

ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 317E
DATABASE SYSTEMS
IMPLEMENTATION

Kuizci Arı

Khayal Huseynov
150180903

8 February 2020

Contents

A	Motivation and Requirements	1
A.1	Idea	1
A.2	Features	1
B	Conceptual Database Design	3
B.1	Data Requirements	3
B.2	ER Model/Diagram	4
C	Logical Database Design	5
C.1	Tables	5
C.2	DDL Statements	5
D	Database Normalization	10
D.1	Functional Dependencies	10
D.1.1	Instructor	10
D.1.2	Instructs	10
D.1.3	Coursework	10
D.1.4	Follows	10
D.1.5	Class	10
D.1.6	User	11
D.1.7	Course	11
D.1.8	Coursework Type	11
D.2	Normalization	12
D.2.1	1NF	12
D.2.2	2NF	12
D.2.3	3NF	13
D.2.4	BCNF	13

E	Application Design and Implementation	14
E.1	Technical Manual	14
E.1.1	Architecture	14
E.1.2	Queries	15
E.1.3	Dependencies	29
E.1.4	Data Sample	29
E.1.5	Issues	29
E.2	User Manual	29
E.2.1	Signup and Login	30
E.2.2	Courses	33
E.2.3	Classes	35
E.2.4	Following	39
E.2.5	Creation, Update and Deletion	40
E.2.6	Profile	47
E.3	Installation Manual	49
E.3.1	Localhost	49
E.3.2	Heroku	49

A Motivation and Requirements

A.1 Idea

Students usually make a list of their own, so they know which course to study for first. This web app will allow for easier management of exams by giving users the ability to keep a dynamic list of exam dates for all the taken courses in a semester. The information will be managed and updated by students and professors.

This web tool will hopefully help alleviate some horror stories I have heard from fellow students. A professor of a course moved the final date forward by three days, causing 14 students to miss the final exam. With the help of this website, students will easily track what exams are coming up next.

This is a contemporary topic since a lot of courses decided to increase the number of quizzes in a course for this semester. This addition has made keeping track of everything harder. "Will we have a quiz this week?" has to be the most asked question among students.

Instructors can use this tool for themselves as well. They can keep track of courses they are teaching and when they will have an exam in a semester. Additionally, they can confirm whether the information on the website is correct or not.

A.2 Features

Here are key operations the visitors, ITU students and instructors can do:

- ITU members are able to create an account with their ITU mail and log in.
- Everyone is able to view the course list and individual course pages.
- Everyone is able to view the class list grouped by semester and individual class pages which display courseworks.
- Registered users (ITU members) are able to follow and unfollow the classes they desire.

- Registered users (ITU members) are able to add, update, and delete classes, courses, and courseworks.
- Followed classes are displayed on a user's personal private page. Courseworks of these followed classes are displayed on the user's another personal private page. Courseworks are sorted by deadline time on this page.

B Conceptual Database Design

B.1 Data Requirements

The following list is the relations of the data requirements of the project proposed for this course:

- A class must have **one and one only** course code.
- A course code may be used by **0 or more** classes.
- A class can have **0 or more** coursework.
- A coursework must be assigned to **one and one only** class.
- A coursework must have **a** type. It can **only have one** type.
- A coursework type can be assigned to **0 or more** coursework.
- A class can be taught by **no or several** instructors.
- An instructor may teach **0 or more** classes during a semester.
- A user may follow **0 or more** classes.
- A class may be followed by **0 or more** users.

B.2 ER Model/Diagram

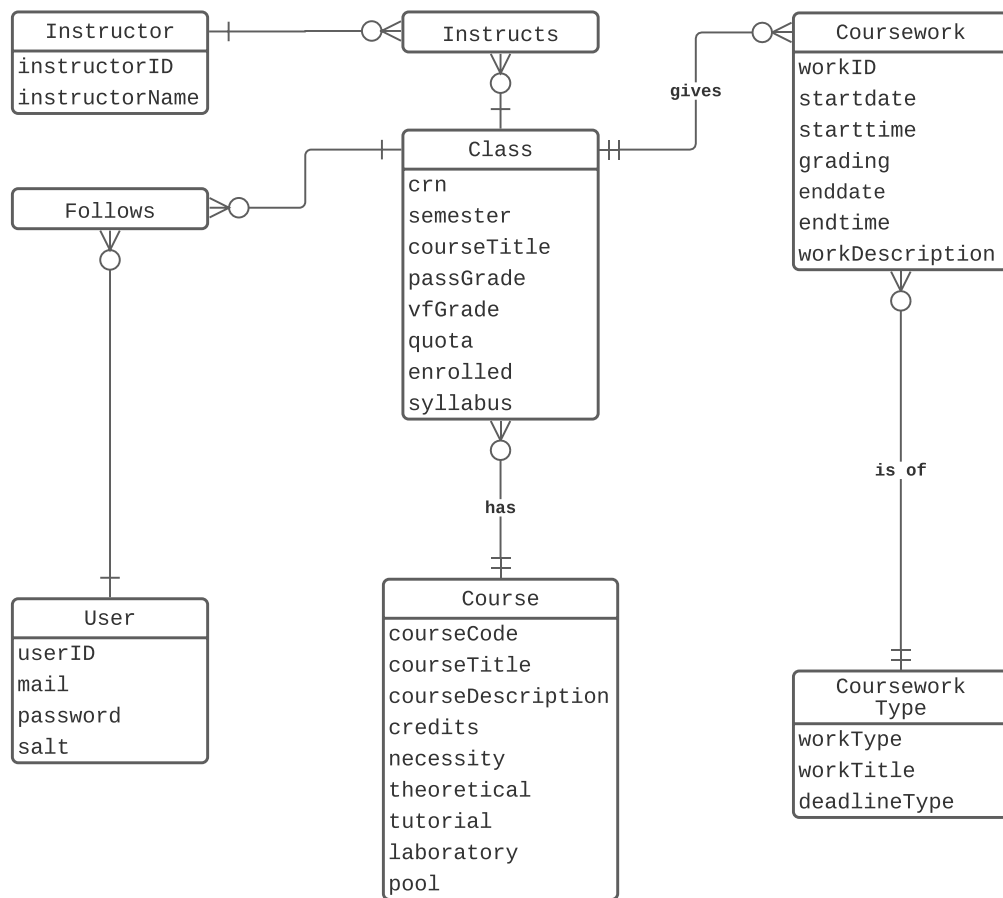


Figure 1: ER diagram in SVG format

C Logical Database Design

C.1 Tables

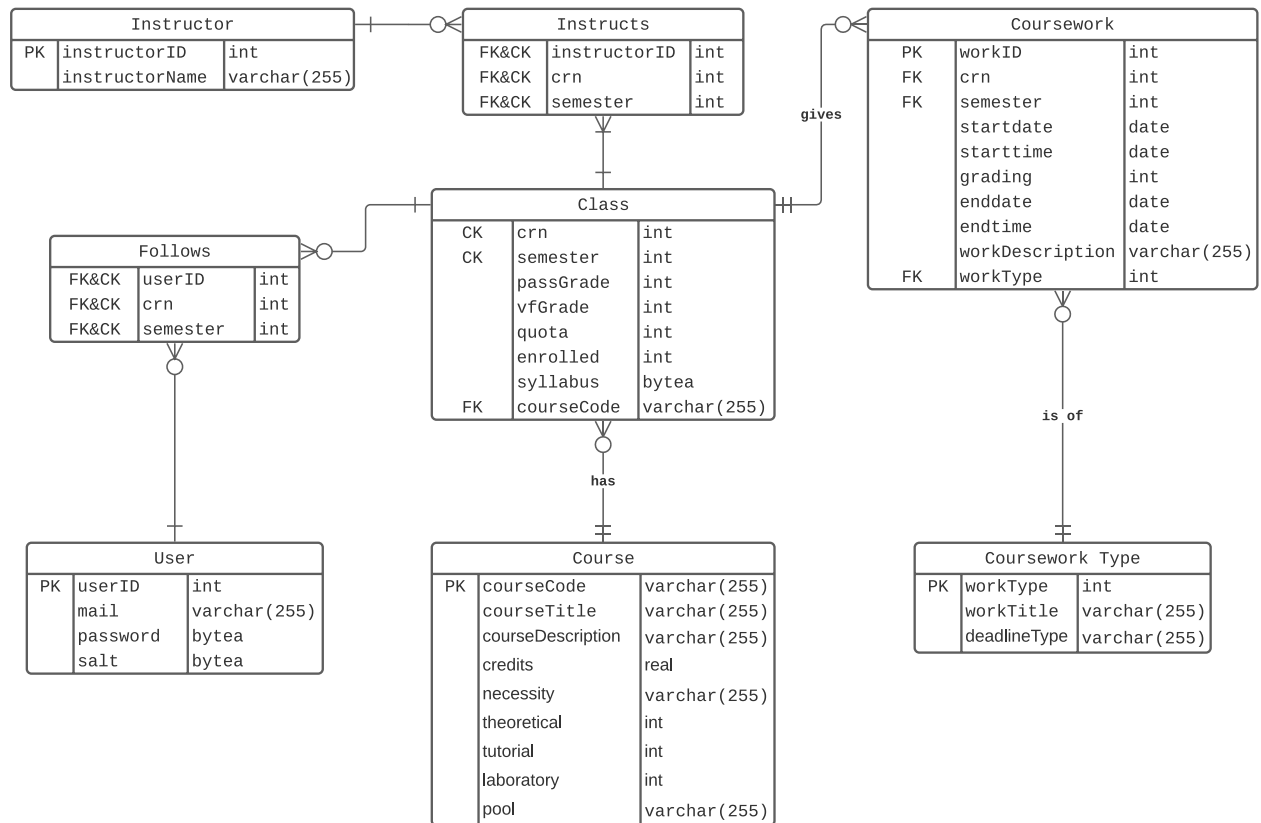


Figure 2: Logical design (tables) in SVG format

C.2 DDL Statements

Listing 1: DDL of User table

```
CREATE TABLE IF NOT EXISTS "User" (
    "userID" SERIAL,
    "mail" VARCHAR(255) NOT NULL UNIQUE,
    "password" BYTEA NOT NULL,
    "salt" BYTEA NOT NULL,
```



```
PRIMARY KEY ("userID")  
);
```

Listing 2: DDL of Coursework Type table

```
CREATE TABLE IF NOT EXISTS "CourseworkType" (  
    "workType" SERIAL,  
    "workTitle" VARCHAR(255) NOT NULL,  
    "deadlineType" VARCHAR(255) NOT NULL,  
    PRIMARY KEY ("workType")  
);
```

Listing 3: DDL of Instructor table

```
CREATE TABLE IF NOT EXISTS "Instructor" (  
    "instructorID" SERIAL,  
    "instructorName" VARCHAR(255) NOT NULL,  
    PRIMARY KEY ("instructorID")  
);
```

Listing 4: DDL of Course table

```
CREATE TABLE IF NOT EXISTS "Course" (  
    "courseCode" VARCHAR(255) NOT NULL UNIQUE,  
    "courseTitle" VARCHAR(255) NOT NULL,  
    "courseDescription" VARCHAR(255),  
    "credits" REAL NOT NULL,
```

```

"pool" VARCHAR(255),
"theoretical" INTEGER NOT NULL,
"tutorial" INTEGER NOT NULL,
"laboratory" INTEGER NOT NULL,
"necessity" VARCHAR(255),
PRIMARY KEY ("courseCode")
);

```

Listing 5: DDL of Class table

```

CREATE TABLE IF NOT EXISTS "Class" (
    "crn" INTEGER NOT NULL,
    "semester" INTEGER NOT NULL,
    "courseCode" VARCHAR(255) NOT NULL,
    "passGrade" INTEGER,
    "vfGrade" INTEGER,
    "quota" INTEGER,
    "enrolled" INTEGER,
    "syllabus" BYTEA,
    FOREIGN KEY ("courseCode") REFERENCES "Course" ("courseCode") ON↵
        DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY ("crn", "semester")
);

```

Listing 6: DDL of Instructs table

```

CREATE TABLE IF NOT EXISTS "Instructs" (
    "instructorID" INTEGER NOT NULL,

```

```

"crn" INTEGER NOT NULL ,
"semester" INTEGER NOT NULL ,
FOREIGN KEY ("instructorID") REFERENCES "Instructor" ("↵
    instructorID") ON DELETE CASCADE ON UPDATE CASCADE ,
FOREIGN KEY ("crn", "semester") REFERENCES "Class" ("crn", "↵
    semester") ON DELETE CASCADE ON UPDATE CASCADE ,
PRIMARY KEY ("instructorID", "crn", "semester")
);

```

Listing 7: DDL of Follows table

```

CREATE TABLE IF NOT EXISTS "Follows" (
    "userID" INTEGER NOT NULL ,
    "crn" INTEGER NOT NULL ,
    "semester" INTEGER NOT NULL ,
    FOREIGN KEY ("userID") REFERENCES "User" ("userID") ON DELETE ↵
        CASCADE ON UPDATE CASCADE ,
    FOREIGN KEY ("crn", "semester") REFERENCES "Class" ("crn", "↵
        semester") ON DELETE CASCADE ON UPDATE CASCADE ,
    PRIMARY KEY ("userID", "crn", "semester")
);

```

Listing 8: DDL of Coursework Table

```

CREATE TABLE IF NOT EXISTS "Coursework" (
    "workID" SERIAL NOT NULL ,
    "crn" INTEGER NOT NULL ,
    "semester" INTEGER NOT NULL ,

```

```

"startdate" DATE NOT NULL,
"starttime" TIME NOT NULL,
"enddate" DATE NOT NULL,
"endtime" TIME NOT NULL,
"grading" INTEGER NOT NULL,
"workDescription" VARCHAR(255),
"workType" INTEGER NOT NULL,
FOREIGN KEY ("workType") REFERENCES "CourseworkType" ("workType"↵
    ) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY ("crn", "semester") REFERENCES "Class" ("crn", "↵
    semester") ON DELETE CASCADE ON UPDATE CASCADE,
PRIMARY KEY ("workID")
);

```

D Database Normalization

D.1 Functional Dependencies

Here are the functional dependencies categorized according to the table they are in

D.1.1 Instructor

$\text{instructorID} \rightarrow \text{instructorName}$

D.1.2 Instructs

There are no functional dependencies other than the trivial one.

D.1.3 Coursework

$\text{workID} \rightarrow \text{startdate}$

$\text{workID} \rightarrow \text{starttime}$

$\text{workID} \rightarrow \text{grading}$

$\text{workID} \rightarrow \text{enddate}$

$\text{workID} \rightarrow \text{endtime}$

$\text{workID} \rightarrow \text{workDescription}$

$\text{workID} \rightarrow \text{workType}$

$\text{workID} \rightarrow \text{crn}$

$\text{workID} \rightarrow \text{semester}$

D.1.4 Follows

There are no functional dependencies other than the trivial one.

D.1.5 Class

$\{\text{crn}, \text{semester}\} \rightarrow \text{courseCode}$

$\{\text{crn}, \text{semester}\} \rightarrow \text{passGrade}$

$\{\text{crn}, \text{semester}\} \rightarrow \text{vfGrade}$

$\{\text{crn}, \text{semester}\} \rightarrow \text{quota}$

$\{\text{crn}, \text{semester}\} \rightarrow \text{enrolled}$

$\{\text{crn}, \text{semester}\} \rightarrow \text{syllabus}$

D.1.6 User

$\text{userID} \rightarrow \text{mail}$

$\text{mail} \rightarrow \text{userID}$

$\text{userID} \rightarrow \text{password}$

$\text{userID} \rightarrow \text{salt}$

D.1.7 Course

$\text{courseCode} \rightarrow \text{courseTitle}$

$\text{courseCode} \rightarrow \text{courseDescription}$

$\text{courseCode} \rightarrow \text{credits}$

$\text{courseCode} \rightarrow \text{necessity}$

$\text{courseCode} \rightarrow \text{theoretical}$

$\text{courseCode} \rightarrow \text{tutorial}$

$\text{courseCode} \rightarrow \text{laboratory}$

$\text{courseCode} \rightarrow \text{pool}$

D.1.8 Coursework Type

$\text{workType} \rightarrow \text{workTitle}$

$\text{workType} \rightarrow \text{deadlineType}$

D.2 Normalization

The functional dependencies that come from the tables seem to be in BCNF. Let's follow the steps one by one and confirm.

D.2.1 1NF

Requirements:

- Attribute values are atomic. Each cell contains a single value.
- Each record is unique.

The first requirement holds as we do not store lists in a single cell. The second requirement also holds as the primary keys or composite keys of tables are already determined and they cover all the entries in a table.

D.2.2 2NF

- Conforms with 1NF.
- Every non-key attribute depends on the primary key.

It has already been established that the functional dependencies conform with 1NF in the previous section. The second argument also holds true. Every non-key attribute is additional information provided for the primary key.

D.2.3 3NF

- Conforms with 2NF.
- Non-key attributes do not depend on any attributes other than the primary key.

It has already been established that the functional dependencies conform with 2NF in the previous section. For the second argument, there are no relationship between non-key attributes.

D.2.4 BCNF

- Conforms with 3NF.
- All functional dependencies must be on candidate keys. There can not be dependencies between key attributes

It has already been established that the functional dependencies conform with 3NF in the previous section. There are no relations between key attributes, except the one between userID and email:

$\text{userID} \rightarrow \text{mail}$

$\text{mail} \rightarrow \text{userID}$

This dependency breaks the BCNF rule. Our weakest link is of type 3NF, meaning our table is 3NF.

E Application Design and Implementation

E.1 Technical Manual

E.1.1 Architecture

The web application has three tiers, namely, Presentation tier, Logic tier, and Data tier (Figure 3).

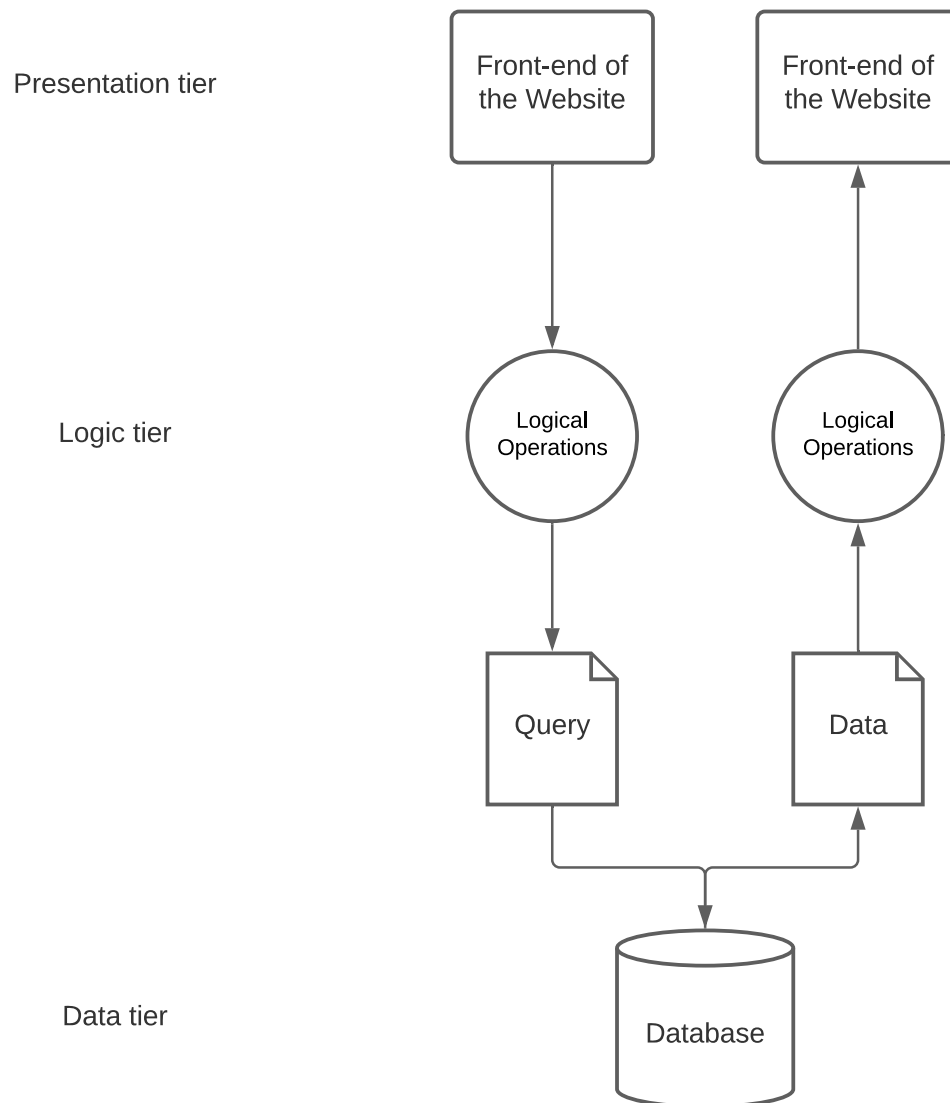


Figure 3: Architecture of the application

Presentation is accomplished through the browser of the user. The content is generated

dynamically by the Logic tier and is pushed to the Presentation tier. The browser renders the received content.

Logic tier is the dynamic content processing and generation level. This layer coordinates the web application and processes requests coming from the Presentation tier (by the user) and queries the database to get data, process it and send it back to the Presentation tier. Logic tier is realized through Flask micro web framework.

Data tier is comprised of a DataBase Management Software (DBMS) that manages incoming queries and provides access to the data.

E.1.2 Queries

Following are the queries used by the application. Keywords are marked with blue, strings are marked with red, and placeholders are marked with green. Underneath each query is its semantic.

Listing 9: get_semesters()

```
SELECT COUNT("Class"."crn"), "Class"."semester"  
FROM "Class"  
GROUP BY "Class"."semester"
```

Semantics: List all semesters that have class and count of the classes.

Listing 10: get_classes(semester)

```
SELECT "Class"."crn", "Class"."courseCode", "courseTitle", "Class"  
      " semester "  
FROM "Class"  
LEFT JOIN "Course"  
ON "Class"."courseCode" = "Course"."courseCode"
```

```
WHERE "Class"."semester" = %(semester)s;
```

Semantics: List all classes of the requested semester.

Listing 11: get_classes_with_instructors(semester)

```
SELECT "instructorName"
FROM "Instructs"
LEFT JOIN "Instructor"
    ON "Instructor"."instructorID" = "Instructs"."instructorID"
WHERE "Instructs"."crn" = %(crn)s
    AND "Instructs"."semester" = %(semester)s;
```

Semantics: List all instructors of the requested class.

Listing 12: get_class(crn,semester)

```
SELECT "Class"."crn", "Class"."semester", "Class"."courseCode", "↔
    courseTitle"
FROM "Class"
LEFT JOIN "Course"
    ON "Class"."courseCode" = "Course"."courseCode"
WHERE "Class"."crn" = %(crn)s
    AND "Class"."semester" = %(semester)s;
```

Semantics: List main information about the requested class.

Listing 13: get_whole_class(crn, semester)

```
SELECT "Class"."crn", "Class"."semester", "Class"."courseCode", "↔
```

```

        courseTitle" , "passGrade", "vfGrade", "quota", "enrolled", "↵
        syllabus"
FROM "Class"
LEFT JOIN "Course"
    ON "Class"."courseCode" = "Course"."courseCode"
WHERE "Class"."crn" = %(crn)s
    AND "Class"."semester" = %(semester)s;

```

Semantics: List all information about the requested class.

Listing 14: get_courseworks(crn, semester)

```

SELECT "Coursework"."workID", "CourseworkType"."workTitle", "↵
        Coursework"."startdate", "Coursework"."starttime", "Coursework"↵
        . "enddate", "Coursework"."endtime", "Coursework"."grading", "↵
        Coursework"."workDescription"
FROM "Coursework"
LEFT JOIN "CourseworkType"
    ON "CourseworkType"."workType" = "Coursework"."workType"
LEFT JOIN "Class"
    ON "Class"."crn" = "Coursework"."crn"
    AND "Class"."semester" = "Coursework"."semester"
WHERE "Class"."crn" = %(crn)s
    AND "Class"."semester" = %(semester)s;

```

Semantics: List courseworks of the requested class.

Listing 15: get_coursework(workID)

```

SELECT "Coursework"."workID", "CourseworkType"."workTitle", "↵
    Coursework"."startdate", "Coursework"."starttime", "Coursework"↵
    ."enddate", "Coursework"."endtime", "Coursework"."grading", "↵
    Coursework"."workDescription", "Coursework"."workType", "↵
    Coursework"."crn", "Coursework"."semester"
FROM "Coursework"
LEFT JOIN "CourseworkType"
    ON "CourseworkType"."workType" = "Coursework"."workType"
WHERE "Coursework"."workID" = %(workID)s;

```

Semantics: List all information about the requested coursework.

Listing 16: get_follow(userID, crn, semester)

```

SELECT "Follows"."userID", "Follows"."crn", "Follows"."semester"
FROM "Follows"
WHERE "Follows"."userID" = %(userID)s
    AND "Follows"."crn" = %(crn)s
    AND "Follows"."semester" = %(semester)s;

```

Semantics: Check if a follow relation exists between the requested user and the requested class.

Comment: A single row is returned if the user follows the class.

Listing 17: delete_class(crn, semester)

```

DELETE FROM "Class"
WHERE "crn" = %(crn)s
    AND "semester" = %(semester)s;

```

Semantics: Delete the requested class.

Listing 18: get_instructors()

```
SELECT "instructorID", "instructorName"
FROM "Instructor";
```

Semantics: List all instructors.

Listing 19: get_class_instructors(crn, semester)

```
SELECT "Instructs"."instructorID", "instructorName"
FROM "Instructs"
LEFT JOIN "Instructor"
    ON "Instructor"."instructorID" = "Instructs"."instructorID"
WHERE "Instructs"."crn" = %(crn)s
AND "Instructs"."semester" = %(semester)s;
```

Semantics: List all instructors of the requested class.

Comment: This query is the exact copy of the Query 11. It was used again inside another function so it is included again.

Listing 20: get_course(courseCode)

```
SELECT "courseCode", "courseTitle", "courseDescription", "credits"↵
    , "pool", "theoretical", "tutorial", "laboratory", "necessity"
FROM "Course"
WHERE "courseCode" = %(courseCode)s;
```

Semantics: List all information about the requested course.

Listing 21: update_course(courseCode, newCourseCode, courseTitle, description, credit, necessity, theoretical, tutorial, laboratory, pool)

```
UPDATE "Course"
SET "courseCode" = %(newCourseCode)s,
    "courseTitle" = %(courseTitle)s,
    "courseDescription" = %(courseDescription)s,
    "credits" = %(credits)s,
    "necessity" = %(necessity)s,
    "theoretical" = %(theoretical)s,
    "tutorial" = %(tutorial)s,
    "laboratory" = %(laboratory)s,
    "pool" = %(pool)s
WHERE "courseCode" = %(courseCode)s;
```

Semantics: Update the requested course with new information.

Listing 22: get_courses()

```
SELECT "Course"."courseCode", "Course"."courseTitle", COUNT("Class"↵
    ".*crn"), COUNT(DISTINCT "Class"."semester")
FROM "Course"
LEFT JOIN "Class"
    ON "Class"."courseCode" = "Course"."courseCode"
GROUP BY "Course"."courseCode"
ORDER BY "courseCode" ASC;
```

Semantics: List all courses in the ascending order according to their code.

Listing 23: delete_course(courseCode)

```
DELETE FROM "Course"  
WHERE "courseCode" = %(courseCode)s;
```

Semantics: Delete requested course.

Listing 24: get_courseworkTypes()

```
SELECT "workType", "workTitle"  
FROM "CourseworkType";
```

Semantics: List coursework types.

Listing 25: add_class(crn, semester, courseCode, passGrade, vfGrade, quota, enrolled, syllabus)

```
INSERT INTO "Class" ("crn", "semester", "courseCode", "passGrade",  
    "vfGrade", "quota", "enrolled", "syllabus")  
VALUES(%(crn)s, %(semester)s, %(courseCode)s, %(passGrade)s, %(  
    vfGrade)s, %(quota)s, %(enrolled)s, %(syllabus)s);
```

Semantics: Add a class.

Listing 26: update_class(crn, semester, newcrn, newsemester, courseCode, passGrade, vfGrade, quota, enrolled, syllabus)

```
UPDATE "Class"  
SET "crn" = %(newcrn)s,  
    "semester" = %(newsemester)s,  
    "courseCode" = %(courseCode)s,
```



```

    "passGrade" = %(passGrade)s,
    "vfGrade" = %(vfGrade)s,
    "quota" = %(quota)s,
    "enrolled" = %(enrolled)s,
    "syllabus" = %(syllabus)s
WHERE "crn" = %(crn)s
AND "semester" = %(semester)s;

```

Semantics: Update the requested class.

Listing 27: add_course(courseCode, courseTitle, description, credit, necessity, theoretical, tutorial, laboratory, pool)

```

INSERT INTO "Course" ("courseCode", "courseTitle", "↵
    courseDescription", "credits", "necessity", "theoretical", "↵
    tutorial", "laboratory", "pool")
VALUES(%(courseCode)s, %(courseTitle)s, %(courseDescription)s, %(↵
    credits)s, %(necessity)s, %(theoretical)s, %(tutorial)s, %(↵
    laboratory)s, %(pool)s);

```

Semantics: Add a course.

Listing 28: add_instructor(instructorName)

```

INSERT INTO "Instructor" ("instructorName")
VALUES(%(instructorName)s);

```

Semantics: Add an instructor.

Listing 29: add_instructs(crn, semester, instructorID)

```
INSERT INTO "Instructs" ("instructorID", "crn", "semester")
VALUES(%(instructorID)s, %(crn)s, %(semester)s);
```

Semantics: Assign the requested instructor to the requested class.

Listing 30: remove_instructs(crn, semester, instructorID)

```
DELETE FROM "Instructs"
WHERE "crn" = %(crn)s
    AND "instructorID" = %(instructorID)s
    AND "semester" = %(semester)s;
```

Semantics: Delete the requested class.

Listing 31: add_courseworktype(workTitle, deadlineType)

```
INSERT INTO "CourseworkType" ("workTitle", "deadlineType")
VALUES(%(workTitle)s, %(deadlineType)s);
```

Semantics: Add a coursework type.

Listing 32: add_coursework(crn, semester, startdate, starttime, enddate, endtime, grading, description, workType)

```
INSERT INTO "Coursework" ("crn", "semester", "startdate", "↵
    starttime", "enddate", "endtime", "grading", "workDescription",↵
    "workType")
VALUES(%(crn)s, %(semester)s, %(startdate)s, %(starttime)s, %(↵
    enddate)s, %(endtime)s, %(grading)s, %(description)s, %(↵
```

```
workType)s);
```

Semantics: Add a coursework.

Listing 33: update_coursework(workID, startdate, starttime, enddate, endtime, grading, description, workType)

```
UPDATE "Coursework"
SET "startdate" = %(startdate)s,
    "starttime" = %(starttime)s,
    "enddate" = %(enddate)s,
    "endtime" = %(endtime)s,
    "grading" = %(grading)s,
    "workDescription" = %(description)s,
    "workType" = %(workType)s
WHERE "workID" = %(workID)s;
```

Semantics: Update the requested coursework with given information.

Listing 34: delete_coursework(id)

```
DELETE FROM "Coursework"
WHERE "workID" = %(id)s;
```

Semantics: Delete the coursework with the given work ID.

Listing 35: add_follow(userID, crn, semester)

```
INSERT INTO "Follows" ("userID", "crn", "semester")
VALUES(%(userID)s, %(crn)s, %(semester)s);
```

Semantics: Add a follow relation between the requested user and the requested class.

Listing 36: remove_follow(userID, crn, semester)

```
DELETE FROM "Follows"  
WHERE "userID" = %(userID)s  
    AND "crn" = %(crn)s  
    AND "semester" = %(semester)s;
```

Semantics: Delete the follow relation between the requested user and the requested class.

Listing 37: signup(mail, password)

```
INSERT INTO "User" ("mail", "password", "salt")  
VALUES(%(mail)s, %(password)s, %(salt)s);
```

Semantics: Add a new user.

Listing 38: checkMail(mail)

```
SELECT "mail"  
FROM "User"  
WHERE "User"."mail" = %(mail)s;
```

Semantics: Select the user with the requested e-mail address.

Listing 39: checkPass(mail, password_attempt)

```
SELECT "password", "salt"  
FROM "User"  
WHERE "User"."mail" = %(mail)s;
```

Semantics: Get the password of the user with the requested e-mail address.

Listing 40: checkPass(mail, password_attempt)

```
SELECT "userID"
FROM "User"
WHERE "User"."mail" = %(mail)s
    AND "User"."password" = %(password)s;
```

Semantics: Get the user ID with the matching e-mail address and password.

Listing 41: get_following_classes(userID)

```
SELECT "Class"."crn", "Class"."courseCode", "Course"."courseTitle"↵
    , "Class"."semester", "passGrade", "vfGrade", "quota", "↵
    enrolled", "syllabus", COUNT("Coursework"."workID"), SUM("↵
    Coursework"."grading")
FROM "Follows"
LEFT JOIN "Class"
    ON "Class"."crn" = "Follows"."crn"
    AND "Class"."semester" = "Follows"."semester"
LEFT JOIN "Course"
    ON "Class"."courseCode" = "Course"."courseCode"
LEFT JOIN "Coursework"
    ON "Coursework"."crn" = "Class"."crn"
    AND "Coursework"."semester" = "Class"."semester"
WHERE "Follows"."userID" = %(userID)s
GROUP BY "Class"."crn", "Class"."semester", "Course"."courseTitle"↵
```

;

Semantics: List various information about the classes followed by the requested user.

Listing 42: get_following_classes(userID)

```
SELECT "instructorName"
FROM "Instructs"
LEFT JOIN "Instructor"
ON "Instructor"."instructorID" = "Instructs"."instructorID"
WHERE "Instructs"."crn" = %(crn)s
    AND "Instructs"."semester" = %(semester)s;
```

Semantics: List all instructors of the requested class.

Comment: This query is the exact copy of the Query 11. It was used again inside another function so it is included again.

Listing 43: get_following_courseworks(userID)

```
(SELECT "workID", "startdate" AS date, "starttime" AS time, "↵
    grading", "Coursework"."crn", "Coursework"."semester", "Class".↵
    "courseCode", "Course"."courseTitle", "CourseworkType"."↵
    workTitle"
FROM "Coursework"
LEFT JOIN "CourseworkType"
    ON "CourseworkType"."workType" = "Coursework"."workType"
LEFT JOIN "Class"
    ON "Class"."crn" = "Coursework"."crn"
    AND "Class"."semester" = "Coursework"."semester"
```

```

LEFT JOIN "Course"
    ON "Class"."courseCode" = "Course"."courseCode"
RIGHT JOIN "Follows"
    ON "Follows"."crn" = "Coursework"."crn"
    AND "Follows"."semester" = "Coursework"."semester"
    AND "Follows"."userID" = %(userID)s
WHERE "CourseworkType"."deadlineType" = 'start'
UNION
SELECT "workID", "enddate" AS date, "endtime" AS time, "grading", ↵
    "Coursework"."crn", "Coursework"."semester", "Class"."↵
    courseCode", "Course"."courseTitle", "CourseworkType"."↵
    workTitle"
FROM "Coursework"
LEFT JOIN "CourseworkType"
    ON "CourseworkType"."workType" = "Coursework"."workType"
LEFT JOIN "Class"
    ON "Class"."crn" = "Coursework"."crn"
    AND "Class"."semester" = "Coursework"."semester"
LEFT JOIN "Course"
    ON "Class"."courseCode" = "Course"."courseCode"
RIGHT JOIN "Follows"
    ON "Follows"."crn" = "Coursework"."crn"
    AND "Follows"."semester" = "Coursework"."semester"
    AND "Follows"."userID" = %(userID)s
WHERE "CourseworkType"."deadlineType" = 'end')
ORDER BY "date" ASC, "time" ASC;

```

Semantics: Get courseworks of the classes that the requested user is following. Order

these courseworks according to their important dates in an ascending manner.

E.1.3 Dependencies

The project is programmed in Python, using the Flask framework as it is required of us. Project is written to work with a PostgreSQL database. The SQL database is accessed via a dbapi2 compatible driver called psycopg2. Additionally, Bulma CSS-framework is used to design the front-end of the website. Dropdown selections are designed with Select2 which is built on jQuery.

E.1.4 Data Sample

The data seen on the demo and hosted Heroku website was manually entered from our university's Student Information System website.

E.1.5 Issues

The website was tested during manual data entry and during the demo. No issues were spotted.

E.2 User Manual

Here are the features of the application copied from Features subsection of the Motivation and Requirements section (Section A.2).

1. ITU members are able to create an account with their ITU mail and log in.
2. Everyone is able to view the course list and individual course pages.
3. Everyone is able to view the class list grouped by semester and individual class pages which display courseworks.
4. Registered users (ITU members) are able to follow and unfollow the classes they desire.

5. Registered users (ITU members) are able to add, update, and delete classes, courses, and courseworks.
6. Followed classes are displayed on a user's personal private page. Courseworks of these followed classes are displayed on the user's another personal private page. Courseworks are sorted by deadline time on this page.

E.2.1 Signup and Login

ITU members are able to create an account with their ITU mail and log in (Item 1).

When a visitor enters the website, the visitor is greeted with the following home page that can be seen from Figure 4.

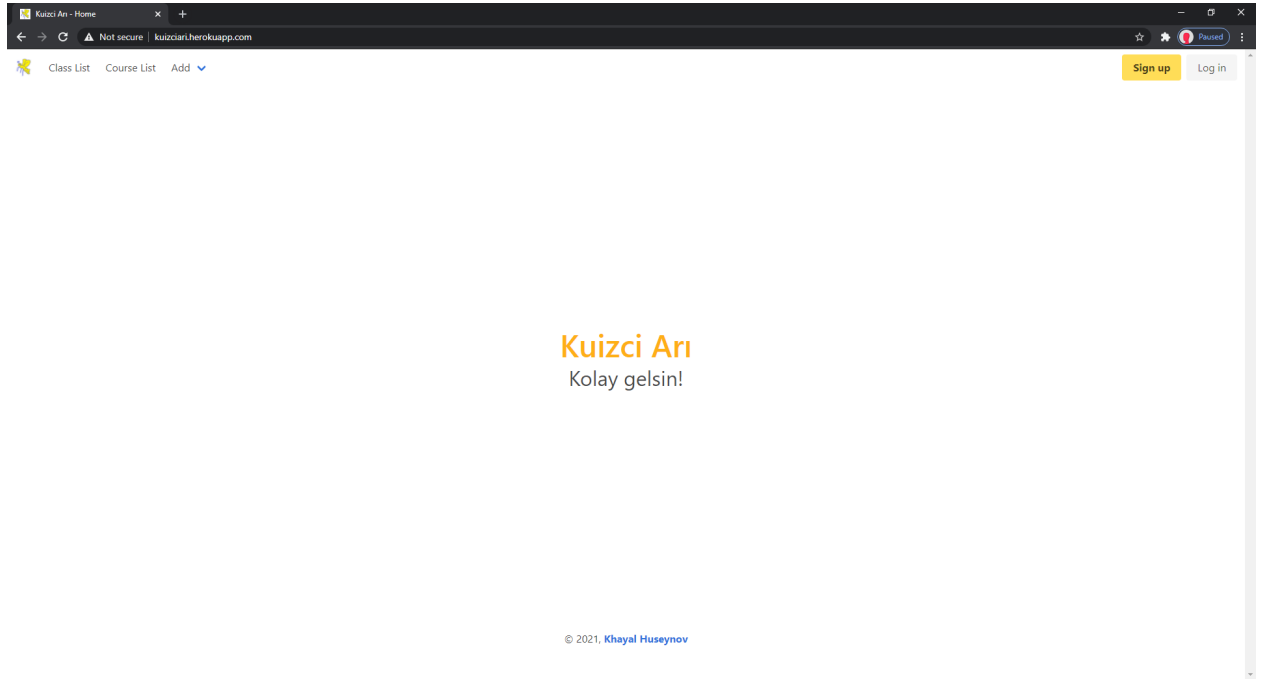


Figure 4: Home Page

The visitor can then create an account by clicking the Sign Up button on the top right corner of the screen. Only e-mails ending with the ITU address are accepted. Otherwise the error message will be displayed (Figure 5).

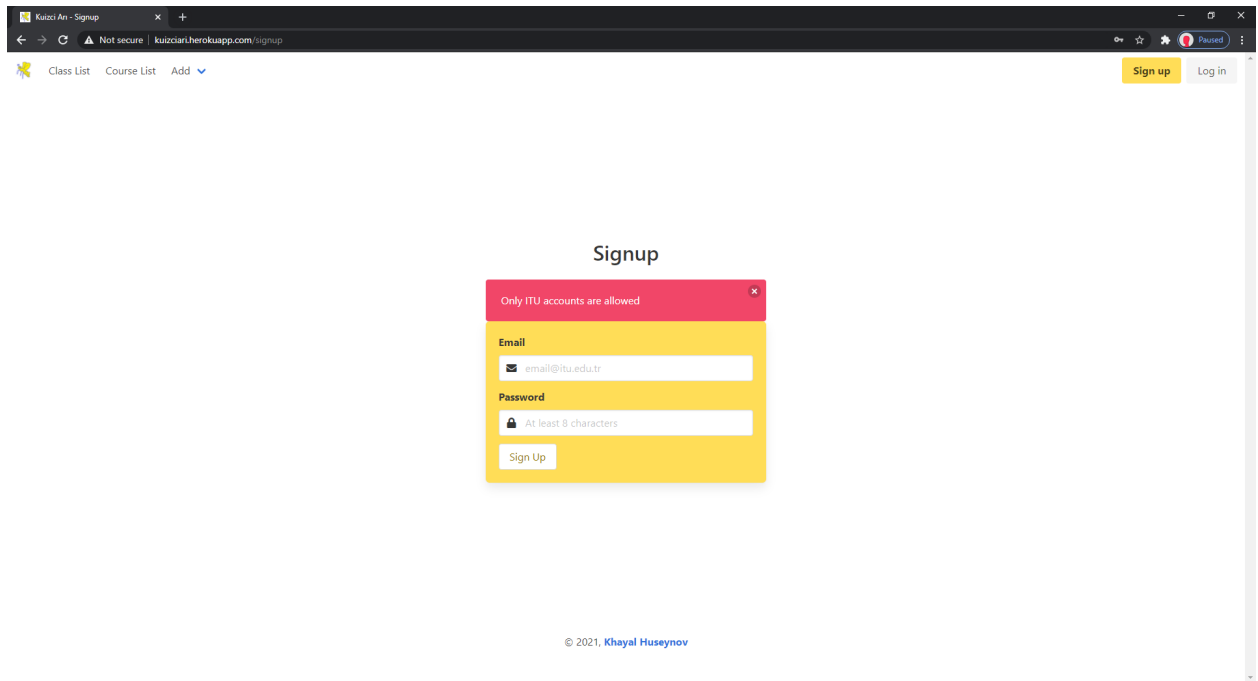


Figure 5: Signup ITU Mail error

After signup, registered user will be transported to the login page to log in with their new credentials (Figure 6).

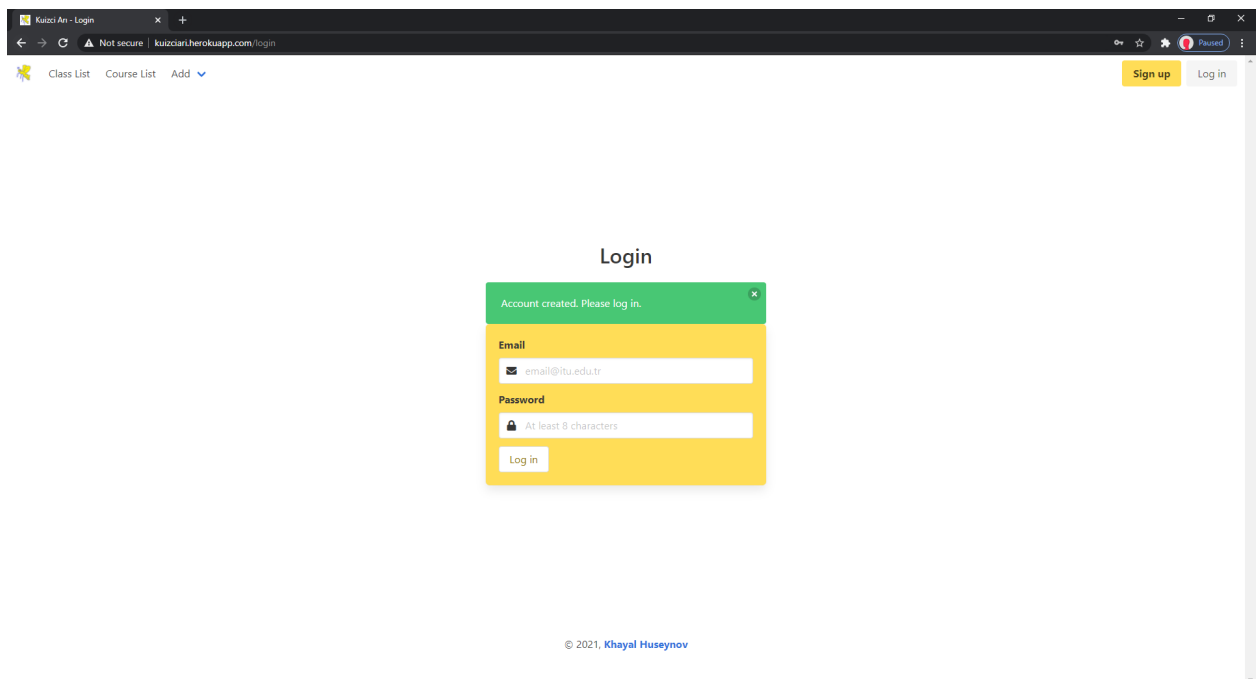


Figure 6: Login Page with Flash

In the case of wrong e-mail entry, user will be prompted with the appropriate message (Figure 7).

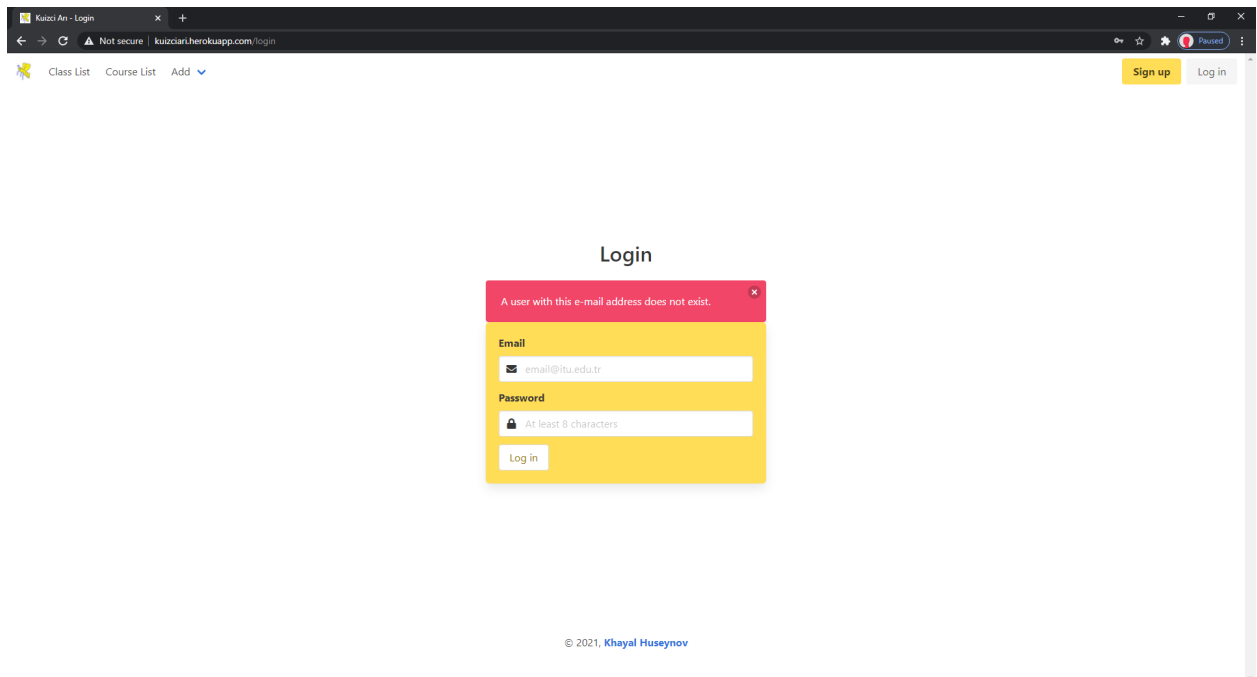


Figure 7: Login with Incorrect Mail

In the case of wrong password entry with the associated e-mail, user will be prompted with the appropriate message (Figure 8).

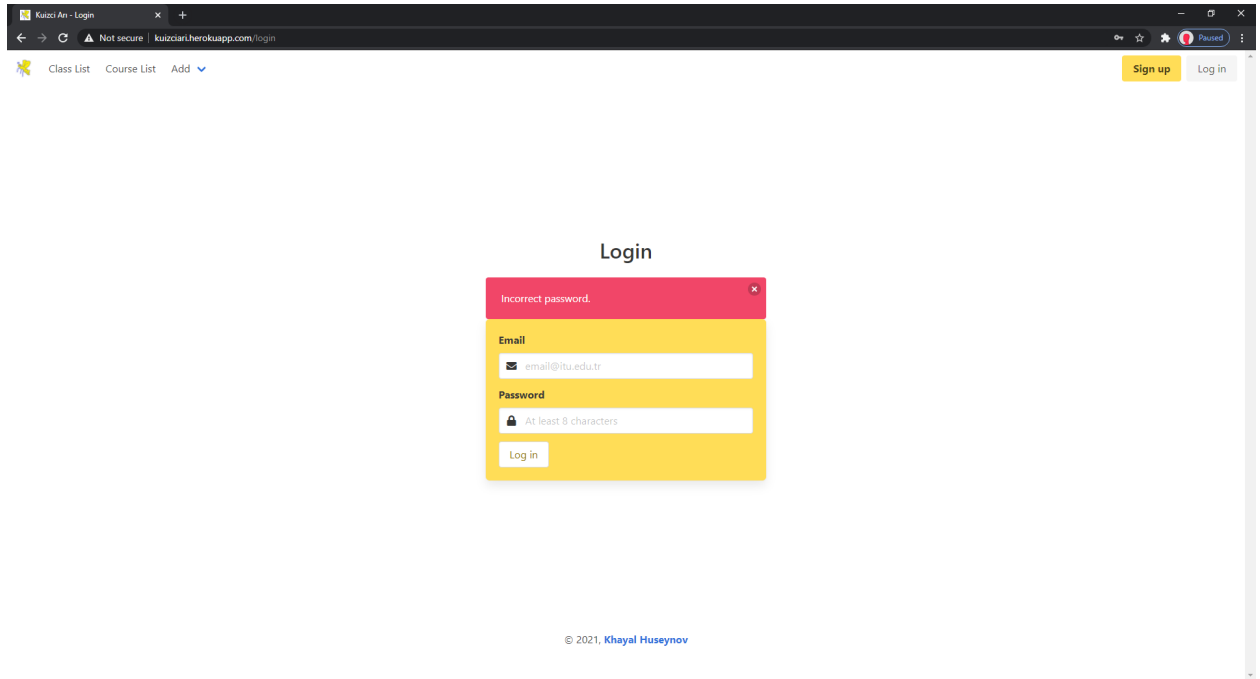


Figure 8: Login with Incorrect Password

E.2.2 Courses

Everyone is able to view the course list and individual course pages (Item 2)

Non-registered and registered users are able to view the course list and individual course pages.

Courses are listed and have additional information such as how many semesters they are in and how many classes have been taught of that course (Figure 9).

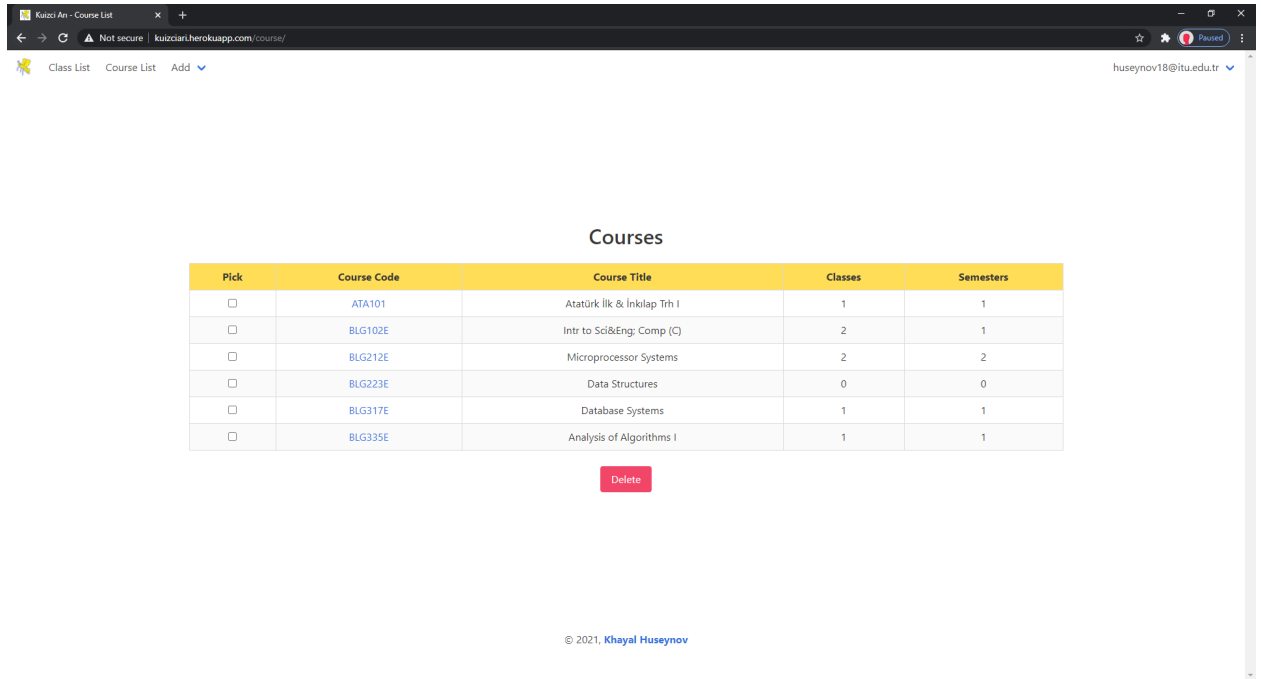


Figure 9: List of Courses

Course Code can be clicked to view detailed information about that course (Figure 10.

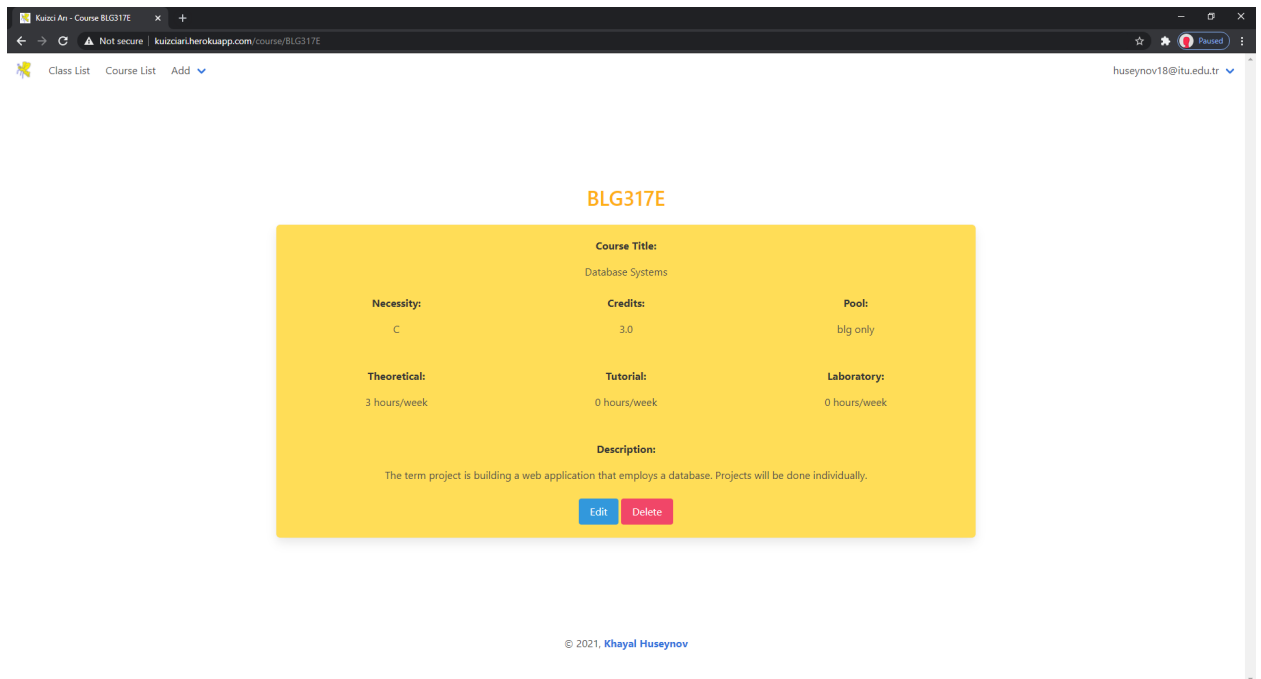
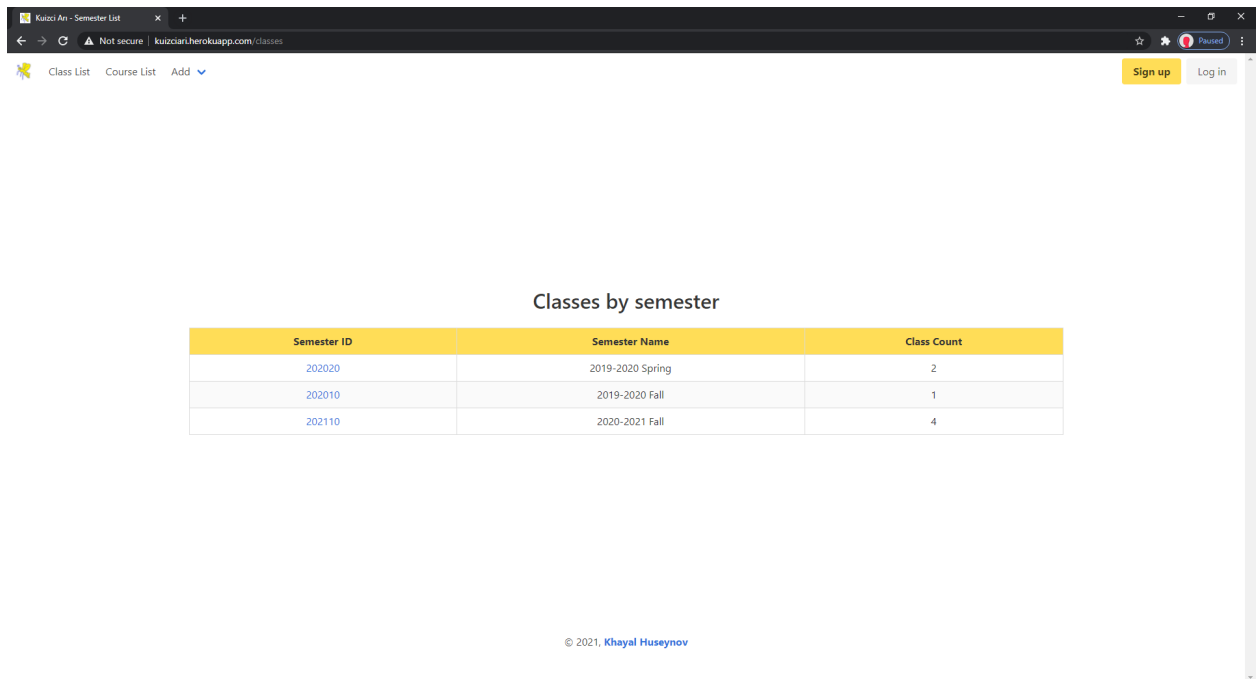


Figure 10: View of a Course

E.2.3 Classes

Everyone is able to view the class list grouped by semester and individual class pages which display courseworks (Item 3).

Registered or non-registered users can view available semesters and how many classes are in that semester (Figure 11).



Semester ID	Semester Name	Class Count
202020	2019-2020 Spring	2
202010	2019-2020 Fall	1
202110	2020-2021 Fall	4

© 2021, Khayal Huseynov

Figure 11: List of Semesters

Clicking on these semesters lists the classes in that semester (Figure 12).

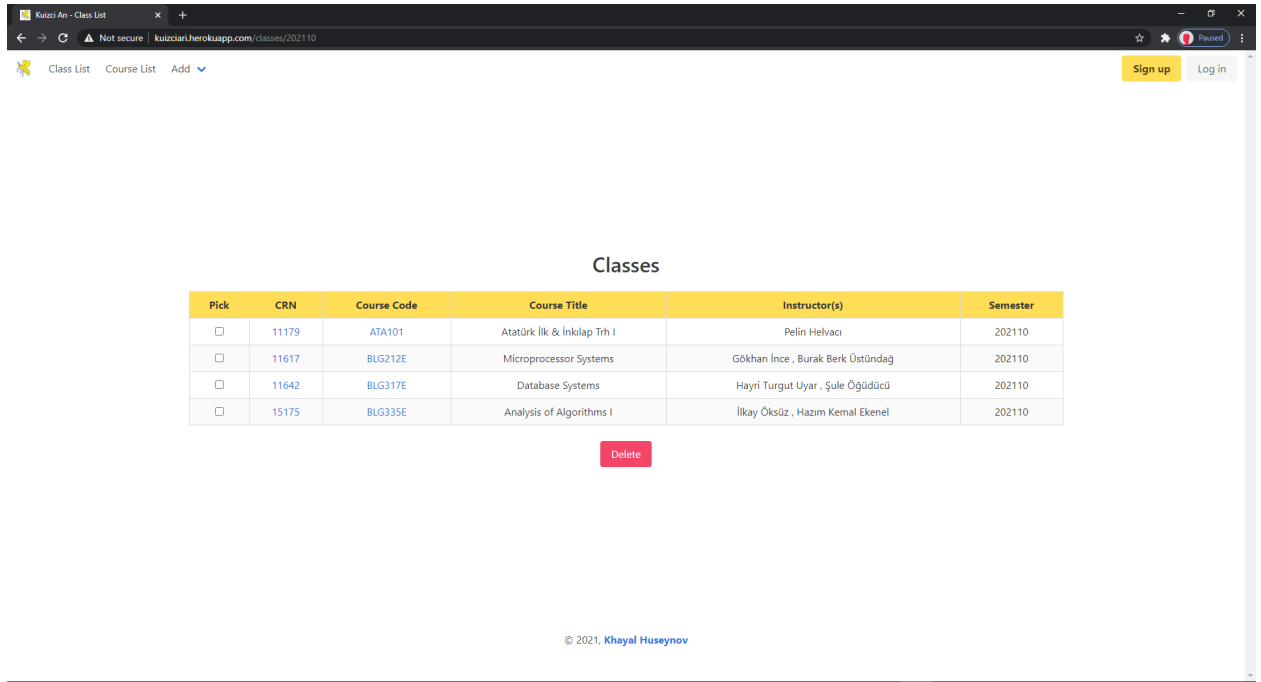


Figure 12: List of Classes

Clicking on a CRN will lead the user to the corresponding class page (Figure 13). Detailed information about the class can be viewed from this page.

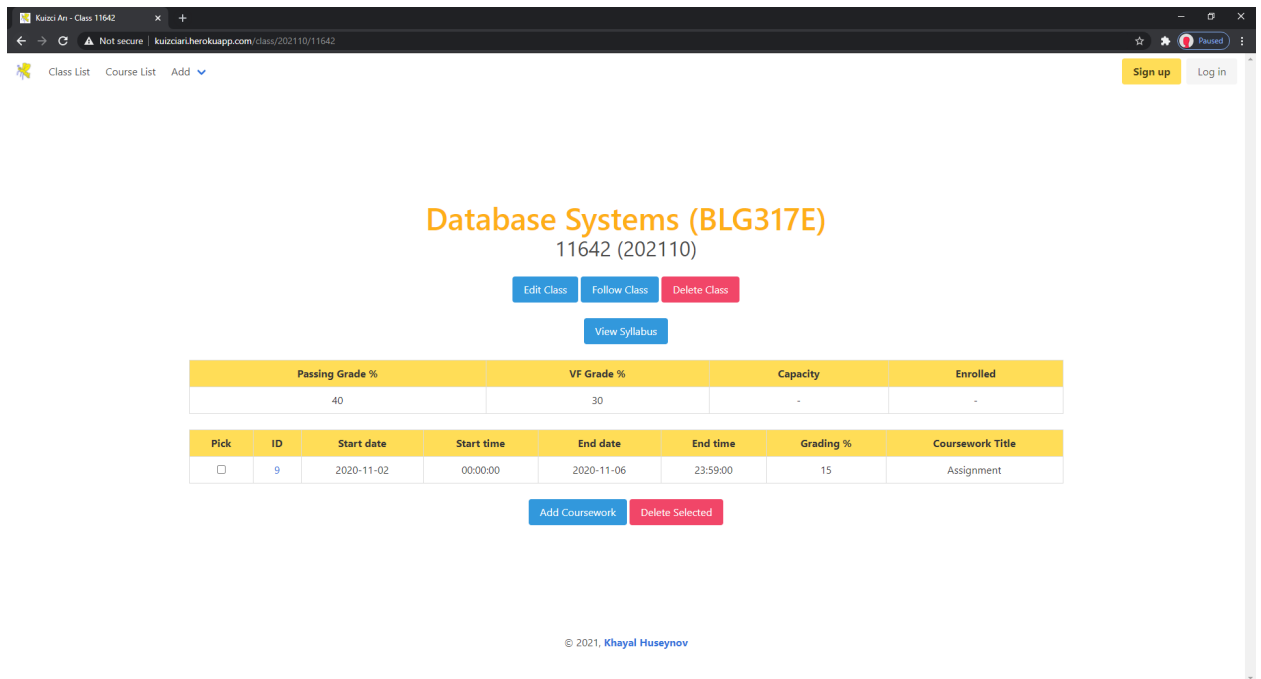


Figure 13: View of a Class

Application supports file uploads for syllabuses. The syllabus of a course can be viewed from the class page. If a syllabus has been uploaded, the button will light up and have text indicating this. Clicking on this button will lead the user to the syllabus (Figure 14).

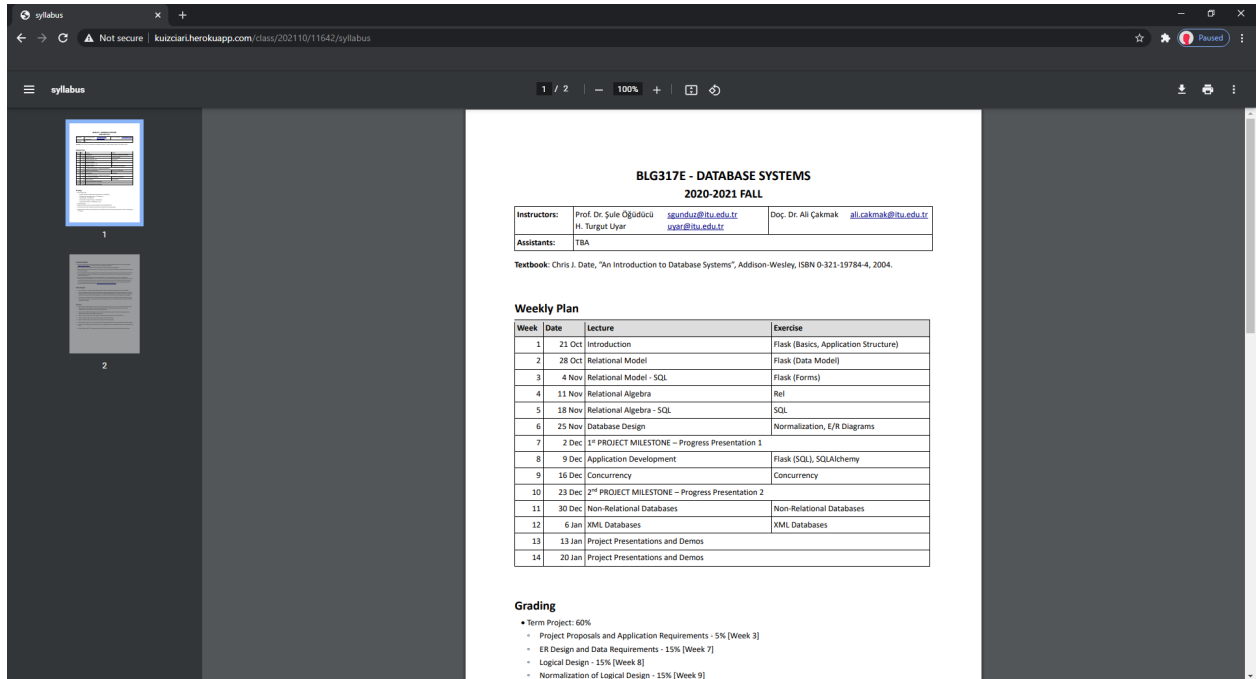


Figure 14: View of a Syllabus

If the syllabus is not available the button will not light up (Figure 15).

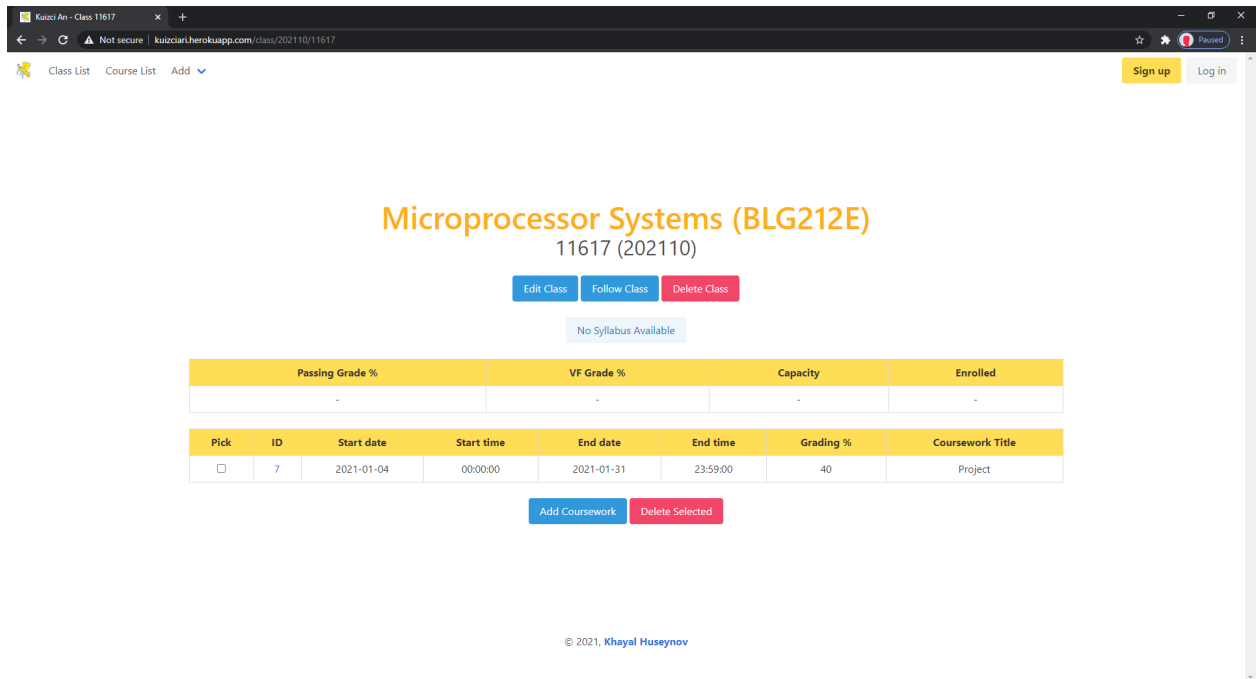


Figure 15: Class Page with no Syllabus

Information about the coursework of that class can be accessed from the the class page (Figure 16).

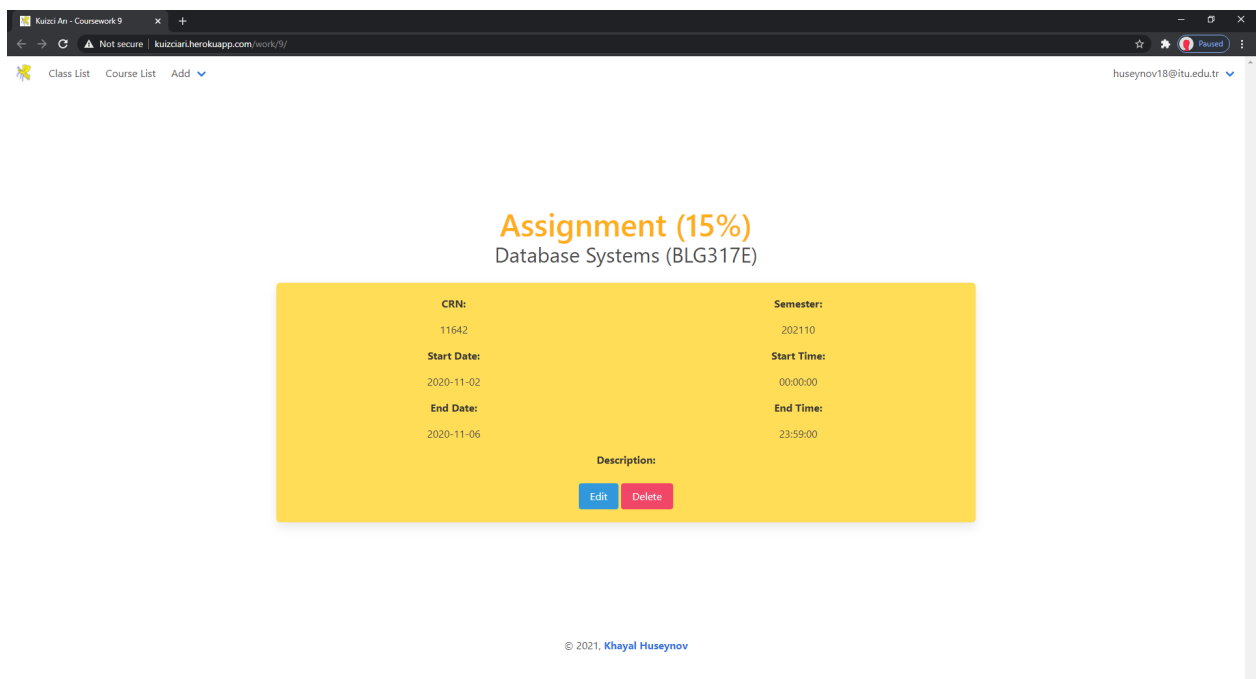
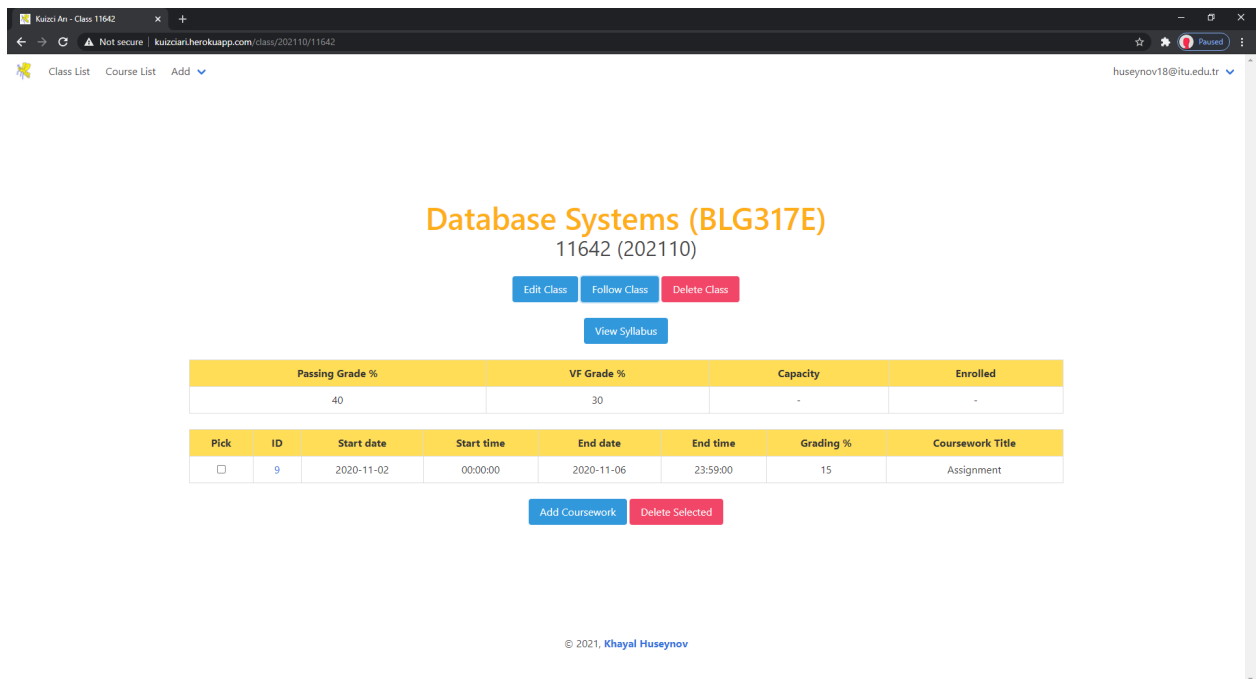


Figure 16: View of a Coursework

E.2.4 Following

Registered users (ITU members) are able to follow and unfollow the classes they desire (Item 4).

If the user is not logged in and the user presses the follow button on a class page, the user will be directed to the login page. If the user is logged in, the follow button will have an outline (Figure 17. If a logged in user clicks on the follow button, the application will store this and display the unfollow button instead of the follow button for that user (Figure 18. A success message will be displayed for following or unfollowing a class.



The screenshot shows a web browser window with the URL `kuizcari.herokuapp.com/class/202110/11642`. The page title is "Database Systems (BLG317E)" and the class ID is "11642 (202110)". There are four buttons: "Edit Class" (blue), "Follow Class" (blue with an outline), "Delete Class" (red), and "View Syllabus" (blue). Below these buttons is a table with the following data:

Passing Grade %	VF Grade %	Capacity	Enrolled
40	30	-	-

Pick	ID	Start date	Start time	End date	End time	Grading %	Coursework Title
<input type="checkbox"/>	9	2020-11-02	00:00:00	2020-11-06	23:59:00	15	Assignment

Below the table are two buttons: "Add Coursework" (blue) and "Delete Selected" (red). At the bottom, there is a copyright notice: "© 2021, Khayal Huseynov".

Figure 17: Page of a not followed Class

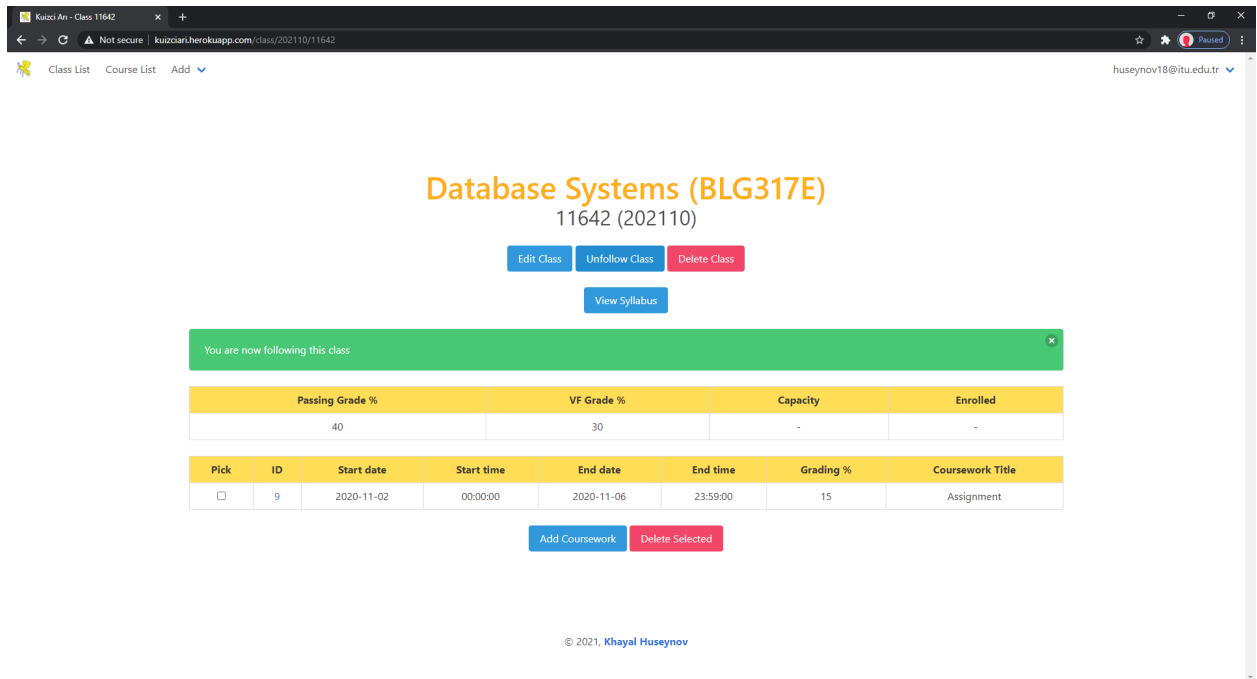


Figure 18: Page of a followed Class

E.2.5 Creation, Update and Deletion

Registered users (ITU members) are able to add, update, and delete classes, courses, and courseworks (Item 5).

Addition operation can be done through the navigation bar's add tab (Figure 19). Only a logged in user can press on this tab. Otherwise, the user will be prompted to log in (Figure 20).

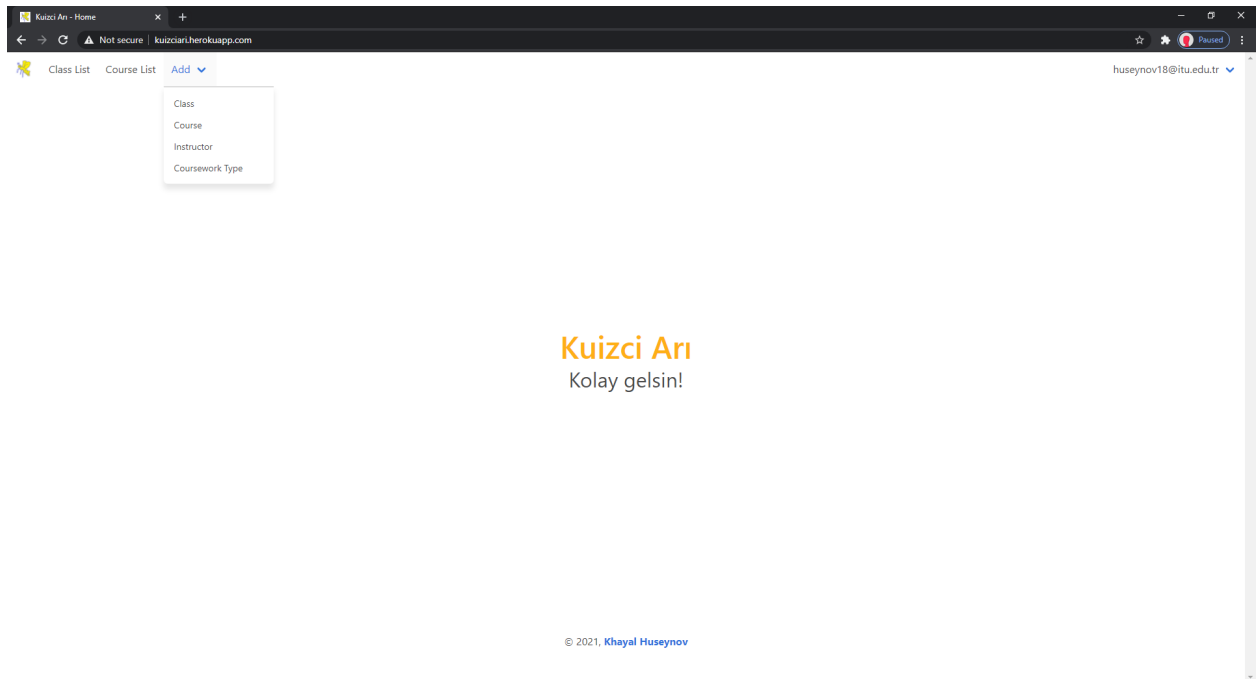


Figure 19: Navigation bar's addition tab

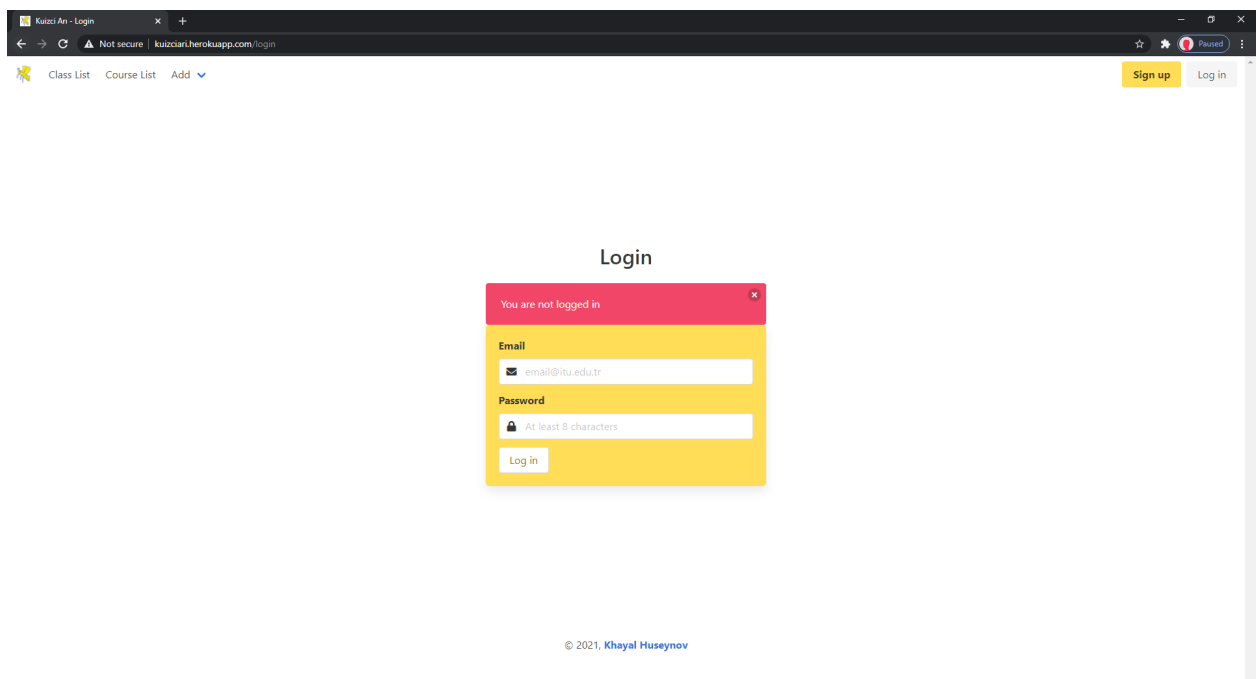


Figure 20: Login Page with Flash

A user should first add an instructor if it does not exist (Figure 21). Whether an instructor exists can be checked from the class addition page by clicking on Instructors field (Figure

22).

The screenshot shows a web browser window with the address bar displaying 'kuizciari.herokuapp.com/add/instructor'. The page title is 'Add instructor'. The main content area is a yellow box with the following fields:

- Instructor Name***: A text input field with a placeholder 'Full name of the instructor (max 255 characters) (ex. Hayri Turgut Uyar)'.
- Save**: A button at the bottom of the yellow box.

At the bottom of the page, there is a copyright notice: '© 2021, Khayal Huseynov'.

Figure 21: Instructor Addition Page

The screenshot shows a web browser window with the address bar displaying 'kuizciari.herokuapp.com/add/class'. The page title is 'Add/edit class'. The main content area is a yellow box with the following fields:

- CRN***: A text input field with a placeholder 'Class Reference Number (5 digits)'.
- Year***: A text input field with a placeholder 'For a 2018-2019 semester, enter latter value (ex. 2019)'.
- Season***: A dropdown menu.
- Course Code***: A dropdown menu.
- Instructor(s)***: A text input field with a placeholder 'Full name of instructor(s)'. Below it is a list of instructors: 'Hayri Turgut Uyar (1)', 'Şule Öğütücü (2)', 'Bora Döken (3)', 'İlkay Öksüz (4)', and 'Hazım Kemal Ekenel (5)'. The first instructor is selected.
- Choose a file...**: A button with a file icon.
- No file uploaded yet**: A text label.
- Save**: A button at the bottom of the yellow box.

At the bottom of the page, there is a copyright notice: '© 2021, Khayal Huseynov'.

Figure 22: Class Addition Page

Similarly, a user can check whether a course exists in the database by clicking on Courses

field of the class addition page (Figure 23). User can also manually go through the courses through the course list page (Figure 9). If the course does not exist, the user should add the course through course addition page (Figure 24).

The screenshot shows a web browser window with the address bar displaying 'kuizciari.herokuapp.com/add/class'. The page title is 'Add/edit class'. The form is titled 'Add/edit class' and contains the following fields:

- CRN***: Class Reference Number (5 digits)
- Year***: For a 2018-2019 semester, enter latter value (ex. 2019)
- Season***: Dropdown menu
- Course Code***: Dropdown menu with options: ATA101, BLG102E, BLG212E, BLG223E, BLG2317E
- Syllabus (PDF)**: Choose a file... button, No file uploaded yet
- Save**: Button

At the bottom of the page, there is a copyright notice: © 2021, Khayal Huseynov.

Figure 23: Class Addition Page

© 2021, Khayal Huseynov

Figure 24: Course Addition Page

After that, the user can finally add the class through class addition page (Figure 25).

© 2021, Khayal Huseynov

Figure 25: Class Addition Page

For coursework addition, user should add a relevant coursework type if the desired course-

work type does not exist (Figure 26). Then, the user can go to a class page and add coursework for that class (Figure 27).

The screenshot shows a web browser window with the address bar displaying 'kuizcari.herokuapp.com/add/courseworktype'. The page title is 'Kutoz An - Add Coursework Type'. The main heading is 'Add coursework type'. Below this, there is a yellow form with the following fields: 'Coursework Title*' with a placeholder 'Generic title of the coursework (ex. Final, Midterm)', 'Deadline Type*' with a dropdown menu, and a 'Save' button. The footer of the page shows '© 2021, Khayal Huseynov'.

Figure 26: Coursework Type Addition Page

The screenshot shows a web browser window with the address bar displaying 'kuizcari.herokuapp.com/class/202110/11642/addwork'. The page title is 'Kutoz An - Add Coursework'. The main heading is 'Add coursework'. Below this, there is a yellow form with the following fields: 'CRN' (11642), 'Semester' (202110), 'Course Code' (BLG317E), 'Instructor(s)' (Hayri Turgut Uyar (1), Şule Öğüdücü (2)), 'Start Date*' (dd/mm/yyyy), 'Start Time*' (dropdown), 'End Date*' (dd/mm/yyyy), 'End Time*' (dropdown), 'Grading(%) *' (Weighted percentage of the coursework), 'Coursework Type*' (dropdown), and 'Description' (Optional short description of the work (max. 255 characters)). There is a 'Save' button at the bottom.

Figure 27: Coursework Addition Page

The user can edit everything about a class (Figure 28), course (Figure 29), and coursework (Figure 30) by clicking the edit button on the page that they can view the corresponding information.

Add/edit class

CRN*
11642

Year*
2021

Season*
Fall

Course Code*
BLG317E

Instructor(s)*
Hayri Turgut Uyar (1) Şule Öğüdücü (2)

Quota
Max quota of class
40

Enrolled
Amount of students enrolled to class
30

Passing Grade
40

VF Grade
30

Syllabus (PDF)
Choose a file... No file uploaded yet

Save

© 2021, Khayal Huseynov

Figure 28: Class Edit Page

Add/Edit course

Faculty Code*
BLG

Course Number*
317

Language*
English Turkish (or Other)

Course Title*
Database Systems

Necessity
Compulsory Elective

Credits*
3.0

Pool
blg only

Description
The term project is building a web application that employs a database. Projects will be done individually.

Theoretical*
3

Tutorial*
0

Laboratory*
0

Save

© 2021, Khayal Huseynov

Figure 29: Course Edit Page

The screenshot shows a web browser window with the address bar displaying 'kuizcari.herokuapp.com/work/9/edit'. The page title is 'Add coursework'. The form is a yellow box with the following fields:

- CRN**: 11642
- Semester**: 202110
- Course Code**: BLG317E
- Instructor(s)**: Hayri Turgut Uyar (1) Şule Öğüdücü (2)
- Start Date***: 02/11/2020
- Start Time***: 00:00
- End Date***: 06/11/2020
- End Time***: 23:59
- Grading(%) ***: 15
- Coursework Type***: Assignment (2)
- Description**: Optional short description of the work (max. 255 characters)

A 'Save' button is located at the bottom left of the form.

Figure 30: Coursework Edit Page

E.2.6 Profile

Followed classes are displayed on a user's personal private page (Figure 31). Courseworks of these followed classes are displayed on the user's another personal private page (Figure 32). Courseworks are sorted by deadline time on this page. (Item 6)

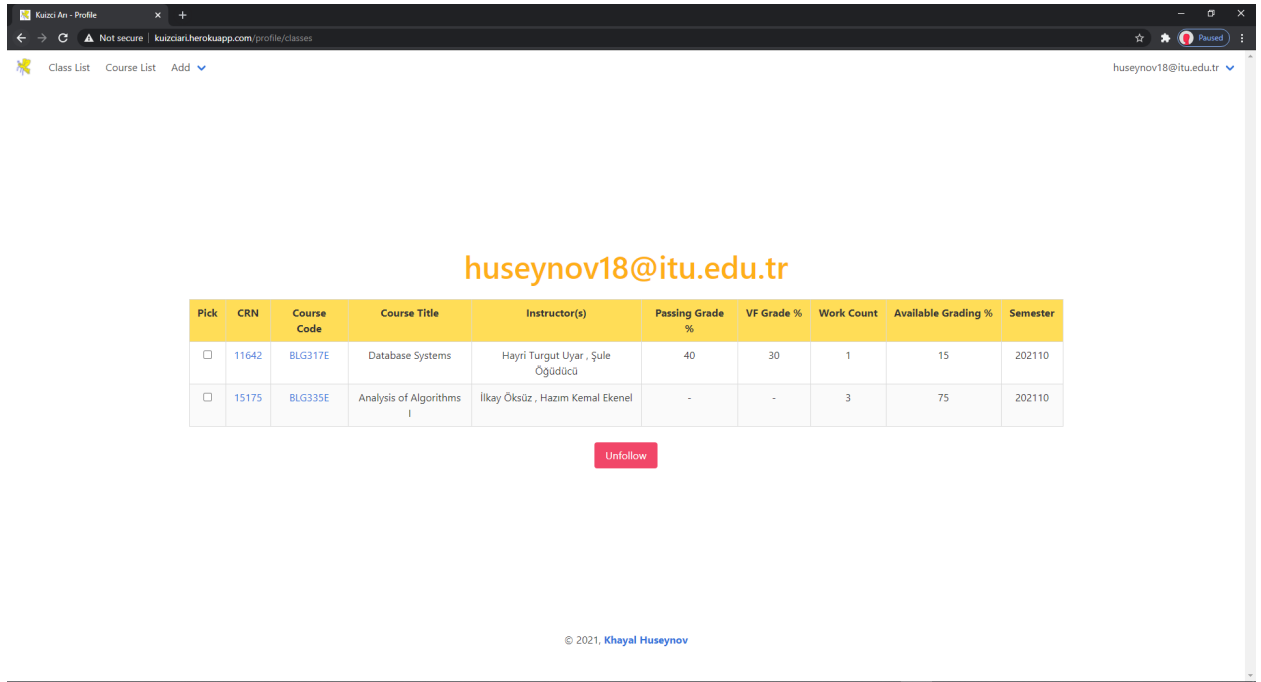


Figure 31: User Profile of Classes

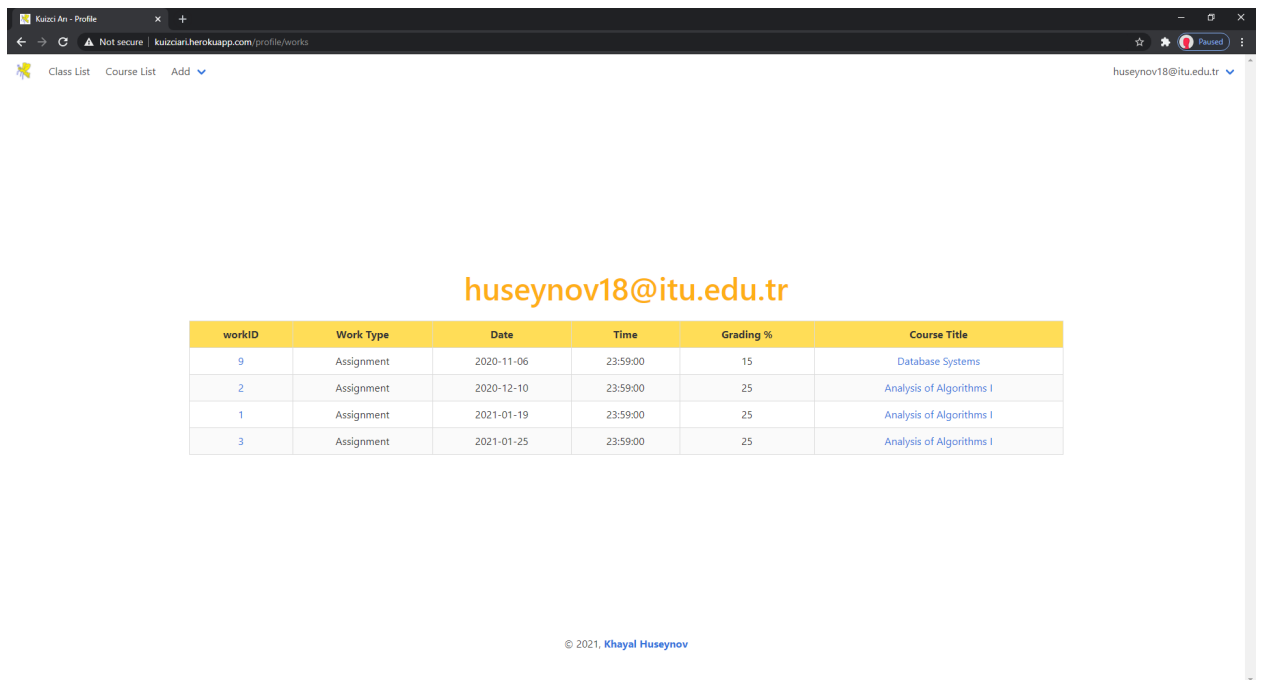


Figure 32: User Profile of Courseworks

E.3 Installation Manual

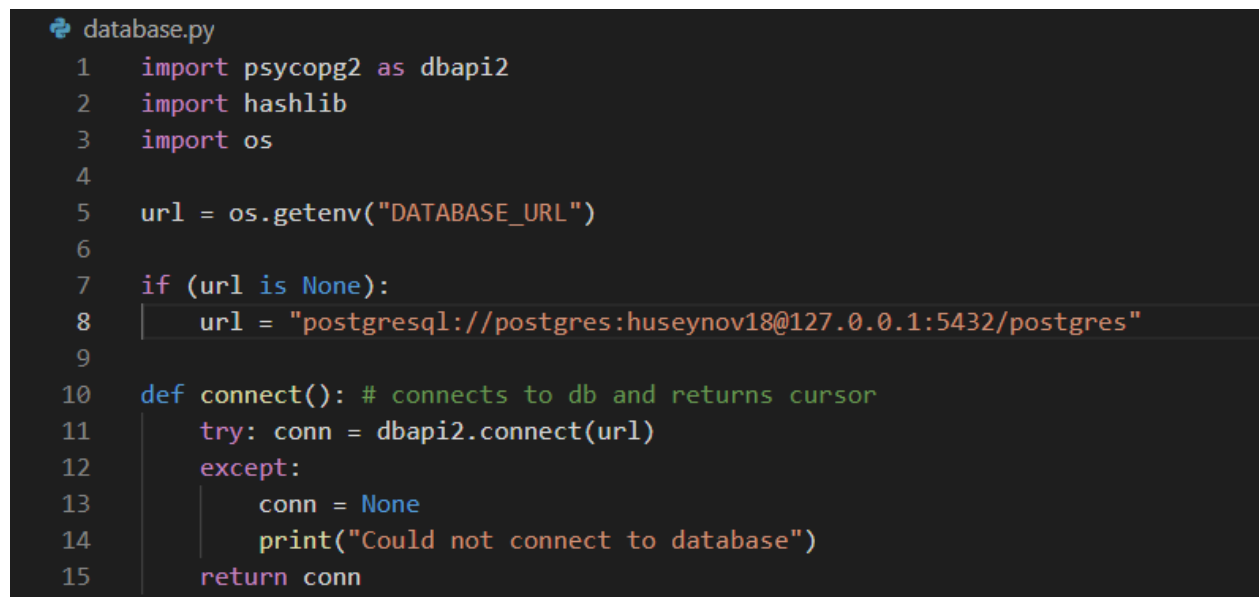
The source code of the application is hosted on GitHub and can be accessed by clicking [here](#).

E.3.1 Localhost

In order to deploy the application from console, only the database URL inside database.py need be updated. Line 8 on Figure 33 needs to be updated with the hosted database address.

URL format can be seen from below:

```
postgresql://username:password@address:port/dbname
```



```
database.py
1  import psycopg2 as dbapi2
2  import hashlib
3  import os
4
5  url = os.getenv("DATABASE_URL")
6
7  if (url is None):
8      url = "postgresql://postgres:huseynov18@127.0.0.1:5432/postgres"
9
10 def connect(): # connects to db and returns cursor
11     try: conn = dbapi2.connect(url)
12     except:
13         conn = None
14         print("Could not connect to database")
15     return conn
```

Figure 33: Line that needs to be updated

The application can be started with the following command from console:

```
python server.py
```

E.3.2 Heroku

In order to deploy the application from Heroku, before deployment Heroku Postgres add-on needs to be added. No extra step is required for this part, application will recreate tables

if they do not exist. Additionally, all detected dynos need to be activated manually from the Heroku dashboard if they are deactivated.