

CodeBuddies

CPSC 471 Final Report

Group 44 members: Zhifan Li, Tianfan Zhou, Taimoor Abrar

Dec 17th, 2021

Abstract:

In today's day and age of education filled with remote learning, seeking help online has proven to be a challenge for most students in the field of programming. Our project has been meticulously designed to tackle the issue at hand, thereby providing remote and personalised coaching for learning programmers. We created a web application, tailored specifically for online tutoring and helping upcoming developers. Code Buddies has been created using php, javascript,html, css and sql to create a simple but elegant online tutoring platform. Three team members have worked tirelessly throughout the semester, one on the databases, one on the integration and one on the design. In order to bring this solution to life and help students with the arduous task of programming problems. The various features such as an email system and personalized course portfolios have certainly been fun to develop and prove to be extremely useful as well. It has been an amazing learning experience for us as a group as well, familiarizing ourselves with the in's and outs of databases and in the process creating a cool web application.

Introduction

Many individuals in today's day and age wish to study programming, and many novices in the field think that learning to program in a one-on-one environment can enhance their learning. Alongside the individuals in question have greater motivation to pursue the task at hand while someone is constantly coaching them. At times there are tricky problems that junior programmers are unable to solve by themselves as well. Some of these individuals would prefer a one-to-one session with seasoned programmers in order to better understand and solve these problems. The solution we have devised is to provide clients with one on one tutoring sessions. This is to aid them when they struggle with a programming problem or if they need further

clarification on high yield programming topics. In today's remote technological age, most businesses are going online and that is exactly how we intend to conduct business. We are going to build a web app called Code Buddies. Code Buddies will be a website that will enable our clients to make a variety of choices; to make orders, select appointments, and talk to their mentors. And we, as the administrative team of the website, will handle these orders and requests catering to all of our clients' needs.

Problem definition

The problem at hand is the lack of human involvement in programming instruction in today's world of tech. We have platforms like Udemy or Coursera where you can learn programming at your own pace, however, this brings students' motivation into question. Self-guided courses tend to lose their students really easily due to a lack of human involvement. If you are in a dynamic learning environment with help constantly available and deadlines to meet, you learn not out of interest but out of necessity as well. We as mentors can offer help to students in order to improve their programming skills and help them avoid common mistakes and errors in programming. Our solution caters to those individuals who seek one on one help with their learning in the field of development. There are many people who do not want to research intricate topics and browse stack overflow for hours looking for a solution. Instead, they want someone to teach them in a one-on-one format, as a tutor does to his pupil. There are similar systems to ours; some training centers for coding, and there are many of these service providers across the world such as Lighthouse Labs. There are many possible improvements to our current proposal, which we will implement after our beta phase. Some of these could be; flexible appointment times, flexible learning choices (languages, etc), and flexible appointment locations (remote or in-person): a cafe, a library, or even online meeting platforms.

Project Design Section:

CodeBuddies is a web application that allows students who want to pursue a career in Computer Science or Software Engineering to take courses with our CodeBuddies Mentors by submitting requests, ask for appointments. Our admin will approve these requests and confirm an appointment date with the client, and contact them by sending them email through the website instead of using their cellphones.

- If you log in as a student: you can see a list of courses, list of teachers, a list of the courses that you have ordered, a list of appointments that you have with your mentor, an area for updating your personal information, and at the end there is a form for submitting a request to take a course.
- First you can view the courses and teachers, then go to the end of the page to submit a request if you want to take a course.
- By entering your username, email, teacher's name and course name, then clicking submit, this request will be shown on the admin's end.
- Now, for the purpose of testing, you can log out and log in as an admin
- You can register as an admin by creating a user with username as admin with any password you like.
- As admin, you can first see a form for sending emails, and a list of orders and requests, list of teachers and list of courses, you can view the users and have full access to them, at the end there is a form for creating appointments with students.
- Now, you will see there is a new request and new order listed at the same time. The list of requests is for recording the requests that the student has sent, and the admin can delete it, the purpose of request is just for the admin to look up, nothing else. The list of

orders is for the admin to approve or reject: the last column of the table has two icons: first is approving the request, second one is rejecting it. Once the admin approves the order, this order will be added to the student's list of ordered courses. Once the admin ignores the order, it will be removed from the order list from both admin and student sides.

- Now you can approve that order, and log out
- Now login as a student, you will see that there's a new row added at the List of Courses You Have section. That is the course that you just ordered.
- Now since the course is ordered, the admin can login and at the Send Email section: Send an email to student for asking appointments. Now they can communicate through email, which is a faster and safer way since the message is encrypted on both ends.
- Once they confirm the date and other stuff. Admin can use the Add Appointment feature located at the end of the admin's website.
- Admin will enter the student's email, name, teacher's name, course and date, then click submit. This appointment information will be added to both Admin's Web UI, and normal student's Web UI.
- Now once the student is logged in, the appointment that the admin added will be shown to the student.
- This website has error checking functionalities, so everytime you put some wrong input, it will prompt you to enter again.
- These are the full functionality of this website. This will also be included in the user manual at the last section of this report.

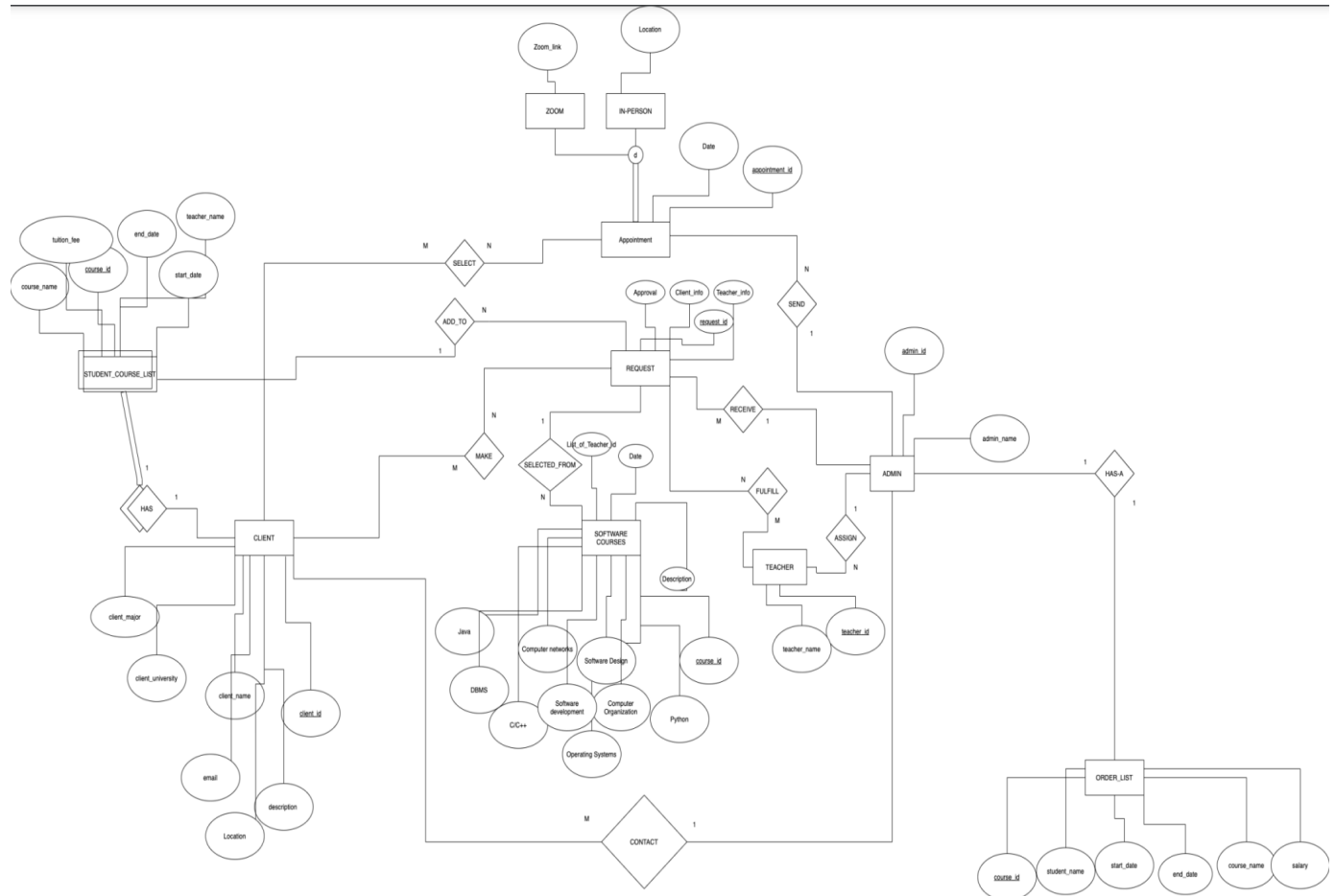
Outcomes:

1. Allow these students to improve their programming skills and related knowledge of the field.

This is so students can have a better understanding of the strategies that are involved in being a good programmer.

2. This project can produce commercial value if a service fee is required for these sessions. Our mentors can make a profit.

CodeBuddies ER Diagram



change log of ER diagram:

Attribute course_id added as primary key of software_courses entity

Attribute list_of_teacher_id is removed from the software_courses entity

Attribute description is removed from the software_courses entity

Attribute date is removed from the software_courses entity

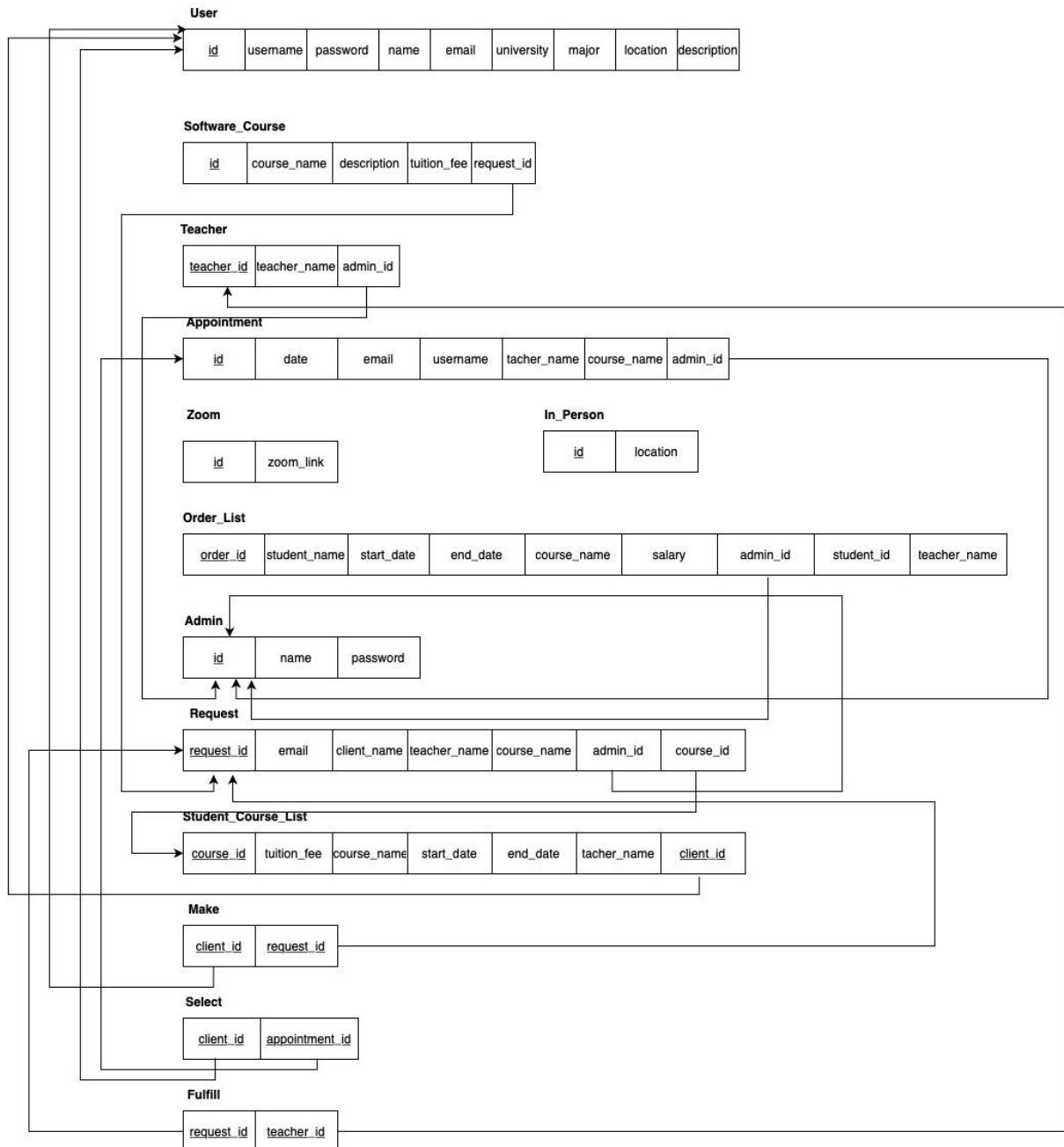
Attribute location is removed from appointment entity

Attribute zoom_link is removed from appointment entity

Attribute Approval is removed from request entity

Implementation Section:

RM Model:



Assumptions

Assume each student has only one course list

Assume there is only one admin who can see order list

Assume Admin will contact client manually by sending email

Assume each request must have client id, course id, teacher id and email

Assume every request will be added to client's list and admin's list

Assume the admin will contact the teacher through its own internal software app.

Assume that the meetings will be recorded by admin directly via accessing database

Implementation session:

Change log for implementation:

During our implementation of the website application, we have encountered some changes to The Entity-Relationship Design.

Some relations such as Select, Makem and Fulfill are not essentially needed in the real implementation section, since the backend can simply do a query to the database to do the lookup, therefore, these relations are not implemented in the system in order to not cause any confusion or any unnecessary difficulties.

Zoom table is not used in the real implementation section since the admin will contact the user through email, then logically speaking it will be better if the admin will also send the user an invitation link through email, since the zoom link contains lots of information: time zone, password, and alternative ways to join, also for users, therefore, it is not included in our design. No attributes are added.

Implementation of the web application:

It is implemented by using php, html, css and a little bit of javascript. We are using Apache 2.0 web server and phpmyadmin for the MySQL database. We have created a user landing page, a login page, registration page, welcome page for users to view the available courses and teachers, make requests, view appointments, and a page for admin to get access to users, user's requests, order list, courses and teachers, and view and book appointments.

Github Repository:

<https://github.com/zhifanl/CodeBuddies>

Implementation of DBMS:

MySQL is used as DBMS.

SQL Statements:

Function: Update Admin

Inputs: name, password

Outputs: Boolean

Pseudocode:

Connect to database

```
Query = UPDATE admin as p
      SET p.password='$password'
      WHERE p.name='$name';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: Delete Admin

Inputs: name

Outputs: Boolean

Pseudocode:

Connect to database

```
Query = DELETE FROM admin WHERE name = '$name';
```

Parse Query

Execute Query

Close Connection to Database
Return true if it worked, else false

Function: Post Admin
Inputs: name, password
Outputs: Boolean
Pseudocode:
Connect to database
Query = `INSERT INTO admin(name, password) VALUES (?, ?);`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getAppointmentByName
Inputs: username
Outputs: result_array
Pseudocode:
Connect to database
Query = `SELECT * FROM appointment WHERE username= $_SESSION["username"];`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getAppointment
Inputs: limit
Outputs: result_array
Pseudocode:
Connect to database
Query = `SELECT * FROM appointment LIMIT ?;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: deleteAppointment
Inputs: id
Outputs: Boolean
Pseudocode:
Connect to database
Query = `DELETE FROM appointment WHERE id = '$id';`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: postAppointment
Inputs: id
Outputs: Boolean
Pseudocode:

Connect to database

```
Query = INSERT INTO appointment(date, admin_id, email, username, teacher_name,  
course_name) VALUES (?, ?, ?, ?, ?, ?);
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: updateMakeRequest

Inputs: student id, request id

Outputs: Boolean

Pseudocode:

Connect to database

```
Query = UPDATE make_request as p  
      SET p.request_id = '$request_id'  
      WHERE p.student_id = '$student_id';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: deleteMakeRequest

Inputs: student id, request id

Outputs: Boolean

Pseudocode:

Connect to database

```
Query = DELETE FROM make_request WHERE request_id = '$request_id';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: postMakeRequest

Inputs: student id, request id

Outputs: Boolean

Pseudocode:

Connect to database

```
Query = INSERT INTO make_request(student_id, request_id ) VALUES (?, ?);
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getOrderById

Inputs: username

Outputs: result_array

Pseudocode:

Connect to database

Query = `SELECT * FROM order_list WHERE order_id=$id;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getOrderList
Inputs: limit
Outputs: result_array
Pseudocode:
Connect to database
Query = `SELECT * FROM order_list LIMIT ?;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: updateOrderList
Inputs: \$order_id, \$student_name, \$student_id, \$course_name, \$start_date, \$end_date, \$salary
Outputs: Boolean
Pseudocode:
Connect to database
Query = `UPDATE order_list as p
SET
p.student_name='$student_name', p.student_id='$student_id', p.course_name='$course_name'
, p.start_date='$start_date', p.end_date='$end_date', p.salary='$salary'
WHERE p.order_id=$order_id;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: deleteOrderList
Inputs: order id
Outputs: Boolean
Pseudocode:
Connect to database
Query = `DELETE FROM order_list
WHERE order_id= '$order_id';`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: postOrderList

Inputs: \$order_id, \$student_name, \$student_id, \$course_name, \$start_date, \$end_date, \$salary

Outputs: Boolean

Pseudocode:

Connect to database

Query = `INSERT INTO order_list(student_name, student_id, course_name, teacher_name, start_date, end_date, salary) VALUES (?, ?, ?, ?, ?, ?, ?);`

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getRequest

Inputs: limit

Outputs: result_array

Pseudocode:

Connect to database

Query = `SELECT * FROM request LIMIT ?;`

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: updateRequest

Inputs: \$request_id, \$email, \$client_name, \$teacher_name, \$course_name

Outputs: Boolean

Pseudocode:

Connect to database

Query = `UPDATE request as p
SET`

`p.request_id='$request_id', p.email='$email', p.client_name='$client_name',`

`p.teacher_name='$teacher_name', p.course_name='$course_name'`

`WHERE p.request_id=$request_id;`

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: deleteRequest

Inputs: \$request_id

Outputs: Boolean

Pseudocode:

Connect to database

Query = `DELETE FROM request
WHERE request_id= '$request_id';`

Parse Query

Execute Query
Close Connection to Database
Return true if it worked, else false

Function: postRequest

Inputs: \$email, \$client_name, \$teacher_name, \$course_name

Outputs: Boolean

Pseudocode:

Connect to database

Query = `INSERT INTO request(email,client_name, teacher_name,course_name) VALUES
(?,?,?,?)`;

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getFeeByName

Inputs: course_name

Outputs: fee

Pseudocode:

Connect to database

Query = `SELECT * FROM software_courses WHERE course_name=$course_name`;

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getCourseById

Inputs: id

Outputs: result_array

Pseudocode:

Connect to database

Query = `SELECT * FROM software_courses WHERE id=$id`;

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getSoftwareCourses

Inputs: limit

Outputs: result_array

Pseudocode:

Connect to database

Query = `SELECT * FROM software_courses LIMIT ?`;

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: updateSoftwareCourses

Inputs: \$course_name, \$description, \$tuition_fee

Outputs: Boolean

Pseudocode:

Connect to database

```
Query = UPDATE software_courses as p
      SET p.description='$description' ,p.tuition_fee='$tuition_fee'
      WHERE p.course_name='$course_name';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: deleteSoftwareCourses

Inputs: \$course_name

Outputs: Boolean

Pseudocode:

Connect to database

```
Query =DELETE FROM software_courses
      WHERE course_name ='$course_name';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: postSoftwareCourses

Inputs: \$course_name, \$description, \$tuition_fee

Outputs: Boolean

Pseudocode:

Connect to database

```
Query =INSERT INTO software_courses(course_name, description,tuition_fee) VALUES (?,
?,?);
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getStudentCourseList

Inputs: limit

Outputs: result_array

Pseudocode:

Connect to database

```
Query = SELECT * FROM student_course_list LIMIT ?;
```

Parse Query

Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getStudentCourseListById
Inputs: id
Outputs: result_array
Pseudocode:
Connect to database
Query = `SELECT * FROM student_course_list WHERE student_id=$_SESSION['id'];`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: checkDuplicateOrder
Inputs: \$course_name, \$description, \$tuition_fee, \$course_name, \$teacher_name
Outputs: Boolean
Pseudocode:
Connect to database
Query = `SELECT * FROM student_course_list WHERE student_id='$student_id' AND
tuition_fee='$tuition_fee' AND course_name='$course_name' AND
teacher_name='$teacher_name';`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: updateStudentCourseList
Inputs: \$student_id, \$tuition_fee, \$course_name, \$start_date, \$end_date, \$teacher_name
Outputs: Boolean
Pseudocode:
Connect to database
Query = `UPDATE student_course_list as p
SET

p.tuition_fee='$tuition_fee', p.course_name='$course_name', p.start_date='$start_date',
p.end_date='$end_date', p.teacher_name='$teacher_name'

WHERE p.student_id=$student_id AND p.course_name= '$course_name';`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: deleteStudentCourseList
Inputs: \$course_name, \$student_id
Outputs: Boolean
Pseudocode:
Connect to database

```
Query =DELETE FROM student_course_list
        WHERE student_id= '$student_id' AND course_name= '$course_name';
```

Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: postStudentCourseList

Inputs: \$student_id, \$tuition_fee, \$course_name, \$start_date, \$end_date, \$teacher_name

Outputs: Boolean

Pseudocode:

Connect to database

```
Query =INSERT INTO student_course_list(student_id,
tuition_fee,course_name,start_date,end_date,teacher_name) VALUES (?, ?, ?, ?, ?, ?);
```

Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getTeacher

Inputs: limit

Outputs: result_array

Pseudocode:

Connect to database

```
Query =SELECT * FROM teacher LIMIT ?;
```

Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getTeacherNameById

Inputs: \$id

Outputs: result_array

Pseudocode:

Connect to database

```
Query =SELECT * FROM teacher WHERE teacher_id=$id;
```

Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: updateTeacher

Inputs: \$teacher_name, \$admin_id

Outputs: Boolean

Pseudocode:

Connect to database

```
Query =UPDATE teacher as p
      SET p.teacher_name='$teacher_name', p.admin_id='$admin_id'
      WHERE p.teacher_name='$teacher_name';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: deleteTeacher

Inputs: \$teacher_nameOutputs: Boolean

Pseudocode:

Connect to database

```
Query =DELETE FROM teacher
      WHERE teacher_name= '$teacher_name';
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: postTeacher

Inputs: \$teacher_name

Outputs: Boolean

Pseudocode:

Connect to database

```
Query =INSERT INTO teacher(teacher_name) VALUE (?);
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getIdByUsername

Inputs: \$username

Outputs: result_array

Pseudocode:

Connect to database

```
Query =SELECT * FROM users WHERE username=$username;
```

Parse Query

Execute Query

Close Connection to Database

Return true if it worked, else false

Function: getEmailByUsername

Inputs: \$username

Outputs: result_array

Pseudocode:

Connect to database

```
Query =SELECT * FROM users WHERE username=$username;
```

Parse Query

Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getUsernameById
Inputs: \$id
Outputs: result_array
Pseudocode:
Connect to database
Query =`SELECT * FROM users WHERE id=$id;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getRealNameByUsername
Inputs: \$username
Outputs: result_array
Pseudocode:
Connect to database
Query =`SELECT * FROM users WHERE username`
`= $username;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: getUsers
Inputs: \$\$limit
Outputs: result_array
Pseudocode:
Connect to database
Query =`SELECT * FROM users LIMIT ?;`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

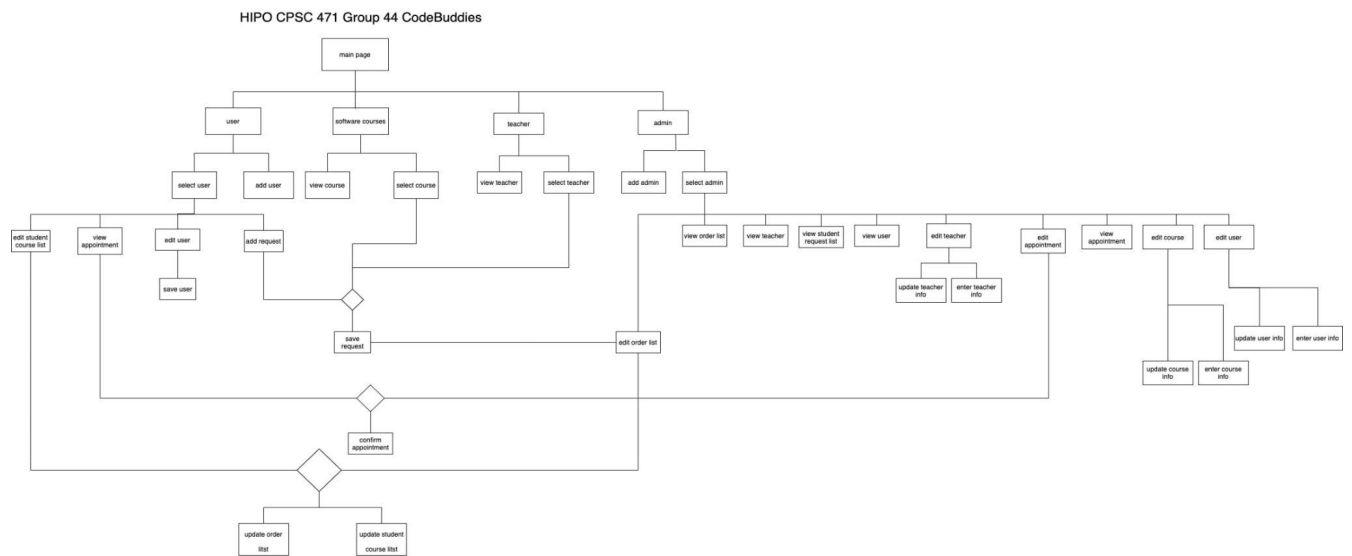
Function: updateUser
Inputs: \$username, \$name, \$email, \$university, \$major, \$location, \$description
Outputs: Boolean
Pseudocode:
Connect to database
Query =`UPDATE users as p`
`SET p.name= '$name', p.email= '$email', p.university= '$university' ,`
`p.major= '$major', p.location= '$location' ,p.description= '$description'`
`WHERE p.username= '$username';`

Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

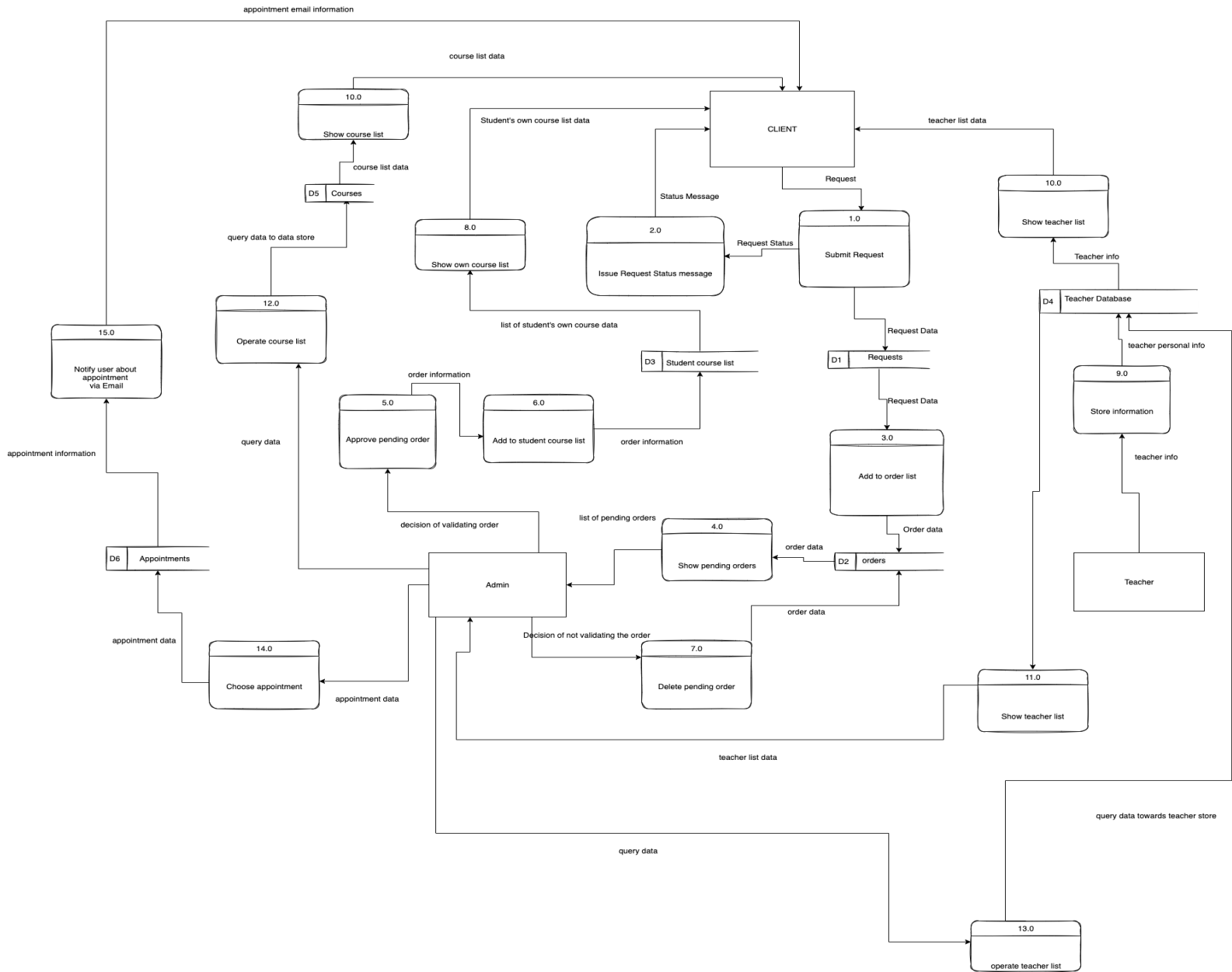
Function: deleteUser
Inputs: \$username
Outputs: Boolean
Pseudocode:
Connect to database
Query =`DELETE FROM users WHERE username = '$username';`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

Function: postUser
Inputs: \$username, \$password
Outputs: Boolean
Pseudocode:
Connect to database
Query =`INSERT INTO users(username, password) VALUES (?, ?);`
Parse Query
Execute Query
Close Connection to Database
Return true if it worked, else false

HIPO diagram



DFD Diagram



API Documentation:

Published CodeBuddies API documentation

<https://documenter.getpostman.com/view/16661608/UVCB9PXG>

User:

Endpoint 1:

Description: Update the user.

URL: <http://localhost/demo/index.php/user/list>

Method: PUT

Input:

```
{  
  "Username": String  
  "name": String  
  "email": String,  
  "university": String  
  "major": String  
  "location": String  
  "description": String  
}
```

Output: return Record updated successfully if worked

Endpoint 2:

Description: Get the user.

<http://localhost/demo/index.php/user/list?limit=integer>

Method: GET

Input: Nothing

Output [{
 "ID": int,
 "username": string,
 "password": string,
 "name": string,
 "email": string,
 "university": string,
 "major": string,
 "location": string,
 "description": string

}]

Endpoint 3:

Description: Delete the user.

<http://localhost/demo/index.php/user/list?username={{string}}>

Method: DELETE

Input: username

Output: return Record updated successfully if worked

Endpoint 4:

Description: Set the user.

http://localhost/demo/index.php/user/list
Method: POST
Input :
{
 "username": string,
 "password": "string"
}
Output: return Record updated successfully if worked

SoftwareCourses

Endpoint 1:
Description: Get the software course.
http://localhost/demo/index.php/software_courses/list
Method: GET
Input: Nothing
Output [{
 "id": int,
 "course_name": string,
 "description": string,
 "tuition_fee": int
}]
Endpoint 2:
Description: Set the software course.
http://localhost/demo/index.php/software_courses/list
Method: POST
Input:
{
 "course_name": String,
 "description": String,
 "Tuition_fee": int
}
Output:
return Record updated successfully if worked

Endpoint 3:
Description: Update the software course.
http://localhost/demo/index.php/software_courses/list
Method: PUT
Input:
{
 "course_name": String,
 "description": String,
 "Tuition_fee": int
}
Output: return Record updated successfully if worked

Endpoint 4:
Description: Delete the software course.
http://localhost/demo/index.php/software_courses/list?course_name={{string}}
Method: DELETE
Input:
courseName: string
Output: return Record updated successfully if worked

Student course list

Endpoint 1:

Description: Get the student course.

http://localhost/demo/index.php/student_course_list/list

Method: GET

Input: Nothing

Output :

```
{
  "student_id": int,
  "tuition_fee": int,
  "course_name": string,
  "start_date": string,
  "end_date": string,
  "teacher_name": string
}
```

}}

Endpoint 2:

Description: Set the student course.

http://localhost/demo/index.php/student_course_list/list

Method: POST

Input:

```
{
  "student_id": int,
  "tuition_fee": int,
  "course_name": String,
  "start_date": String,
  "end_date": String,
  "teacher_name": String
}
```

Endpoint 3:

Description: Update the student course.

http://localhost/demo/index.php/student_course_list/list

Method: PUT

Input:

```
{
  "student_id": int,
  "tuition_fee": int,
  "course_name": String,
  "start_date": String,
  "end_date": String,
  "teacher_name": String
}
```

Endpoint 4:

Description: Delete the student course.

http://localhost/demo/index.php/student_course_list/list?course_name={{string}}&student_id={{int}}

Method: DELETE

Input:

```
"course_name": String,
"student_id": int
```

Output: return Record updated successfully if worked

Admin

Endpoint 1:

Description: Get the Admin.

http://localhost/demo/index.php/student_course_list/list

Method: GET

Input: Nothing

```
Output [{
  "id": int,
  "name": string,
```

"password": string

}}

Endpoint 2:

Description: Set the admin.

<http://localhost/demo/index.php/admin/list>

Method: POST

Input:

```
{
  "name": String,
  "password": String
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the admin.

<http://localhost/demo/index.php/admin/list>

Method: PUT

Input:

```
{
  "name": String,
  "password": String
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the admin.

<http://localhost/demo/index.php/admin/list?name={{string}}>

Method: DELETE

Input

"name": String

Output: return Record updated successfully if worked

Teacher:

Endpoint 1:

Description: Get the Teacher.

<http://localhost/demo/index.php/teacher/list>

Method: GET

Input: Nothing

Output [{

```
  "teacher_id": int,
  "teacher_name": string,
  "admin_id": string
```

}]

Endpoint 2:

Description: Set the teacher.

<http://localhost/demo/index.php/teacher/list>

Method: POST

Input:

```
{
  "teacher_name": String
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the teacher.

<http://localhost/demo/index.php/admin/list>

Method: PUT

Input:

```
{  
  "teacher_name": String  
  "admin_id": int  
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the teacher.

http://localhost/demo/index.php/teacher/list?teacher_name={{string}}

Method: DELETE

Input: teacher_name

Output: return Record updated successfully if worked

SelectAppointment:

Endpoint 1:

Description: Get the Appointment.

http://localhost/demo/index.php/select_appointment/list

Method: GET

Input: Nothing

```
Output [{  
  "student_id": int,  
  "apointment_id": int  
}]
```

Endpoint 2:

Description: Set the appointment.

http://localhost/demo/index.php/select_appointment/list

Method: POST

Input:

```
{  
  "student_id": int,  
  "apointment_id": int  
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the appointment.

http://localhost/demo/index.php/select_appointment/list

Method: PUT

Input:

```
{  
  "student_id": int,  
  "apointment_id": int  
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the appointment.

http://localhost/demo/index.php/select_appointment/list?appointment_id={{id}}

Method: DELETE

Input:

"appointment_id": int

Output: return Record updated successfully if worked

Request:

Endpoint 1:

Description: Get the request.

<http://localhost/demo/index.php/request/list>

Method: GET

Input: Nothing

Output [{
 "email": string,
 "client_name": string,
 "teacher_name": string,
 "course_name": string
}]

Endpoint 2:

Description: Set the request.

<http://localhost/demo/index.php/request/list>

Method: POST

Input:

```
{  
  "email": string,  
  "client_name": string,  
  "teacher_name": string,  
  "course_name": string  
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the request.

<http://localhost/demo/index.php/request/list>

Method: PUT

Input:

```
{  
  "email": string,  
  "client_name": string,  
  "teacher_name": string,  
  "course_name": string  
  "request_id": int  
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the appointment.

http://localhost/demo/index.php/request/list?request_id={{id}}

Method: DELETE

Input:

"request_id": int

Output: return Record updated successfully if worked

Order List:

Endpoint 1:

Description: Get the order list.

http://localhost/demo/index.php/order_list/list

Method: GET

Input: Nothing

Output [{
 "student_name": string,
 "student_id": int,
 "course_name": string,
 "teacher_name": string,
 "start_date": string,
 "end_date": string,
 "salary": int
}]

Endpoint 2:

Description: Set the order list.

http://localhost/demo/index.php/order_list/list

Method: POST

Input:

```
{
  "student_name": string,
  "student_id": int,
  "course_name": string,
  "teacher_name": string,
  "start_date": string,
  "end_date": string,
  "salary": int
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the order list.

http://localhost/demo/index.php/order_list/list

Method: PUT

Input:

```
{
  "order_id": int,
  "student_name": string,
  "student_id": int,
  "course_name": string,
  "teacher_name": string,
  "start_date": string,
  "end_date": string,
  "salary": int
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the order list.

http://localhost/demo/index.php/order_list/list?order_id={{id}}

Method: DELETE

Input:

"order_id": int

Output: return Record updated successfully if worked

Appointment:

Endpoint 1:

Description: Get the appointment.

http://localhost/demo/index.php/appointment/list

Method: GET

Input: Nothing

Output [{

```
  "date": string,
  "email": string,
  "user_name": string,
  "teacher_name": string,
  "course_name": string
}]
```

Endpoint 2:

Description: Set the appointment.

http://localhost/demo/index.php/appointment/list

Method: POST

Input:

```
{
  "date": string,
  "email": string,
```

```
"user_name": string,  
"teacher_name": string,  
"course_name": string  
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the appointment.

<http://localhost/demo/index.php/appointment/list>

Method: PUT

Input:

```
{  
  "id": int,  
  "date": string,  
  "admin_id": string  
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the appointment.

<http://localhost/demo/index.php/appointment/list?id={{id}}>

Method: DELETE

Input:

"id": int

Output: return Record updated successfully if worked

Fulfill Request Teacher:

Endpoint 1:

Description: Get the teacher request.

http://localhost/demo/index.php/fulfill_request_teacher/list?limit=10

Method: GET

Input: Nothing

Output [{

```
  "request_id": int,  
  "teacher_id": string
```

}]

Endpoint 2:

Description: Set the teacher request.

http://localhost/demo/index.php/fulfill_request_teacher/list

Method: POST

Input:

```
{  
  "request_id": int,  
  "teacher_id": string  
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the teacher request.

http://localhost/demo/index.php/fulfill_request_teacher/list

Method: PUT

Input:

```
{  
  "request_id": int,  
  "teacher_id": string  
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the teacher request.

http://localhost/demo/index.php/fulfill_request_teacher/list?id={{id}}

Method: DELETE

Input:

"id": int

Output: return Record updated successfully if worked

In person:

Endpoint 1:

Description: Get the in person request.

http://localhost/demo/index.php/in_person/list

Method: GET

Input: Nothing

Output [{

"id": string,

"location": string

}]

Endpoint 2:

Description: Set the in person request.

http://localhost/demo/index.php/in_person/list

Method: POST

Input:

{

"Id": string,

"location": string

}

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the in person request.

http://localhost/demo/index.php/in_person/list

Method: PUT

Input:

{

"Id": string,

"location": string

}

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the in person request.

http://localhost/demo/index.php/fulfill_request_teacher/list?id={{id}}

Method: DELETE

Input:

"id": int

Output: return Record updated successfully if worked

Make Request:

Endpoint 1:

Description: Get the make request.

http://localhost/demo/index.php/make_request/list?limit=10

Method: GET

Input: Nothing

Output [{

"student_id": string,

"request_id": string

}]

Endpoint 2:

Description: Set the make request.

http://localhost/demo/index.php/make_request/list

Method: POST

Input:

```
{  
  "student_id": string,  
  "request_id": string  
}
```

Output: return Record updated successfully if worked

Endpoint 3:

Description: Update the make request.

http://localhost/demo/index.php/make_request/list

Method: PUT

Input:

```
{  
  "student_id": string,  
  "request_id": string  
}
```

Output: return Record updated successfully if worked

Endpoint 4:

Description: Delete the make request.

http://localhost/demo/index.php/make_request/list?id={{id}}

Method: DELETE

Input:

"id": int

Output: return Record updated successfully if worked

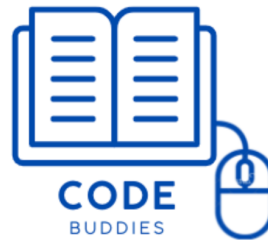
Website Design

Home page:

- Show user login option
- Register account
- View courses
- See the "about us" page.

Code Buddies

fast and innovative way to learn about programming

[Login](#)

Sign up here

Join our community to boost your programming skills

☐ Remember me[Sign up](#)

Already have an account? [Login here](#)

By clicking Sign up, you agree to the terms of use.

Login Page as admin:

Login

Please fill in your credentials to login.

Username

Password

Don't have an account? [Sign up now.](#)

Code Buddies; 2021 Company, Inc

Login Page as normal client:

Login

Please fill in your credentials to login.

Username

Password

Don't have an account? [Sign up now.](#)

Code Buddies; 2021 Company, Inc

About us page:

Where we at



Our team is made up with 3 software engineering students at the University of Calgary with professional experience in the job industry and school. All of our team members have excellent grades at school with GPA above 3.90 and The solution we have devised is to provide clients with one on one tutoring sessions. This is to aid them when they struggle with a programming problem or if they need further clarification on high yield programming topics. In today's remote technological age, most businesses are going online and that is exactly how we intend to conduct business.

CodeBuddies will enable our clients to make a variety of choices; to make orders, select appointments, and talk to their mentors, ask any problems that they do not understand.

Standard User Page:

Courses

#	Course Name	Description	Tuition Fee
1	ENCM511	Embedded System interfacing	600
3	ENSF480	Principles of Software Design	250
4	CPSC457	Principles of OS	500
5	CPSC319	Data structures and Algorithms	5

List of Teachers

#	Teacher Name
6	Norman





List of Courses you have

#	Course Name	Teacher Name	Description	Tuition Fee	Start Date	End Date
---	-------------	--------------	-------------	-------------	------------	----------

List of Courses you have

#	Course Name	Teacher Name	Description	Tuition Fee	Start Date	End Date	
13	CPSC319	Jorg		5	NULL	NULL	  

Appointments

#	email	user name	teacher name	course name	date	
8	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 Jan 8	
9	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 Jan 8	
10	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 Jan 8	
11	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 jan 8	

Order your course here

Email

User Name

Teacher Name

Course Name

Submit

Enter all the fields above.

Update your information

User Name

Real Name

Email

University

Major

Location



Description

Submit










Enter all the fields above.

Admin Page:

Orders


#	Student name	Student id	Course name	Teacher name	Start Date	End Date	Tuition Fee	
21	Zhifan Li	13	CPSC319	Jorg	NULL	NULL	5	 

Requests





#	Student email	Student name	Teacher name	Course name	
1	zhifanli2000@gmail.com	CPSC319	zhifan	Zhifan-admin	
3	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	
20	zhifanli2000@gmail.com	zhifan	jorg	ENSF409	
23	zhifanli2000@gmail.com	zhifan	Tianfan Zhou	ENSF409	
24	zhifanli2000@gmail.com	zhifan	Tianfan Zhou	ENSF409	
25	zhifanli2000@gmail.com	zhifan	Tianfan Zhou	ENSF409	
26	zhifanli2000@gmail.com	zhifan	Tianfan Zhou	ENSF409	
27	zhifanli2000@gmail.com	zhifan	IDK	CPSC319	
28	zhifanli2000@gmail.com	zhifan	RINKER	ENGG201	

--	--	--	--	--	--



List of Teachers

#	Teacher Name	
6	Norman	

Courses





#	Course Name	Description	Tuition Fee	
1	ENCM511	Embedded System interfacing	600	
3	ENSF480	Principles of Software Design	250	
4	CPSC457	Principles of OS	500	
5	CPSC319	Data structures and Algorithms	5	

Users

#	username	name	email	university	major	location	description	
11	admin							
13	zhifan	TOMMM	Zhifanli211000@gmail.com	Ucal	SENG	Calgary	123	

Appointments

Appointments

#	email	user name	teacher name	course name	date	
8	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 Jan 8	
9	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 Jan 8	
10	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 Jan 8	
11	zhifanli2000@gmail.com	zhifan	Jorg	CPSC319	2022 jan 8	

Add Appointment Here

Email

User Name

Teacher Name

Course Name

Date and Time

User Manual:

How to run it

1. Clone the repository: <https://github.com/zhifanli/CodeBuddies>
2. Download XAMPP
3. run the Apache web server and MySQL database on XAMPP
4. copy the ``demo`` folder from this repository to the inside of ``htdocs`` folder at XAMPP's general folder

5. Go to browser to enter ``http://localhost:{{your port#}}/phpmyadmin``, then create a new database called `471`, then import the sql file from the DB_Backup folder, that will be our database for the website.
6. Now you can access the website via ``http://localhost/demo``
7. If it does not work, make sure the port number is correct (you will know which port the Apache server is using by finding it on XAMPP, mine is 80, someone's port is 8080), then try ``http://localhost:{{your port#}}/demo``
8. To use sending email features, you need to go to XAMPP application folder: xamppfiles/etc, then go to httpd.conf file and add ``SetEnv SENDGRID_API_KEY {{YOUR_API_KEY}}`` at the very end of the file and save it. Then restart XAMPP.
9. To login as Admin, simply create a new user on your localhost with username: admin

How to test the website's functionality and use it:

- First, Create an admin account: Use admin as username, with any password you want, you can login as admin.
- Create a student account
- log in as a student: you can see a list of courses, list of teachers, a list of the courses that you have ordered, a list of appointments that you have with your mentor, an area for updating your personal information, and at the end there is a form for submitting a request to take a course.
- First you can view the courses and teachers, then go to the end of the page to submit a request if you want to take a course.
- By entering your username, email, teacher's name and course name, then clicking submit, this request will be shown on the admin's end.
- Now, for the purpose of testing, you can log out and log in as an admin

- As admin, you can first see a form for sending emails, and a list of orders and requests, list of teachers and list of courses, you can view the users and have full access to them, at the end there is a form for creating appointments with students.
- Now, you will see there is a new request and new order listed at the same time. The list of requests is for recording the requests that the student has sent, and the admin can delete it, the purpose of request is just for the admin to look up. The list of orders is for the admin to approve or reject: the last column of the table has two icons: first is approving the request, second one is rejecting it. Once the admin approves the order, this order will be added to the student's list of ordered courses. Once the admin ignores the order, it will be removed from the order list.
- Now you can approve that order, and log out
- Now login as a student, you will see that there's a new row added at the List of Courses You Have section. That is the course that you just ordered.
- Now since the course is ordered, the admin can login and at the Send Email section: Send an email to student for asking appointments. Now they can communicate through email, which is a faster and safer way since the message is encrypted on both ends.
- Once they confirm the date and other stuff. Admin can use the Add Appointment feature located at the end of the admin's website.
- Admin will enter the student's email, name, teacher's name, course and date, then click submit. This appointment will be added to both Admin's Web UI, and normal student's Web UI.
- Now once the student is logged in, the appointment that the admin added will be shown to the student.
- This website has error checking functionalities, so everytime you put some wrong input, it will prompt you to enter again.

