Bilkent University

Department of Computer Engineering

CS 353

Database Systems Project

Online Technical Interview Preparation and Coding Platform

Design Report

Project Group No: 10

Our Web Page's Address: https://aatahanm.github.io/

Group Members: Doğa Acar - 21502842 - Section 2

Mustafa Oğuz Güngör - 21501182 - Section 2

Ahmet Atahan Mutlu - 21604085 - Section 3

Alptekin Ay - 21601849 - Section 3

Course TA: Mustafa Can Çavdar

Table of Contents

Design Report

Online Technical Interview Preparation and Coding Platform

1. Revised E/R Model

What is changed:

We changed our model according our TA’s feedback in the following ways:

- A submission entity has been added which keeps track of each submissions towards a coding challenge. It holds information about the answer and the score it received.
- A question entity has been added which is in a weak relationship with coding challenges.
- Interview entity has been added that shows information such as, challenges, duration position of the interview and the company created it.
- A survival mode entity has been added as an extra feature. In the survival practise mode, users can challenge themselves against time. They will start the survival challenge with an initial time and every challenge they solve correctly, they will earn extra time.
- Various attributes and relations have also been added/changed according to the received feedback.

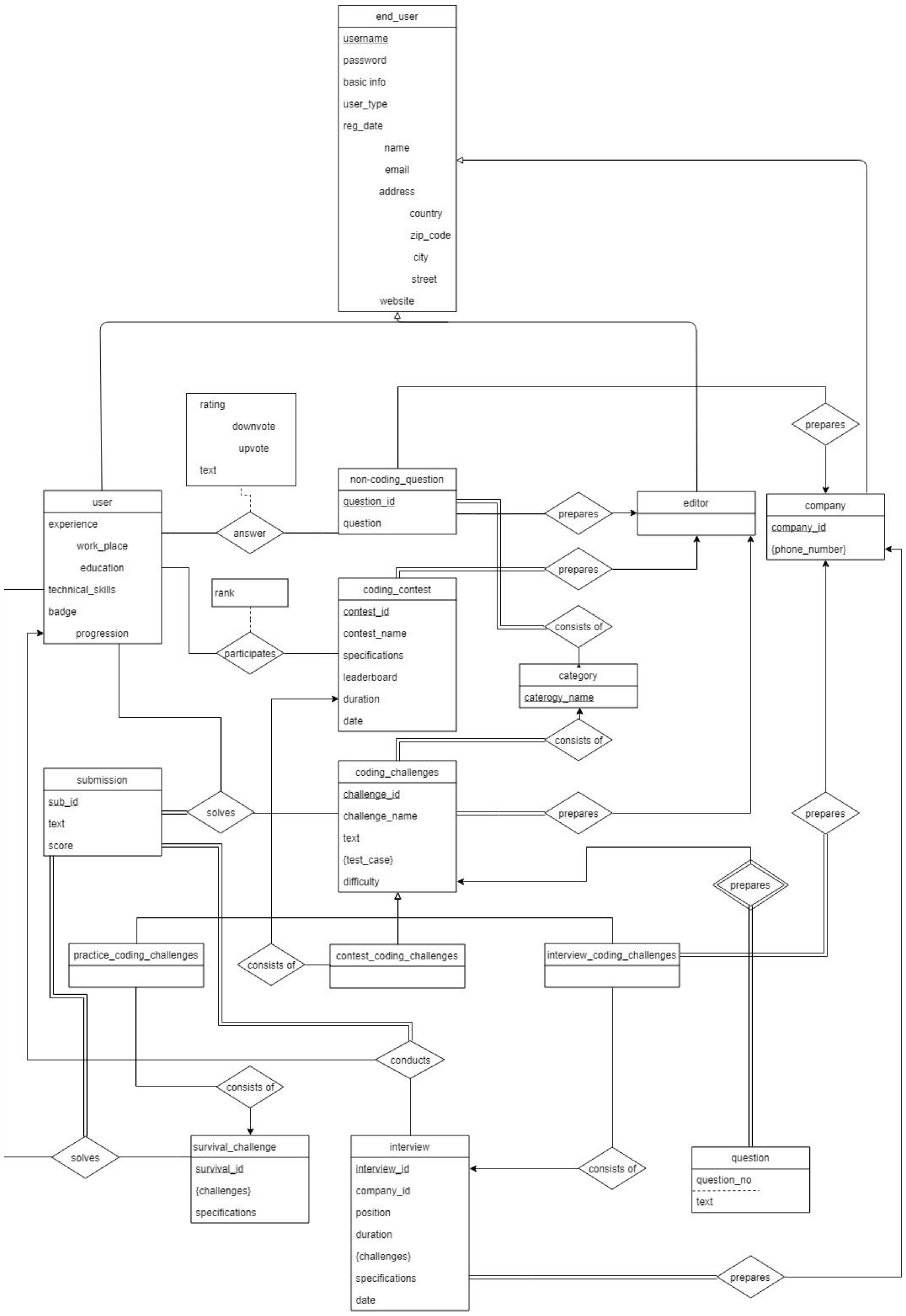


Fig. 1: Revised E/R model

2. Relation Schemas

This section includes the relation schemas that our system will have.

2.1 End User

Relational Model:

end_user(<u>username</u> , password , name, email, country, zip_code, city, street, website)

Functional Dependencies:

username → password, name, email, country, zip_code, city, street, website

Candidate Keys:

{(username)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE member(
        username varchar(20) PRIMARY KEY,
        password varchar(40) NOT NULL,
        name varchar(20),
        user_type varchar(7),
        email varchar(20),
        country varchar(20),
        zip_code varchar(5),
        city varchar(20),
        street varchar(20),
        website varchar(20),
        FOREIGN KEY (challenge_id) references coding_challenges,
        check(user_type in(“user”,”editor”,”company”))
);
```

2.2 User

Relational Model:

user(username , technical_skills)

Functional Dependencies:

NONE

Candidate Keys:

{ (username)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user(
        Username varchar(20) PRIMARY KEY,
        work_place varchar(400),
        education varchar(400),
        technical_skills varchar(400),
        FOREIGN KEY (username) references member);
```

2.3 Editor

Relational Model:

editor(<u>username</u>)

Functional Dependencies:

NONE

Candidate Keys:

{ (username)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor(
        username varchar(20) PRIMARY KEY,
        FOREIGN KEY (username) references member);
```

2.4 Company

Relational Model:

company(<u>username</u>, <u>company_id</u>, phone_number)

Functional Dependencies:

NONE

Candidate Keys:

{(username), (company_id), (username, company_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE company(
        company_id int PRIMARY KEY,
        username varchar(20),
        phone_number char(10) NOT NULL,
        FOREIGN KEY(username) references member);
```

2.5 Non-coding Question

Relational Model:

non-coding_question(<u>question_id</u>, question)

Functional Dependencies:

NONE

question_id → question

Candidate Keys:

{(<u>question_id</u>)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE non-coding_question(
        question_id int PRIMARY KEY,
        question varchar(200) NOT NULL);
```

2.6 Coding Contest

Relational Model:

coding_contest(<u>contest_id</u>, leaderboard, time)

Functional Dependencies:

Candidate Keys:

{(contest_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE coding_contest(
        contest_id int PRIMARY KEY,
        leaderboard varchar(200),
        time NOT NULL);
```

2.7 Coding Challenges

Relational Model:

coding_challenges(challenge-id, challenge-name, text, test_case, difficulty)

Functional Dependencies:

NONE

Candidate Keys:

{(challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE coding_challenges(
        challenge-id int PRIMARY KEY,
        challenge-name varchar(60) NOT NULL,
        text varchar(1000),
        test_case char(40),
        difficulty varchar (10));
```

2.8 Practice Coding Challenges

Relational Model:

practice_coding_challenges(challenge_id)

Functional Dependencies:

NONE

Candidate Keys:

{(challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE practice_coding_challenges(
        challenge_id int PRIMARY KEY,
        FOREIGN KEY (challenge_id) references coding_challenges);
```

2.9 Contest Coding Challenges

Relational Model:

contest_coding_challenges(challenge_id)

Functional Dependencies:

NONE

Candidate Keys:

{(challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE contest_coding_challenges(
        challenge_id int PRIMARY KEY,
        FOREIGN KEY (challenge_id) references coding_challenges);
```

2.10 Interview Coding Challenges

Relational Model:

interview_coding_challenges(<u>challenge_id)</u>

Functional Dependencies:

NONE

Candidate Keys:

{(challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE interview_coding_challenges(
        challenge_id int PRIMARY KEY,
        FOREIGN KEY (challenge_id) references coding_challenges
);
```

2.11 Category

Relational Model:

category(category_name)

Functional Dependencies:

none

Candidate Keys:

{(category_name)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE category(
        category_name varchar(20) PRIMARY KEY
);
```

2.12 Solves_Coding_Challenge

Relational Model:

solves_coding_challenge(<u>challenge_id</u>, <u>username</u>, <u>sub_id</u>)

Functional Dependencies:

none

Candidate Keys:

{(challenge_id, username, sub_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE solves(
        challenge_id int,
        username varchar(20),
        sub_id int,
        PRIMARY KEY (challenge_id, username, sub_id),
        FOREIGN KEY (challenge_id) references coding_challenges,
        FOREIGN KEY (username) references user(username)
        FOREIGN KEY (sub_id) references submission(sub-id)
);
```

2.13 Submission

Relation Model:

submission(<u>sub-id</u>, text, score)

Functional Dependencies:

none

Candidate Keys:

{(sub-id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE submission(
        sub-id int PRIMARY KEY,
        text varchar(1000),
        score float NOT NULL
);
```

2.14 Survival Challenge

Relation Model:

survival_coding_challenge(survival-id,challenges,specifications)

Functional Dependencies:

none

Candidate Keys:

{(survival-id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE survival_coding_challenge(
        survival-id int PRIMARY KEY,
        challenges challenge,
        specifications int NOT NULL
);
```

2.15 coding_challenge_has

Relation Model:

coding_challenge_has(challenge_id, question_no)

Functional Dependencies:

none

Candidate Keys:

{(challenge_id, question_no)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE coding_challenge_has(
        challenge_id int,
        question_no int,
        PRIMARY KEY (challenge_id, question_no),
        FOREIGN KEY challenge_id references challenges,
        FOREIGN KEY question_no references question
);
```

2.16 interview

Relation Model:

interview(interview_id,company_id,position,duration,specifications,date,)

Functional Dependencies:

none

Candidate Keys:

{(interview_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE interview(
        interview_id int,
        company_id int,
        position varchar(30),
        duration date,
        specifications varchar(400),
        date date,
        PRIMARY KEY (interview_id)
);
```

2.17 question

Relation Model:

question(<u>challenge_id</u>, <u>question_no,text</u>)

Functional Dependencies:

none

Candidate Keys

{(challenge_id, question_no)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE question(
        challenge_id int PRIMARY KEY,
        question_no int NOT NULL,
        text varchar (800) NOT NULL,
        PRIMARY KEY (challenge_id,question_no),
        FOREIGN KEY (challenge_id) references coding_challenges
);
```

2.18 user_non_coding_question

Relational Model:

user_non_coding_question(username, question_id, rate, text)

Functional Dependencies:

NONE

Candidate Keys:

{username, question_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user_non_coding_question(
        username varchar(20),
        question_id int NOT NULL,
        text varchar (800) NOT NULL,
        rate int NOT NULL,
        PRIMARY KEY (username, question_id),
        FOREIGN KEY (username) references user,
        FOREIGN KEY (question_id) references non_coding_question
);
```

2.19 editor_non_coding_question

Relational Model:

editor_non_coding_question(username, question_id)

Functional Dependencies:

NONE

Candidate Keys:

{username, question_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor_non_coding_question(
        username varchar(20),
        question_id int NOT NULL,
        PRIMARY KEY (username, question_id),
        FOREIGN KEY (username) references editor,
        FOREIGN KEY (question_id) references non_coding_question
);
```

2.20 company_non_coding_question

Relational Model:

company_non_coding_question(username, company_id, question_id)

Functional Dependencies:

NONE

Candidate Keys:

{username, company_id, question_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE company_non_coding_question(
        username varchar(20),
        company_id int NOT NULL,
        question_id int NOT NULL,
        PRIMARY KEY (username, company_id, question_id),
        FOREIGN KEY (username) references company,
        FOREIGN KEY (company_id) references company,
        FOREIGN KEY (question_id) references non_coding_question
);
```

2.21 category_non_coding_question

Relational Model:

category_non_coding_question(category_name, question_id)

Functional Dependencies:

NONE

Candidate Keys:

{category_name, question_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor_non_coding_question(
        category_name varchar(20) NOT NULL,
        question_id int NOT NULL,
        PRIMARY KEY (category_name, question_id),
        FOREIGN KEY (category_name) references category,
        FOREIGN KEY (question_id) references non_coding_question
);
```

2.22 user_coding_contest

Relational Model:

user_coding_contest(<u>username</u>, <u>contest_id</u>)

Functional Dependencies:

NONE

Candidate Keys:

{username, contest_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user_coding_contest(
        username varchar(20),
        contest_id int NOT NULL,
        PRIMARY KEY (username, contest_id),
        FOREIGN KEY (username) references user,
        FOREIGN KEY (contest_id ) references coding_contest
);
```

2.23 editor_coding_contest

Relational Model:

editor_coding_contest(<u>username</u>, <u>contest_id</u>)

Functional Dependencies:

NONE

Candidate Keys:

{username, contest_id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor_coding_contest(
        username varchar(20),
        contest_id int NOT NULL,
        PRIMARY KEY (username, contest_id),
        FOREIGN KEY (username) references editor,
        FOREIGN KEY (contest_id ) references coding_contest
);
```

2.24 category_coding_challenges

Relational Model:

category_coding_challenges(<u>challenge_id,category_name)</u>

Functional Dependencies:

NONE

Candidate Keys:

{(challenge_id),(category_name,challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE category_coding_challenges(
        challenge_id int,
        category_name varchar(20),
        PRIMARY KEY (challenge_id),
        FOREIGN KEY (category_name) references category(category_name),
        FOREIGN KEY (challenge_id) references coding_challenge(challenge_id)
);
```

2.25 editor_coding_challenges

Relational Model:

editor_coding_challenges(<u>challenge_id,username)</u>

Functional Dependencies:

NONE

Candidate Keys:

{(username,challenge_id),(challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE editor_coding_challenges(
        interview_id int,
        challenge_id int,
        PRIMARY KEY (challenge_id),
        FOREIGN KEY (username) references editor(username),
        FOREIGN KEY (challenge_id) references coding_challenge(challenge_id)
);
```

2.26 company_interview_coding_challenges

Relational Model:

company_interview_coding_challenges(interview_id,challenge_id)

Functional Dependencies:

NONE

Candidate Keys:

{(interview_id,challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE interview_interview_coding_challenges(
        interview_id int,
        challenge_id int,
        PRIMARY KEY (contest,challenge_id),
        FOREIGN KEY (interview_id) references interview(interview_id),
        FOREIGN KEY (challenge_id) references interview_coding_challenge(challenge_id)
);
```

2.27 interview_interview_coding_challenges

Relational Model:interview_interview_coding_challenges(interview_id,challenge_id)

Functional Dependencies:

NONE

Candidate Keys:

{(interview_id,challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE interview_interview_coding_challenges(
        interview_id int,
        challenge_id int,
        PRIMARY KEY (contest,challenge_id),
        FOREIGN KEY (interview_id) references interview(interview_id),
        FOREIGN KEY (challenge_id) references interview_coding_challenge(challenge_id)
);
```

2.28 coding_contest_contest_coding_challenges

Relational Model:coding_contest_contest_coding_challenges(contest_id,challenge_id)

Functional Dependencies:

NONE

Candidate Keys:

{(contest,challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE coding_contest_contest_coding_challenges(
        contest_id int,
        challenge_id int,
        PRIMARY KEY (contest,challenge_id),
        FOREIGN KEY (contest_id) references coding_contest(contest_id),
        FOREIGN KEY (challenge_id) references contest_coding_challenge(challenge_id)
);
```

2.29 user_submission_survival_challenge

Relational Model: user_submission_survival_challenge(sub_id,survival_id,username)

Functional Dependencies:

NONE

Candidate Keys:

{(sub_id),(survival_id,sub_id),(username,sub_id)(username,sub_id,survival_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user_submission_survival_challenge(
        survival_id int,
        sub_id int,
        username varchar(20),
        PRIMARY KEY (sub_id),
        FOREIGN KEY (survival_id) references survival_challenge(survival_id),
        FOREIGN KEY (username) references user(username)
        FOREIGN KEY (sub_id) references submission(usb_id)
);
```

2.30 practice_coding_challenge_survival_challenge

Relational Model:practice_coding_challenge_survival_challenge(survival_id,challenge_id)

Functional Dependencies:

NONE

Candidate Keys:

{(survival_id,challenge_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE practice_coding_challenge_survival_challenge(
        survival_id int,
        challenge_id int,
        PRIMARY KEY (survival_id,challenge_id),
        FOREIGN KEY (survival_id) references survival_challenge(survival_id),
        FOREIGN KEY (challenge_id) references practice_coding_challenge(challenge_id)
);
```

2.31 conducts

Relation Model:

conducts(<u>interview_id,sub_id ,username</u>)

Functional Dependencies:

none

Candidate Keys:

{(interview_id, sub_id, username), (interview_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE interviewer_interviewee(
        interview_id int,
        sub_id int,
        username varchar(20),
        PRIMARY KEY (interview_id),
        FOREIGN KEY (interview_id) references interview(interview_id),
        FOREIGN KEY (sub_id) references submission(sub_id)
        FOREIGN KEY (username) references user(username)
);
```

3. Functional Dependencies and Normalization of Tables

Every functional dependency and every normal form are given in the relation schemas which is Section 2 of this Project Design Report. Every relation is checked in our design if the relation is in Boyce-Codd Normal Form. Since the left side of the functional dependencies in our schemas are foreign keys, they are in BCNF form.

4. Functional Components

This section includes the use cases of the system and possible user scenarios for potential user groups. Also, algorithms and data structures to be used take place in this section.

4.1 Use Cases / Scenarios

4.1.1 Register

Participating actor(s): User, Company, Editor

Entry condition:

- Actors have entered the system.

Flow of events:

1. Actors enter a username.
2. Actors enter a password.
3. Actors select the account type.
4. Actors click on “Create New Account”
5. A new account has been created.

Exit condition:

- Actors exit the system without entering their credentials.

Alternative flow of events:

- The system gives an error because an account with the given username already exists.

4.1.2 Login

Participating actor(s): User, Company, Editor

Entry condition:

- Actors have entered the system.

Flow of events:

6. Actors enter a username.
7. Actors enter a password.
8. Actors select the account type.
9. Actors click on “Login”
10. The entered username password combination is correct the user is redirected to the home page.

Exit condition:

- Actors exit the system without entering their credentials.

Alternative flow of events:

- The system gives an error because no account with the given username and password exists.

4.1.3 View and Edit Profile

Participating actor(s): User, Company, Editor

Entry condition:

- Actors should be registered and logged in to the system.

Flow of events:

1. Actors open a profile page. Actors are able to view any profile but to edit they need to have an authority. To edit a profile, actors click on “Edit Profile” button available on the profile page. The system shows a new page that includes information about the profile which can be edited.
2. Actors make changes to their profile information and save them. The system changes the new data with the old one, and displays it to the actor.

Exit condition:

- Actors exit the system or changes to another page.

4.1.4 User Solves Coding Challenges to Practice

Participating actor(s): User

Entry condition:

- User should be registered and logged in to the system.

Flow of events:

1. User chooses coding challenges option from the practises.
2. System lists practice coding challenge questions.
3. User filters desired category.
4. System lists coding challenge questions in the desired category.
5. User chooses a challenge and submits his/her solution.
6. Solution is checked with test cases by system.
7. Answer is correct according to all the test cases.

Exit condition:

- User exits the question or the system by completing or not completing the challenge.

Alternative flow of events:

- Answer is not correct to all or some test cases, user resolves the question and makes a new submission. He/She is also able to preview his/her submissions.

4.1.5 User Participates in Coding Contests

Participating actor(s): User

Entry condition:

- User should be registered and logged in to the system.

Flow of events:

1. Users lists all ongoing coding contests.
2. User selects a coding contest.
3. User types their answers to questions after selecting the programming language.
4. Answer is correct points are added to the user.

Exit condition:

- User exits the question or the system by completing or not completing the challenge.

Alternative flow of events:

- Answer is not correct to all or some test cases, user resolves the question and makes a new submission. He/She is also able to preview his/her submissions.

4.1.6 User Upvotes/Downvotes Non-Coding Questions

Participating actor(s): User

Entry condition:

- User should be registered and logged in to the system.

Flow of events:

1. User chooses non-coding challenges option from the practises.
2. System lists non-coding questions.
3. User filters desired category.
4. System lists non-coding questions in the desired category.
5. User chooses a question and sees all answers on that question.
6. User either upvotes or downvotes a specific answer.

Exit condition:

- User exits the system or reviews the non-coding question.

4.1.7 Editor Prepares Coding Contest

Participating actor(s): Editor

Entry condition:

- Editor should be registered and logged in to the system.

Flow of events:

1. Editor click on creating a coding contest.
2. Editor picks the coding challenges that will be in the contest.
3. Editor enters a time period for the contest.
4. A new contest has been created.

Exit condition:

- Editor exits the system without providing the necessary information needed to create a contest.

4.1.8 Editor Creates Coding Challenge

Participating actor(s): Editor

Entry condition:

- Editor should be registered and logged in to the system.

Flow of events:

1. Editor selects creating new coding challenges.
2. Editors enters the name, description, category, supported programming languages, difficulty and test cases of the challenge.
3. A new challenge is created on the given category.

Exit condition:

- Editor exits the system without providing the necessary information needed to create a coding challenge.

4.1.9 Company Prepares Interview Coding Challenge

Participating actor(s): Company

Entry condition:

- Company should be registered and logged in to the system.
- There should be already coding challenges created in the system.

Flow of events:

1. Company selects creating a interview coding challenge.
2. Company selects the challenges which they want to put to the interview from challenges that were created by editors.
3. A new interview coding challenge is created.

Exit condition:

- Company exits the the system by either creating a interview challenge or not.

4.1.10 Company Reviews Interview and Noties a User

Participating actor(s): Company

Entry condition:

- Company should be registered and logged in to the system.

Flow of events:

1. Company displays the result of the interview.
2. Company notifies the user that the interview was successful and they want to get in contact with him.

Exit condition:

- Company exits the system or the selected interview by either notifying a user or not.

Alternative flow of events:

- If the user didn't met expectations, company notifies the user that the interview was unsuccessful.

4.1.11 User Solves Survival Challenge

Participating actor(s): User

Entry condition:

- User should be registered and logged in to the system.

Flow of events:

1. User list all survival challenges
2. User selects a survival challenge
3. User completed the challenge within the time limit.
4. Additional time has been added to remaining time and next challenge pops up.

Exit condition:

- User exits the system with or without completing the challenge.

Alternative flow of events:

- Time has run out and challenge has finished.

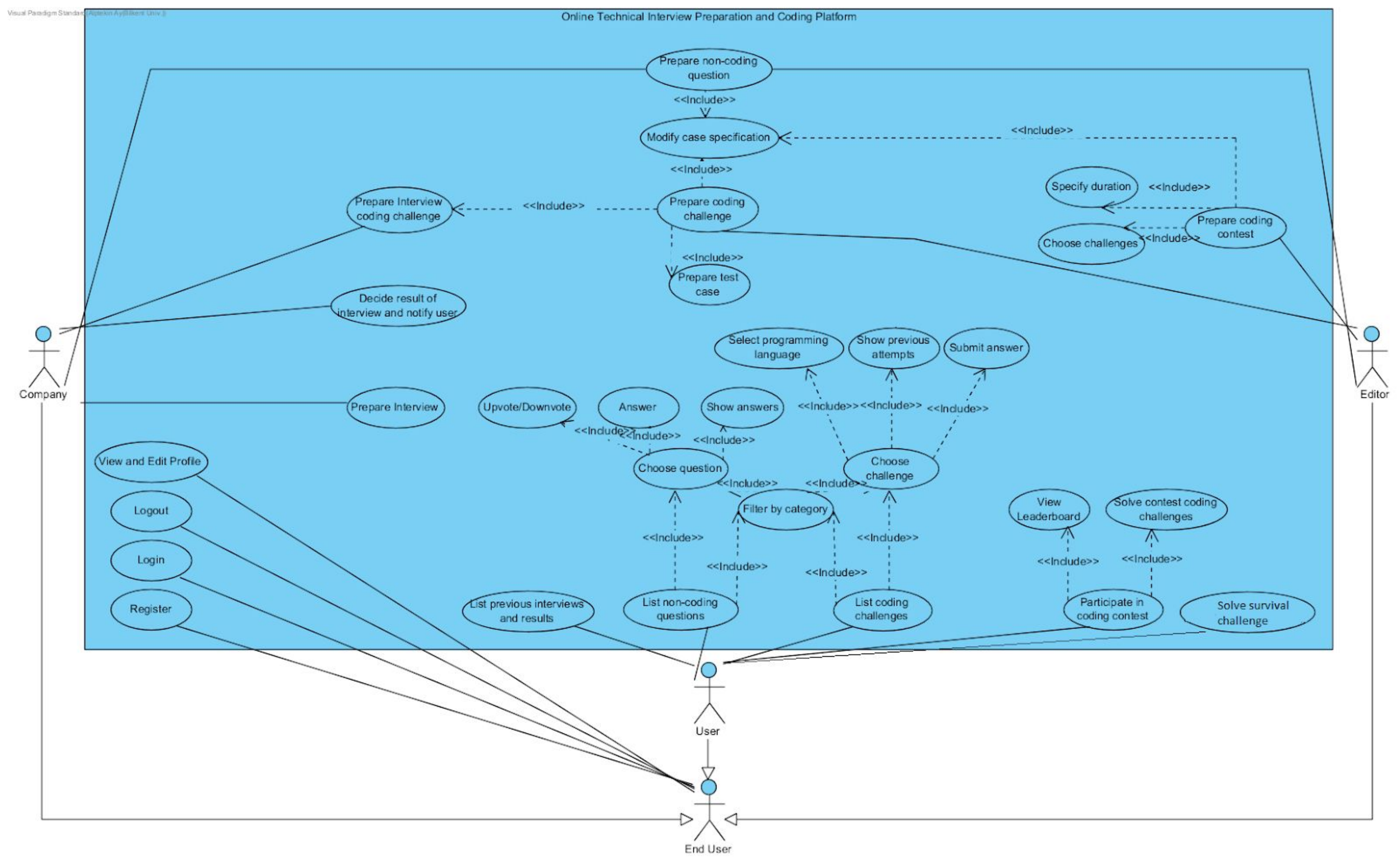

Visual Paradigm Standard(Alperen Ay(Bilkent Univ.))
Online Technical Interview Preparation and Coding Platform
Company
Editor
User
End User
Prepare non-coding question
<<Include>>
Modify case specification
<<Include>>
Prepare coding challenge
<<Include>>
Prepare Interview coding challenge
<<Include>>
Prepare test case
Specify duration
<<Include>>
Choose challenges
<<Include>>
Prepare coding contest
<<Include>>
Decide result of interview and notify user
Prepare Interview
Select programming language
Show previous attempts
Submit answer
<<Include>>
<<Include>>
<<Include>>
Upvote/Downvote
Answer
Show answers
<<Include>>
<<Include>>
<<Include>>
Choose question
Choose challenge
<<Include>>
<<Include>>
Filter by category
View and Edit Profile
Logout
Login
Register
<<Include>>
<<Include>>
<<Include>>
<<Include>>
List previous interviews and results
List non-coding questions
List coding challenges
View Leaderboard
Solve contest coding challenges
<<Include>>
<<Include>>
Participate in coding contest
Solve survival challenge

Fig. 2: UML Use Case Diagram

4.2 Algorithms

4.2.1 Score-Related Algorithms

Every coding contest contains various contest coding challenges. Users who participate in these contests obtain various scores and the users are ranked in the contest's leaderboard according to their scores. Therefore, our system requires a score calculating algorithm to form leaderboards. Score of users will be calculated according to the number of correctly solved test cases of a coding challenge. Scores obtained by the users in (that contest's) coding challenges will be summed to find the total score of a user.

In our system users gain category specific badges by their progresses in practise questions in that category. Score of users will be calculated according to the number of correctly solved test cases of a coding challenge. When a user reaches a predetermined number of total score he/she will earn that badge. (*i.e.* A user solves 10 practice coding challenges in the data structures category. If we assume that all challenges have 10 test cases and test cases are 10 pts. each, user will have 100 pts. If the predetermined score of "data structures badge" is 200 pts., user will need to solve 10 more practise challenges.)

4.2.2 Survival Challenge Algorithm

In the survival practise, users can challenge themselves against time. They will start the survival challenge with an initial time and every challenge they solve correctly, they will earn extra time. An algorithm to calculate the remaining time is required. (Survival challenge score will also be calculated according to the section 4.2.1.)

4.2.3 Test Case Algorithm

Solution submissions of users are required to be checked with test cases. Functions that are coded by users get inputs from test cases and outputs accordingly. The user's code outputs are required to be compared with that test case's output. An algorithm is required for this comparison.

4.3 Data Structures

For the attribute domains we will use char, varchar, date and int domains in the MySql tables.

5. User Interface Design and Corresponding SQL Statements

5.1 Register Page

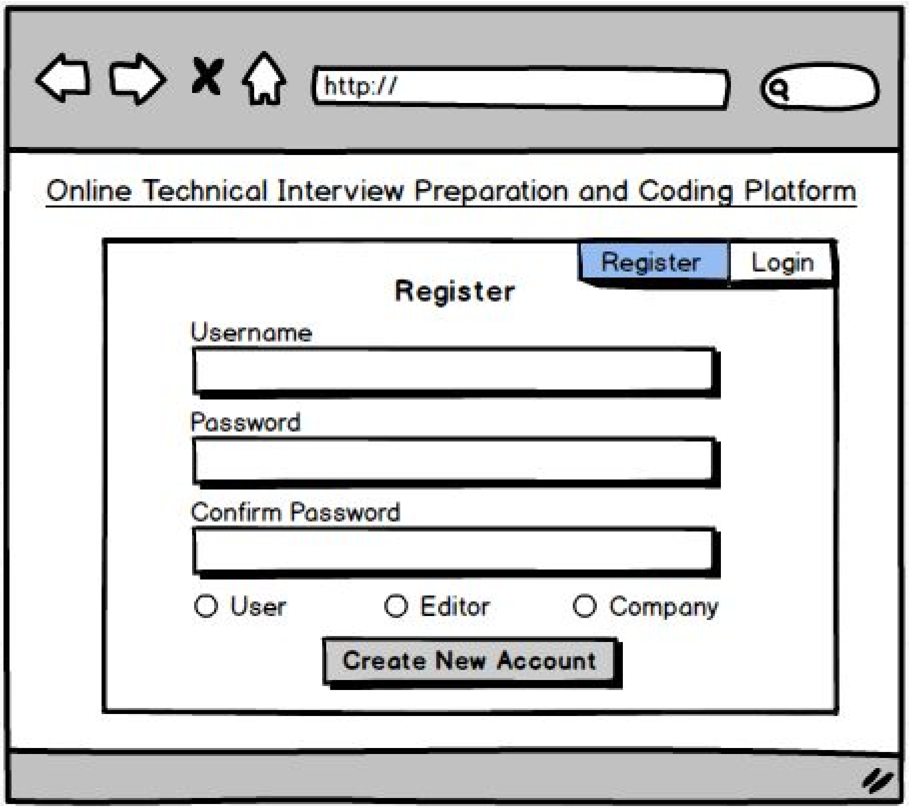


Figure 3: Register Page

If user has no account, he/she can create one by selecting register tab. To register, all users have to select a distinct username and password. Password is asked for two times in terms of reduce the likelihood of typo. Specifying type of the account is also needed here.

SQL Statements:

Checking If User Exists

```
SELECT username
FROM end_user
WHERE end_user.username = @username;
```

Registering a User

```
INSERT INTO end_user
VALUES (@username, @password, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
```

5.2 Login Page

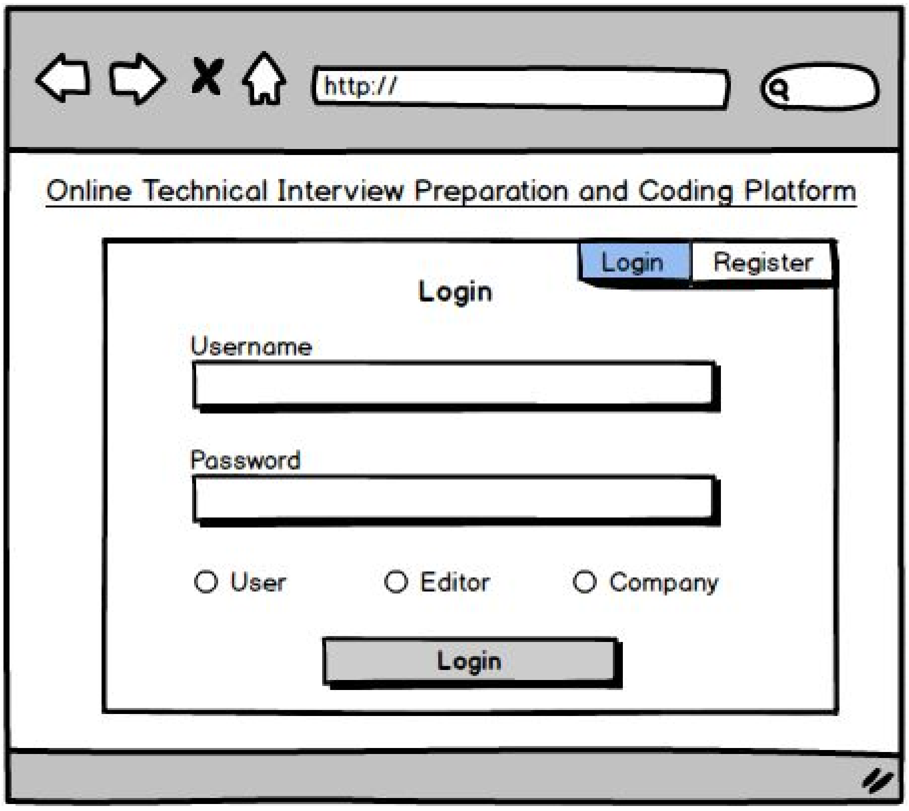


Figure 4: Login Page

In this page, user can login if he/she has already an account. Specifying type of the account (user account, editor account or company account) is needed for login process.

SQL Statements:

```
SELECT username, password
FROM end_user
WHERE end_user.username = @username
AND end_user.password = @password;
```

5.3 Users' Main Menu

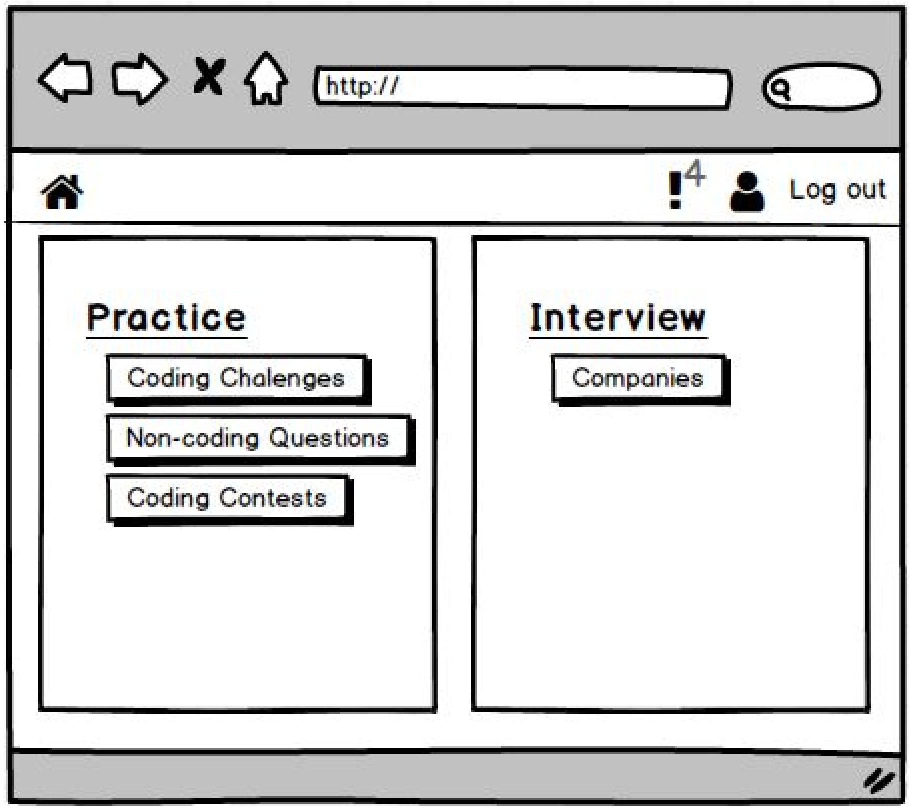


Figure 5: Main Menu for User

In this page, user can select any 3 options to practice or press "Companies" button to search the companies he/she wants to solve their interview questions. When user presses the "Companies" button, a list of companies show up. After the selection of the company, user selects the position of the job and starts the interview.

Other than that, user has a main menu button, profile page button, a logout button on the top bar. Furthermore, user can press the exclamation mark button in order to view his/her notifications from the interviews which is on the top bar.

SQL Statements:

Listing Coding Challenges

```
SELECT challenge_name
FROM practise coding challenges;
```

Listing Coding Challenges (Filtered by Category)

```
SELECT challenge_name, category_name
FROM practise_coding_challenges, category c
WHERE c.category_name = @category_name;
```

Listing Non-Coding Questions

```
SELECT question
FROM non_coding_question;
```

Listing Non-Coding Challenges (Filtered by Category)

```
SELECT question, category_name
FROM non_coding_question, category c
WHERE c.category_name = @category_name;
```

Listing Coding Contests

```
SELECT contest_name, specifications, duration
FROM coding_contest;
```

Listing Available Interviews

```
SELECT name, position, duration, name, company_id
FROM company c, interview i
WHERE c.company_id = i.company_id;
```

5.4 Editors' Main Menu

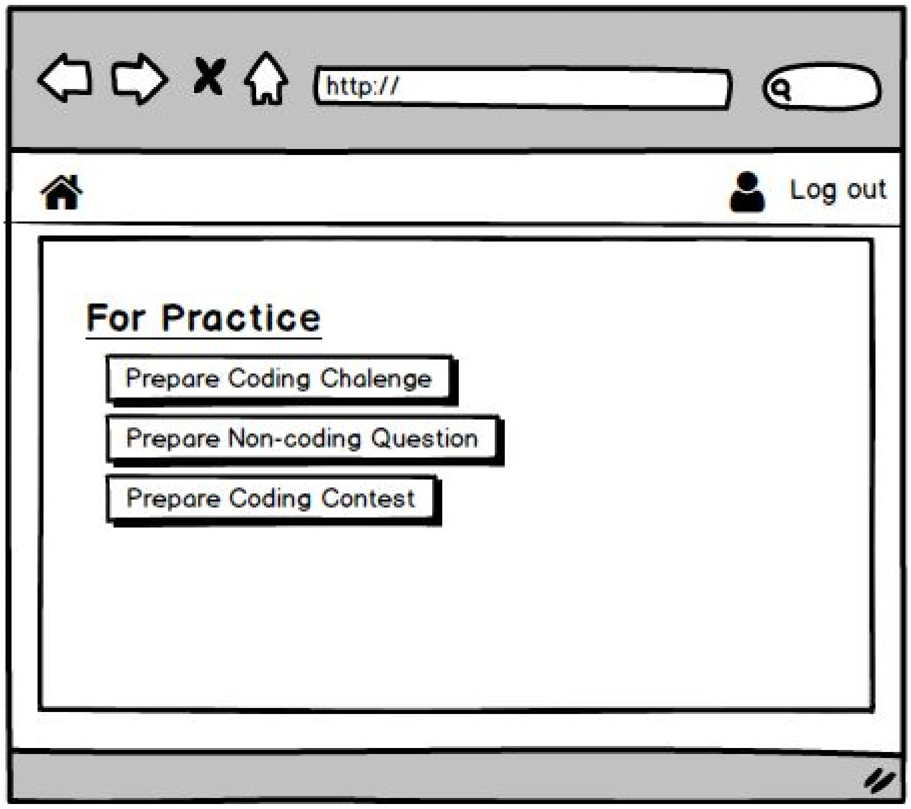


Figure 6: Main Menu for Editor

In this page, editor can select any 3 options.

SQL Statements:

Preparing New Challenges, Questions and Contests

INSERT INTO coding_challenges (challenge_id, challenge_name, test_case, text)

VALUES (value1, value2, value3, value4);

INSERT INTO non_coding_question (question_id, question)

VALUES (value1, value2);

INSERT INTO coding_contest (contest_id, contest_name, specifications, duration)

VALUES (value1, value2, value3, value4);

5.5 Companies’ Main Menu

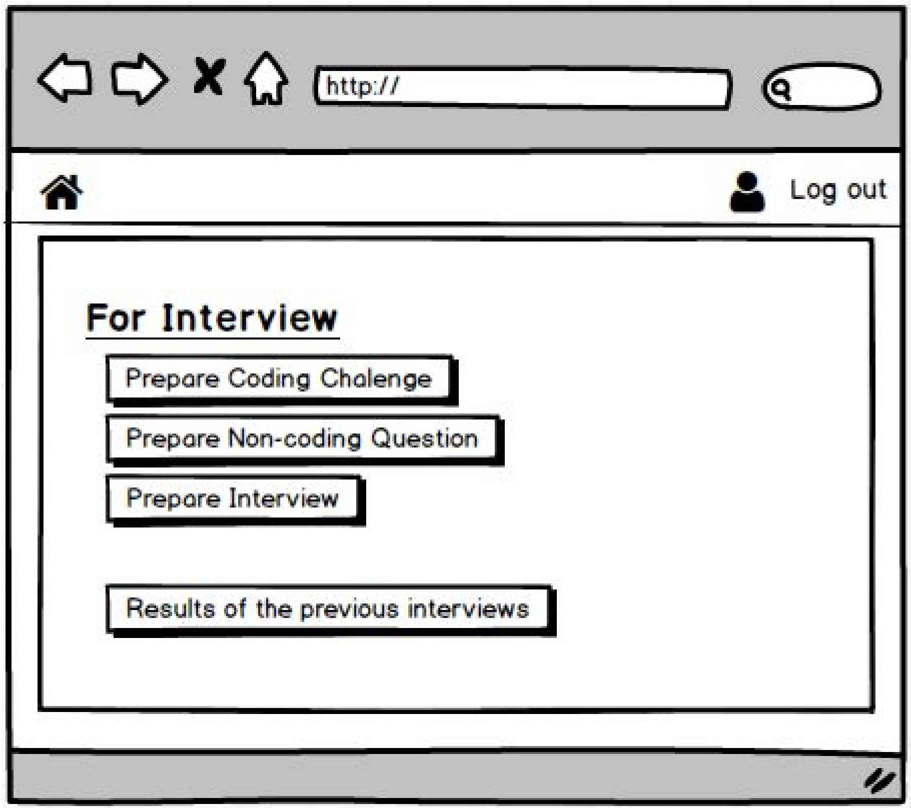


Figure 7: Main Menu for Company

In this page, company can select any 4 options.

SQL Statements:

Preparing New Challenges, Questions and Interviews

INSERT INTO coding_challenges (challenge_id, challenge_name, test_case, text)

VALUES (value1, value2, value3, value4);

INSERT INTO non_coding_question (question_id, question)

VALUES (value1, value2);

INSERT INTO interview (interview_id, company_id, position, duration, challenges, specifications)

VALUES (value1, value2, value3, value4, value5, value6);

5.6 Profile Page

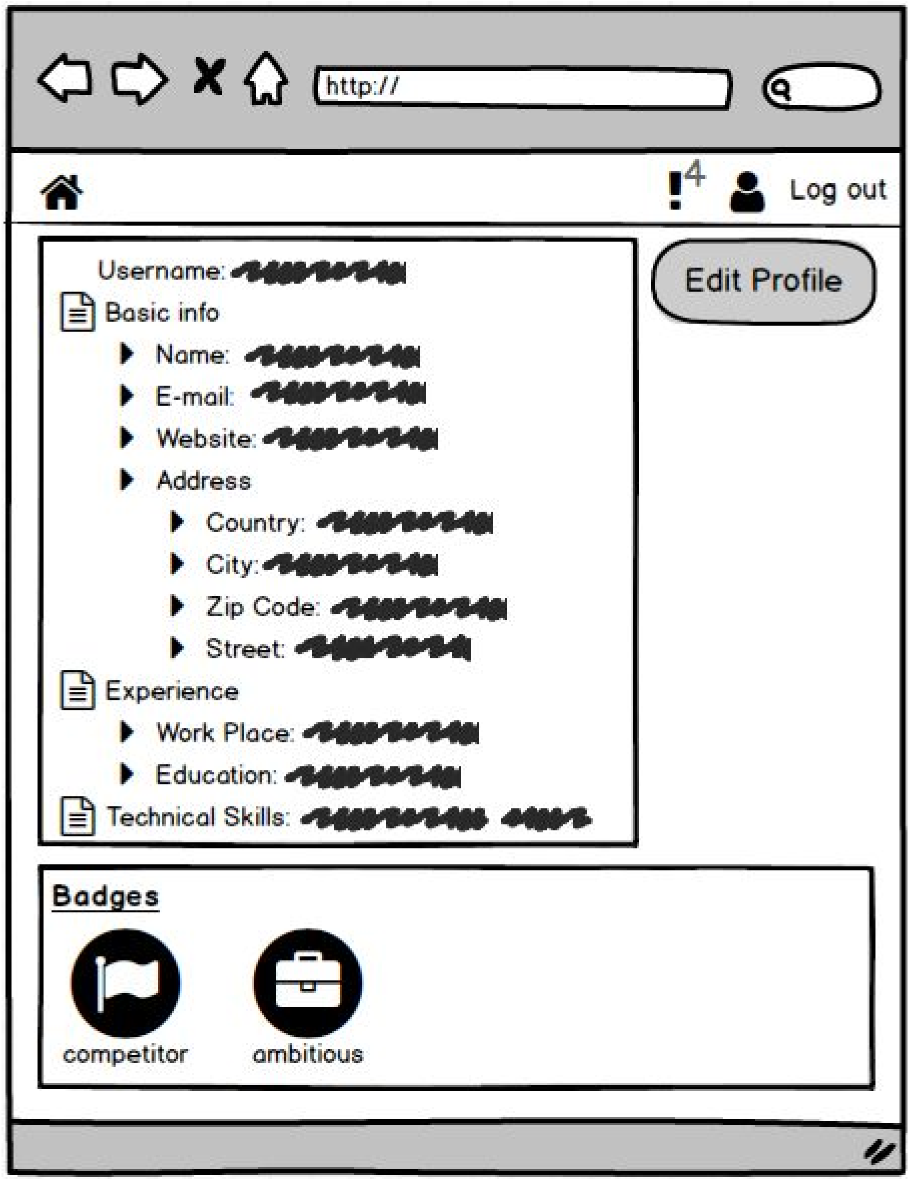


Figure 8: Profile Page

All three type of the account has common features such as username, name, email, website, country, city, zip code and street. For user account, there are extra information which are work place, education and technical skills. For company account, there is only one extra information which is phone number. Moreover, changing these information can be done in here by pressing “Edit Profile” button.

SQL Statements:

Editing Profile Information of User

```
UPDATE user
SET
        name = @name,
        email = @email,
        website = @website,
        country = @country,
        city = @city,
        zip_code = @zip_code,
        street = @street,
        work_place= @work_place,
        education= @education,
        technical_skills= @technical_skills
WHERE user.username = @username;
```

Editing Profile Information of Editor

```
UPDATE editor
SET
        name = @name,
        email = @email,
        website = @website,
        country = @country,
        city = @city,
        zip_code = @zip_code,
        street = @street
WHERE editor.username = @username;
```

Editing Profile Information of Company

```
UPDATE company
SET
        name = @name,
        email = @email,
        website = @website,
        country = @country,
        city = @city,
        zip_code = @zip_code,
        street = @street,
        phone_number= @phone_number
WHERE company.username = @username;
```

5.7 Category

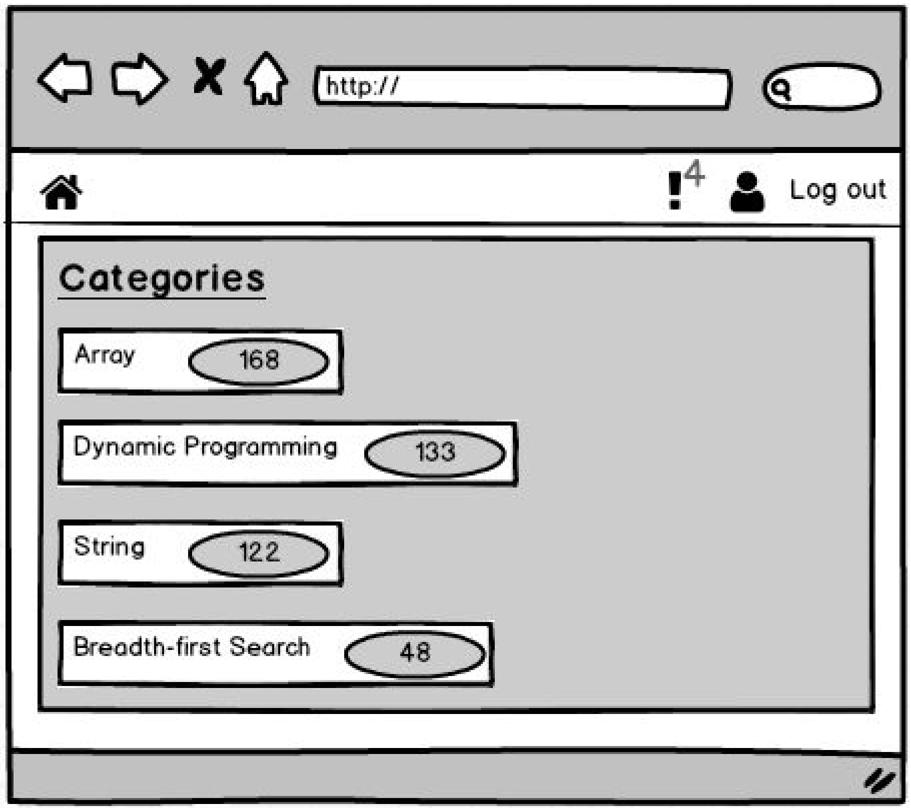


Figure 9: Categories

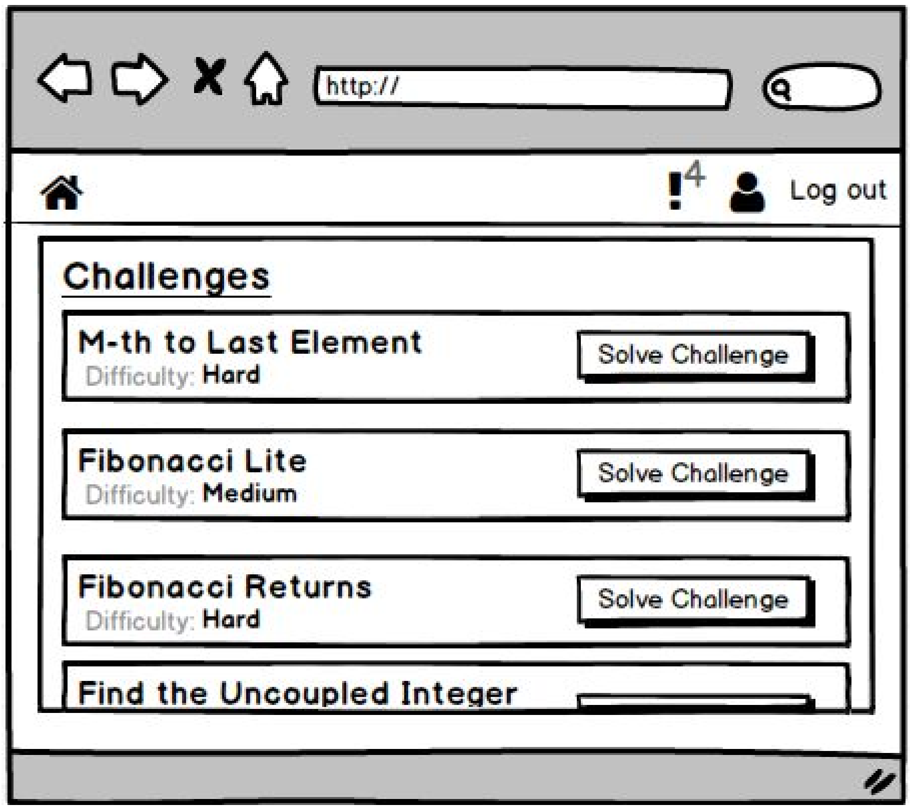
In this page, categories of coding challenges can be seen with their number of coding challenges. Also, categories of non-coding questions looks exactly like that. User picks one of the categories.

SQL Statements:

```
SELECT category_name, CNT
FROM category, (SELECT count(*) CNT FROM coding_challenges);

SELECT category_name, CNT
FROM category, (SELECT count(*) CNT FROM non_coding_challenges);
```

5.8 Coding Challenges


Challenges

M-th to Last Element
Difficulty: Hard
Solve Challenge

Fibonacci Lite
Difficulty: Medium
Solve Challenge

Fibonacci Returns
Difficulty: Hard
Solve Challenge

Find the Uncoupled Integer

Figure 10: List of Coding Challenges

In this page coding challenges are listed with their names and difficulties. Non-coding questions' list looks almost like coding challenges.

SQL Statements:

Listing Coding Challenges

SELECT challenge_name, difficulty

FROM practise coding challenges;

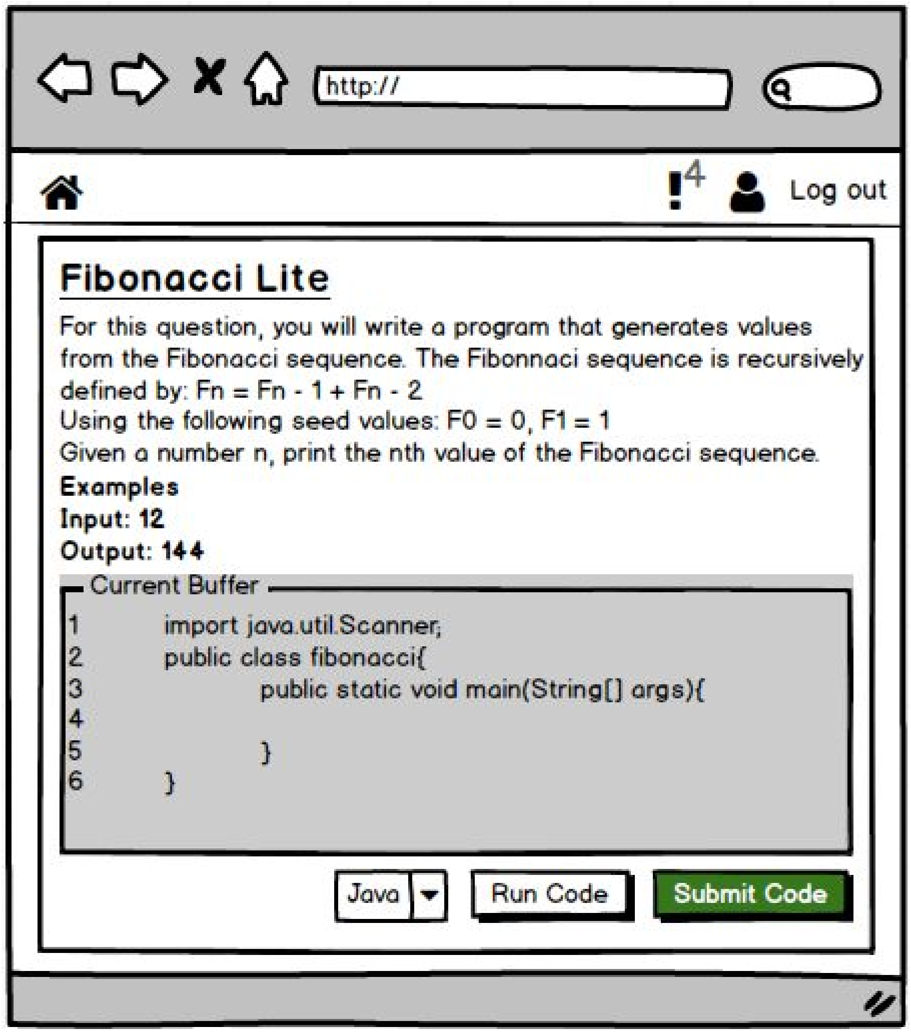
Listing Coding Challenges (Filtered by Category)

SELECT challenge_name, category_name, difficulty

FROM practise_coding_challenges, category c

WHERE c.category_name = @category_name;

5.9 A Coding Challenge



Fibonacci Lite

For this question, you will write a program that generates values from the Fibonacci sequence. The Fibonnaci sequence is recursively defined by: Fn = Fn - 1 + Fn - 2
Using the following seed values: F0 = 0, F1 = 1
Given a number n, print the nth value of the Fibonacci sequence.

Examples
Input: 12
Output: 144

Current Buffer

```
import java.util.Scanner;
public class fibonacci{
        public static void main(String[] args){

        }
}
```

Figure 11: A Coding Challenge

In this page, user can select the programming language of the compiler. The code can be tested and submitted afterwards.

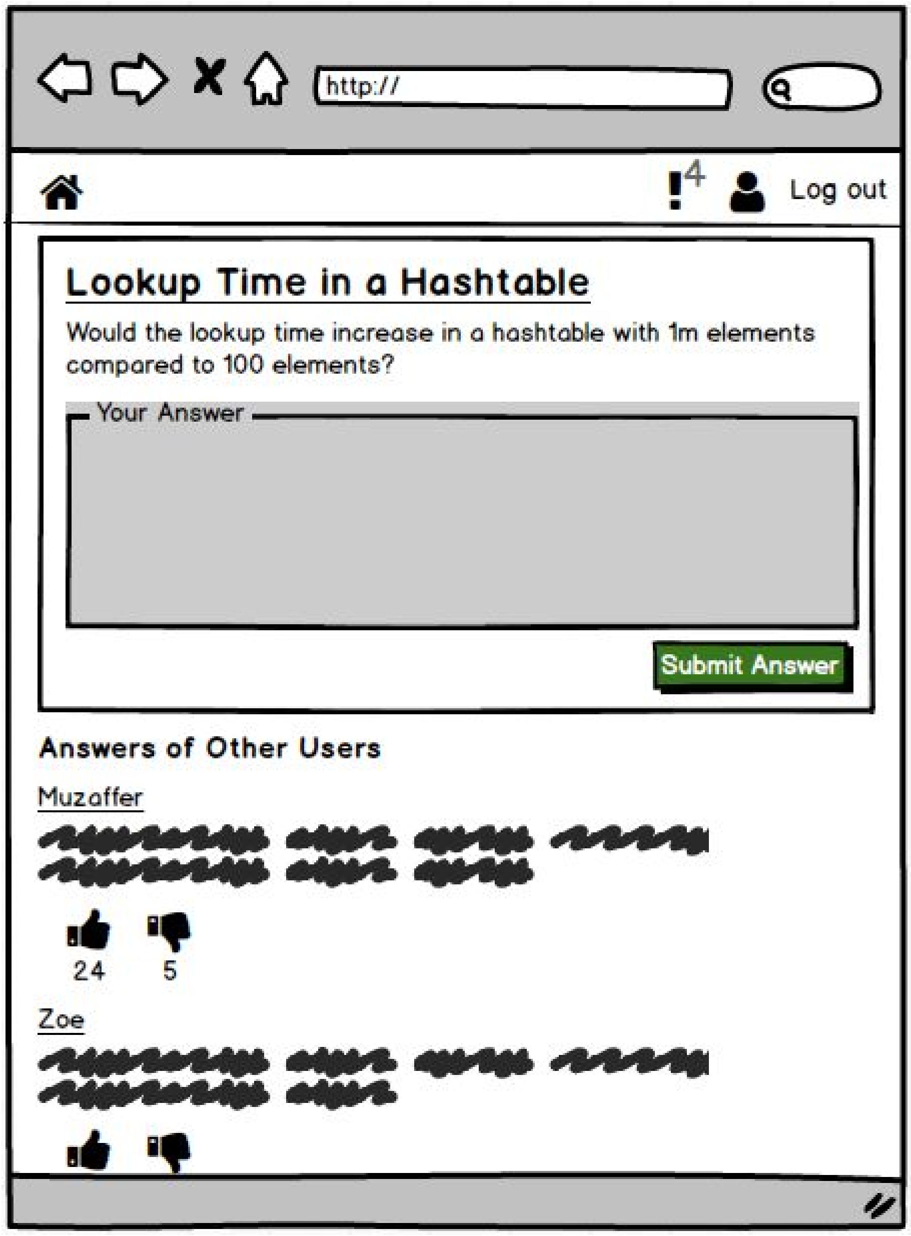
SQL Statements:

SELECT *

FROM coding_challenges;

INSERT INTO submission(sub_id, text, score) VALUES (val1, val2, val3);

5.10 A Non-coding Question



Lookup Time in a Hashtable

Would the lookup time increase in a hashtable with 1m elements compared to 100 elements?

Answers of Other Users

Muzaffer

Zoe

Figure 12: A Non-coding Question

In this page, user writes his/her answer and submits the answer. Also, the user can upvote or downvote others' answers.

SQL Statements:

Showing Question, Its Answers and Its Rating

```
SELECT question, text, rating
FROM answer;
```

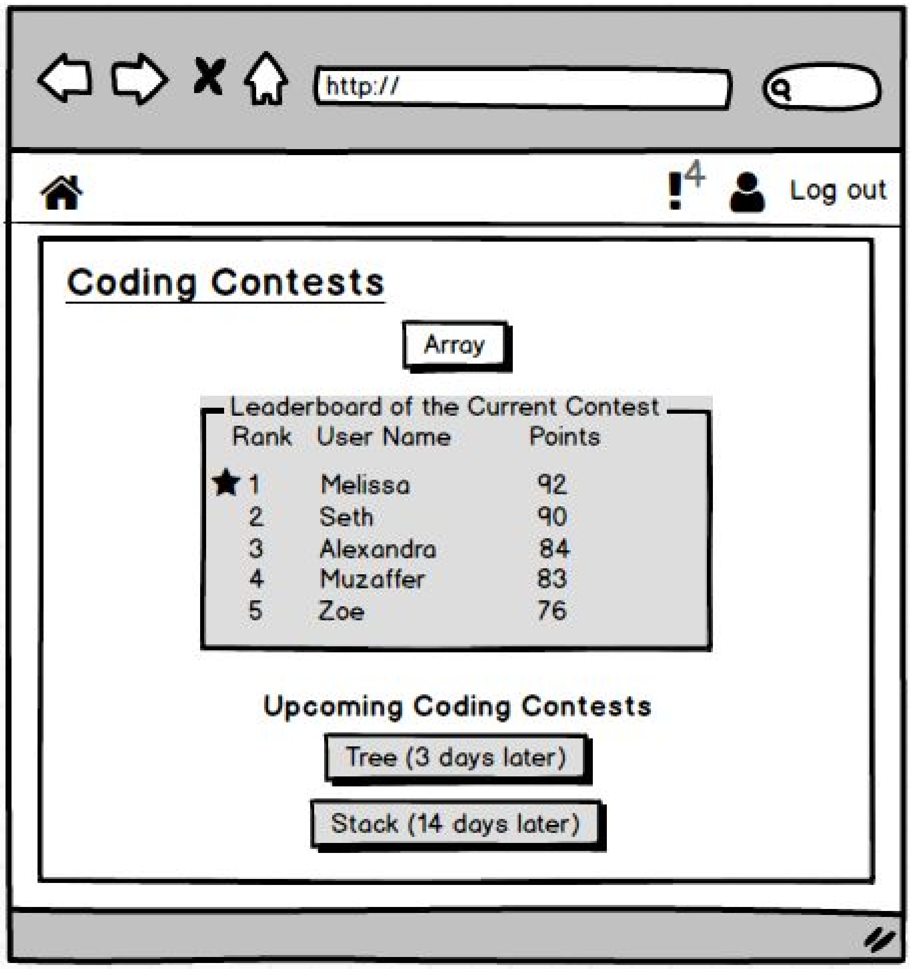
User Rates an Answer

```
UPDATE answer
SET rating = @rating;
```

User Answers the Question

```
INSERT INTO answer(text)
VALUES (@text);
```

5.11 A Coding Contest



Leaderboard of the Current Contest

Rank	User Name	Points
★ 1	Melissa	92
2	Seth	90
3	Alexandra	84
4	Muzaffer	83
5	Zoe	76

Figure 13: A Coding Contest

In this page, user can see the leaderboard of the current contest and the upcoming contests. User can select the current coding contest which consists several coding challenges based on the topic.

SQL Statements:

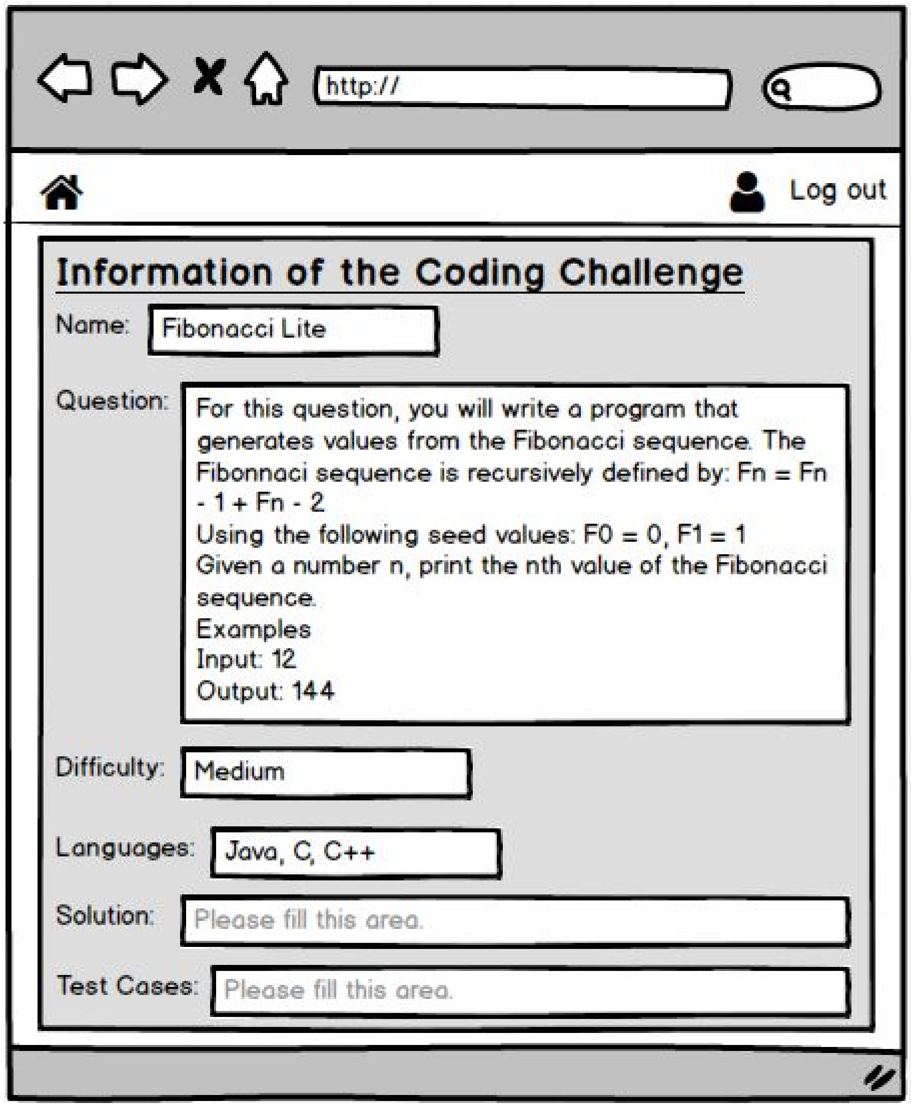
Listing Current Contest’s Leaderboard

```
SELECT leaderboard
FROM coding_contest
WHERE date - CURDATE() > 0;
```

Listing Upcoming Contests

```
SELECT contest_name, date - CURDATE()
FROM coding_contest
WHERE date > CURDATE();
```

5.12 Prepare



Information of the Coding Challenge

Name: Fibonacci Lite

Question: For this question, you will write a program that generates values from the Fibonacci sequence. The Fibonnaci sequence is recursively defined by: Fn = Fn - 1 + Fn - 2
Using the following seed values: F0 = 0, F1 = 1
Given a number n, print the nth value of the Fibonacci sequence.
Examples
Input: 12
Output: 144

Difficulty: Medium

Languages: Java, C, C++

Solution: Please fill this area.

Test Cases: Please fill this area.

Figure 14: Preparing a Coding Challenge

In this page, company or editor user types can prepare a coding challenge by defining its information. Preparing a non-coding question is almost the same except it does not have test cases, a solution and the language of the question. Preparing an interview or a coding contest is just selecting multiple coding challenges.

SQL Statements:

INSERT INTO coding_challenges (challenge_id, challenge_name, test_case, text)
VALUES (value1, value2, value3, value4);

5.13 Results of the Interviews and Informing Users

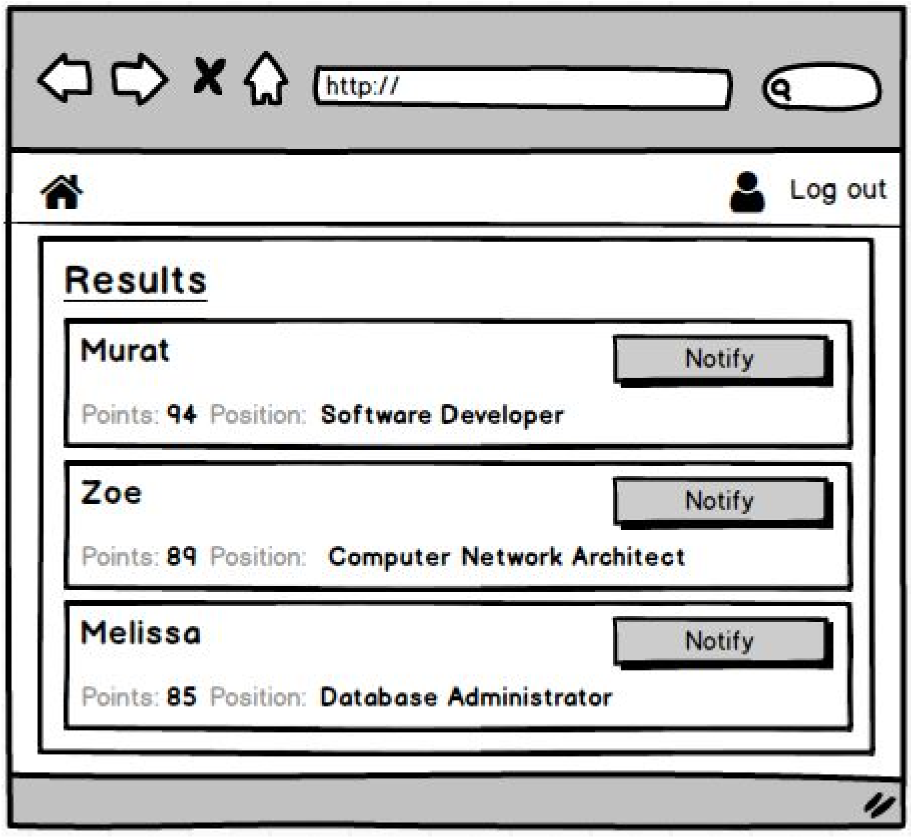

http://
Log out
Results
Murat
Notify
Points: 94 Position: Software Developer
Zoe
Notify
Points: 89 Position: Computer Network Architect
Melissa
Notify
Points: 85 Position: Database Administrator

Figure 15: Results of the Interviews

In this page, company user type can see the results of the previous interviews and notify the successful users.

SQL Statements:

SELECT username, score, position
FROM conducts;

6. Advanced Database Components

6.1 Views

6.1.1 Leaderboards by Categories

This view provides username-score tables for all categories. These tables are in ascending order in terms of scores.

```
CREATE VIEW scores_categories as
SELECT username,category_name,score
FROM  submission_challenge NATURAL JOIN submission NATURAL JOIN category
GROUP BY category_name
ORDER BY scores DESC
```

6.1.2 Contests by Date

This view provides a table of challenges and dates. These tables are ordered in terms of Create date.

```
CREATE VIEW most_recently_added_contests
SELECT contest_id,category_name,date ,TIMESTAMPDIFF (YEAR,
patient_date_of_birth,CURDATE()) AS age
FROM  coding_contest
ORDER BY age ASC
```

6.1.3 Top Rated Editors

This view provides a ordered table of the editors respect to their question's upvote amount.

```
CREATE VIEW top_rated_editors
SELECT username,count(rating)
FROM (editor inner join prepares on editor.username = prepares.username) inner join on
answer on answer.question_id = non_coding_question.question_id
WHERE rating = "upvote"
GROUP BY editor.username
ORDER BY count(rating) ASC
```

6.2 Stored Procedures

View of contest, coding challenges and non-coding questions will be stored procedure because of its existence as a main functionality and frequent usage in program UI. Tuples of this entities also will be taken from a view to be protected from unprivileged information leakage.

6.3 Reports

6.3.1 Total Number of Interviews Annually for Each Position

Number of interviews that were inserted in DB system in last year will be calculated,classified respect to position type and provided by this report.

```
SELECT position, count(interview_id) as interview_numbers
FROM interview inner join interviewer_interviewee on interview.interview_id =
interviewer_interviewee.interview_id
WHERE interview.date  >= DATE(NOW()) - INTERVAL 365 DAY
GROUP BY position
```

6.3.2 Total Number of Coding Contests Annually for Each Category

Number of contests that were initialized to DB system in last year will be calculated, classified respect to category and provided by this report.

```
SELECT category_name, count(contest_id) as contest_numbers
FROM coding_contest inner join coding_contest_category on coding_contest.contest_id =
coding_contest_category.contest_id
WHERE contest.date >= DATE(NOW()) - INTERVAL 365 DAY
GROUP BY category_name
```

6.3.3 Total Number of New Users Annually for Each User Type

Number of users that registered to DB system in last year will be calculated, classified respect to user type and provided by this report.

```
SELECT user_type, count(username) as user_counts
FROM (SELECT user_type,username
        FROM user
        WHERE user.reg_date >= DATE(NOW()) - INTERVAL 365 DAY
        UNION
        SELECT user_type,username
        FROM editor
        WHERE editor.reg_date >= DATE(NOW()) - INTERVAL 365 DAY
        UNION
        SELECT user_type,username
        FROM company
        WHERE company.reg_date >= DATE(NOW()) - INTERVAL 365 DAY)
GROUP BY user_type
```

6.3.4 Total Number of Each User Type

Total number of each user type is calculated, classified respect to their user_type, and provided via this report.

```
SELECT user_type, count(username) as tot_user_counts
FROM (SELECT user_type,username
          FROM user
          UNION
          SELECT user_type,username
          FROM editor
          UNION
          SELECT user_type,username
          FROM company)
GROUP BY user_type
```

6.4 Constraints

- System cannot be used without logging in.
- End users cannot have the same username.
- Entity id's are going to be integers, auto-incremented by 1, and NOT NULL.
- A user can participate in a contest only once.
- Contest are only available for certain durations.
- A user can participate in an interview only once.
- Interviews have time limits. Users are done after time ends.
- Interviews also have a date, it is not available after the date passed.
- Users can only evaluate a non-coding question once, but they can change their decision.

6.5 Triggers

Existence of triggers are mandatory to prevent undesired and illogical situations. Therefore, following triggers are used:

- Each user has right to participate a coding contest only once, so a trigger is used to prevent multiple participation from same user. This trigger will be called after insertions on answer table if there is another tuple with same username-contest_id.

- Coding challenges consist of one or more questions, and this questions has consecutive question numbers. To ensure questions has consecutive numbers, a trigger will be triggered after insertions and deletion to prepares table.

- Each user should have distinct e-mail address, so e-mail address should be unique for each user. Therefore, a trigger is placed after insertion to end_user table check whether there is another user with the same e-mail address or not.

7. Implementation Plan

We will implement the database in our project using MySQL. PHP will be used for implementing web development and HTML, CSS and Javascript will be used for user interface development.