Bilkent University

Department of Computer Engineering

CS 353 Term Project

CSCareer: A Quiz Based Hiring System

Design Report

Group 22: Hamza PEHLİVAN, Meryem EFE, Fırat YÖNAK, Selen UYSAL

Instructor: Özgür ULUSOY

Teaching Assistant: Arif USTA

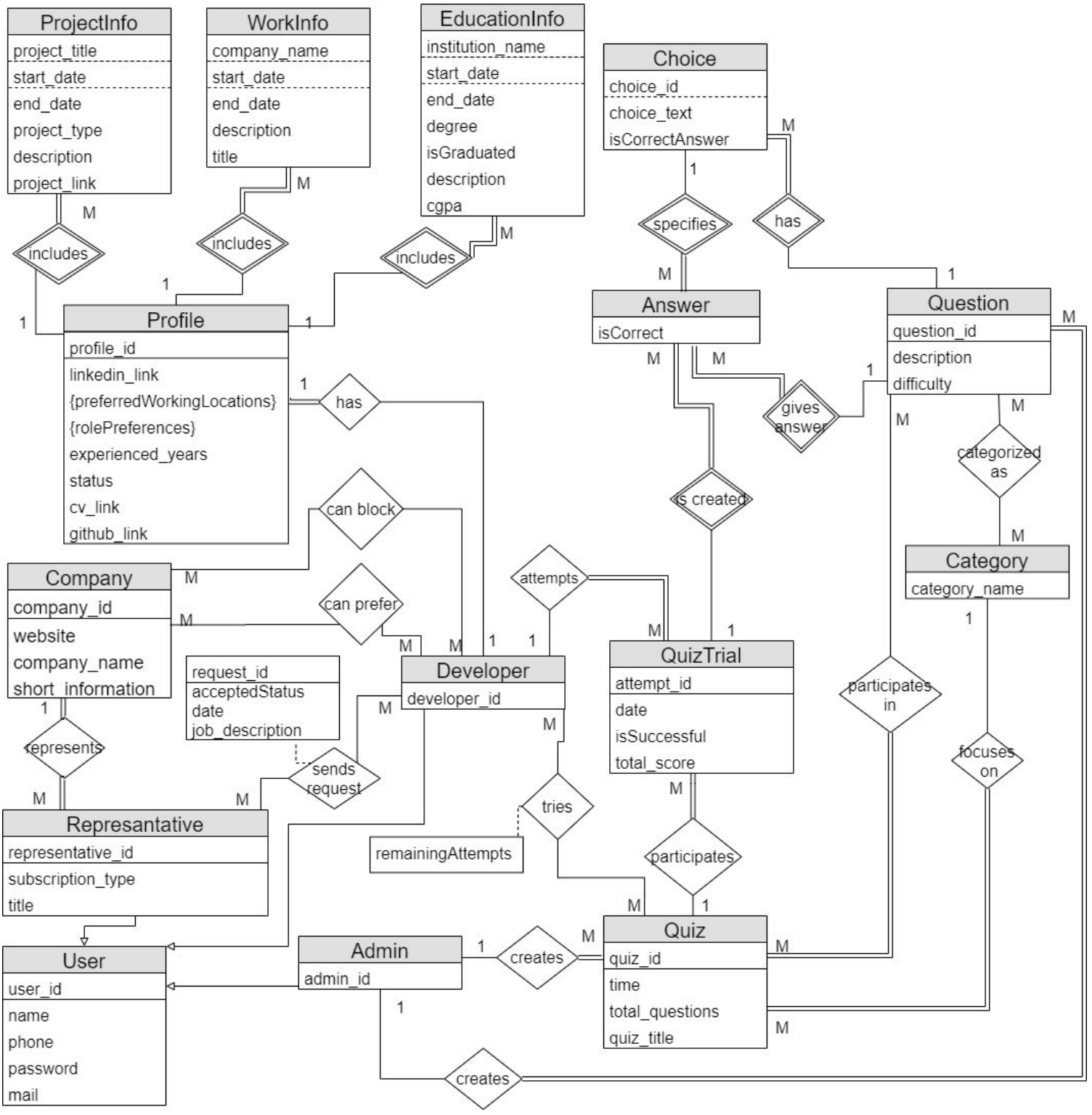
Contents

Design Report

CSCareer: A Quiz Based Hiring System

1. Revised E/R Model

- Various mistakes related to full participation are corrected.
- Account entity is replaced with User entity. Developer, Representative and Admin inherit from User.
- Date attribute is added to Send Request relation.
- Choice entity is added as a weak entity. That is because, choices cannot exist without a question.
- Category entity is added.
- Categorized As relation is added between Question and Category entities. There is many to many relation because a question can be included in many categories. Also, it is required that each question should have at least one category.
- QuizResult entity is removed and its attributes added to the QuizTrial entity.
- New attributes are added to the Quiz entity such as time and quiz title.
- Weak entity Answer is added between QuizTrial, Question and Choice entities. That is because, which answers given to which questions in a particular attempt should be addressed.
- Tries relation is added between Developer and Quiz entities to keep track of remaining attempts for a certain quiz.
- Company blacklist and preferred companies relations are added.


ProjectInfo
project_title
start_date
end_date
project_type
description
project_link
WorkInfo
company_name
start_date
end_date
description
title
EducationInfo
institution_name
start_date
end_date
degree
isGraduated
description
cgpa
Choice
choice_id
choice_text
isCorrectAnswer
includes
includes
includes
specifies
has
Profile
profile_id
linkedin_link
{preferredWorkingLocations}
{rolePreferences}
experienced_years
status
cv_link
github_link
Answer
isCorrect
Question
question_id
description
difficulty
has
gives answer
is created
categorized as
Category
category_name
can block
Company
company_id
website
company_name
short_information
can prefer
attempts
request_id
acceptedStatus
date
job_description
Developer
developer_id
QuizTrial
attempt_id
date
isSuccessful
total_score
participates in
represents
sends request
tries
focuses on
Represantative
representative_id
subscription_type
title
remainingAttempts
participates
User
user_id
name
phone
password
mail
Admin
admin_id
creates
Quiz
quiz_id
time
total_questions
quiz_title
creates

2. Relation Schemas

2.1. User

Relational Model:

User (user_id, email, password, fullname, phone)

Functional Dependencies:

user_id → email, password, fullname, phone

email → user_id, password, fullname, phone

Candidate Keys:

{ (user_id), (email) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE User(
    user_id     INT PRIMARY KEY AUTO_INCREMENT,
    email       VARCHAR(45) NOT NULL UNIQUE,
    password    VARCHAR(45) NOT NULL,
    fullname    VARCHAR(45) NOT NULL,
    phone       VARCHAR(10) );
```

2.2. Developer

Relational Model:

Developer (developer_id, profile_id)

Functional Dependencies:

developer_id → profile_id

profile_id → developer_id

Candidate Keys:

{ (developer_id), (profile_id) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Developer(
    developer_id        INT PRIMARY KEY,
    profile_id          INT NOT NULL,
    FOREIGN KEY (developer_id) REFERENCES User (user_id)
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

2.3. Representative

Relational Model:

Representative (representative_id, subscription_type, title, company_id)

Functional Dependencies:

representative_id → subscription_type, title, company_id

Candidate Keys:

{ (representative_id) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Representative(
    representative id       INT PRIMARY KEY,
    subscription_type       VARCHAR(5) NOT NULL,
    title                   VARCHAR(45) NOT NULL,
    company_id              INT NOT NULL,
```

FOREIGN KEY (representative_id) REFERENCES User (user_id),

FOREIGN KEY (company_id) REFERENCES Company (company_id),

CHECK (subscription_type IN (‘month’, ‘year’)));

2.4. Company

Relational Model:

Company (company_id, website, company_name, short_information)

Functional Dependencies:

company_id → website, company_name, short_information

website → company_id, company_name, short_information

Candidate Keys:

{ (company_id), (website) }

Normal Form:

BCNF

Table Definition:

CREATE TABLE Company(

company_id	INT PRIMARY KEY AUTO_INCREMENT,
website	VARCHAR(45) NOT NULL UNIQUE,
company_name	VARCHAR(45) NOT NULL,
short_information	TEXT(200) NOT NULL);

2.5. Admin

Relational Model:

Admin (admin_id)

Functional Dependencies:

None

Candidate Keys:

{ (admin_id) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Admin(
    admin_id            INT PRIMARY KEY,
    FOREIGN KEY (admin_id) REFERENCES User (user_id) );
```

2.6. Profile

Relational Model:

Profile (profile_id, linkedin_link, experienced_years, status, cv_link, github_link)

Functional Dependencies:

profile_id → linkedin_link, experienced_years, status, cv_link, github_link

Candidate Keys:

({profile_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Profile (
    profile_id          INT PRIMARY KEY AUTO_INCREMENT,
    linkedin_link       VARCHAR(50) UNIQUE,
    experienced_years   INT,
```

```
    status              TINYINT(1) NOT NULL,
    cv_link             VARCHAR(50) UNIQUE,
    github_link         VARCHAR(50) UNIQUE );
```

2.7. PreferredWorkingLocations

Relational Model:

PreferredWorkingLocations (developer_id, city)

Functional Dependencies:

None

Candidate Keys:

{ (developer_id, city) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE PreferredWorkingLocations(
    developer_id        INT PRIMARY KEY,
    city                VARCHAR(15) PRIMARY KEY,
    FOREIGN KEY (developer_id) REFERENCES Developer (developer_id) );
```

2.8. PreferredCompanies

Relational Model:

PreferredCompanies (developer_id, company_id)

Functional Dependencies:

None

Candidate Keys:

{ (developer_id, company_id) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE PreferredCcompanies(
    developer_id        INT PRIMARY KEY,
    company_id          INT PRIMARY KEY,
    FOREIGN KEY (developer_id) REFERENCES Developer (developer_id),
    FOREIGN KEY (company_id) REFERENCES Company (company_id) );
```

2.9. RolePreferences

Relational Model:

RolePreferences (profile_id, position)

Functional Dependencies:

None

Candidate Keys:

{ (profile_id, position) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE PreferredWorkingLocations(
    profile_id      INT PRIMARY KEY,
    position        VARCHAR(30) PRIMARY KEY,
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

2.10. CompanyBlackList

Relational Model:

CompanyBlackList (profile_id, company_id)

Functional Dependencies:

None

Candidate Keys:

{ (profile_id, company_id) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE CompanyBlackList (
    profile_id      INT PRIMARY KEY,
    company_id  INT PRIMARY KEY,
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id),
    FOREIGN KEY (company_id) REFERENCES Company (company_id) );
```

2.11. ProjectInfo

Relational Model:

ProjectInfo (profile_id, project_title, start_date, end_date, project_type, description, project_link)

Functional Dependencies:

profile_id, project_title, start_date → end_date, project_type, description, project_link

Candidate Keys:

({profile_id, project_title, start_date })

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE ProjectInfo (
    profile_id          INT PRIMARY KEY,
    project_title       VARCHAR(30) PRIMARY KEY,
    start_date          DATE PRIMARY KEY,
    end_date            DATE,
    project_type        VARCHAR(20),
    description         VARCHAR(100) NOT NULL,
    project_link        VARCHAR(100),
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

2.12. WorkInfo

Relational Model:

WorkInfo (profile_id ,company_name, start_date, end_date, title, description)

Functional Dependencies:

profile_id, company_name, start_date → title, end_date, description

Candidate Keys:

({profile_id , company_name, date})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE WorkInfo (
    profile_id          INT PRIMARY KEY,
    company_name        VARCHAR(45) PRIMARY KEY,
    start_date          DATE PRIMARY KEY,
```

```
    end_date            DATE,
    title               VARCHAR(20) NOT NULL,
    description         VARCHAR(100) NOT NULL,
    FOREIGN KEY (profile_id) REFERENCES profile (profile_id) );
```

2.13. EducationInfo

Relational Model:

EducationInfo (profile_id, instution_name, start_date, end_date, degree, cgpa, isGraduated, description)

Functional Dependencies:

profile_id, institution_name, start_date → end_date, degree, cgpa, isGraduated, description

Candidate Keys:

({profile_id, institution_name, date})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE WorkInfo (
    profile_id          INT PRIMARY KEY,
    institution_name    VARCHAR(45) PRIMARY KEY,
    start_date          DATE PRIMARY KEY,
    end_date            DATE,
    isGraduted          TINYINT(1) NOT NULL,
    cgpa                DOUBLE NOT NULL,
    description         VARCHAR(100),
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

2.14. Question

Relational Model:

Question (question_id, admin_id, description, difficulty)

Functional Dependencies:

question_id → admin_id, description, difficulty

description → question_id, admin_id, difficulty

Candidate Keys:

({question_id}, {description})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE question (
    question_id         INT PRIMARY KEY AUTO_INCREMENT,
    admin_id            INT NOT NULL,
    description         varchar(100) NOT NULL,
    difficulty          ENUM('easy','medium','hard') NOT NULL,
    UNIQUE KEY description_UNIQUE (description),
    UNIQUE KEY question_id_UNIQUE (question_id),
    FOREIGN KEY (admin_id) REFERENCES admin (admin_id) );
```

2.15. Choice

Relational Model:

Choice (question_id, choice_id, choice_text, isCorrectAnswer)

Functional Dependencies:

question_id, choice_id → choice_text, isCorrectAnswer

Candidate Keys:

({question_id, choice_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE choice (
    choice_id          INT PRIMARY KEY,
    question_id        INT PRIMARY KEY,
    choice_text        VARCHAR(45) NOT NULL,
    isCorrectAnswer    TINYINT(1) NOT NULL,
    FOREIGN KEY (question_id) REFERENCES question (question_id) );
```

2.16. Category

Relational Model:

Category(category_name)

Functional Dependencies:

None

Candidate Keys:

({category_name})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE category (
    category_name VARCHAR(15) PRIMARY KEY );
```

2.17. CategorizedAs

Relational Model:

CategorizedAs (category_name,question_id)

Functional Dependencies:

None

Candidate Keys:

({category_name,question_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE categorized_as (
    category_name       VARCHAR(15) PRIMARY KEY,
    question_id         INT PRIMARY KEY,
    FOREIGN KEY (category_name) REFERENCES category (category_name),
    FOREIGN KEY (question_id) REFERENCES question (question_id) );
```

2.18. Quiz

Relational Model:

Quiz (quiz_id, admin_id, category_name, total_questions, quiz_title, time)

Functional Dependencies:

quiz_id→ admin_id, category_name, total_questions, quiz_title, time

Candidate Keys:

({quiz_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE quiz (
    quiz_id             INT PRIMARY KEY AUTO_INCREMENT,
    admin_id            INT NOT NULL,
    category_name       VARCHAR(15) NOT NULL,
    total_questions     INT NOT NULL,
    quiz_title          VARCHAR(10) NOT NULL,
    time                TIME NOT NULL,
    UNIQUE KEY quiz_id_UNIQUE (quiz_id),
    FOREIGN KEY (admin_id) REFERENCES admin (admin_id),
    FOREIGN KEY (category_name) REFERENCES category (category_name));
```

2.19. Quiz_questions

Relational Model:

Quiz_questions (quiz_id, question_id)

Functional Dependencies:

None

Candidate Keys:

({quiz_id, question_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE quiz_questions (
    quiz_id             INT PRIMARY KEY,
    question_id         INT PRIMARY KEY,
    FOREIGN KEY (question_id) REFERENCES question (question_id),
```

FOREIGN KEY (quiz_id) REFERENCES quiz (quiz_id));

2.20. QuizTrial

Relational Model:

QuizTrial (attempt_id, developer_id, quiz_id, date, isSuccessful, total_score)

Functional Dependencies:

attempt_id → developer_id, quiz_id, date, isSuccessful, total_score

Candidate Keys:

({attempt_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE quiztrial (
    attempt_id          INT PRIMARY KEY AUTO_INCREMENT,
    developer_id        INT NOT NULL,
    quiz_id             INT NOT NULL,
    date                DATE NOT NULL,
    isSuccessful        TINYINT(1),
    total_score         INT,
    UNIQUE KEY attempt_id_UNIQUE (attempt_id),
    FOREIGN KEY (developer_id) REFERENCES developer (developer_id),
    FOREIGN KEY (quiz_id) REFERENCES quiz (quiz_id) );
```

2.21. Answer

Relational Model:

Answer(attempt_id,question_id,choice_id, isCorrect)

Functional Dependencies:

attempt_id, question_id, choice_id→ isCorrect

Candidate Keys:

({attempt_id, question_id, choice_id})

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE answer (
    attempt_id          INT PRIMARY KEY,
    question_id         INT PRIMARY KEY,
    choice_id           INT PRIMARY KEY,
    iscorrect           TINYINT(1) NOT NULL,
    FOREIGN KEY (attempt_id) REFERENCES quiztrial (attempt_id),
    FOREIGN KEY (question_id) REFERENCES question (question_id),
    FOREIGN KEY (choice_id) REFERENCES choice(choice_id) );
```

2.22. Tries

Relational Model:

Tries (developer_id,quiz_id, remainingAttempts)

Functional Dependencies:

developer_id,quiz_id → remainingAttemps

Candidate Keys:

({developer_id, quiz_id})

Normal Form:

BCNF

Table Definition:

CREATE TABLE tries(

developer_id INT PRIMARY KEY,

quiz_id INT PRIMARY KEY,

remaining_attemps INT NOT NULL DEFAULT 3,

FOREIGN KEY (developer_id) REFERENCES developer (developer_id)

FOREIGN KEY (quiz_id) REFERENCES quiz (quiz_id));

2.23. Request

Relational Model:

Request (request_id, acceptedStatus, date, job_description, developer_id, representative_id)

Functional Dependencies:

request_id → developer_id, representative_id, acceptedStatus, date, job_description

Candidate Keys:

{ (request_id) }

Normal Form:

BCNF

Table Definition:

CREATE TABLE Request(

request_id INT PRIMARY KEY,

acceptedStatus ENUM (‘accepted’, declined, ‘waiting’) NOT NULL,

date DATE NOT NULL,

job_description VARCHAR (100) NOT NULL,

developer_id INT NOT NULL,

representative_id INT NOT NULL,

FOREIGN KEY (developer_id) REFERENCES Developer (developer_id),

FOREIGN KEY (representative_id) REFERENCES Representative (representative_id));

3. Functional Dependencies and Normalization of Tables

All of the relations in our design report are in BCNF (Boyce-Codd Normal Form). Since it is a strict form of normalization that removes the redundancies caused by functional dependencies, there is no need for additional normalization.

4. Functional Components

4.1. Use Cases / Scenarios

4.1.1. Developer

- Create Account: Developers can create an account if they do not have one already. They enter their email, name, phone and password to create it.
- Login to Account: Developers can login to their existing accounts. They need to login to change their profile and user information, take quizzes, view their quiz results and number of remaining quiz trials as well as responding to the representatives’ requests.
- Change Profile Information: Developers can change their profile information such as their GitHub link, Linkedin link, preferred working locations, preferred companies, role preferences, experienced years, company blacklist (which companies should not see their profile), status and the link to their CV.

- Change User Information: Developers can change their user information such as their name, phone and password. However, they cannot change their mail, they need to create a new account with the desired mail if that is the case.
- Take Quiz: Developers can take quizzes in different categories to show their knowledge in that category.
- View Quiz Result: Developers can view their quiz result and get feedback after completing each quiz.
- View Number of Remaining Quiz Trials: Developers have three trials for each quiz. They can view the remaining number of trials for each quiz.
- Respond Representative's Request: When the representatives want to communicate with a developer, they send a request. Developers will be able to accept or decline that request.

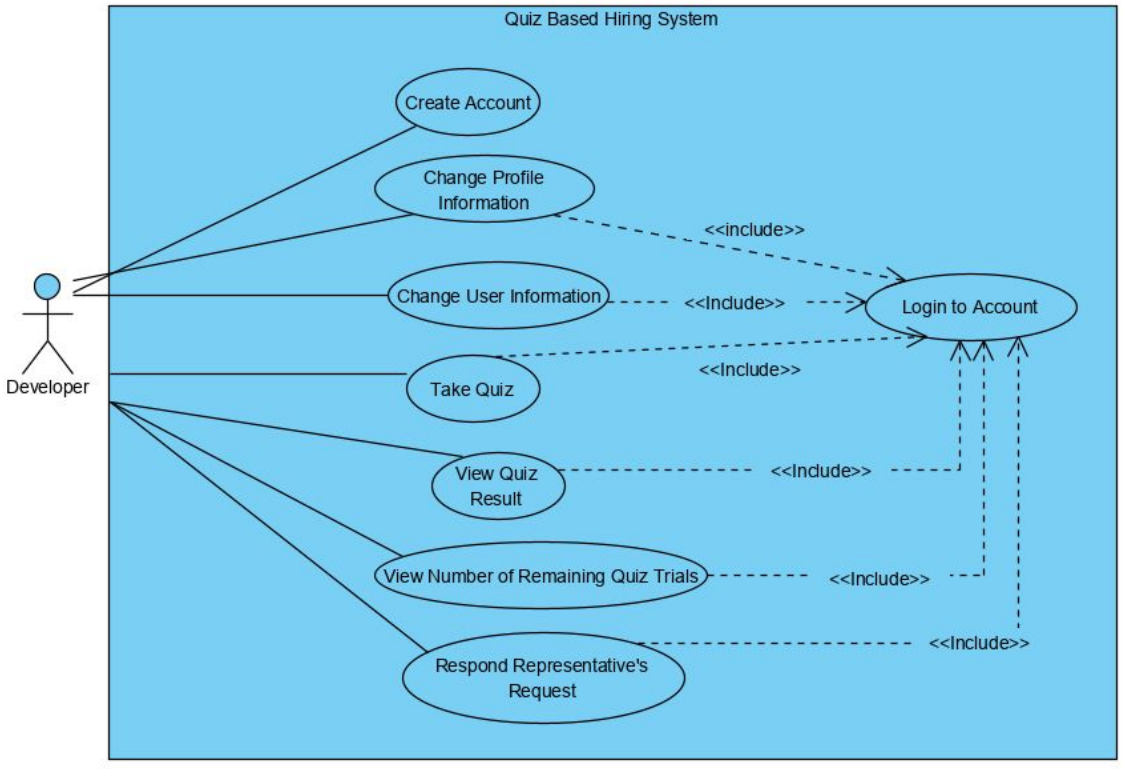


Figure 1. Use Case of Developer

4.1.2. Company Representative

- Sign up: If the company of representative is registered, the representative can sign up to the system by entering her/his information and selecting the company.
- Sign in: The representatives can sign in to the system by email and password if they already have an account. In order to perform the following functions, the representative should sign in to the system.
- View Quiz Results: The representatives can view all the quiz results in the descending score order.
- Select Category: The representatives can select a category or categories if the company needs a developer in a particular category. Then, they can show the results in that category.
- View Developer's Profile: The representative can review developers' profiles and see the other quiz results.
- Send Interview Request: The representatives can send an interview request to a developer. Then, the system sends the contact information with the representative unless there is any exception as the followings.
- Get Rejection Message: After sending the interview request, if the developer rejects this request, the developer is informed and gets a rejection message.
- Get Time Out Error: After sending the interview request, if one week passes and there is no response, the developer is informed and gets a time error message.

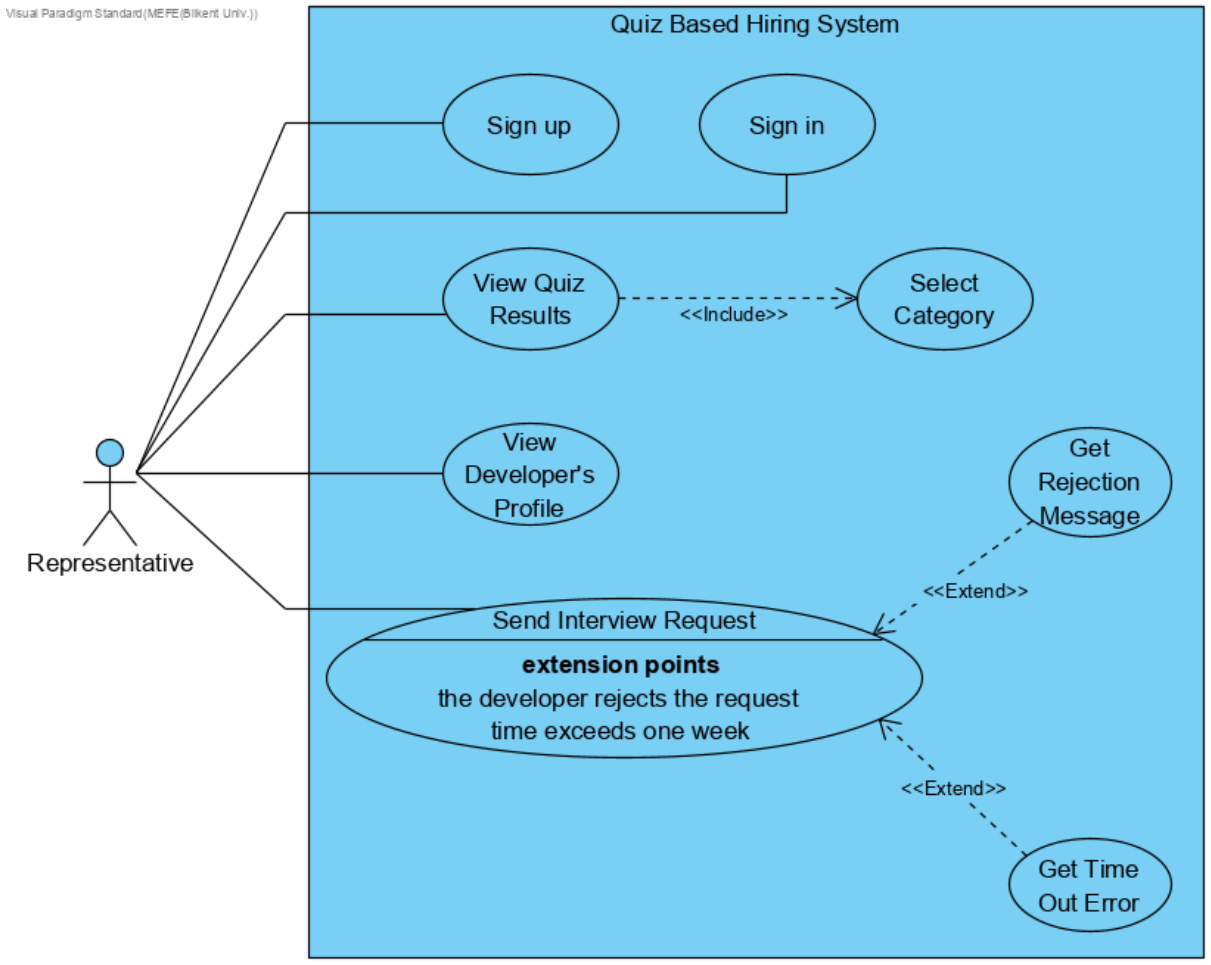


Figure 2. Use Case of Representative

4.1.3. Admin

- Sign Up: Admins must sign up when they first enter the system like each user and specify themselves as an admin.
- Sign In: Admins have to sign in to the system with their password and email like every user if they have an account. Sign in process is necessary to do the following actions.
- Add Personal Information: Admin can add their name and phone number into their user account.
- Change Personal Information: Admins are able to change their user information like phone, name and password; however, they are not allowed to change their email like other users.

- Create a Question: Questions are created by admins in this system. They can create questions if they provide necessary information about the question.
- Create a Quiz: Admins can create a quiz on this system. They benefit from existing questions when they prepare a quiz.
- Select a Category: Admins need to select a category when they prepare either a question or quiz because the system categorizes them by this information.

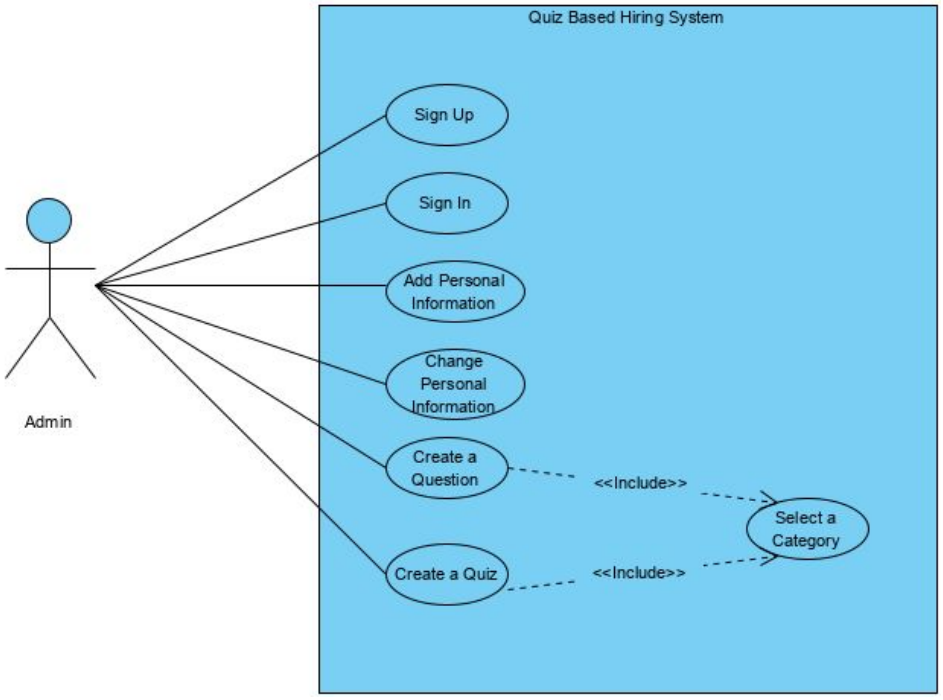


Figure 3. Use Case of Admin

4.2. Algorithms

4.2.1. Algorithm of Calculating the Quiz Results

Every developer can take any kind of quiz from this system to show his or her skills and knowledge. To display their capabilities, their results should be calculated at the end of a quiz. Each quiz has its own scoring distribution; that's why, every one of them needs to be calculated question by question. When a developer completes the quiz, the system compares the developer's choices with answers one by one. If the developer's answer is wrong, he or she cannot get any point from that question. If

the developer's answer is true, s/he gets points by question category and difficulty. Each difficulty level has a different score. Receiving points increase from easy to hard. Scoring is also different for each category. Each quiz can include questions from different categories and every one of them gives points by the category of quiz. The system checks the difficulty level of the question and the importance of the quiz category. According to its checking process, it calculates the final result of the developer.

4.2.2. Algorithm of Sorting and Listing the Developers

According to the total scores that the developers got from the quizzes in a specific category, they will be listed so that the developers with better scores will be listed at the top. Therefore, the company representatives can see the developers' names and their scores in a descending order. Representatives will be able to view the list in each category.

4.2.3. Algorithm of Company Representative Request

A company may need a developer who is good at a particular area. Therefore, the representative can select the desired category and see the developers' results in that category. If a developer doesn't want to reveal her/his results to this company, the representative cannot see them in the list. If the company likes a developer's result and wants to interview with the developer, the representative can send an interview request to the developer on behalf of the company. After the interview request, the developer can accept or reject it. If the request is accepted, the system shares the contact information of the developer with the representative. Otherwise, the system asks the developer whether s/he wants to block this company or not. The developer can block the company and never see this company's requests again or s/he rejects only this request.

5. User Interface Design and Corresponding SQL Statements

5.1. Homepage

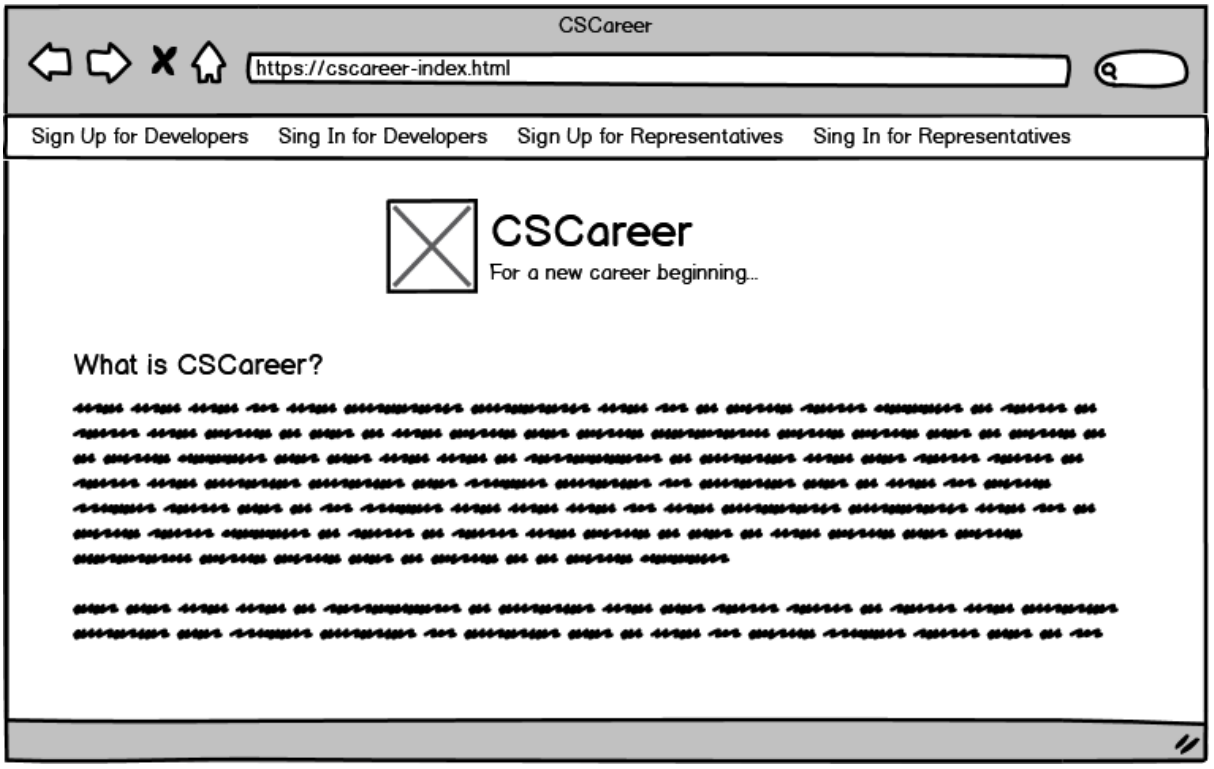


Figure 4. Homepage

In the homepage, there will be a brief description about what CSCareer is. Users will be able to get detailed information by clicking *How to Use* link which will be in the header. Also, in the header, there will be links to sign up and sign in for both developers and company representatives.

5.2. Sign up for Developer Page

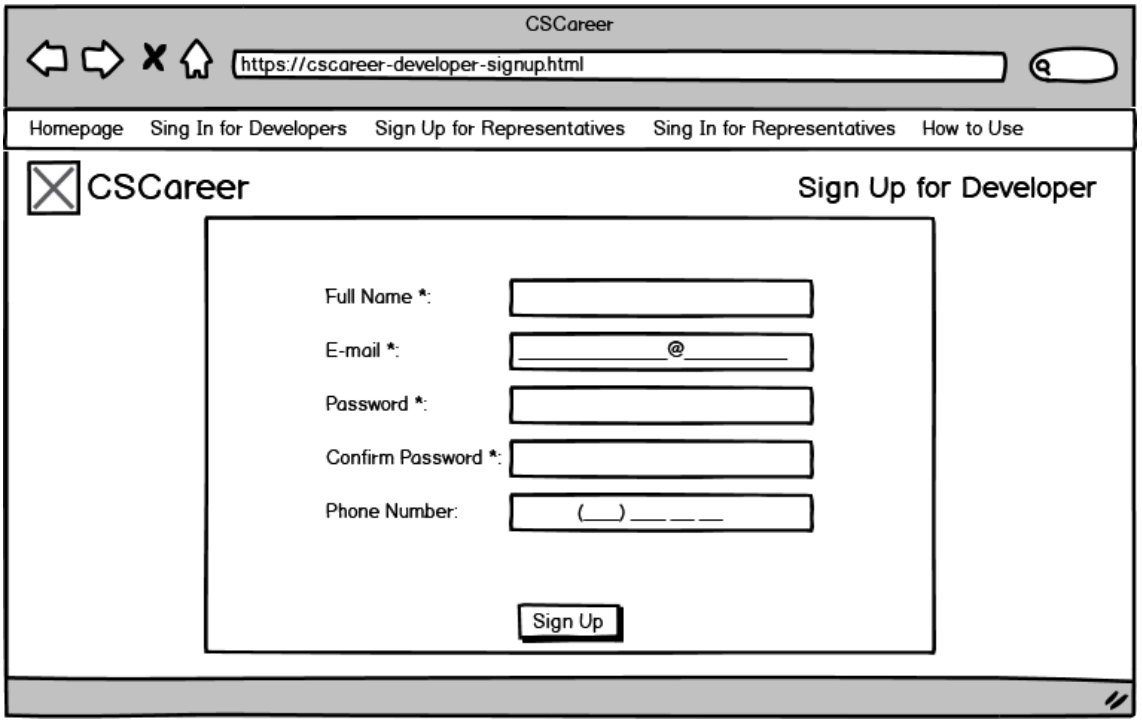


Figure 5. Sign Up for Developer

If the developer has no account, s/he can create an account by entering the necessary information in this page. In the form, it is mandatory to fill in the sections with an asterisk. Also, the ones written in the password and confirm password sections must be the same.

Inputs: @fullname, @email, @password, @confirmpassword, @phonenumber

SQL Statements:

- Firstly, values are inserted to *user* table:

```
INSERT INTO user (email, password, fullname, phone)
VALUES (@email, @password, @fullname, @phonenumber)
```

- Secondly, a new insertion is made in *developer* table since this user is a developer:

```
INSERT INTO developer
VALUES (SELECT user_id FROM user WHERE email = @email)
```

5.3. Sign in for Developer Page

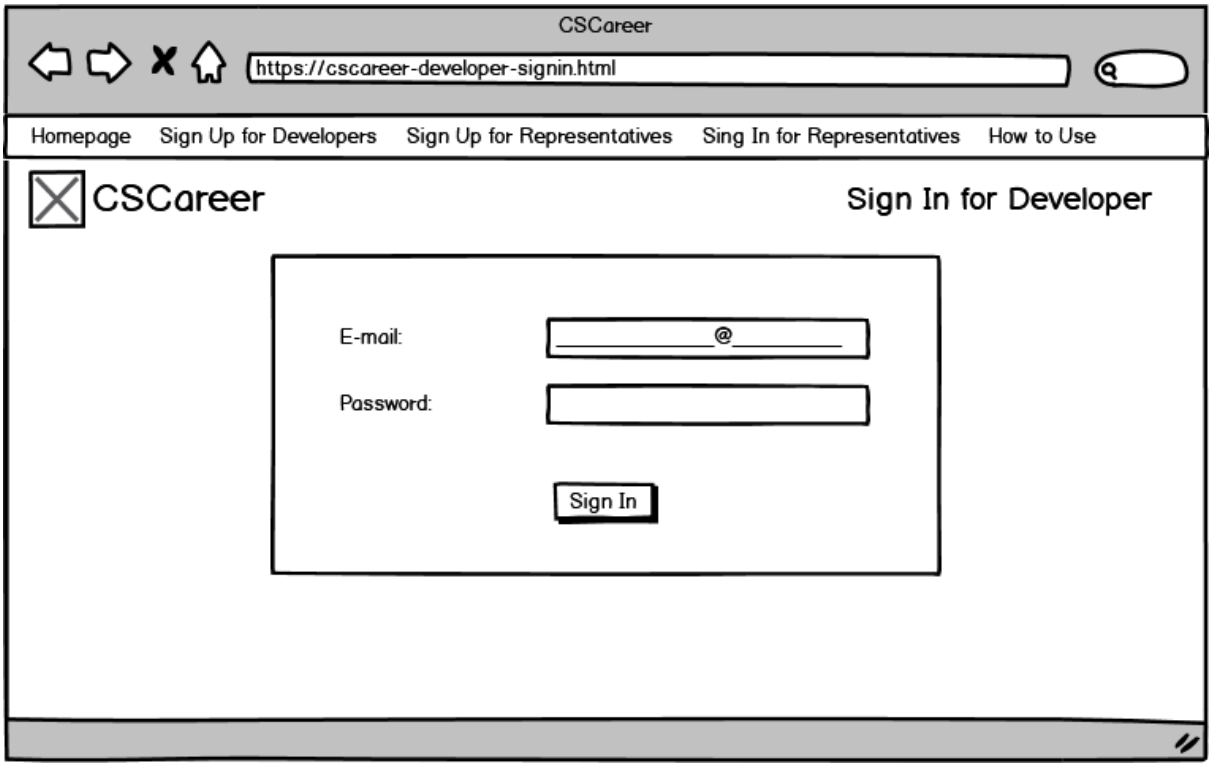


Figure 6. Sign in for Developer

In this page, developers will be able to sign in to the system by entering their email and password.

Inputs: @email, @password

SQL Statements:

```
SELECT *
FROM user
WHERE ( email = @email AND
        password = @password AND
        user_id IN (SELECT developer_id FROM developer) );
```

5.4. Developers' Edit Profile Page

In the Edit Profile page, there will be four options. Developers will be able to edit general information, edit / add education information, edit / add project information, and edit / add work information.

5.4.1. Edit General Information

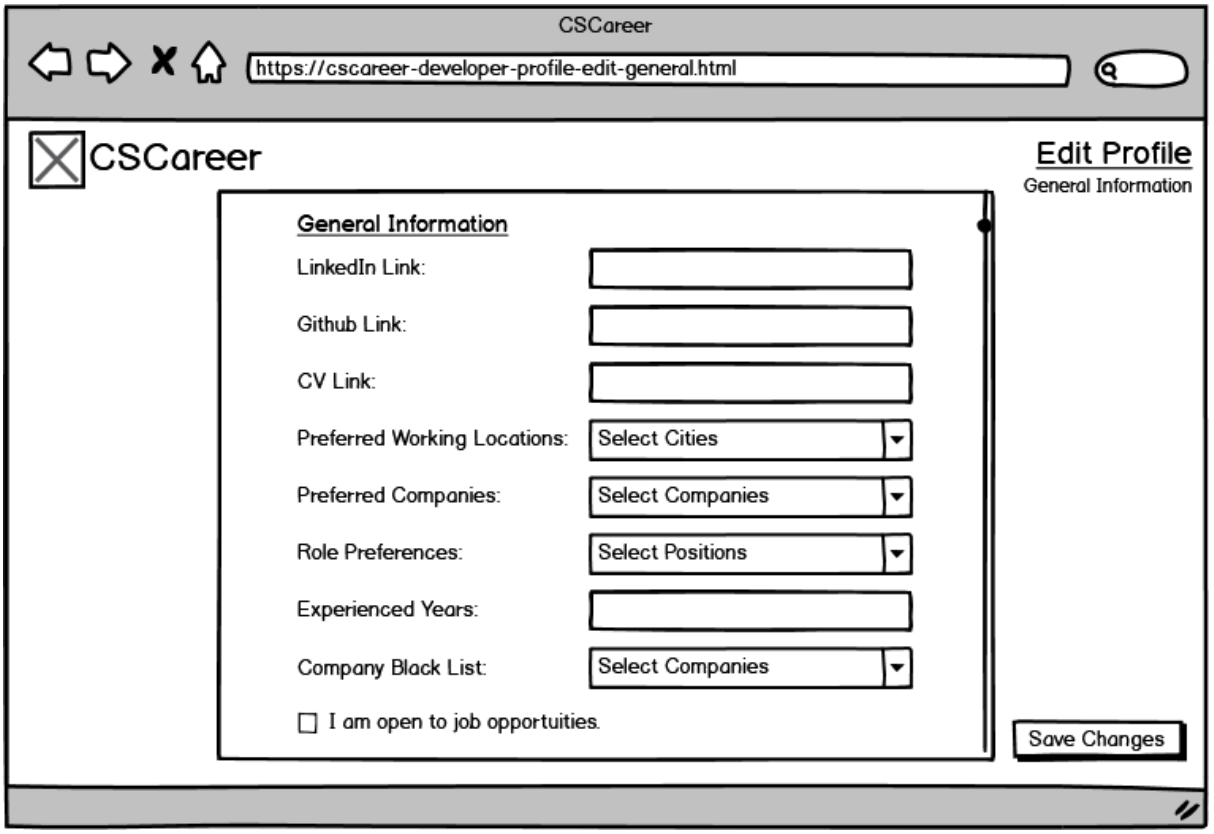


Figure 7. Edit Developer's General Information

Inputs: @profile_id, @developer_id, @linkedin_link, @github_link, @cv_link, @preferred_cities, @preferred_companies, @preferred_roles, @experienced_years, @blocked_companies, @status

SQL Statements:

In this part, there will be changes in several tables.

Update profile informations:

```
UPDATE profile
SET
    linkedin_link = @linkedin_link,
    github_link = @github_link,
    cv_link = @cv_link,
    experienced_years = @experienced_years,
    status = @status,
WHERE ( profile_id = @profile_id);
```

Add Preferred Working Location:

```
INSERT INTO preferredworkinglocations
VALUES (@profile_id, @preferred_cities)
```

Delete Preferred Working Location:

```
DELETE FROM preferredworkinglocations
WHERE profile_id = @profile_id AND preferred_cities = @preferred_cities
```

Add Preferred Company:

```
INSERT INTO preferredcompanies
VALUES (@profile_id, @preferred_companies)
```

Delete Preferred Company:

```
DELETE FROM preferredcompanies
WHERE       developer_id = @developer_id AND
            preferred_companies= @preferred_companies
```

Add Role Preferences:

```
INSERT INTO rolepreferences
VALUES (profile_id, preferred_roles)
```

Delete Role Preferences:

```
DELETE FROM rolepreferences
WHERE profile_id = @profile_id AND preferred_roles = @preferred_roles
```

Add Company to Black List:

```
INSERT INTO companyblacklist
VALUES (@profile_id, @bocked_companies)
```

Delete Company from Black List:

```
DELETE FROM companyblacklist
WHERE       developer_id = @developer_id AND
            bocked_companies = @bocked_companies
```

5.4.2. Edit Education Information

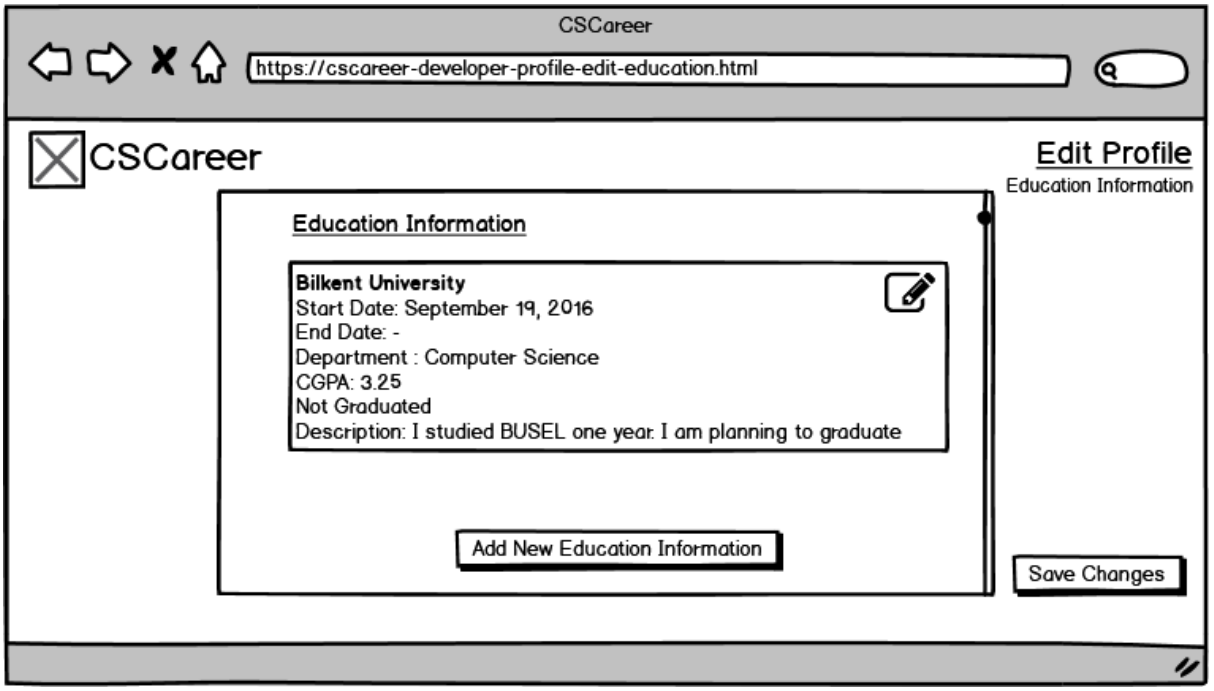


Figure 8. Edit Education Information

Inputs: @profile_id, @instution_name, @start_date, @end_date, @degree, @cgpa, @isGraduated, @description

SQL Statements:

Update education information:

```
UPDATE EducationInfo
SET    end_date = @end_date,
       degree = @degree,
       cgpa = @cgpa,
       isGraduated = @isGraduated,
       description = @description
WHERE profile_id = @profile_id AND instution_name = @instution_name
AND start_date = @start_date;
```

Add new education information:

INSERT INTO EducationInfo

VALUES (@profile_id, @instution_name, @start_date, @end_date, @degree, @cgpa, @isGraduated, @description)

5.4.3. Edit Work Information

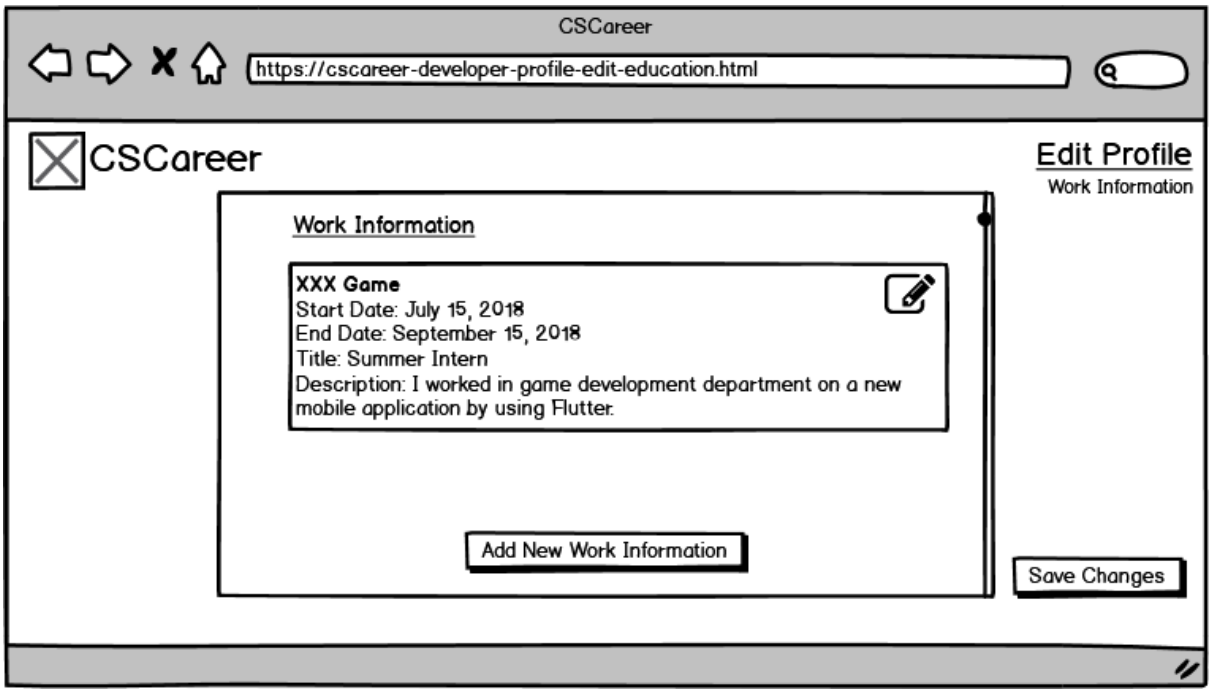


Figure 9. Edit Work Information

Inputs: @profile_id, @company_name, @start_date, @end_date, @title, @description

SQL Statements:

Update work information:

UPDATE WorkInfo

SET end_date = @end_date,

title = @title,

description = @description

WHERE profile_id = @profile_id AND company_name = @company_name AND start_date = @start_date;

Add new work information:

INSERT INTO WorkInfo

VALUES (@profile_id, @company_name, @start_date, @end_date, @title, @description)

5.4.4. Edit Project Information

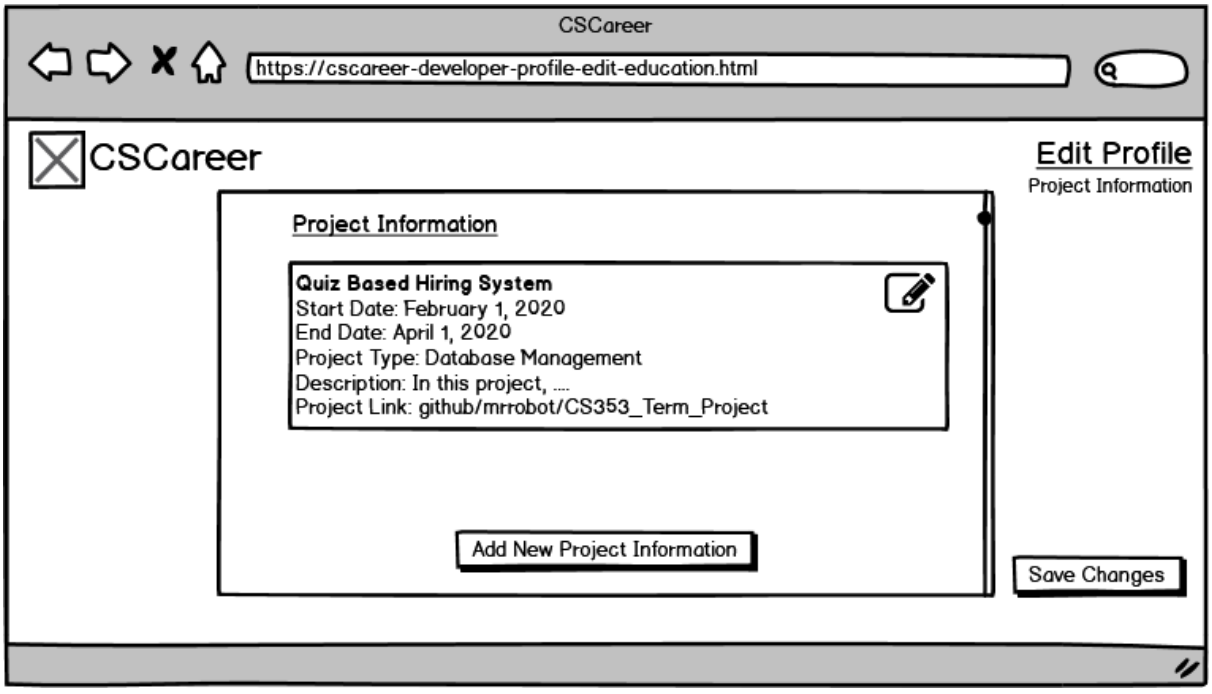


Figure 10. Edit Project Information

Inputs: @profile_id, @project_title, @start_date, @end_date, @project_type, @description, @project_link

SQL Statements:

Update project information:

UPDATE ProjectInfo

```
SET    end_date = @end_date,
       project_type = @project_type,
       description = @description,
       project_link = @project_link
WHERE profile_id = @profile_id AND project_title = @project_title AND
start_date = @start_date;
```

Add new project information:

```
INSERT INTO ProjectInfo
VALUES ( @profile_id, @project_title, @date, @project_type, @description,
@project_link)
```

5.5. Sign up for Representatives Page

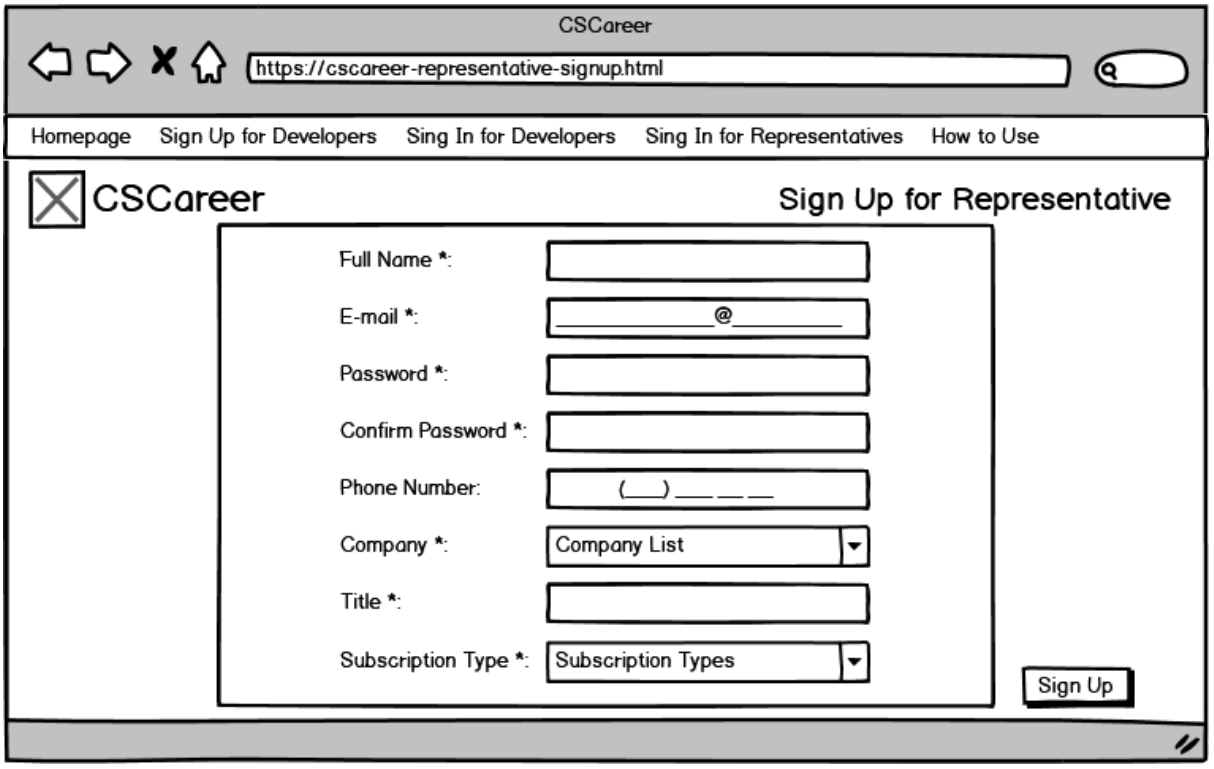


Figure 11. Sign up for Representatives

Representatives' sign up page will be similar to developers' sign up page. However, in this form, differently from the developer' sign up page, there will be company, title, and subscription type sections. Representatives should select her/his company from the list. Also, s/he should type her/his position in the company to the title section.

Inputs: @fullname, @email, @password, @confirmpassword, @phonenumber, @company_id, @title, @subscriptiontype

SQL Statements:

- Values are inserted into the user table.

```
INSERT INTO user (email, password, fullname, phone)
VALUES (@email, @password, @fullname, @phonenumber)
```

- Since this user is representative, the rest of the information is inserted to the representative table.

```
INSERT INTO representative
VALUES ( (SELECT user_id FROM user WHERE email = @email), @subscription_type, @title, @company_id )
```

5.6. Sign in for Representatives Page

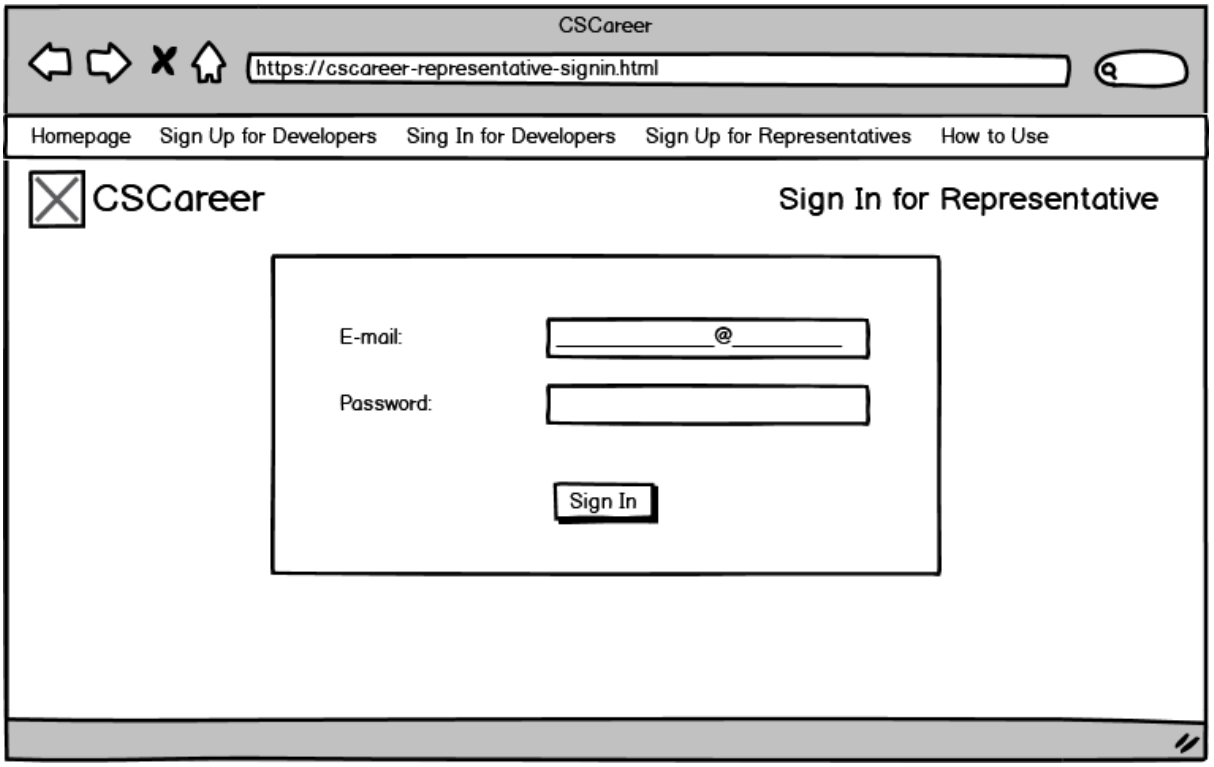


Figure 12. Sign in for Representatives

In this page, representatives will be able to sign in to the system by entering their email and password.

Inputs: @email, @password

SQL Statements:

```
SELECT *
FROM user
WHERE ( email = @username AND
        password = @password AND
        user_id IN (SELECT representative_id FROM representative) );
```

*** Since sign up/sign in functions of admin will be similar to developer and representative, the use cases related to admin is not added to this report.

5.7. Developer Takes a Quiz

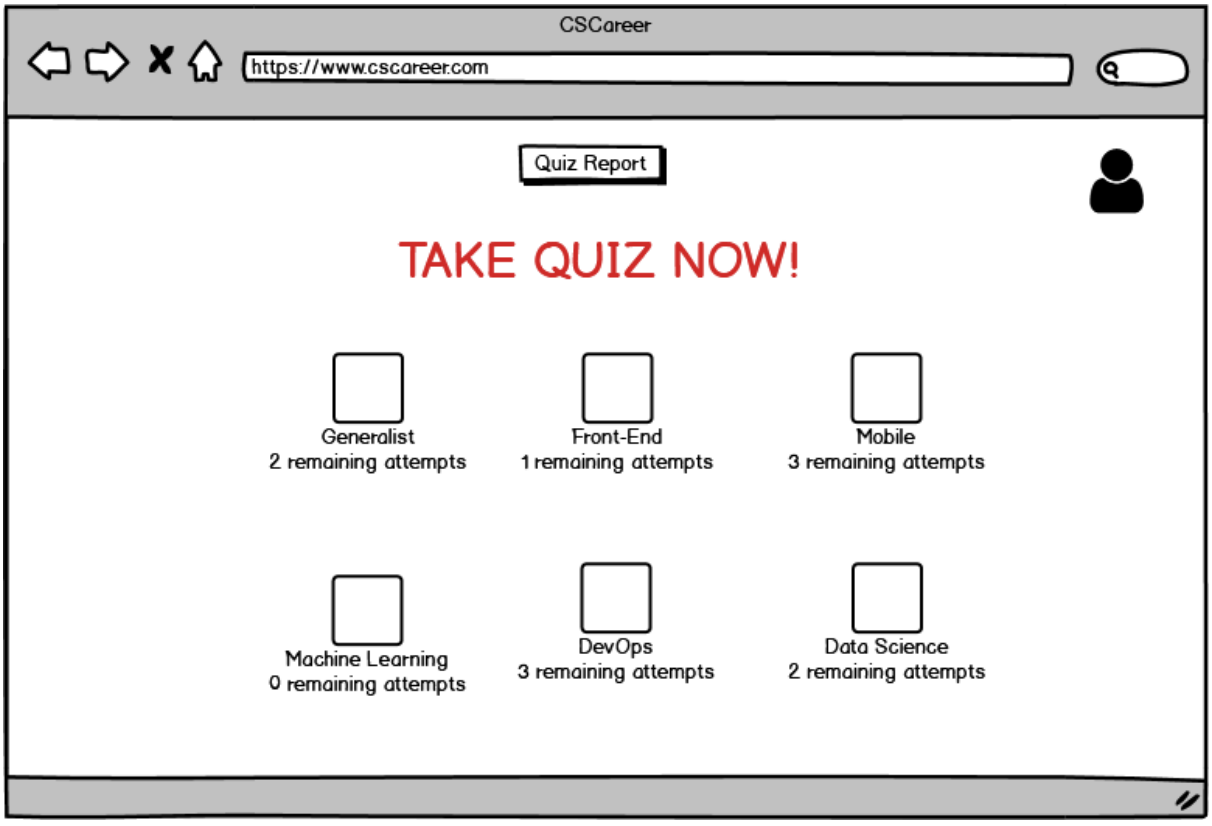


Figure 13. Developer Takes a Quiz

Inputs: @developer_id, @quiz_id, @attempt_id

The developer can see different quizzes and how many attempts they have for a quiz. This information will be displayed using view.

```
CREATE VIEW view_quizzes AS
SELECT quiz_id, remainingAttempts, category_name
FROM tries NATURAL JOIN quiz
WHERE developer_id = @developer_id
```

When the developer selects a quiz, a new quiz trial will be added.

```
INSERT INTO quiztrial (attempt_id, developer_id, quiz_id, date, isSuccessful,
total_score)

VALUES (@developer_id, @quiz_id, @date, null, null)
```

5.8. Developer Solves a Question

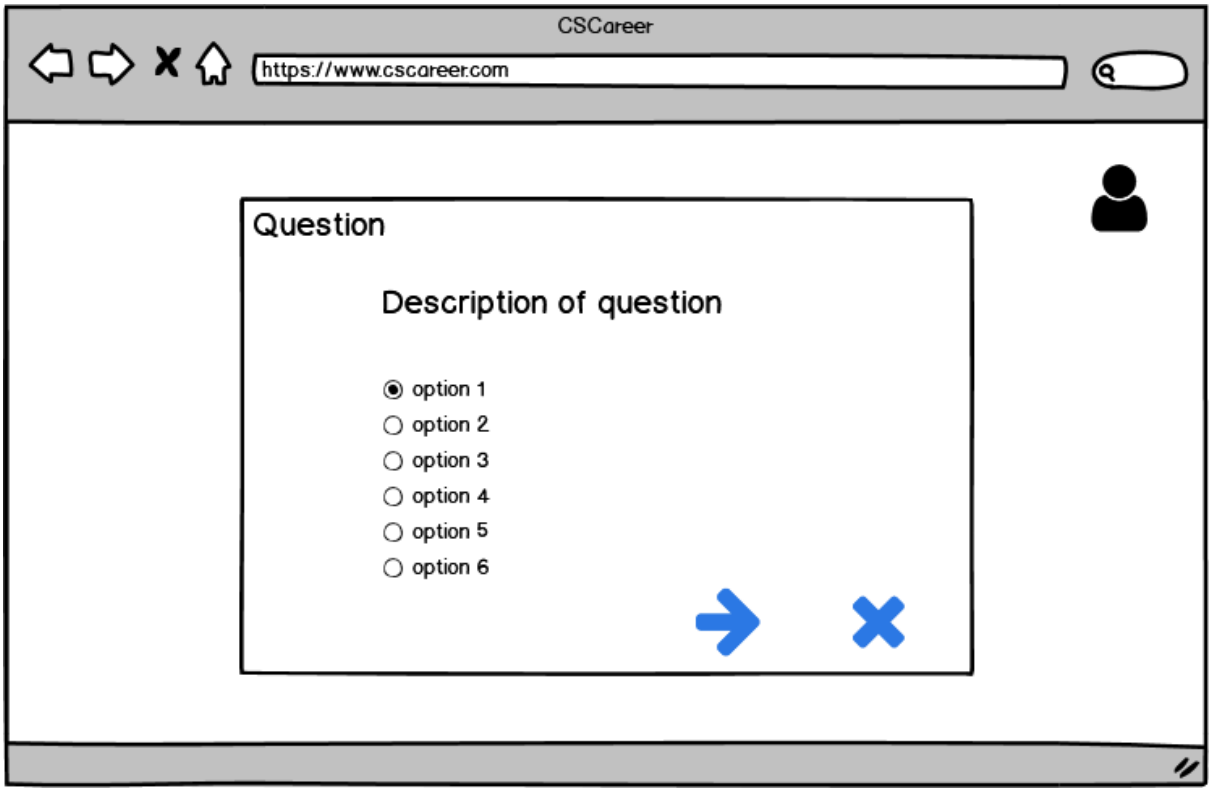


Figure 14. Developer Solves a Question

Inputs: @developer_id, @question_id, @choice_id

The questions should be found after selecting a quiz.

```
SELECT question_id, question_description

FROM quiz_question NATURAL JOIN question

WHERE quiz_id = @quiz_id

ORDER BY question_id ASC
```

For each question_id found by the above query, choices will be found.

```
SELECT choice_id, choice_text
FROM choice
WHERE question_id = @question_id
```

When the developer chooses an answer and presses the next button, the answer for the question will be recorded. The developer can also pass the question by not marking an option, then pressing the next button. Passing a question corresponds to the choice_id = 0.

First, it will be determined whether the choice is correct or not.

```
SELECT isCorrectAnswer
FROM choice
WHERE question_id = @question_id and choice_id = @choice_id
```

Then the answer will be recorded in the answer table together with isCorrectAnswer value.

```
INSERT INTO answer
VALUES ( @attempt_id, @question_id, @choice_id, isCorrectAnswer)
```

If the developer presses end the quiz button (denoted as X in above mock-up), isSuccessful and total_score values will be updated.

Firstly, total_score will be found

```
SELECT SUM (*) AS total_score
FROM answer
WHERE isCorrect = 1
```

Secondly, it will be determined whether the result is successful or not. If the total score > 15, the quiz is successful.

Finally, quiz trial will be updated with found values

```
UPDATE quiztrial
SET isSuccessful = @succesful, total_score = @total_score
WHERE attempt_id = @attempt_id
```

After finishing the quiz, remaining attempts for the quiz will be decreased by one.

```
UPDATE tries
SET remainingAttempts = remainingAttempts – 1
WHERE quiz_id = @quiz_id and developer_id = @developer_id
```

5.9. Developer Views Quiz Results

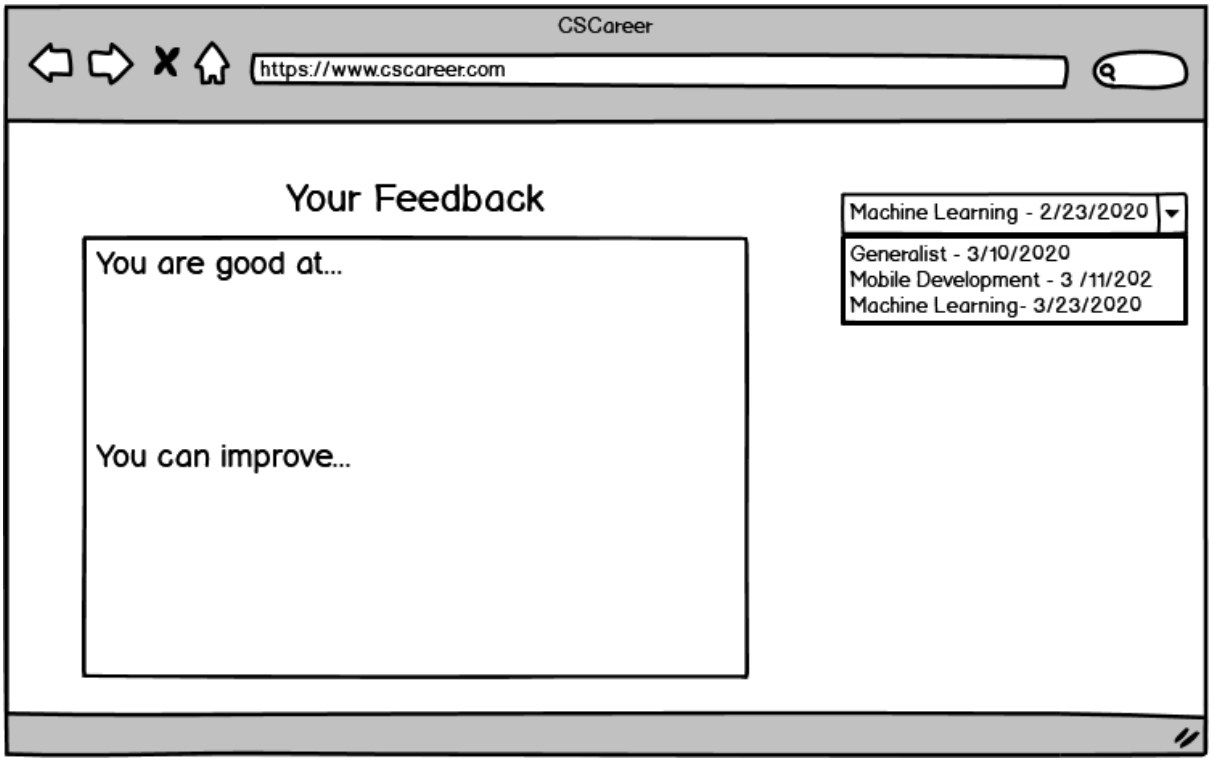


Figure 15. Developer Views Quiz Results

The drop-down menu will be displayed using the following command.

```
CREATE VIEW view_result AS
SELECT attempt_id, category_name, date
FROM quiztrial natural join quiz
WHERE developer_id = @developer_id
```

Feedback will be displayed according to the chosen attempt. If the score calculated by the below query is greater than 10, the category will be displayed in the “You are good at” part. Otherwise, it will be displayed in the “You can improve” part.

```
CREATE VIEW feedback AS
SELECT category_name, SUM (CASE WHEN isCorrect = 1 THEN 1 ELSE -1
END) AS score
FROM answer NATURAL JOIN categorized_as
WHERE attempt_id = @attempt_id
GROUP BY category_name
```

5.10. Admin Creates a New Quiz

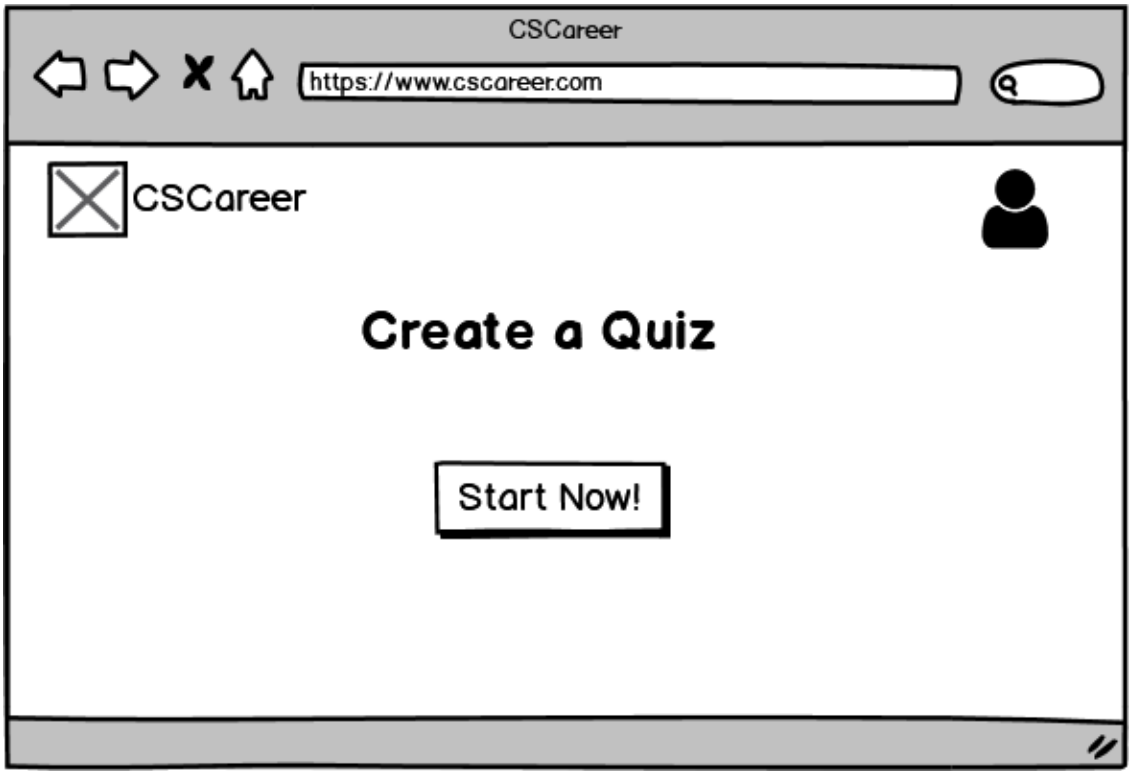


Figure 16. Admin Creates a New Quiz

Only admins can create a quiz. An admin can start creating by the “Start Now!” button.

5.11. Enter and Choose the Quiz's Properties

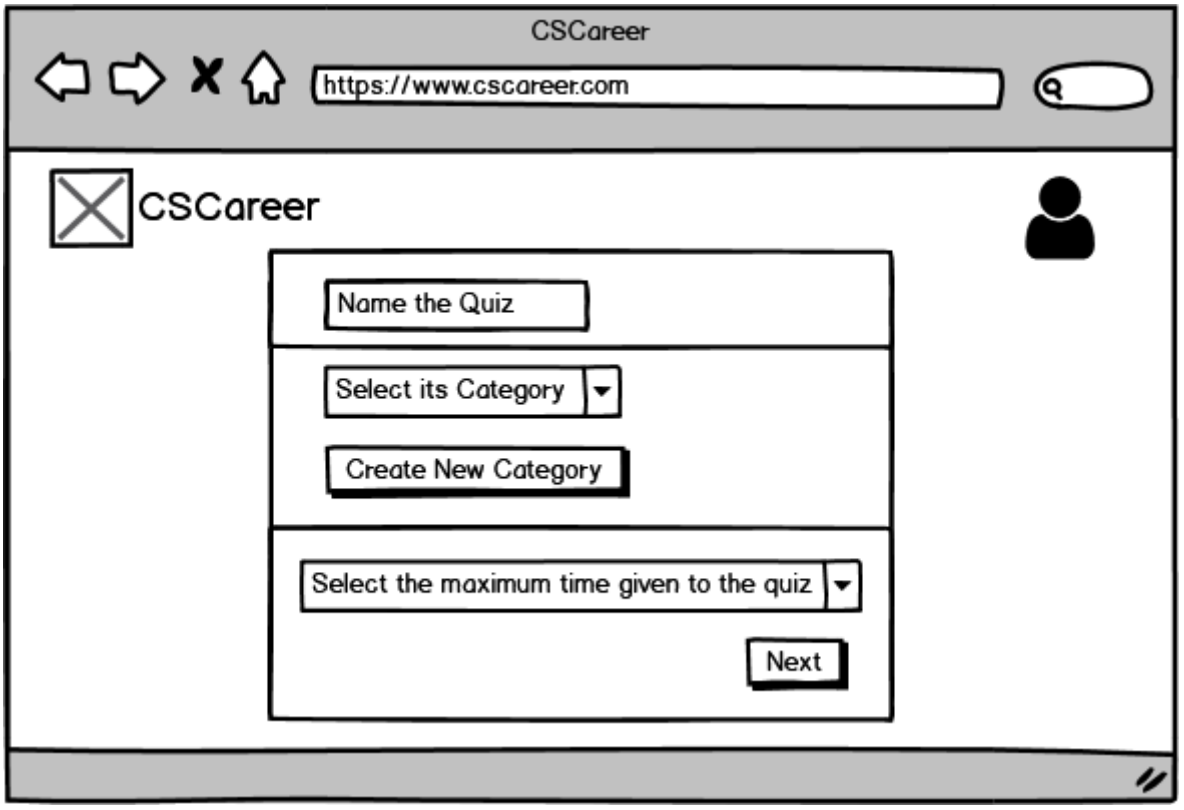


Figure 17. Enter and Choose the Quiz's Properties

Inputs: @admin_id, @quiz_title, @category_name, @time

SQL Statements:

Creating a quiz:

INSERT INTO quiz (quiz_id, admin_id, category_name, total_questions, quiz_title, time)

VALUES (@quiz_id, @admin_id, @category_name, 0, @quiz_title, @time)

Creating a new category:

INSERT INTO category

VALUES (@category_name)

5.12. Add Question and Complete Preparing the Quiz

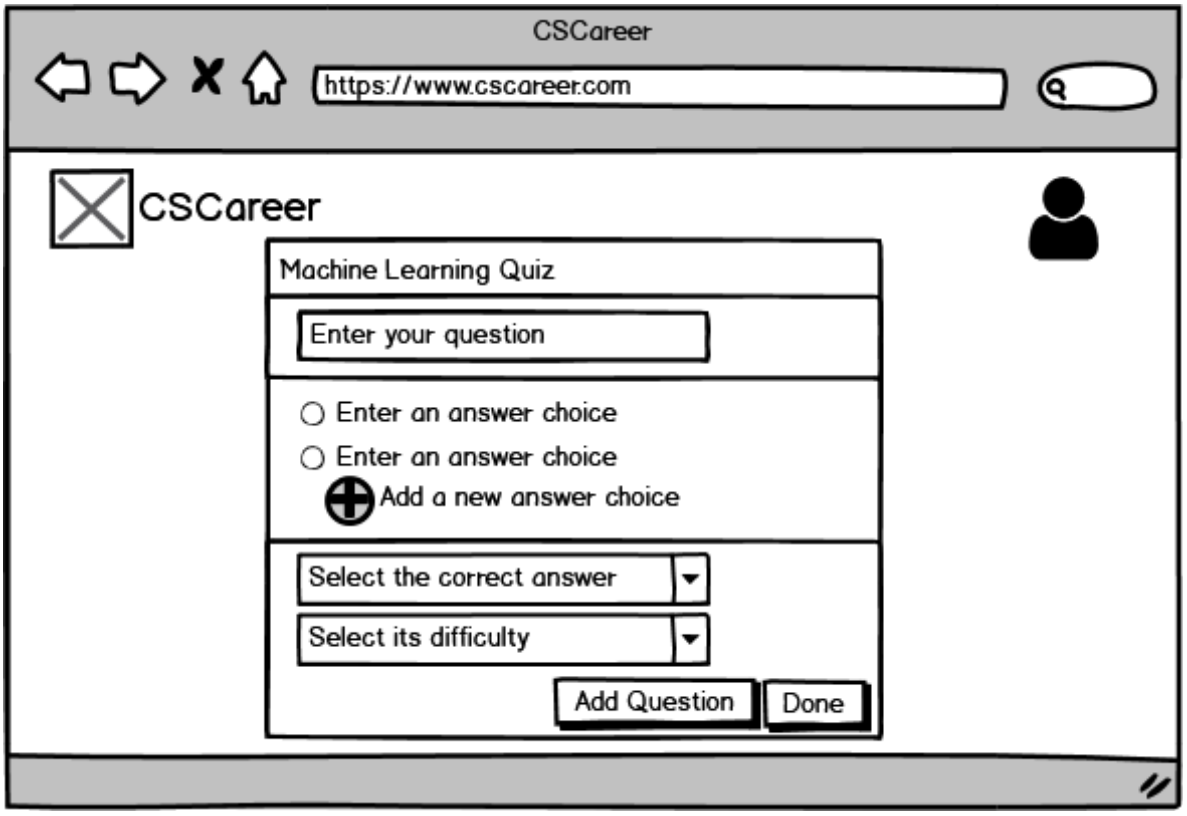


Figure 18. Add Question and Complete Preparing the Quiz

Inputs: @admin_id, @quiz_id, @description, @choice_text, @isCorrectAnswer, @difficulty

SQL Statements:

Creating a new choice:

```
INSERT INTO choice (question_id, choice_id, choice_text, isCorrectAnswer)
VALUES (@question_id, @choice_id, @choice_text, @isCorrectAnswer)
```

Creating a new question:

```
INSERT INTO question (question_id, admin_id, description, difficulty)
VALUES (@question_id, @admin_id, @description, @difficulty)
```

INSERT INTO quiz_questions (quiz_id, question_id)

VALUES (@quiz_id, @question_id)

Incrementing the number of questions in the quiz by 1 after each question is added:

UPDATE quiz (total_questions)

SET total_questions = total_questions + 1

WHERE quiz_id = @quiz_id AND admin_id = @admin_id

5.13. Representative Send an Interview Request

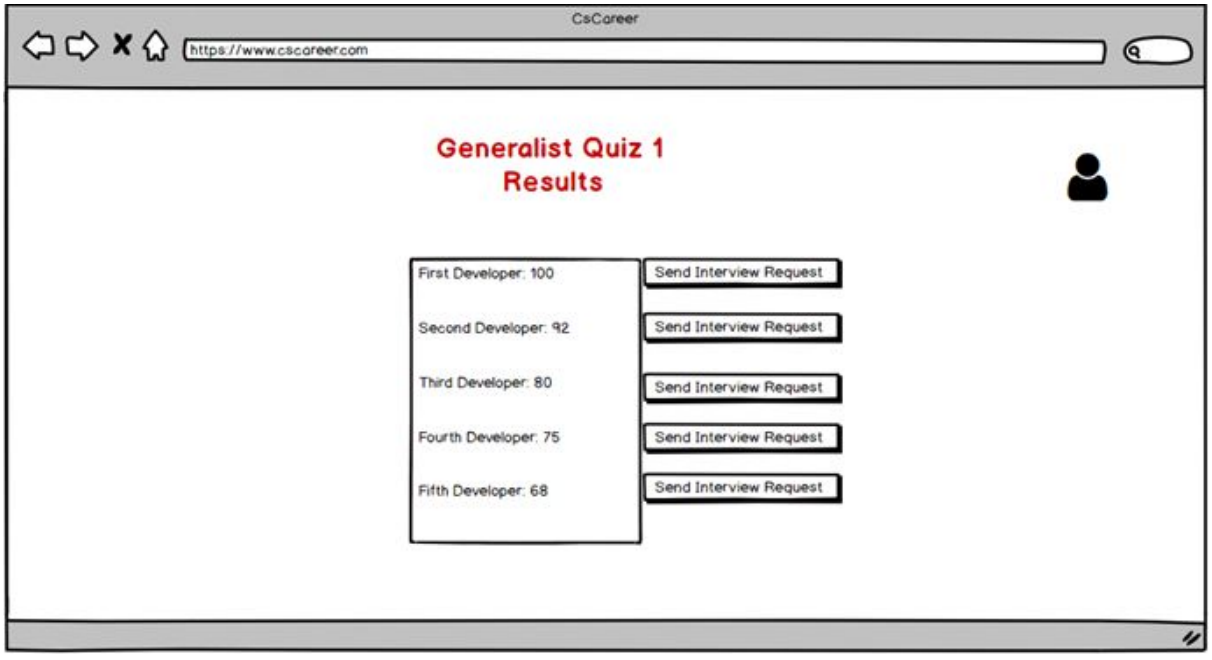


Figure 19. Representative Send an Interview Request

Inputs: @request_id, @date, @job_description

It is the result page example for each quiz. Company representatives can see the score of developers in descending order. According to this table, representatives can send an interview request to any developer.

INSERT INTO sendRequest (request_id, acceptedStatus, date, job_description)

VALUES (@request_id, @acceptedStatus, @date, @job_description)

5.13.1. Developer Responds to Interview Request

Figure 20. Developer Responds to Interview Request

Inputs: @request_id, @accepted_status

When a developer clicks on any interview request, he or she will access this page. They can see the interview request with its description. They can accept or decline the request.

If developer accepts the request:

UPDATE status

SET acceptedStatus = accepted

WHERE request_id = @request_id and developer_id = @developer_id

If developer declines the request:

UPDATE status

SET acceptedStatus = declined

WHERE request_id = @request_id and developer_id = @developer_id

6. Implementation Plan

In this project, MySQL is going to be used for database while PHP is going to be used for server side. To design user interface, HTML, CSS, and JavaScript are going to be used.

We shared the work as offered in the Project Functionality Document. According to the plan, each team member is responsible from one part as follows:

Meryem Efe → Common functionalities & Additional requirements

Hamza Pehlivan → Developer takes quiz.

Selen Uysal → Admin prepares questions.

Fırat Yönak → Representative sends interview request.

7. Website

Our website for further project information:

https://hamzapehlivan.github.io/Database-Project/