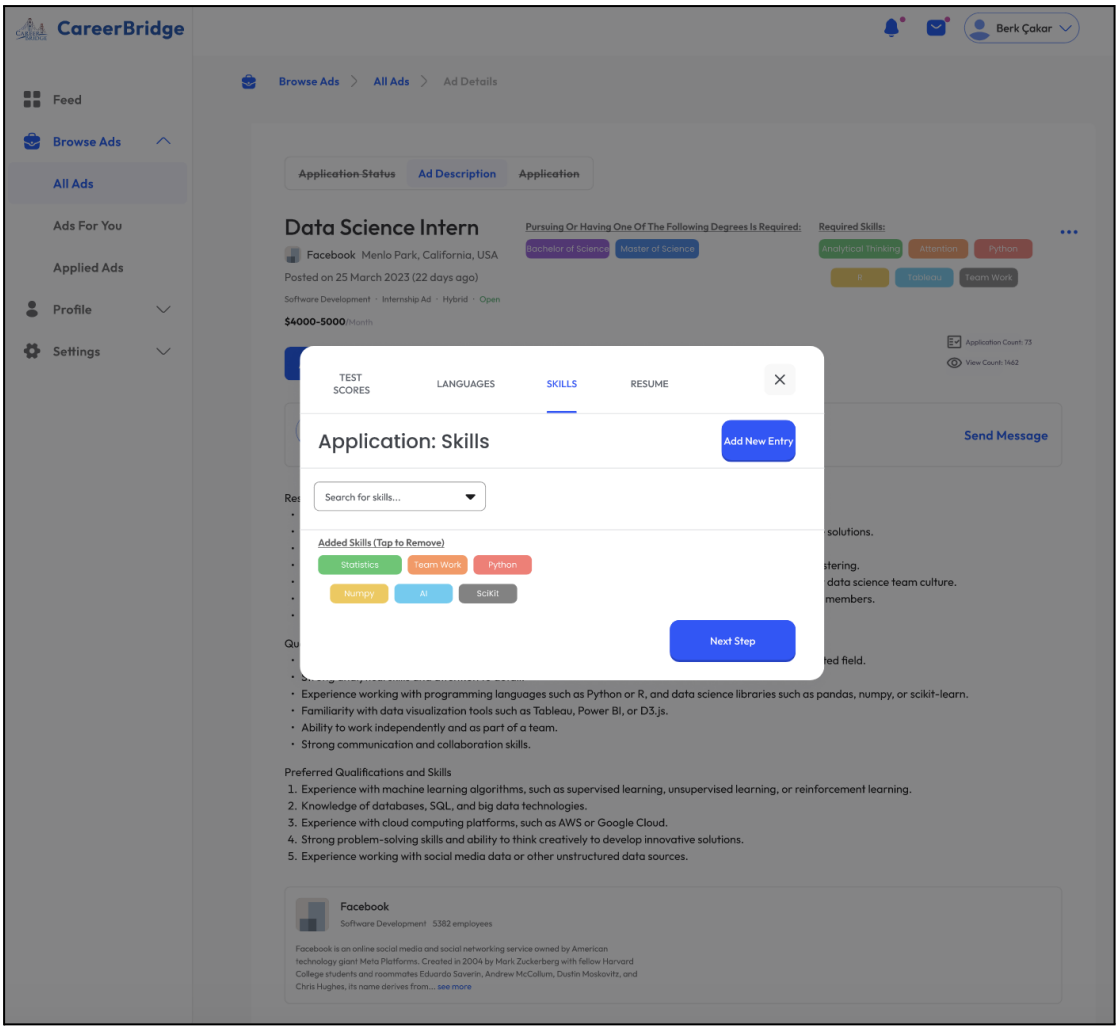

Figure 21

Inputs: @skill_name1, @skill_name2, @skill_name3, @skill_name4, @skill_name5, @skill_name6

4.8.12 Upload Resume

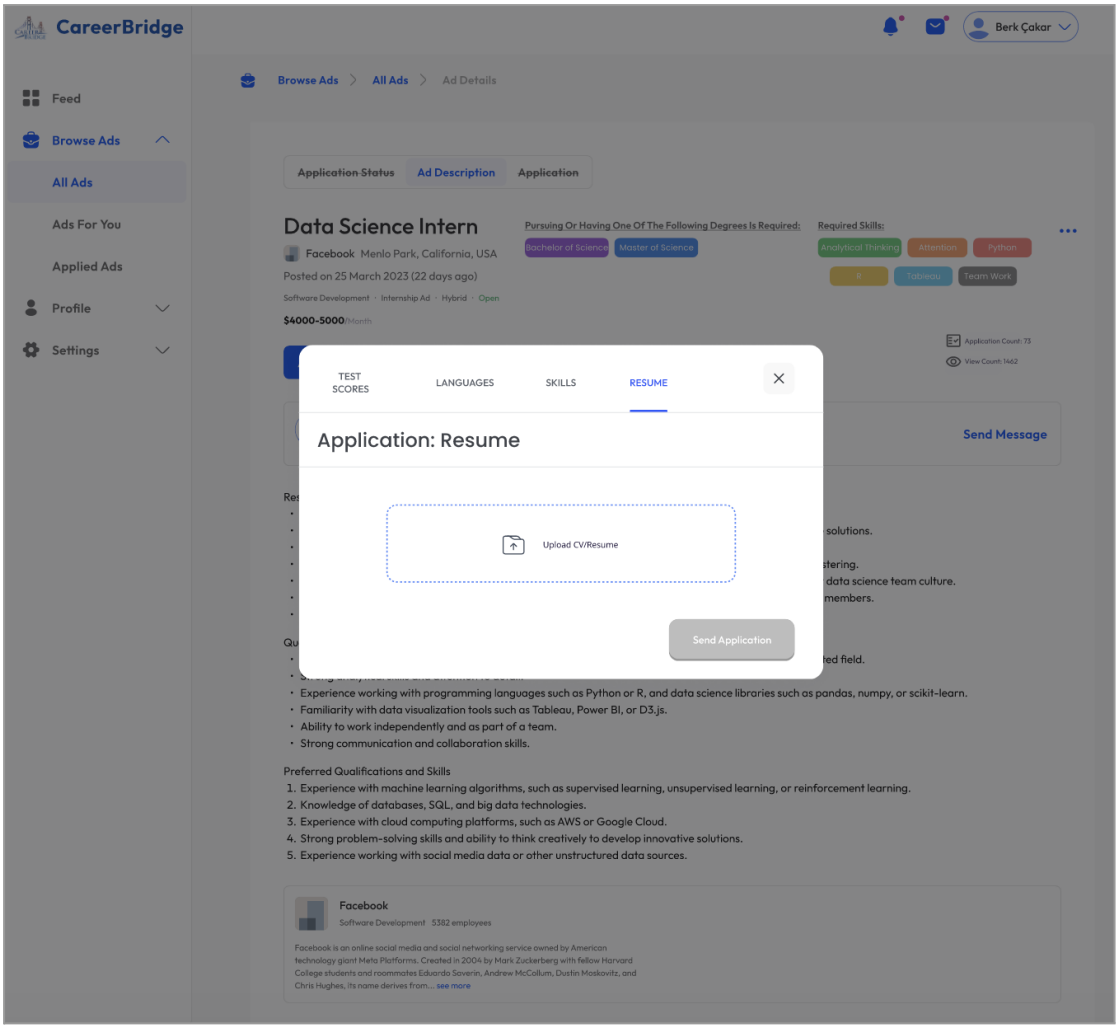

Figure 22

4.8.13 Uploaded Resume

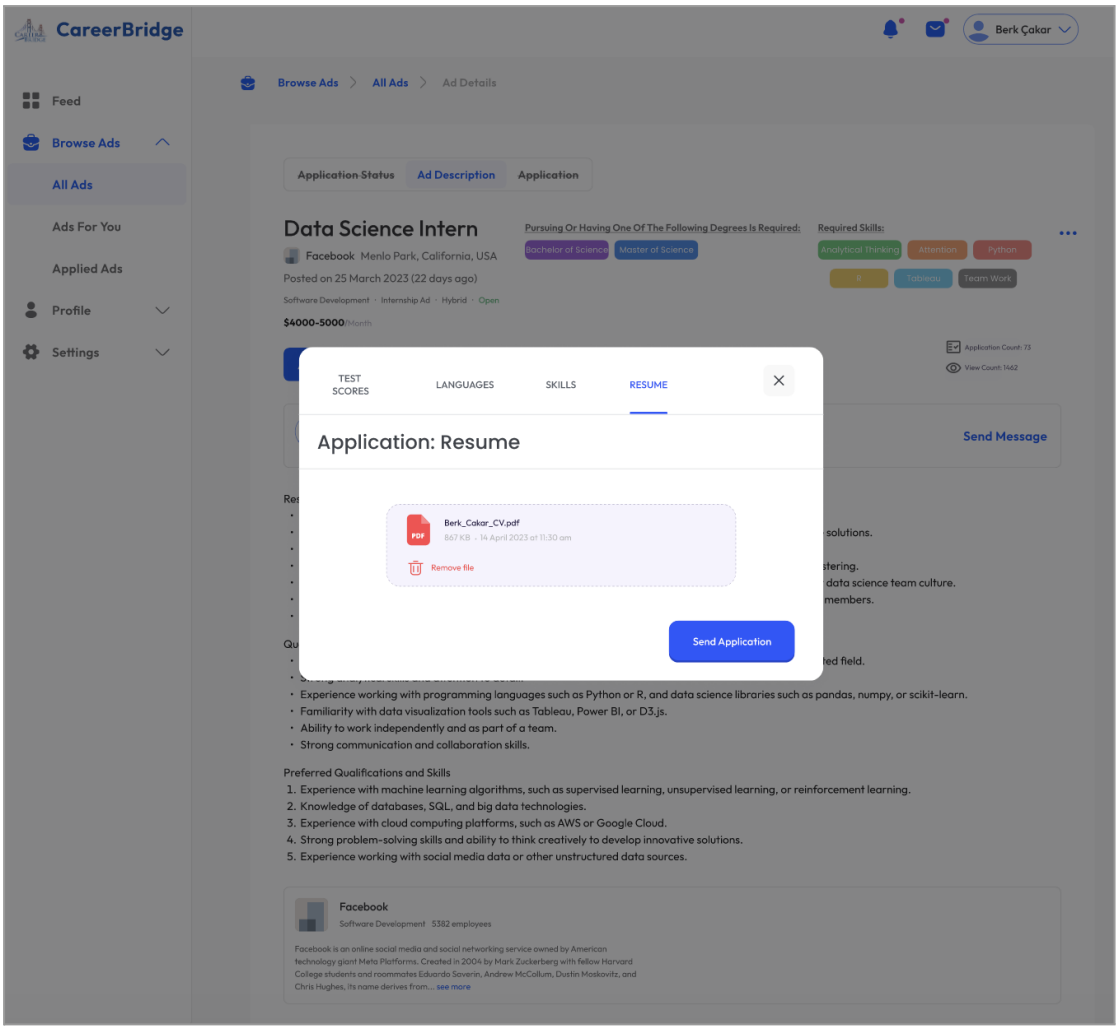

Figure 23

Inputs: @resume

SQL query on Quick Apply with User Profile Info:

```
INSERT INTO JobAdvertisementResponse(profile_id, ad_id,
apply_date)
SELECT
  profile_id,
  @ad_id,
  NOW()
FROM
  Profile
WHERE
  user_id = @user_id
  AND is_application_specific IS FALSE;
```

SQL query on Submit of Job Application (all inputs from the Section 4.8 are used):

```
INSERT INTO PROFILE(
  avatar, country, external_portfolio_url,
  address, biography, is_private, resume,
  phone_number, is_application_specific,
  created_at
)
SELECT
  @avatar,
  @country,
  @external_portfolio_url,
  @address,
  @biography,
  0,
  @resume,
  @phone_number,
  1,
  NOW();
```

Now assume the ID of the profile created is @profile_id:

```
INSERT INTO Experience(
  profile_id, start_date, end_date,
  title, description, current_status,
  location
)
VALUES
  (
    @profile_id, @work_start_date1, @word_end_date1,
    @work_title1, @work_description1,
    @work_status1, @work_location1
  ),
  (
    @profile_id, @work_start_date2, @word_end_date2,
    @work_title2, @work_description2,
    @work_status2, @work_location2
  ),
  (
    @profile_id, @education_start_date,
    @education_end_date, @education_title,
    @education_description, @education_current_status,
    @education_location
  ),
  (
    @profile_id, @voluntary_start_date,
    @voluntary_end_date, @voluntary_title,
    @voluntary_description, @voluntary_current_status,
    @voluntary_location);
```

Now assume the ID of the experiences created are @experience_id1, @experience_id2, @experience_id3 and @experience_id4, respectively. Also, the corresponding entries of the NonProfitOrganization, Company and School already exist in the database:

```
INSERT INTO WorkExperience(experience_id, setting, type)
VALUES
  (
    @experience_id1, @work_setting1,
    @work_type1, @work_company_name1
  ),
  (
    @experience_id2, @work_setting2,
    @work_type2, @work_company_name2
  );
```

SQL query to insert into Degree:

```
INSERT INTO Degree(name) VALUES (@education_degree_name);
```

Now assume the ID of the degree created is @degree_id. SQL query to insert into EducationalExperience:

```
INSERT INTO EducationalExperience(
  experience_id, grade, field_of_study,
  school_name, degree_id
)
VALUES
  (
    @experience_id3, @education_field_of_study,
    @education_school, @degree_id
  );
```

SQL query to insert into VoluntaryExperience:

```
INSERT INTO VoluntaryExperience(
  experience_id, responsibility, organization_name
)
VALUES
  (
    @experience_id4, @voluntary_responsibility,
    @voluntary_organization_name
  );
```

SQL query to insert into Project:

```
INSERT INTO Project(
  profile_id, title, description, start_date,
  end_date, project_url
```

```
)
VALUES
  (
    @profile_id, @project_title1, @project_description1,
    @project_start_date1, @project_end_date1,
    @project_url1
  ),
  (
    @profile_id, @project_title2, @project_description2,
    @project_start_date2, @project_end_date2,
    @project_url2
  );
```

SQL query to insert into Certification:

```
INSERT INTO Certification(
  profile_id, certification_name, description,
  credential_url, issue_date, issuer,
  expiration_date
)
VALUES
  (
    @profile_id, @certification_name,
    @certification_description,
    @certification_credential_url,
    @certification_issue_date, @certification_issuer,
    @certification_expiration_date
  );
```

SQL query to insert into Award:

```
INSERT INTO Award(
  profile_id, title, description, issue_date,
  issuer
)
VALUES
  (
    @profile_id, @award_title, @award_description,
    @award_issuer, @award_issue_date
  );
```

SQL query to insert into TestScore:

```
INSERT INTO TestScore(
  profile_id, test_name, description,
  test_date, score, attachment
)
VALUES
  (
    @profile_name, @test_name, @test_description,
```

```
    @test_date, @test_score, @test_attachment);
```

SQL query to insert into LanguageProficiency:

```
INSERT INTO LanguageProficiency(
  profile_id, language_name, proficiency
)
VALUES
  (
    @profile_id, @language_name1, @language_proficiency1
  ),
  (
    @profile_id, @language_name2, @language_proficiency2
  );
```

SQL query to insert into Skill:

```
INSERT INTO Skill(profile_id, name)
VALUES
  (@profile_id, @skill_name1),
  (@profile_id, @skill_name2),
  (@profile_id, @skill_name3),
  (@profile_id, @skill_name4),
  (@profile_id, @skill_name5),
  (@profile_id, @skill_name6);
```

SQL query to insert into JobAdvertisementResponse:

```
INSERT INTO JobAdvertisementResponse(profile_id, ad_id,
apply_date)
SELECT
  @profile_id,
  @ad_id,
  NOW();
```

After the application, update the application_count:

```
CALL increase_application_count(@ad_id);
```

Similar to the section where we perform filtering, the number of possible queries is infinite. In this series of queries we handled a specific case with the given inputs to apply an advertisement. Again, with the help of a high level programming language's capabilities we can construct the corresponding queries based on the filled fields or number of elements in a field (for example multiple skills or work experiences may appear). In a similar way, users will be warned to fill in the fields which do not have a default value or set not to be NULL.

4.9 Applied Ads Page

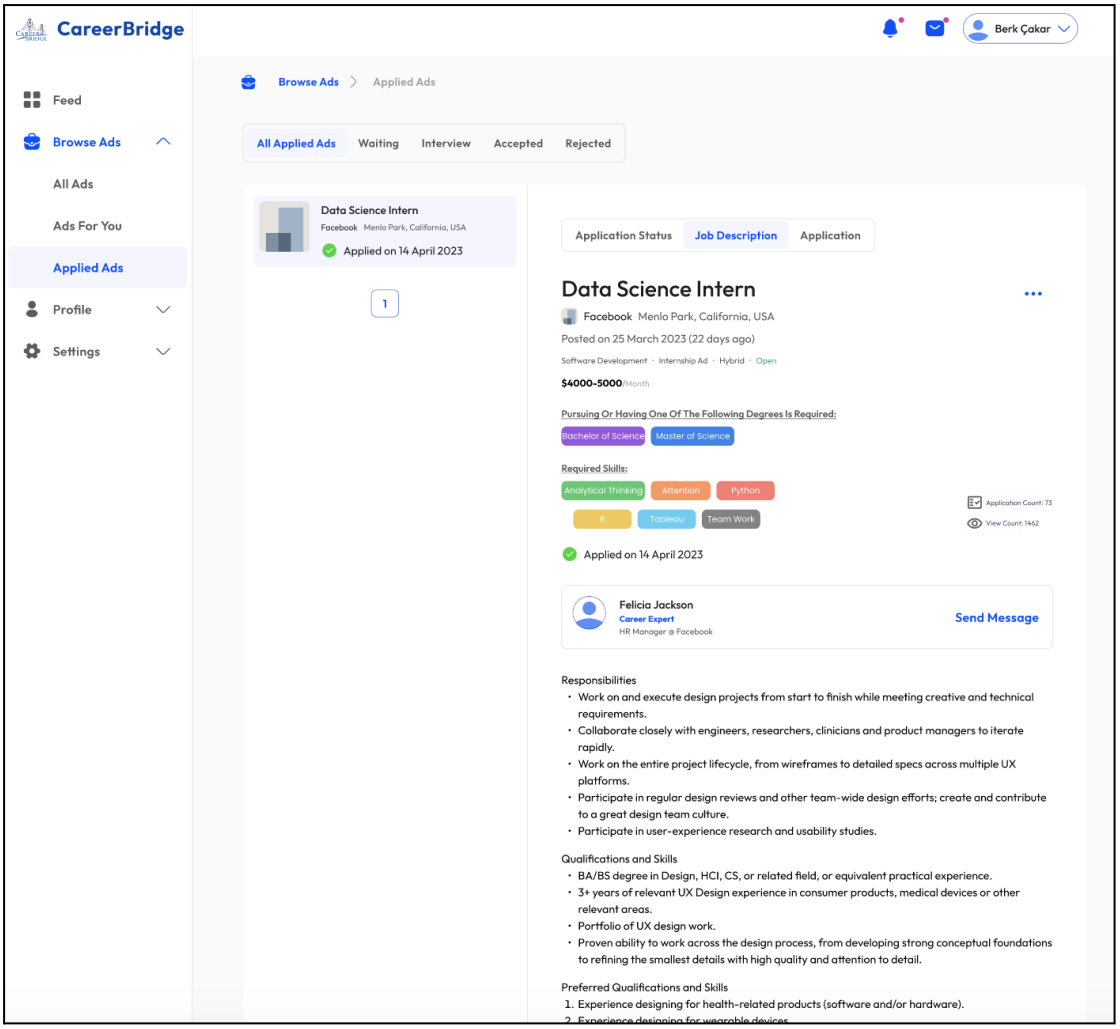

Figure 24

Inputs: @user_id, @ad_id

SQL query to get all ads that user applied ordered by their application date:

```
SELECT *
FROM
  JobAdvertisementResponse
WHERE
  user_id = @user_id
ORDER BY
  apply_date DESC;
```

SQL query to get the info of the selected ad:

```
SELECT *
FROM
  JobAdvertisement JA,
  JobAdvertisementResponse JAR
WHERE
  JA.ad_id = JAR.ad_id
  AND ja.ad_id = @ad_id
  AND JAR.user_id = @user_id;
```

SQL query to get the recruiter info of the selected ad:

```
SELECT
  User.first_name,
  User.last_name,
  AppUser.user_role,
  Experience.title,
  WorkExperience.company_name
FROM
  User
  INNER JOIN JobAdvertisement ON User.user_id =
JobAdvertisement.user_id
  INNER JOIN AppUser ON User.user_id = AppUser.user_id
  INNER JOIN Profile ON User.user_id = Profile.user_id
  INNER JOIN Experience ON Profile.profile_id =
Experience.profile_id
  LEFT OUTER JOIN WorkExperience ON Experience.experience_id =
WorkExperience.experience_id
WHERE
  Profile.is_application_specific IS FALSE
  AND JobAdvertisement.ad_id = @ad_id
  AND end_date IS NULL
  AND status = “Working”
ORDER BY
  start_date DESC
LIMIT
  1;
```

4.10 Published Ads Page

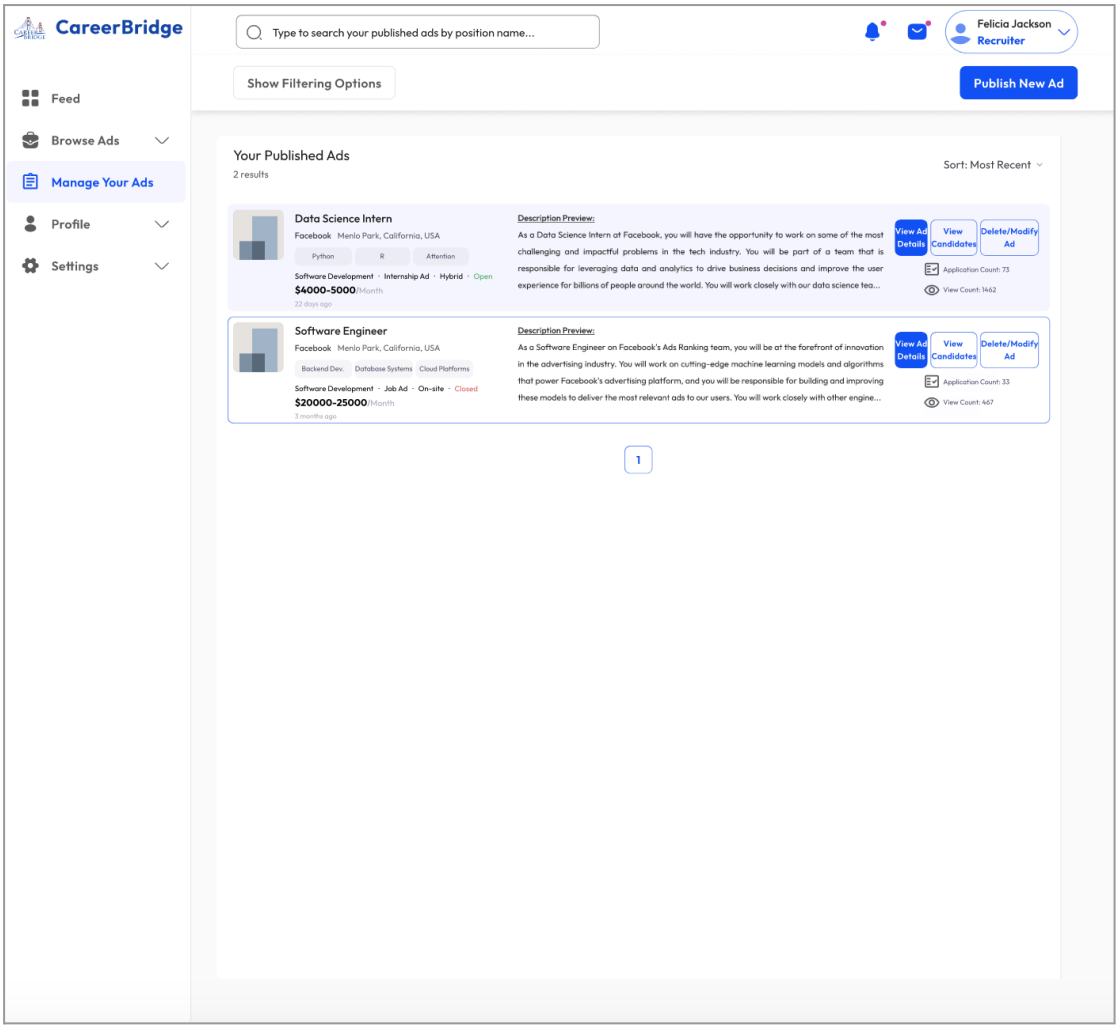

Figure 25

Inputs: @user_id, @ad_id

SQL query to get all ads that is published by the user logged in:

```
SELECT *
FROM JobAdvertisement
WHERE
  user_id = @user_id
ORDER BY
  created_at DESC;
```

SQL query to get 3 skills for each published ad:

```
SELECT name
FROM SkillInJobAdvertisement NATURAL JOIN Skill
WHERE ad_id = @ad_id
LIMIT 3;
```

Delete of a published ad by clicking on the delete button:

```
DELETE FROM JobAdvertisement
WHERE ad_id = @ad_id;
```

4.11 Candidates (Applicants) of an Ad Page

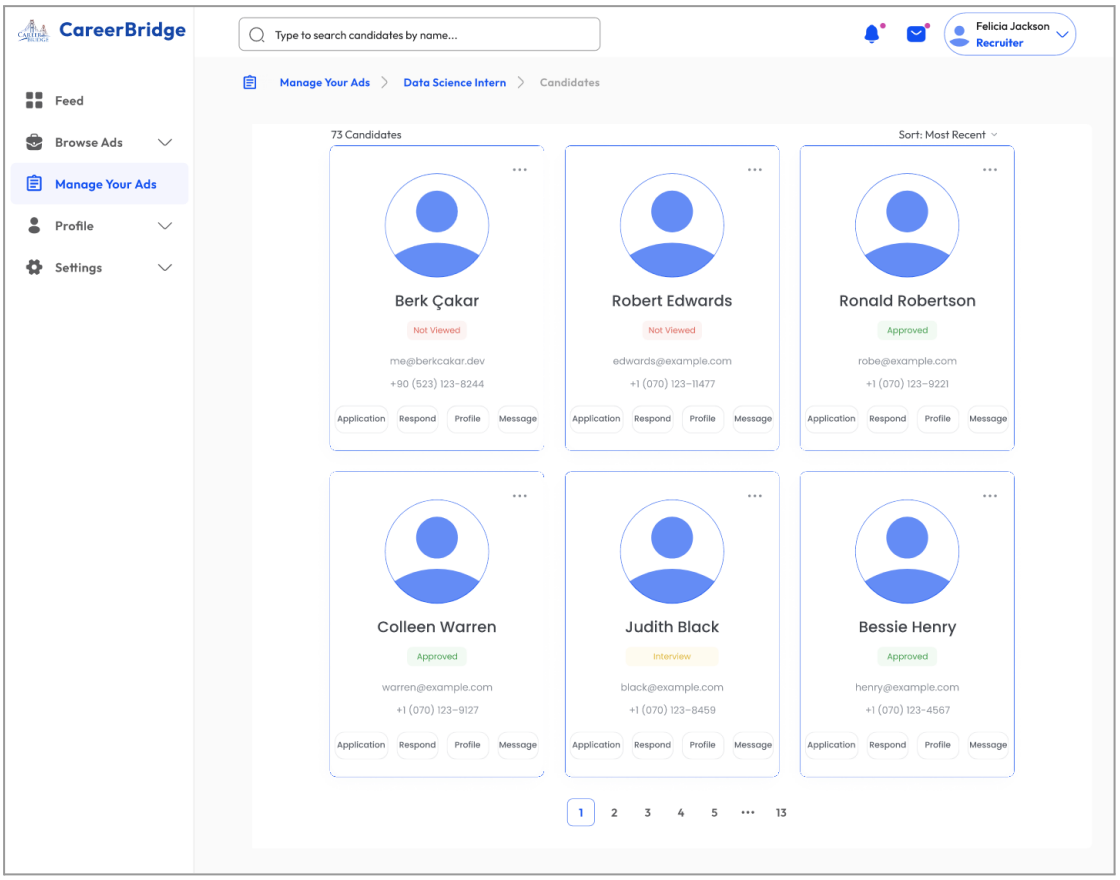

Figure 26

Inputs: @ad_id, @profile_id, @response, @search_input

SQL query to fetch the applicants and their relevant information:

```
SELECT
  P.avatar,
  P.phone_number,
  U.first_name,
  U.last_name,
  U.email,
  JR.response,
FROM
  JobAdvertisementResponse JR,
  Profile P,
  User U
WHERE
  JR.user_id = U.user_id
```

```
  AND JR.ad_id = @ad_id
  AND U.user_id = P.user_id;
```

SQL query to sort the applicants when a sorting option is selected, for example most recent first:

```
SELECT
  P.avatar,
  P.phone_number,
  U.first_name,
  U.last_name,
  U.email,
  JR.response,
FROM
  JobAdvertisementResponse JR,
  Profile P,
  User U
WHERE
  JR.user_id = U.user_id
  AND JR.ad_id = @ad_id
  AND U.user_id = P.user_id
ORDER BY
  JR.apply_date DESC;
```

SQL query to retrieve the candidates when the candidates are searched by their names:

```
SELECT
  P.avatar,
  P.phone_number,
  U.first_name,
  U.last_name,
  U.email,
  JR.response,
FROM
  JobAdvertisementResponse JR,
  Profile P,
  (SELECT *
    FROM User
    WHERE
      CONCAT(first_name, ' ', last_name) LIKE “%@search_input%”
  ) as U
WHERE
  JR.user_id = U.user_id
  AND JR.ad_id = @ad_id
  AND U.user_id = P.user_id
ORDER BY
  JR.apply_date DESC;
```

SQL query when a recruiter respond the job application:

```
UPDATE JobAdvertisementResponse
SET response = @response
WHERE profile_id = @profile_id AND ad_id = @ad_id;
```

4.12 CareerBridge Specific Pages

4.12.1 Profile Page

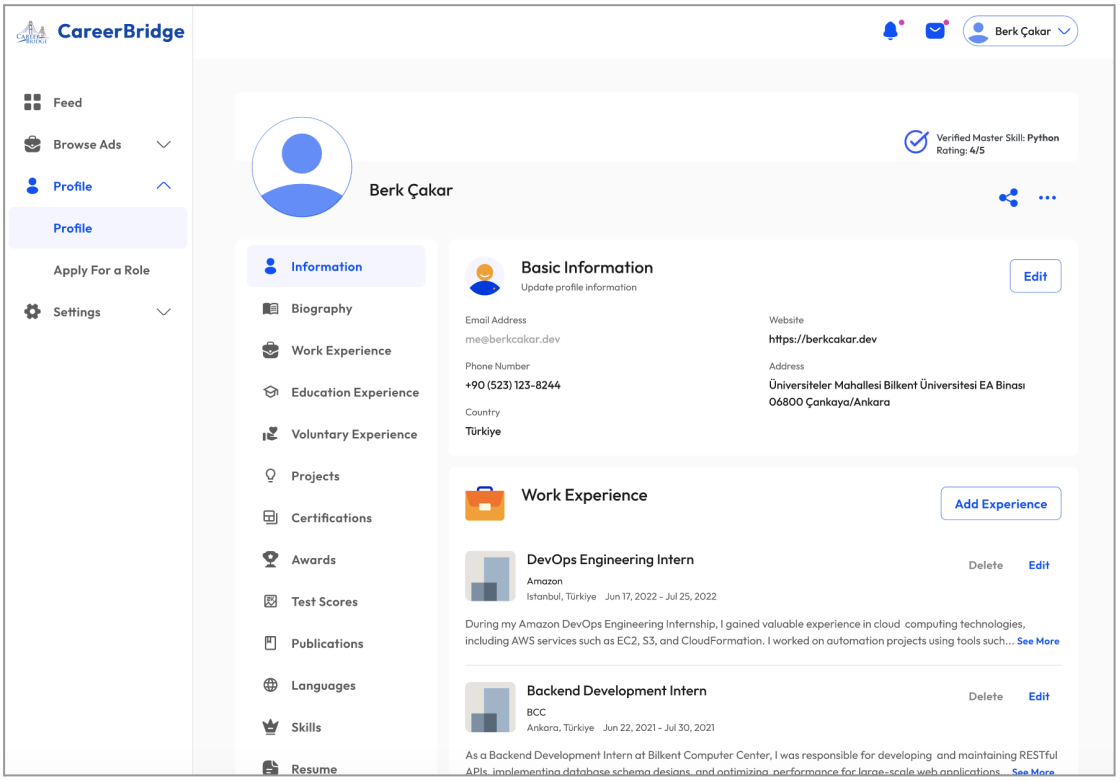

Figure 27

Inputs: All variables prefixed with '*' are inputs in this section.

Note that the queries to fetch all of the profile information are below although the mock-up demonstrates some part of it for simplicity.

SQL query for retrieving basic information:

```
SELECT *
FROM
  User U
  JOIN Profile P ON U.user_id = P.profile_id
WHERE
  U.user_id = @user_id;
```

SQL query to fetch the notifications of a user:

```
SELECT * FROM Notification WHERE user_id = @user_id;
```

SQL query to fetch the master skill of a user:

```
SELECT *
FROM
  Skill S,
  Profile P,
  SkillAssesment SA
WHERE
  S.profile_id = P.skill_id
  AND P.user_id = @user_id
  AND S.is_master_skill = 1
  AND SA.skill_id = S.skill_id
LIMIT 1;
```

SQL query to fetch the work experiences of a user:

```
SELECT *
FROM
  WorkExperience WE,
  Experience E,
  Profile P
WHERE
  WE.experience_id = E.experience_id
  AND E.profile_id = P.profile_id
  AND P.user_id = @user_id;
```

SQL query for retrieving biography of a user:

```
SELECT biography from Profile P where P.user_id = @user_id;
```

SQL query to fetch the educational experiences of a user:

```
SELECT *
FROM
  EducationalExperience EE,
  Experience E,
  Profile P
WHERE
  P.user_id = @user_id
  AND P.profile_id = E.profile_id
  AND E.experience_id = EE.experience_id;
```

SQL query to fetch the voluntary experiences of a user:

```
SELECT *
FROM
  VoluntaryExperience VE,
  Experience E,
  Profile P
WHERE
  P.user_id = @user_id
  and P.profile_id = E.profile_id
  AND E.experience_id = VE.experience_id;
```

SQL query to fetch the projects of a user:

```
SELECT *
FROM
  Project,
  Profile,
WHERE
  Profile.user_id = @user_id
  AND Profile.profile_id = Project.profile_id;
```

SQL query to fetch the certifications of a user:

```
SELECT *
FROM
  Certification C,
  Profile P
WHERE
  P.user_id = @user_id
  AND P.profile_id = C.profile_id;
```

SQL query to fetch the awards of a user:

```
SELECT *
FROM
  User U
  JOIN Profile P ON U.user_id = P.profile_id
  JOIN Awards A ON P.profile_id = A.profile_id
WHERE
  U.user_id = @user_id;
```

SQL query to fetch the test scores of a user:

```
SELECT *
FROM
  TestScore T,
  Profile P
WHERE
  P.user_id = @user_id
```

```
  AND P.profile_id = T.profile_id;
```

SQL query to fetch the language proficiencies of a user:

```
SELECT
  *
FROM
  LanguageProficiency L,
  Profile P
WHERE
  P.user_id = @user_id
  AND P.profile_id = L.profile_id;
```

SQL query to fetch the skills of a user:

```
SELECT *
FROM
  Skill S,
  SkillAssesment SA,
  Profile P
WHERE
  P.user_id = @user_id
  AND P.profile_id = S.profile_id
  AND S.skill_id = SA.skill_id;
```

SQL query to fetch the resume of a user:

```
SELECT resume FROM Profile P WHERE P.user_id = @user_id;
```

SQL query to delete an experience from a user:

```
DELETE FROM Experience WHERE experience_id = @experience_id;
```

SQL query to delete a certification from a user:

```
DELETE FROM Certification WHERE certification_id =
@certification_id;
```

SQL query to delete an award from a user:

```
DELETE FROM Certification WHERE award_id = @award_id;
```

SQL query to delete a test score from a user:

```
DELETE FROM TestScore WHERE test_score_id = @test_score_id;
```

SQL query to delete a publication from a user:

```
DELETE FROM Publication WHERE publication_id =
@publication_id;
```

SQL query to delete a language proficiency from a user:

```
DELETE FROM LanguageProficiency WHERE language_id =
@language_id;
```

SQL query to delete a skill from a user:

```
DELETE FROM Skill WHERE skill_id = @skill_id;
```

SQL query to delete the resume of a user:

```
UPDATE Profile SET resume = NULL WHERE Profile.profile_id =
@profile_id;
```

4.12.2 Messages Page (Request for Skill Assessment)

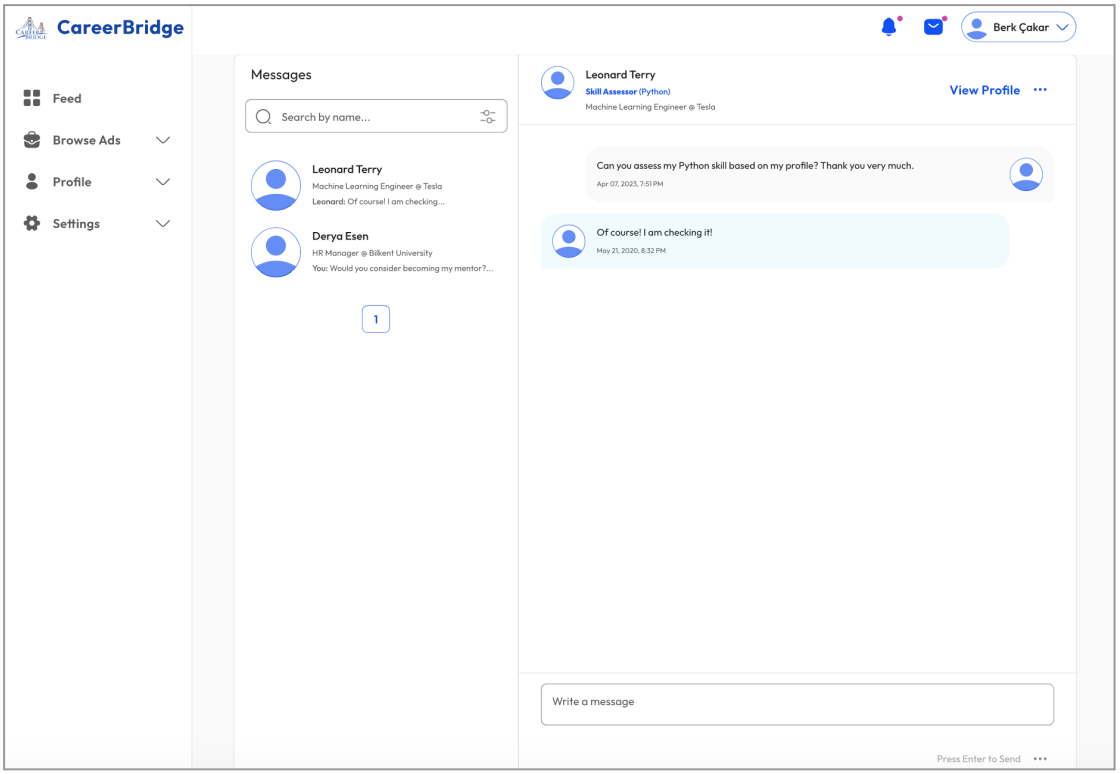

Figure 28

Inputs: @user_id, @receiver_id, @content, @file, @other_user_id, @skill_id, @rating

SQL query to send message when the thread already exists between two users:

```
INSERT INTO Message(
  thread_id, content, file, sent_time,
  from_thread_creator)
SELECT
  thread_id,
  @content,
  @file,
  NOW(),
  CASE WHEN creator_id = @user_id THEN TRUE ELSE FALSE END
FROM
  (SELECT
      thread_id,
      creator_id
    FROM
```

```
      Thread T
    WHERE
      T.creator_id = @user_id
      AND T.receiver_id = @receiver_id
    UNION
    SELECT
      thread_id,
      creator_id
    FROM
      Thread T
    WHERE
      T.creator_id = @receiver_id
      AND T.receiver_id = @user_id
  );
```

SQL query to send message when the thread does not exist between two users:

```
INSERT INTO Thread(creator_id, receiver_id)
VALUES
  (@user_id, @receiver_id);
```

Assume the ID of the thread created is @thread_id:

```
INSERT INTO Message(
  thread_id, content, file, sent_time,
  from_thread_creator
)
SELECT
  @thread_id,
  @content,
  @file,
  NOW(),
  TRUE;
```

SQL query to load conversation:

```
SELECT
  M.content,
  M.file,
  M.sent_time,
  M.has_read,
  M.from_thread_creator
FROM
  Message M JOIN Thread T ON M.thread_id = T.thread_id
WHERE
  (T.creator_id = @user_id
    AND T.receiver_id = @receiver_id)
  OR (
    T.creator_id = @receiver_id
```

```
    AND T.receiver_id = @sender_id
  );
UPDATE
  Message
SET
  has_read = TRUE
WHERE
  Message.message_id IN (
    SELECT M.message_id
    FROM
      Message M JOIN Thread T ON M.thread_id = T.thread_id
    WHERE
      (T.creator_id = @user_id
       AND T.receiver_id = @receiver_id)
      OR (
        T.creator_id = @receiver_id
        AND T.receiver_id = @sender_id)
      );
```

SQL query to load all threads for constructing the list of previous conversations:

```
SELECT
  U.user_id,
  U.first_name,
  U.last_name
FROM
  Thread T
  JOIN User U ON T.creator_id = U.user_id
WHERE
  T.receiver_id = @user_id
UNION
SELECT
  U.user_id,
  U.first_name,
  U.last_name
FROM
  Thread T
  JOIN User U ON T.receiver_id = U.user_id
WHERE
  T.creator_id = @user_id;
```

SQL query to retrieve the information about the other side of the thread:

```
SELECT
  User.first_name,
  User.last_name,
  AppUser.user_role,
  Experience.title,
```

```
  WorkExperience.company_name
FROM
  User
  INNER JOIN AppUser ON User.user_id = AppUser.user_id
  INNER JOIN Profile ON User.user_id = Profile.user_id
  INNER JOIN Experience ON Profile.profile_id =
Experience.profile_id
  LEFT OUTER JOIN WorkExperience ON Experience.experience_id =
WorkExperience.experience_id
WHERE
  Profile.is_application_specific IS FALSE
  AND User.id = @other_user_id
  AND end_date IS NULL
  AND status = “Working”
ORDER BY
  start_date DESC
LIMIT
  1;
```

SQL query for Skill Assessors to assess a skill of the user:

```
UPDATE SkillAssessment
SET rating = @rating
WHERE skill_id = @skill_id;
```

4.12.3 Messages Page (Request for Mentorship & Appointment)

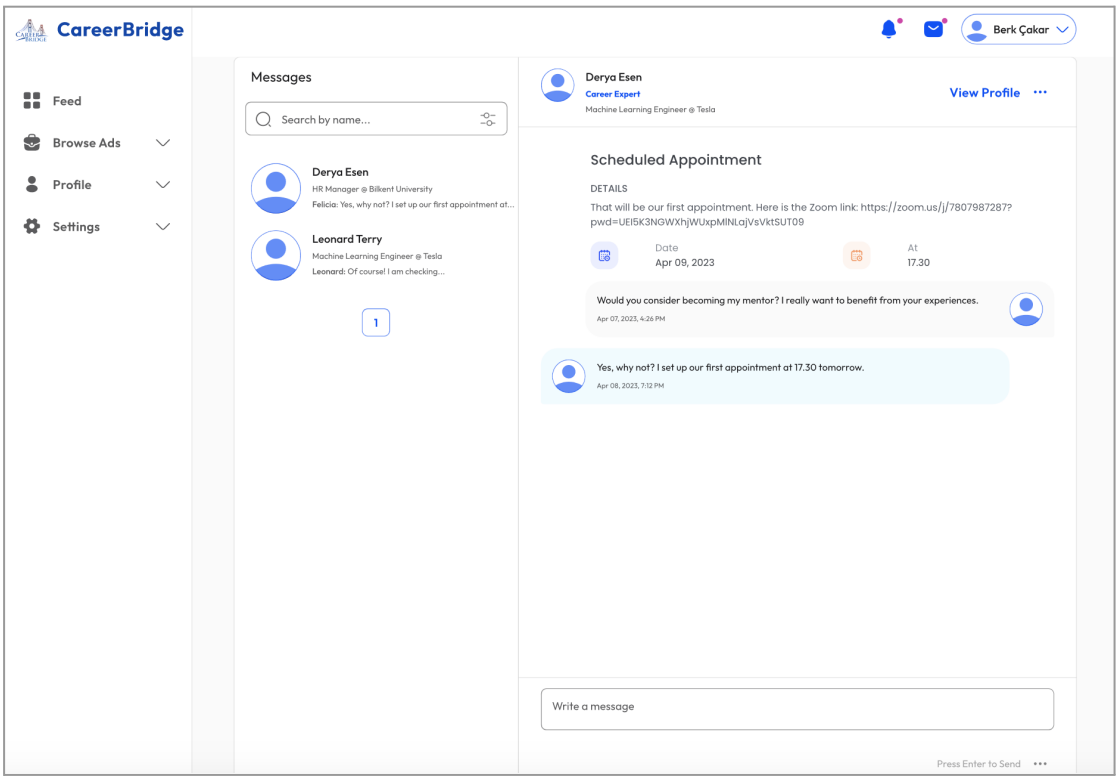

Figure 29

Inputs: @user_id, @receiver_id, @thread_id, @date, @details, @response

SQL query to create appointment (appears when clicked on the three dots):

```
INSERT INTO Appointment(thread_id, date, details)
VALUES(@thread_id, @date, @details);
```

SQL query to retrieve existing appointment:

```
SELECT *
FROM Appointment
WHERE thread_id = @thread_id AND date = @date;
```

SQL query to send mentorship request:

```
INSERT INTO Mentorship(mentor_id, mentee_id)
VALUES (@user_id, @receiver_id);
```

SQL query to respond to mentorship request:

```
UPDATE Mentorship
SET response = @response
WHERE mentee_id = @user_id AND mentor_id = @receiver_id;
```

4.12.4 Apply For a Role Page

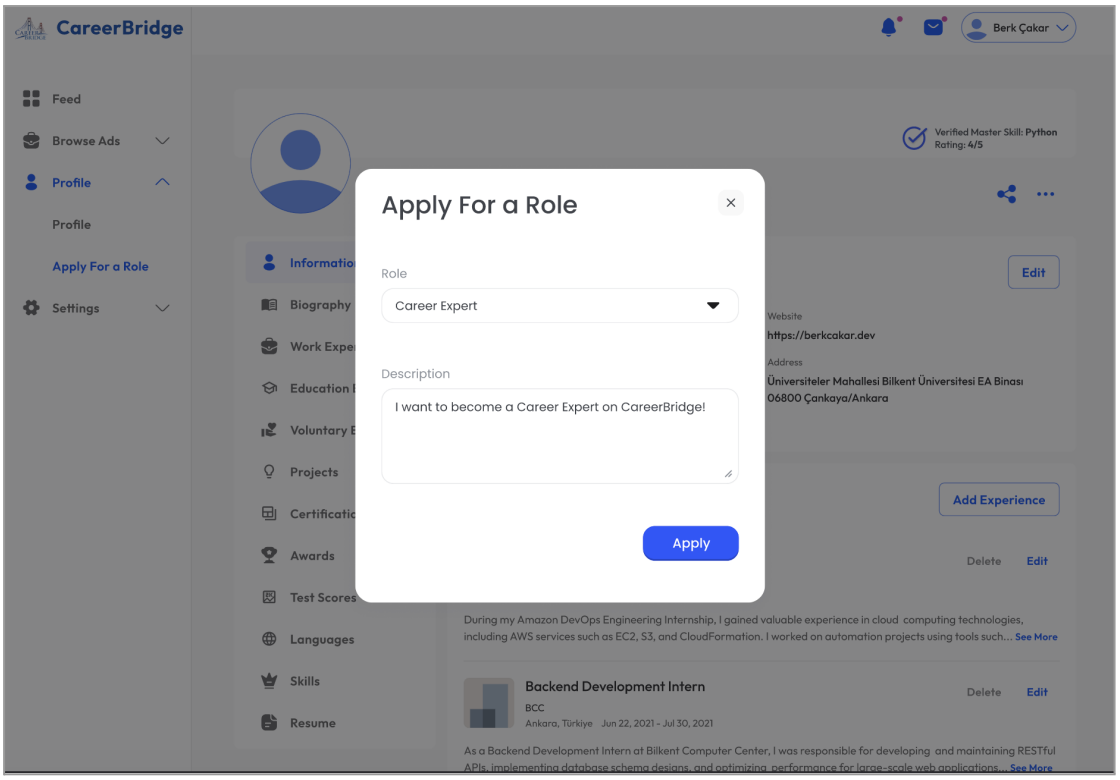

Figure 30

Inputs: @user_id, @role, @description

SQL query to apply for a role:

```
INSERT INTO Application(
  description, application_type, application_date)
SELECT @description,
  @role,
  GETDATE();
```

Assume the ID of application created is @application_id:

```
INSERT INTO ApplicationReview(@application_id)
VALUES(@application_id);
```

SQL query to approve or reject an application for administrators (assume the ID of the administrator who makes the decision is @admin_id, and the response is @response):

```
UPDATE ApplicationReview
SET admin_id = @admin_id,
  response = @response
WHERE application_id = @application_id;
```

4.12.5 Send a Connection Request

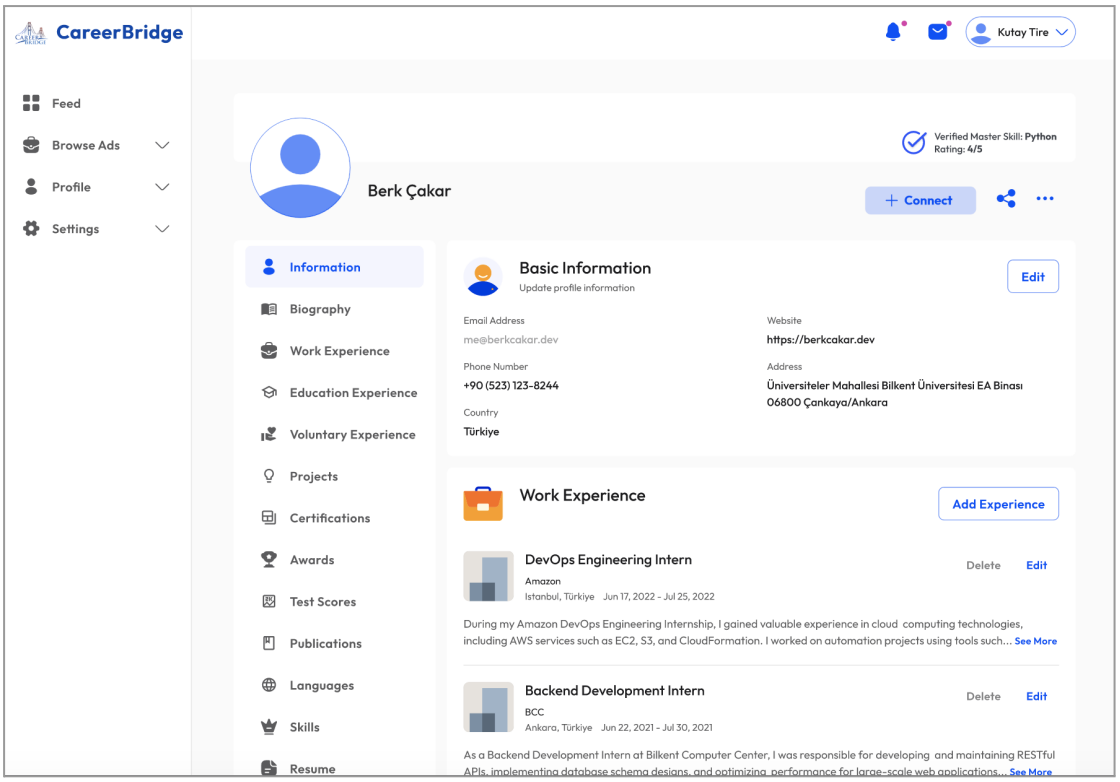

Figure 31

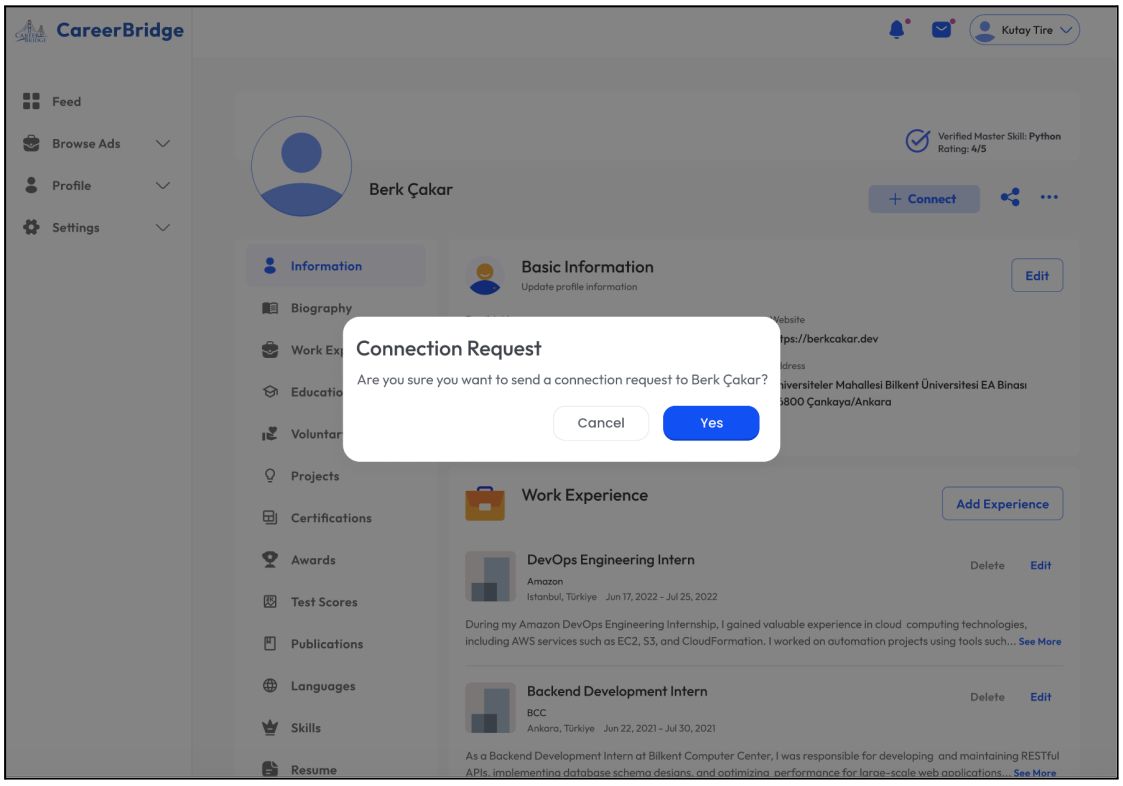

Figure 32

Inputs: @user_id, @receiver_id

SQL query to create a connection between on click Yes:

```
INSERT INTO Connection(sender_id, receiver_id)
VALUES (@user_id, @receiver_id);
```

4.12.6 Accept or Reject a Connection Request

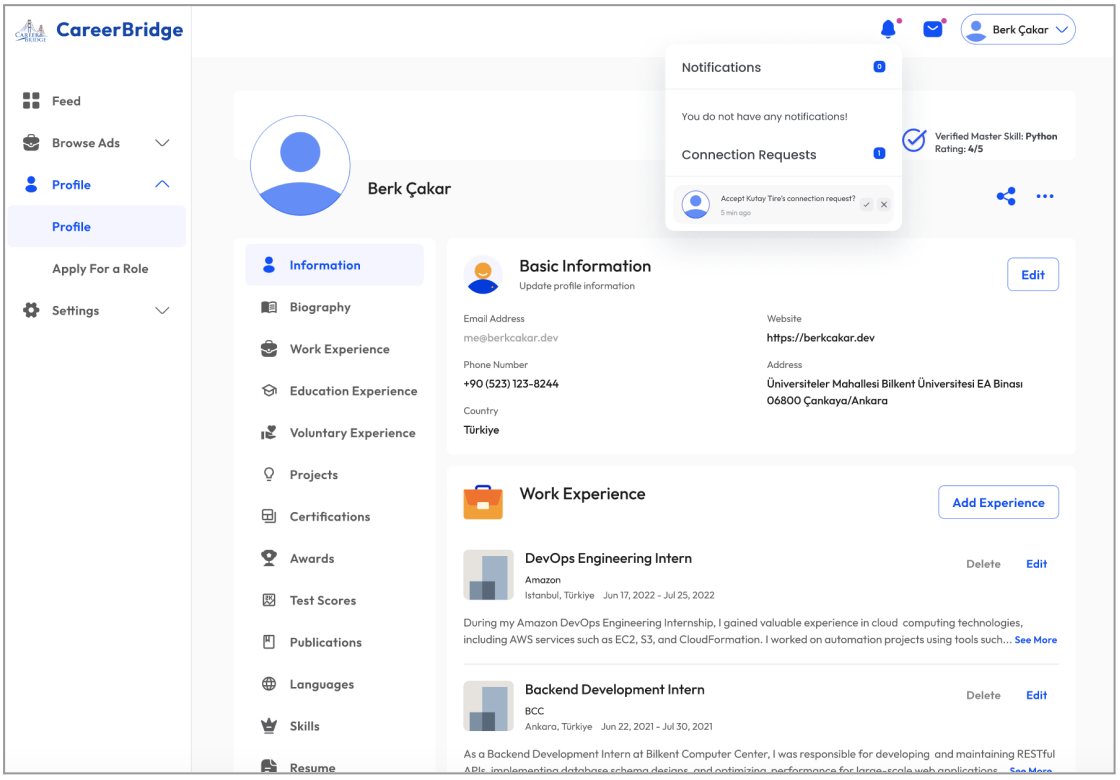

Figure 33

Inputs: @user_id, @sender_id

SQL query to accept connection request for a user:

```
UPDATE Connection
SET response = 1
WHERE
  sender_id = @sender_id
  AND receiver_id = @user_id;
```

SQL query to reject connection request for a user:

```
UPDATE Connection
SET response = 0
WHERE
  sender_id = @sender_id
  AND receiver_id = @user_id;
```

4.12.7 User Feed

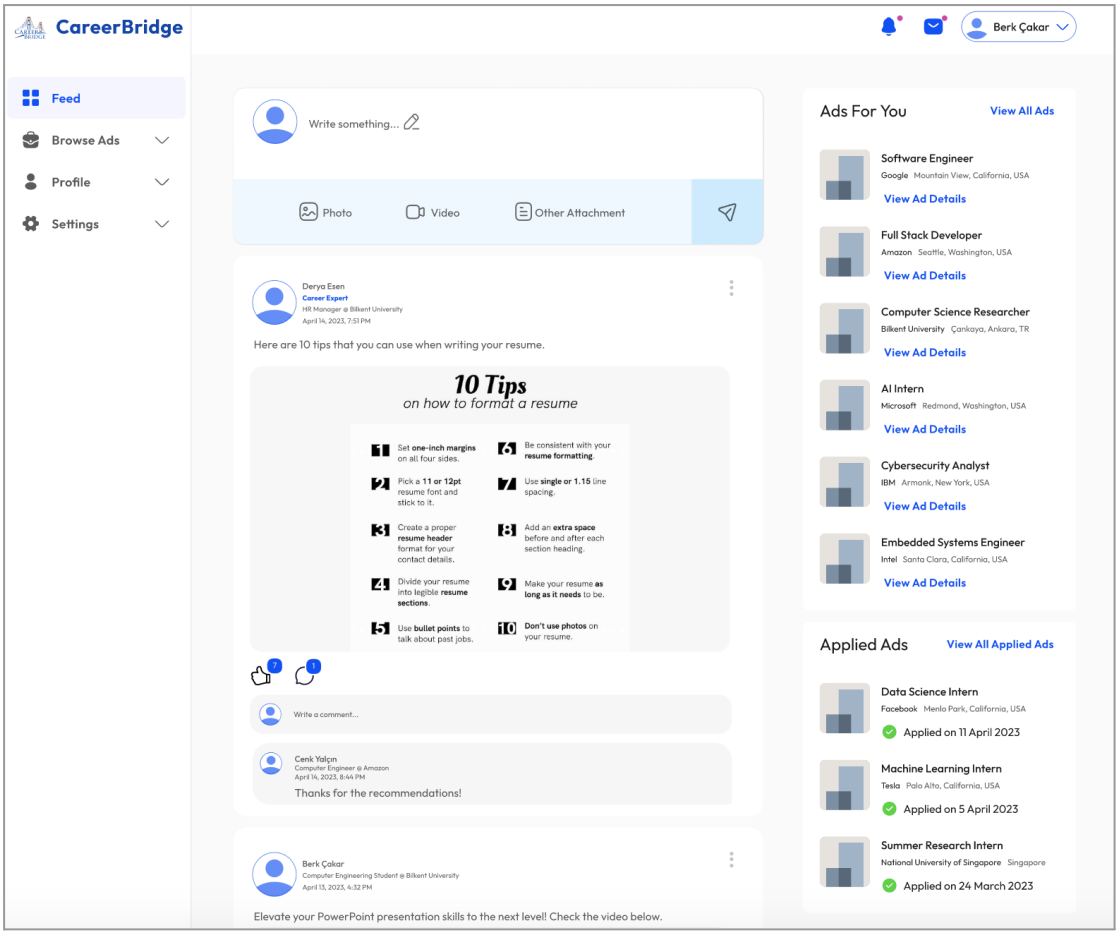

Figure 34

Inputs: @user_id, @attachement, @content_post, @post_id, @content_comment

SQL query to retrieve the user feed:

```
WITH Connected AS (
  SELECT sender_id
  FROM Connection C
  WHERE
    C.receiver_id = @user_id
    AND C.response IS TRUE
  UNION
  SELECT receiver_id
  FROM Connection C
  WHERE
    C.sender_id = @user_id
    AND C.response IS TRUE
)
SELECT *
FROM Post P,
```

```
  Comment C
WHERE
  P.user_id = @user_id
  OR P.user_id IN Connected
  AND C.post_id = P.post_id
ORDER BY
  P.post_date DESC;
```

SQL query to retrieve the applied ads:

```
SELECT
  J.title,
  J.organization,
  J.location,
  JR.apply_date
FROM
  JobAdvertisementResponse JR,
  JobAdvertisement J
WHERE
  JR.ad_id = J.ad_id
  AND JR.user_id = @user_id
ORDER BY JR.apply_date DESC
LIMIT 3;
```

SQL query to retrieve the ads for you:

```
SELECT
  J.title,
  J.organization,
  J.location
FROM JobAdvertisement J
WHERE
  (SELECT S.skill_id
    FROM Skill S,
      Profile P
    WHERE
      S.profile_id = P.profile_id
      AND P.user_id = @user_id
  ) IN (
    SELECT S2.skill_id
    FROM
      Skill S2,
      SkillInJobAdvertisement SIJ
    WHERE
      S2.skill_id = SIJ.skill_id
      AND J.ad_id = SIJ.ad_id
  );
```

SQL query to create a post:

```
INSERT INTO Post (
  user_id, content, attachment, post_date)
VALUES
  (@user_id, @content_post, @attachment,
    NOW()
  );
```

SQL query to create a comment on a post:

```
INSERT INTO Comment (
  post_id, user_id, content, commented_at)
VALUES
  (@post_id, @user_id, @content_comment,
    NOW()
  );
```

4.12.8 Admin Panel

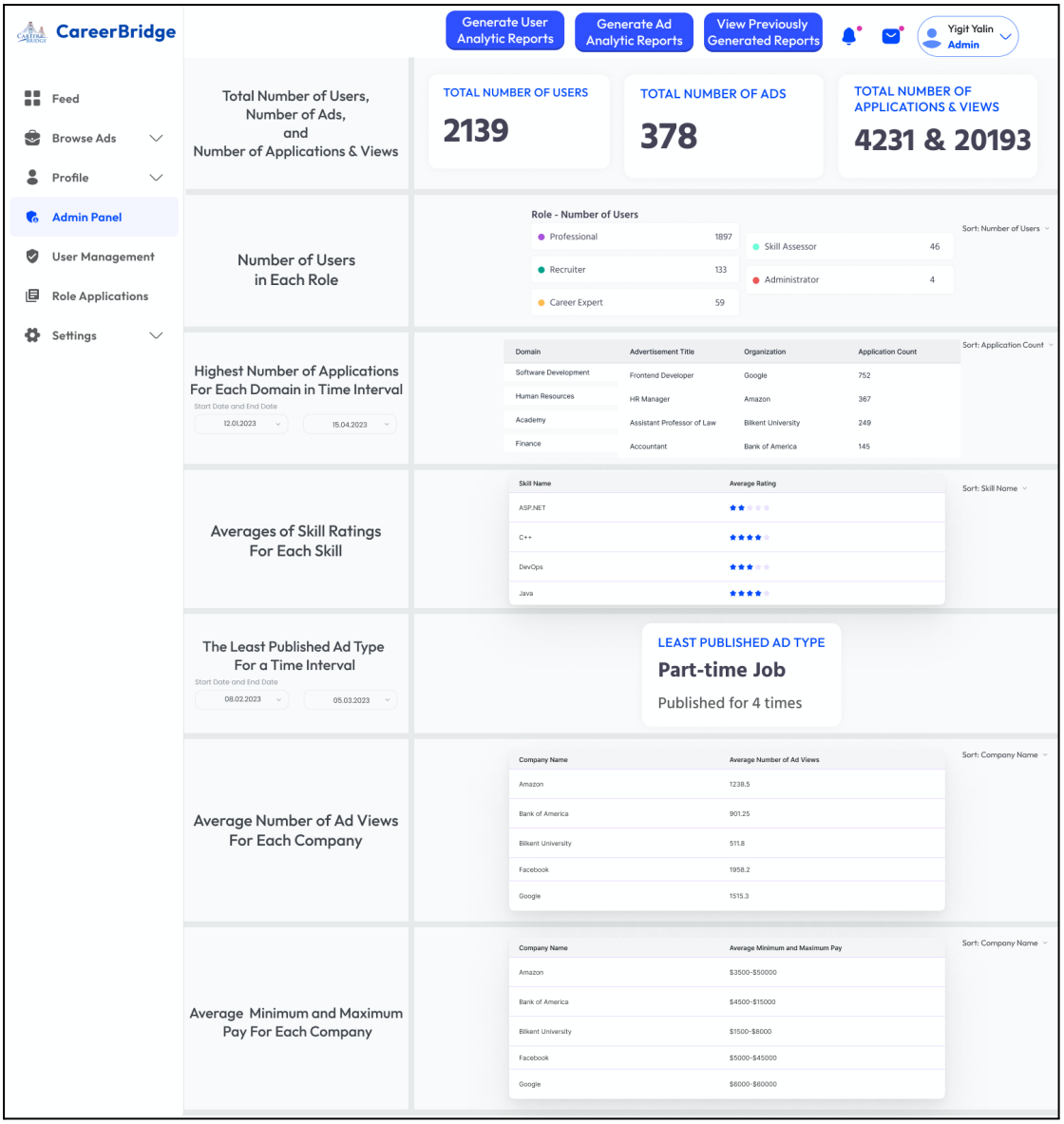

Figure 35

SQL query to retrieve total number of users, total number of ads, total number of ad applications, and total number of ad views in CareerBridge:

```
SELECT
  COUNT(*) AS user_count
FROM
  User;
SELECT
  COUNT(*) AS ad_count,
  SUM(view_count) as ad_view_count,
  SUM(application_count) as ad_application_count
FROM
```

```
  JobAdvertisement;
```

SQL query to retrieve the total number of users for each role in CareerBridge (sorting according to role name or number of users is possible):

```
SELECT
  user_role,
  COUNT(*) AS user_count
FROM
  (
    SELECT
      user_role
    FROM
      AppUser
    UNION ALL
    SELECT
      'Admin' AS user_role
    FROM
      Admin
  ) AS combined_table
GROUP BY
  user_role
ORDER BY
  user_count DESC;
```

SQL query to retrieve the most applied ad for each domain in a given time interval (sorting according to domain, advertisement title, company name, or application count is possible):

```
WITH domain_counts AS (
  SELECT
    JA.domain_name AS domain_name,
    JA.ad_id AS ad_id,
    COUNT(*) as application_count
  FROM
    JobAdvertisement JA,
    JobAdvertisementResponse JAR
  WHERE
    JA.ad_id = JAR.ad_id
    AND JAR.apply_date BETWEEN @start_date
    AND @end_date
  GROUP BY
    JA.domain_name,
    JA.ad_id
)
SELECT
  DC.domain_name,
  DC.ad_id,
  JA.title,
```

```
  JA.organization,
  DC.application_count
FROM
  domain_counts DC,
  JobAdvertisement JA
WHERE
  JA.ad_id = DC.ad_id
  AND DC.application_count = (
    SELECT
      MAX(application_count)
    FROM
      domain_counts
  )
ORDER BY
  DC.application_count DESC;
```

SQL query to retrieve the average rating of available skills in CareerBridge (sorting according to skill name or average rating is possible):

```
SELECT
  S.skill_name AS skill_name,
  AVG(SA.rating) AS average_rating
FROM
  Skill S,
  SkillAssesment SA
WHERE
  S.skill_id = SA.skill_id
GROUP BY
  S.skill_name
ORDER BY
  S.skill_name ASC;
```

SQL query to retrieve the least applied type of ad in a given time interval:

```
WITH ad_counts AS (
  SELECT
    JA.ad_type AS ad_type,
    COUNT(*) as application_count
  FROM
    JobAdvertisement JA,
    JobAdvertisementResponse JAR
  WHERE
    JA.ad_id = JAR.ad_id
    AND JAR.apply_date BETWEEN @start_date
    AND @end_date
  GROUP BY
    JA.ad_type
)
```

```
SELECT
  AC.ad_type,
  AC.application_count
FROM
  ad_counts AC
WHERE
  AC.application_count = (
    SELECT
      MIN(application_count)
    FROM
      ad_counts
  );
```

SQL query to retrieve the average number of job advertisement view for each company (sorting according to average view count or company name is possible):

```
SELECT
  C.company_name,
  AVG(JA.view_count) AS average_view_count
FROM
  Company C,
  JobAdvertisement JA
WHERE
  C.company_name = JA.organization
GROUP BY
  C.company_name
ORDER BY
  C.company_name ASC;
```

SQL query to retrieve the average pay range for job advertisements for each company (sorting according to company name is possible):

```
SELECT
  C.company_name,
  AVG(JA.pay_range_min) AS average_min_pay,
  AVG(JA.pay_range_mix) AS average_max_pay
FROM
  Company C,
  JobAdvertisement JA
WHERE
  C.company_name = JA.organization
GROUP BY
  C.company_name
ORDER BY
  C.company_name ASC;
```

In order to store the generated reports for user analytics and ad analytics the following queries will be executed upon the click on

“Generate Report” button. A description will be provided by the user which is @description, and assume the ID of the admin who is generating the report is @admin_id:

SQL query to generate a system report of user analytics:

```
INSERT INTO SystemReport (report_type, content, description,
creation_date, admin_id)
VALUES (
  'User Analytics',
  JSON_OBJECT(
    'total_users', (
      SELECT COUNT(*) AS user_count FROM User
    ),
    'users_by_role', (
      SELECT JSON_ARRAYAGG(
        JSON_OBJECT('role', user_role, 'count', user_count)
      )
      FROM (
        SELECT user_role, COUNT(*) AS user_count
        FROM (SELECT user_role FROM AppUser UNION ALL SELECT
'Admin' AS user_role FROM Admin) AS combined_table
        GROUP BY user_role
      ) AS user_counts
    ),
    'average_skill_ratings', (
      SELECT JSON_ARRAYAGG(
        JSON_OBJECT('skill_name', skill_name,
'average_rating', average_rating)
      )
      FROM (
        SELECT S.skill_name AS skill_name, AVG(SA.rating) AS
average_rating
        FROM Skill S
        INNER JOIN SkillAssessment SA ON S.skill_id =
SA.skill_id
        GROUP BY S.skill_name
      ) AS skill_ratings
    )
  ),
  @description,
  NOW(),
  @admin_id
);
```

SQL Query for generating a system report of ad analytics:

```
INSERT INTO SystemReport (report_type, content,description,
creation_date, admin_id)
VALUES (
```

```
  'Ad Analytics',
  JSON_OBJECT(
    'most_applied_ads_by_domain', (
      WITH domain_counts AS (
        SELECT JA.domain_name AS domain_name, JA.ad_id AS
ad_id, COUNT(*) as application_count FROM JobAdvertisement JA,
JobAdvertisementResponse JAR WHERE JA.ad_id = JAR.ad_id AND
JAR.apply_date BETWEEN @start_date AND @end_date GROUP BY
JA.domain_name, JA.ad_id
      )
      SELECT DC.domain_name, DC.ad_id, JA.title,
DC.application_count FROM domain_counts DC, JobAdvertisement
JA WHERE JA.ad_id = DC.ad_id AND DC.application_count =
(SELECT MAX(application_count) FROM domain_counts)
    ),
    'least_applied_ad_type', (
      WITH ad_counts AS (
        SELECT JA.ad_type AS ad_type, COUNT(*) as
application_count FROM JobAdvertisement JA,
JobAdvertisementResponse JAR WHERE JA.ad_id = JAR.ad_id AND
JAR.apply_date BETWEEN @start_date AND @end_date GROUP BY
JA.ad_type
      )
      SELECT AC.ad_type, AC.application_count FROM ad_counts
AC WHERE AC.application_count = (SELECT MIN(application_count)
FROM ad_counts)
    ),
    'average_job_ad_views_per_company', (
      SELECT C.company_name, AVG(JA.view_count) AS
average_view_count FROM Company C, JobAdvertisement JA WHERE
C.company_name = JA.organization GROUP BY C.company_name
    ),
    'average_pay_range_per_company', (
      SELECT C.company_name, AVG(JA.min_pay) AS
average_min_pay, AVG(JA.max_pay) AS average_max_pay FROM
Company C, JobAdvertisement JA WHERE C.company_name =
JA.organization GROUP BY C.company_name
    )
  ),
  @description,
  NOW(),
  @admin_id
);
```