CSC490: Software Engineering

**Documentation Document of VLA**

# Phase 3: Implementation as an Android App

**Mohamad Kawas: 202001858**

**Mohamad Yateem: 202002828**

**Taleb Farhat: 202005888**

# Introduction:

In this phase we implemented our use cases we discussed in the previous phase. the three use cases are:

1. Add task by instructor (taleb farhat)
2. add board: add notes by student (Mohamad kawas)
3. add study sessions by student (Mohamad yateem)

The idea here is we showed how we cam implement our ideas on Android STudio using kotlin, WIth storing, saving our data on a mysql database using php and creating rest apis for the liking between our database and user experience.

Since all our methods used were api calls(get and post requests), we couldn’t use unit testing, but something called integration testing usinp postman.

# System Implementation and Testing:

In order to implement our program, we need to establish the grounds of it. we are going to start with the database. the database stores our data, which are the users(students and teachers), courses, tasks, study sessions, boards, and notes. each table represents a class we shared in our class diagram in the previous phase.

we should mention first that mohamad kawas worked on the code using kotlin because he knows the language at best, helped mohamad yateem on working on the apis and assisted taleb on working on the database study and relations.

WE manually inserted the users and the courses into the database and we assumed that each instructor teaches only one section of a course.

* LET’S start with our first use case, add task. here we have several classes: coures, teachers, and tasks. let’s say an instructor named fareed wants to add a task for a course called csc490. in fareed’s class, he has 2 students, AHMAD, and george. fareed logs in to his ACCOUNT AND is redirected to a teacher’s course view (i must let you know our program has 2 different views, student and teacher). in this view, all the courses that fareed teaches are shown in a grid layout (let’s say he also teaches csc310 and csc599). fareed clicks on the csc490 course and he is redirected to another view that shows the course content. fareed now can access the tasks he assigned to his students. after accessing the tasks, he has the option to add a new task. fareed clicks on the add button and now can add the task he wants in the task text box. after that, fareed submits and is redirected back to the tasks view WHERE he can see the new task he added. thus, fareed added a new task successfully and stored in the database.

explanation:

when fareed logged in, the system called the log in api which returns the name of the user, his id. after logging in and being redirected to the home page of the teacher, an automatic api is called, named getCourses which takes as parameters the teacher’s id and returns the courses he gives in lau. after selecting a course, the teacher is now redirected to another page.

in this course view page, the course selected is saved in a local memory. the only option is to click on the tasks button to view the tasks the teacher already assigned or want to assign a new task. for this phase we only going to add tasks only. when he clicks on tasks button, he is redirected to a new page called teacher\_tasks\_view where an api is called automatically named getTasks. gettasks takes parameters the teacher id and the course he already selected and brings back the task ids from the courses table where the ids are saved in a string separated by “;”.

after that, the if there exist tasks assigned by fareed in the course he taught, then they are listed. there is an option of adding a new task. when loading the tasks, the last task id and the string of ids are save in local memory.

adding a new task makes the teacher to add description for the task. when clicking on the submit button, the system checks if the description is null or not, if yes it gives an error and to retry, else it calls another api. this api is called addtask which takes parameters teacher id, course selected, and the new task id which is the last task id we stored in the local memory when loading the task incremented by 1 (new task = last task + 1). if succeeded and saved to the tasks table in the database, another api is called, named updatetasks, which updated the task ids for each student taking the course in the courses table. this api takes parameter: the initial task ids, the new task id, the course selected and the teacher id. the api concatenates the task ids with the new task and saved in each record of each student taking the course selected by the teacher in the vla.

if success, the teacher is redirected back to the tasks\_view where he can see his new task added. if we want to make sure if it really worked, let’s ask ahmad and george to access their tasks in their course section in the vla (they tried and it works). 😊

* Our second use case is add board to add notes. let’s take the scenario of george wassouf, a student we talked about earlier. george has to study for the csc490 exam his instructor told him about earlier. he wants to take notes while using his material. so, george logs in to his account and redirected to student view. once he is logged in, he can see his courses he is taking this semester. he clicks on csc490 and is redirected to the course view. he sees his the course instructor name and can access his tasks, boards and materials ( materials are not accessible at the moment for their no need in this phase). george clicks on the boards button and is redirected to a new view called boardS\_view. when accessed, george doesn’t see any thing just an add new board button. this is because george did use this vla before. when clicked on add new board, he is redirected to a new view called add\_Board\_view. he now can type the title of the board he wants to create. after typing the title and clicking submit, he is redirected back to the boards view and now can see his new created board. george clicks on the board he created and is redirected to a new view called selected\_board\_view. he now can see his notes if the board was created before and added notes previously. in our case, george just created his board. george named the board as “chapter 1”. he can now add a new note to his newly create board “chapter 1”. when clicking on the create new button, he is redirected to an add\_note\_view where he can type the note he created. after clicking submit he can see his note added to his selected\_board\_view. and that’s it for adding a new note

Explanation:

when logging in, the log in api returns the name, and id of the student where they are saved in local memory. after redirecting to the student home page, the api getcourses where it takes parameter student\_id is called and returns the student courses from the students table where they are stored as string separated by “;” for making the database smaller. the courses are then shown after the splitting of the courses from the string returned in a recyclerview ( android view type). after clicking on the CSC490 course, the page redirects the student to the course view where the course he selected is saved in local memory. the student clicks on boards which redirects him to another page named boards view. the boards are then shown after being returned by another api called taking parameters student id and course selected. if boards exists, it will return a string concatenated with boards ids separated by “;” which we will store in the local memory as old\_board\_ids, else returns a message that SAYS, “student didn’t create any boards yet”. the board ids for each student are stored in the courses table in the database. the ids are then viewed in a recyclerview in the boards view, where an api named getBoard is called for each api to fetch it’s title which is stored in the boards table in the database. the api takes parameters : student id, course selected and board id selected. the student can access his previous boards he has or create a new one, if not he needs to create a new board. we have several situations here;we need the last board id in order to add a new board whether preious boards for this course exist or no. thus we locate in memory an int that has a base case = 999. if we create a new board with no previous boards ever created, it will take an id of 1000, else it will increment the last id of the last board created. we now have the new board id we want to create( it is either 1000 or 1002 if we have other 2 boards created before). we click on create new board and are redirected to a Add\_new\_board\_view. student types title of board and clicks submit. if the title textbox is EMPTY, it will trigger an error forcing the student to add title, else if the same name of a board is already created, an error will be triggered saying the board already exists, else the system will call an api called addboard which takes parameters: student id, new board id( we explained earlier) and selected course that we saved earlier. it will return us a message whether the board is saved in the boards table in our database. now we need to update the board ids in the courses table since the board ids that link the student to the course he is taking and boards he created. so, another api is called, named updateboards, which takes the parameters: student\_id, old\_board\_ids, course selected, and the new\_board\_id which we all stored in our local memory to use. it will return a message whether it was successful or not. if successful, it will redirect the user back to the boards\_view. the new board is visible. the new board is now clicked and it redirects us to a selected\_board\_view. in parallel, another api is called, named getnotes which takes the parameters, student id, course and board\_selected\_id which returns the note\_ids of the student has already created before if they exist which will be saved in local memory under old\_note\_ids, else it will return a message saying student didn’t create notes yet. the notes\_ids are now fetched if they exist previously and an api is called, named getnote, which retrieves the actual note not the id, where it is saved in the notes table in the database. it takes parameters: student id, course selected, board\_id\_selected and note\_id. the api is called for each note fetched (the note\_ids are stored in a concatenated string separated by “;”. if not no note\_ids are returned thus empty string which the api takes care of and store it as empty in an empty string in local memory. similary what we did in trying to add a new board for the new\_board\_id. we set as base case new\_board\_id as 999 if there are no notes saved before, else saved as the last note id incremenTED by 1 to save as the new\_note\_id. when clicked on add new note, it will redirect the user to a new view called new\_note\_id. when adding new note, you enter the description you want. if the description is empty when clicked on submit, it will trigger an error showing the description is important. if not, it will return success and call another Api named updatenotes, same idea as the update boards because each student’s note ids are saved in the courses table in our database. and that’s simply it. (yeah that was really brain power consuming to do)

* on our last use case, the other student, ahmad shawqi, wants to study for the csc490 exam assigned by dr. fareed al atrache. So, ahmad wan’t to assign a study session for him to track down his study hours. ahmad logs in and directly has access to assign a study session. once clicked on study sessions button, it redirects george to another view called Study\_sessions\_view. it shows all the other timings george already has assigned previously. if no exist, then it will show nothing. so, george wants to add a study session for tomorrow at 1 pm. george clicks on the add study session button and is redirected to another temporary page called add\_study\_session view. he puts that he wants to study at 2 pm for 1 hour tomorrow(sets actual day let’s say 11 march). however, george has another timing for csc310 tomorrow at 2 pm for 1 hour. he faces a problem. the vla shows there is a time conflict when clicking submit. it tells george to change the timing because there is a timing conflict. so, george changes it to 3 pm and it redirects him back to the study sessions view where it shows him his new study session assigned for tomorrow at 3 pm to study csc490 for 1 hour. that’s it.

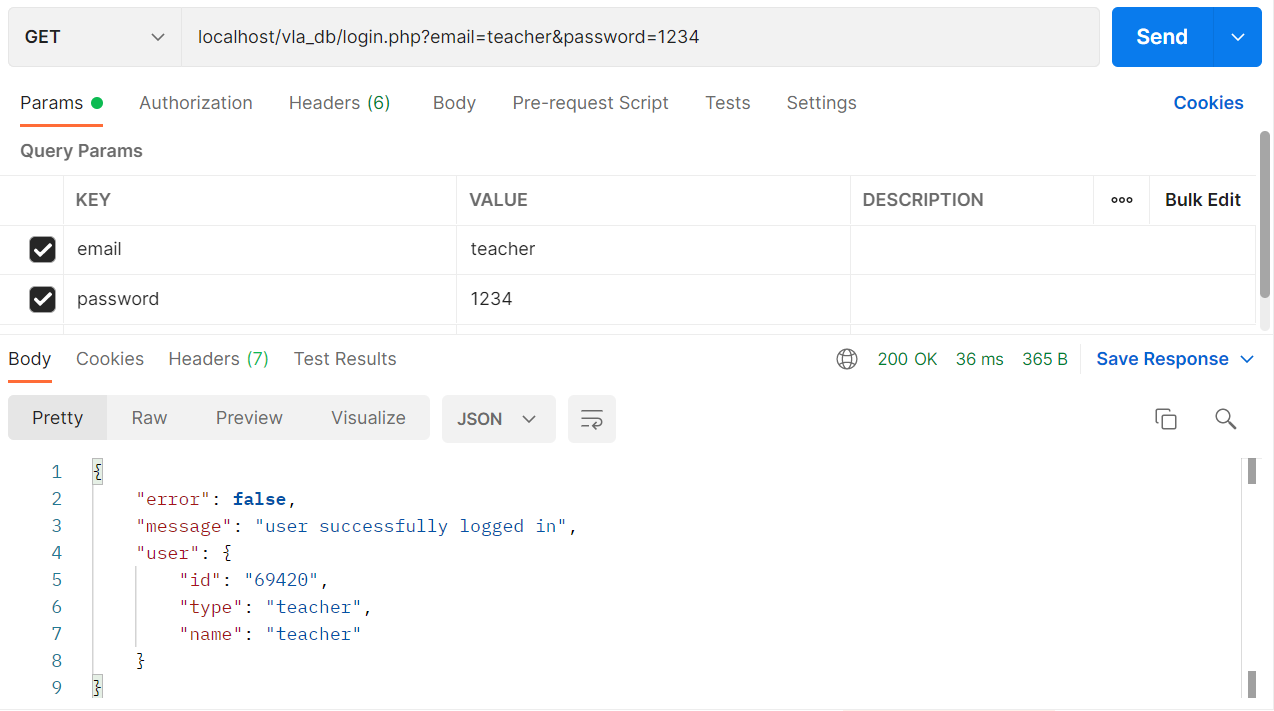
explanation:

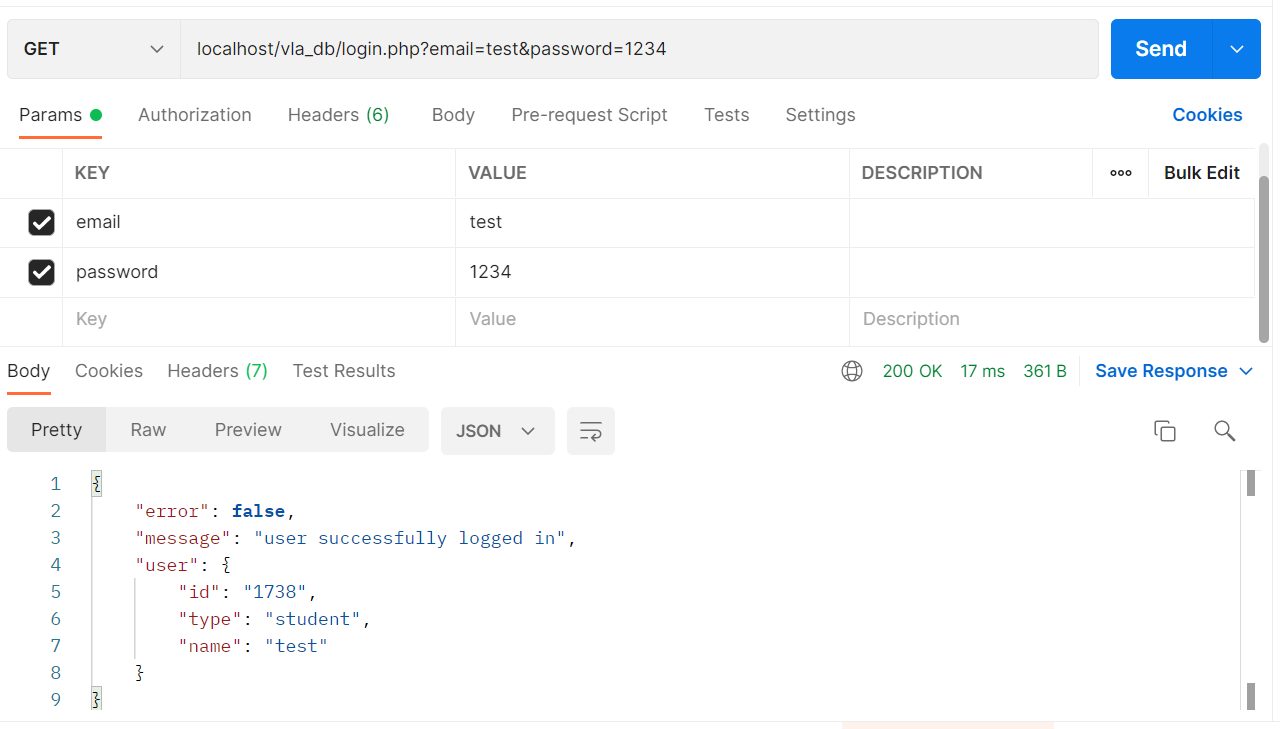
when the student logs in, like before the log in api is called returning the student id. the student is redirected to the home page where he can see his courses and study sessions. once clicked on study sessions, he is redirected to the study sessions view where in parallel an api called getsessions which takes parameter student id. it returns the study\_session\_ids, like before as a string separated with “;”. as before, each study\_session\_id is used as a parameter with student id to call the api getsession. similar to all the api calls but this one targets the students table were the session\_ids are saved to link them to the sessions saved in the Sessions table. for adding a new task, once the data are entered from the user( day, time, duration and course), once clicked on the submit button, an api is called, named addsession, where it checks first if the time is available in that day. it aslo checks if the user has already set a time for the same course, since a course cannot be set a study session more than once per day. after checking, it is inserted inside the study sessions table. now the system needs to update the student’s record in the students table, where the study session ids are stored as linking. so, an api is called named updatesessions where it takes parameters, student id, old\_session\_ids, and new\_session\_id that were saved in local history in the same fashion as the other apis. (you can say all the apis have the same idea being repeated but with different values and parameters)

# Integration Testing (Instead of Unit)

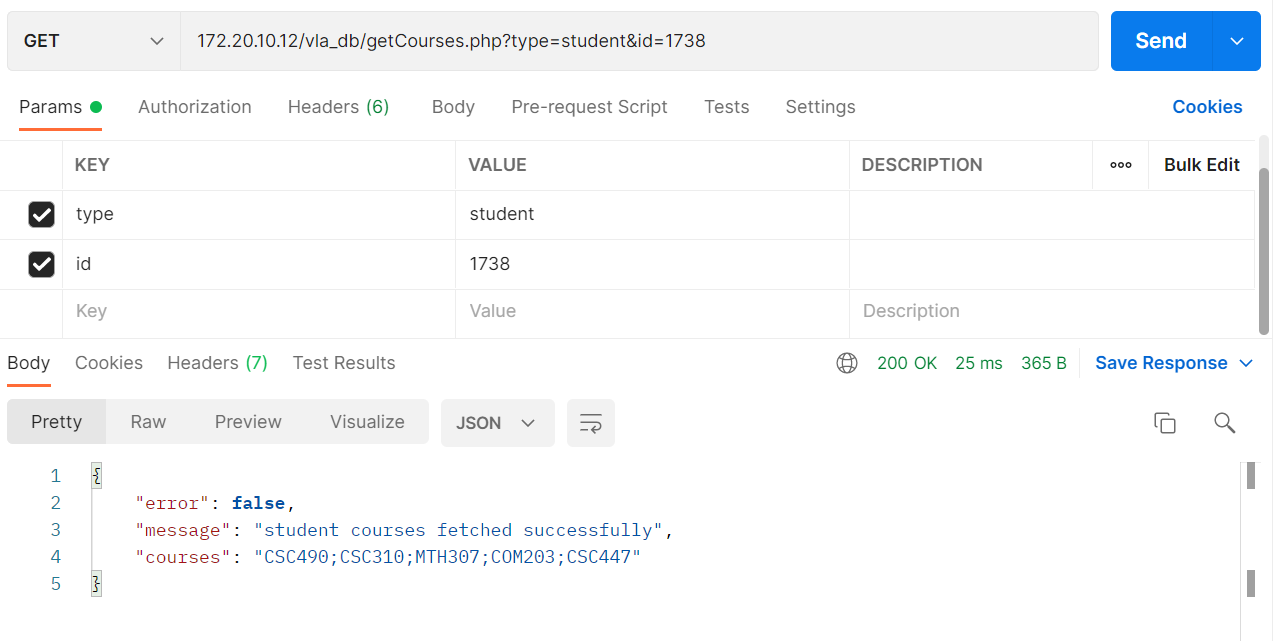
because we didn’t use a static demo, we had to use apis. so, we had to use another software instead of junit( for android development/java unit testing) which is called postman. postman is a service that lets you test your api call methods (in our case, get and post requests). so, let’s start with our api calls: (will provide screenshots of them without description because they are so obvious)

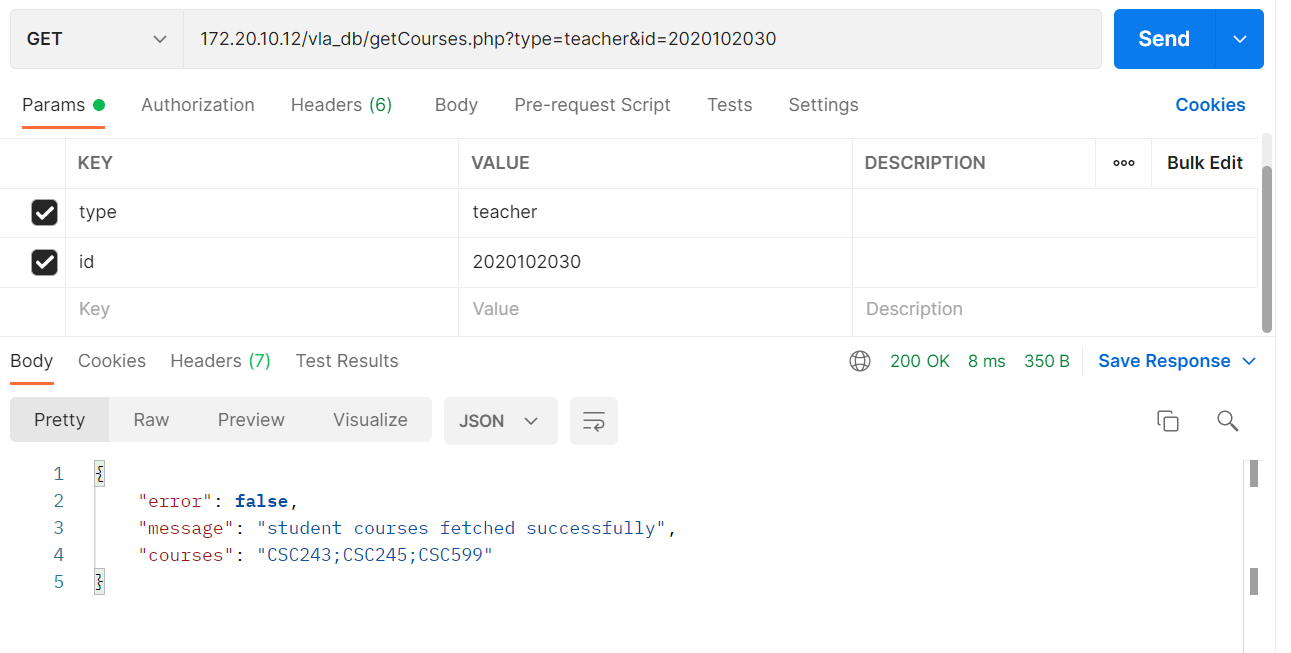
1. Login api:



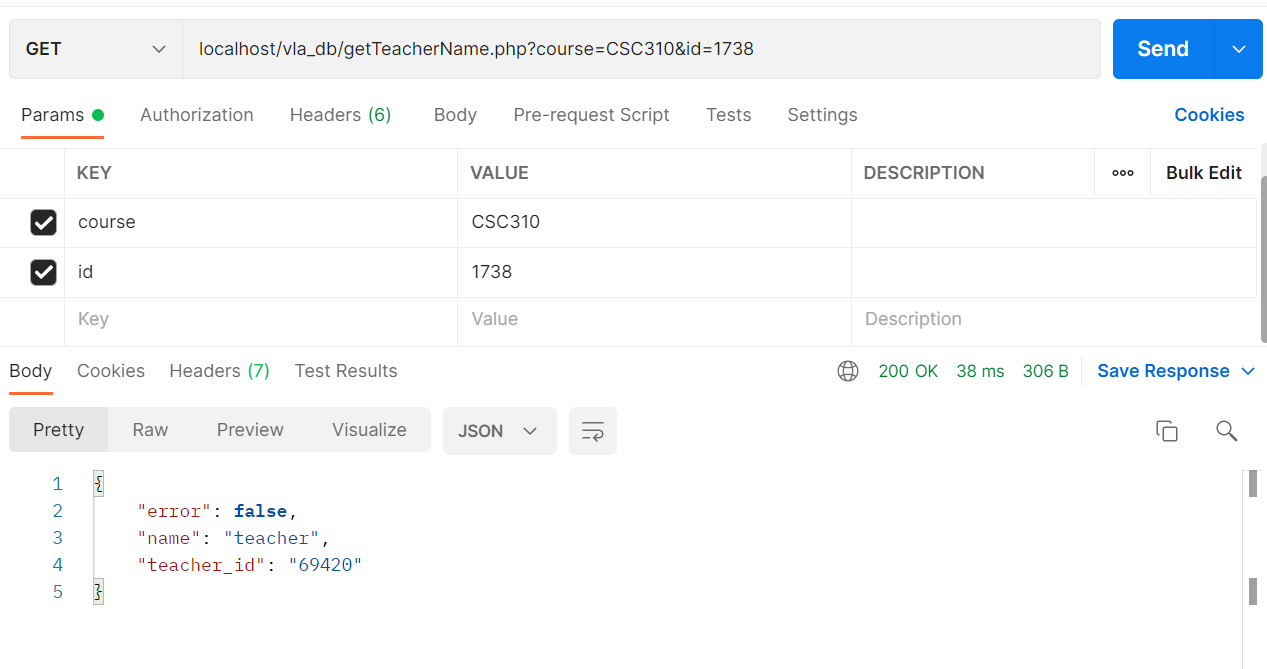


1. getCourses api:

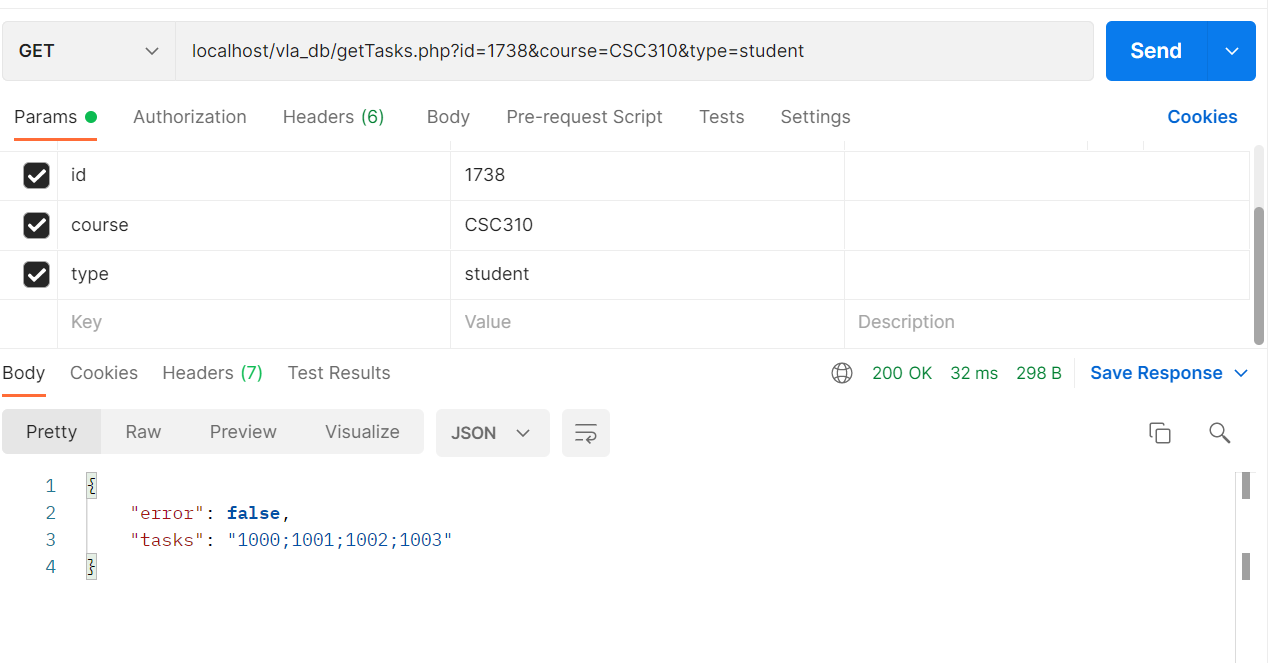
**

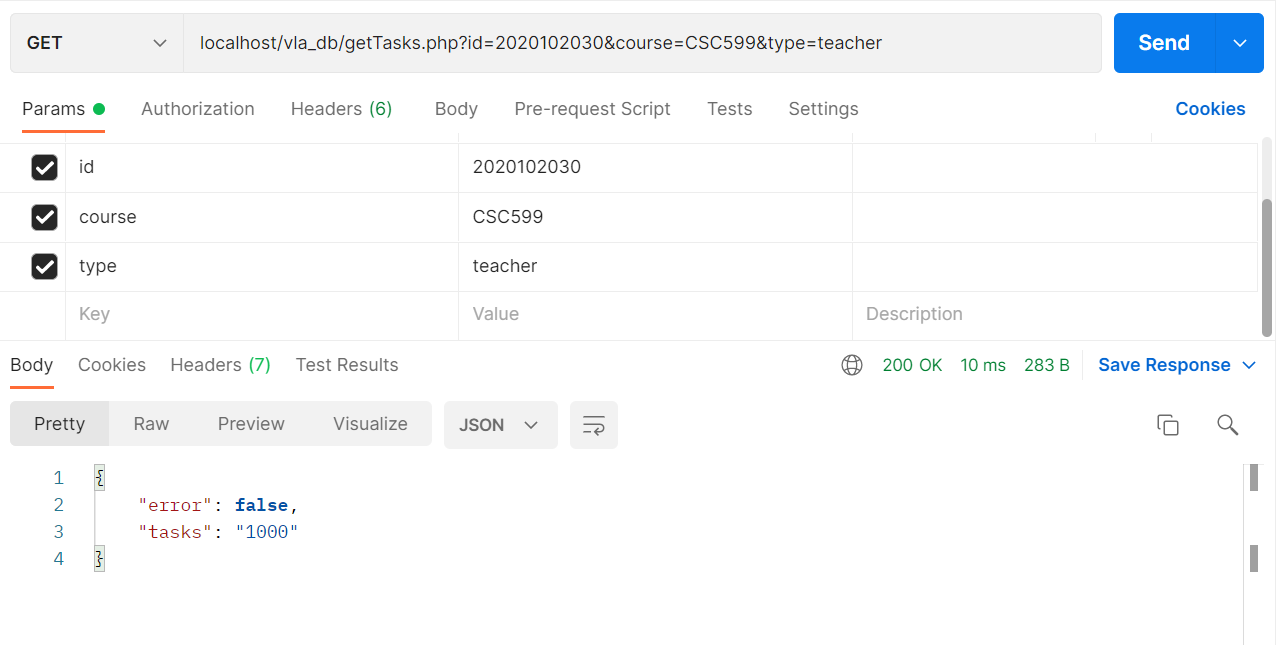
**

1. getTeacherName api:

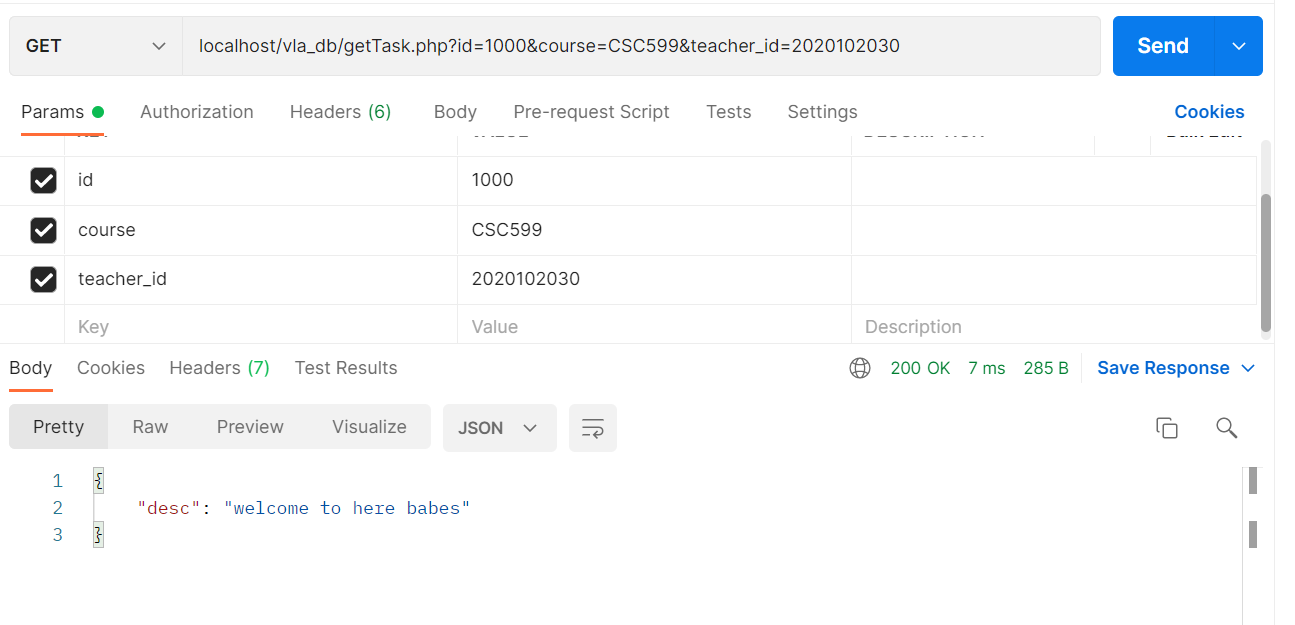
**

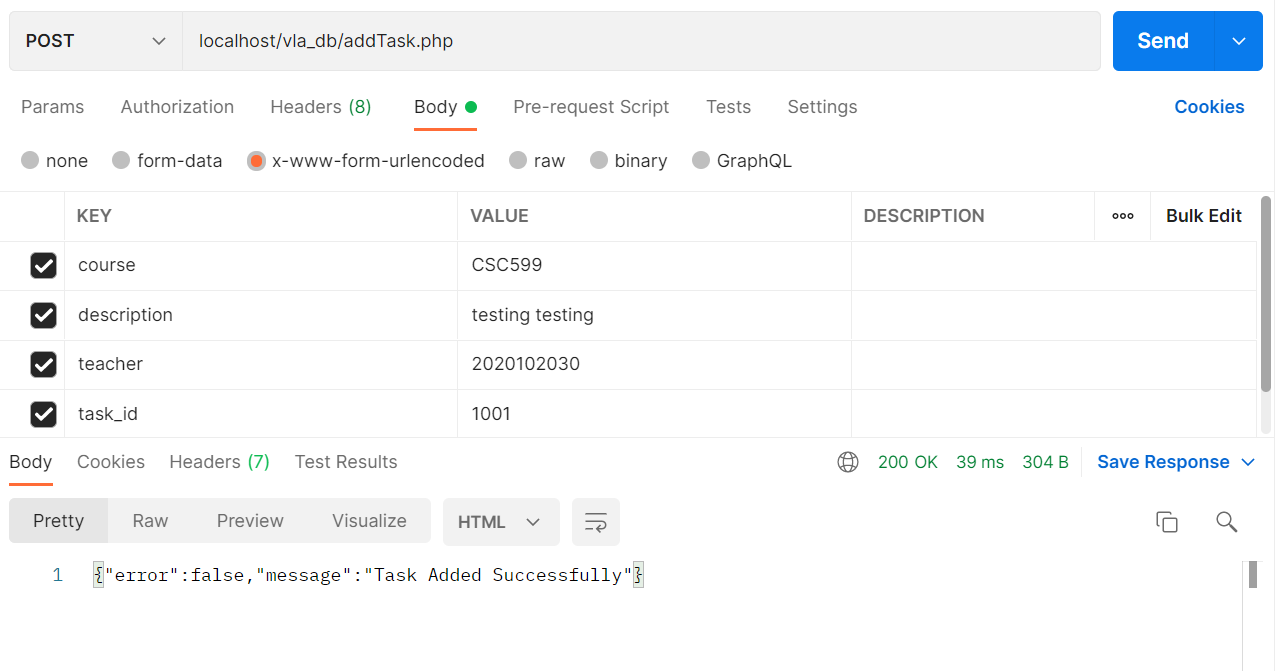
1. getTasks api:

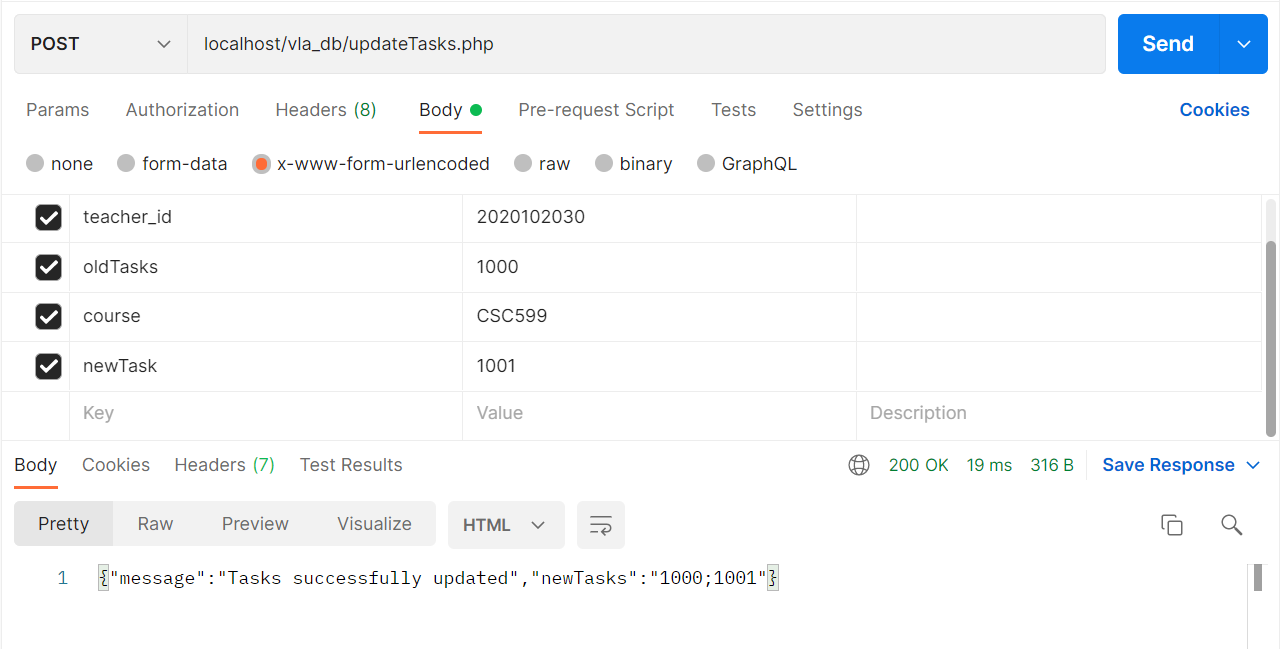
**

**

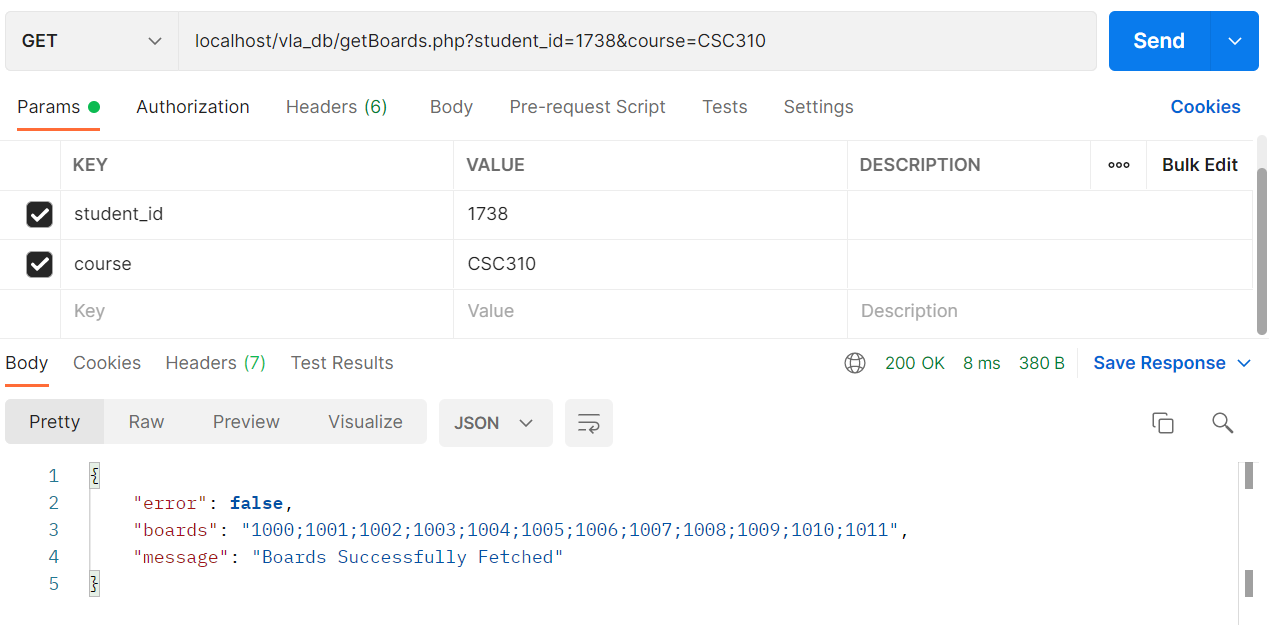
1. getTask api:

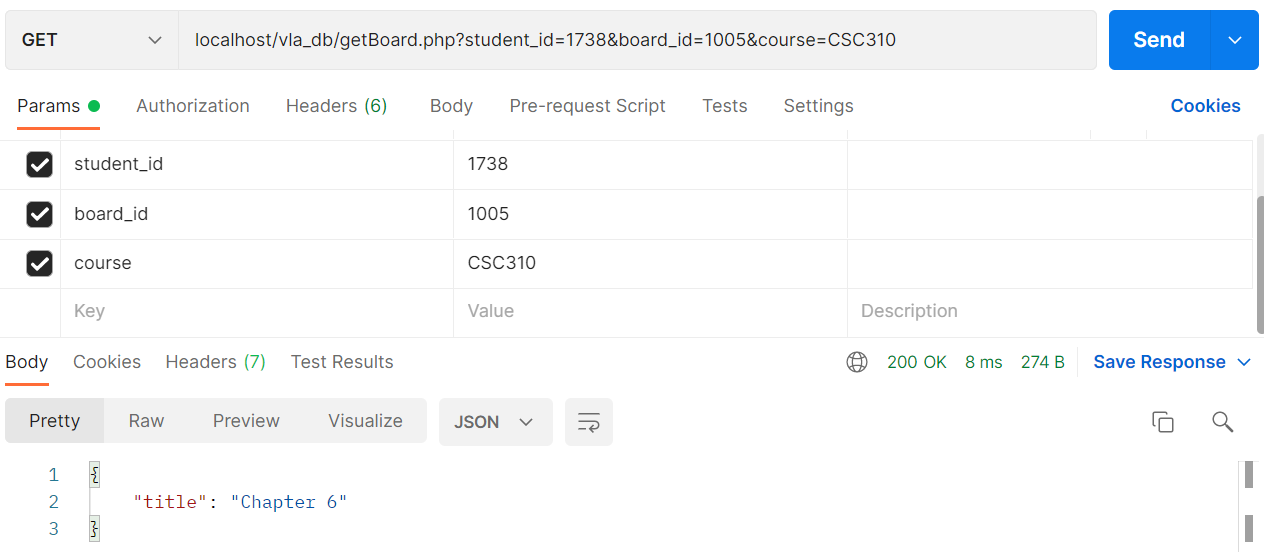
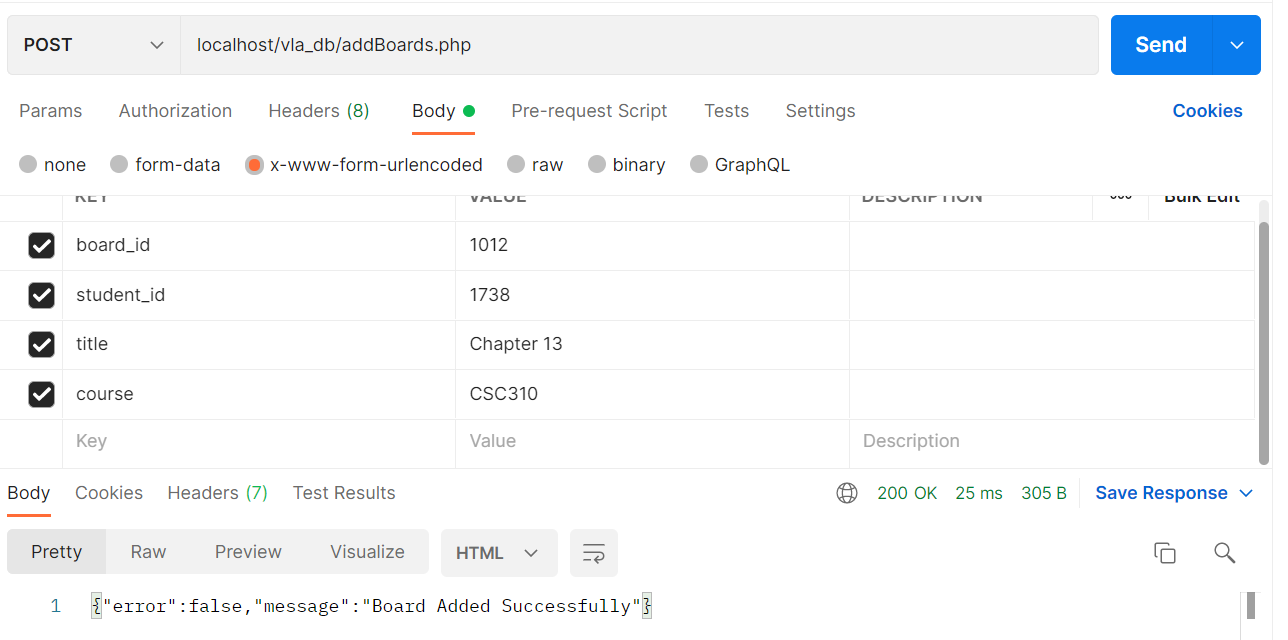
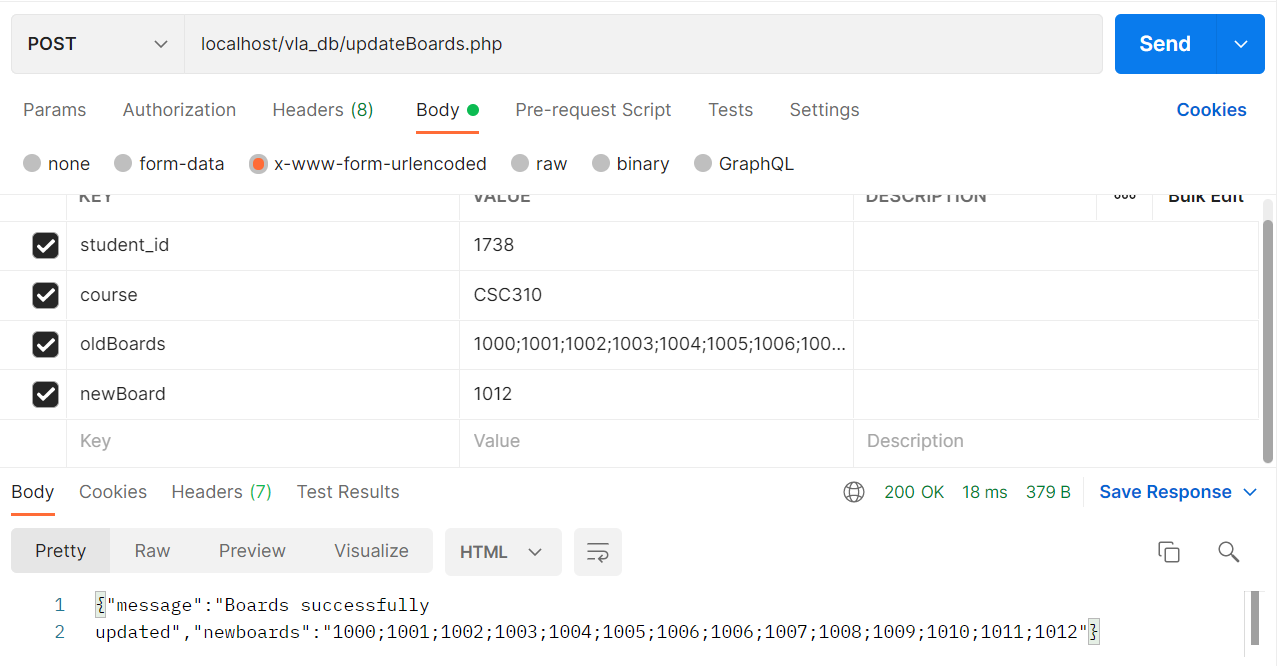
**

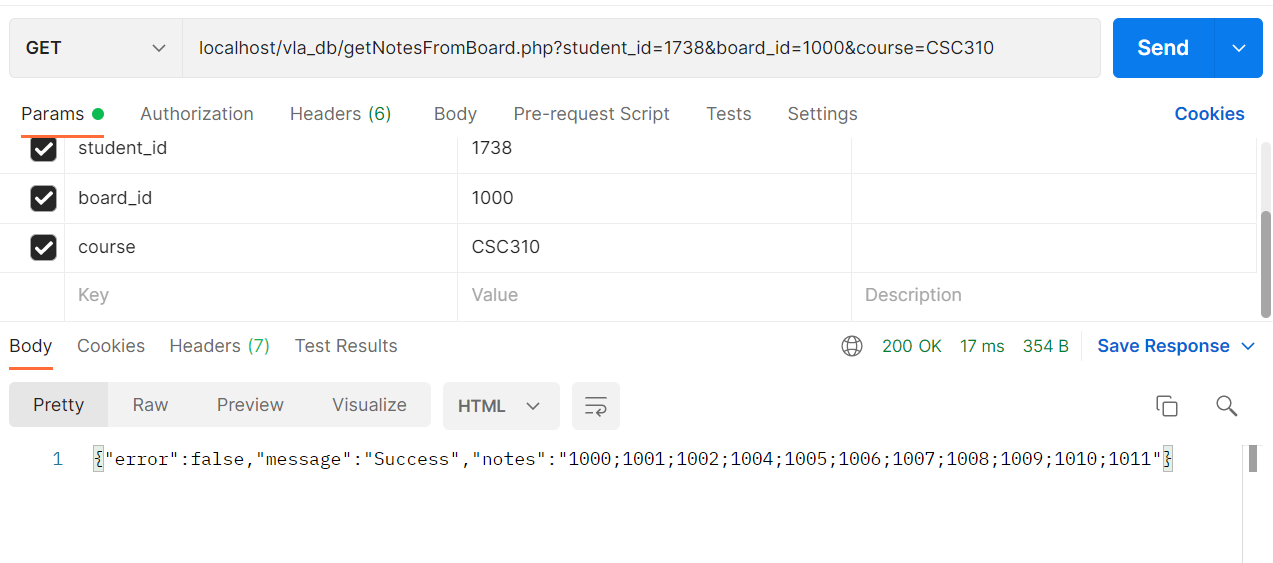
1. **addTask api:
2. updateTasks api:

**

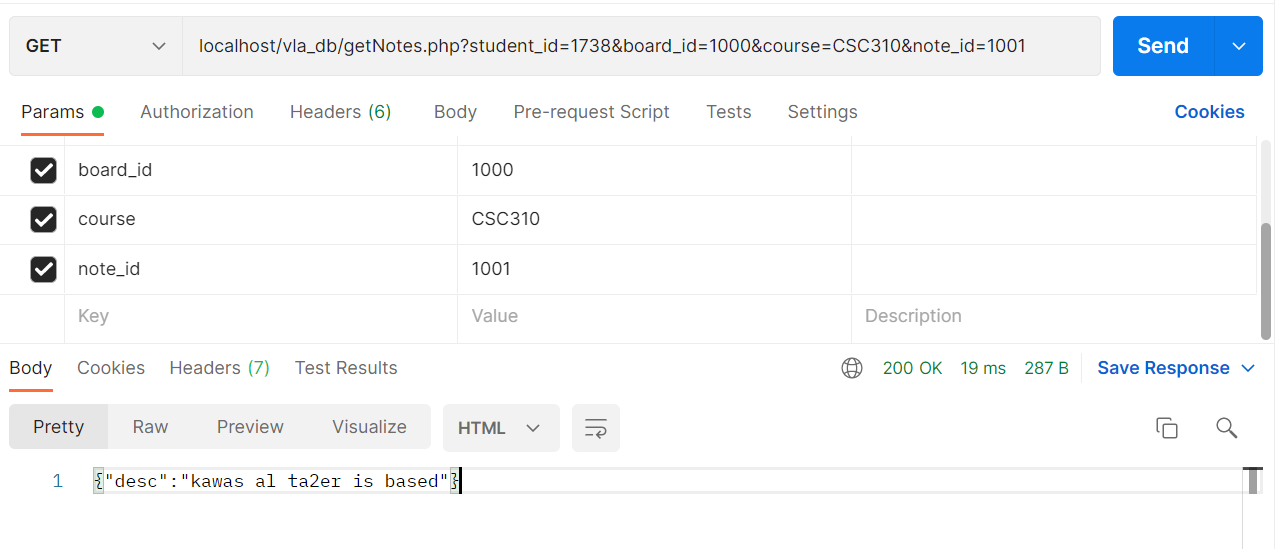
1. getBoards api:

**

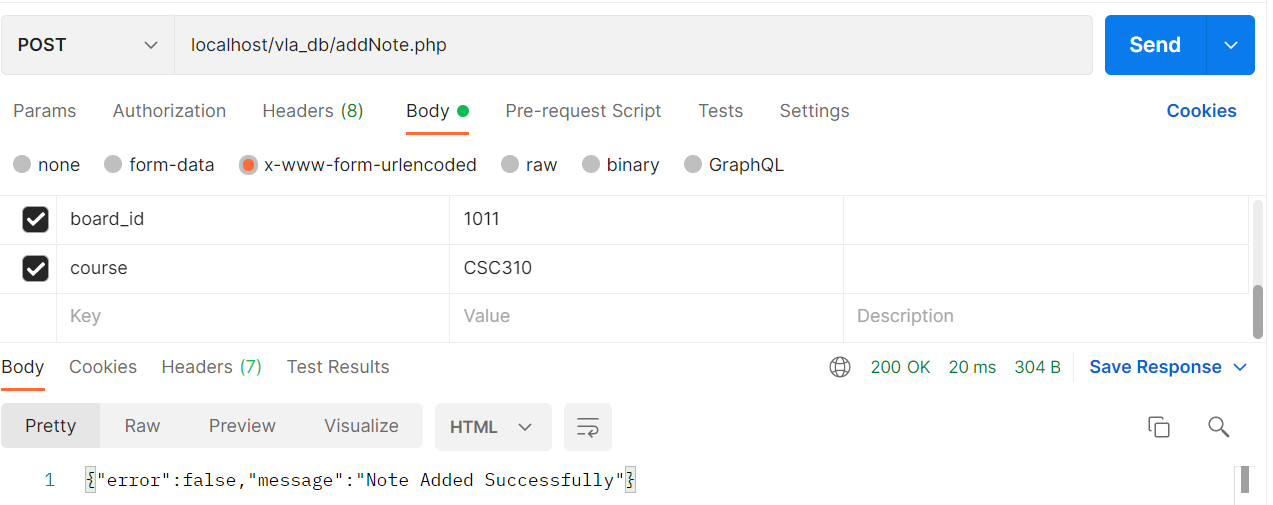
1. **getBoard api:
2. **addBoard api:
3. **updateBoards api:
4. getNotes api:

**

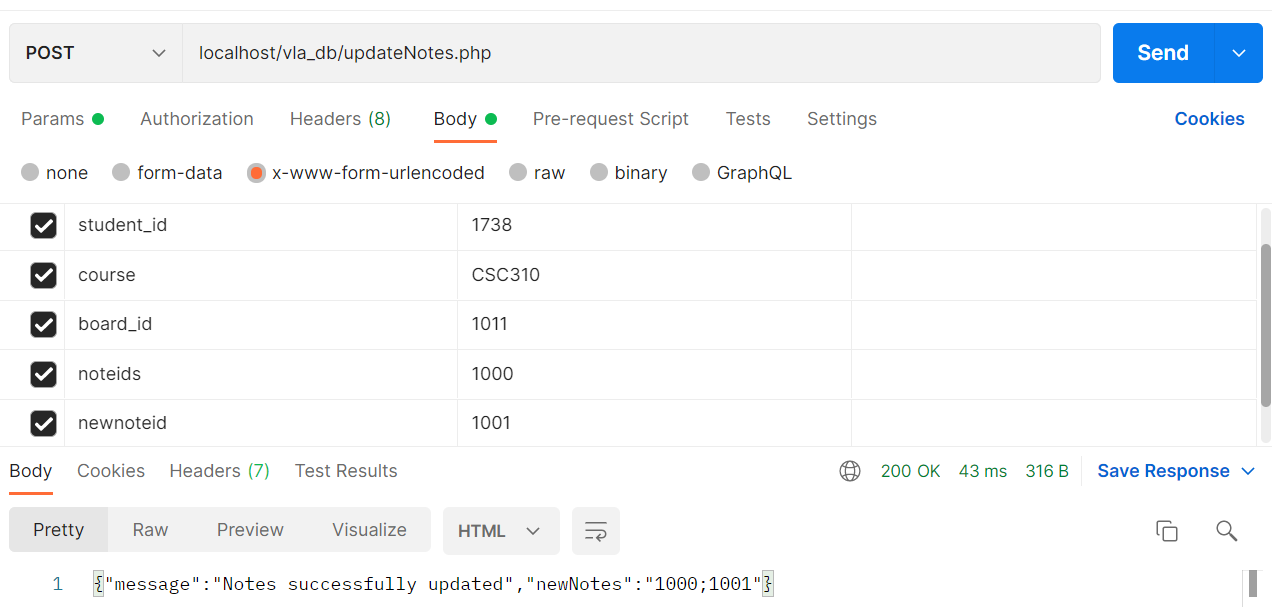
1. getNote api:

**

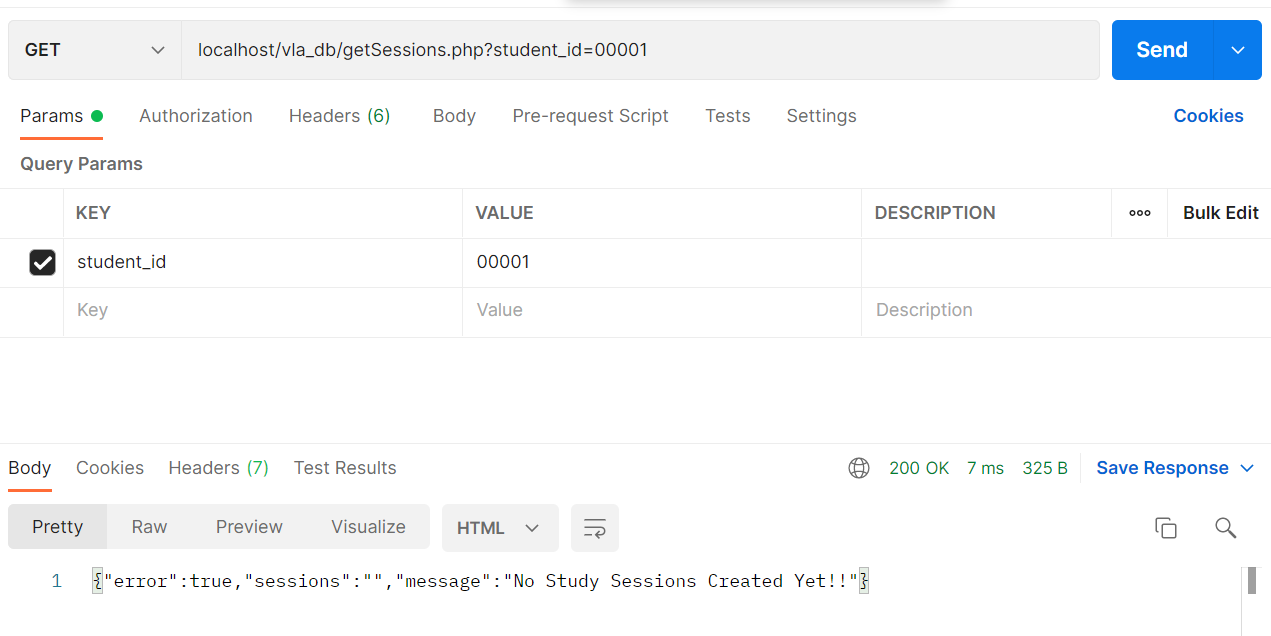
1. addNote api:

**

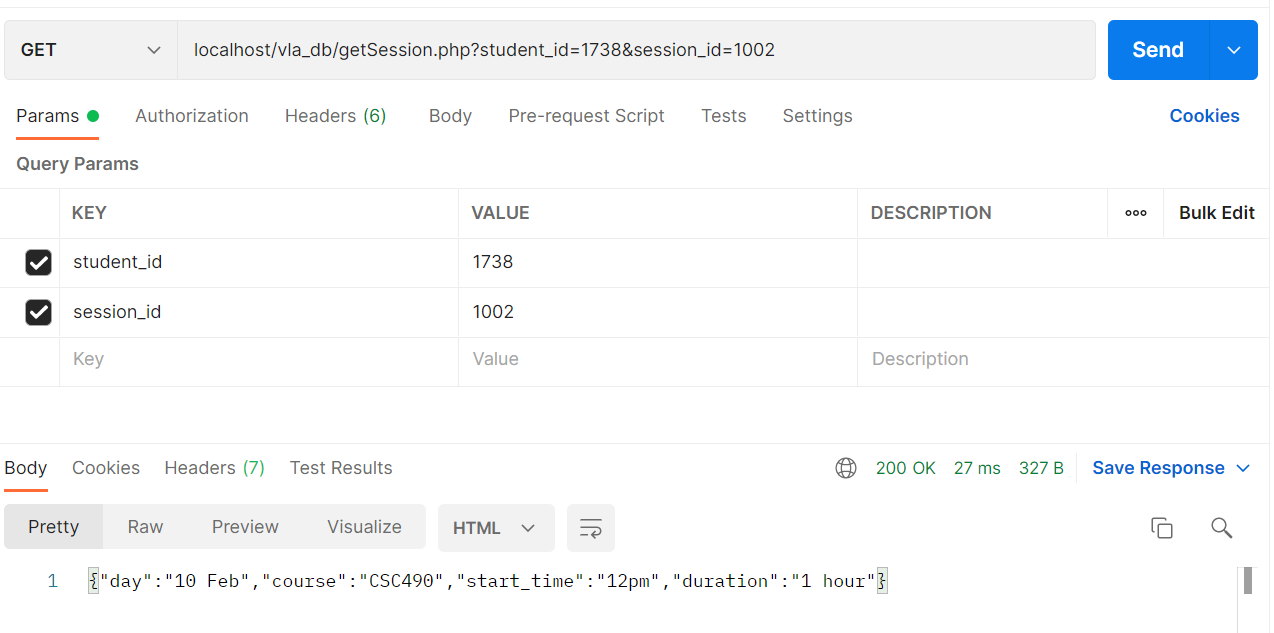
1. updateNotes api:

**

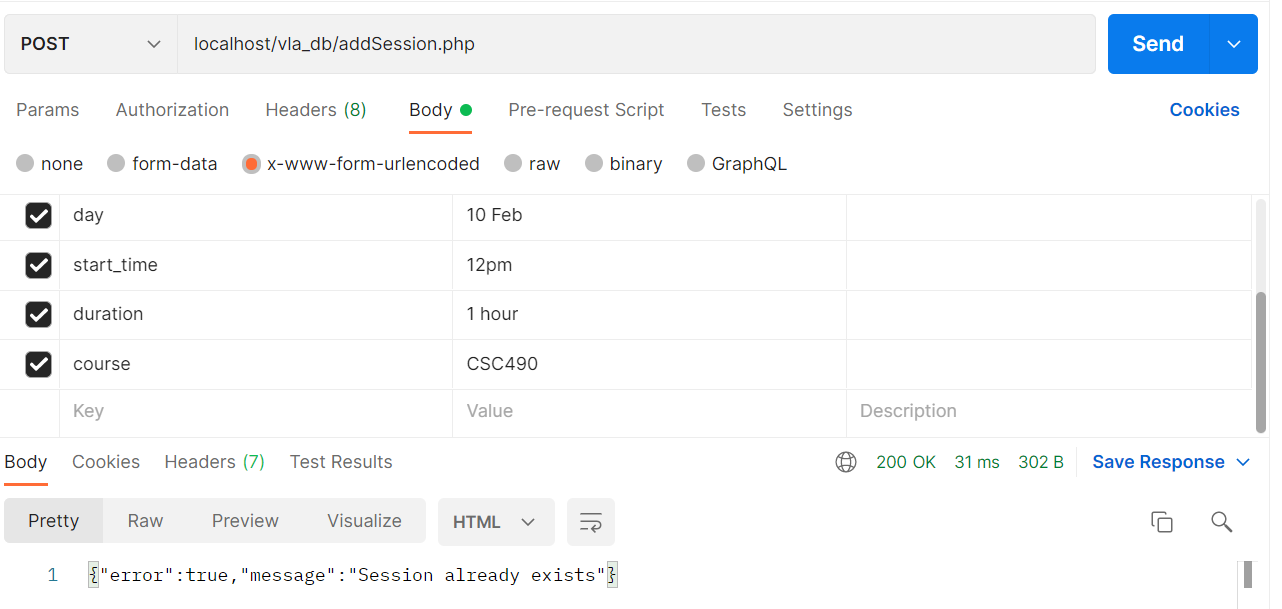
1. getSessions api:

**

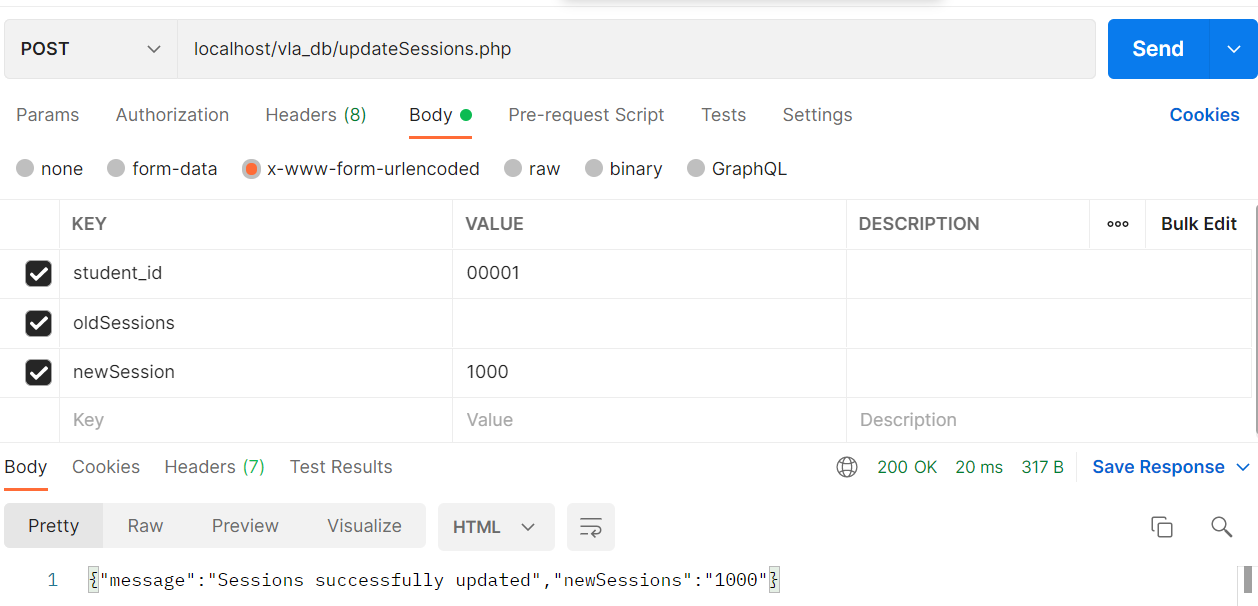
1. getSession api:

**

1. addSession api:

**

1. updateSessions api:

**

# Conclusion:

because our case is fully dynamic, we can’t actually do unit testing on our software, but instead integration testing. all our functions we specified in the class diagrams WERE transformed from functions to apis (application program interfaces) which are our linking from our database, which represents our classes as data storage, to our user experience and user interface (frontend and backend).the demo is fully functional and is saved into my (mohamad kawas github profile, link will be available at the end). we hope that it meets your expectations. i, mohamad kawas, have dedicated a lot of time and effort in doing the final phase and hope you like it, it is right now 7 am before our final finishing this project. I lost count of days after not sleeping for the past week only for 7 hours due to exams and other projects. thank you dr. khaleel for giving us this course.

😊