Senior Project

Design Document

Drop-in Tutoring Web Application

Team Members:

Timothy Baker, Jeremiah Caban, Eric Ray Carpizo, Aanchal Chaturvedi, Huy Ly, Christopher Mariani, and Bryan Nunez

|  |  |
| --- | --- |
| Table of Contents | Page |
| 1. Summary |  |
| 1. Technology Stack    1. HTML / CSS    2. JavaScript    3. PHP    4. Java |  |
| 1. Application Views |  |
| * 1. Home   2. Log In   3. Drop-in Tutoring Schedule   4. Survey/Feedback Form   5. Navigation Bar   6. Tutor Session Logbook   7. Shared Knowledgebase   8. Session Report Page |  |
| 1. RESTful API |  |
| 1. Database Design |  |
| 1. Implementation Plan    1. Features: Log-in, API implementation    2. Views: Home, Log-in, Navigation bar    3. Database instantiation |  |

Summary

The group will design and create a web application that will assist various departments with the drop-in tutoring services that they offer. Tutors themselves will be able to log in to access a session logbook, which will help keep track of each individual tutoring session as they take place. They will also have access to a shared knowledgebase for passing back and forth useful information to other tutors as they see fit. Admins, typically a departmental chair or secretary, will be able to log in to view detailed reports of all sessions that have been held over any given period. The system will allow for admins to easily add new tutors or courses to the schedule to keep up with demand as time goes on. Basic features (no login required) will be made available for students, these include: home screen with scrolling news panel, survey/feedback form, and a schedule viewer for the drop-in tutoring services.

Technology Stack

# Bootstrap (HTML / CSS / JavaScript)

The team will utilize the open source Bootstrap framework to build the front-end of the application. Bootstrap is a CDN-accessible library containing predesigned stylesheets as well as JavaScript files.

# PHP

To communicate with the SQL database, the PHP scripting language will be used to perform the backend operations.

# MySQL

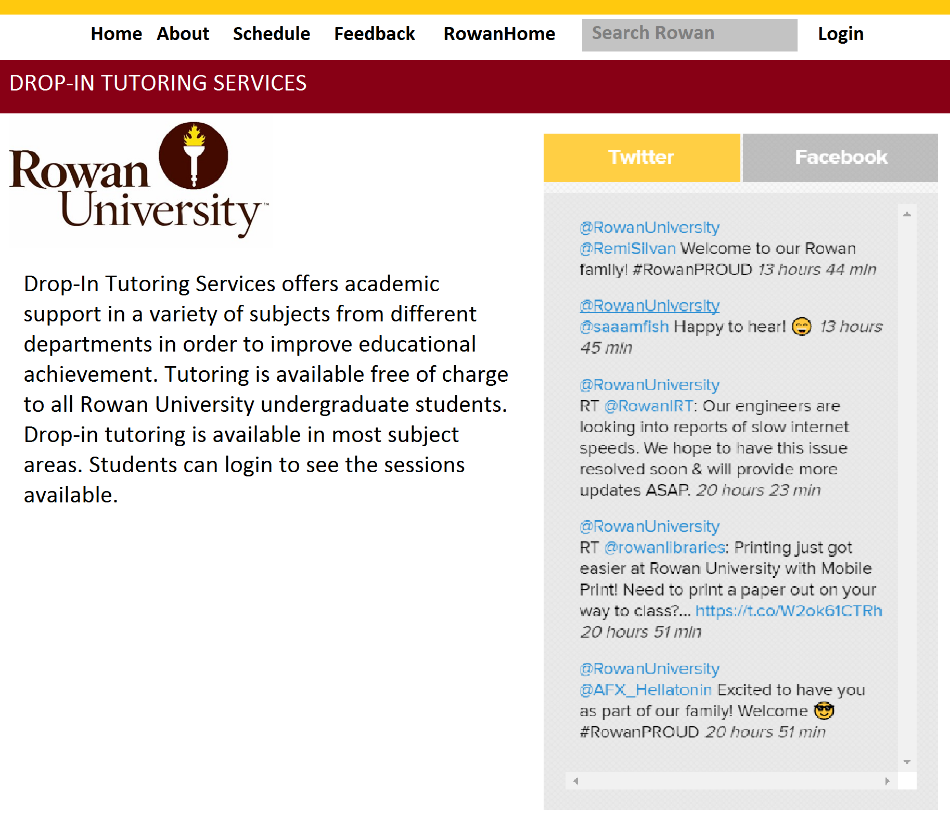
A MySQL database will be used for the storage of all data relevant to the application.

# Amazon Web Services

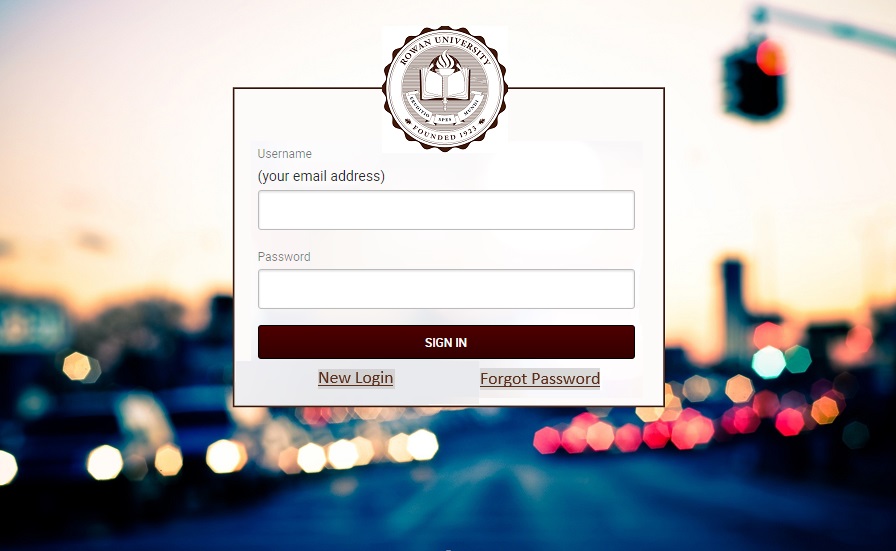
An AWS server will serve as the working environment for which the application will be designed in

Application Views

# Home



# Log In



# Drop-in Tutoring Schedule

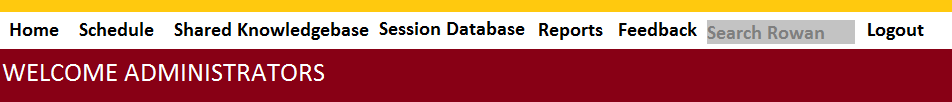


# Survey/Feedback Form

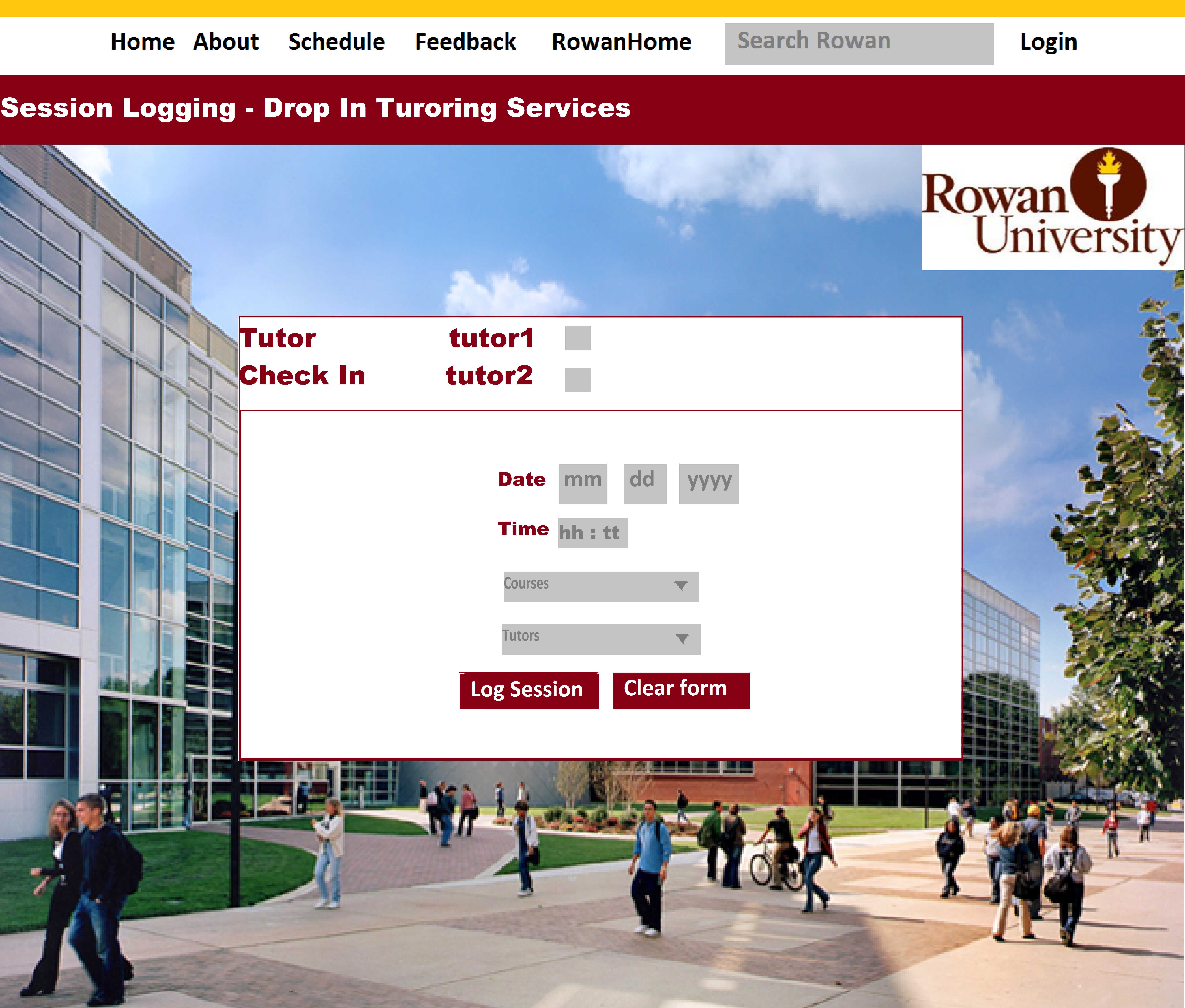


# Navigation Bar

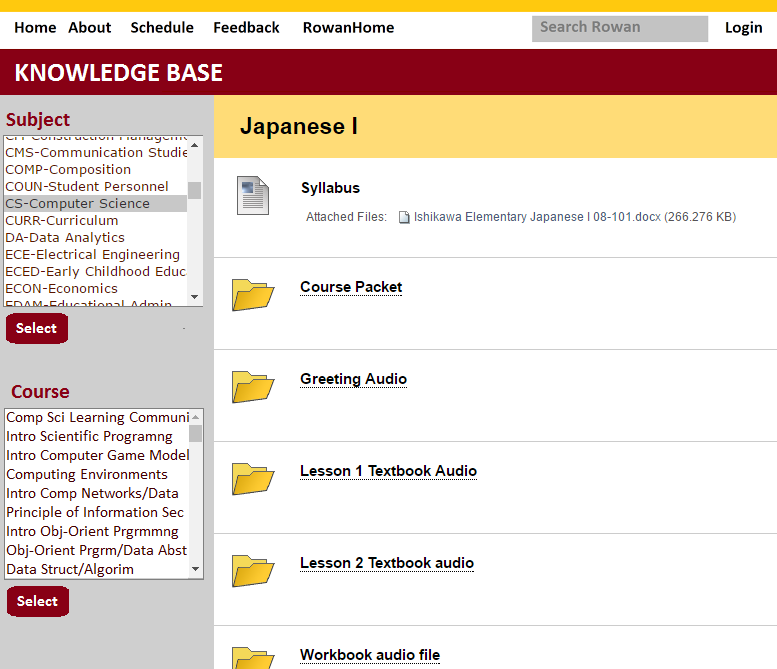




# Tutor Session Logbook



# Shared Knowledgebase



# Session Report Page



RESTful API

***Home***

**Endpoint: GET /home/getAnnouncements/:code**

Purpose: Retrieves Announcements for displaying on every end users Homepage. Can return the first 15 announcements (if input is zero) or all stored announcements (if input is one).

Input:

code: path parameter

Output: JSON array of:

announcement-number : Integer

announcement : String

Implementation:

    Construct a list from the announcements table starting from the bottom of the table and working up until desired number of announcements is reached (based on input).

**Endpoint: POST /home/addAnnouncement**

Purpose: Sends an announcement to the database. Functionality is only available to administrative users.

   Input:

announcement : HTML POST parameter

    Output:

confirmation : boolean

Implementation:

The announcement string will be part of the HTML POST request and be added to announcements table.

***User Login***

**Endpoint: GET /users/check/:username**

Purpose: Checks to see if a user exists in the database

Input:

username : path parameter

Output:

confirmation : boolean

Implementation:

Checks the user table to see if the username can be found. Returns true if so and false if not.

**Endpoint: GET /users/type/:username**

Purpose: Checks to see if a user is a tutor or admin.

Input:

username : path parameter

Output:

user-type : String

Implementation:

Finds the user in the user table and checks the value of the admin column and then returns the appropriate string.

    Endpoint:

Purpose: Check Token?

    Input:

    Output:

    Implementation:

Endpoint:

Purpose: Validates whether an end user's session requires another login or not

    Input:

    Output:

    Implementation:

    Endpoint:

Purpose: Creates a token for validation.

    Input:

    Output:

    Implementation:

    Endpoint:

Purpose: Destroys a token

    Input:

    Output:

    Implementation:

***Schedule***

**Endpoint: GET /courses/getall**

Purpose: Returns a list of courses available

Input:

None

Output: JSON array of:

course-number : Integer

course-name : String

course-department : String

Implementation:

Gets the information from the Course Table in the database, constructs the JSON object

**Endpoint: GET /courses/times/:course**

Purpose: Returns a list of times a course is tutored

Input:

course : path parameter

Output: JSON array of:

start time : String

start time : String

Implementation:

Use the course table to identify course ID then use the TimeSlot table’s startTime and endTime columns to create the JSON objects.

**Endpoint: GET /courses/getAt**

Purpose: Returns a list of courses available a specified time

Input:

startTime : HTML POST parameter

endTime : HTML POST parameter

Output: JSON array of:

course-name : String

course-location : String

Implementation:

Use the TimeSlot table to identify which course IDs fall within the specified start and end time. Use those course IDs with the Courses table to construct a list of course names.

**Endpoint: GET /tutors/getAt**

Purpose: Returns a list of tutors available a specified time

Input:

startTime : HTML POST parameter

endTime : HTML POST parameter

Output: JSON array of:

tutor-name : String

Implementation:

Use the TimeSlot table to identify which course IDs fall within the specified start and end time. Use those course IDs with the TutorTimeSlots table to get userIDs that will be used with the Users Table to get each tutors name.

**Endpoint: GET /tutor/available/:tutorname**

Purpose: Returns whether a tutor is currently available.

Input:

tutorname : path parameter

Output:

available : boolean

Implementation:

Use the tutorname with the User’s table to get the tutor’s ID. Then use the tutor’s ID with the TutorTimeSlots Table to get all the timeslot IDs for that tutor. Use the timeslot ID’s with the TimeSlot’s table to see if the current time falls between the startTime or endTime of any of those timeslots.

**SessionLoginTutor**

**Endpoint: GET /SessionLoginTUTOR/getCheckedIn/: tutorID**

Purpose: Returns the clocked in status of a tutor as seen from the server.

    Input:

tutorID : path parameter

    Output:

currentlyActive : boolean

    Implementation:

Checks the Tutor Table using a tutors ID to see if they clocked in. A tutor who is currently clocked in returns a true value, and being clocked out returns a false value.

**Endpoint: POST /SessionLoginTUTOR/toggleCheckedIn/:tutorID**

Purpose:

    Input:

tutorID : path parameter

    Output:

None

    Implementation:

Create a punch card with the tutor ID and add it to the punch card table in the database. Populate time stamp with the current time. Set checked-in to true.

**Endpoint: GET /SessionLoginTUTOR/getTutors/:courseID**

Purpose: Returns a list of tutors belonging to a subject.

    Input:

courseID : path parameter

    Output: JSON array of

firstName : String

lastName : String

    Implementation:

Matches a course ID with the Course table and returns a list of User ID’s of entities which tutor the subject.

**SessionLoginAdmin**

**Endpoint: GET /SessionLoginADMIN/getCheckins/:**

Purpose: Returns the tutors which have clocked in. Functionality is only available to administrative users.

    Input:

None

    Output: JSON array of

firstName : String

lastName : String

    Implementation:

Constructs a list from the Punch card Table. If any given punch card has its checked-in value set to true, add it to the List.

**Endpoint: POST /SessionLoginADMIN/setTime/:punchcardID, timestamp**

Purpose : Sets the time associated with a punch card to another value in case a tutor made an error.

    Input:

punchcardID : path parameter

timestamp : path parameter

    Output:

None

    Implementation:

Match the punch card ID with one available from the Punch card Table. Change the Time stamp associated with the punch card ID.

**Knowledge Base**

**Endpoint: GET SessionLoginTutor/KnowledgeBase/getFolders/:**

Purpose: Returns a sorted list of course IDs.

    Input:

None

    Output: JSON array of

courseName : String

    Implementation: Construct a list of folders and sort them in alphabetical order.

**Endpoint: GET SessionLoginTutor/KnowledgeBase/getFile/: fileID**

Purpose: Returns a file from the KnowledgeFiles table.

    Input:

fileID : path parameter

    Output:

content : Longtext

    Implementation: Match the file ID with one from the Knowledge Files Table. The content is then displayed if it has been approved by an administrator.

**Endpoint: GET SessionLoginTutor/KnowledgeBase/getFiles/: courseID**

Purpose: Returns all file ID’s associated with a course ID.

    Input:

courseID: path Parameter

    Output: JSON array of

fileName : String

    Implementation: Constructs a list of Files from the Knowledge Files table. If any given entry has a corresponding course ID and has been approved, add it to the list.

**Endpoint: POST SessionLoginTutor/KnowledgeBase/setFiles/: fileID, contents**

Purpose: Replaces content held in a file.

    Input:

fileID : path parameter

contents : path parameter

    Output:

None

    Implementation:

Matches a file ID with one available in the Knowledge Files table. Content held in the table is then replaced by the content parameter.

**Endpoint: POST SessionLoginTutor/KnowledgeBase/addFolder/: courseID**

Purpose: Adds a new course to the knowledge Files table. Functionality is only given to administrators.

    Input:

courseID: path parameter

    Output:

None

    Implementation:

If no instance of the course ID exists in the Knowledge Files table, call addFile and add the file and course ID at the same time.

**Endpoint: POST SessionLoginTutor/KnowledgeBase/addFile/: courseID, tutorID, contents**

Purpose: Adds a file to the Knowledgebase Files Table.

    Input:

courseID : path parameter

tutorID : path parameter

contents : path parameter

    Output:

None

    Implementation:

Constructs a new file with the author (a tutor’s) ID. Course ID’s already existing in the Knowledge File Table are the only available options for this field. Tutor ID is populated with the current end-users ID. Once a course ID, tutor ID and contents are submitted to the Knowledge File Table, they are set with an approved value of false and a file ID. An administrator will then have to approve the file contents, which will change the approved value to true.

**Endpoint: POST SessionLoginTutor/KnowledgeBase/rm Folder/: courseID**

Purpose: Removes a course ID from the Knowledge File table. Functionality is only given to administrators

    Input:

courseID : path parameter

    Output:

None

    Implementation:

Matches an inputted course id with entities in the Knowledge Files table. Once matched, these entities are removed from the Knowledge Files table.

**Endpoint: POST SessionLoginTutor/KnowledgeBase/rm File/: fileID**

Purpose: Removes a file ID from the Knowledge File table.

    Input:

fileID: path parameter

    Output:

None

    Implementation:

Matches an inputted file id to one if the Knowledge File table, if it exists. Once matched, this entity is removed from the Knowledge File Table

**Feedback**

**Endpoint: GET /feedback/getSurveys**

Purpose: Returns all the surveys

Input:

Output:

Implementation:

**Endpoint: GET /feedback/receiveNotifications**

Purpose:

Input:

Output:

Implementation:

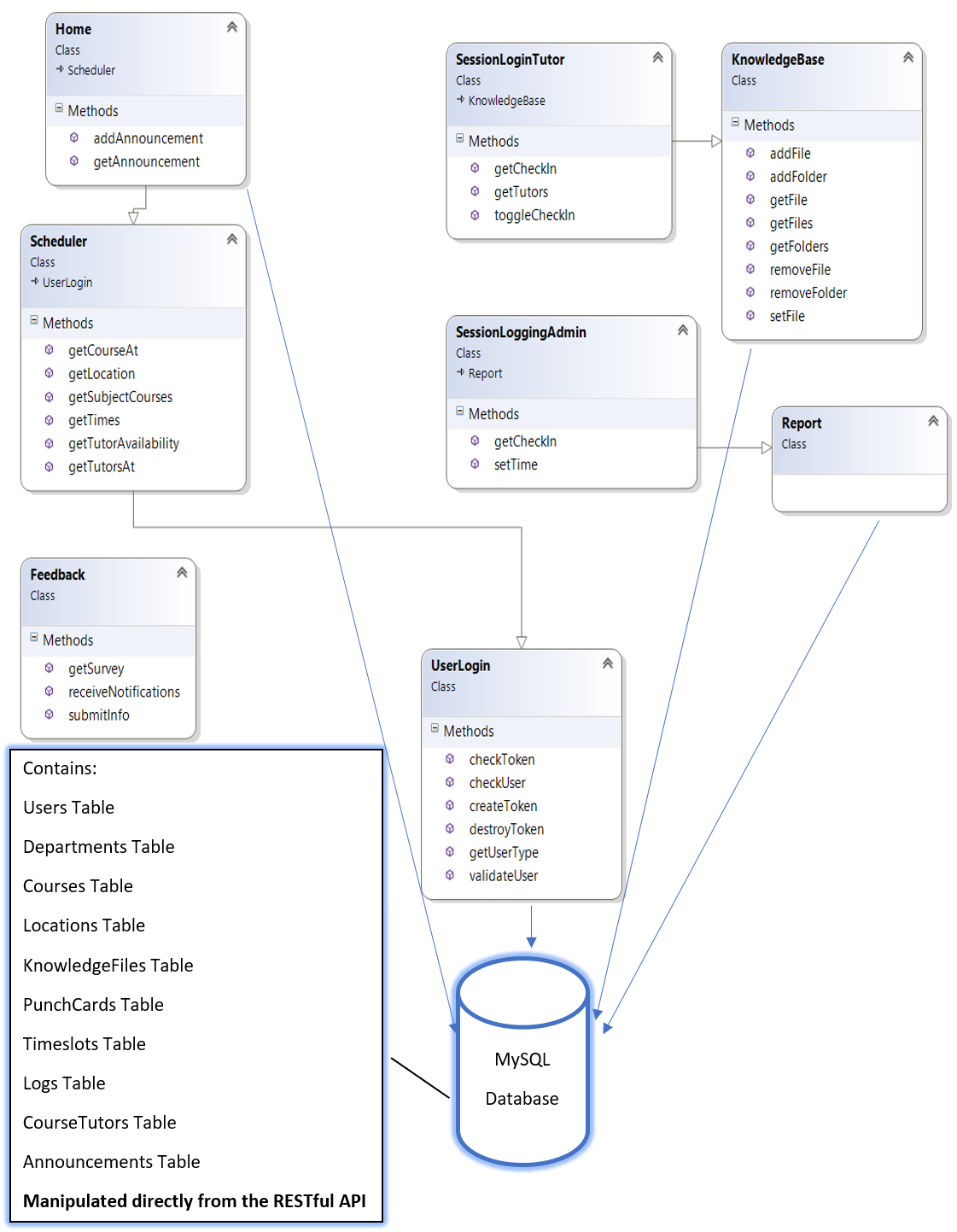
**Endpoints: POST /feedback/submitInfo**

Purpose:

Input:

Output:

# Class Diagram



All endpoints in the RESTful service. For each endpoint, provide all inputs and outputs. Reflect briefly (in pseudo code) on how you will implement the endpoint. As an example, consider the “cities in a country” endpoint in my sample code. This might be written up as follows:  
  
Endpoint: POST /cities/:country  
Purpose: Returns cities in a given country  
Inputs:  
        country name: path parameter  
Output: JSON array of  
        city-name: String  
        population: integer  
Implementation:  
        Join the city and country tables listed in the database section and retrieve desired information

Database Design





Implementation Plan

Write am implementation plan for the first two weeks in the implementation phase. This must include all the action items for this time period and an assignment of responsibilities for individual team members. Note that I will use your writeup to assess how you have progress at the midway point in the implementation phase.

Features: Log-in, API implementation

Views: Home, Log-in, Navigation bar

Database instantiation