BILKENT UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

CS353
DATABASE SYSTEMS

TECHNICAL INTERVIEW AND CODING PLATFORM DESIGN REPORT

Ahmet Ayrancıoğlu
21601206
Deniz Dalkılıç
21601896
Kaan Gönç
21602670

Table of Contents

1. Revised E/R Model

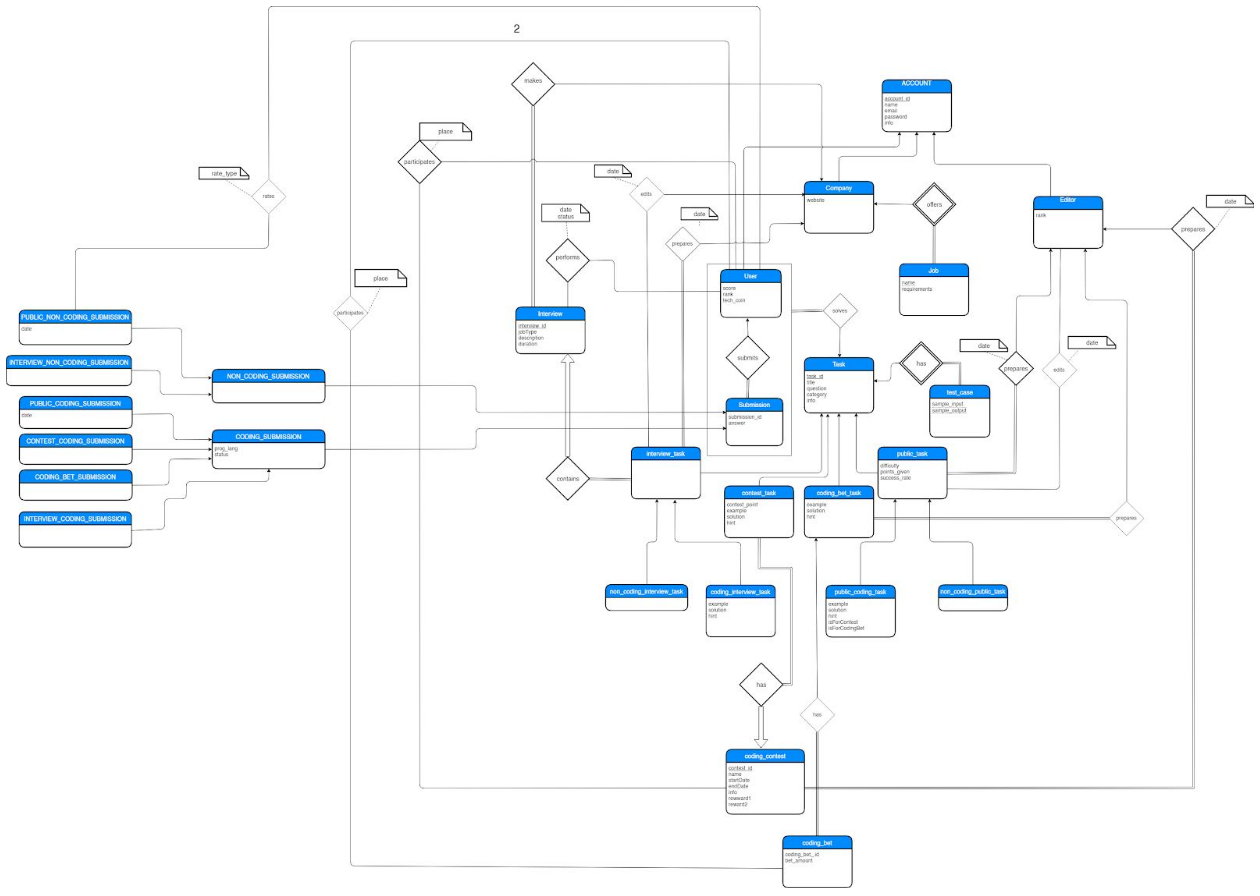


Figure 1 - E/R Diagram
(See Appendix 1)

Changes:

- New Feature “Coding Bet Challenge” added. Two users are able to compete on a coding question. They both bet coins and the first one to solve the coding task takes it all.
- Entities for different kinds of submissions are added.
- Entities for new types of tasks are added.
- Constraints are updated. (See Constraints Section)
- Votes for non-coding questions are stored in the database for each answer
- Test cases are being stored in the database for each task
- Rewards for coding contests are no longer given to only the first place but to the first three places.
- Coding Contests are no longer composed of public tasks, instead they have specialized type of tasks.
- Primary keys for schemas are changed in a way that we can use NATURAL JOIN safely

2. Table Schemas

2.1 ACCOUNT

Relational Model:
ACCOUNT(account_id, name, email, password, info)

Foreign Keys:
None

Candidate Keys:
{account_id, email}

Table Definition:
```
CREATE TABLE ACCOUNT(
        account_id INT PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        email VARCHAR(50) NOT NULL UNIQUE,
        password VARCHAR(50) NOT NULL,
        info VARCHAR(500)
);
```

Normal Form:
account_id -> name, email, password, info
email -> account_id, name, password, info
account_id and email are super keys so the table is in 3NF.

2.2 USER

Relational Model:
USER(<u>user_id</u>, score, tech_coin)

Foreign Keys:
user_id references ACCOUNT.id

Candidate Keys:
{user_id}

Table Definition:
```
CREATE TABLE USER(
        user_id INT PRIMARY KEY,
        score INT,
        tech_coin INT,
        FOREIGN KEY(user_id) REFERENCES ACCOUNT(account_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
user_id -> score, tech_coin
user_id is a super key so the table is in 3NF.

2.3 EDITOR

Relational Model:
EDITOR(<u>editor_id</u>, rank)

Foreign Keys:
editor_id references ACCOUNT.id

Candidate Keys:
{editor_id}

Table Definition:
```
CREATE TABLE EDITOR(
	editor_id INT PRIMARY KEY,
	rank INT,
	FOREIGN KEY (editor_id) REFERENCES ACCOUNT(account_id)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

Normal Form:
editor_id -> rank
editor_id is a super key so the table is in 3NF.

2.4 COMPANY

Relational Model:
COMPANY(<u>company_id</u>, website)

Foreign Keys:
company_id references ACCOUNT.id

Candidate Keys:
{company_id}

Table Definition:
```
CREATE TABLE COMPANY(
        editor_id INT PRIMARY KEY,
        website VARCHAR(50) NOT NULL,
        FOREIGN KEY (company_id) REFERENCES ACCOUNT(account_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
company_id -> website
company_id is a super key so the table is in 3NF.

2.5 JOB

Relational Model:
JOB(<u>company_id, name</u>, requirements)

Foreign Keys:
company_id references COMPANY.company_id

Candidate Keys:
{(company_id, name)}

Table Definition:
```
CREATE TABLE JOB(
	company_id INT NOT NULL,
	name VARCHAR(50) NOT NULL,
	requirements VARCHAR(50),
	FOREIGN KEY (company_id) REFERENCES COMPANY
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	PRIMARY_KEY((company_id, name))
);
```

Normal Form:
company_id, name -> website
company_id and name are super keys so the table is in 3NF.

2.6 INTERVIEW

Relational Model:
INTERVIEW(<u>interview_id</u>, job_type, description, duration)

Foreign Keys:
None

Candidate Keys:
{interview_id}

Table Definition:
```
CREATE TABLE INTERVIEW(
	interview_id INT PRIMARY KEY,
	job_type VARCHAR(50) NOT NULL,
	description VARCHAR(50),
	duration INTERVAL NOT NULL
);
```

Normal Form:
interview_id -> job_type, description, duration
ineterview_id is a super key so the table is in 3NF.

2.7 CODING_CONTEST

Relational Model:
CODING_CONTEST(contest_id, name, start_date, end_date, reward1, reward2, reward3)

Foreign Keys:
None

Candidate Keys:
{contest_id, name}

Table Definition:
```
CREATE TABLE CONTEST(
        contest_id INT PRIMARY KEY,
        name VARCHAR(50) NOT NULL UNIQUE,
        start_date TIMESTAMP NOT NULL,
        end_date TIMESTAMP NOT NULL,
        reward1 INT NOT NULL,
        reward2 INT NOT NULL,
        reward3 INT NOT NULL
);
```

Normal Form:
contest_id-> name, start_date, end_date, reward1, reward2, reward3
name -> contest_id, start_date, end_date, reward1, reward2, reward3
company_id and name are super keys so the table is in 3NF.

2.8 CODING_BET

Relational Model:
CODING_BET(coding_bet_id, bet_amount)

Foreign Keys:
None

Candidate Keys:
{coding_bet_id}

Table Definition:
```
CREATE TABLE CODING_BET(
	codiıng_bet_id INT PRIMARY KEY,
	bet_amount INT NOT NULL,
);
```

Normal Form:
coding_bet_id -> bet_amount
coding_bet_id is a super key so the table is in 3NF.

2.9 TASK

Relational Model:
TASK(task_id, title, question, category, info)

Foreign Keys:
None

Candidate Keys:
{task_id, title, question}

Table Definition:

```
CREATE TABLE TASK(
        task_id INT PRIMARY KEY,
        title VARCHAR(50) NOT NULL UNIQUE,
        question VARCHAR(500) NOT NULL UNIQUE,
        category VARCHAR(20) NOT NULL,
        info VARCHAR(500)
);
```

Normal Form:
task_id -> title, question, category, info
title -> task_id, question, category, info
question-> title, task_id, category, info
task_id, question and title are super keys so the table is in 3NF.

2.10 PUBLIC_TASK

Relational Model:
PUBLIC_TASK(task_id, difficulty, points_given, success_rate)

Foreign Keys:
task_id references TASK.task_id

Candidate Keys:
{task_id}

Table Definition:
```
CREATE DOMAIN difficulty_level VARCHAR(7)
CONSTRAINT difficulty_level_test
CHECK ( VALUE IN ( ‘Easy’, ‘Medium’,  ‘Hard’));
CREATE TABLE PUBLIC_TASK(
        task_id INT PRIMARY KEY,
        difficulty difficulty_level NOT NULL,
        points_given INT NOT NULL,
        success_rate INT,
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
task_id -> difficulty, points_given, success_rate
task_id is a super key so the table is in 3NF.

2.11 PUBLIC_CODING_TASK

Relational Model:
PUBLIC_CODING_TASK(<u>task_id</u>, example, solution, hint)

Foreign Keys:
task_id references PUBLIC_TASK.id

Candidate Keys:
{task_id, example, solution}

Table Definition:

```
CREATE TABLE PUBLIC_CODING_TASK(
        task_id INT PRIMARY KEY,
        example VARCHAR(500) NOT NULL UNIQUE,
        solution VARCHAR(500) NOT NULL UNIQUE,
        hint VARCHAR(500),
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
task_id -> example, solution, hint
example -> task_id, solution, hint
solution -> example, task_id, hint
task_id, example and solution are super keys so the table is in 3NF

2.12 PUBLIC_NON_CODING_TASK

Relational Model:
PUBLIC_NON_CODING_TASK(task_id)

Foreign Keys:
task_id references PUBLIC_TASK.task_id

Candidate Keys:
{task_id}

Table Definition:
```
CREATE TABLE PUBLIC_NON_CODING_TASK(
        task_id INT PRIMARY KEY,
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
task_id is a super key so the table is in 3NF.

2.13 INTERVIEW_TASK

Relational Model:
INTERVIEW_TASK(<u>task_id</u>, interview_id)

Foreign Keys:
task_id references TASK.task_id
Interview_id references INTERVIEW.interview_id

Candidate Keys:
{task_id}

Table Definition:
```
CREATE TABLE INTERVIEW_TASK(
        task_id INT PRIMARY KEY,
        Interview_id INT NOT NULL,
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (interview_id) REFERENCES INTERVIEW
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
task_id -> interview_id
task_id is a super key so the table is in 3NF.

2.14 NON_CODING_INTERVIEW_TASK

Relational Model:
NON_CODING_INTERVIEW_TASK(<u>task_id</u>)

Foreign Keys:
task_id references INTERVIEW_TASK.task_id

Candidate Keys:
{task_id}

Table Definition:
```
CREATE TABLE NON_CODING_INTERVIEW_TASK(
        task_id INT PRIMARY KEY,
        FOREIGN KEY (task_id) REFERENCES INTERVIEW_TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
task_id is a super key so the table is in 3NF.

2.15 CODING_INTERVIEW_TASK

Relational Model:
CODING_INTERVIEW_TASK(task_id, example, solution, hint)

Foreign Keys:
task_id references INTREVIEW_TASK.task_id

Candidate Keys:
{task_id, example, solution}

Table Definition:

```
CREATE TABLE CODING_INTERVIEW_TASK(
        task_id INT PRIMARY KEY,
        example VARCHAR(500) NOT NULL UNIQUE,
        solution VARCHAR(500) NOT NULL UNIQUE,
        hint VARCHAR(500),
        FOREIGN KEY (task_id) REFERENCES INTERVIEW_TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

task_id -> example, solution, hint
example -> task_id, solution, hint
solution -> example, task_id, hint
task_id, example and solution are super keys so the table is in 3NF

2.16 CONTEST_TASK

Relational Model:
CONTEST_TASK(task_id, contest_point, example, solution, hint, contest_id)

Foreign Keys:
task_id references TASK.task_id
contest_id references CODING_CONTEST.contest_id

Candidate Keys:
{task_id, example, solution}

Table Definition:
```
CREATE TABLE CONTEST_TASK(
        task_id INT PRIMARY KEY,
        contest_point INT NOT NULL
        example VARCHAR(500) NOT NULL UNIQUE,
        solution VARCHAR(500) NOT NULL UNIQUE,
        hint VARCHAR(500),
        contest_id INT NOT NULL,
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
        FOREIGN KEY (contest_id) REFERENCES CODING_CONTEST
                ON DELETE CASCADE
                ON UPDATE CASCADE

);
```

Normal Form:
task_id -> example, solution, hint
example -> task_id, solution, hint
solution -> example, task_id, hint
task_id, example and solution are super keys so the table is in 3NF

2.17 CODING_BET_TASK

Relational Model:
CODING_BET_TASK(task_id, example, solution, hint)

Foreign Keys:
task_id references TASK.task_id

Candidate Keys:
{task_id, example, solution}

Table Definition:
```
CREATE TABLE CODING_BET_TASK(
        task_id INT PRIMARY KEY,
        example VARCHAR(500) NOT NULL UNIQUE,
        solution VARCHAR(500) NOT NULL UNIQUE,
        hint VARCHAR(500),
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
task_id -> example, solution, hint
example -> task_id, solution, hint
solution -> example, task_id, hint
task_id, example and solution are super keys so the table is in 3NF

2.18 SUBMISSION

Relational Model:
SUBMISSION(<u>submission_id</u>, user_id, task_id, answer)

Foreign Keys:
user_id references USER.user_id
task_id references TASK.task_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE SUBMISSION(
        submission_id INT PRIMARY KEY,
        user_id INT NOT NULL
        task_id INT NOT NULL
        FOREIGN KEY (user_id) REFERENCES USER
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id -> user_id, task_id, answer
submission_id a super key so the table is in 3NF

2.19 NON_CODING_SUBMISSION

Relational Model:
NON_CODING_SUBMISSION(<u>submission_id</u>)

Foreign Keys:
submission_id references SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE NON_CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        FOREIGN KEY (submission_id) REFERENCES SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```
Normal Form:
submission_id a super key so the table is in 3NF

2.20 CODING_SUBMISSION

Relational Model:
CODING_SUBMISSION(submission_id, prog_lang, status)

Foreign Keys:
submission_id references SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        prog_lang VARCHAR(20) NOT NULL,
        status VARCHAR(20) NOT NULL,
        FOREIGN KEY (submission_id) REFERENCES SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id -> prog_lang, status
submission_id a super key so the table is in 3NF

2.21 PUBLIC_NON_CODING_SUBMISSION

Relational Model:
PUBLIC_NON_CODING_SUBMISSION(<u>submission_id</u>, date)

Foreign Keys:
submission_id references NON_CODING_SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE PUBLIC_NON_CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        date TIMESTAMP NOT NULL,
        FOREIGN KEY (submission_id) REFERENCES NON_CODING_SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id -> date
submission_id a super key so the table is in 3NF

2.22 INTERVIEW_NON_CODING_SUBMISSION

Relational Model:
INTERVIEW_NON_CODING_SUBMISSION(<u>submission_id</u>)

Foreign Keys:
submission_id references NON_CODING_SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE INTERVIEW_NON_CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        FOREIGN KEY (submission_id) REFERENCES NON_CODING_SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id a super key so the table is in 3NF

2.23 INTERVIEW_CODING_SUBMISSION

Relational Model:
INTERVIEW_CODING_SUBMISSION(<u>submission_id</u>)

Foreign Keys:
submission_id references INTERVIEW_CODING_SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        FOREIGN KEY (submission_id) REFERENCES CODING_SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id a super key so the table is in 3NF

2.24 PUBLIC_CODING_SUBMISSION

Relational Model:
PUBLIC_CODING_SUBMISSION(<u>submission_id</u>, date)

Foreign Keys:
submission_id references CODING_SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE PUBLIC_CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        date TIMESTAMP NOT NULL,
        FOREIGN KEY (submission_id) REFERENCES CODING_SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id -> date
submission_id a super key so the table is in 3NF

2.25 CONTEST_CODING_SUBMISSION

Relational Model:
CONTEST_CODING_SUBMISSION(<u>submission_id</u>)

Foreign Keys:
submission_id references CODING_SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE CONTEST_CODING_SUBMISSION(
        submission_id INT PRIMARY KEY,
        FOREIGN KEY (submission_id) REFERENCES CODING_SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id a super key so the table is in 3NF

2.26 CODING_BET_SUBMISSION

Relational Model:
CODING_BET_SUBMISSION(<u>submission_id</u>)

Foreign Keys:
submission_id references CODING_SUBMISSION.submission_id

Candidate Keys:
{submission_id}

Table Definition:
```
CREATE TABLE CODING_BET_SUBMISSION(
        submission_id INT PRIMARY KEY,
        FOREIGN KEY (submission_id) REFERENCES CODING_SUBMISSION
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Normal Form:
submission_id a super key so the table is in 3NF

2.27 COMPANY_INTERVIEW_TASK_PREPARE

Relational Model:
COMPANY_INTERVIEW_TASK_PREPARE(<u>company_id, task_id</u>, date)

Foreign Keys:
company_id references COMPANY.company_id
task_id references INTERVIEW_TASK.task_id

Candidate Keys:
{(company_id, task_id)}

Table Definition:
```
CREATE TABLE COMPANY_INTERVIEW_TASK_PREPARE(
        company_id INT NOT NULL,
        task_id INT NOT NULL,
        date TIMESTAMP NOT NULL,
        FOREIGN KEY (company_id) REFERENCES COMPANY
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (task_id) REFERENCES INTERVIEW_TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((company_id, task_id))
);
```

Normal Form:
Company_id, task_id -> date
task_id and company_id are super keys so the table is in 3NF

2.28 EDITOR_PUBLIC_TASK_PREPARE

Relational Model:
EDITOR_PUBLIC_TASK_PREPARE(editor_id, task_id, date)

Foreign Keys:
editor_id references EDITOR.editor_id
task_id references PUBLIC_TASK.task_id

Candidate Keys:
{(editor_id, task_id)}

Table Definition:
```
CREATE TABLE EDITOR_PUBLIC_TASK_PREPARE(
        editor_id INT NOT NULL,
        task_id INT NOT NULL,
        date TIMESTAMP NOT NULL,
        FOREIGN KEY (editor_id) REFERENCES EDITOR
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (task_id) REFERENCES PUBLIC_TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((editor_id, task_id))
);
```

Normal Form:
company_id, task_id -> date
task_id and company_id are super keys so the table is in 3NF

2.29 EDITOR_CODING_CONTEST_PREPARE

Relational Model:
EDITOR_CODING_CONTEST_PREPARE(<u>editor_id, contest_id</u>, date)

Foreign Keys:
editor_id references EDITOR.editor_id
contest_id references CODING_CONTEST.contest_id

Candidate Keys:
{(editor_id, contest_id)}

Table Definition:
```
CREATE TABLE EDITOR_CODING_CONTEST_PREPARE(
        editor_id INT NOT NULL,
        contest_id INT NOT NULL,
        date TIMESTAMP NOT NULL,
        FOREIGN KEY (editor_id) REFERENCES EDITOR
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (task_id) REFERENCES CODING_CONTEST
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((editor_id, contest_id))
);
```

Normal Form:
Editor_id, contest_id -> date
editor_id and contest_id are super keys so the table is in 3NF

2.30 COMPANY_INTERVIEW_PREPARE

Relational Model:
COMPANY_INTERVIEW_RELATION(<u>company_id, interview_id</u>)

Foreign Keys:
company_id references COMPANY.company_id
interview_id references INTERVIEW.interview_id

Candidate Keys:
{(company_id, interview_id)}

Table Definition:
```
CREATE TABLE COMPANY_INTERVIEW_ PREPARE(
        company_id INT NOT NULL,
        interview_id INT NOT NULL,
        FOREIGN KEY (company_id) REFERENCES COMPANY
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (interview_id) REFERENCES INTERVIEW
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((company_id, interview_id))
);
```

Normal Form:
interview_id and contest_id are super keys so the table is in 3NF

2.31 USER_CONTEST_PARTICIPATE

Relational Model:
USER_CONTEST_PARTICIPATE(<u>user_id, contest_id</u>, place)

Foreign Keys:
user_id references USER.user_id
contest_id references CODING_CONTEST.contest_id

Candidate Keys:
{(user_id, contest_id)}

Table Definition:
```
CREATE TABLE USER_CONTEST_PARTICIPATE(
        user_id INT NOT NULL,
        contest_id INT NOT NULL,
        place INT,
        FOREIGN KEY (user_id) REFERENCES USER
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (contest_id) REFERENCES CODING_CONTEST
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((user_id, contest_id))
);
```

Normal Form:
user_id, contest_id -> place
user_id and contest_id are super keys so the table is in 3NF

2.32 USER_CODING_BET_PARTICIPATE

Relational Model:
USER_CONTEST_PARTICIPATE(<u>user_id, coding_bet_id</u>, place)

Foreign Keys:
user_id references USER.user_id
coding_bet_id references CODING_BET.coding_bet_id

Candidate Keys:
{(user_id, coding_bet_id)}

Table Definition:
```
CREATE TABLE USER_CODING_BET_PARTICIPATE(
        user_id INT NOT NULL,
        coding_bet_id INT NOT NULL,
        place INT,
        FOREIGN KEY (user_id) REFERENCES USER
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (coding_bet_id) REFERENCES CODING_BET
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((user_id, coding_bet_id))
);
```

Normal Form:
user_id, coding_bet_id -> place
user_id and coding_bet_id are super keys so the table is in 3NF

2.33 USER_INTERVIEW_PERFORM

Relational Model:
USER_CONTEST_PARTICIPATE(<u>user_id, interview_id</u>, date, result)

Foreign Keys:
user_id references USER.user_id
interview_id references INTERVIEW.interview_id

Candidate Keys:
{(user_id, interview_id)}

Table Definition:
```
CREATE TABLE USER_INTERVIEW_PERFORM(
        user_id INT NOT NULL,
        interview_id INT NOT NULL,
        date TIMESTAMP NOT NULL,
        result VARCHAR(20),
        FOREIGN KEY (user_id) REFERENCES USER
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (interview_id) REFERENCES INTERVIEW
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((user_id, interview_id))
);
```

Normal Form:
user_id, contest_id -> date result
user_id and contest_id are super keys so the table is in 3NF

2.34 TEST_CASE

Relational Model:
TEST_CASE(task_id, sample_input, sample_output)

Foreign Keys:
task_id references TASK.task_id

Candidate Keys:
{(task_id, sample_input, sample_output)}

Table Definition:
```
CREATE TABLE TEST_CASE(
        task_id INT NOT NULL,
        sample_input VARCHAR(1000) NOT NULL,
        sample_ioutput VARCHAR(1000) NOT NULL,
        FOREIGN KEY (task_id) REFERENCES TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((task_id, sample_input, sample_output))
);
```

Normal Form:
task_id, sample_input and sample_output are super keys so the table is in 3NF

2.35 CODING_BET_TASK_RELATION

Relational Model:
CODING_BET_TASK_RELATION(coding_bet_id, task_id)

Foreign Keys:
Coding_bet_id references CODING_BET.coding_bet_id
task_id references CODING_BET_TASK.task_id

Candidate Keys:
{(coding_bet_id, task_id)}

Table Definition:
```
CREATE TABLECODING_BET_TASK_RELATION(
        coding_bet_id INT NOT NULL,
        task_id INT NOT NULL,
        FOREIGN KEY (coding_bet_id) REFERENCES CODING_BET
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (task_id) REFERENCES CODING_BET_TASK
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((coding_bet_id, task_id))
);
```

Normal Form:
Coding_bet_id and task_id are super keys so the table is in 3NF

2.36 RATE_ANSWER

Relational Model:
COMPANY_INTERVIEW_TASK_PREPARE(user_id, answer_id, rate_type)

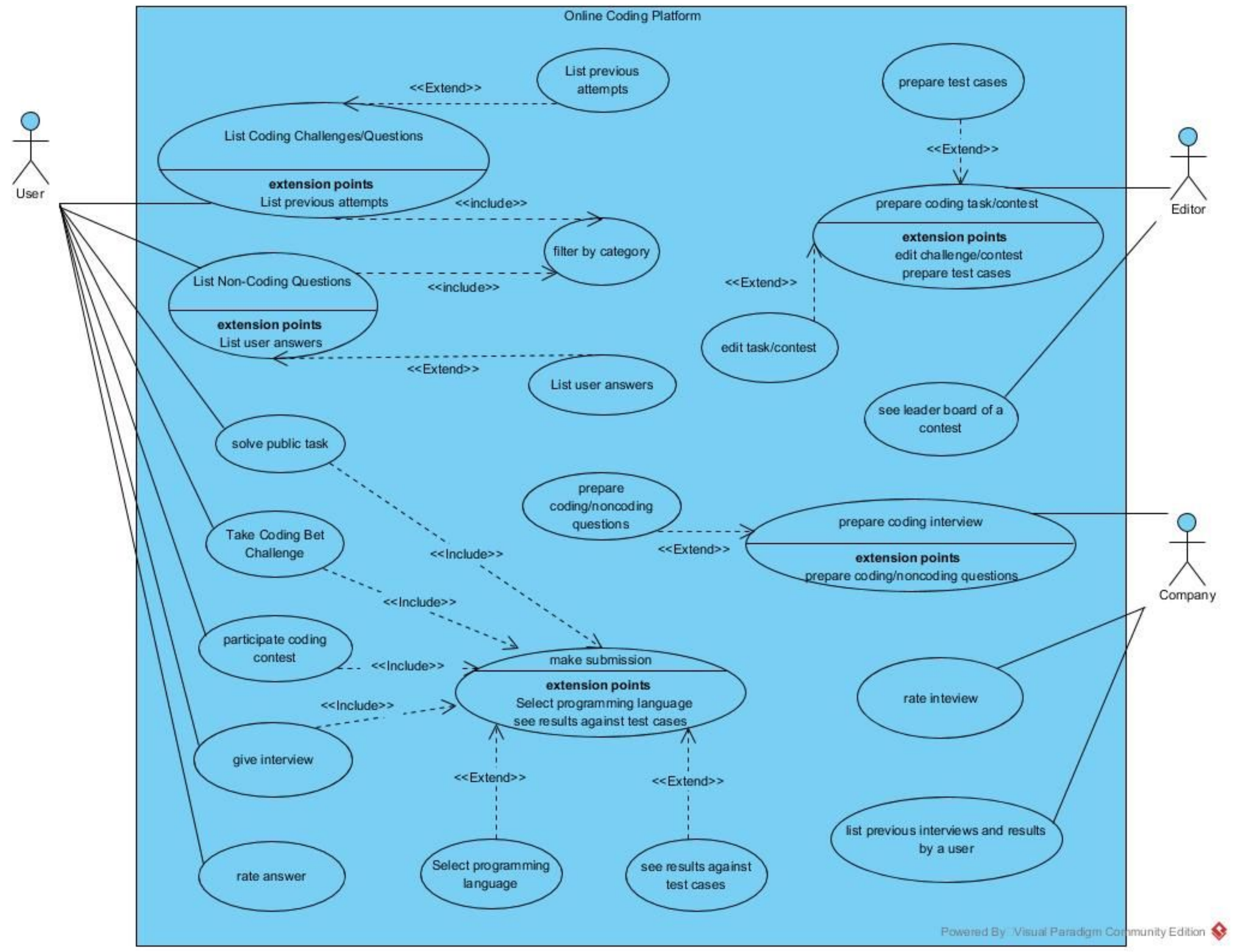
Foreign Keys:
user_id references USER.user_id
answer_id references PUBLIC_NON_CODING_SUBMISSION.submission_id

Candidate Keys:
{(user_id, answer_id)}

Table Definition:
```
CREATE DOMAIN vote VARCHAR(9)
CONSTRAINT vote
CHECK ( VALUE IN ( ‘Downvote’, ‘Upvote’));

CREATE TABLE RATE_ANSWER(
        user_id INT NOT NULL,
        answer_id INT NOT NULL
        FOREIGN KEY (user_id) REFERENCES USER
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (answer_id)
        REFERENCES PUBLIC_NON_CODING_SUBMISSION(submission_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        PRIMARY_KEY((user_id, answer_id))

);
```
Normal Form:
user_id, answer_id -> rate_type
user_id and answer_id are super keys so the table is in 3NF

3. Functional Components

● Use Case Diagram



- ## Main Use Case Scenarios

Descriptions of Make Submission Use Case

Name: Make Submission

Participating actor: User

Entry condition:
- The user has a valid account and currently logged in

Exit condition:
- User successfully makes a submission to a non-coding / coding question

Event Flow:
- The user navigates one of the following pages: Contest Page, Problem Page and Interview Page
- Available coding challenges, contests, and questions are listed from all categories
- The user may select a desired category
- Results are filtered according to the user's selection
- The user selects a task and navigates to the problem page
- The question, test cases(for coding questions) and examples are displayed
- The user applies his solution and presses the submit button

Special Requirements
- The user can select the desired Programming Language if the task is a coding challenge

Descriptions of Take Coding Bet Challenge Use Case

Name: Take Coding Bet Challenge

Participating actor: 2 Users

Entry condition:

- Users have a valid account and currently logged in

Exit condition:

- Both users successfully participate in a coding bet challenge
- One of the users won the bet

Event Flow:

- Users navigate to the coding bet screen
- They choose to send or accept an invitation received from another user
- The users are able to change the current value of the bet
- The challenge starts and both users try to solve the task
- The first user to solve the task wins the challenge and takes all of the coins

Special Requirements

- If a user withdraws from the contest, he automatically loses the contest

Descriptions of Check Test Cases Use Case

Name: Check Test Cases

Participating actor: User

Entry condition:
- The user has a valid account and currently logged in
- The user has successfully started a task

Exit condition:
- The user sees the result of different test cases against his code

Event Flow:
- The user presses the run button
- If there are no compile errors the code runs against the test cases
- Different results for different test cases are displayed
- The user is notified which test cases are correct or false

Special Requirements
- The user is able to use their own test cases

Descriptions of Rate an Answer Use Case

Name: Rate an Answer

Participating actor: User

Entry condition:
- The user has a valid account and currently logged in
- The user is viewing a non-coding question

Exit condition:
- User rates an answer given to a specific question

Event Flow:
- The User displays a non-coding question, and below the question, the answers given by other users are displayed.
- The user chooses to downvote or upvote the question and presses the respective button
- The updated rate for the answer is displayed

Descriptions of Prepare Coding Challenge/Contest Use Case

Name: Prepare Coding Challenge/Contest

Participating actor: Editor

Entry condition:
- The editor has a valid account and currently logged in

Exit condition:
- Editor successfully prepares a coding challenge or a contest

Event Flow:
- Editor navigates to create a contest page
- Editor navigates to create a task page
- Editor specifies a problem, its difficulty, and its category
- Editor prepares test cases for a coding challenge and specifies the duration of the contest
- Editor presses publish button

Special Requirements:
- The editor is able to prepare a series of coding questions and publish it as a coding challenge

Descriptions of See Leaderboard of a Contest Use Case

Name: See Leaderboard of a Contest

Participating actor: Editor

Entry condition:
- The editor is currently logged in

Exit condition:
- Editor successfully views the current leader board

Event Flow:
- Editor navigates to the coding contests page
- Editor chooses a specific coding contest
- In the coding contest page, the current leaderboard is displayed at the right

Descriptions of Give Interview Use Case

Name: Give Interview

Participating actor: User

Entry condition:
- User has a valid account and currently logged in

Exit condition:
- The user successfully gives an interview

Event Flow:
- The user navigates to the interview page
- The available interviews with specific companies are displayed
- The user starts the interview by pressing the start button
- The user gives an interview
- The user answers questions one by one and completes the interview
- The user submits his interview by using the submit button which will automatically inform the company.
- The user is directed to the interview result page where the current status of the interview is displayed

Special Requirements:
- The user is able to check his interview result any time from the interview page

Descriptions of Prepare Interview Use Case

Name: Prepare Interview

Participating actor: Company

Entry condition:
- Company is currently logged in

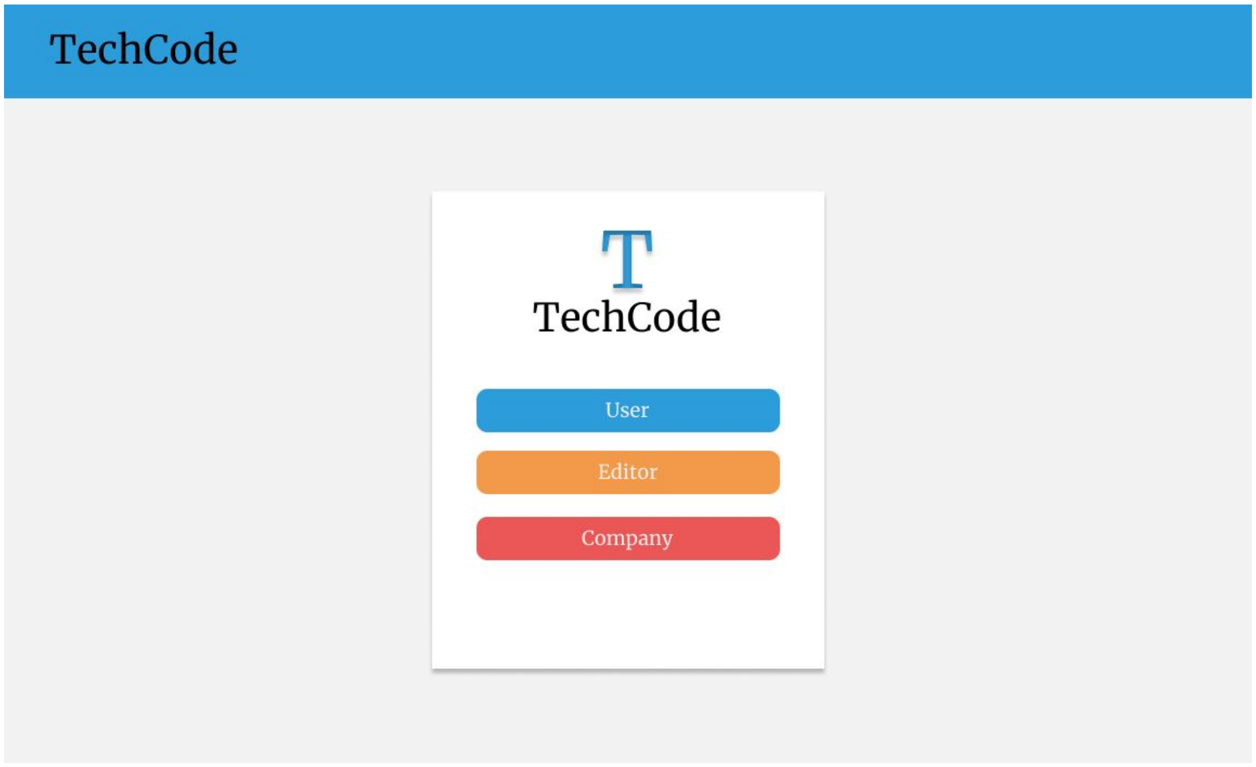
Exit condition:
- The company successfully creates a coding interview

Event Flow:
- Company navigates to prepare coding interview page
- Company specifies a problem, its difficulty, time limit and category
- Company presses the create interview button

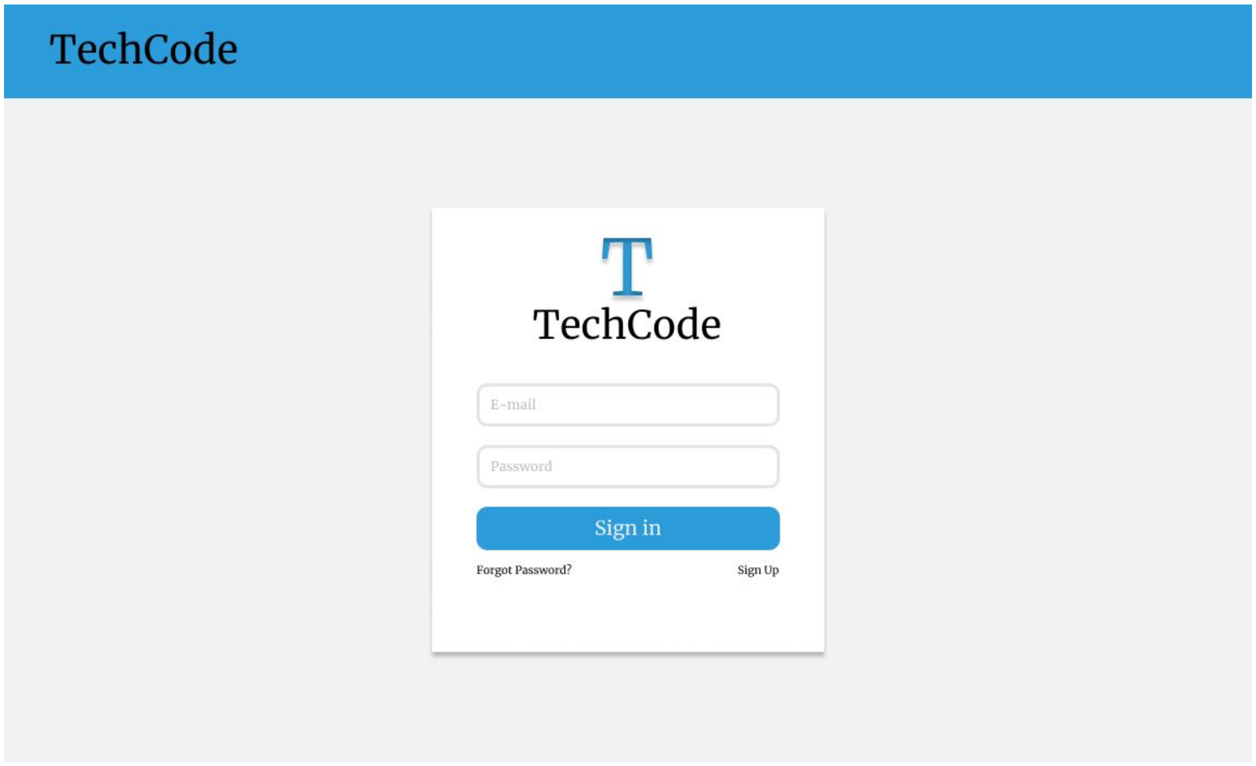
Special Requirements:
- The company is able to list previous interviews and results by a user
- The company editor decides on the result of an interview and notifies the user

4. User interface design and corresponding SQL statements

Sign in Page 1



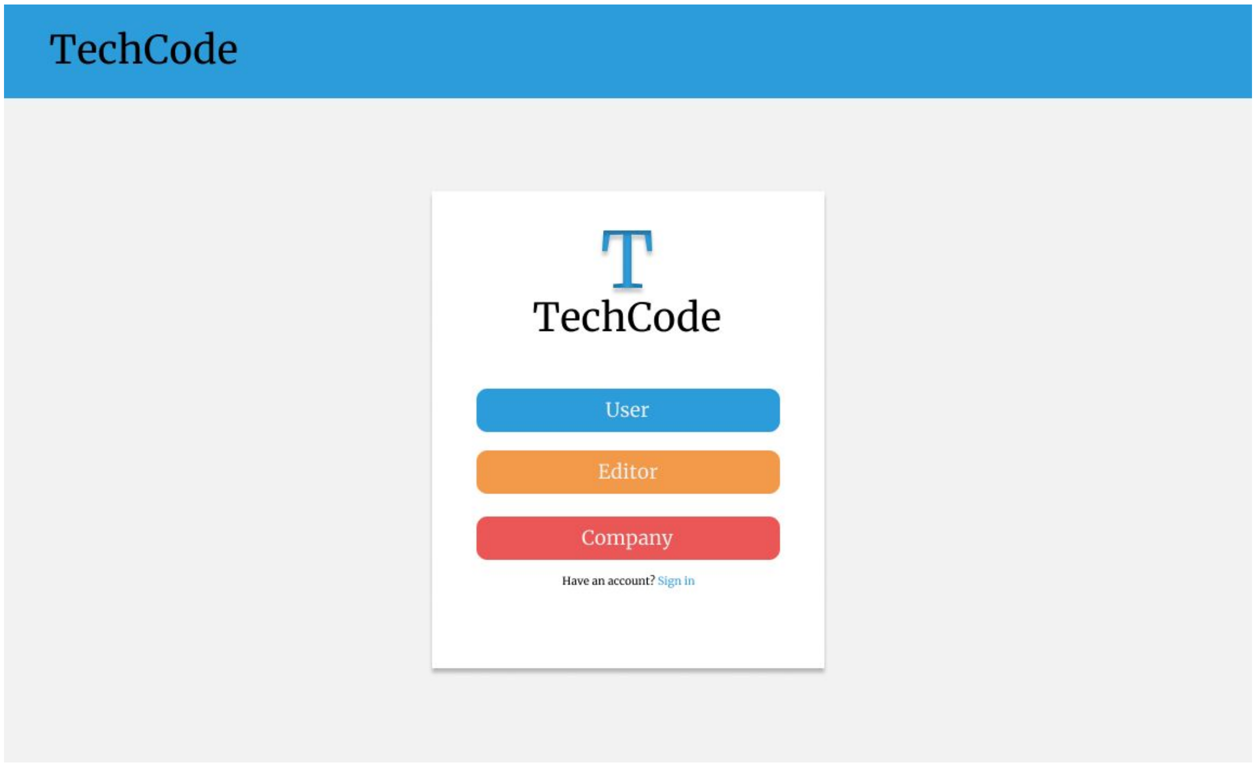
Sign in Page 2



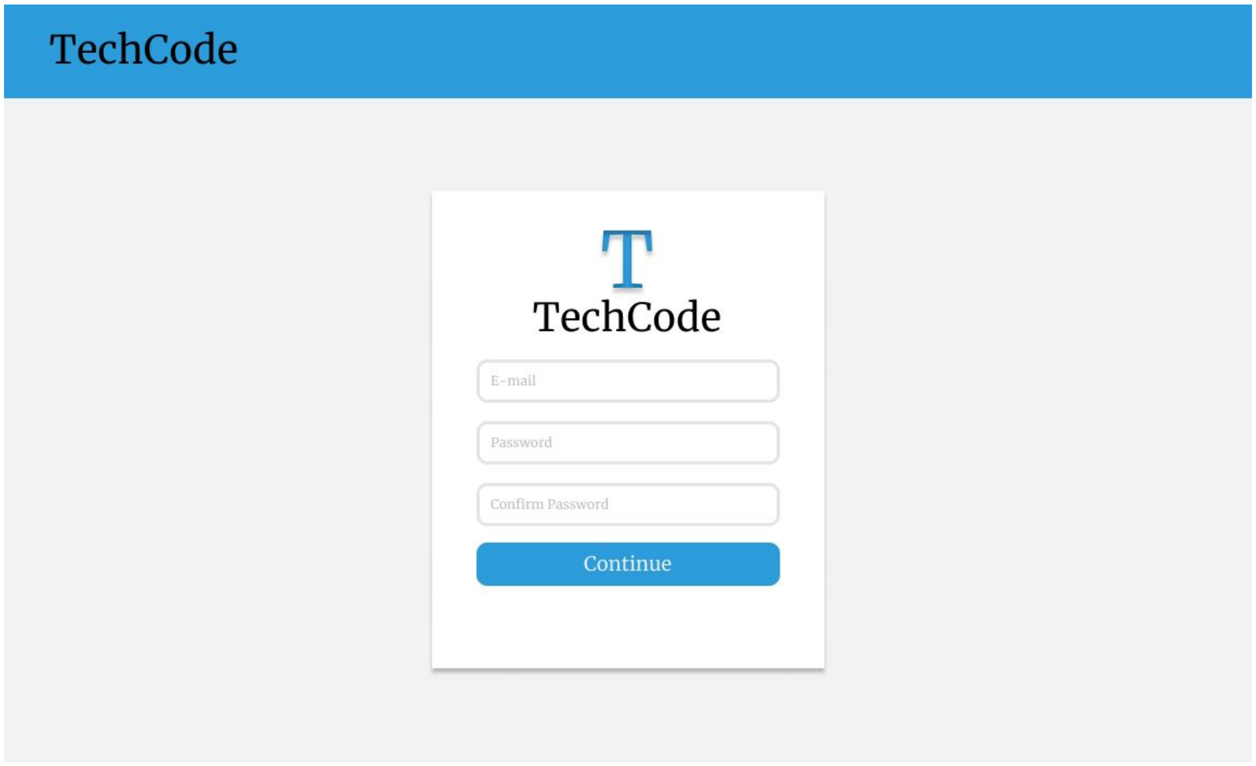
When the user fills the forms and clicks Sign In to log the user in

The getUserInformation Procedure will be used on this page.

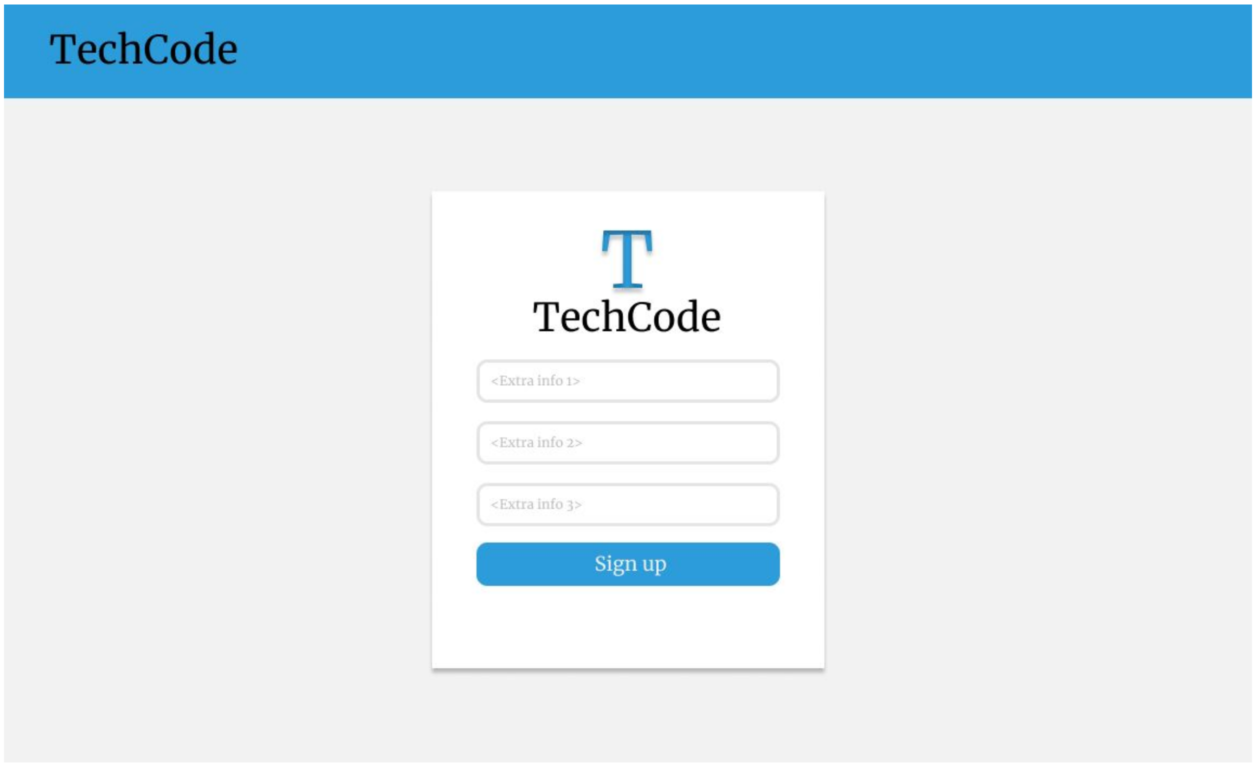
Sign up Page 1



Sign up Page 2



Sign up Page 3



When the user has filled all the previous forms and clicks Sing up to sign the user up

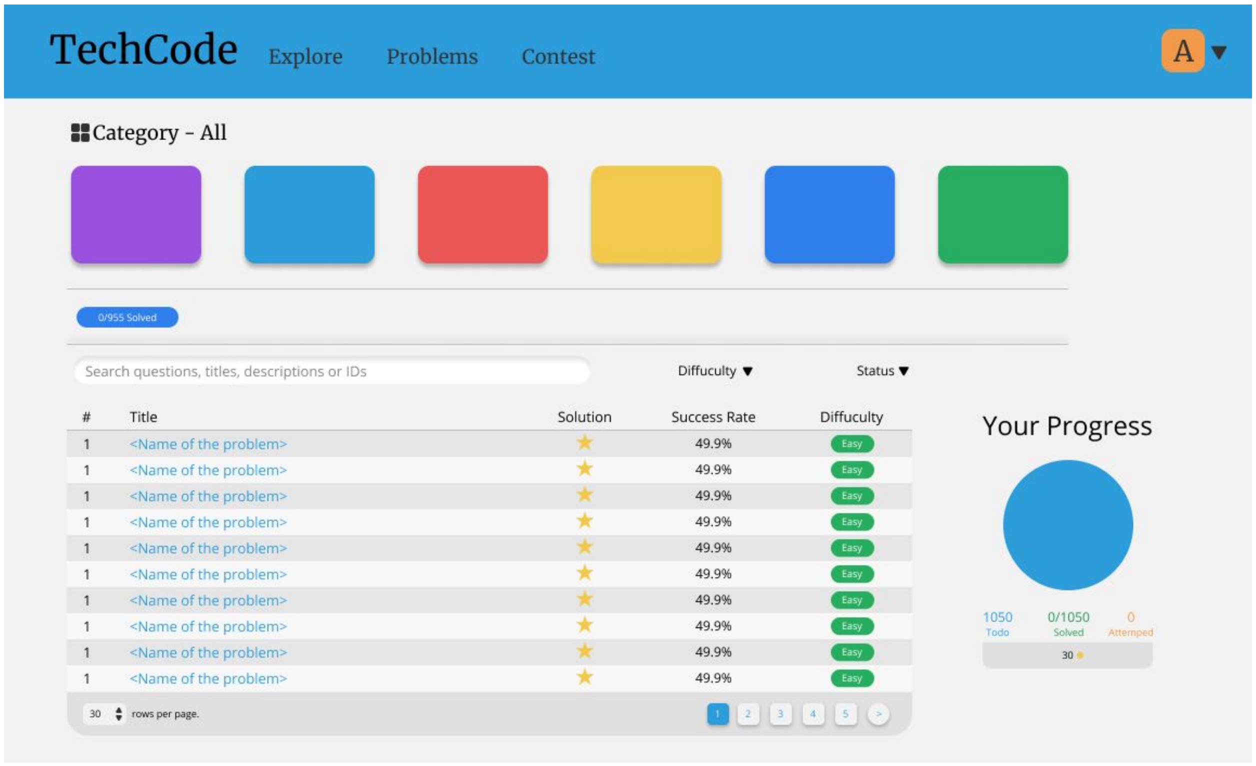
INSERT INTO account
VALUES(<id>, <name>, <email>, <password>, <info>);

INSERT INTO user
VALUES(<id>, 0, 0);

INSERT INTO company
VALUES(id, “ ”);

INSERT INTO editor
VALUES(id, "noob");

Problems Page



#	Title	Solution	Success Rate	Diffuculty
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy
1	<Name of the problem>	★	49.9%	Easy

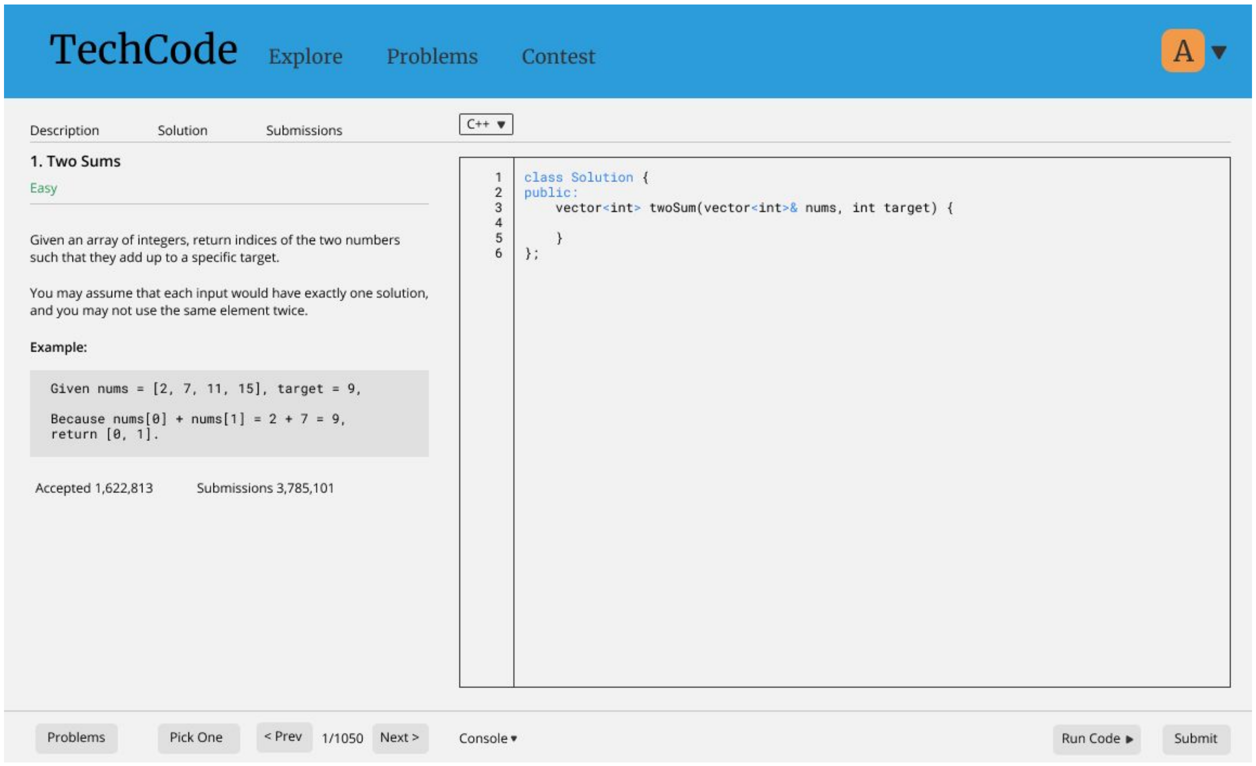
When the user selects a category to filter the questions according to the category

The selectAllFromCategory Procedure will be used with the respective category input

When the user makes a search using the search bar to filter the questions according to the search input

The selectAllFromCategoryWithSearchInput Procedure will be used with the respective category input and the search input

Problem Page



TechCode Explore Problems Contest A

Description Solution Submissions

C++

1. Two Sums

Easy

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Example:

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

Accepted 1,622,813 Submissions 3,785,101

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

    }
};
```

Problems Pick One < Prev 1/1050 Next > Console Run Code Submit

When the user selects a problem to display information about the question

SELECT *
FROM task NATURAL JOIN public_task NATURAL JOIN public_coding_task
WHERE id = <task_id>

When the user makes a submission

INSERT INTO public_coding_submission
VALUES(<submission_id>, <date>)

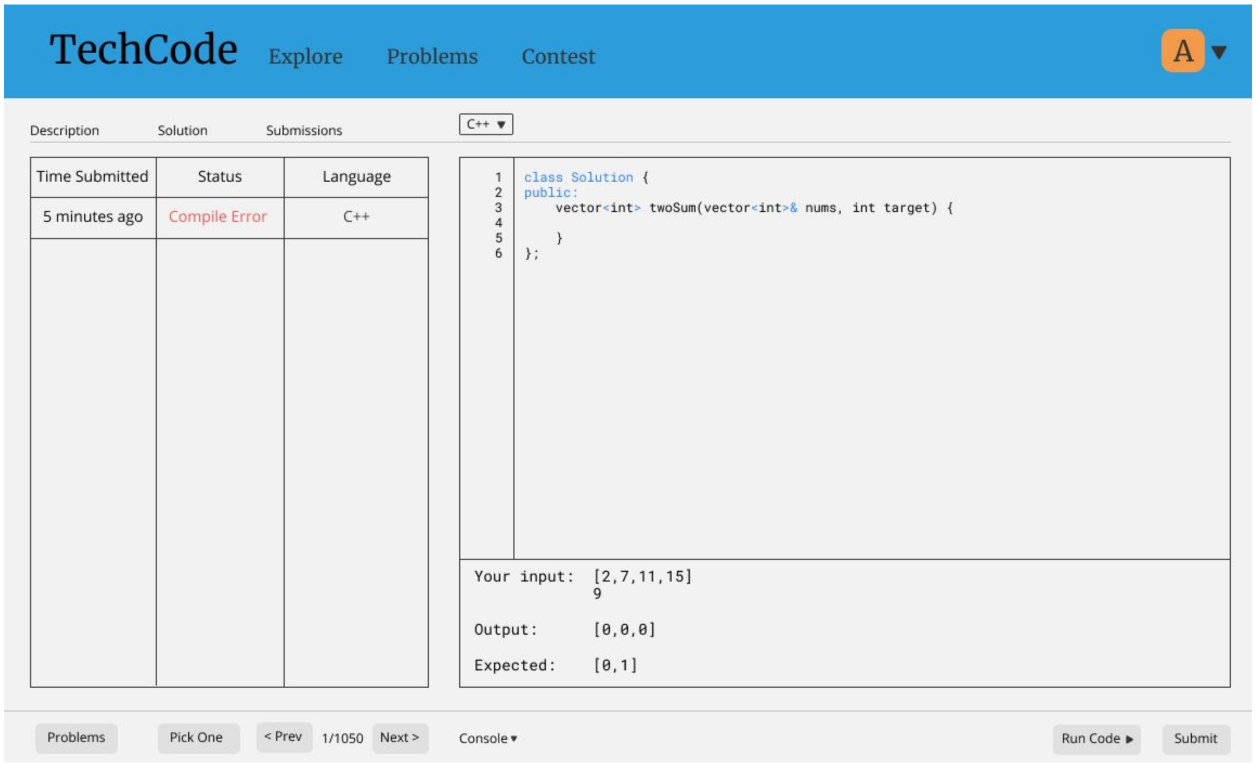
INSERT INTO coding_submission
VALUES(<submission_id>, <date>, <prog_lang>, <status>)

INSERT INTO submission
VALUES(<submission_id>,<user_id>, <task_id>, <answer>)

When the user makes a submission

SELECT *
FROM test_case
WHERE task_id = <task_id>

Submissions Page



Time Submitted	Status	Language
5 minutes ago	Compile Error	C++

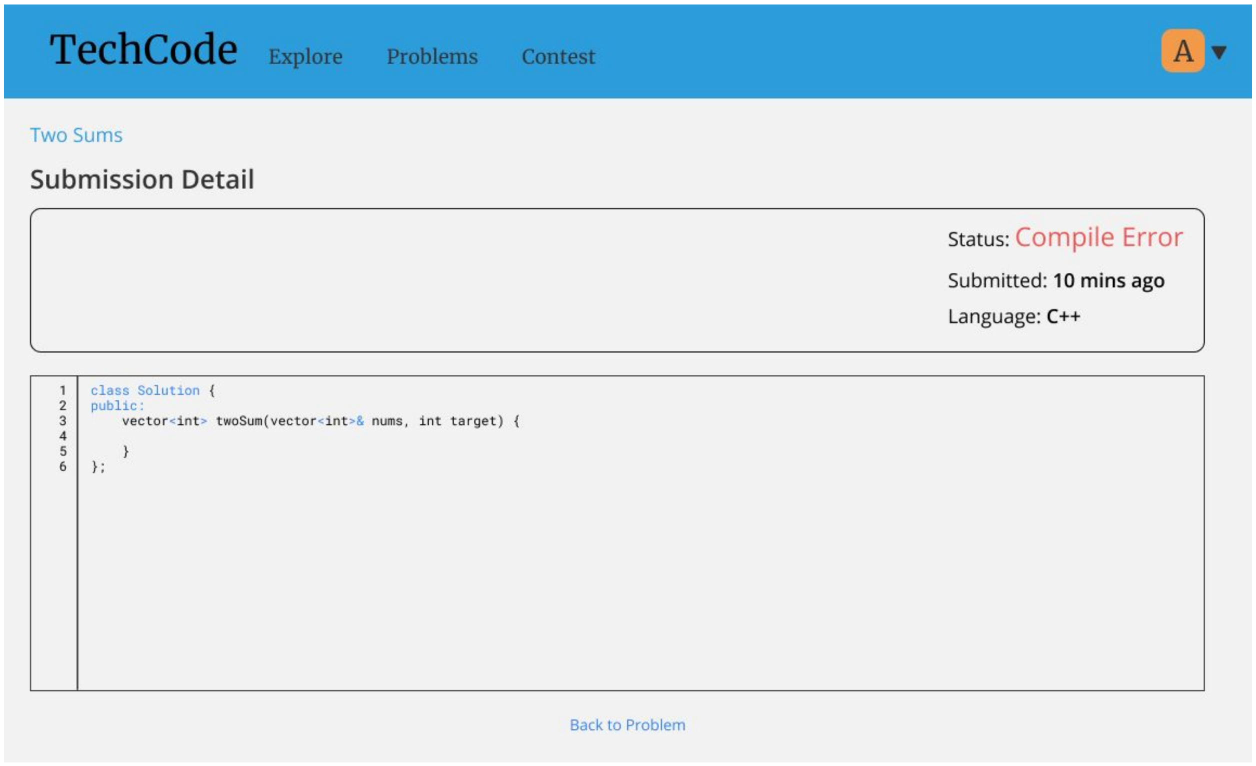
```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

    }
};
```

```
Your input:  [2,7,11,15]
             9

Output:      [0,0,0]

Expected:    [0,1]
```

When the user clicks on the submission tab to show the previous submissions

SELECT *
FROM user_submission
WHERE user_id = <user_id> and task_id = <task_id>
(user_submission is a view)

Submission Page



Status: Compile Error

Submitted: **10 mins ago**

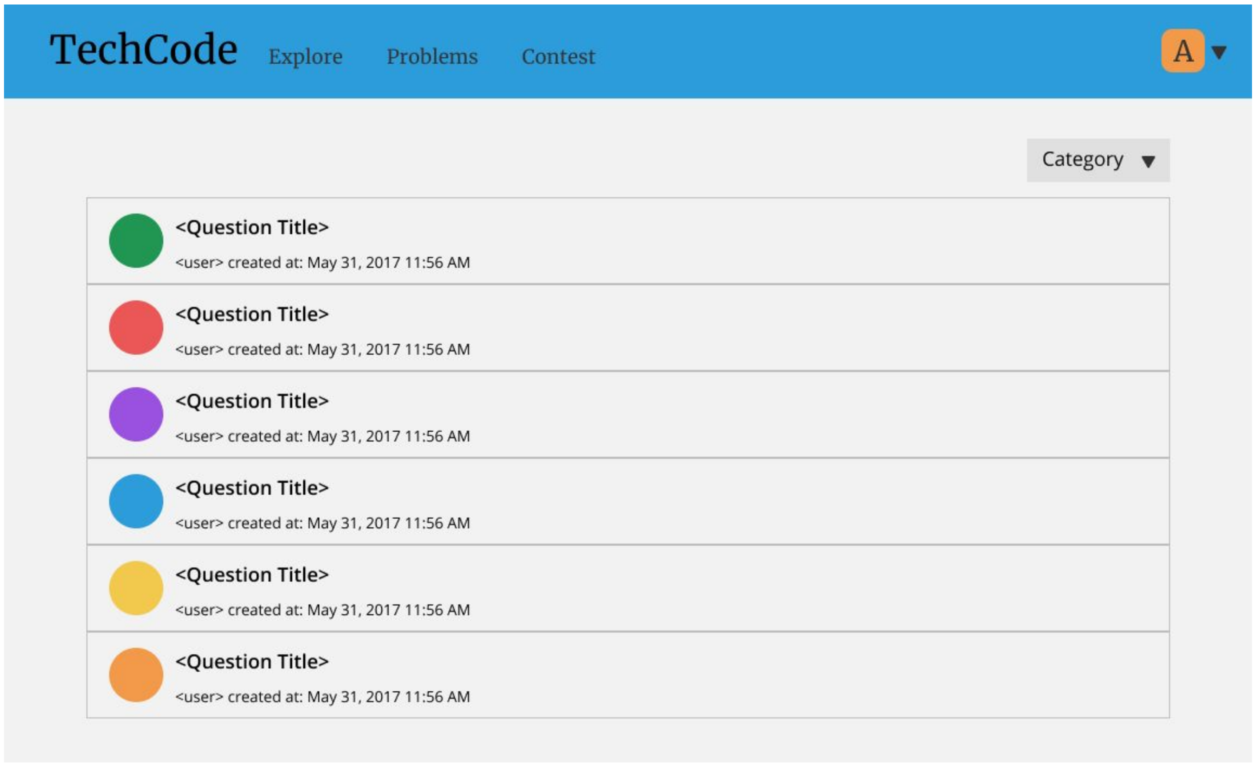
Language: **C++**

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

    }
};
```

When the user selects a specific submission to show the details of the submission

SELECT *
FROM user_submission_detail
WHERE user_id = <user_id> and task_id = <task_id> and submission_id = <submission_id>
(user_submission_detail is a view)

Non Coding Questions Page



<Question Title>
<user> created at: May 31, 2017 11:56 AM

<Question Title>
<user> created at: May 31, 2017 11:56 AM

<Question Title>
<user> created at: May 31, 2017 11:56 AM

<Question Title>
<user> created at: May 31, 2017 11:56 AM

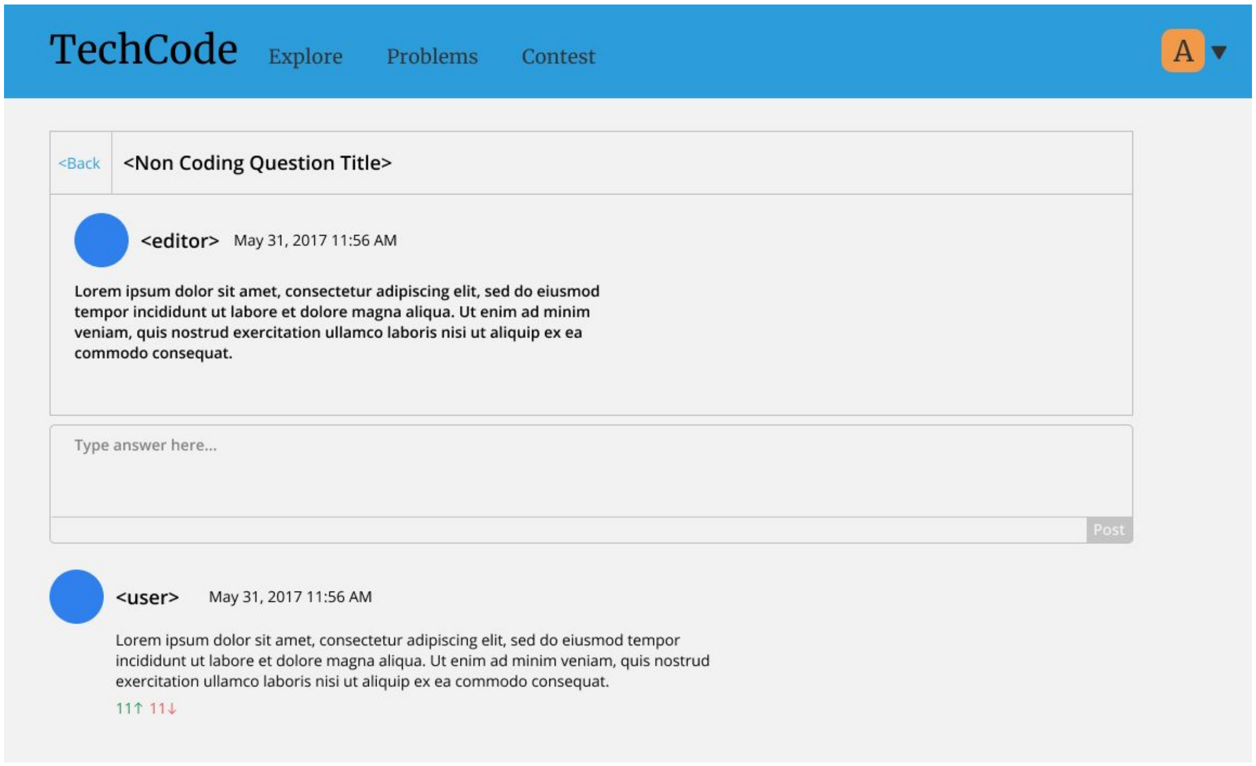
<Question Title>
<user> created at: May 31, 2017 11:56 AM

<Question Title>
<user> created at: May 31, 2017 11:56 AM

When the user selects the non-coding problems page to show the non coding problems

The selectAllNonCodingFromCategory will be used with the respective category

Non Coding Question Page



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

When the user selects a specific non-coding question to show the details of the question

```
SELECT *
FROM public_non_coding_task NATURAL JOIN public_task NATURAL JOIN task
WHERE task_id = <task_id>
```

To display the answers given to the non-coding question

```
SELECT DISTINCT *
FROM  public_non_coding_submission NATURAL JOIN submission
WHERE task_id = <task-id>
```

When the user gives an answer to the non-coding question

```
INSERT INTO  public_non_coding_submission
VALUES (<submission_id>, <date>)

INSERT INTO submission
VALUES(<submission_id>, <user_id>, <task_id>, <answer>)
```

When the user gives a rating to an answer

```
INSERT INTO  rate_answer
VALUES (<user_id>, <answer_id>, rate_type)
```

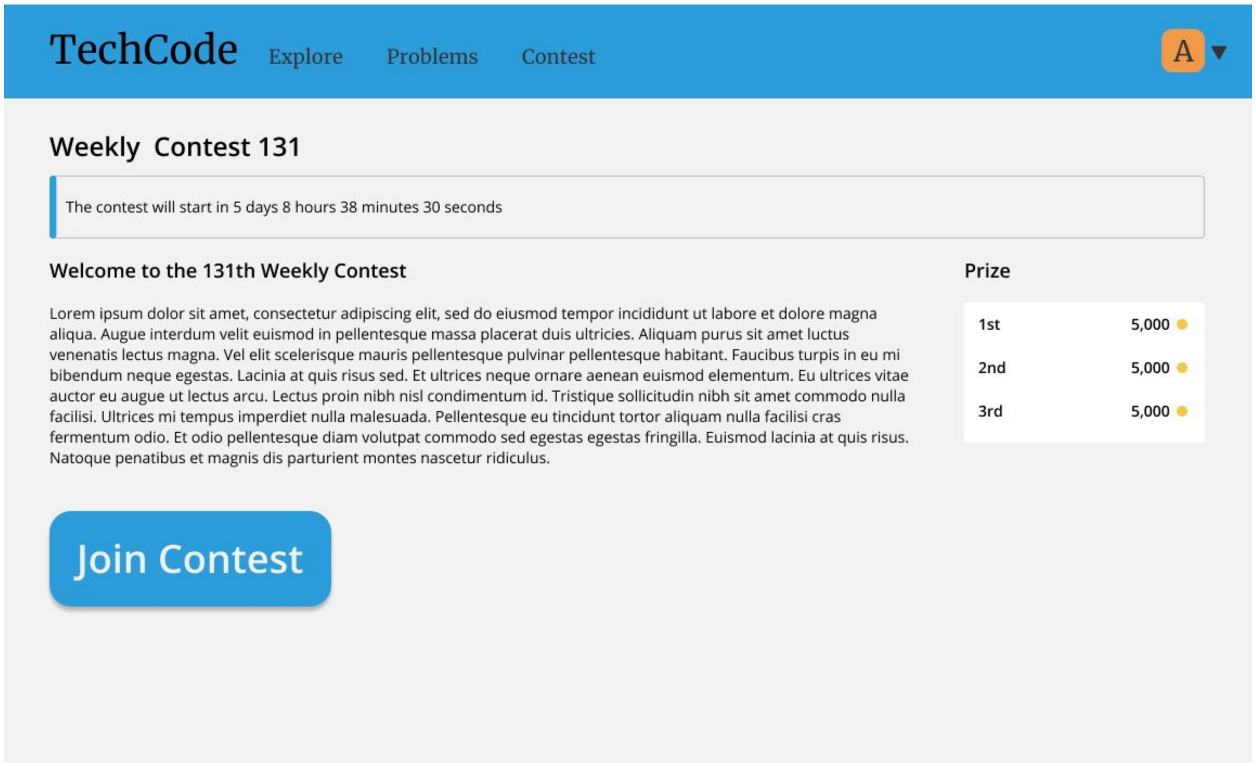
When the user removes a rating to an answer

```
DELETE FROM TABLE rate_answer
WHERE user_id = <user_id> and answer_id = <answer_id>
```

To display the current ratings of an answer

Stored reports will be used to count the number of upvotes and downvotes on an answer. The query can be seen in the advanced SQL part of the report

Contest Page



Weekly Contest 131

The contest will start in 5 days 8 hours 38 minutes 30 seconds

Welcome to the 131th Weekly Contest

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Augue interdum velit euismod in pellentesque massa placerat duis ultricies. Aliquam purus sit amet luctus venenatis lectus magna. Vel elit scelerisque mauris pellentesque pulvinar pellentesque habitant. Faucibus turpis in eu mi bibendum neque egestas. Lacinia at quis risus sed. Et ultrices neque ornare aenean euismod elementum. Eu ultrices vitae auctor eu augue ut lectus arcu. Lectus proin nibh nisl condimentum id. Tristique sollicitudin nibh sit amet commodo nulla facilisi. Ultrices mi tempus imperdiet nulla malesuada. Pellentesque eu tincidunt tortor aliquam nulla facilisi cras fermentum odio. Et odio pellentesque diam volutpat commodo sed egestas egestas fringilla. Euismod lacinia at quis risus. Natoque penatibus et magnis dis parturient montes nascetur ridiculus.

Prize

1st	5,000 ●
2nd	5,000 ●
3rd	5,000 ●

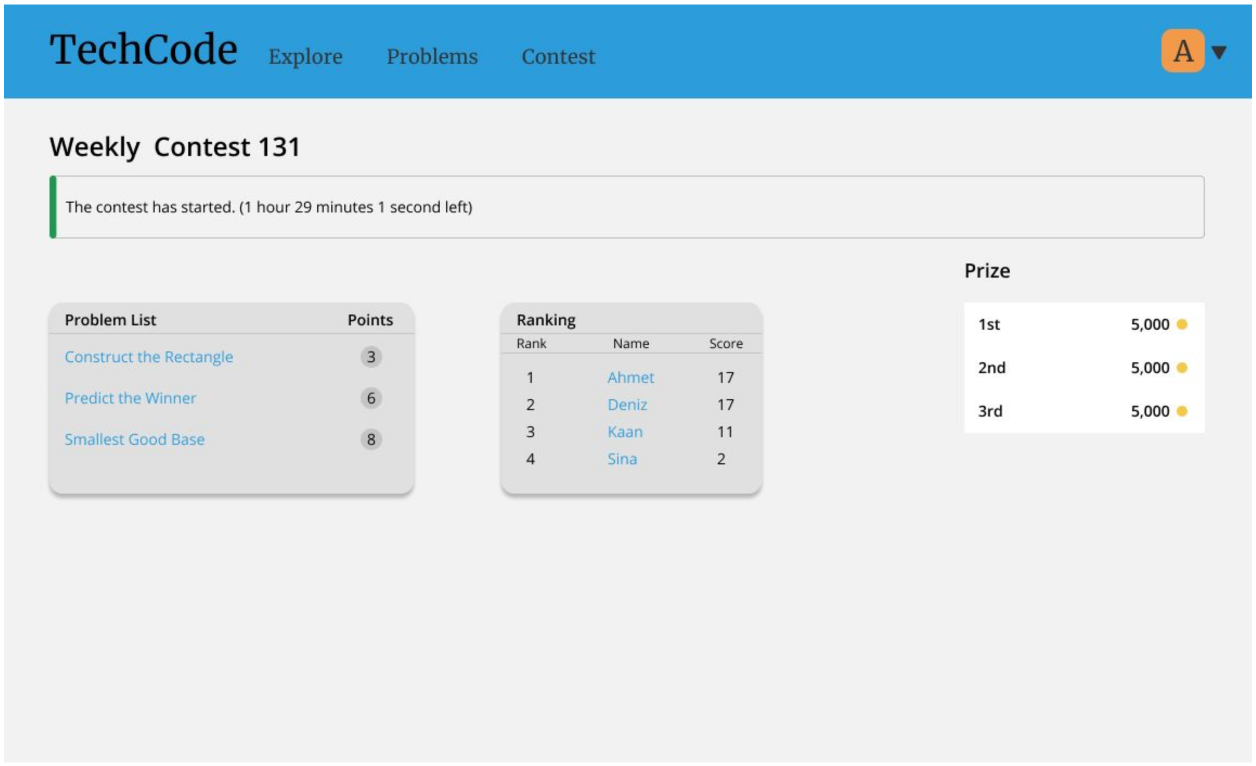
When the user selects the contest page to show the contest details

```
SELECT *
FROM contest
WHERE id = <contest-id>
```

When the user joins the contest

```
INSERT INTO user_contest_participate
VALUES (<user-id>,<contest-id>, 0)
```

Joined Contest Page



Weekly Contest 131

The contest has started. (1 hour 29 minutes 1 second left)

Problem List	Points
Construct the Rectangle	3
Predict the Winner	6
Smallest Good Base	8

Ranking

Rank	Name	Score
1	Ahmet	17
2	Deniz	17
3	Kaan	11
4	Sina	2

Prize

1st	5,000 ●
2nd	5,000 ●
3rd	5,000 ●

When the user selects the contest page after joining the contest to display the contest details

```
SELECT *
FROM contest
WHERE contest_id = <id>
```

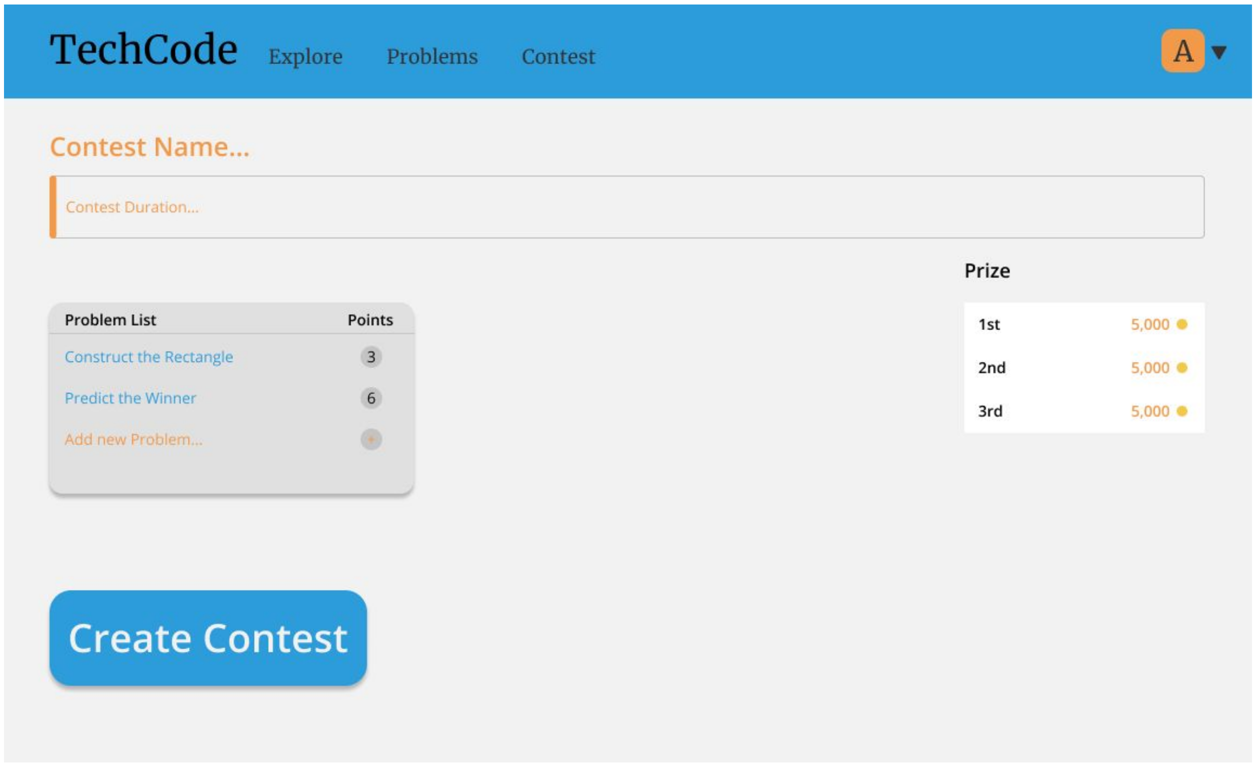
When the user selects the contest page after joining the contest to display the contest problems

```
SELECT *
FROM contest_task NATURAL JOIN task
WHERE  contest_id = <contest_id>
```

When the user selects the contest page after joining the contest to display the current ranking of the contest

The showRankingOfContest will be used with the respective contest_id

Create Contest Page



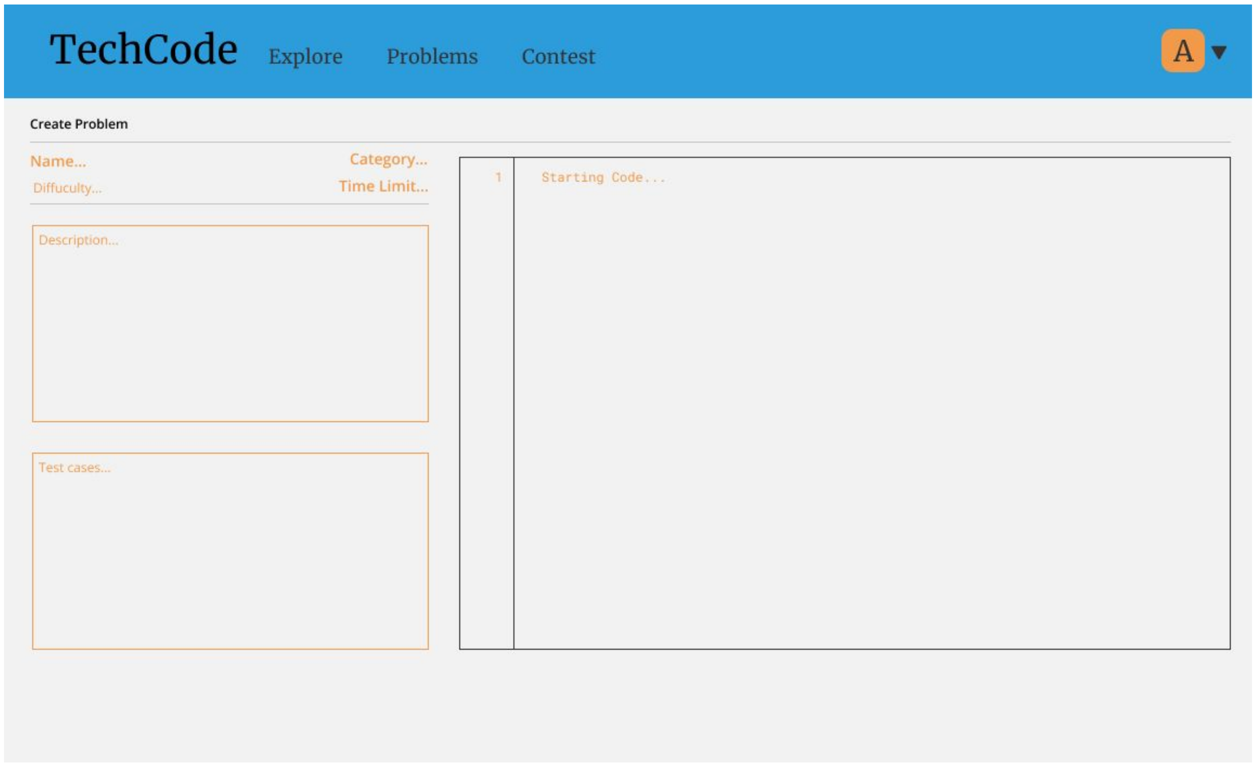
Problem List	Points
Construct the Rectangle	3
Predict the Winner	6
Add new Problem...	+

1st	5,000 ●
2nd	5,000 ●
3rd	5,000 ●

When the editor creates a contest

INSERT INTO coding_contest
VALUES
(<contest_id>, <name>, <start_date>, <end_date>, <reward1>, <reward2>, <reward3>)

Create Problem Page



When the editor creates a coding problem for a contest

INSERT INTO contest_task
VALUES (<task_id>, <contest_point>, <example>, <solution>, <hint>, <contest_id>)

INSERT INTO task
VALUES (<task_id>, <title>, <question>, <category>, <info>)

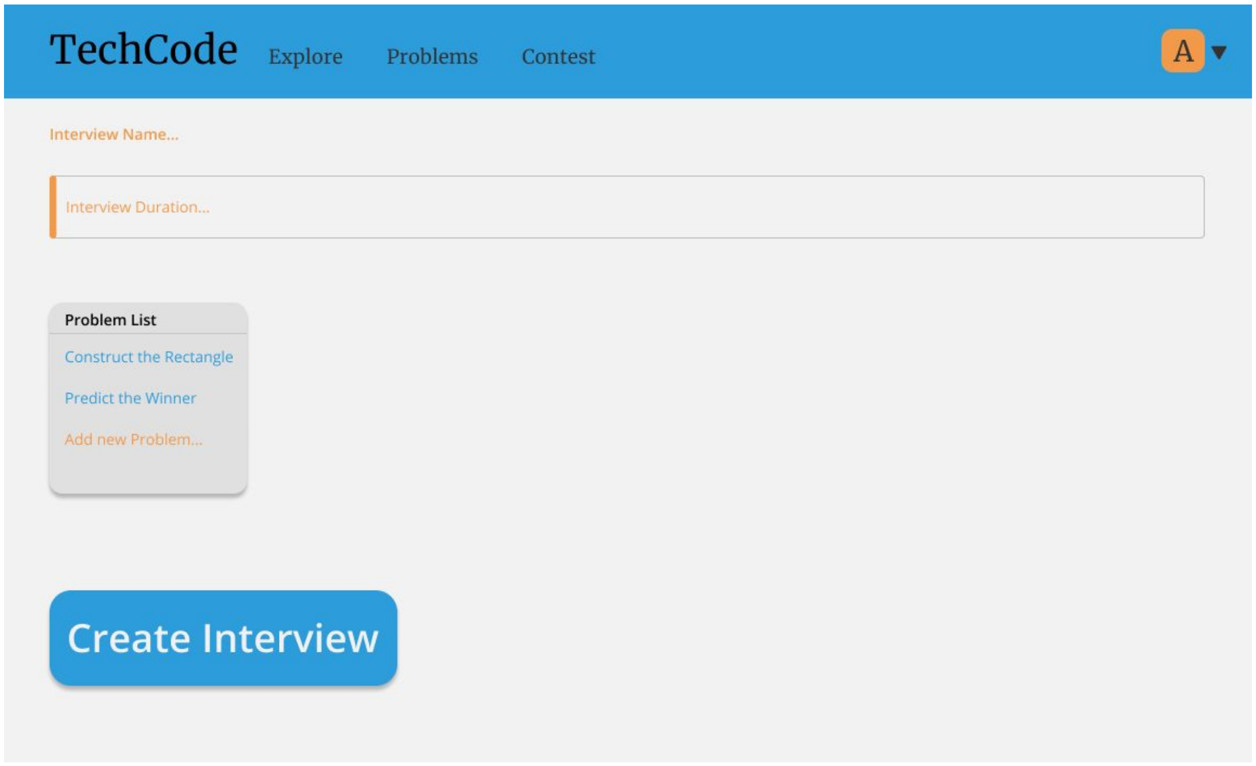
When the editor creates a coding problem for the public

INSERT INTO public_coding_task
VALUES (<task_id>, <example>, <solution>, <hint>)

INSERT INTO public_task
VALUES (<task_id>, <difficulty>, <points_given>, 0)

INSERT INTO task
VALUES (<task_id>, <title>, <question>, <category>, <info>)

Create Interview Page



When the company creates an interview

```
INSERT INTO interview
VALUES  (<interview_id>, <job-type>, <description>, <duration>)

INSERT INTO company_interview_task_prepare
VALUES (<company_id>, <interview_id>)
```

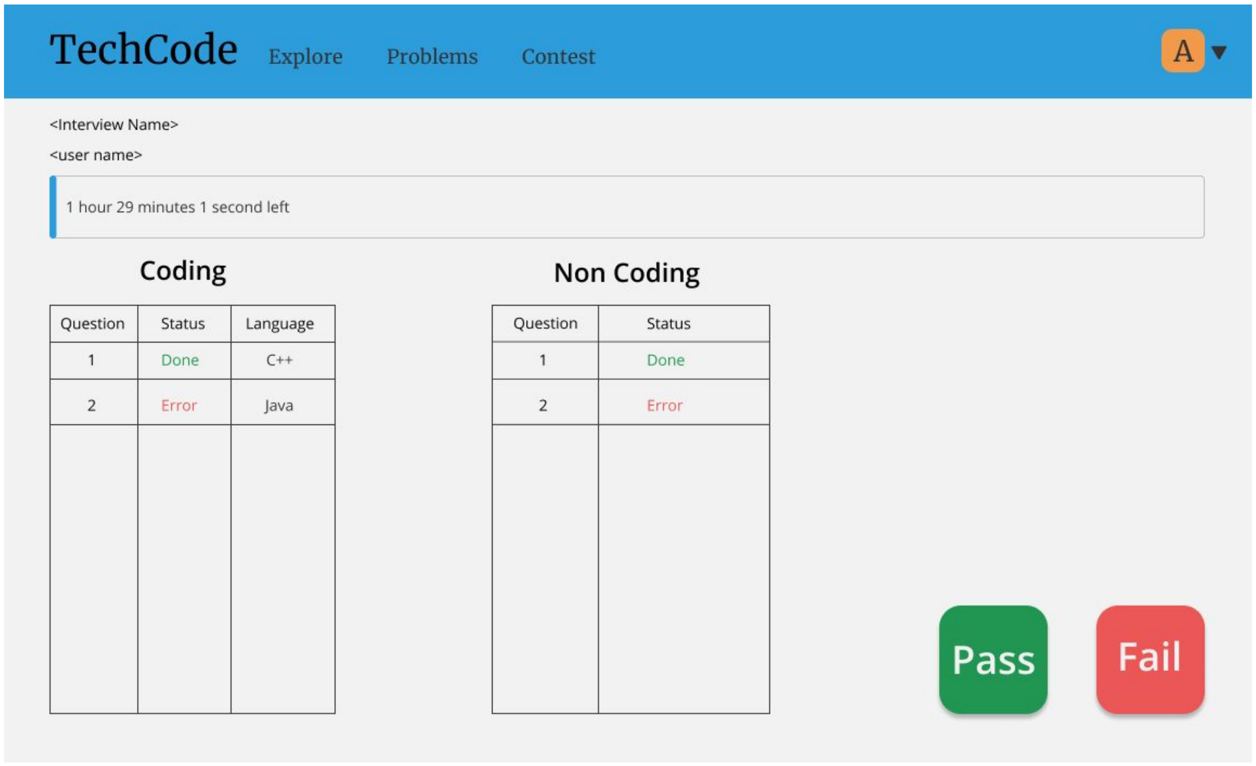
To display the question created for the interview

```
SELECT *
FROM interview_task NATURAL JOIN task
WHERE interview_id = <interview_id>
```

When the company creates a coding/non-coding question for the interview

```
INSERT INTO task
VALUES (<task_id>, <title>, <question>, <category>, <info>)

INSERT INTO interview_task
VALUES (<task_id>, <interview_id>)

INSERT INTO coding_interview_task
VALUES (id,<example>,<solution>,<hint>)

INSERT INTO non_coding_interview_task
VALUES (<task>)
INSERT INTO company_interview_task_prepare
VALUES (<company_id>, <task_id>, <date>)
```

Interview Result Page (Company)



<Interview Name>

<user name>

1 hour 29 minutes 1 second left

Coding

Question	Status	Language
1	Done	C++
2	Error	Java

Non Coding

Question	Status
1	Done
2	Error

When displaying the submissions for each coding question in the interview

WITH SELECT *
FROM interview_coding_submission NATURAL JOIN coding_submission NATURAL JOIN submission
WHERE interview_id = <interview_id> and user_id = <user_id>

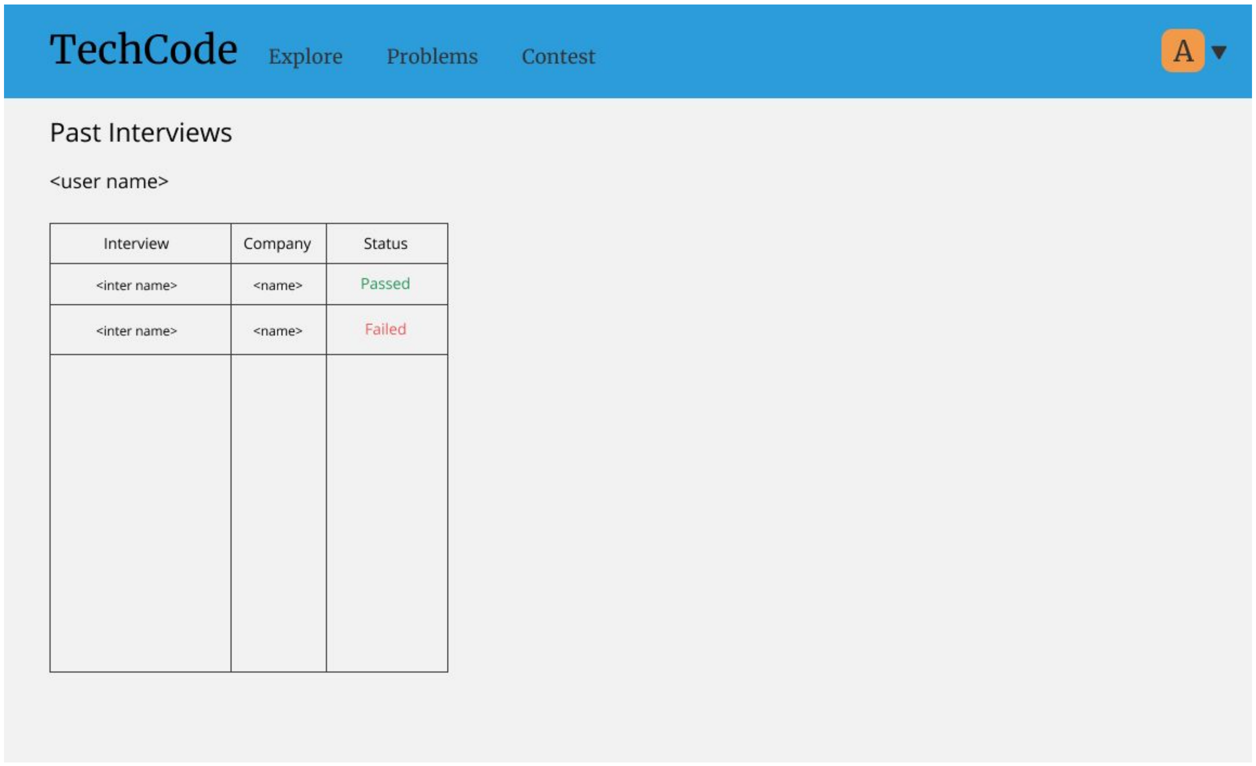
When displaying the submissions for each non-coding question in the interview

WITH SELECT *
FROM interview_non_coding_submission NATURAL JOIN non_coding_submission NATURAL JOIN submission
WHERE interview_id = <interview_id> and user_id = <user_id>

When the company determines the result of the interview

UPDATE user_interview_perform
SET result = <result>
WHERE user_id = <user_id> and interview_id = <interview_id>

Interview Results Page (User)



Past Interviews

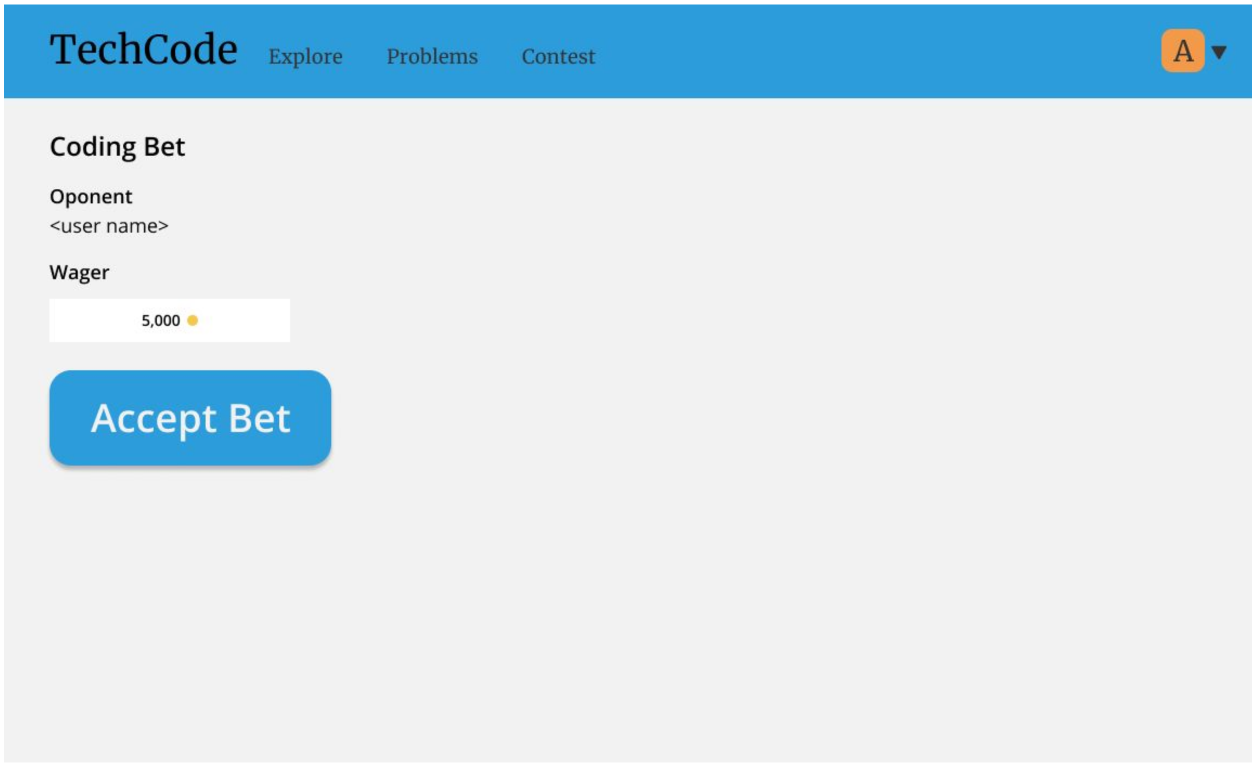
<user name>

Interview	Company	Status
<inter name>	<name>	Passed
<inter name>	<name>	Failed

To display the previous interview given by the user

```
SELECT *
FROM user_interview_perform NATURAL JOIN interview NATURAL JOIN
company_interview_prepare C  JOIN account A ON C.company_id = A.account_id
WHERE user_id = <user_id>
```

Coding Bet Page



Coding Bet

Oponent
<user name>

Wager

5,000 ●

When the user joins a coidng bet with another user

INSERT INTO coding_bet
VALUES (<coding_bet_id>, <bet_amount>)

INSERT INTO user_coding_bet_participate
VALUES (<user_id>, <coding_bet_id>)

INSERT INTO coding_bet_task_relation
VALUES (<coding_bet_id>, <task_id>)

5. Advanced database components

5.1. Views

Public Account View

This view will be used in queries that do not include a sign in or a signup operation. By creating such a view we ensure that sensitive information such as the email or the password of the account is not publicly displayed.

```
CREATE VIEW account_public(id,name,info) as
SELECT id, name, info
FROM account;
```

User Submission Detail View

This view will be used to query the submission details of a user. By creating this view makes our queries, which we execute a lot for each task, simpler.

```
CREATE VIEW user_submission_detail
(submission_id, user_id, task_id, answer, prog_lang, status, date)
SELECT *
FROM public_coding_submission NATURAL JOIN coding_submission
NATURAL JOIN submission
```

User Submissions View

This view will be used to query the submissions of a user. By creating this view we ensure that other sensitive information about the submission is not shown and this view makes our queries, which we execute a lot for each task, simpler.

```
CREATE VIEW user_submission
(submission_id, user_id, task_id, prog_lang, status, date) as
SELECT submission_id, user_id, task_id, prog_lang, status, date
FROM user_submission_detail
```

5.2. Stored Procedures and Reports

Stored Procedures

We believe that reusing and recycling code is critical and we believe that writing code that is simple, easy to read and easy to understand is beneficial to the development of the project. Therefore, in TechCode we will use procedures for queries instead of writing the same query over and over in several pages. These procedures will be very similar to methods and functions. They will take parameters that a query needs and execute it using the passed parameters.

To display every public coding problem that is in a given category

```
CREATE PROCEDURE selectAllPublicCodingTaskFromCategory
 (@Category varchar(30))
AS
BEGIN
        SELECT *
        FROM public_coding_task NATURAL JOIN public_task
        NATURAL JOIN task
        WHERE category = @Category
        ORDER BY task_id ASC
END
```

To display every public coding problem that is in a given category with a given search input

```
CREATE PROCEDURE selectAllPublicCodingTaskFromCategoryWithSearch
(@category varchar(30), @search_input)
AS
BEGIN
        SELECT *
        FROM public_coding_task NATURAL JOIN public_task
        NATURAL JOIN task
        WHERE category = @category and title LIKE %search_input%
        ORDER BY task_id ASC
END
```

To display every public non-coding problem that is in a given category

```
CREATE PROCEDURE selectAllPublicNonCodingFromCategory
(@category varchar(30))
AS
BEGIN
    SELECT *
    FROM (
    public_non_coding_task NATURAL JOIN public_task
    NATURAL JOIN task) as T
    JOIN editor_public_task_prepare as E on E.task_id = T.task_id
    WHERE category = @category
    ORDER BY task_id ASC
END
```

To get the account information of a signed in user

```
CREATE PROCEDURE getUserInformation (@email varchar(30), @password
varchar(10), @account_type)
AS
BEGIN
    IF @account_type = ‘user’
    BEGIN
        SELECT *
        FROM user as U JOIN account as A on U.user_id = A.account_id
        WHERE email = <email> and password = <password>
    END
    ELSE IF @account_type = ‘editor’
    BEGIN
        SELECT *
        FROM editor  E JOIN account  A on E.editor_id = A.account_id
        WHERE email = <email> and password = <password>
    END
    ELSE IF @account_type = ‘company’
    BEGIN
        SELECT *
        FROM
        company C JOIN account A on C.company_id = A.account_id
        WHERE email = <email> and password = <password>
    END
END
```

To display the current ranking of a contest given the contest_id

```
CREATE PROCEDURE showRankingOfContest (@category varchar(30))
AS
BEGIN
        WITH user_points as SELECT S.user_id ,SUM(points_given) as total
        FROM (contest_task NATURAL JOIN task) as T
        JOIN (submission NATURAL JOIN user) as S ON T.task_id = S.task_id
        WHERE contest_id = <contest_id> and status = ‘solved’
        GROUP BY S.user_id
        ORDER BY total DESC,
        SELECT distinct name, total
        FROM user_points NATURAL JOIN user
END
```

Reports

Number of public-coding tasks

```
SELECT count(distinct *) as count
FROM coding_public_task
```

Number of solved public coding questions

```
SELECT count(distinct task_id) as count
FROM public_coding_submission NATURAL JOIN coding_submission
WHERE user_id = <user_id> and status=’solved’
```

Number of attempted public coding questions

```
SELECT count(distinct task_id) as count
FROM  public_coding_submission NATURAL JOIN coding_submission
WHERE user_id = <user_id> and status <> ‘solved’
```

Number of submissions on a specific public coding question by a spesific user

```
SELECT count(distinct submission_id) as count
FROM public_coding_submission NATURAL JOIN coding_submission
NATURAL JOIN submission
WHERE user_id = <user_id> and task_id = <task_id>
```

Number of people that solved a public coding task

```
SELECT count(distinct user_id) as count
FROM public_coding_submission NATURAL JOIN coding_submission
NATURAL JOIN submission
WHERE task_id = <task_id> and status = ‘solved’
```

Number of submissions on a public coding task

```
SELECT count(distinct submission_id) as count
FROM public_coding_submission NATURAL JOIN  coding_submission
NATURAL JOIN submission
WHERE task_id = <task_id>
```

Number of public-non-coding tasks

```
SELECT count(distinct *) as count
FROM non_coding_public_task
```

Number of upvotes on an answer

```
SELECT count(distinct user_id) as count
FROM rate_answer
WHERE answer_id = <answer_id> and R.rate_type = 'upvote'
```

Number of downvotes on an answer

```
SELECT count(distinct user_id) as count
FROM rate_answer
WHERE answer_id = <answer_id> and R.rate_type = 'downvote'
```

Number of people that are competing in a contest

```
SELECT count(distinct user_id)
FROM user_contest_participate
WHERE contest_id = <contest_id>
```

Number of tasks created by an editor

```
SELECT count(distinct task_id)
FROM editor_public_task_prepare
WHERE editor_id = <editor_id>
```

Number of interviews given by a user

```
SELECT count(distinct interview_id)
FROM user_interview_perform
WHERE user_id = <user_id>
```

5.3. Triggers

When the user solves a public coding task total score of the user is increased by the points given by the task

```
CREATE TRIGGER addPoints
AFTER INSERT ON public_coding_submission S
FOR EACH ROW
BEGIN
UPDATE user SET score = score + (SELECT points_given
FROM public_coding_task as T WHERE T.task_id = S.task_id)
END;
```

When the user wins a coding bet total tech coins of the winner increased by the wager on the bet

5.4. Constraints

1. Users must register to the system in order to use the web-based application. If they are already registered to the system they must log in to the system in order to use it.
2. There can be at most 30 rows of questions displayed in the Problems Page.
3. Editors cannot post non-coding questions with an empty title.
4. Editors cannot post empty non-coding questions.
5. Users cannot post empty answers to non-coding questions.
6. A coding bet must have a positive wager.
7. Users cannot bet more than their coin amount in a coding bet.
8. Users cannot join a contest that has already started.
9. Editors cannot create interviews with a duration that is less than 10 minutes.
10. A task can not belong to more than one coding contest
11. A task can not belong to more than one coding interview
12. Only two users can participate in a coding bet challenge

6. Implementation Plan

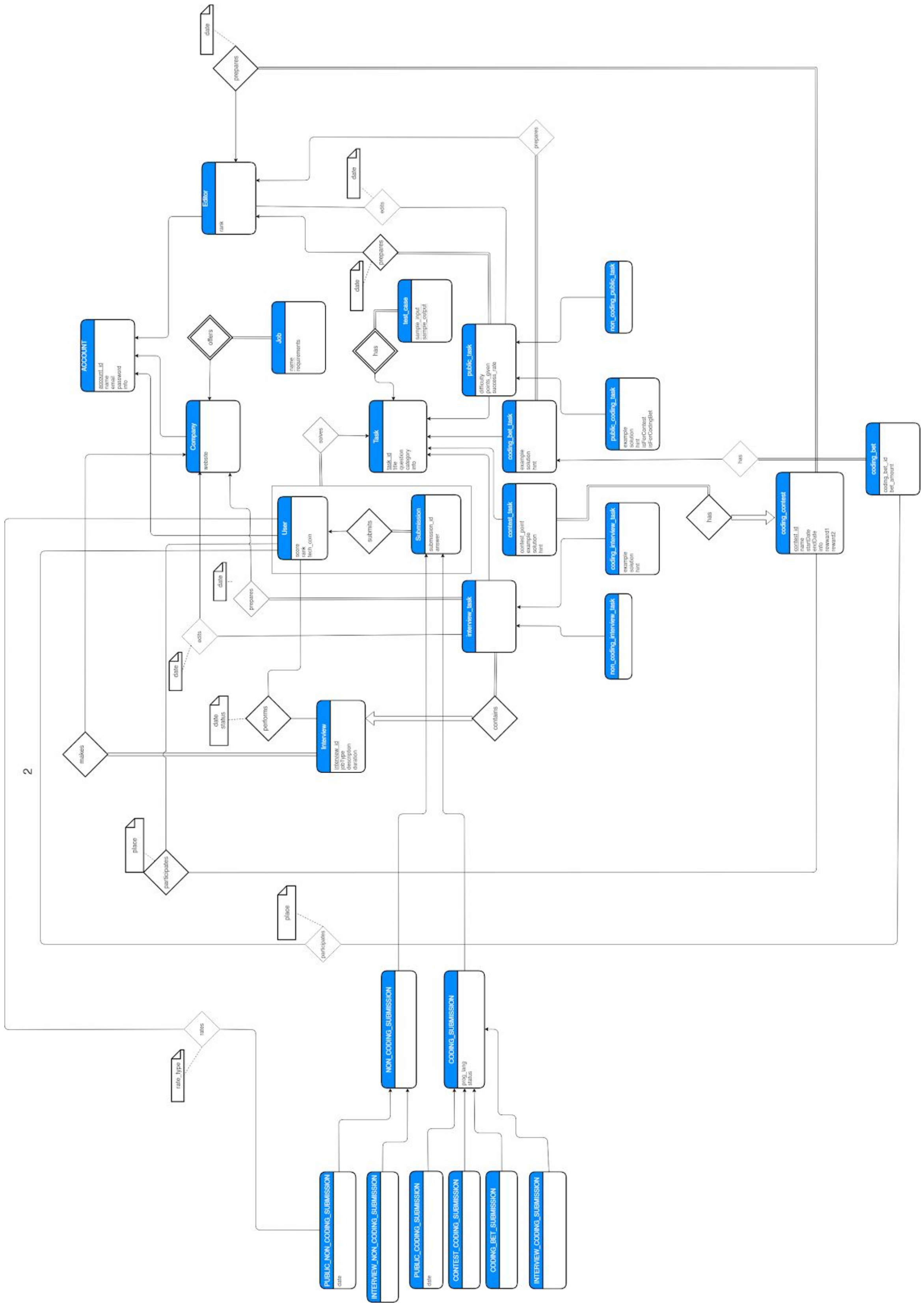
We are planning to use PHP, Javascript, Angular, Jquery and Bootstrap for designing and developing the UI of our web application. HTML and CSS will be used to implement the front end of our web application. We are planning to use PHP for server side and Javascript for client side. We will use MySQL for database management.

7. Website

Url: https://technicalinterviews.github.io/

It is a one-page HTML site, please use the "Download Design" button to access our Project Design as a pdf.

Appendix 1


ACCOUNT
Editor
Company
Job
User
Submission
Task
test_case
public_task
coding_bet_task
contest_task
interview_task
coding_contest
coding_bet
non_coding_public_task
public_coding_task
coding_interview_task
non_coding_interview_task
Interview
NON_CODING_SUBMISSION
CODING_SUBMISSION
PUBLIC_NON_CODING_SUBMISSION
INTERVIEW_NON_CODING_SUBMISSION
PUBLIC_CODING_SUBMISSION
CONTEST_CODING_SUBMISSION
CODING_BET_SUBMISSION
INTERVIEW_CODING_SUBMISSION
prepares
edits
offers
has
solves
submits
performs
contains
makes
participates
rates
date
place
rate_type
2