Bilkent University

Department of Computer Engineering

CS353

Project Design Report

Online Language Learning Platform

Date: 08.04.2022

Section: 2

Group: 32

Ata Seren 21901575

Berkan Sivrikaya 21803052

Can Avşar 21902111

Osman Semih Tiryaki 21801994

Table of Contents

Website of our project reports: https://qlanduril.github.io/cs353/

1. Revised ER Diagram

1.1 Changes

– We removed the primary key property of some attributes and even deleted some of them. In the previous diagram, each user type had a primary key. We changed them and used the "user" entity's primary key for the user types which are inherited from this entity.

– There were 2 relationships with many additional attributes, called "lectures" and "give_exercises". We deleted them and created 4 entities called "lecture_requests, lectures, exercise_requests and native_exercises". 2 of them keep the lecture requests to teachers and exercise requests to natives and other 2 keep approved and planned sessions for students.

– We destroyed the "works_on" ternary relationship and created an entity and 2 relationships instead. These are called "course","enroll" and " teach", respectively. By using the "course" entity, we created a course logic which certain teachers and natives work on a language and students can enroll to a certain coıuse of a language with a level.

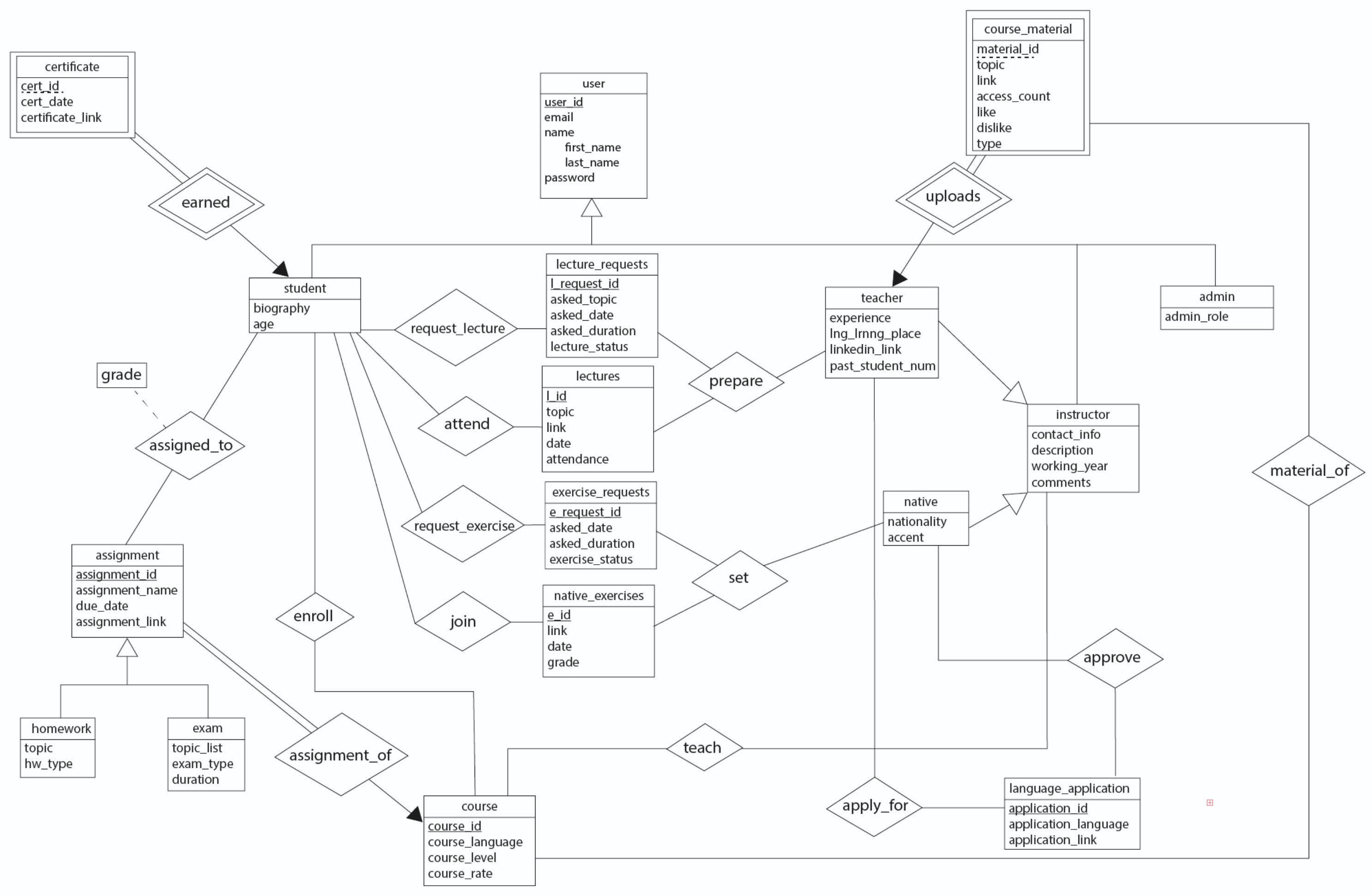
– We created a "material_of " relationship between course materials and course. With this relationship, different course materials under different courses can be displayed in a grouped way.

– We got rid of some of the total participations which causes errors in the system such as total participation of instructors to the courses. Admins will add the instructors to the system and create and save courses for them.

1.2 ER Diagram

Link of the diagram for better image quality:

https://drive.google.com/file/d/1cPIfqtcx_l9P9rGQGuhmknKGMfKDMQJW/view?usp=sharing



2. Relational Schemas

2.1 User Schema

user(user_id, email, first_name, last_name, password)

PK: user_id

FD: user_id → email, first_name, last_name, password

Normal Form: BCNF

Query:
```
CREATE TABLE user(
user_id INT AUTO_INCREMENT,
email VARCHAR(64),
first_name VARCHAR(64),
last_name VARCHAR(64),
password VARCHAR(64),
PRIMARY KEY (user_id)
);
```

2.2 Instructor Schema

instructor(user_id, contact_info, description, working_year, comments)

PK: user_id

FK: user_id references user

FD: user_id→ rate, contact_info, description, working_year, comments

Normal Form: BCNF

Query:
```
CREATE TABLE instructor(
user_id INT,
contact_info VARCHAR(64),
description VARCHAR(256),
working_year INT,
comments VARCHAR(256),
PRIMARY KEY(user_id),
FOREIGN KEY(user_id) REFERENCES user(user_id)
);
```

2.3 Teacher Schema

teacher(<u>user_id</u>, experience, lng_lrnng_place, linkedin_link, past_student_num)

PK: user_id

FK: user_id references user

FD: userID → experience, lng_lrnng_place, linkedin_link, past_student_num

Normal Form: BCNF

Query:
```
CREATE TABLE teacher(
user_id INT,
experience VARCHAR(256),
lng_lrnng_place VARCHAR(64),
linkedin_link VARCHAR(128),
past_student_num INT,
PRIMARY KEY(user_id),
FOREIGN KEY(user_id) REFERENCES user(user_id)
);
```

2.4 Native Schema

native(<u>user_id</u>, nationality, accent)

PK: user_id

FK: user_id references user

FD: user_id → nationality, accent

Normal Form: BCNF

Query:
```
CREATE TABLE native(
user_id INT,
nationality VARCHAR(32),
accent VARCHAR(32),
PRIMARY KEY(user_id),
FOREIGN KEY(user_id) REFERENCES user(user_id)
);
```

2.5 Lecture Request Schema

lecture_requests(<u>l_request_id</u>, asked_topic, asked_date, asked_duration, lecture_status)

PK: l_request_id

FD: l_request_id → asked_topic, asked_date, asked_duration, lecture_status

Normal Form: BCNF

Query:
```
CREATE TABLE lecture_requests(
    l_request_id INT AUTO_INCREMENT,
    asked_topic VARCHAR(64),
    asked_date DATE,
    asked_duration INT,
    lecture_status BOOLEAN,
    PRIMARY KEY(l_request_id),
    CHECK (asked_duration BETWEEN 15 AND 120)
    );
```

2.6 Lectures Schema

lectures(<u>l_id,</u> topic, link, date, attendance)

PK: l_id

FD: l_id → topic, link, date, attendance

Normal Form: BCNF

Query:
```
CREATE TABLE lectures(
```

l_id INT AUTO_INCREMENT,

topic VARCHAR(64),

link VARCHAR(256),

date DATE,

attendance BOOLEAN,

PRIMARY KEY(l_id)

);

2.7 Exercise Request Schema

exercise_requests(<u>e_request_id</u>, asked_date, asked_duration, exercise_status)

PK: e_request_id

FD: e_request_id → asked_date, asked_duration, exercise_status

Normal Form: BCNF

Query: CREATE TABLE exercise_requests(

e_request_id INT AUTO_INCREMENT,

asked_date DATE,

asked_duration INT,

exercise_status BOOLEAN,

PRIMARY KEY(e_request_id),

CHECK (asked_duration BETWEEN 15 AND 120)

);

2.8 Native Exercises Schema

native_exercises(<u>e_id</u>, link, date, grade)

PK: e_id

FD: e_id → link, date, grade

Normal Form: BCNF

```
Query:  CREATE TABLE native_exercises (
        e_id INT AUTO_INCREMENT,
        link VARCHAR(256),
        date DATE,
        grade INT,
        CHECK (grade BETWEEN 0 AND 100),
        PRIMARY KEY(e_id)
        );
```

2.9 Set Schema

set(<u>user_id, e_request_id, e_id</u>)

PK: {user_id, e_request_id, e_id}

FK: user_id references native

e_request_id references exercise_requests

e_id references exercises

FD: There are no non-trivial functional dependencies

```
Query: CREATE TABLE set(
        user_id INT,
        e_request_id INT,
        e_id INT,
        PRIMARY KEY(user_id, e_request_id, e_id),
        FOREIGN KEY(user_id) REFERENCES native(user_id),
        FOREIGN KEY(e_request_id) REFERENCES exercise_requests (e_request_id),
        FOREIGN KEY(e_id) REFERENCES native_exercises (e_id)
        );
```

2.10 Homework Schema

homework(<u>assignment_id</u>, topic, hw_type)

PK: assignment_id

FK: assignment_id references assignment

Normal Form: BCNF

FD: assignment_id → topic, hw_type

Query:
```
CREATE TABLE homework(
        assignment_id INT,
        topic VARCHAR(256),
        hw_type  VARCHAR(256),
        PRIMARY KEY(assignment_id),
        FOREIGN KEY(assignment_id) REFERENCES assignment(assignment_id)
        );
```

2.11 Exam Schema

exam(<u>assignment_id</u>, topic_list, exam_type, duration)

PK: assignment_id

FK: assignment_id references assignment

Normal Form: BCNF

FD: assignment_id → topic_list, exam_type, duration

Query:
```
CREATE TABLE exam(
        assignment_id INT,
        topic VARCHAR(256),
        exam_type  VARCHAR(256),
        topic_list VARCHAR(256),
        PRIMARY KEY(assignment_id),
        FOREIGN KEY(assignment_id) REFERENCES assignment(assignment_id)
);
```

2.12 Assignment of Schema

assignment_of: table is not needed because of one-to-many redundancy

2.13 Uploads Schema

- The uploads table is not include in relational schema and SQL because it was an one to many relationship which means uploads table would be redundant

2.14 Course Schema

course(course_id, course_language, course_level, course_rate)

PK: course_id

FD: course_id → course_language, course_level, course_rate

Normal form: BCNF

```
Query: CREATE TABLE course(
        course_id INT AUTO_INCREMENT,
        course_language VARCHAR(32),
        course_level VARCHAR(2),
        course_rate INT,
        CHECK (course_rate  BETWEEN 1 AND 5),
        PRIMARY KEY(course_id)
        );
```

2.15 Enroll Schema

enroll(user_id, course_id)

FK: user_id references student

course_id references course

Normal form: BCNF

FD: There are no non-trivial dependencies.

```
Query: CREATE TABLE enroll (
       user_id INT,
       course_id INT,
       PRIMARY KEY(user_id, course_id),
       FOREIGN KEY(user_id) references student(user_id),
       FOREIGN KEY(course_id) references course(course_id)
       );
```

2.16 Teach Schema

teach(user_id, course_id)

FK: user_id references instructor

course_id references course

Normal form: BCNF

FD: There are no non-trivial dependencies.

```
Query: CREATE TABLE enroll (
       user_id INT,
       course_id INT,
       PRIMARY KEY(user_id, course_id),
       FOREIGN KEY(user_id) references instructor(user_id),
       FOREIGN KEY(course_id) references course(course_id)
       );
```

2.17 Apply For Schema

apply_for(user_id, application_id)

PK: {user_id, application_id}

FK: user_id references teacher

application_id references language_application

Normal form: BCNF

FD: There are no non-trivial functional dependencies

Query: CREATE TABLE apply_for(

```
user_id INT,
application_id INT,
PRIMARY KEY(user_id, application_id),
FOREIGN KEY(user_id) references teacher(user_id ),
FOREIGN KEY(application_id) references
language_application(application_id  )
);
```

2.18 Language Application Schema

language_application(application_id, application_language, application_link)

PK: application_id

Normal form: BCNF

FD: application_id → application_language, application_link

Query: CREATE TABLE language_application(

```
application_id INT AUTO_INCREMENT,
application_language VARCHAR(32),
application_link VARCHAR(256),
PRIMARY KEY(application_id)
);
```

2.19 Approve Schema

approve(user_id, application_id)

PK: {user_id, application_id}

FK:	user_id references native

	application_id references language_application

Normal form: BCNF

FD: There are no non-trivial functional dependencies

Query: CREATE TABLE approve(

```
user_id INT,
application_id  INT,
PRIMARY KEY(user_id, application_id),
FOREGN KEY(user_id) references native(user_id),
FOREGN KEY(application_id) references language_application(application_id)
);
```

2.20 Request Lecture Schema

request_lecture(<u>user_id</u> , l<u>_request_id</u>)

PK:	{ user_id , l_request_id }

FK:	user_id references student

	l_request_id references lecture_requests

FD: There are no non-trivial functional dependencies

Query: CREATE TABLE request_lecture(

```
user_id INT,
l_request_id INT,
PRIMARY KEY (user_id, l_request_id),
FOREIGN KEY(user_id) REFERENCES student(user_id),
FOREIGN KEY(l_request_id) REFERENCES lecture_requests(l_request_id)
);
```

2.21 Attend Schema

attend(<u>user_id</u> , <u>l_id</u>)

PK: { user_id , l_id }

FK: user_id references student

l_request_id references lectures

FD: There are no non-trivial functional dependencies

Query: CREATE TABLE attend(

```
user_id INT,
l_request_id INT,
PRIMARY KEY (user_id, l_id),
FOREIGN KEY(user_id) REFERENCES student(user_id),
FOREIGN KEY(l_id) REFERENCES lectures(l_id)
);
```

2.22 Request Exercise Schema

request_excersise(<u>user_id</u> , <u>e_request_id</u>)

PK: { user_id , e_request_id }

FK: user_id references student

e_request_id references exercise_requests

FD: There are no non-trivial functional dependencies

Query: CREATE TABLE request_excercise(

```
user_id INT,
e_request_id INT,
PRIMARY KEY (user_id, e_request_id),
FOREIGN KEY(user_id) REFERENCES student(user_id),
FOREIGN KEY(e_request_id) REFERENCES exercise_requests(e_request_id)
);
```

2.23 Join Schema

join(user_id , e_id)

PK: {user_id,e_id}

FK: user_id references student

e_id references native_excercises

FD: There are no non-trivial functional dependencies

Query: CREATE TABLE join(

```
user_id INT,
e_id INT,
PRIMARY KEY (user_id, e_id),
FOREIGN KEY(user_id) REFERENCES student(user_id),
FOREIGN KEY(e_id) REFERENCES native_requests(e_id)
);
```

2.24 Student Schema

student(user_id, biography, age)

PK: user_id

FK: user_id references user

FD: user_id → biography, age

Normal Form: BCNF

Query: CREATE TABLE student(

```
user_id INT ,
biography VARCHAR(256),
age INT,
PRIMARY KEY(user_id),
FOREIGN KEY(user_id) REFERENCES user(user_id),
```

);

2.25 Assigned To Schema

assigned_to(user_id, assignment_id, grade)

PK: {user_id, assignment_id}

FK: user_id references student

assignment_id references assignment

FD: user_id, assignment_id → grade

Normal Form: BCNF

Query: CREATE TABLE assigned_to(

user_id INT,

assignment_id INT,

grade INT

PRIMARY KEY (user_id, assignment_id)

FOREIGN KEY(user_id) REFERENCES student(user_id),

FOREIGN KEY(assignment_id) REFERENCES assignment(assignment_id)

CHECK (grade BETWEEN 0 AND 100),

);

2.26 Assignment Schema

assignment(assignment_id, assignment_name , due_date, assignment_link, course_id)

PK: assignment_id

FK: course_id references course

FD: assignment_id → assignment_name, due_date, assignment_link, course_id

Normal Form: BCNF

Query: CREATE TABLE assignment(

assignment_id INT AUTO_INCREMENT,

```
        assignment_name VARCHAR(64),
        due_date DATE,
        assignment_link VARCHAR(256),
        PRIMARY KEY (assignment_id),
        FOREIGN KEY (course_id) REFERENCES course(course_id)
        );
```

2.27 Certificate Schema

certificate(cert_id, cert_date, certificate_link, user_id)

PK: {cert_id, user_id}

FK: user_id references student

FD: cert_id → cert_date, certificate_link, user_id

Normal Form: BCNF

Query:
```
CREATE TABLE certificate(
        cert_id INT AUTO_INCREMENT,
        cert_date DATE,
        certificate_link VARCHAR(64),
        user_id INT,
        PRIMARY KEY (cert_id, user_id ),
        FOREIGN KEY (user_id ) REFERENCES student(user_id )
        );
```

2.28 Prepare Schema

prepare(user_id, l_request_id, l_id)

PK: {user_id, l_request_id, l_id}

FK: user_id references teacher

l_request_id references lecture_requests l_id references lectures

FD: There are no non-trivial functional dependencies

Query:
```
CREATE TABLE prepare(
        user_id INT,
        l_request_id INT,
        l_id INT,
        PRIMARY KEY(user_id, l_request_id, l_id),
        FOREIGN KEY(user_id) REFERENCES teacher(user_id),
        FOREIGN KEY(l_request_id) REFERENCES lecture_requests(l_request_id),
        FOREIGN KEY(l_id) REFERENCES lectures(l_id)
        );
```

2.29 Course Material Schema

course_material(material_id, topic, link, type, access_count, like, dislike, user_id)

PK: {material_id,user_id}

FK: user_id references teacher

FD: material_id → topic, link, type, access_count, like, dislike, user_id

Normal Form: BCNF

Query:
```
CREATE TABLE course_material(
        material_id INT AUTO_INCREMENT,
        topic VARCHAR(64),
        type  VARCHAR(16),
        link VARCHAR(64),
        access_count INT DEFAULT 0,
        like INT DEFAULT 0,
        dislike INT DEFAULT 0,
        user_id INT,
```

PRIMARY KEY(material_id, user_id),

FOREIGN KEY(user_id) REFERENCES teacher(user_id)

);

2.30 Admin Schema

admin(user_id, admin_role)

PK: user_id

FK: user_id references user

FD: user_id -> admin_role

Normal Form: BCNF

Query:
```
CREATE TABLE admin(
user_id INT,
admin_role VARCHAR(64) ,
PRIMARY KEY(user_id),
FOREIGN KEY(user_id) REFERENCES user(user_id)
);
```

2.31 Material Of Schema

material_of(course_id, material_id)

PK: { course_id, material_id }

FK: course_id references course

material_id references course_material

Normal Form: BCNF

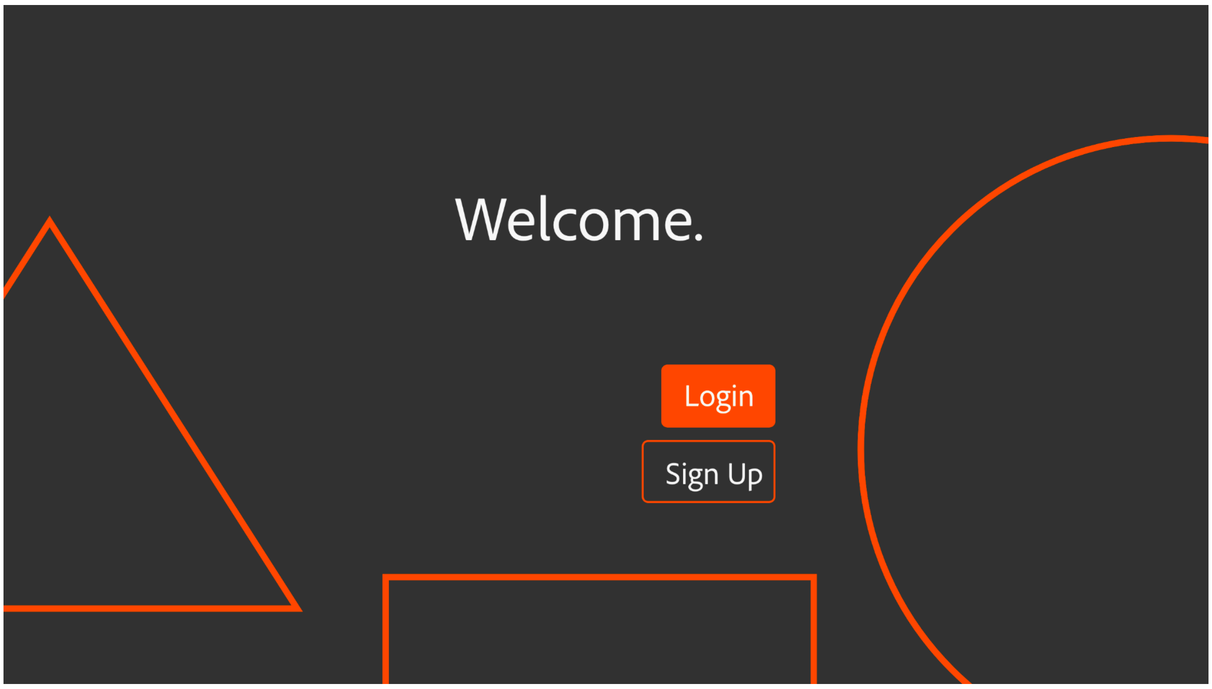
FD: There are no non-trivial functional dependencies.

Query:
```
CREATE TABLE material_of(
course_id INT,
material_id INT,
PRIMARY KEY (course_id , material_id),
FOREIGN KEY(course_id) REFERENCES course(course_id),
FOREIGN KEY(material_id) REFERENCES course_material(material_id),
);
```

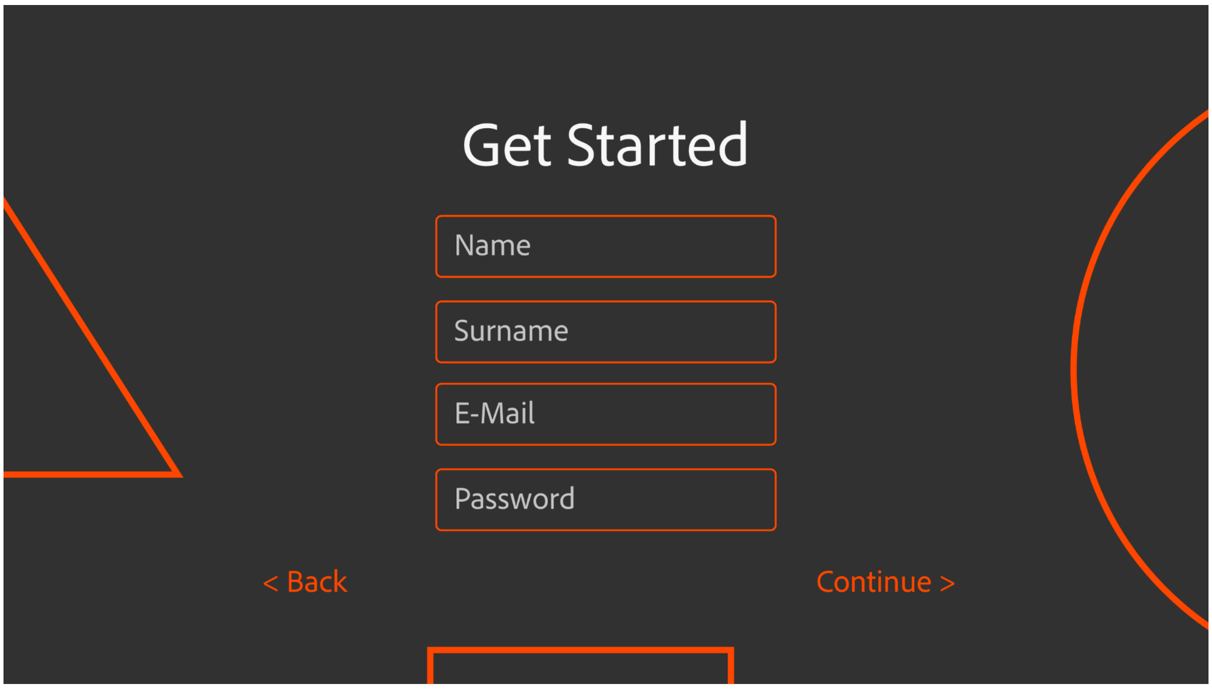
3. User Interface Design and SQL Queries

3.1 Login & Register

3.1.1 Register



This is a welcome page to greet users. No SQL statement is needed for this page



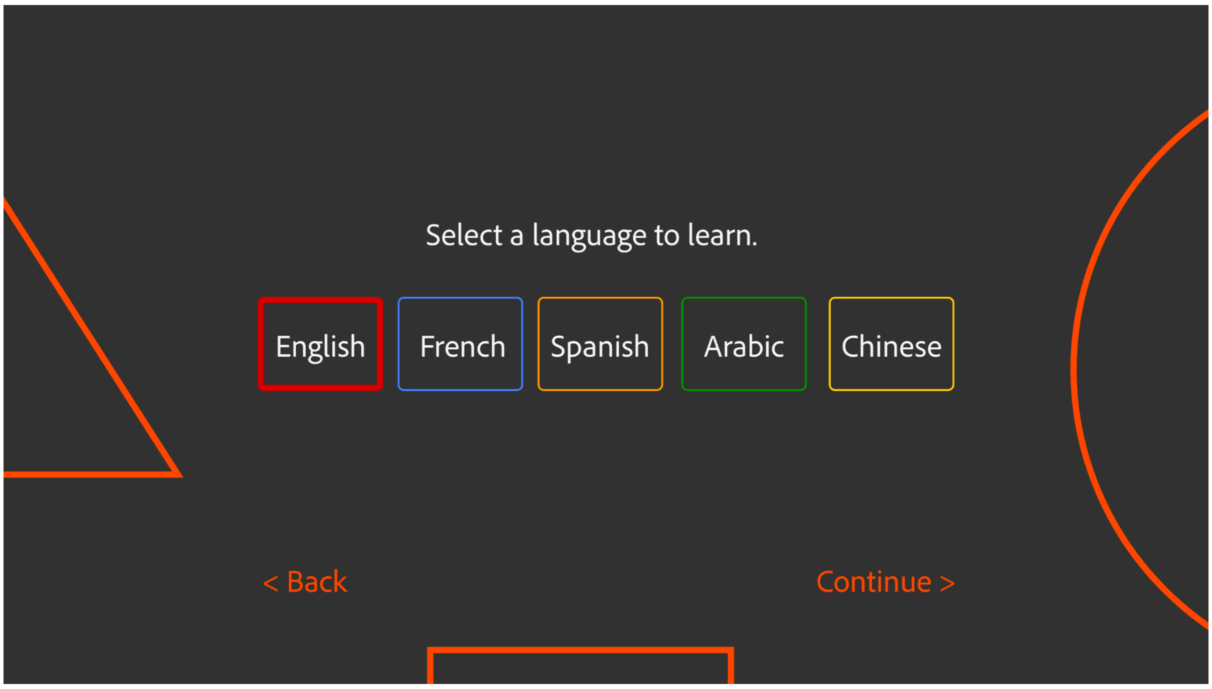
This is a sign up page where the user can enter his/her information to be enrolled in the system. After this step the user is saved into the users table in the database.

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

SQL Query to Sign Up a User:

INSERT INTO user(email, first_name, last_name, password)

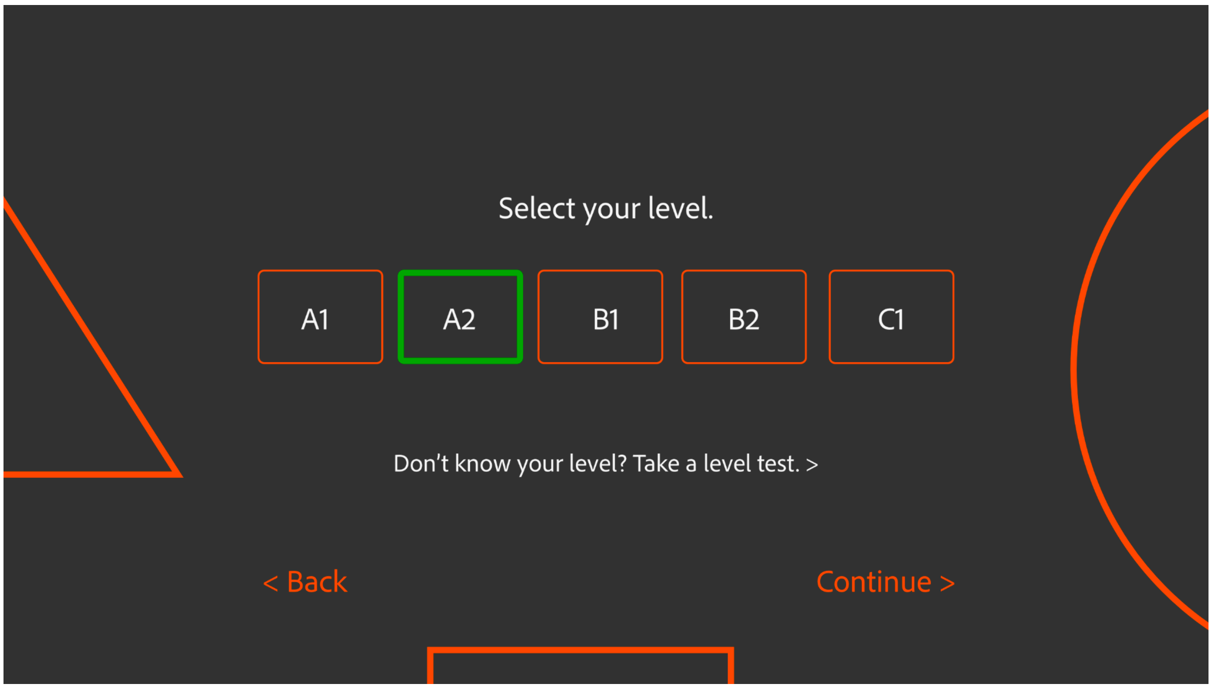
VALUES (@email, @first_name, @last_name, @password);



When a recently enrolled user enters the system, the user is guided to pick his first language to learn.

SQL Query to Display Language List:

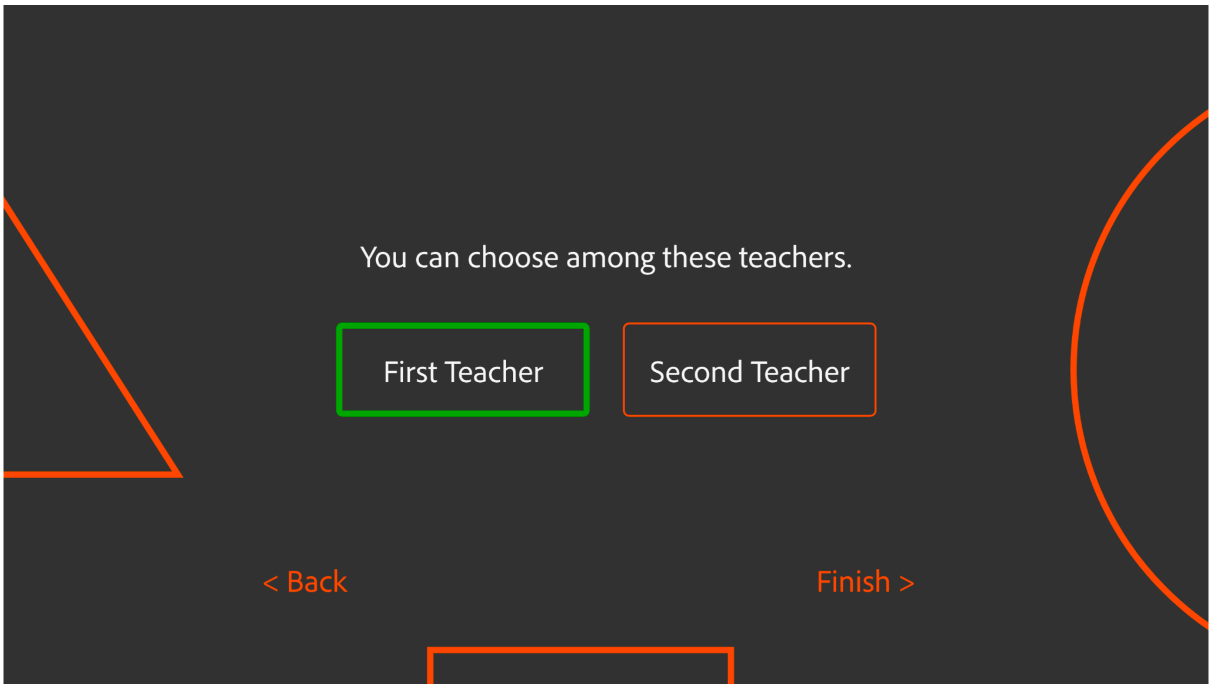
```
SELECT DISTINCT course_language

FROM course;
```



User proceeds by choosing the current level in that language.

The levels will always be the same for all languages so this selection will be handled in the frontend with buttons.

No SQL queries needed.



After that, all teachers teaching the language in that level will be displayed.

Inputs: @course_language, @course_level (all carried from sessions)

SQL Query to Show All Teachers:

```
SELECT DISTINCT first_name, last_name
FROM course NATURAL JOIN teach
WHERE course_id IN
        (SELECT course_id
        FROM course
        WHERE course_language = @course_language
        AND course_level = @course_level)
```

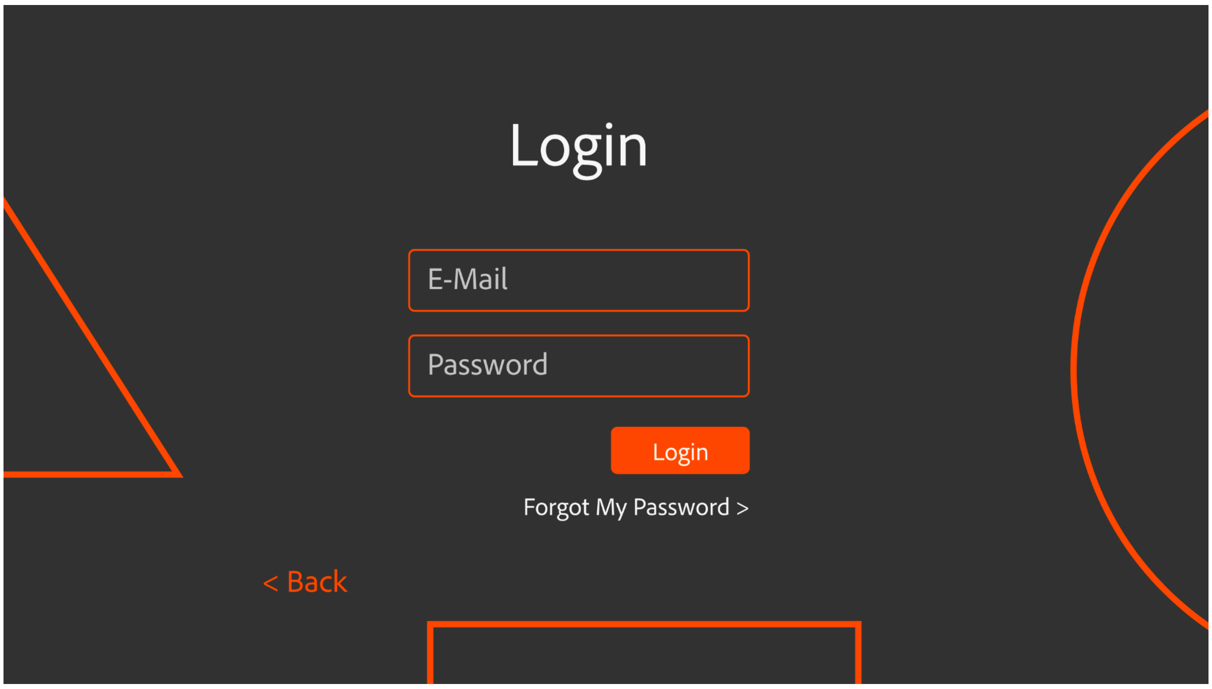
Finally when all information is obtained the student will be enrolled in that course.

Inputs: @user_id, @course_id (all carried from sessions) (user_id from enroll)

SQL Query to Enroll in a Course:

```
INSERT INTO enroll(student_id, course_id)
VALUES ( @user_id, @course_id)
```

3.1.2 Login



After registration, users can login their account whenever they want with their credentials.

Inputs: @email, @password

SQL Query to Login:

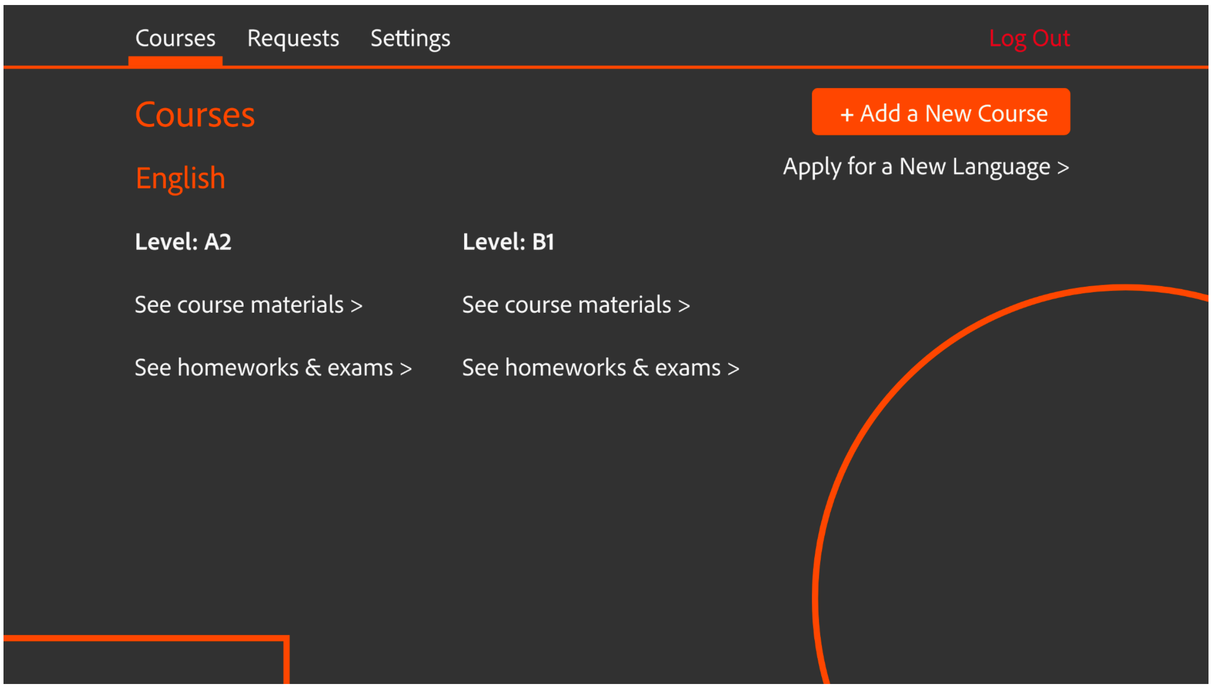
SELECT email, password

FROM user

WHERE email = @email AND password = @password;

3.2 Accessing Course Materials

3.2.1 Uploading Course Material



In this page, courses that an instructor gives are displayed. In this scenario, an instructor can upload a material for the access of students of a certain course.

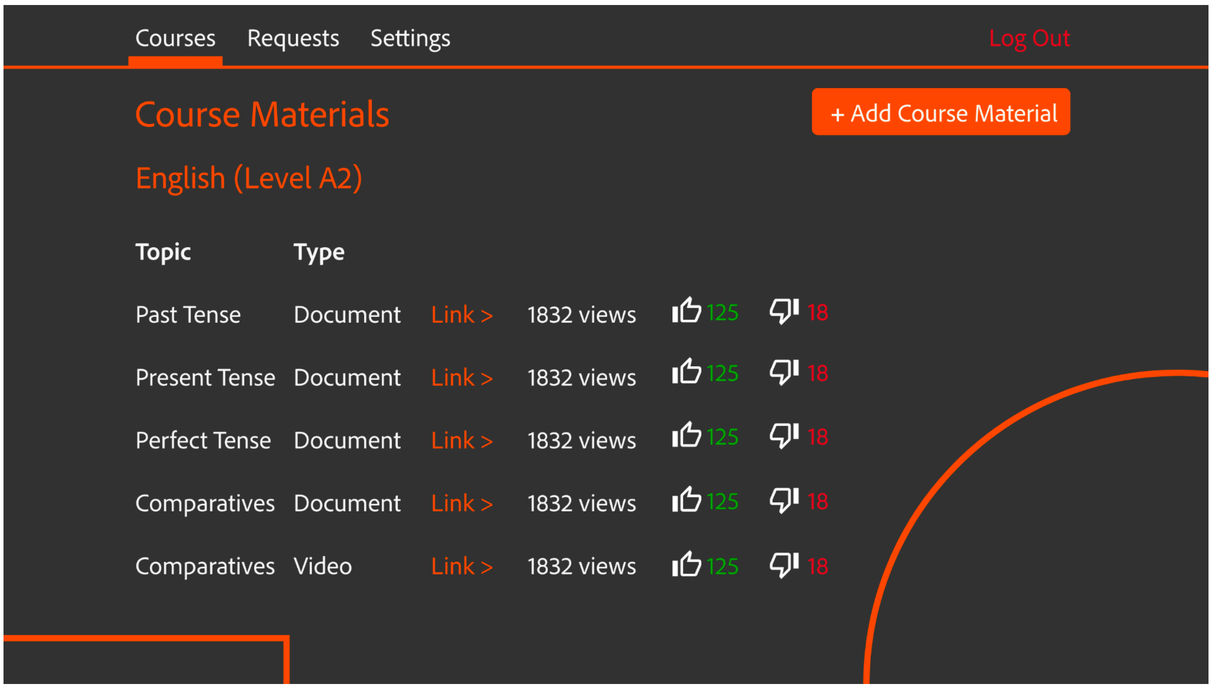
Inputs: @user_id (carried from session)

SQL Query to Display Course List to the Instructor:

SELECT course_language, course_level

FROM course NATURAL JOIN teach

WHERE user_id = @user_id;



Course Materials

English (Level A2)

Topic	Type				
Past Tense	Document	Link >	1832 views	125	18
Present Tense	Document	Link >	1832 views	125	18
Perfect Tense	Document	Link >	1832 views	125	18
Comparatives	Document	Link >	1832 views	125	18
Comparatives	Video	Link >	1832 views	125	18

In this page, course materials uploaded by the instructor for a certain course are displayed. Instructors can see different values about their uploaded materials and by using the "Add Course Material" button, he or she can upload a new material for a certain course.

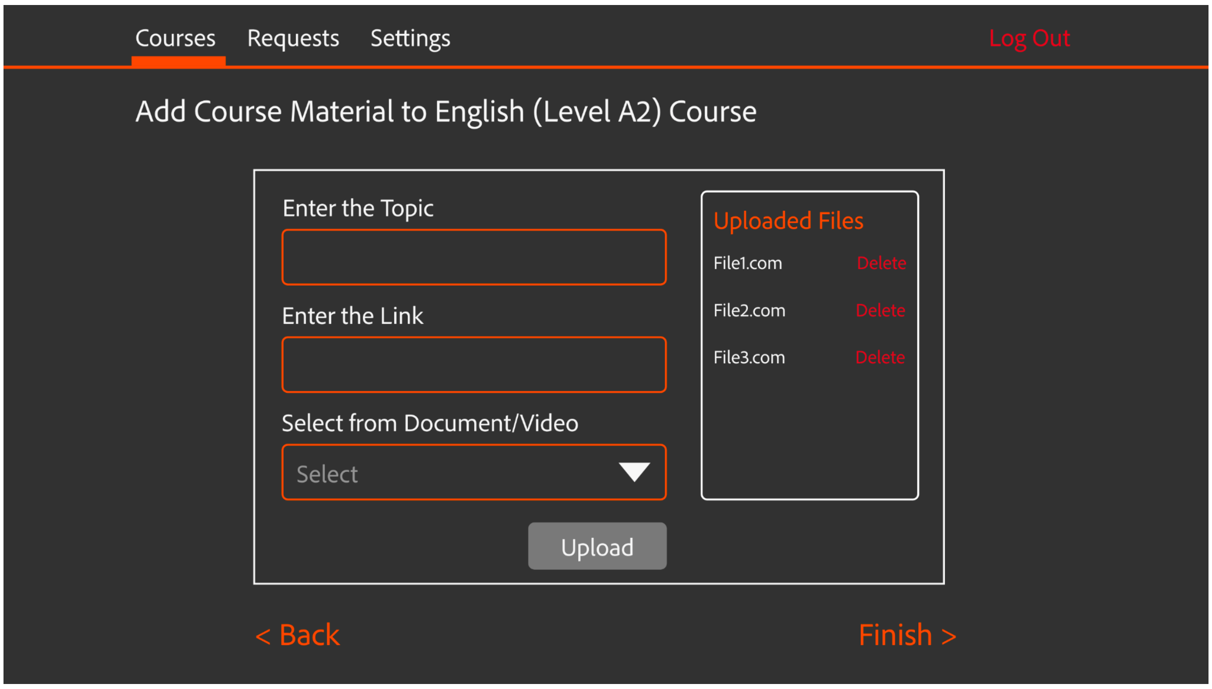
Inputs: @course_id (carried from session)

SQL Query to Display Course Material List to the Instructor:

SELECT topic, type, link, like, dislike, access_count

FROM course_material NATURAL JOIN material_of

WHERE course_id = @course_id;



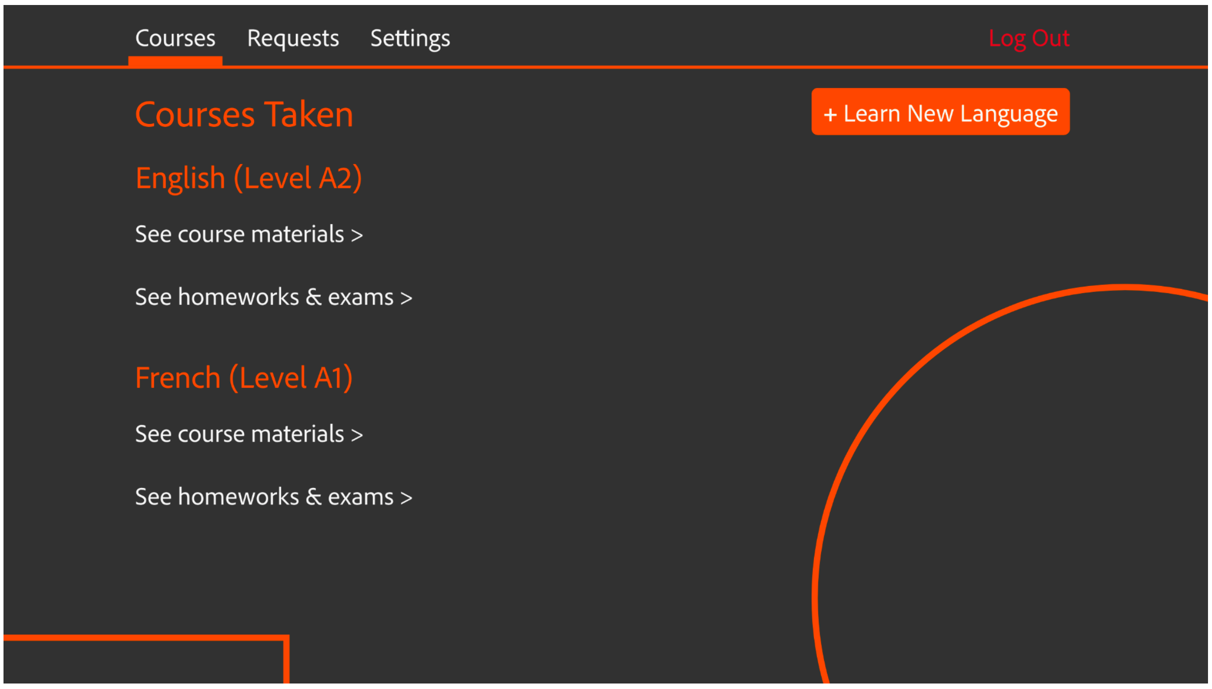
In this page, an instructor can add a material with a topic, access link and type of the material. Students can access this material after the instructor uploads.

Inputs: @topic, @link, @type

SQL Query to Add Course Material:

```
INSERT INTO course_material(topic, link, type)

VALUES(@topic, @link, @type);
```

3.2.2 Accessing Course Material

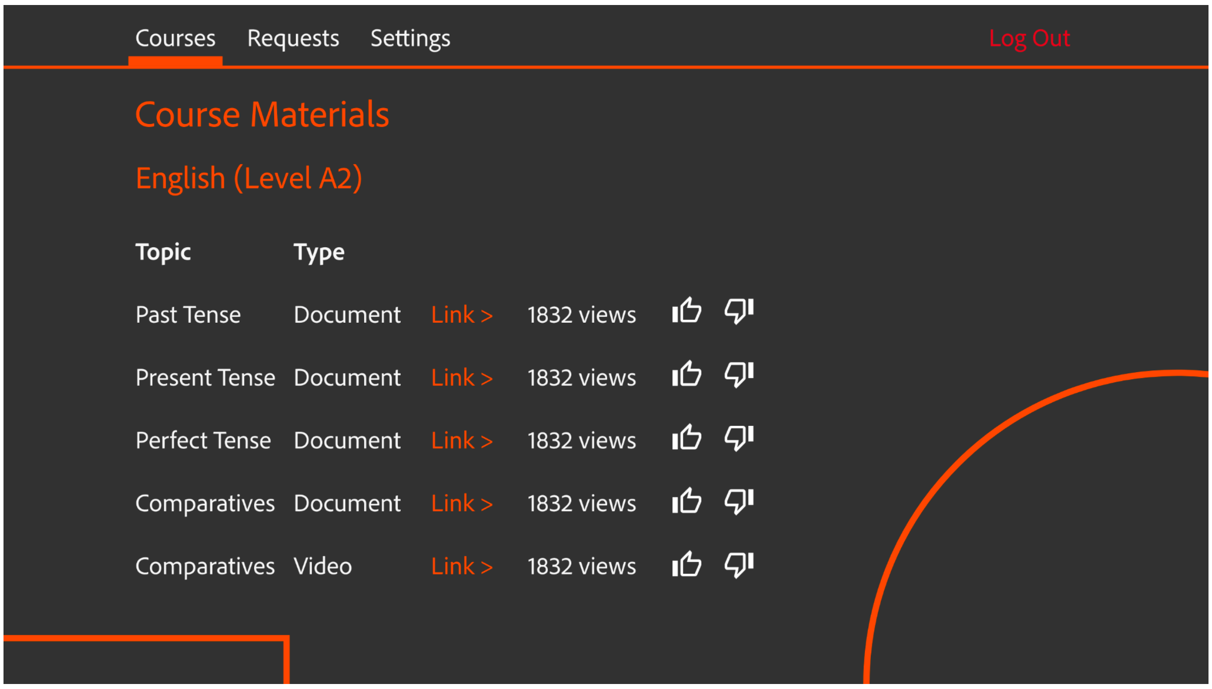


In this page, courses taken by a student are displayed. A student can access course materials, homeworks and exams through this page. Also a new course can be added from this page too.

Inputs: @user_id (carried from session)

SQL Query to Display Course List to the Student:

```
SELECT course_id, course_language, course_level

FROM course NATURAL JOIN enroll

WHERE user_id = @user_id;
```



Course Materials

English (Level A2)

Topic	Type				
Past Tense	Document	Link >	1832 views	👍	👎
Present Tense	Document	Link >	1832 views	👍	👎
Perfect Tense	Document	Link >	1832 views	👍	👎
Comparatives	Document	Link >	1832 views	👍	👎
Comparatives	Video	Link >	1832 views	👍	👎

In this page, a student can access the course materials of a certain course uploaded by the teacher of that course. Students can access the material and give a like or a dislike which is visible to the teacher.

Inputs: @course_id (carried from session)

SQL Query to Display Course Material List to the Student:

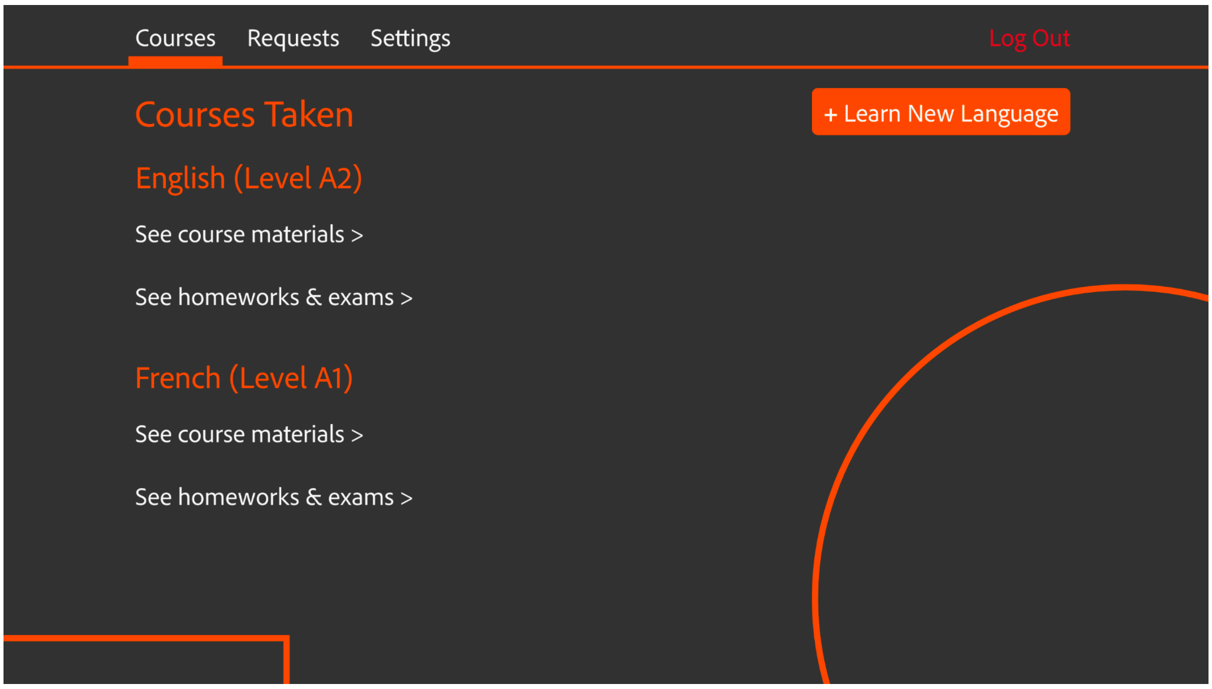
SELECT topic, type, link, like, dislike, access_count

FROM course_material NATURAL JOIN material_of NATURAL JOIN course

WHERE course_id = @course_id

3.3 Taking a Language Class

3.3.1 Select a Language and Level



Courses Taken

English (Level A2)

See course materials >

See homeworks & exams >

French (Level A1)

See course materials >

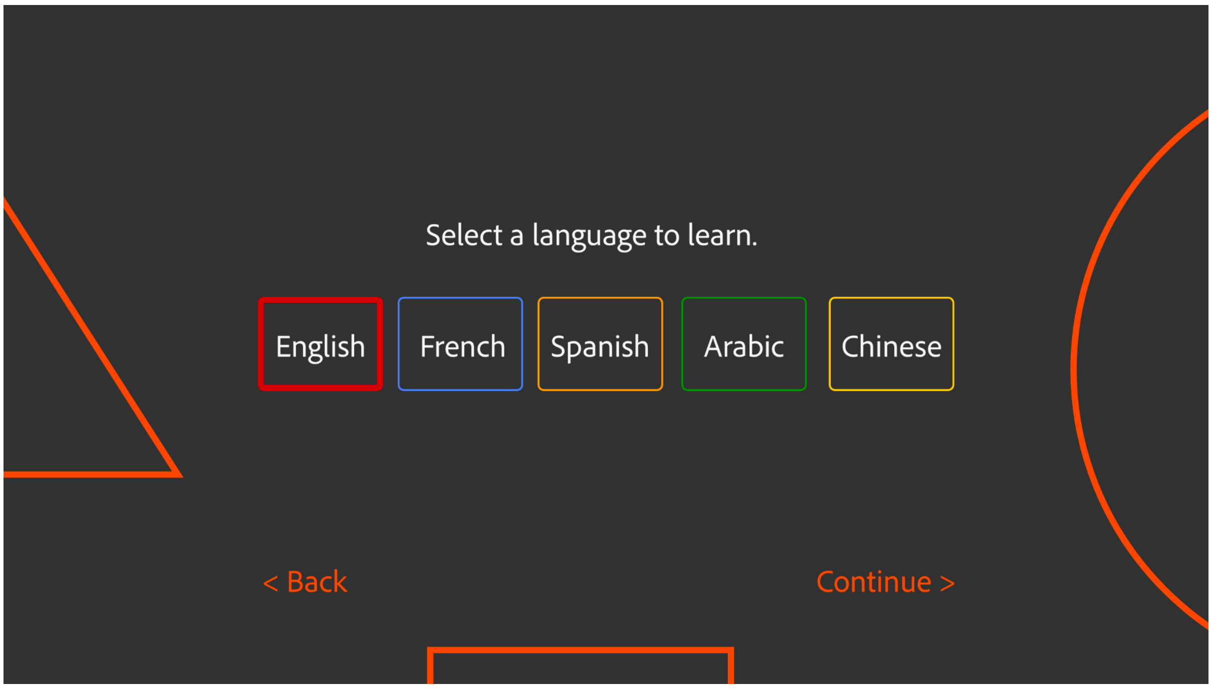
See homeworks & exams >

By pressing the learn new language button, the users can learn extra languages. This button is located inside my courses page so already taken languages will be displayed here.

Inputs: @user_id (carried from session)

SQL Statement to Show Already Taken Courses and Their Levels:

```
SELECT course_id, course_language, course_level

FROM course NATURAL JOIN enroll

WHERE user_id = @user_id;
```

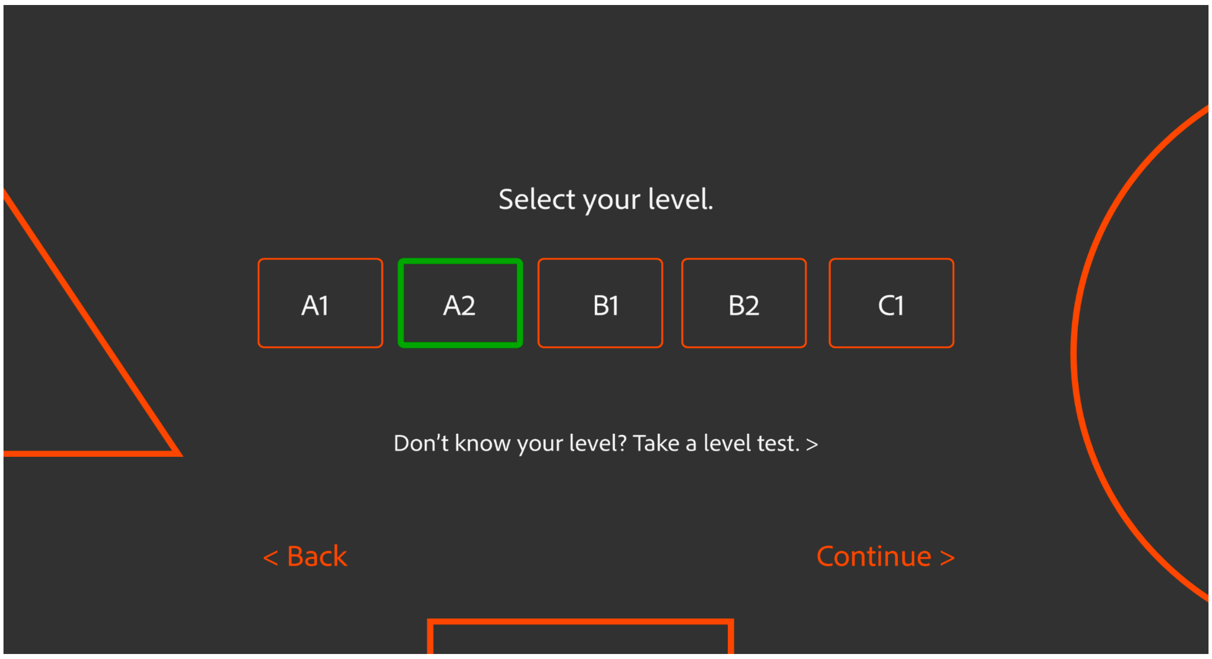


After pressing the learn a new language button, the languages that have not been currently taken by the user will be displayed.

Inputs: @user_id (carried from session)

SQL Statement to Display Languages that User is not Currently Taking.

```
SELECT DISTINCT course_language

FROM course NATURAL JOIN enroll

WHERE course_id NOT IN (SELECT course_id

                        FROM enroll

                        WHERE user_id = @user_id);
```

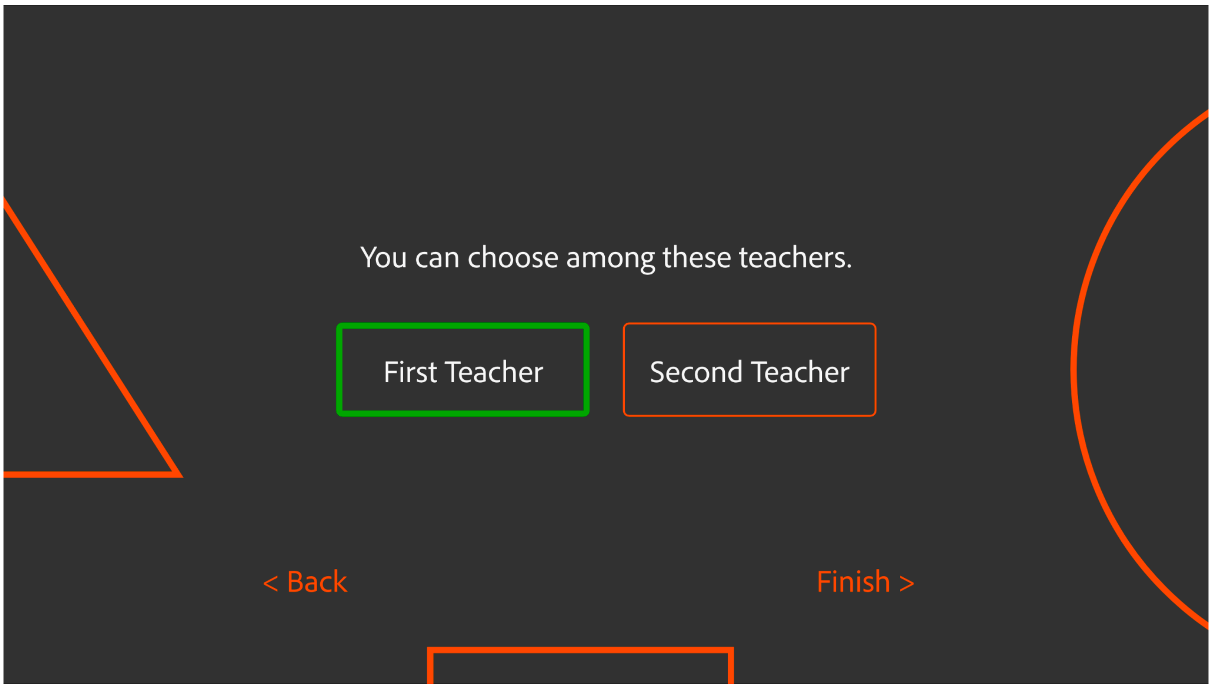


User proceeds by choosing the current level in that language.

The levels will always be the same for all languages so this selection will be handled in the frontend with buttons.

No SQL queries needed.

3.3.2 Select a Teacher



After selecting language and level, user needs to choose from available teachers.

Inputs: @course_language, @course_level (all carried from sessions)

SQL Query to Choose From Available Teachers Teaching a Particular Language Level:

```
SELECT DISTINCT first_name, last_name
FROM course NATURAL JOIN teach
WHERE course_id IN
        (SELECT course_id
        FROM course
        WHERE course_language = @course_language
        AND course_level = @course_level)
```

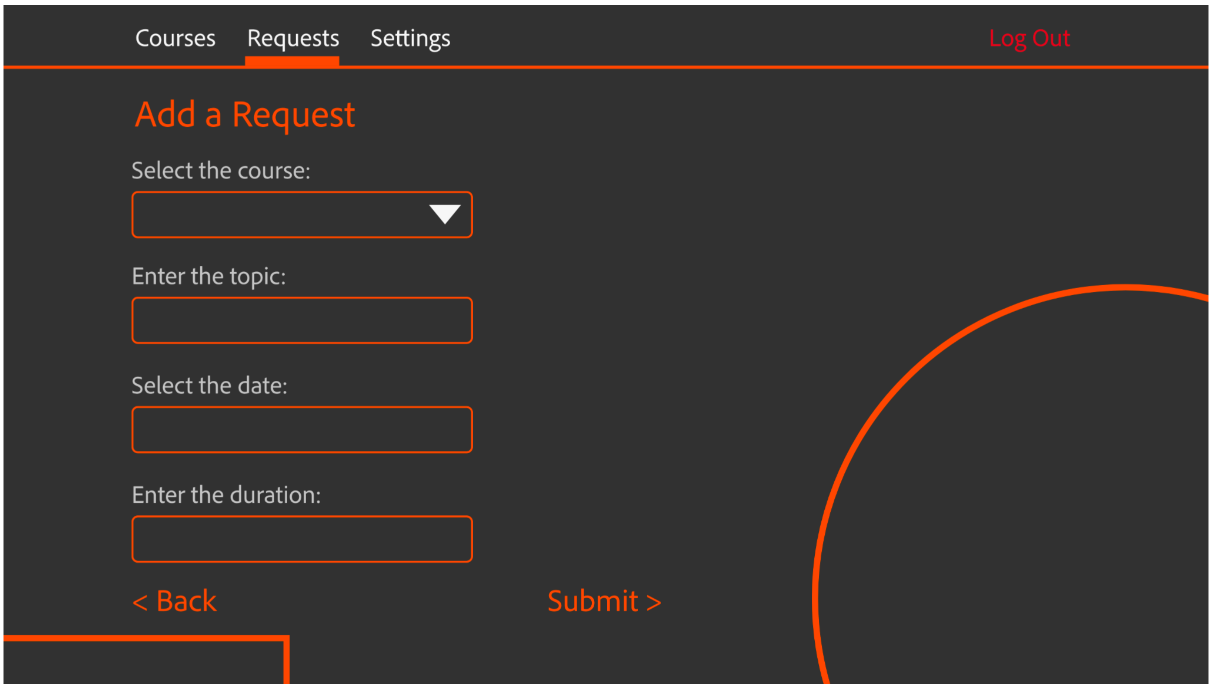
After choosing language, level and teacher the user is enrolled into the course.

Inputs: @user_id, @course_id (all carried from sessions) (user_id from enroll)

SQL Statement to Enroll into a Course:

```
INSERT INTO enroll(student_id, course_id)
VALUES ( @user_id, @course_id);
```

3.3.3 Send a Class Request to the Teacher



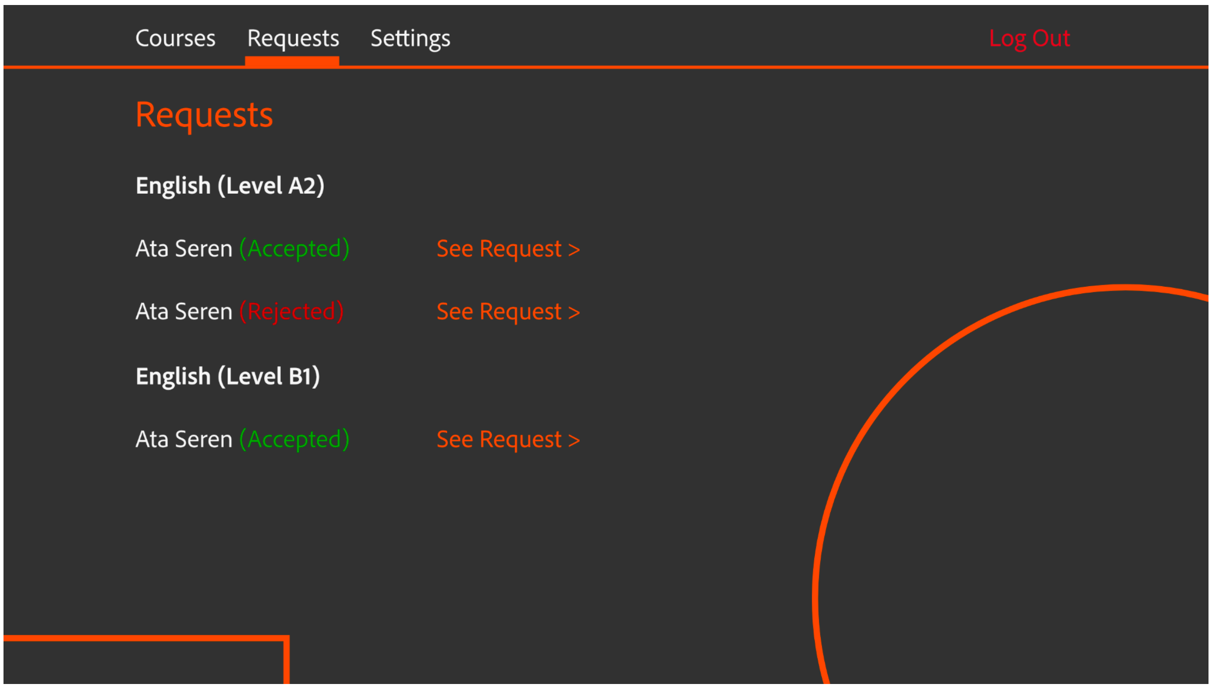
To apply for a class request, the user specifies the course, topic ,date and duration and sends a request to the teacher.

Inputs: @asked_topic, @asked_date, @asked_duration

SQL Statement to Add a Lecture Request:

INSERT INTO lecture_requests(asked_topic, asked_date, asked_duration)

VALUES (@asked_topic, @asked_date, @asked_duration);



Requests

English (Level A2)

Ata Seren (Accepted) See Request >

Ata Seren (Rejected) See Request >

English (Level B1)

Ata Seren (Accepted) See Request >

Teachers can display all current requests and their status. By clicking the see request button, details of a request can be viewed.

Inputs: @user_id (carried from session)

SQL Statement That is Needed:

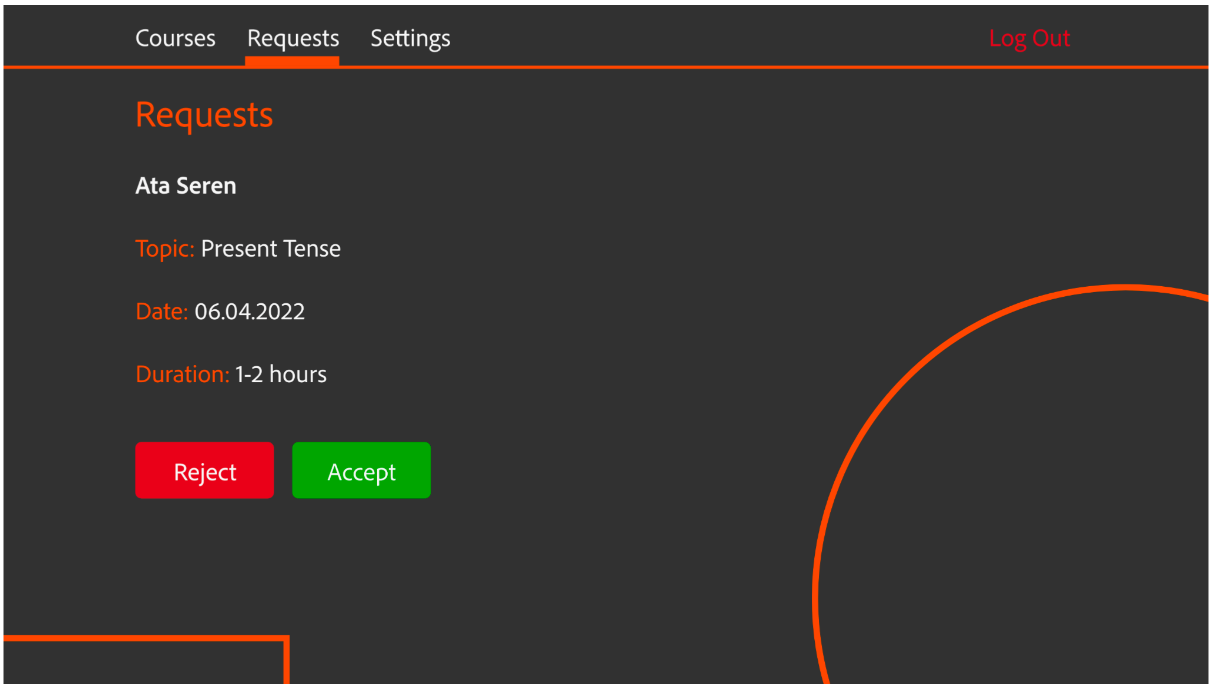
SELECT course_language, course_level, first_name, last_name, lecture_status

FROM lecture_requests NATURAL JOIN request_lecture

NATURAL JOIN student NATURAL JOIN user

NATURAL JOIN enroll NATURAL JOIN course

WHERE user_id = @user_id;



Requests

Ata Seren

Topic: Present Tense

Date: 06.04.2022

Duration: 1-2 hours

After clicking the details button teachers can see the details and reject or accept lecture requests.

Inputs: @l_request_id (carried from session)

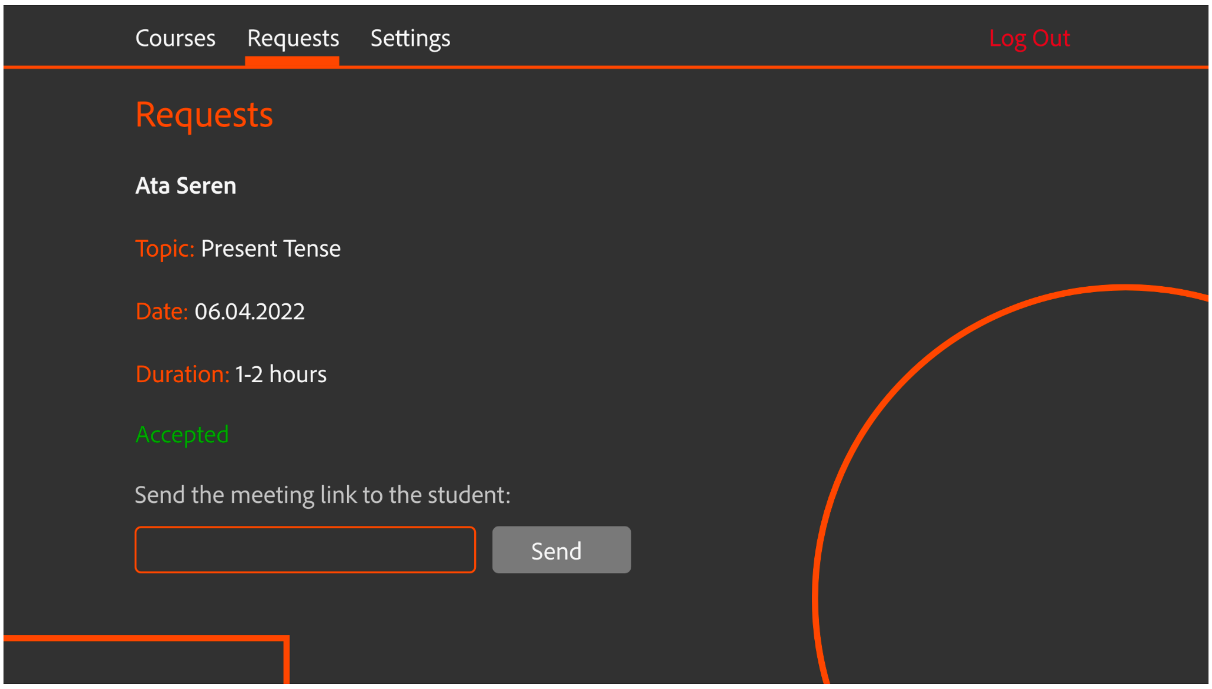
SQL Query to Show Request Details:

SELECT first_name, last_name, asked_topic, asked_date, asked_duration

FROM lecture_requests NATURAL JOIN request_lecture

NATURAL JOIN student NATURAL JOIN user

WHERE l_request_id = @l_request_id;


Courses Requests Settings Log Out

Requests

Ata Seren

Topic: Present Tense

Date: 06.04.2022

Duration: 1-2 hours

Accepted

Send the meeting link to the student:

If the teacher accepts the request, the status of request is updated and it is also added into the lectures table.

Inputs: @lecture_status (considered as true for rest of the scenario)

SQL Statement to Update Lecture Request Status:

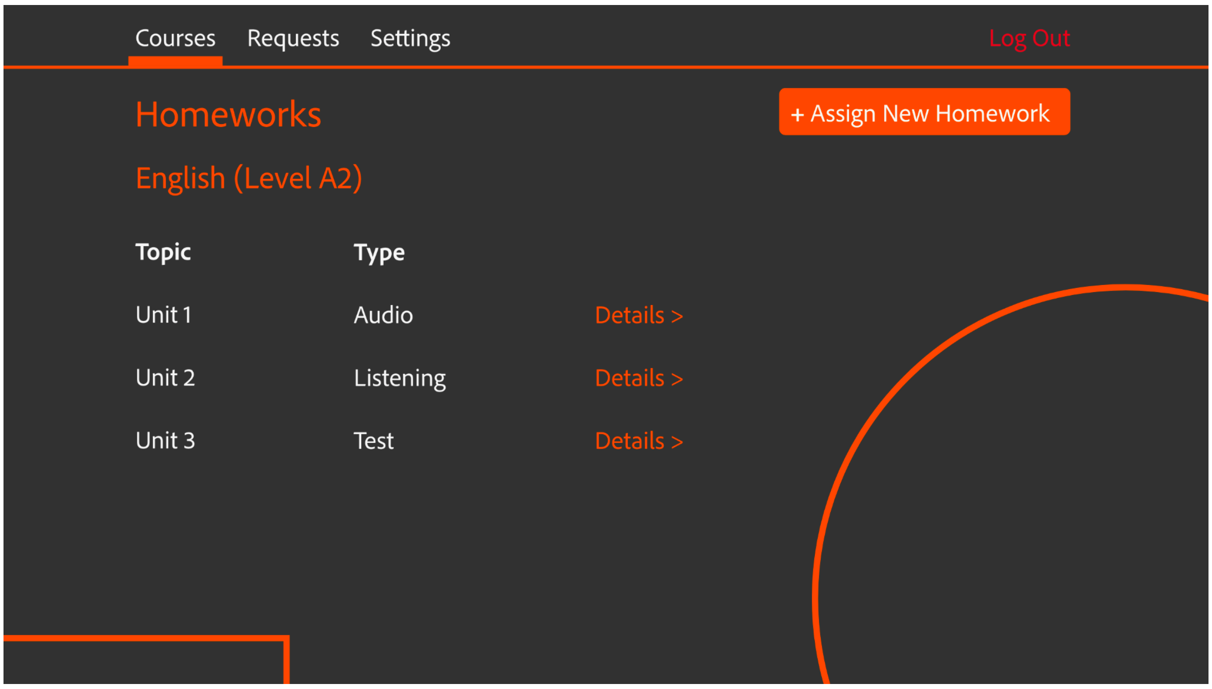
```
UPDATE lecture_request

SET lecture_status = @lecture_status;
```

Inputs: @topic, @link, @date, @attendance

SQL Statement to Add the New Lecture into its Corresponding Table:

```
INSERT INTO lectures(topic, link, date, attendance)

VALUES (@topic, @link, @date, @attendance);
```

3.3.4 Assign Homework from Teachers to Students



Homeworks

English (Level A2)

Topic	Type	
Unit 1	Audio	Details >
Unit 2	Listening	Details >
Unit 3	Test	Details >

On this page, homeworks submitted by a teacher for a certain course is listed. Every student is supposed to access this homework and do it.

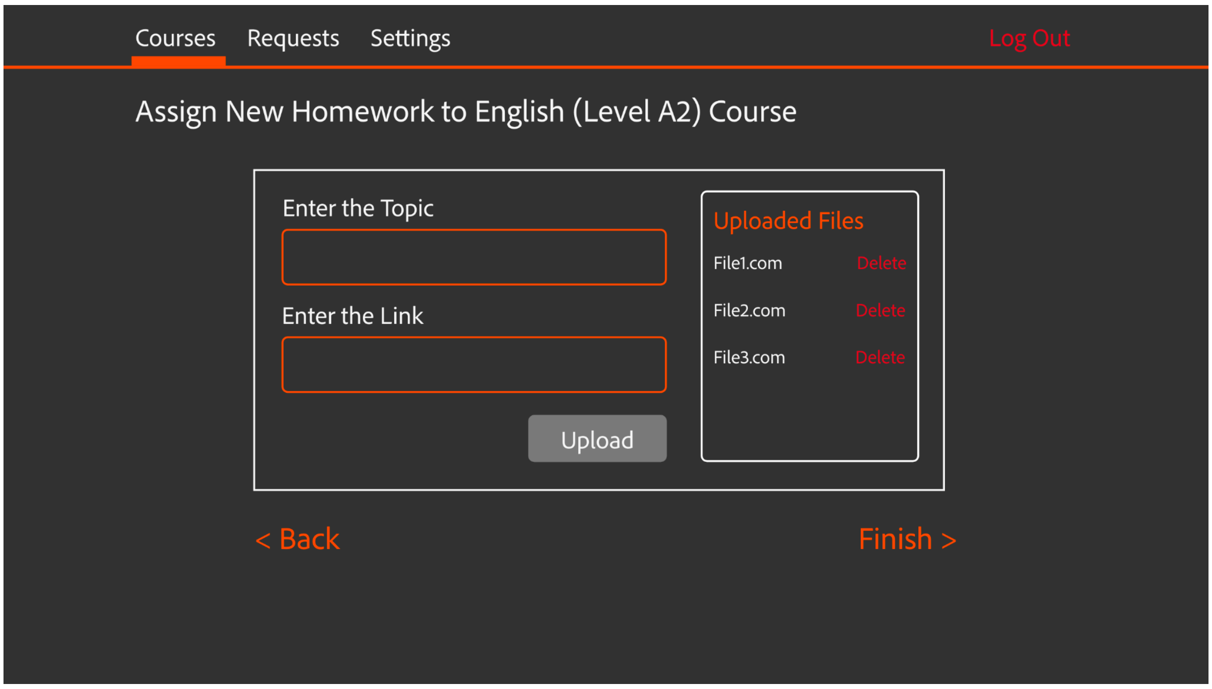
Inputs: @course_id (carried from session)

SQL Query to Display Homeworks Assigned to a Course:

SELECT assignment_name, due_date, topic, hw_type

FROM assignment NATURAL JOIN homework

WHERE course_id = @course_id



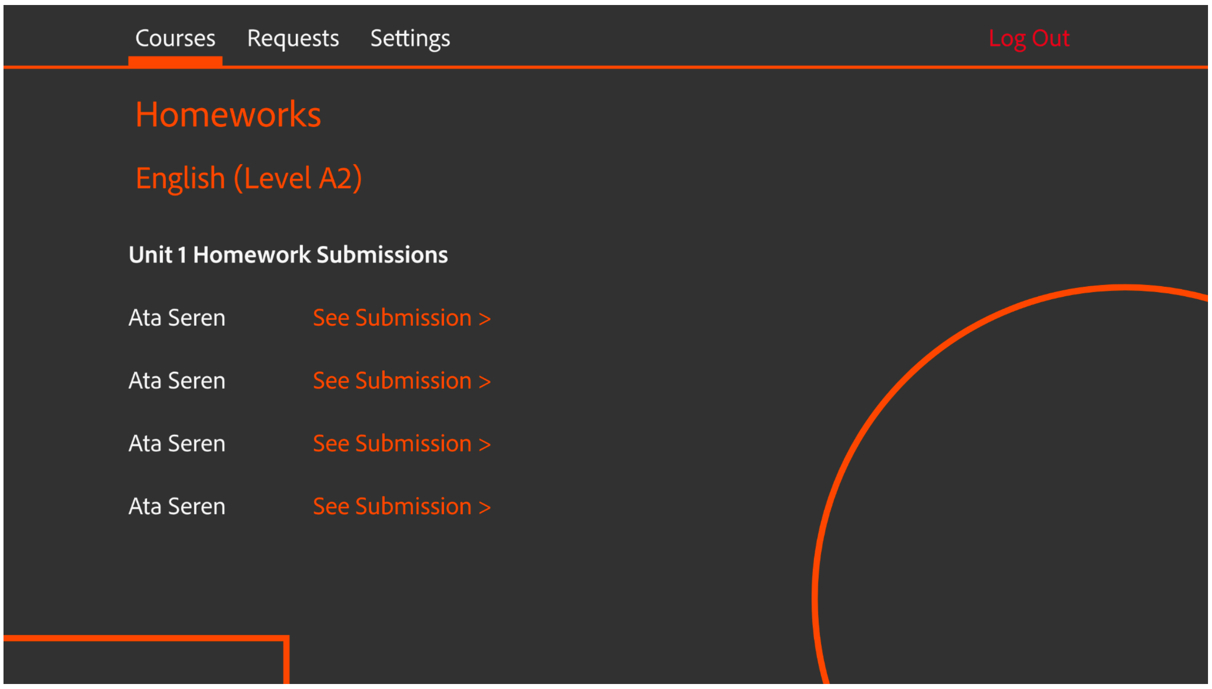
In this page, instructors can add a new assignment to the system by specifying the topic and the link of the assignment.

Inputs: @assignment_name, @progress, @due_date, @assignment_link

SQL Query to Assign a Homework to a Course:

INSERT INTO assignments(assignment_name, progress, due_date, assignment_link)

VALUES (@assignment_name, @progress, @due_date, @assignment_link);



Homeworks

English (Level A2)

Unit 1 Homework Submissions

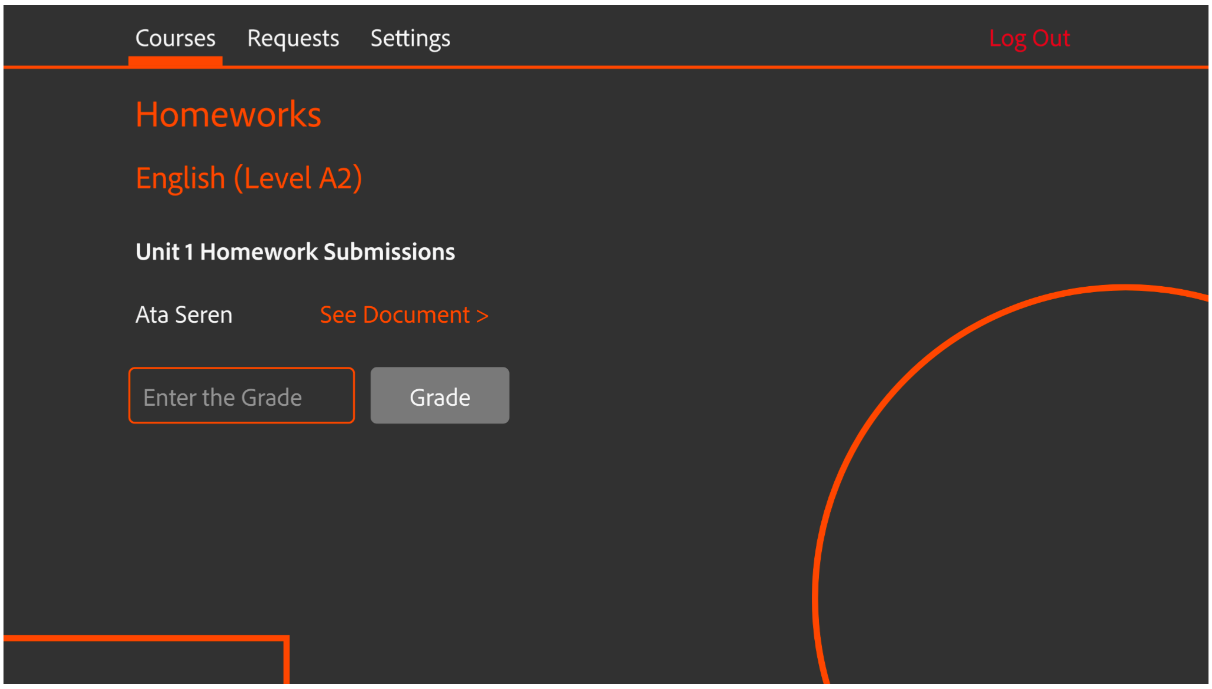
Ata Seren	See Submission >
Ata Seren	See Submission >
Ata Seren	See Submission >
Ata Seren	See Submission >

On this page, submissions of the students for a certain homework are listed. An instructor can see the submission of a certain student and grade it.

Inputs: @assignment_id (carried from session)

SQL Query to See Homework Submissions:

```
SELECT first_name, last_name, assignment_link

FROM assignment NATURAL JOIN homework NATURAL JOIN assigned_to

NATURAL JOIN student NATURAL JOIN user

WHERE assignment_id = @assignment_id
```



On this page, an instructor can see the submitted document and give the grade for it.

Inputs: @assignment_id, @grade, @user_id (carried from session)

SQL Query to Grade the Homeworks:

```
UPDATE assigned_to
SET grade = @grade
WHERE assignment_id = @assignment_id
AND user_id = @user_id
```