CS 353 Database Systems

Project Design Report

Coding Practice and Interview System

MasterCoding

Group 16

Tanay Toksoy | 21703919 - Sec 2

Ali Kemal Özkan | 21302087 - Sec 3

Osman Can Yıldız | 21302616 - Sec 3

Göktuğ Öztürkcan | 21702290 - Sec 2

10th April, 2020

Project Website

https://alikemalozkan.github.io/CodingPractice353/

Table of Contents

1. E-R Diagram

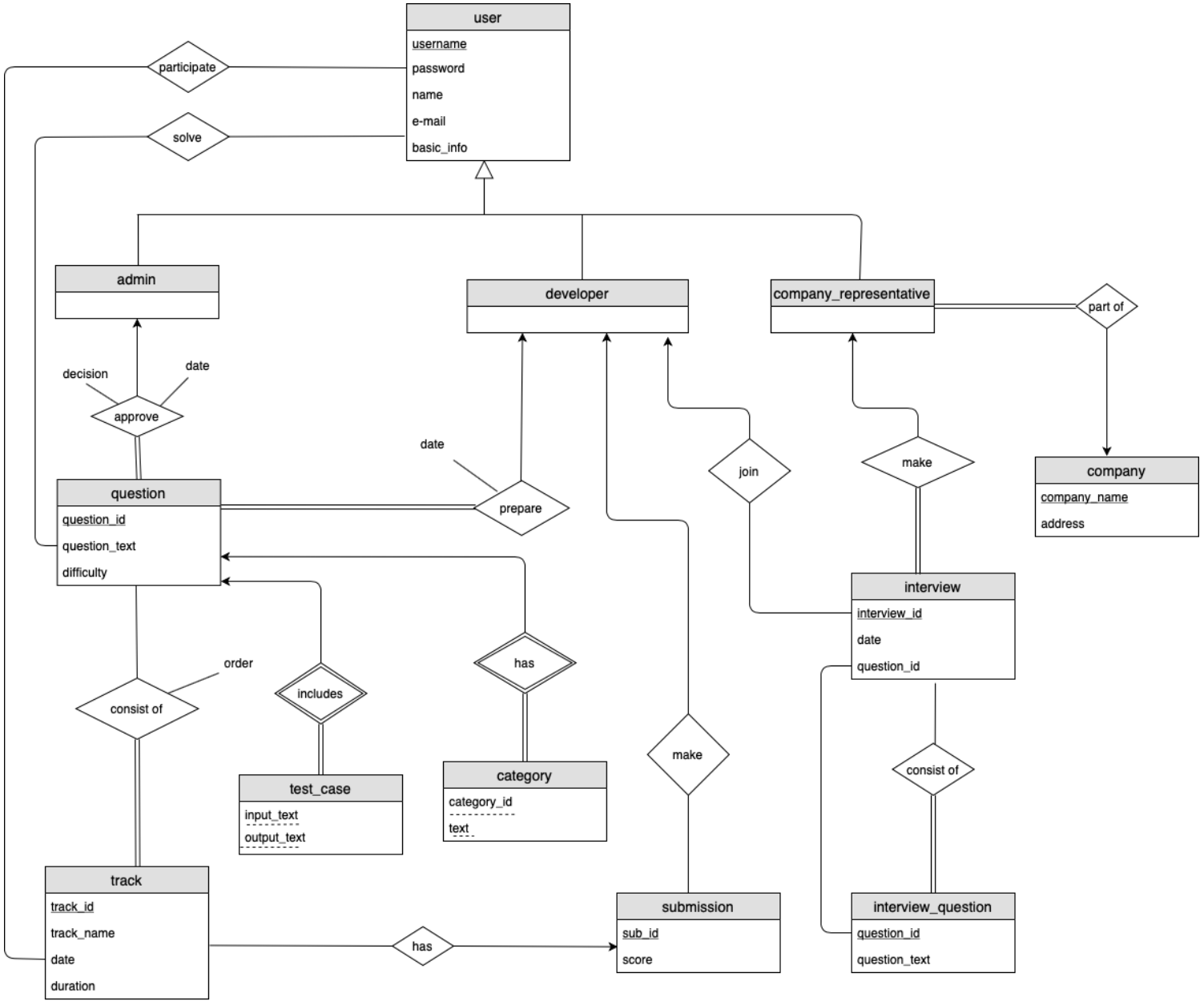


Figure 1: E-R Diagram

2. Table Schemas

2.1. User

Relational Model:

user(<u>username</u>, password, name, e-mail, basic_info)

Functional Dependencies:

username → password, name, e-mail, basic_info

Candidate Keys:

{(username)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE user(
        username varchar(20) PRIMARY KEY,
        password varchar(20) NOT NULL,
        name varchar(30) NOT NULL,
        e-mail varchar(50) NOT NULL,
        basic_info varchar(500)
);
```

2.2. Admin

Relational Model:

admin(<u>username</u>)

Functional Dependencies:

Candidate Keys:

{(username)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE admin(
        username varchar(20) PRIMARY KEY,
        FOREIGN KEY (username) references (user)
);
```

2.3. Developer

Relational Model:

developer(<u>username</u>)

Functional Dependencies:

Candidate Keys:

{(username)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE developer(
        username varchar(20) PRIMARY KEY,
        FOREIGN KEY (username) references (user)
);
```

2.4. Company Representative

Relational Model:

company_representative(<u>username</u>, company_name)

Functional Dependencies:

Candidate Keys:

{(username)}

Table Definition:

```
CREATE TABLE company_representative(
        username varchar(20) PRIMARY KEY,
        company_name varchar(30),
        FOREIGN KEY (username) references (user),
        FOREIGN KEY (company_name) references (company)
);
```

2.5. Company

Relational Model:

company(<u>company_name</u>, address, username)

Functional Dependencies:

company_name → address

Candidate Keys:

{(company_name), (username)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE company(
        company_name varchar(30) PRIMARY KEY,
        address varchar(200),
        FOREIGN KEY (username) references (company_representative)
);
```

2.6. Question

Relational Model:

question(<u>question_id</u>, username, question_text, difficulty)

Functional Dependencies:

question_id → question_text, difficulty

Candidate Keys:

{(question_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE question(
        question_id int PRIMARY KEY,
        question_text varchar(500) NOT NULL,
        difficulty varchar(10),
        username varchar(20),
        FOREIGN KEY (username) references (developer)
);
```

2.7. Interview

Relational Model:

interview(<u>interview_id</u>, date, question_id, username)

Functional Dependencies:

interview_id → date, question_id

Candidate Keys:

{(interview_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE interview(
        interview_id int PRIMARY KEY,
        date date,
        question_id int,
        developer_id int,
        FOREIGN KEY (question_id) references (interview_question),
        FOREIGN KEY (username) references (company_representative)
);
```

2.8. Interview Question

Relational Model:

interview_question(question_id, question_text)

Functional Dependencies:

question_id → question_text

Candidate Keys:

{(question_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE interview_question(
        question_id int PRIMARY KEY,
        question_text varchar(500)
);
```

2.9. Track

Relational Model:

track(track_id, track_name, date, duration)

Functional Dependencies:

track_id → track_name, date, duration

Candidate Keys:

{(track_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE track(
        track_id int PRIMARY KEY,
```

```
        track_name varchar(50),
        date date,
        duration date
);
```

2.10. Test case

Relational Model:

test_case(<u>question_id</u>, <u>input_text</u>, <u>output_tex,t</u>)

Functional Dependencies:

Candidate Keys:

{(input_text, output_text)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE question(
        question_id int PRIMARY KEY,
        input_text varchar(500),
        output_text varchar(500),
        PRIMARY KEY (question_id, input_text, output_text),
        FOREIGN KEY (question_id) references (question)
);
```

2.11. Category

Relational Model:

category(<u>question_id</u>, <u>category_id</u>, <u>text</u>)

Functional Dependencies:

Candidate Keys:

{(question_id, category_id, text)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE category(
        question_id int PRIMARY KEY,
        category_id int,
        text varchar(30),
        PRIMARY KEY (question_id, category_id, text),
```

```
        FOREIGN KEY (question_id), references (question)
);
```

2.12. Submission

Relational Model:

submission(<u>sub_id</u>, score)

Functional Dependencies:

sub_id → score

Candidate Keys:

{(sub_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE user(
        sub_id int PRIMARY KEY,
        score int,
        FOREIGN KEY (username) references developer
);
```

2.13. Admin_Question

Relational Model:

admin_question(<u>username</u>, <u>question_id</u>, <u>date</u>, <u>decision</u>)

Functional Dependencies:

Candidate Keys:

{(username, question_id, date, decision)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE admin_question(
        date date,
        decision varchar(10),
        username int,
        question_id int,
        PRIMARY KEY (username, question_id, date, decision),
        FOREIGN KEY (username) references (admin),
        FOREIGN KEY (question_id) references (question)
```

```
);
```

2.14. Question_Track

Relational Model:

question_track(question_id, track_id, order)

Functional Dependencies:

Candidate Keys:

{(question_id, track_id, order)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE question_track(
        order int,
        question_id int,
        track_id int,
        PRIMARY KEY (question_id, track_id, order),
        FOREIGN KEY (question_id) references (question),
        FOREIGN KEY (track_id) references (track)
);
```

2.15. Interview_Interview_Question

Relational Model: interview_interview_question(interview_id, question_id);

Functional Dependencies:

Candidate Keys: {(interview_id, interview_question_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE interview_interview_question(
    interview_id int,
    interview_question_id int,
    PRIMARY KEY (interview_id, interview_question_id)
    FOREIGN KEY (interview_id) references interview(interview_id)
    FOREIGN KEY (interview_question_id) references
interview_question(question_id)
);
```

2.16. User_Track

Relational Model: user_track(<u>username</u>, <u>track_id</u>, track_name, date, duration);

Functional Dependencies:

Candidate Keys: {(username, track_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE user_track(
    username varchar(20),
    track_id int,
    track_name varchar(50),
    date date,
    duration date,
    PRIMARY KEY (username, track_id)
user_track(track_id)
);
```

2.17. User_Question

Relational Model: user_question(<u>username</u>, <u>question_id</u>, question_text);

Functional Dependencies:

Candidate Keys: {(username, question_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE user_question(
    username varchar(20),
    question_id int ,
    question_text varchar(500),
    PRIMARY KEY (username, question_id)
user_question(question_id)
);
```

2.18. Developer_Submission

Relational Model: developer_submission(<u>username</u>, <u>sub_id</u>, score);

Functional Dependencies:

Candidate Keys: {username, sub_id)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE developer_submission(
    username varchar(20),
    sub_id int ,
    score int,
   PRIMARY KEY (username, sub_id)
developer_submission(sub_id)
);
```

3. User Interface Design and Sql Statements

3.1. Sign Up

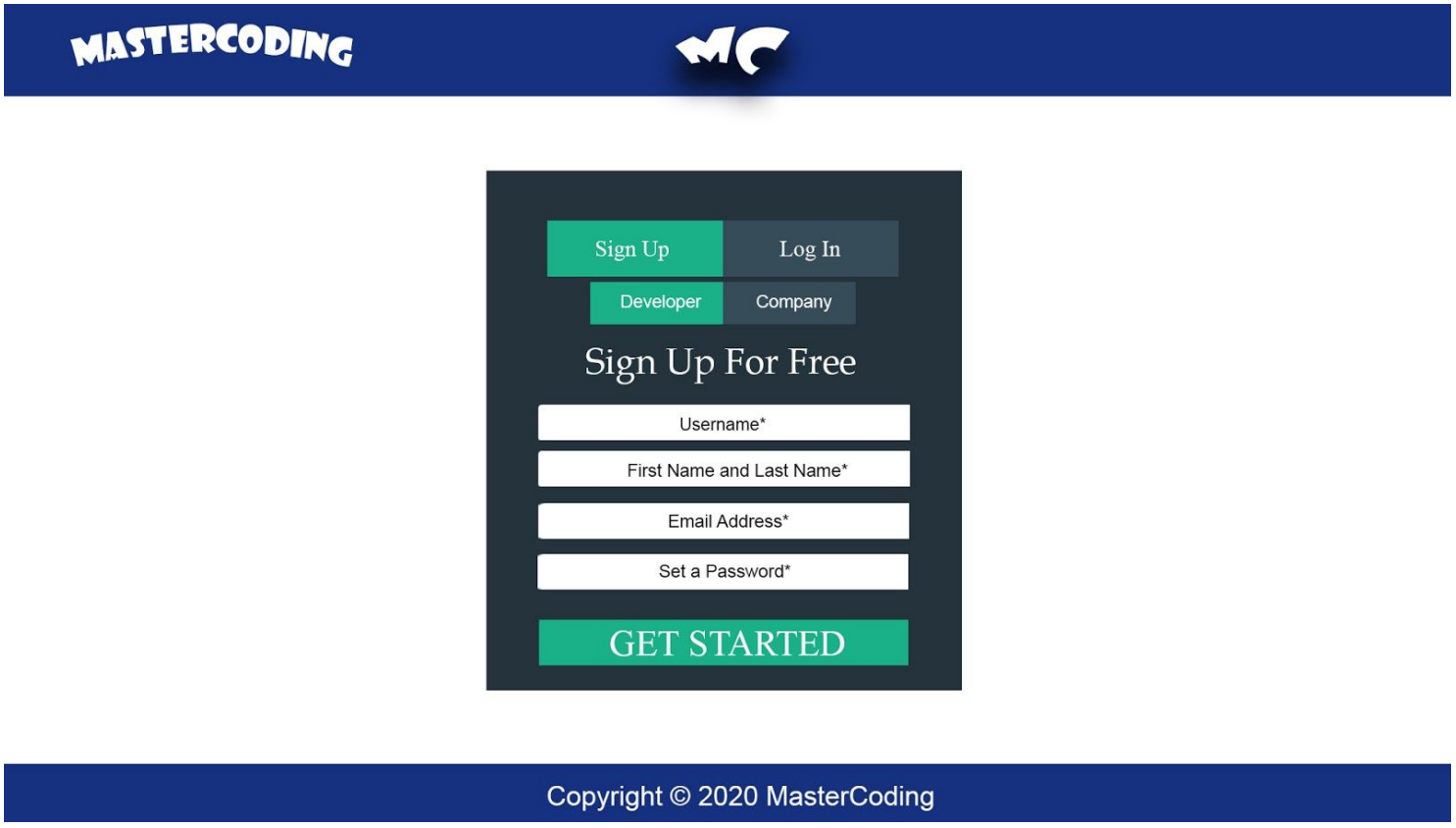


Figure 2: Sign up screen

Users can sign up with username, full name, e-mail and password as a developer or company representative.

Sql Statements

```
INSERT INTO user
VALUES (@username, @password, @name, @e-mail, NULL);
```

3.2. Login

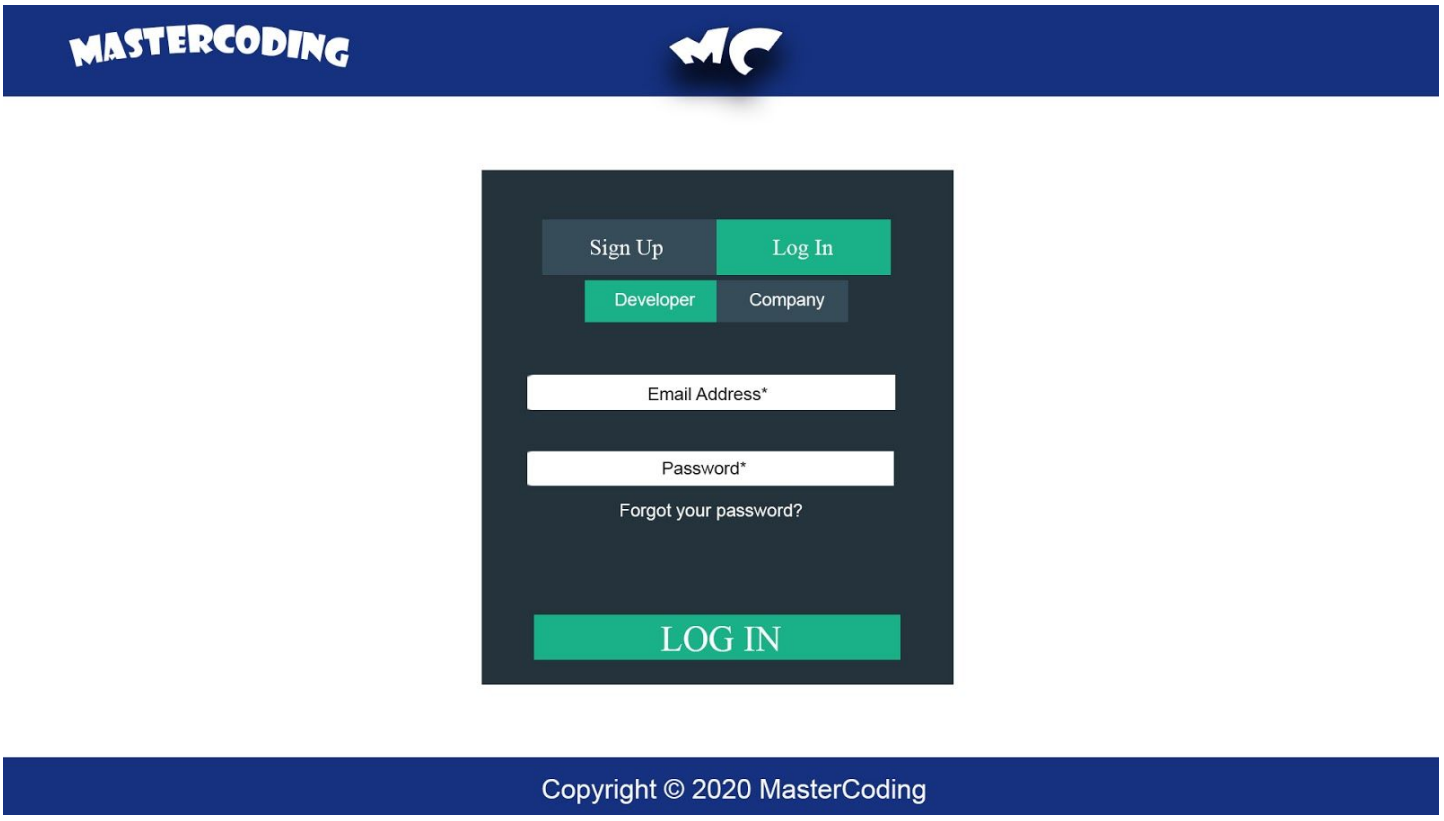


Figure 3: Login screen for developers

Users can login with their e-mail address and password.

Sql Statements

```
SELECT *
FROM user
WHERE user.username = @username AND user.password = @password;
```

3.3. Dashboard

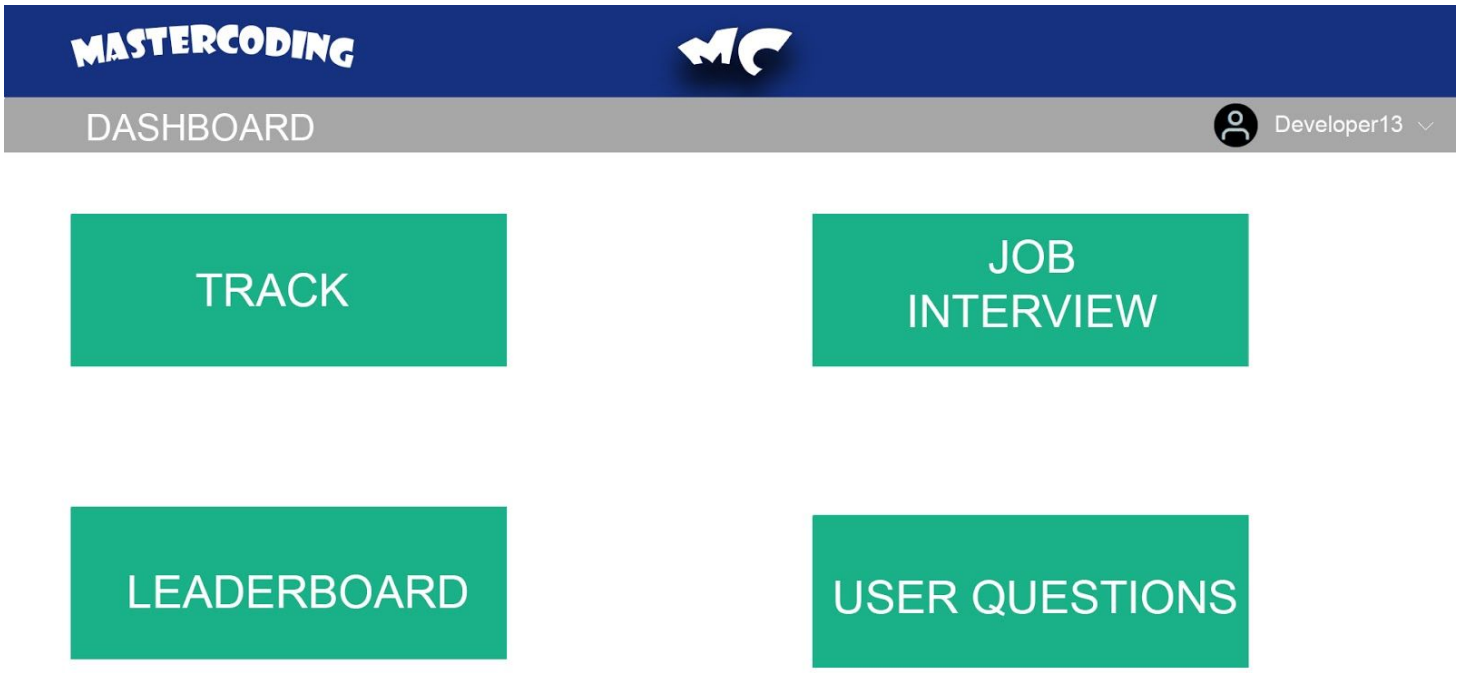


Figure 4: Dashboard for Developer

Dashboard screen works like a main menu where the developer can either access a track, job interview,leaderboard or the user questions.

Sql Statements

Listing Tracks:

```
SELECT track_name
FROM track
```

Listing User Questions:

```
SELECT question_text, difficulty
FROM question
```

Listing Job Interviews:

```
SELECT interview_id,date
FROM interview
```

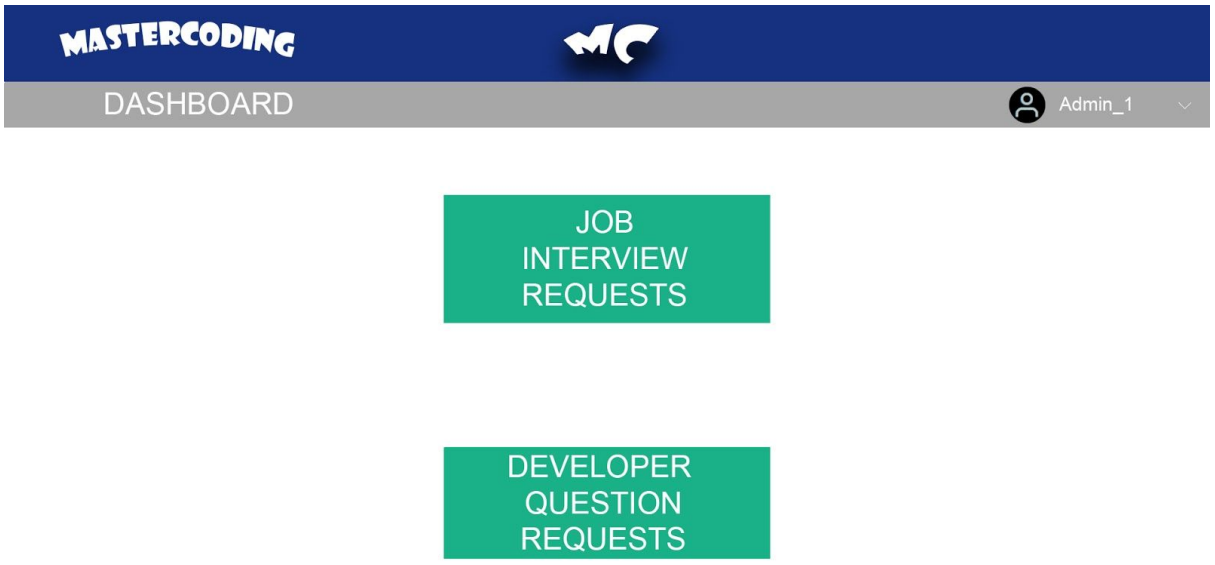


Figure 5: Dashboard for Admin

Admin can see job interview and developer question requests. Only admin can approve requests.

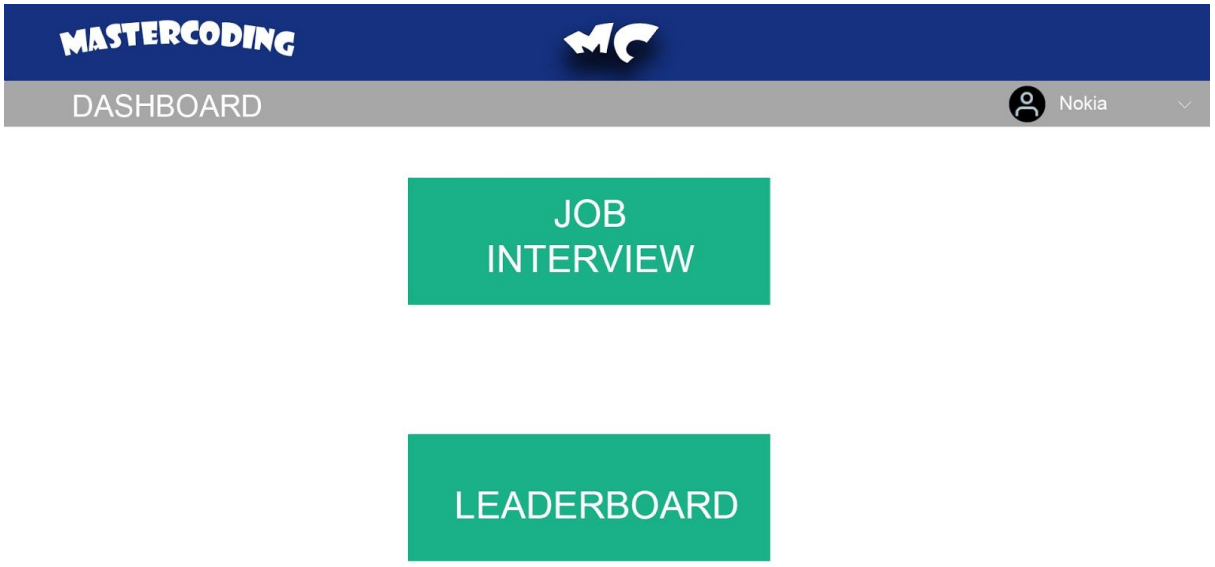
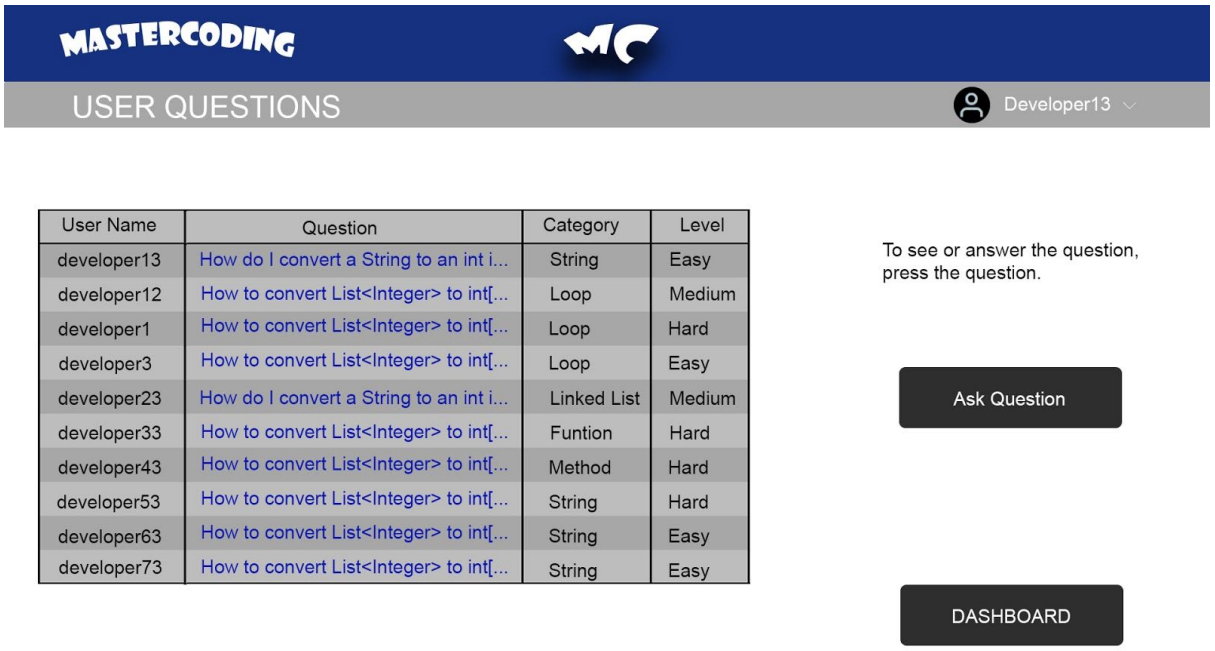


Figure 6: Dashboard for Company

Company representatives can see leaderboard and can make a job interview. Companies can prepare a track for interview.

3.4. Question List



User Name	Question	Category	Level
developer13	How do I convert a String to an int i...	String	Easy
developer12	How to convert List<Integer> to int[...	Loop	Medium
developer1	How to convert List<Integer> to int[...	Loop	Hard
developer3	How to convert List<Integer> to int[...	Loop	Easy
developer23	How do I convert a String to an int i...	Linked List	Medium
developer33	How to convert List<Integer> to int[...	Funtion	Hard
developer43	How to convert List<Integer> to int[...	Method	Hard
developer53	How to convert List<Integer> to int[...	String	Hard
developer63	How to convert List<Integer> to int[...	String	Easy
developer73	How to convert List<Integer> to int[...	String	Easy

Figure 7: Question List

This screen shows the question list created by the developers which are all categorized and separated by their difficulty level. Ask question button allows the users to create their own questions. Dashboard button takes the user back to Dashboard.

Sql Statements

Selecting a Question to do:

```
SELECT *
FROM question
WHERE question.question_id=@question_id
```

Asking a Question:

```
INSERT INTO question(question_id,question text,author)
VALUES(value1,value2,value3);
```

3.5. Ask Question Page

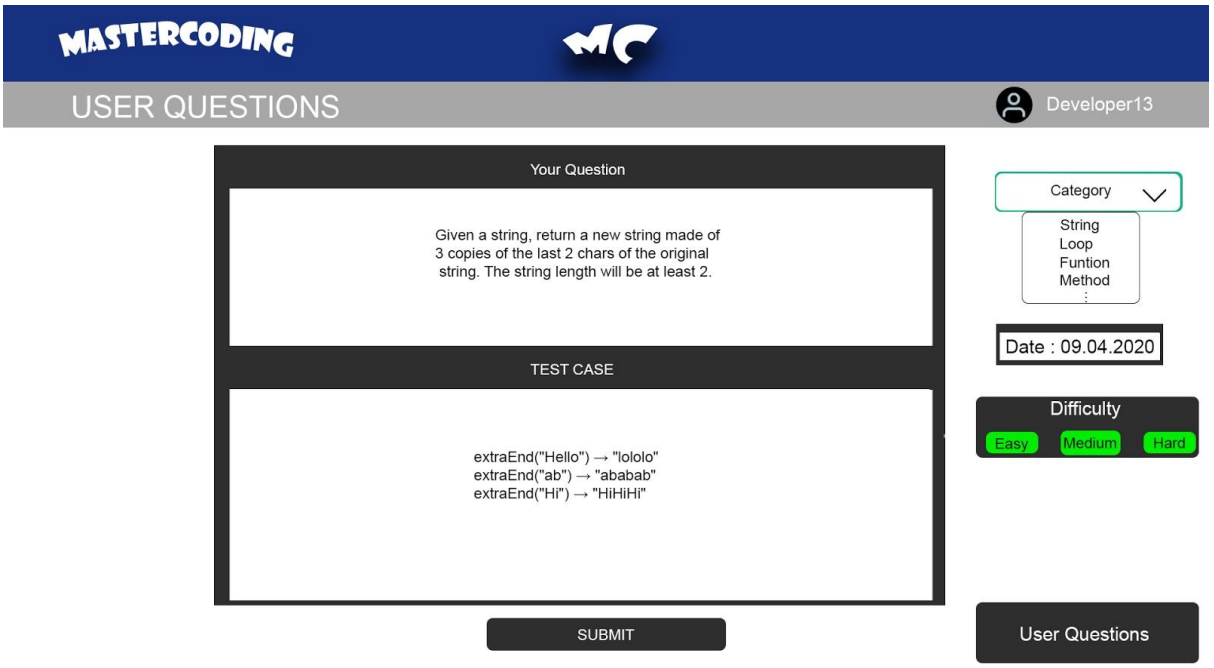


Figure 8: Ask Question Screen for Developer

Users can submit a question with test cases. Developer have to choose difficulty and category of the question.

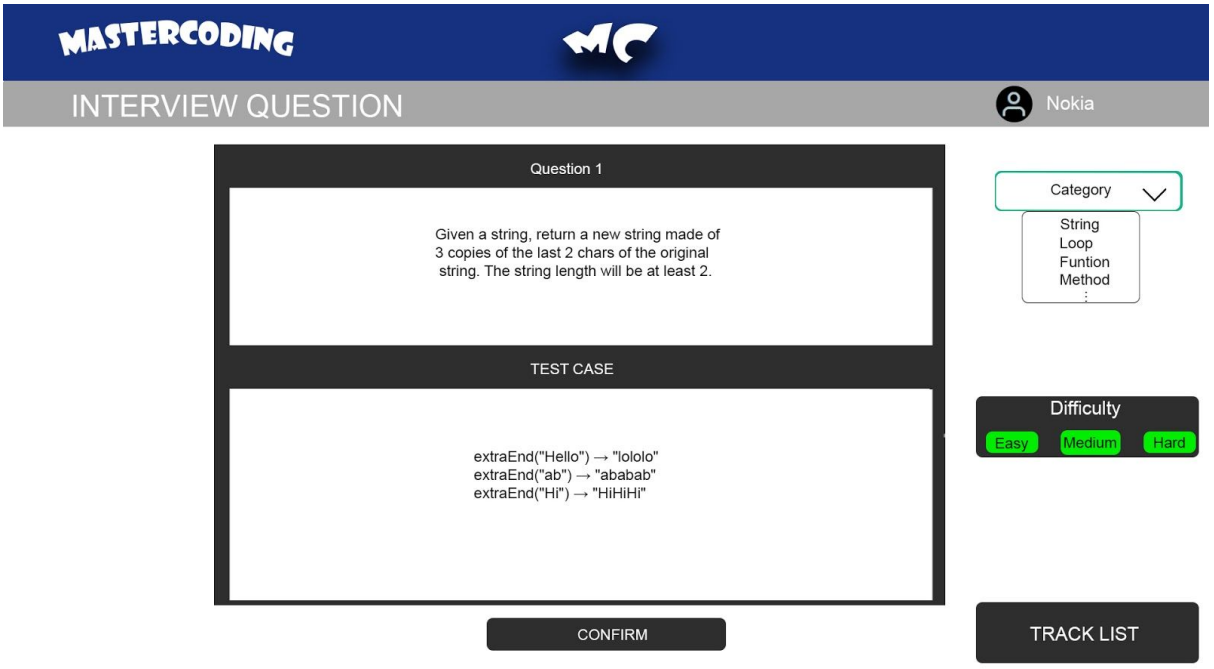


Figure 9: Prepare Question Page for Company

3.6. Solve Question Page

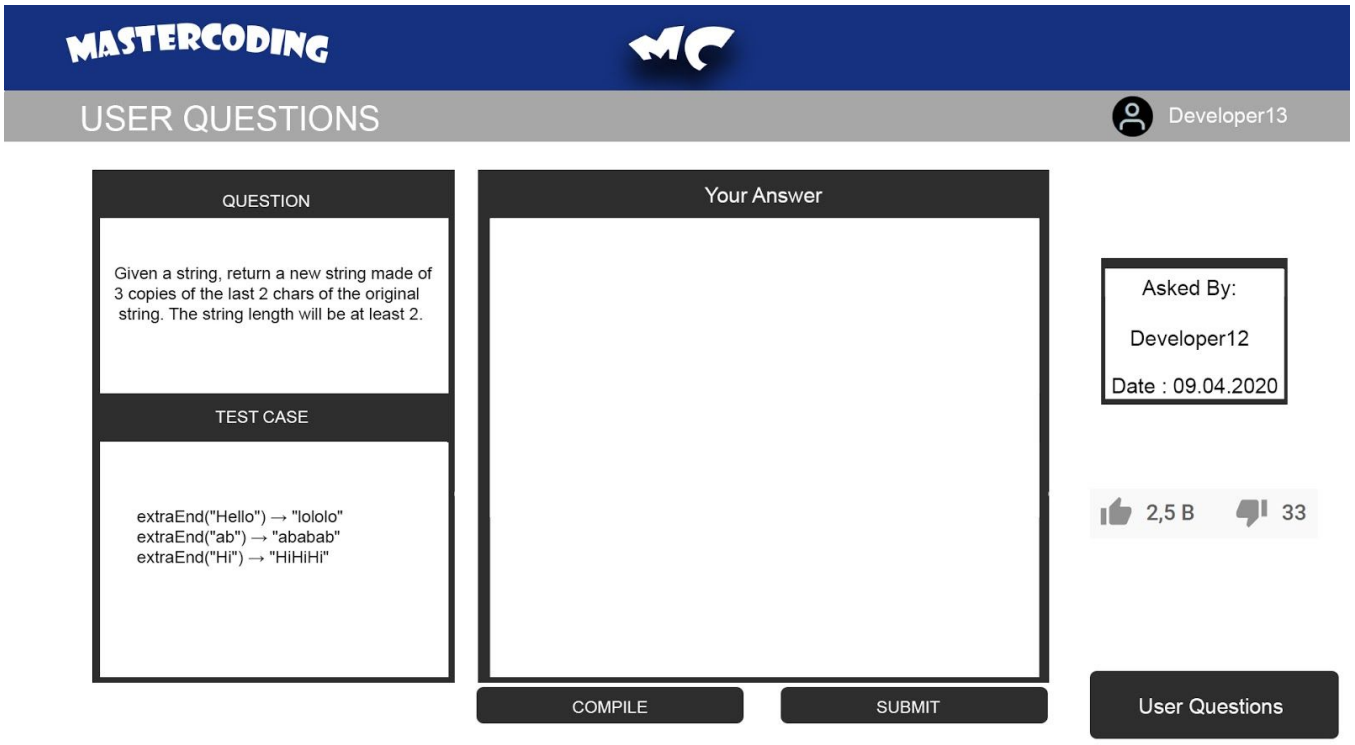


Figure 10: Solve Question Page for Developer Questions

In this screen the user can see the questions asked by other developers and then compile their programs with a test case and then submit their answers. If the User Questions is pressed it Lists all the user questions again. Users can also like or dislike the question asked.

Sql Statements

Submit answer:

INSERT INTO submission(sub_id) VALUES(val1);

3.7. Leaderboard

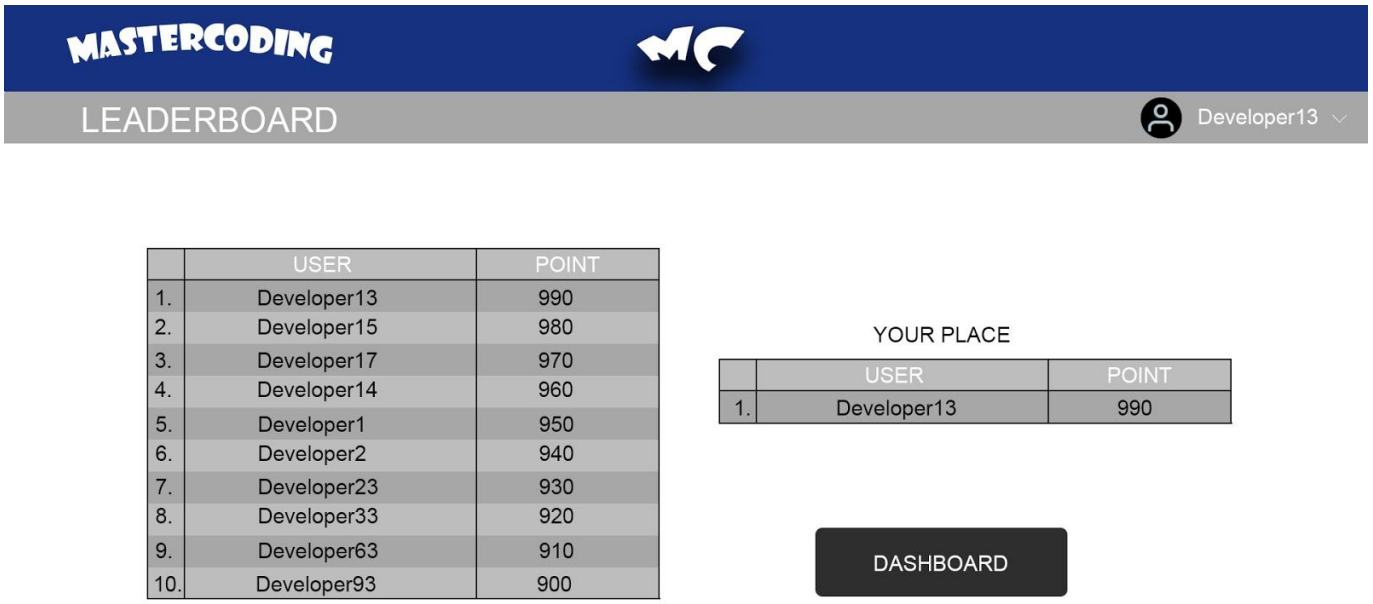


Figure 11: Leaderboard

Leaderboard screen shows the leaderboard and specifically shows the current place of the user in the leaderboard.

Sql Statements

Listing LeaderBoard

```
SELECT *
FROM leaderboard
```

3.8. Choose Track

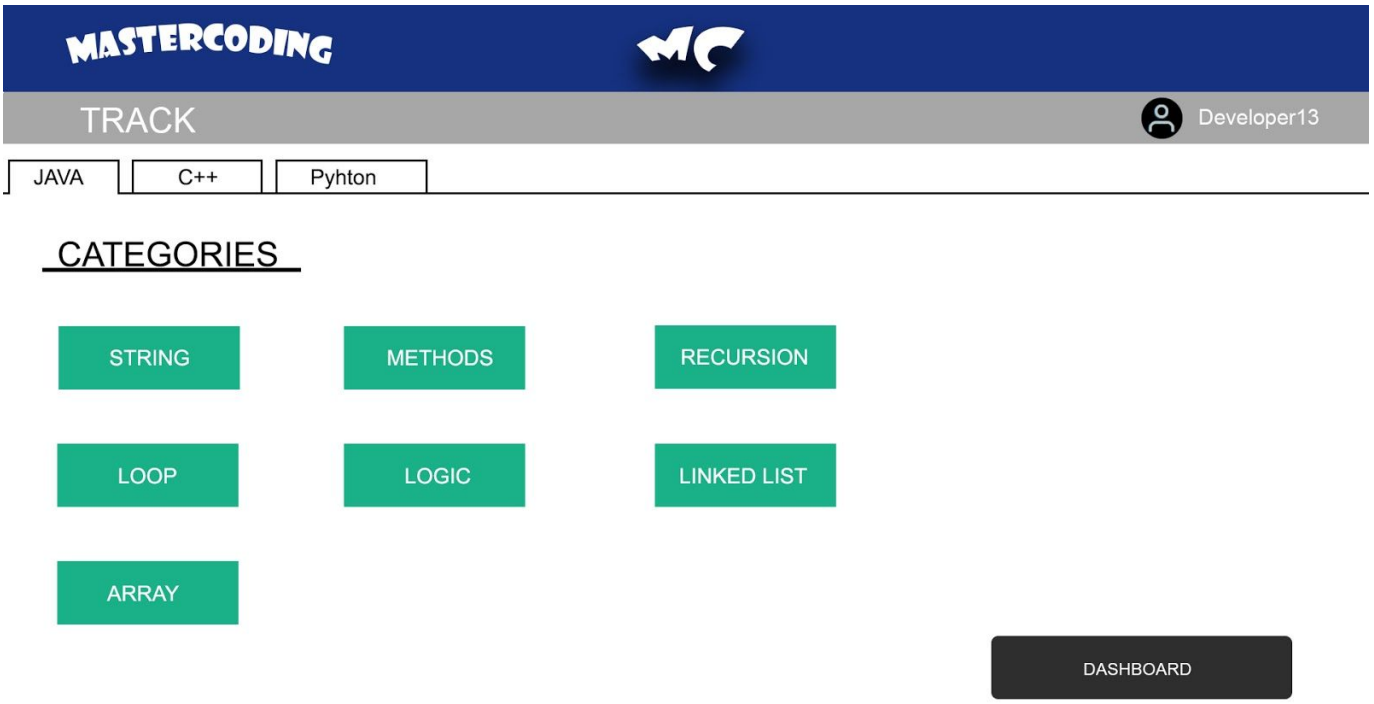


Figure 12: Choose Track Page

This acts as the main screen for tracks from where users can choose the programming language and then choose a category for which he/she will take part in. Or the user can return to the dashboard.

Sql Statements

```
SELECT track_name
FROM track
WHERE track.track_id=@track_id
```

3.9. Track Progress

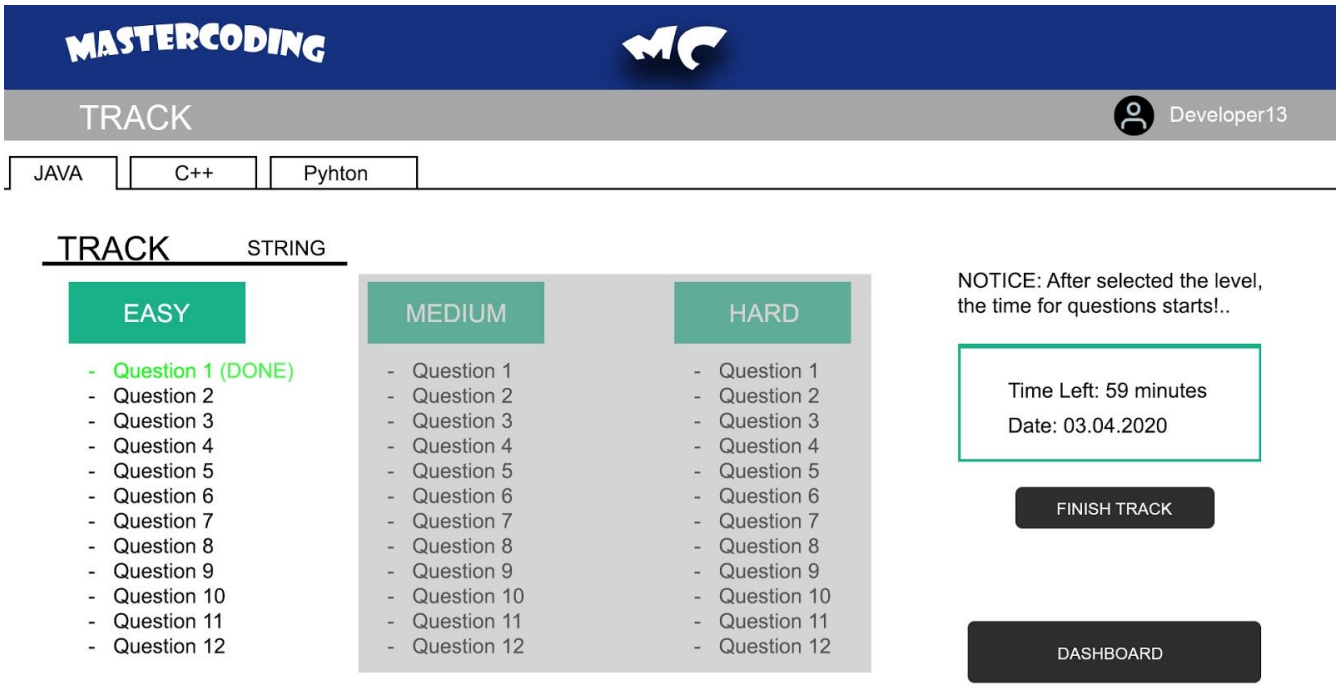


Figure 13: Track Progress Page

Tracks main screen shows which questions have been completed and time remaining. Through here users can access their next questions or return to the dashboard.

Sql Statements

Finish track:

INSERT INTO submission(sub_id, sub_score) VALUES (val1,val2,val3)

3.10. Submit Track Question

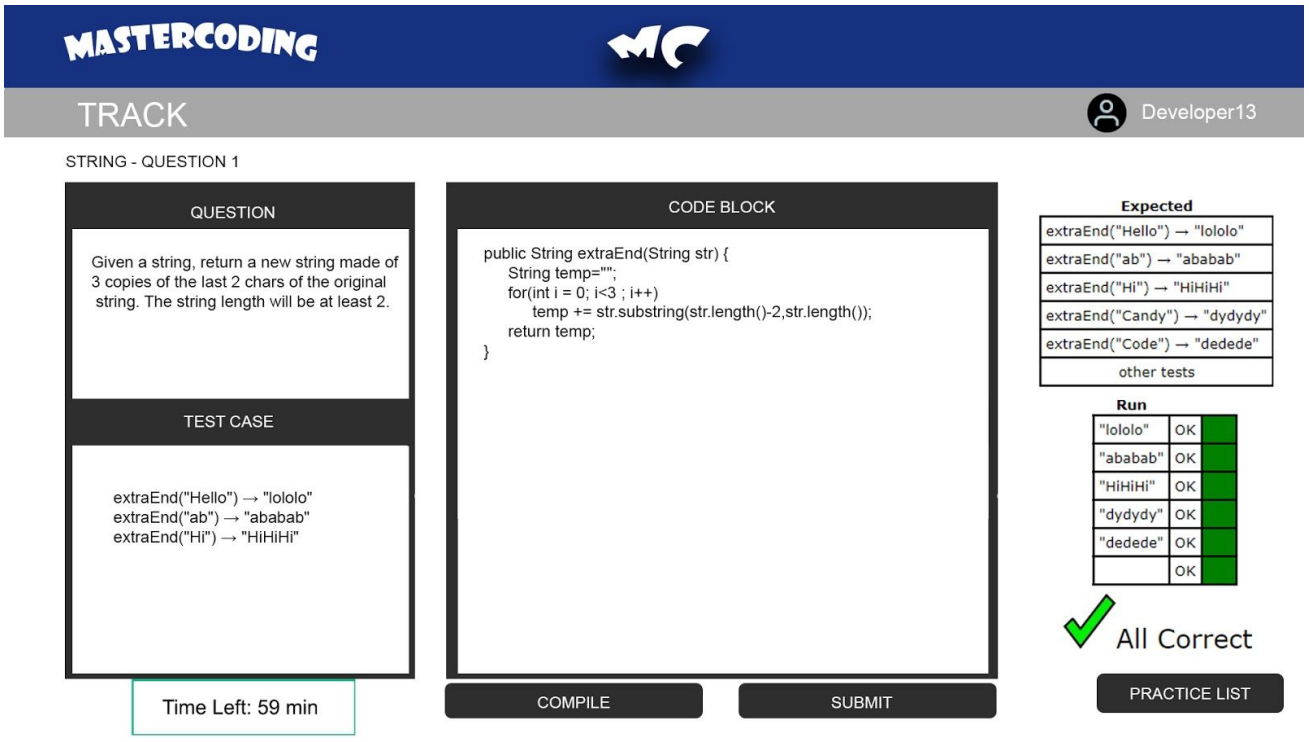


Figure 14: Submit Track Question Page (Correct answer)

Question screen on track shows the question and the user can enter the answer and compile his/her code and see results from the test cases. After user submits the answer and it is correct then the user proceeds on to the next question automatically.

Sql Statements

Post answer:

INSERT INTO submission(sub_id, answer) VALUES(val1,val2)

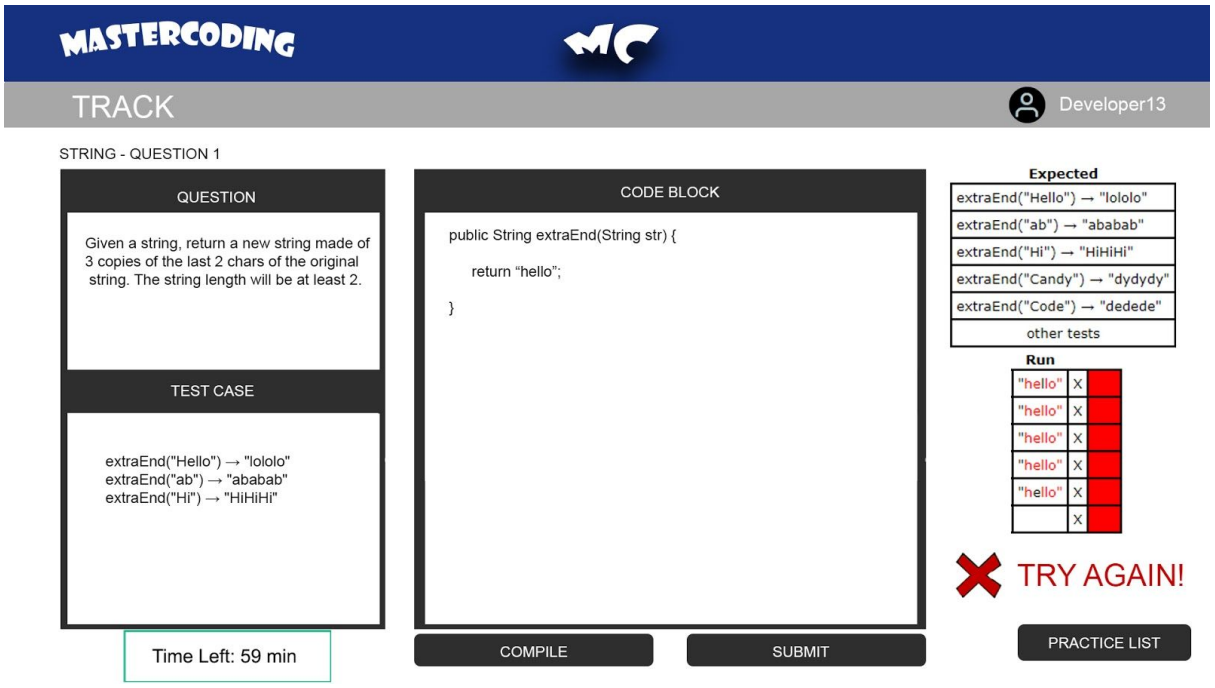


Figure 15: Submit Track Question Page (False answer)

If the user code is wrong after compilation the test cases will be checked and will be shown to the user which test cases worked and which did not.

3.11. Job Interview List

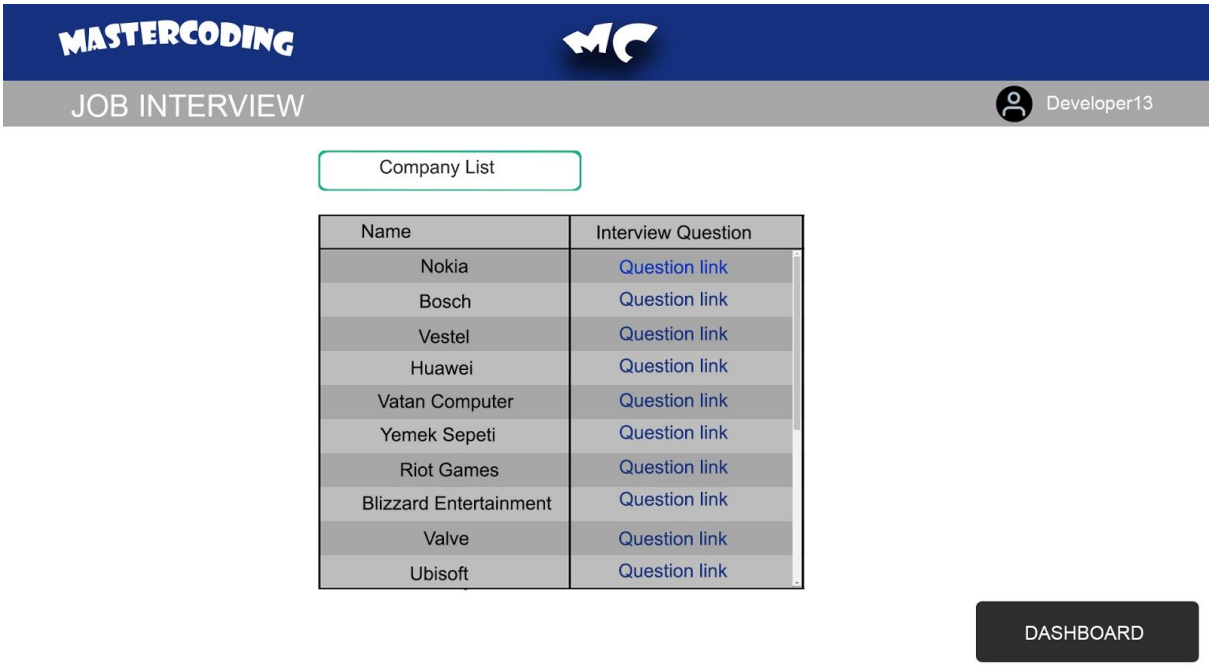


Figure 16: Interviews List

This page shows all the interviews available to the user. User can take the interviews by clicking on them or either return to dashboard by clicking the dashboard menu.

Sql Statements

List all interviews

SELECT *

FROM interview

3.12. Job Interview Question List

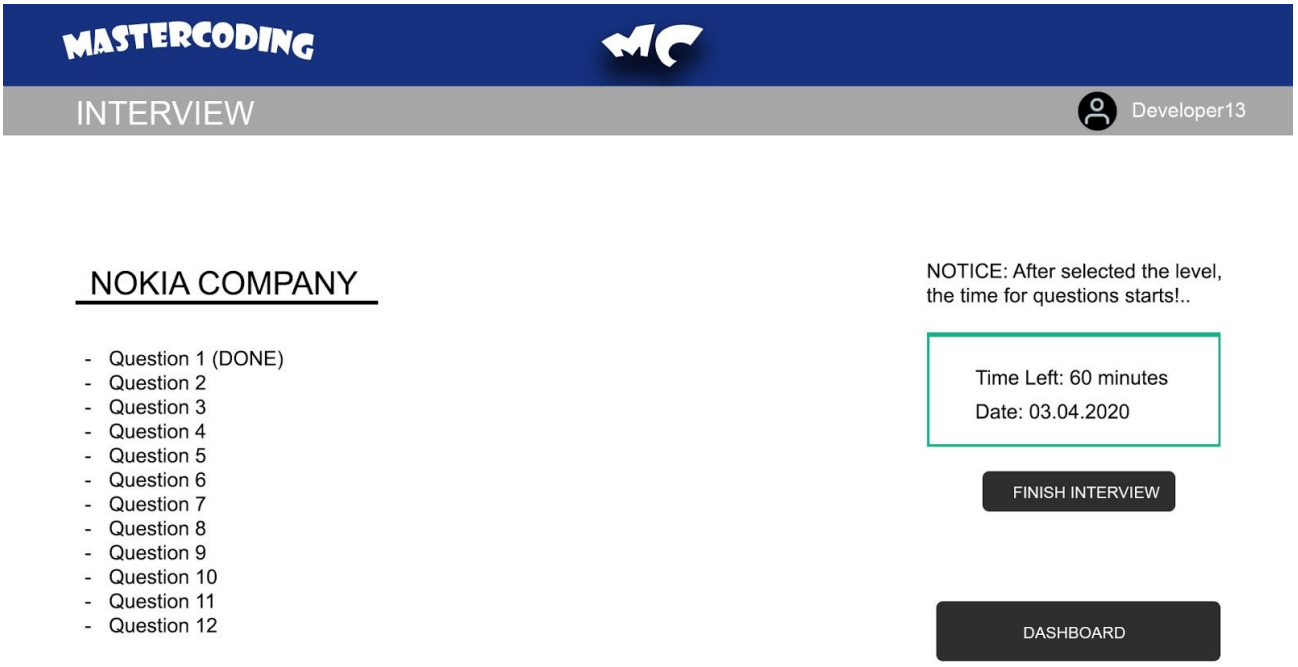


Figure 17: Interviews List

This screen shows the process of the user in interview user has a time limit and can see their whole process through this screen. By clicking finish interview user can end the interview or the user can return to dashboard by clicking on dashboard button.

3.13. Company Interview Requests

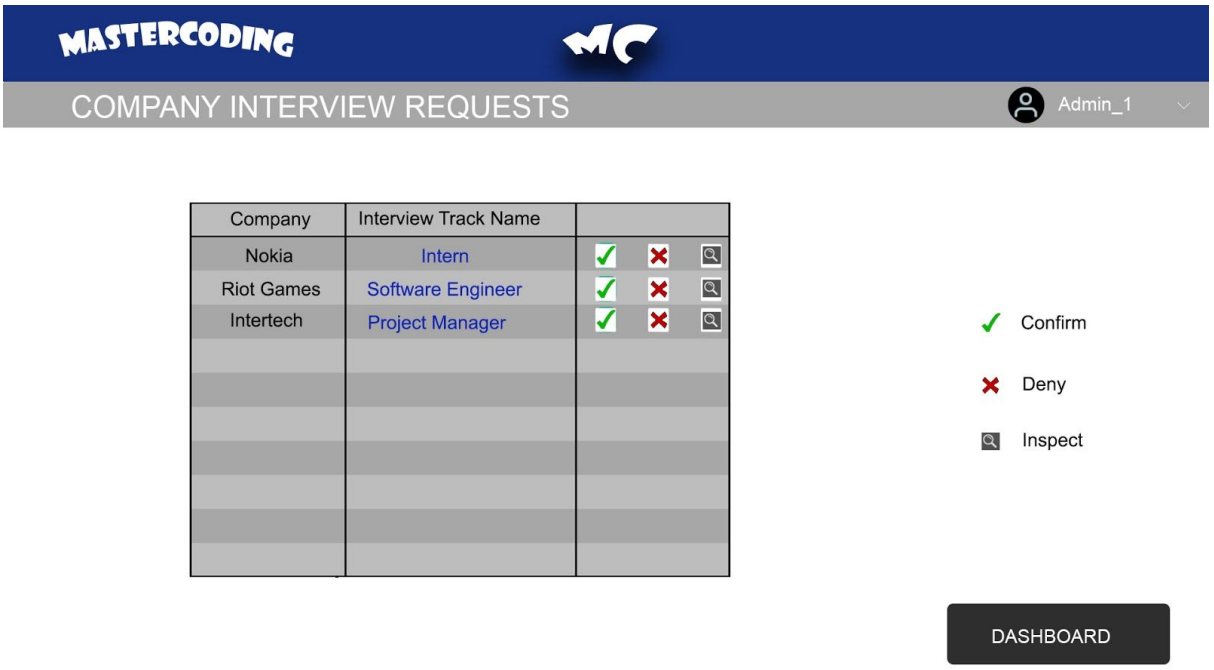


Figure 18: Company Interview Requests

In this page admin can look at interview requests and make confirm or deny.

3.14. Prepare Track Page

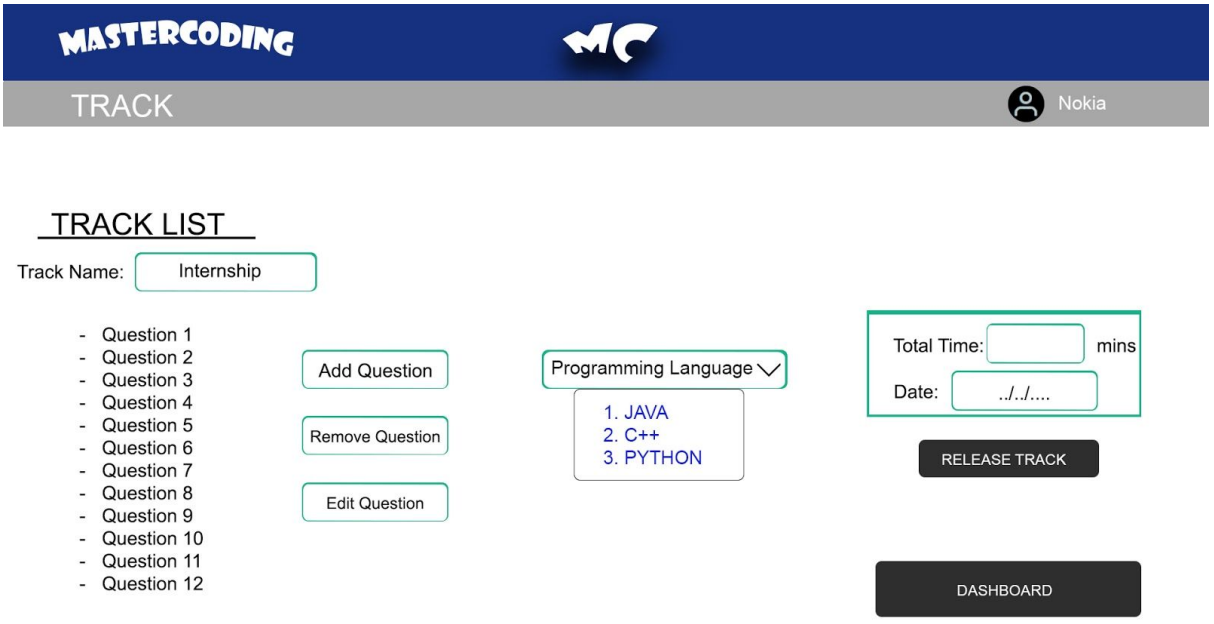


Figure 19: Prepare Track Page

This page is preparing track for companies. Companies can determine language, time, date etc. They can add or remove question as they want.

Sql Statements

INSERT INTO track(track_name) VALUES(val1)

3.15. Interview Track List for Company

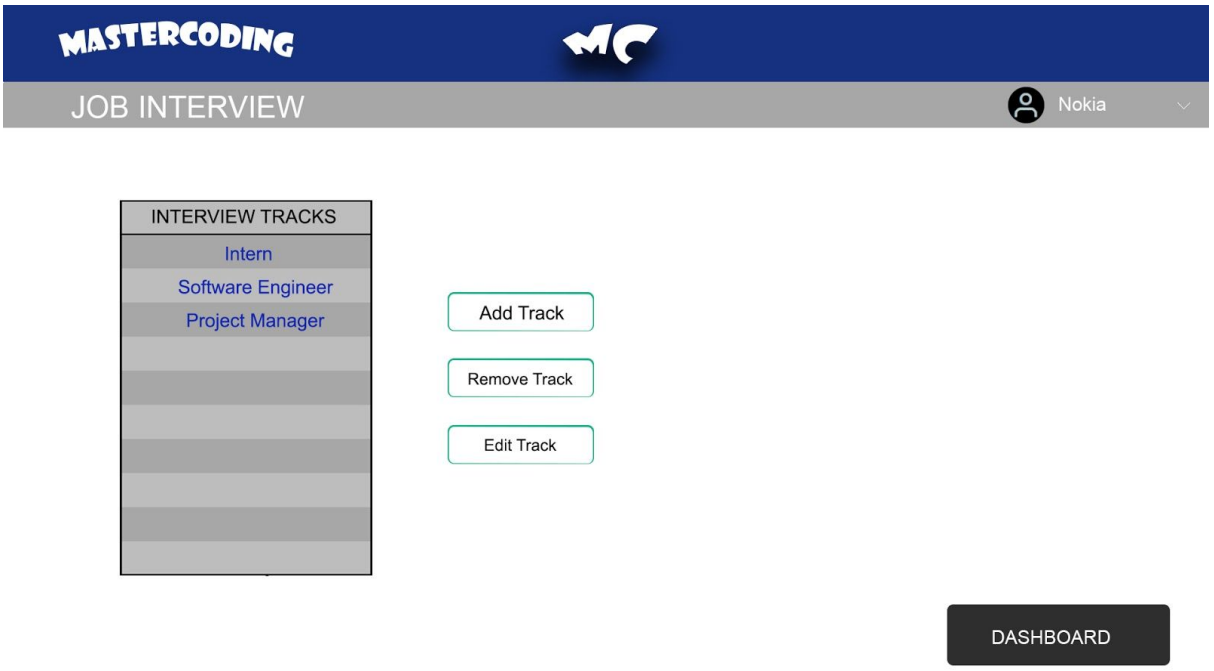


Figure 20: View Interview List

This page is viewing interview track list for companies. Companies can arrange that list.

3.16. Profile Page for Users

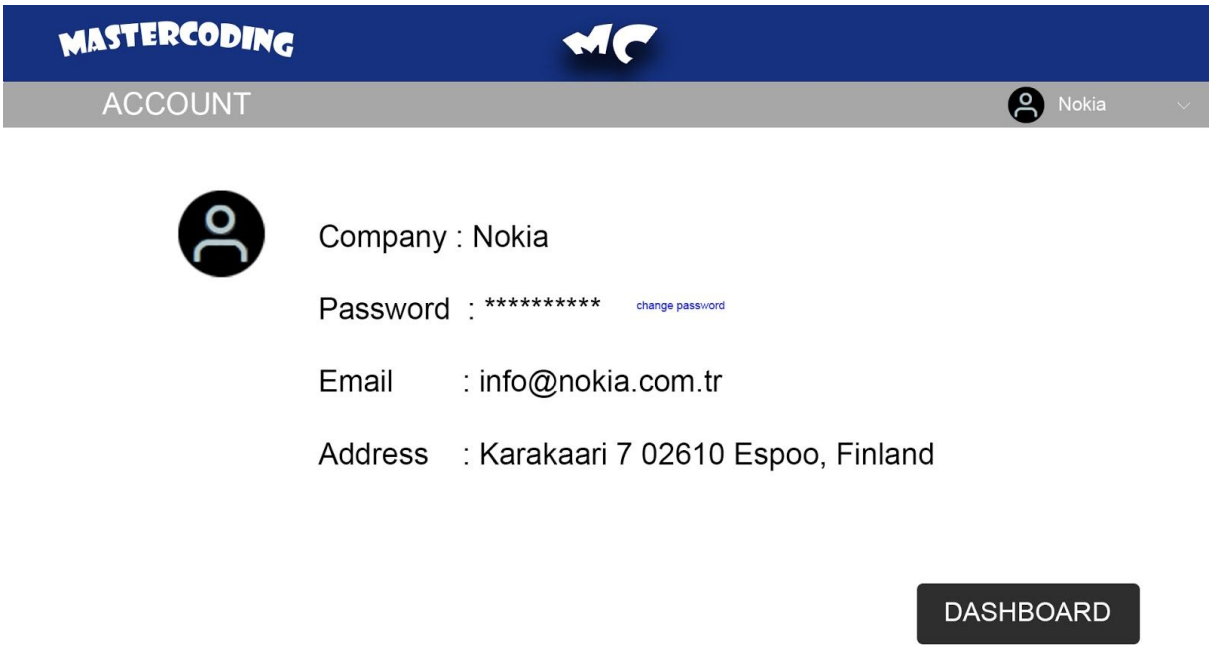


Figure 21: Company Profile Page

Users can see their profile page information.

Sql Statements

SELECT *

FROM user

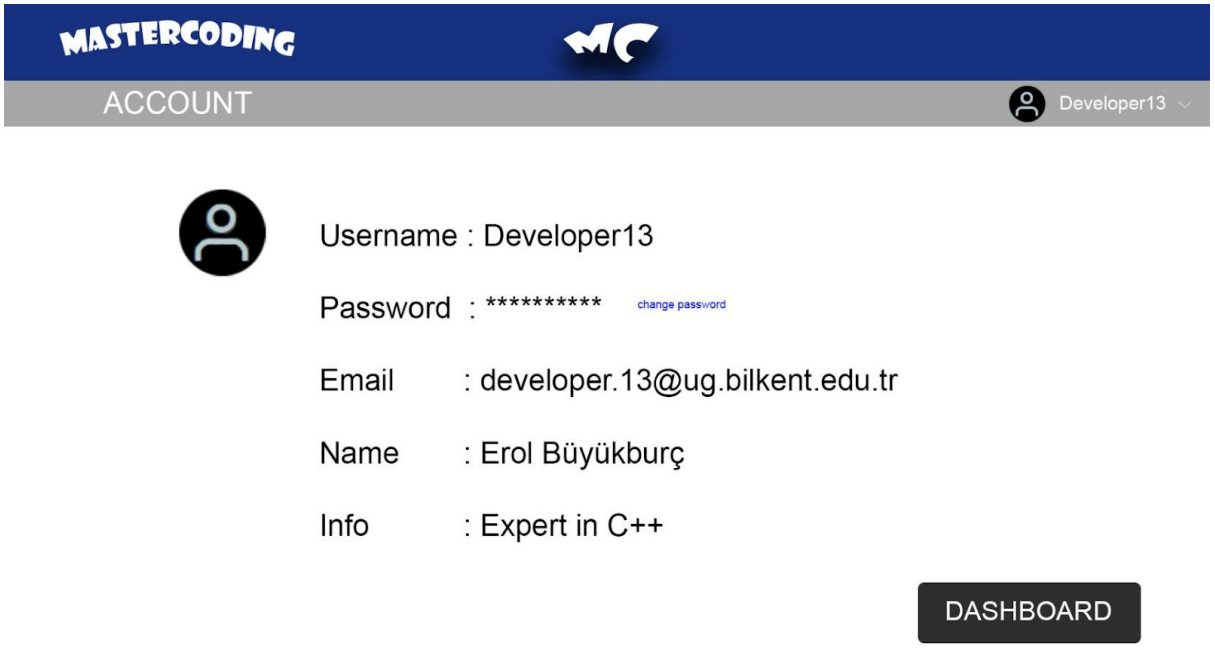


Figure 22: Developer Profile Page

Users can see their profile page or others' profile page. They can change password from here.