**T.C**

**Marmara University**

**Faculty of Engineering**

**Computer Engineering Department**

CSE4074 – Project Report

**Socket Programming – HTTP-based Room Reservation**

**Group Members**

150119905 Fatih Genç

**Supervisor**

Doç.Dr. Ömer Korçak

Jan. 03, 2023

# General View

First of all, we tried to set up the database structure. We created activity table, reservation table and room table. We added the name variable to the room table and the name variable to the activity table. Finally, we added the room name, activity name, day, hour and duration variables to the reservation table.

If the database file is under the code file, it does not create a new database. But if there is no database file under the code file, it creates an empty database file containing the tables we mentioned above.

# Room Server

Then we started to write the code part of our Room server. First of all, we created an object using the 8080 port over the local host using the socket library. Then we created an area in the loop where we will respond to incoming requests. This is where we first split the incoming request into parts by performing a split operation. We checked if there is a get request. If we got the get request, we looked at the other parts of the request. In order to distinguish the operation we need to do, we used split according to the "/" character. We created conditions according to the command of the incoming request. When the Add operation comes, it checks whether the room table in our database is already added. It sends 403 if attached. If it is not attached, it saves it to our database.

We checked our database again for the Remove operation. If the entered data is not available in our database, we sent a 403 message. If attached, we deleted this data from our database. And in the reservation table, if a reservation was made, we also deleted this data.

To make a reservation from the room server, we first checked our database to see if the room was available. If there is, we checked whether a reservation was made according to the days and hours given in advance. If a reservation has been made, we have sent a 403 message. If not, we added it to our database. For the check availability operation, we found and printed the available hours of the room according to the given day. If the room does not have free time, we print that there is no free time.

# Activity Server

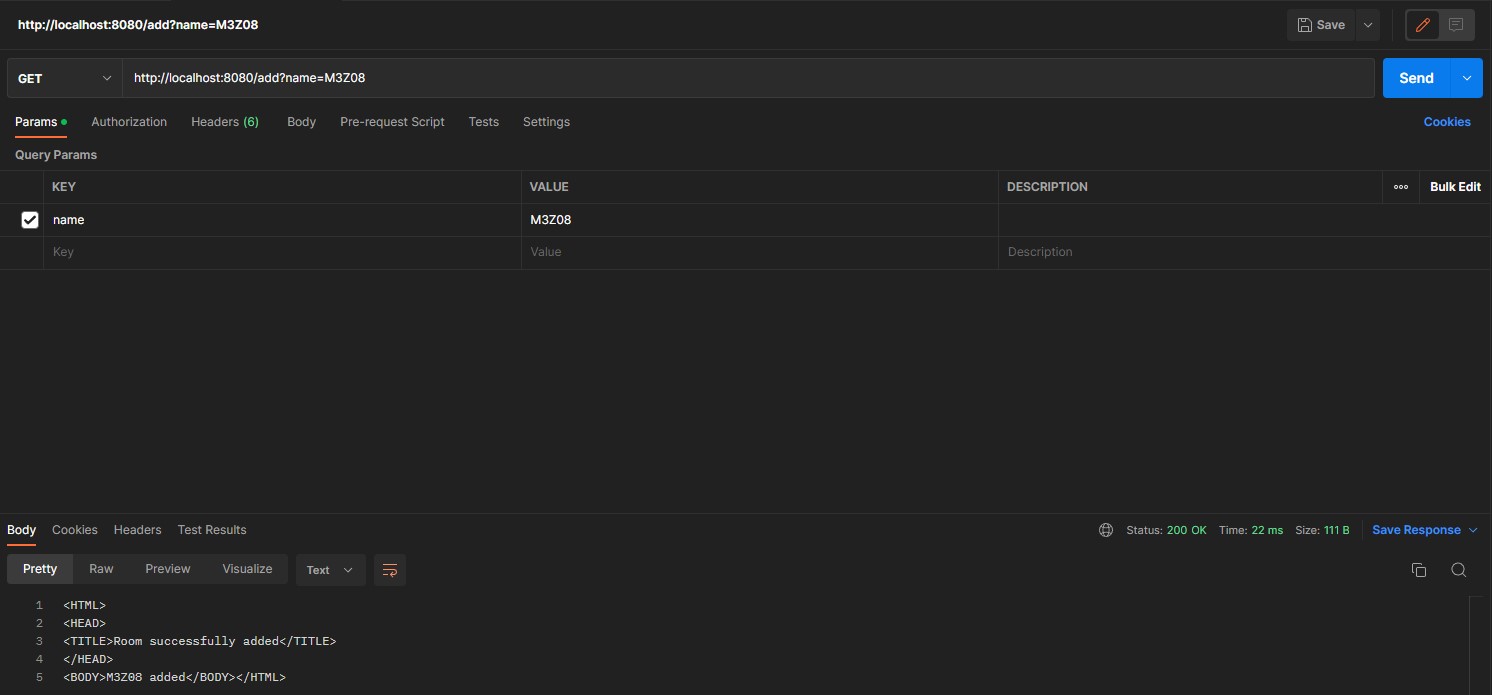
We checked the database whether there is such an activity in the add operation for the activity server. If there is, we print an error message. If not, we added the activity to our database. For the remove operation, we checked our database in the same way and printed the necessary messages. For the check operation, we looked at our database again and checked whether it existed. Accordingly, we have printed the necessary messages.

# Reservation Part

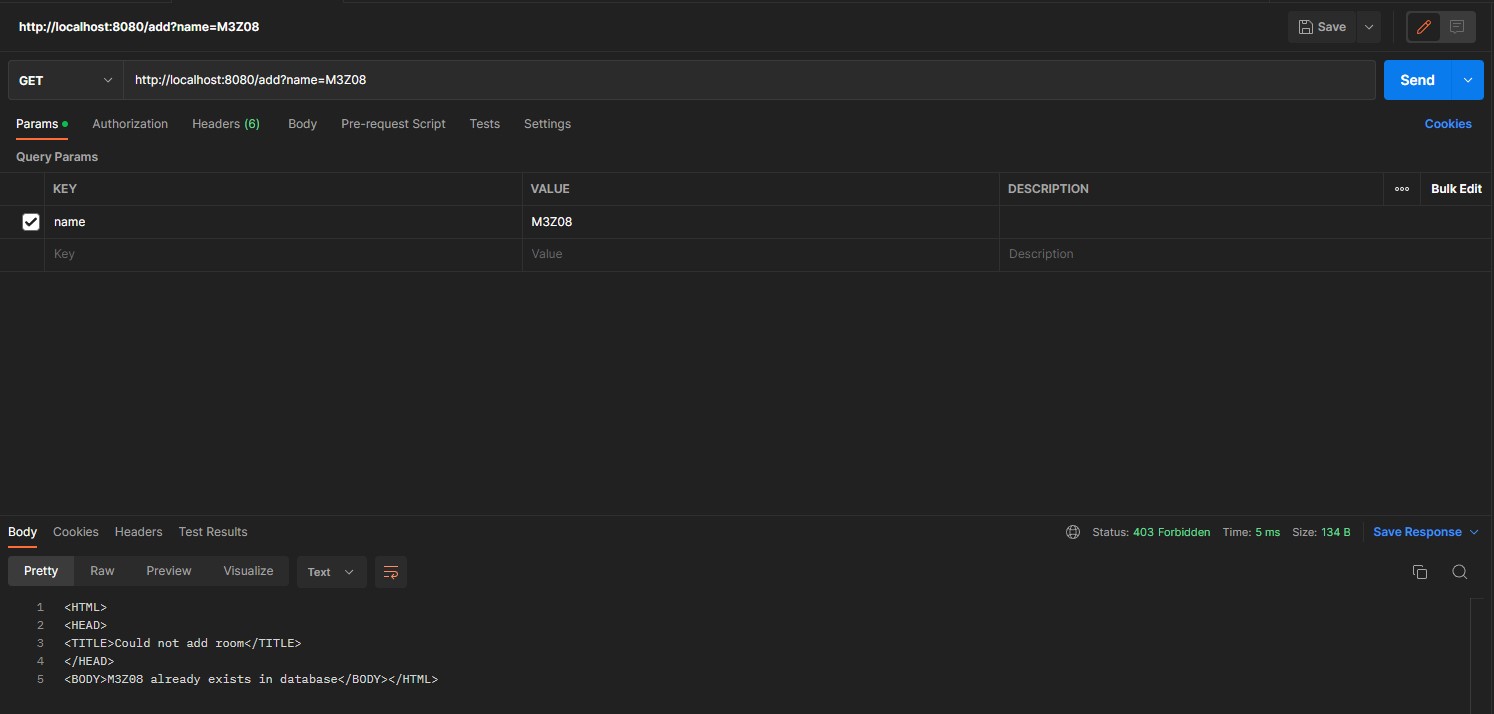
In the reservation server part, we first sent a request to the activity server for the reservation operation. According to the message returned from the activity server, we understood whether there is such an activity. If the activity is not available, we print an error message. If the activity is available, we sent a request to the Room server. According to the request from there, we understood whether the data we entered was suitable or whether it was reserved beforehand. Accordingly, we responded to our messages. For the list availability operation, we first checked how many variables were sent. We divided this operation into two contions. If it has two variables (roomname, day) it enters one of them (roomname) if it has only one variable. If there are two variables, Room servara sends a request and splits the content of the incoming message, takes the available hours and sends the necessary message. If there is no room, it prints an error message. If it has a single variable, it sends requests from 1 to 7 for all days of the week. It records the messages of these requests and prints the available hours of all days as a single message when the days are over. For the display method, we first check our database. If the id is not available, we print an error message. If available, we print the details of the data we find. Finally, we ran these 3 servers at the same time using threads.

# Sample Output

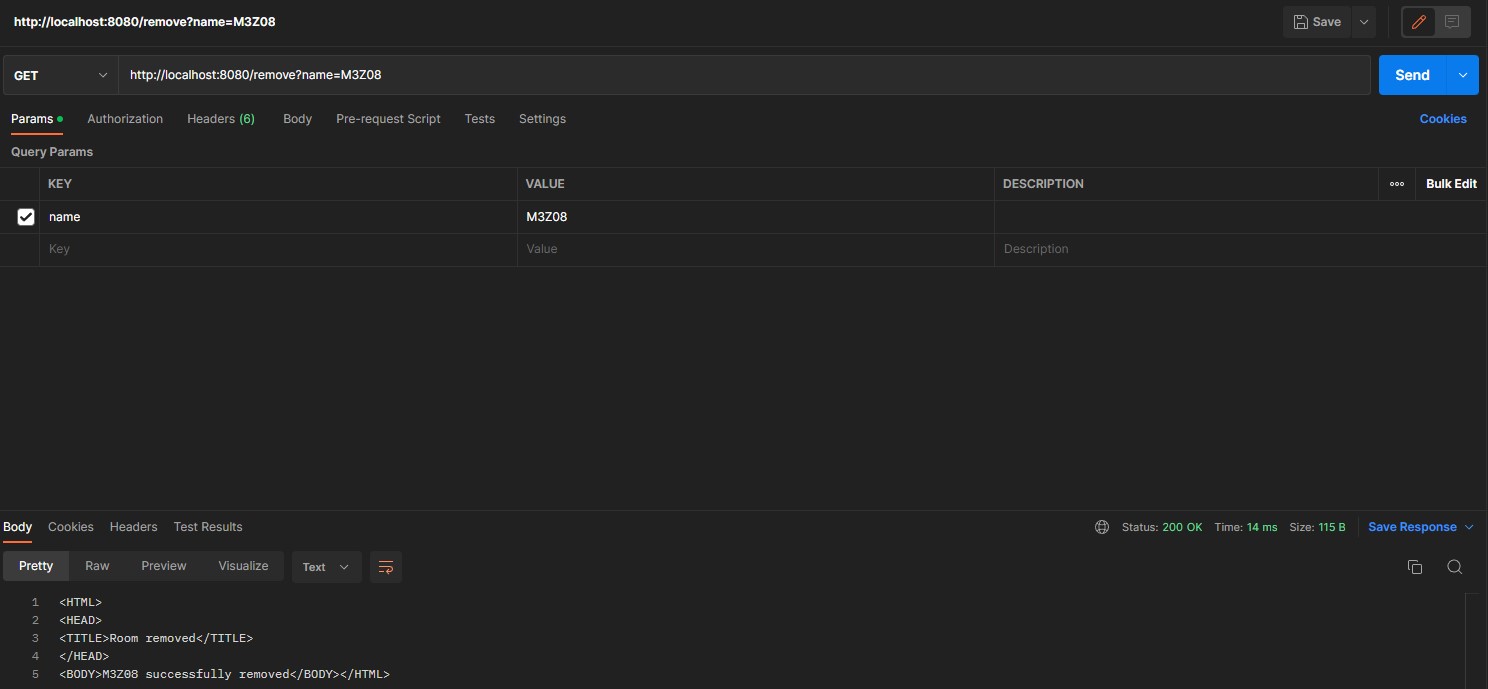
1. Adds the room to the database, and sends back a 200 OK message including an HTML file in the body that informs that the room is added.



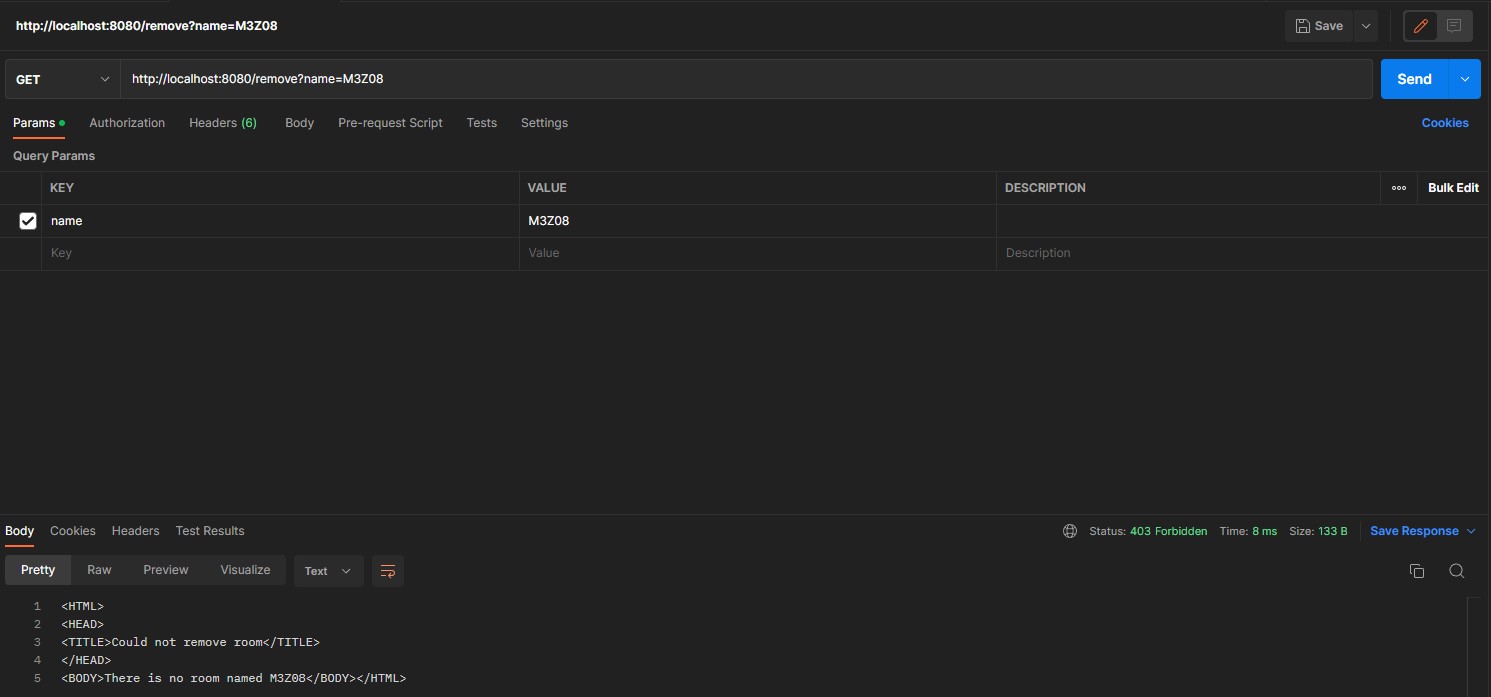
1. If the room is already added, this server should return an HTTP 403 Forbidden message indicating that the room already exists in the database.



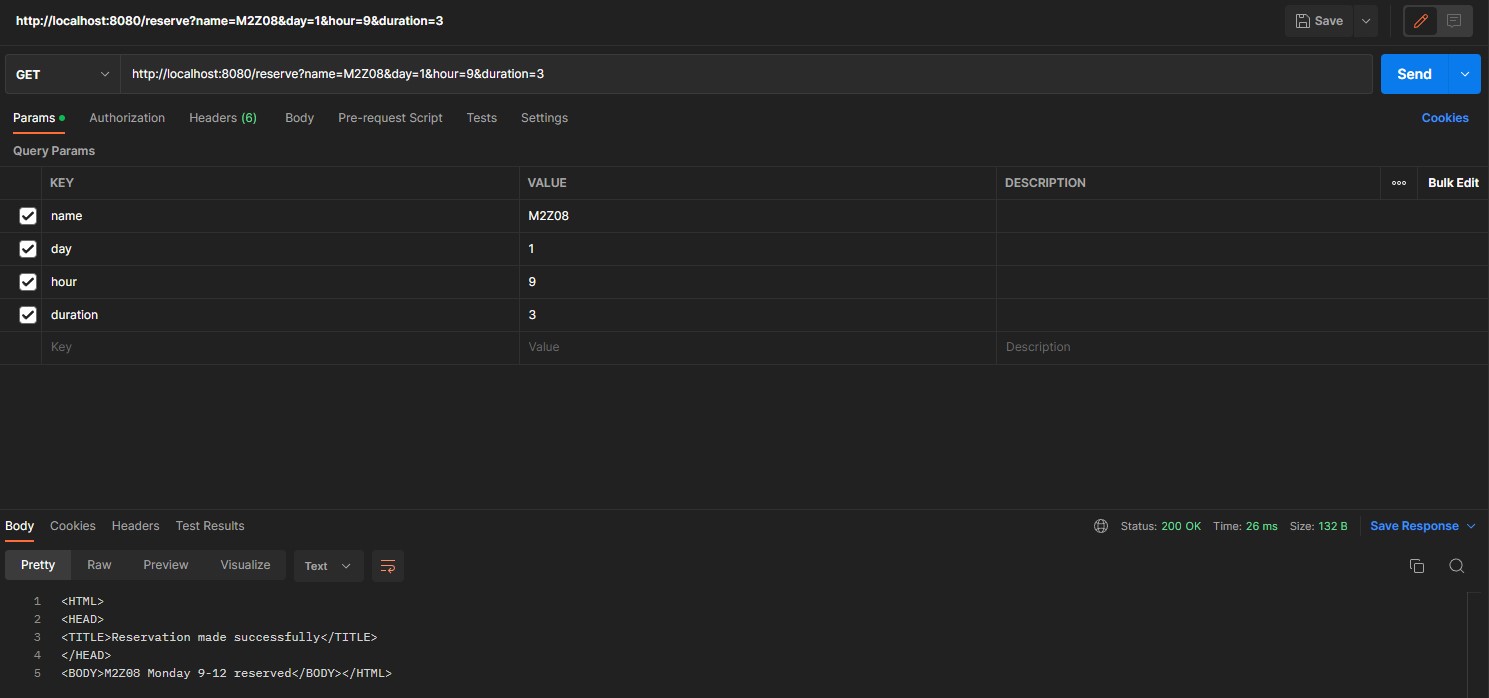
1. If the room exists, it removes the room from the database and sends back HTTP 200 OK message



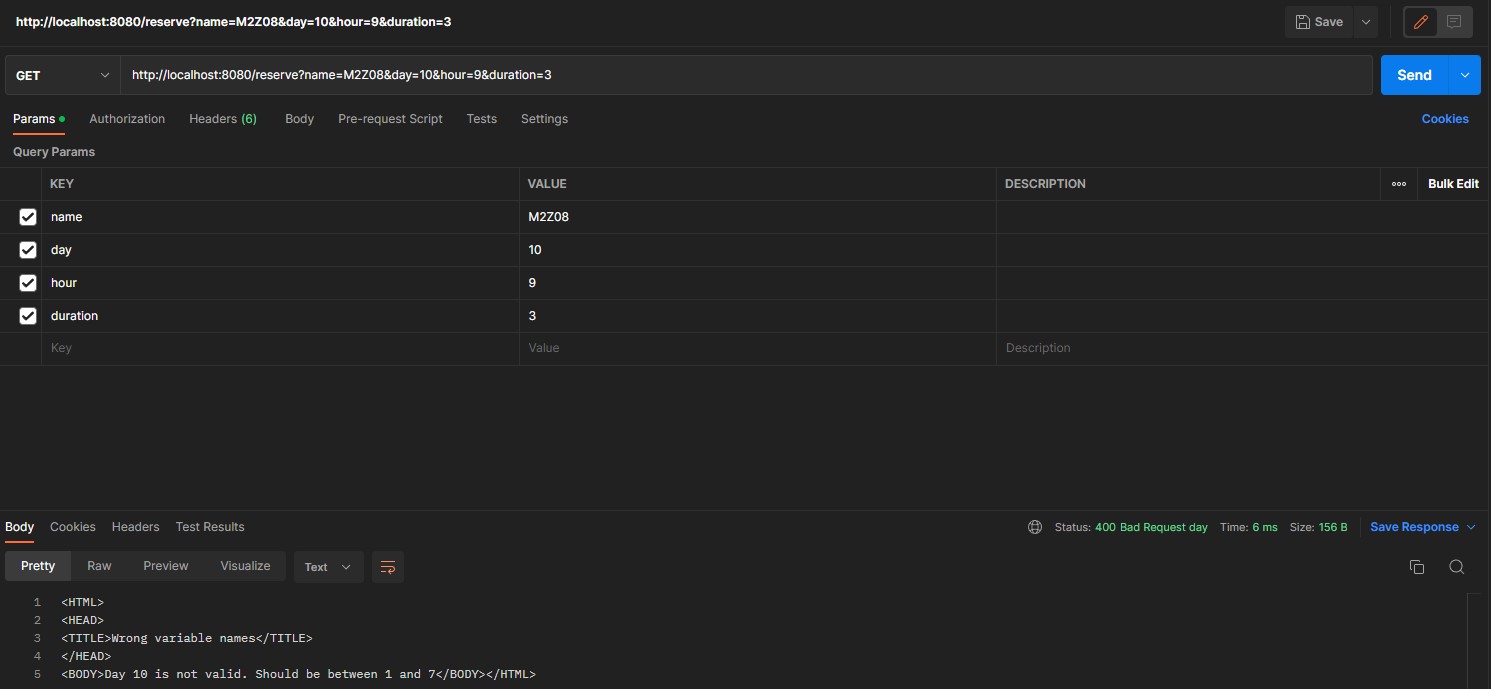
1. If the room doesn’t exist, then it sends back an HTTP 403 Forbidden message.



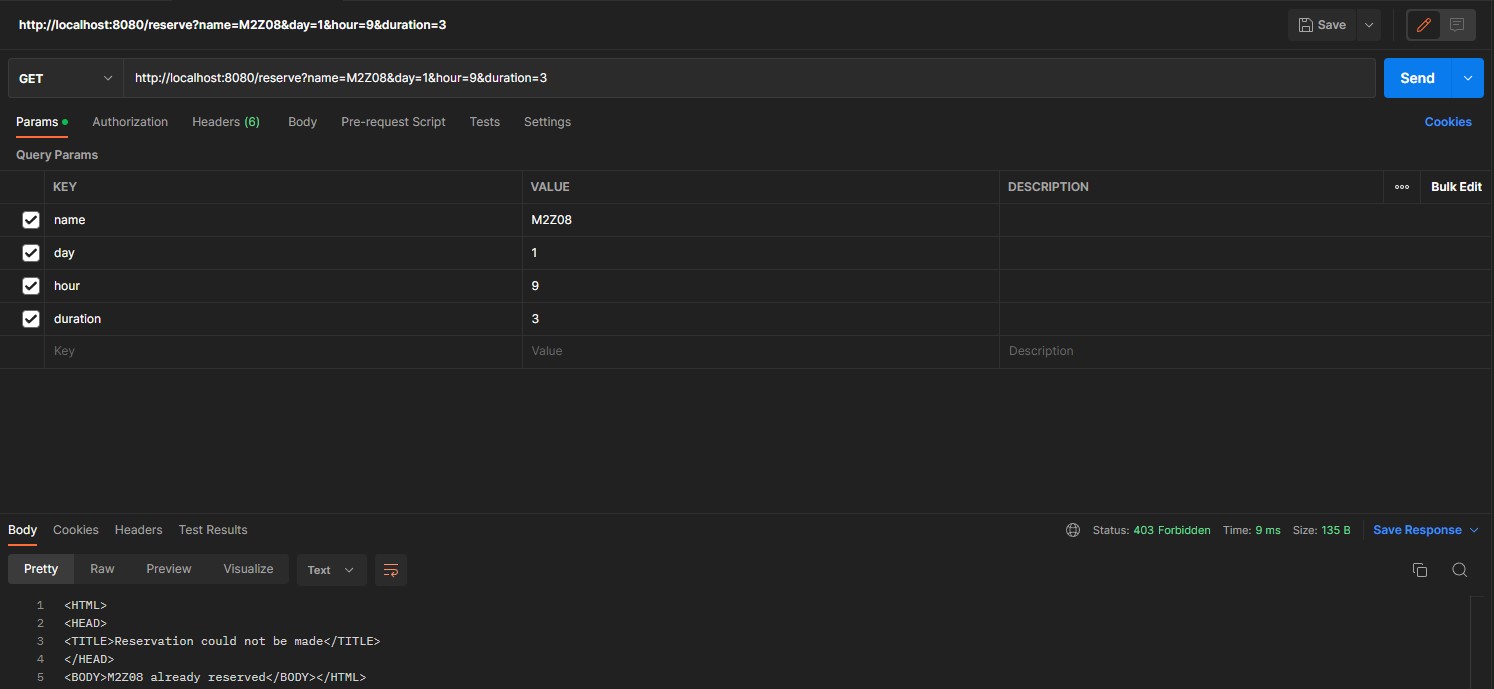
1. Reserves the room with name=roomname for the day x of the week at y:00 for z hours.



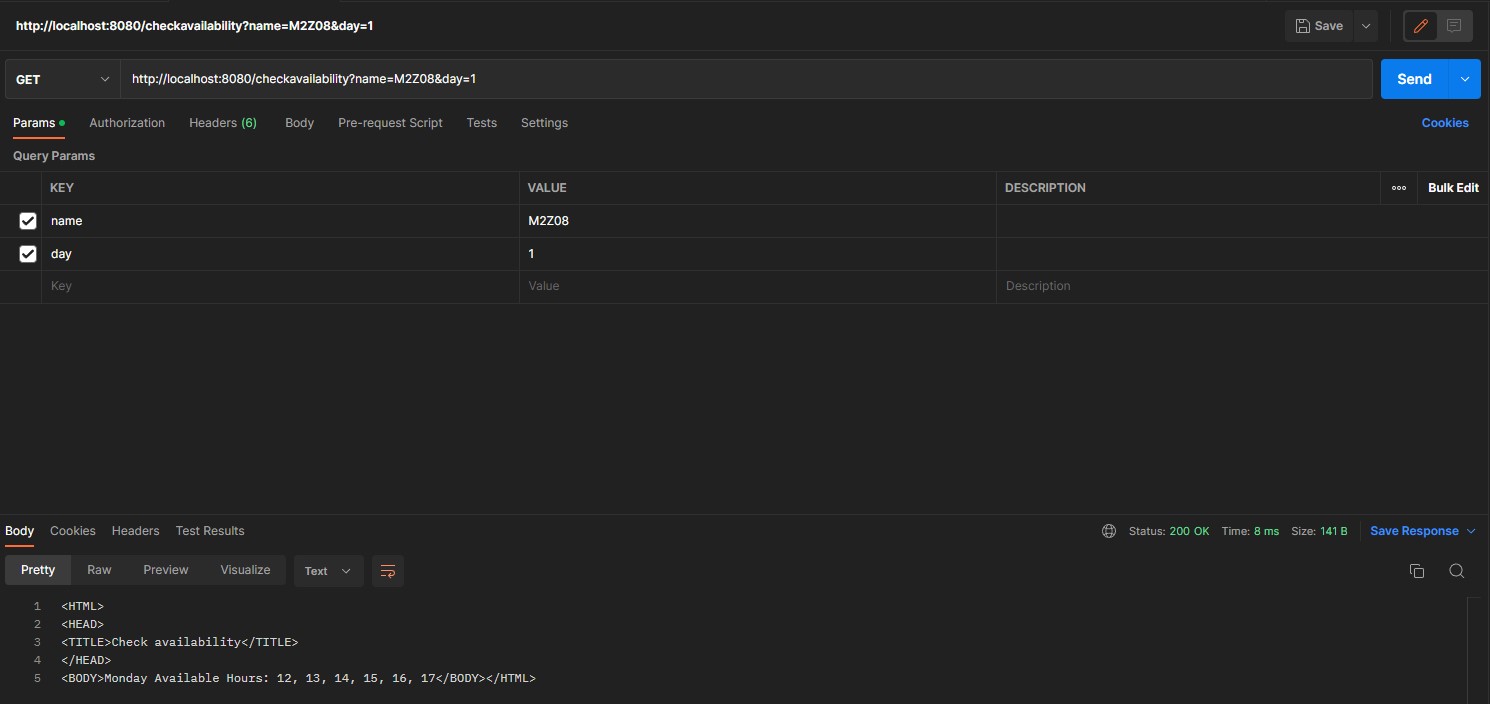
1. If any of these inputs are not valid, it sends back an HTTP 400 Bad Request message.



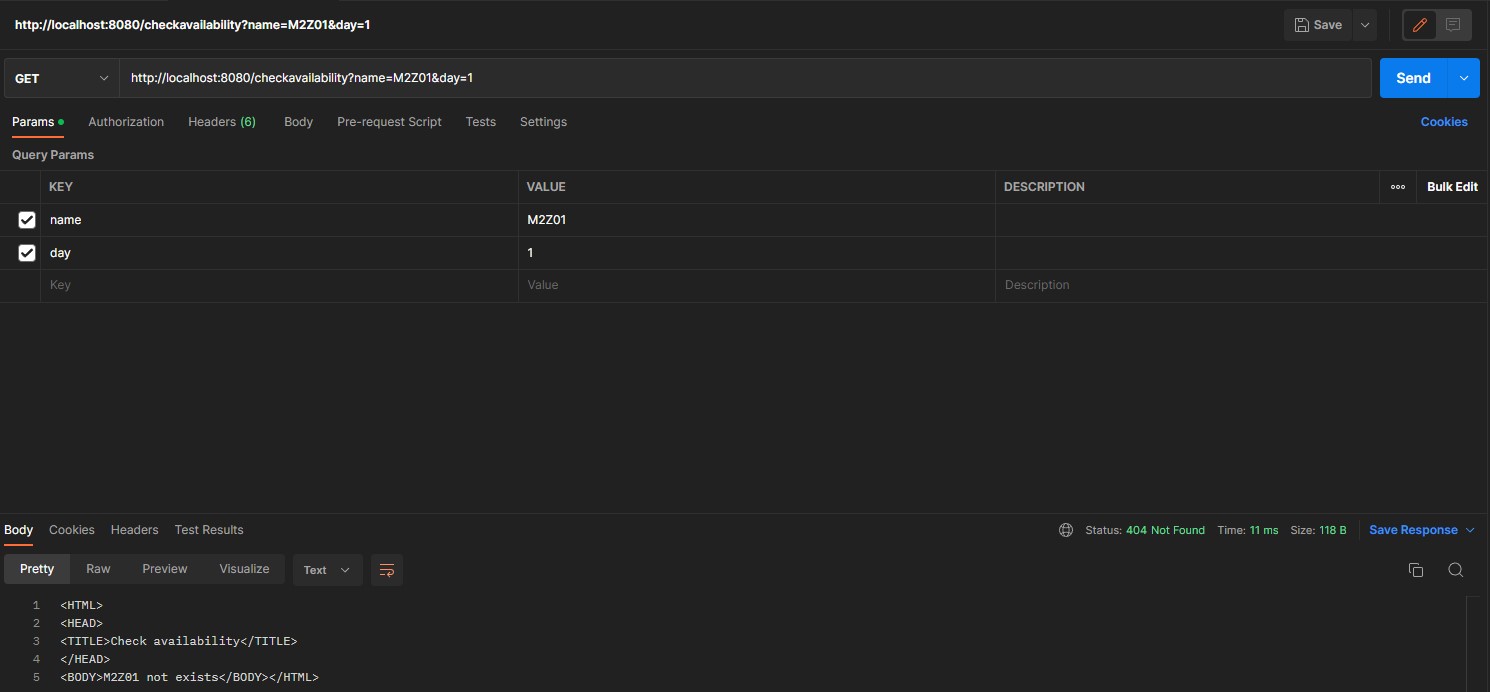
1. If the room is already reserved then HTTP 403 Forbidden message will be sent



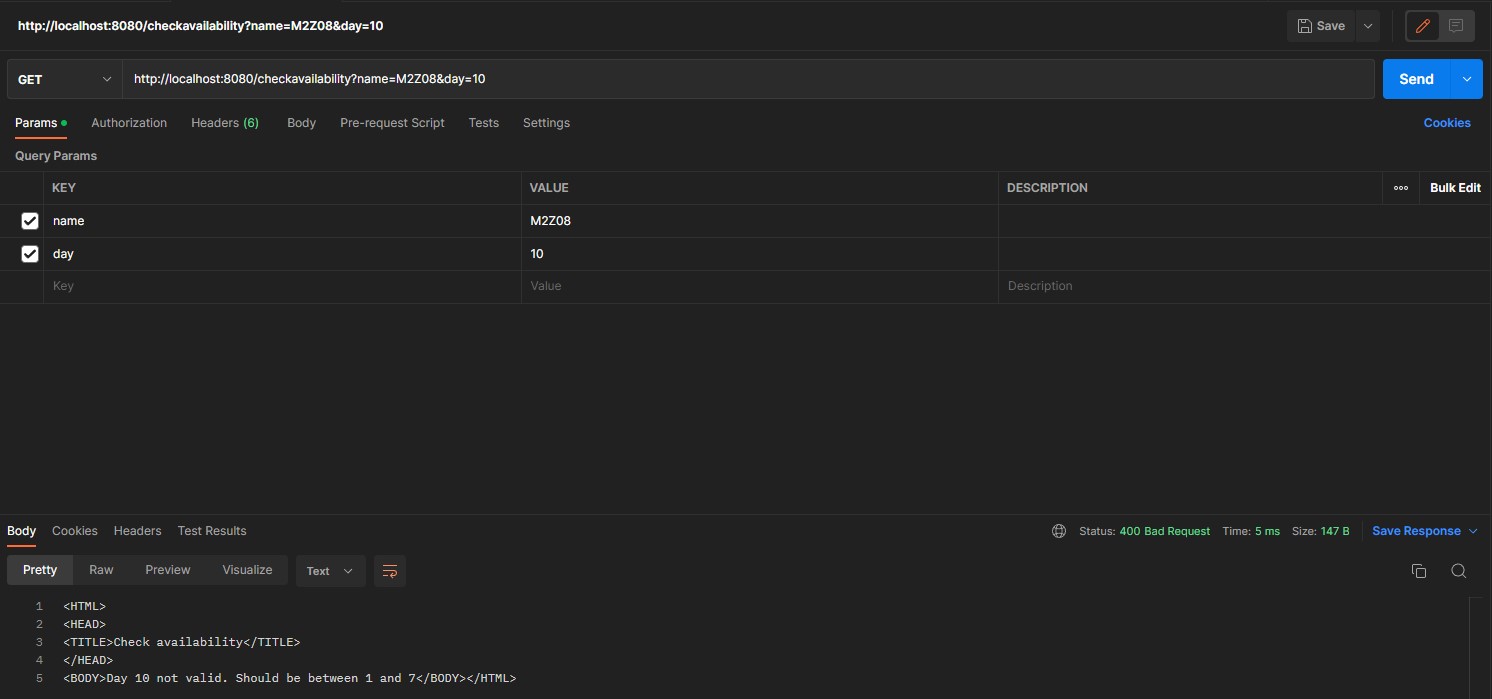
1. Checks the available hours for the room with name=roomname and returns back all the available hours



1. If no such room exists it sends back an HTTP 404 Not Found message

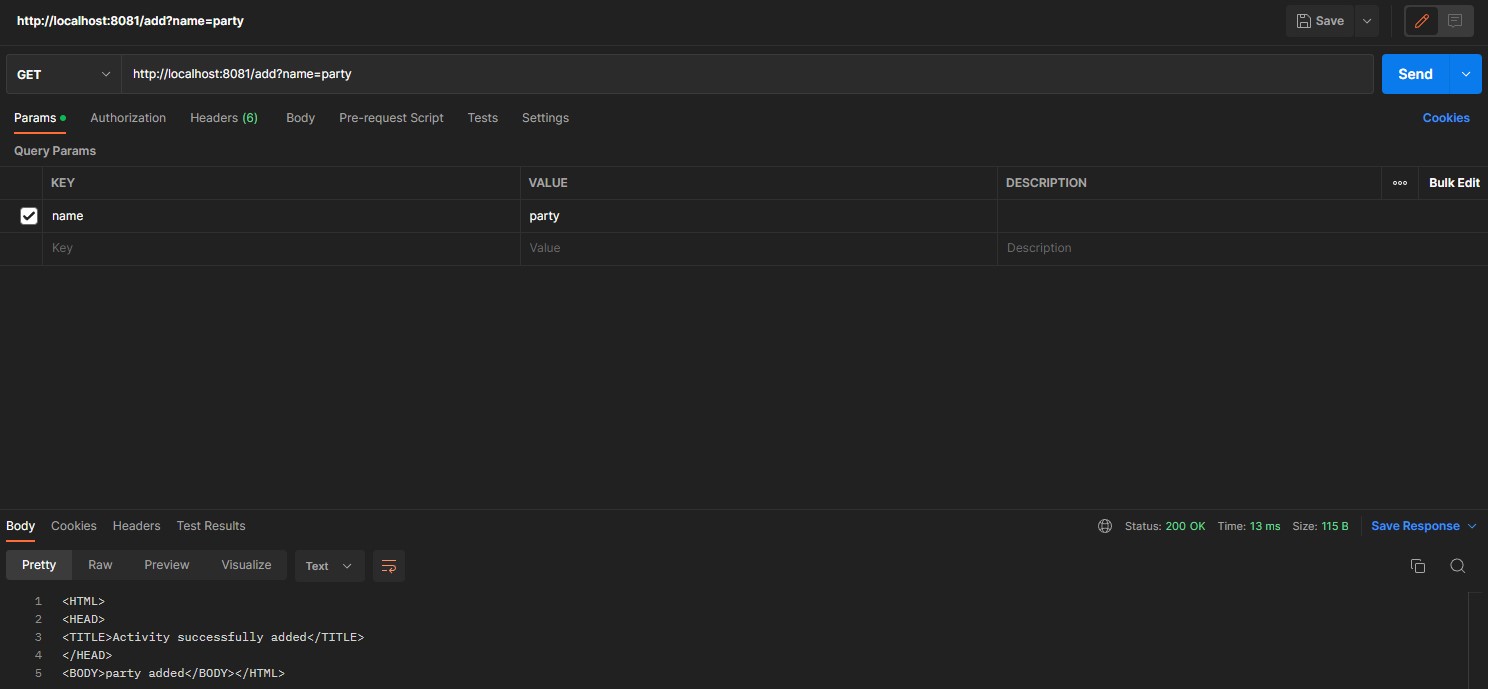


1. If x is not a valid input then it sends back an HTTP 400 Bad Request message.

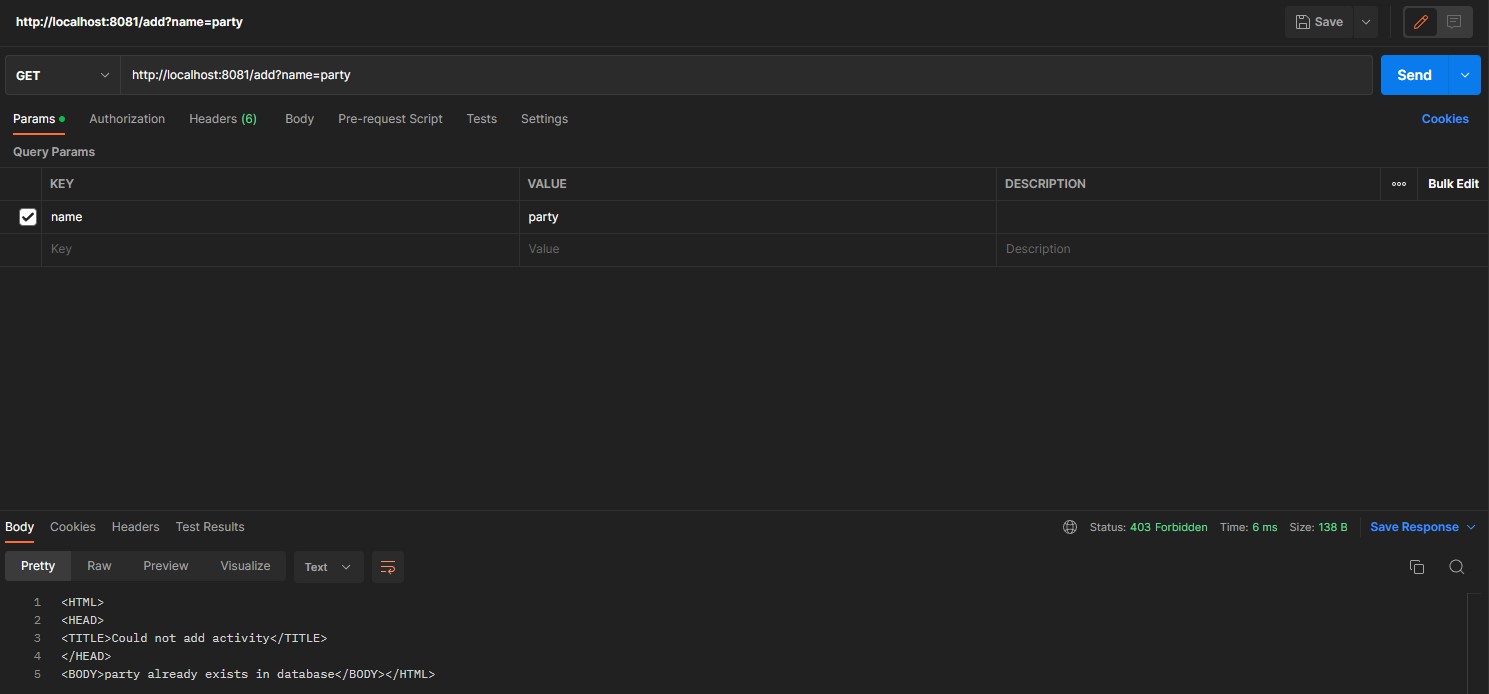


# Activity Server

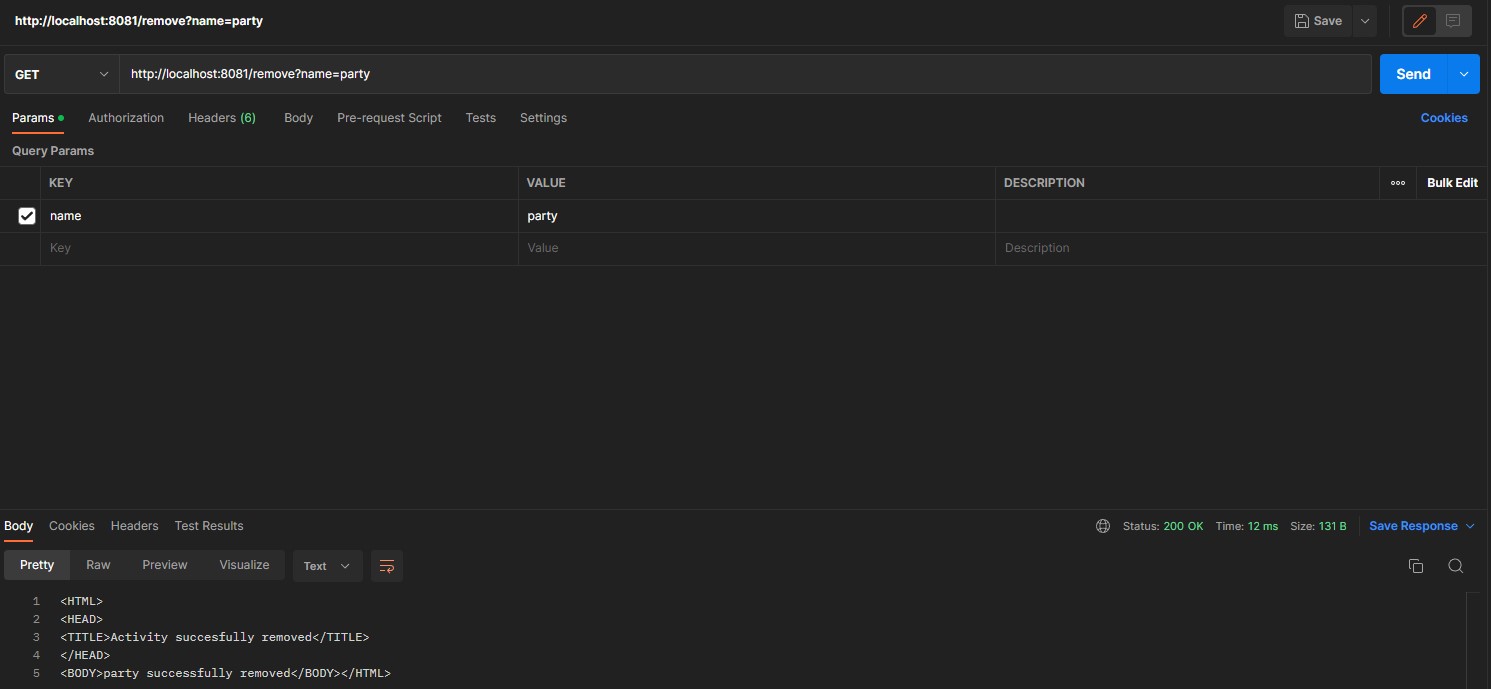
1. Adds new activity



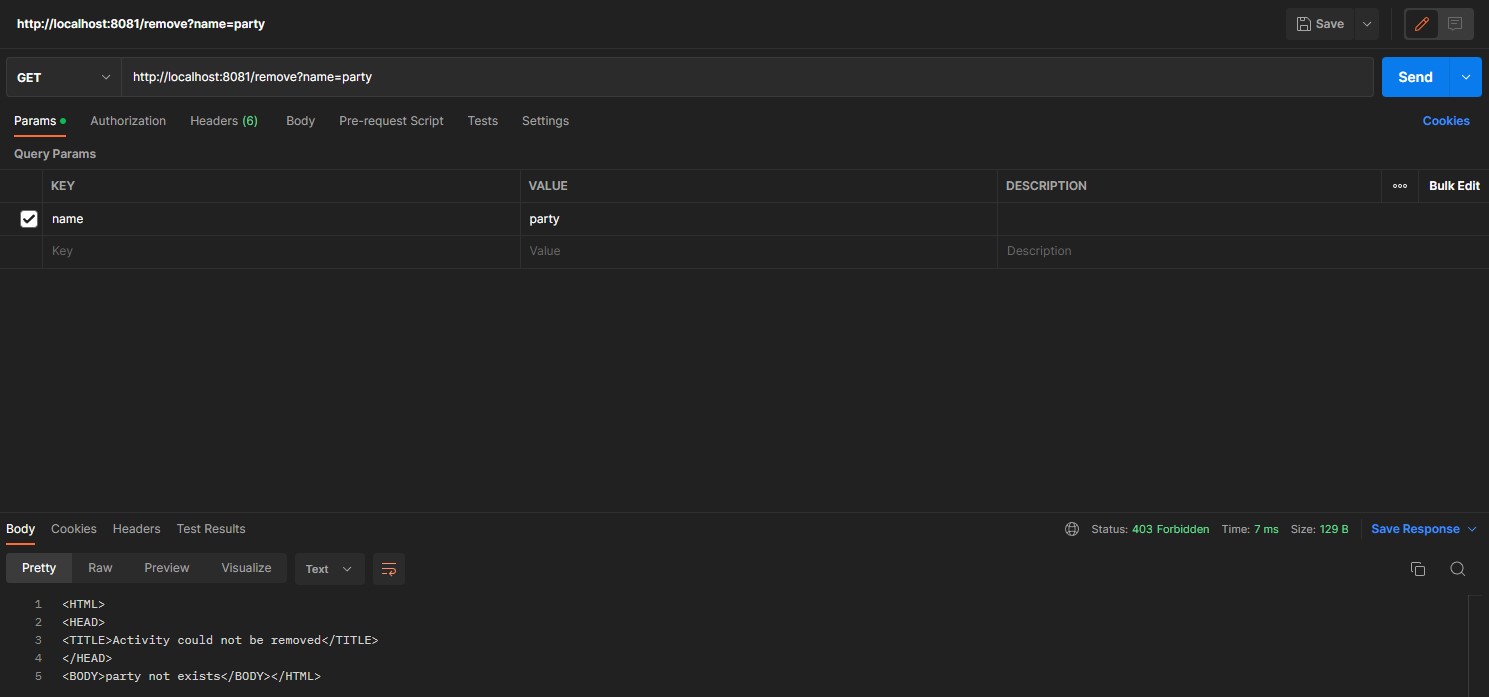
1. If the activity is already added, this server should return an HTTP 403 Forbidden message indicating that the activity already exists in the database



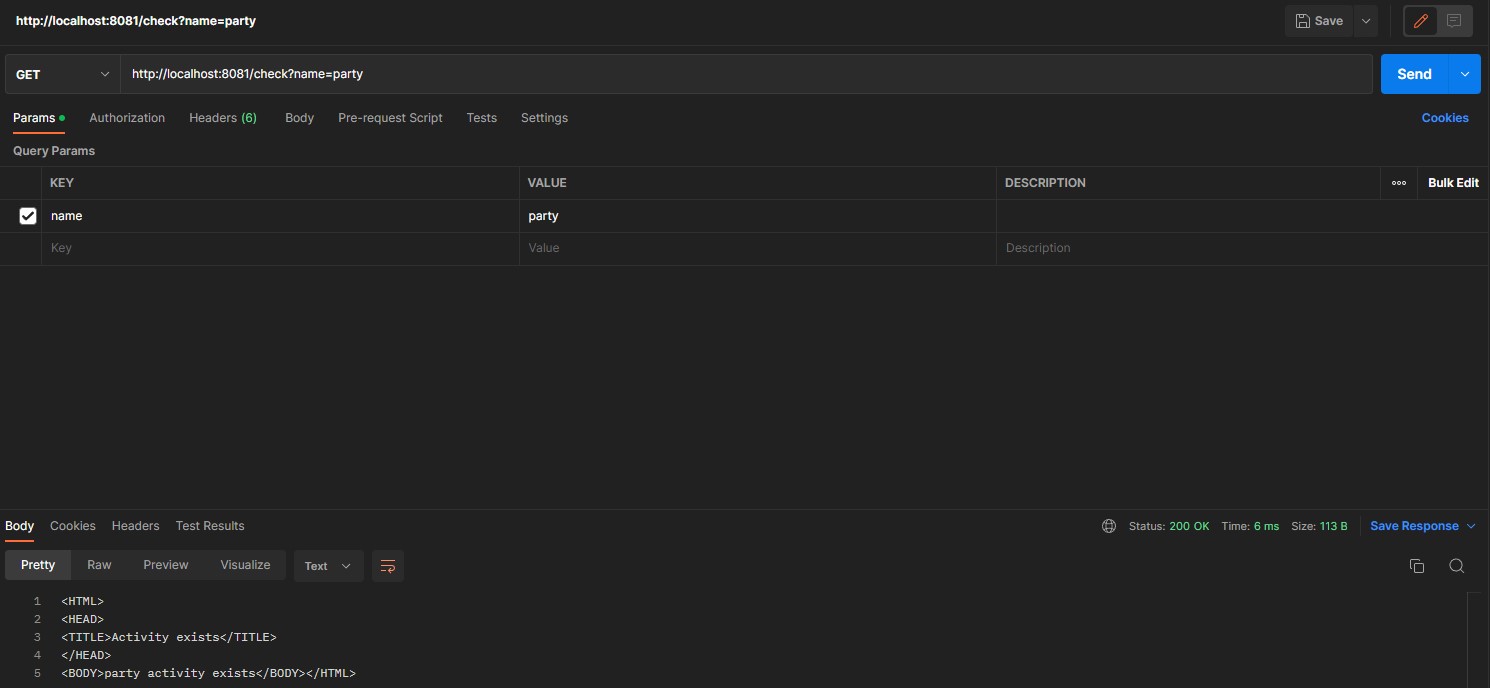
1. If the activity exists, it removes the activity from the database and sends back HTTP 200 OK message



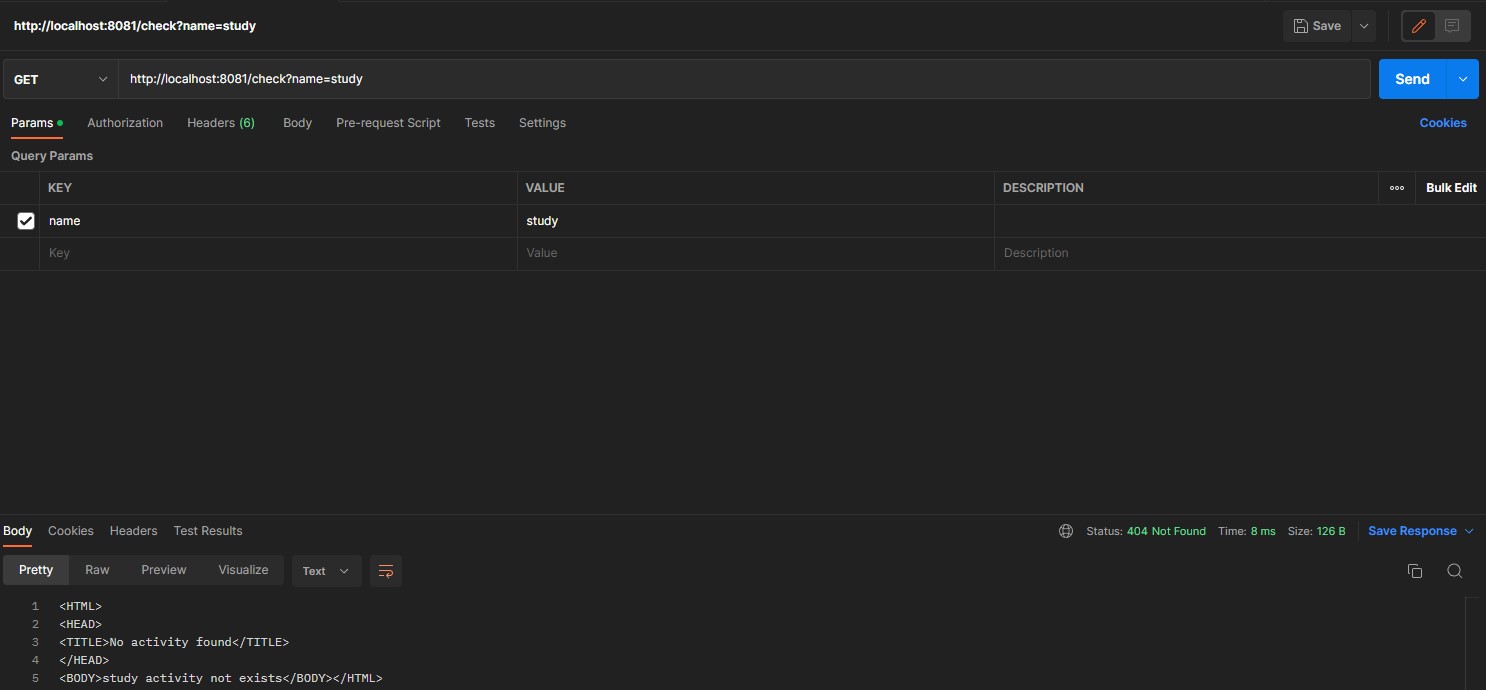
1. If the activity doesn’t exist, then it sends back an HTTP 403 Forbidden message.



1. If the activity exists, it sends back an HTTP 200 OK message

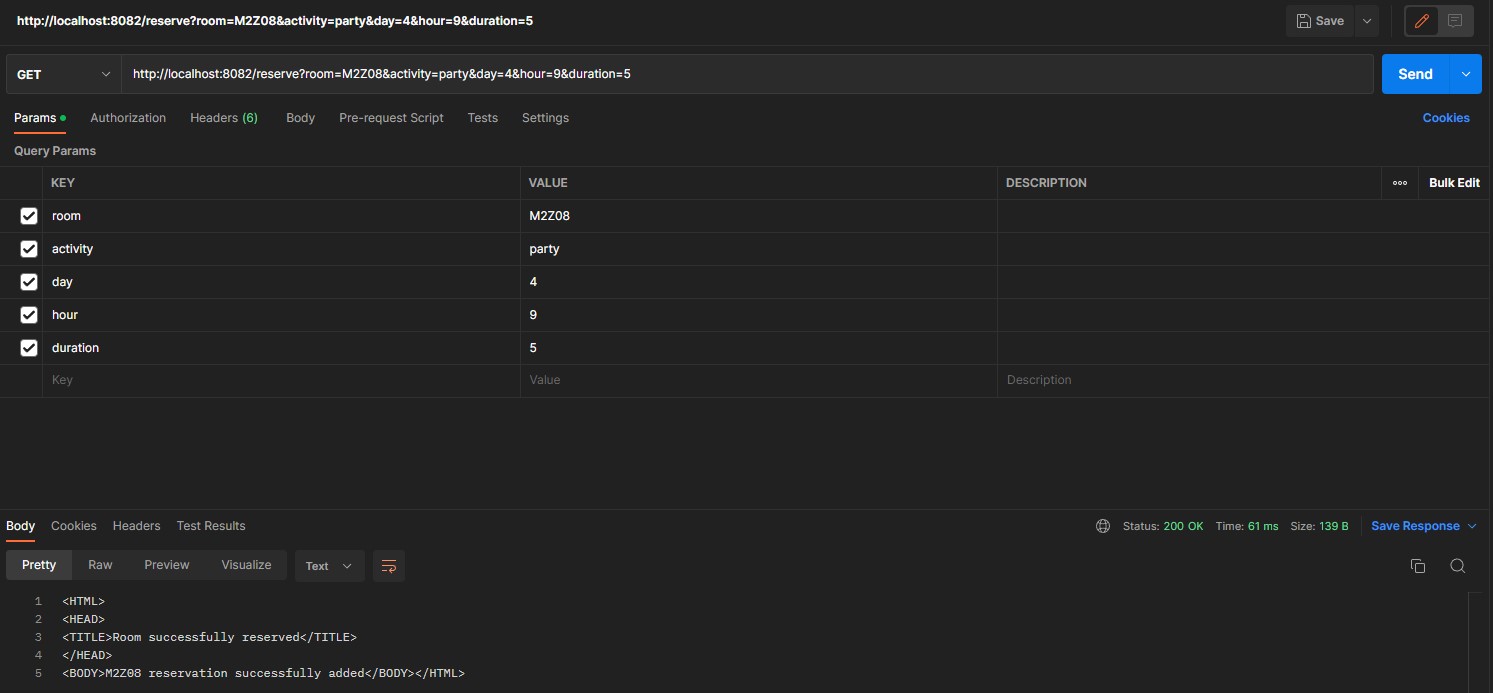


1. If activity does not exist, it sends HTTP 404 Not Found message.

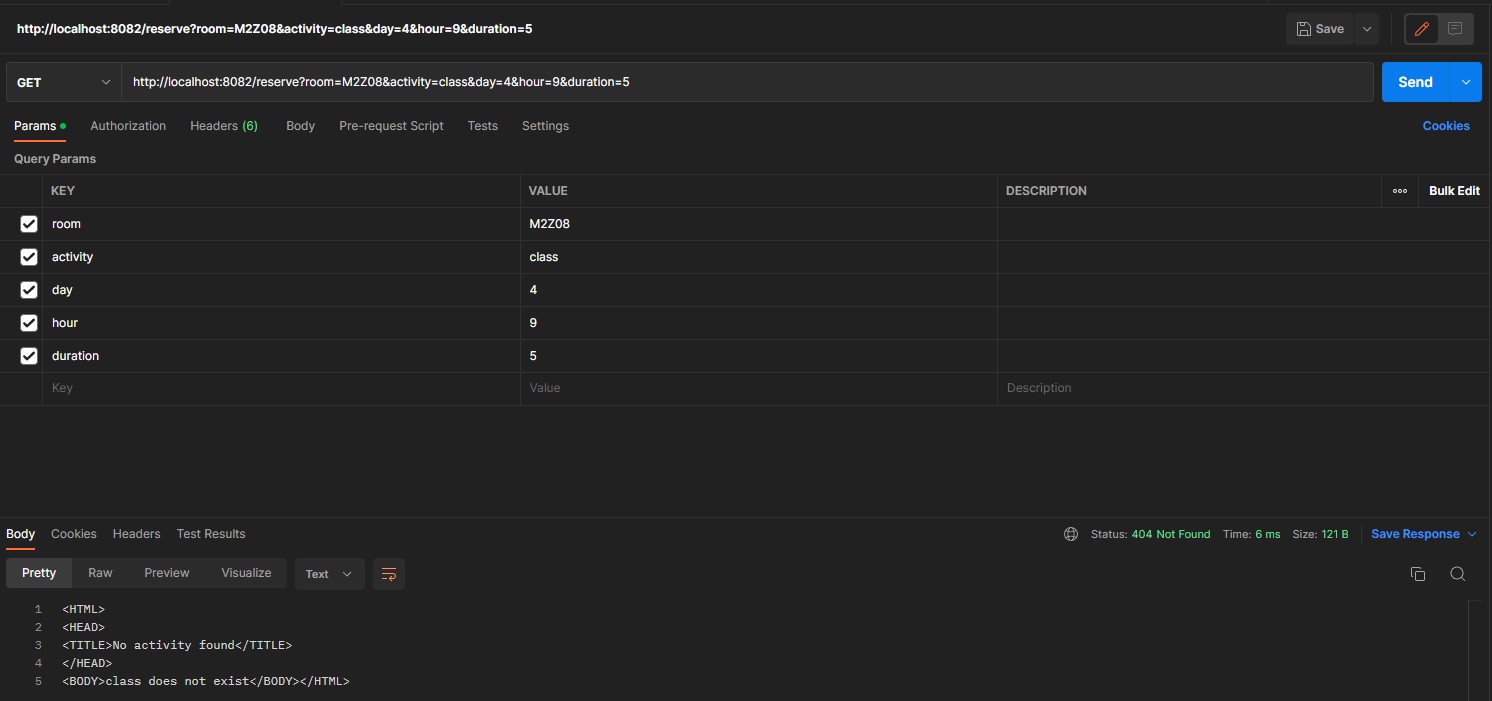


# Reservation Server

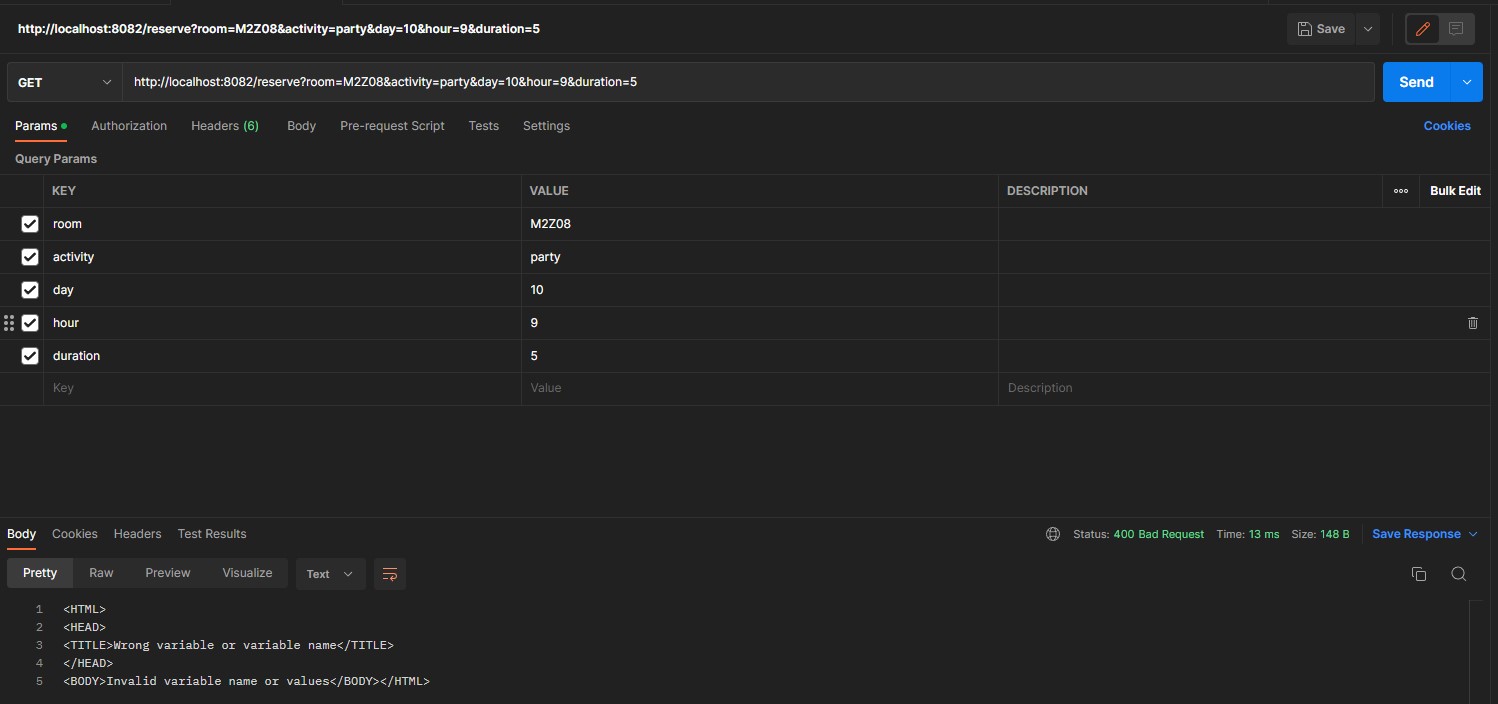
1. All the inputs are valid, then it either reserves the room and sends back an HTTP 200 OK message



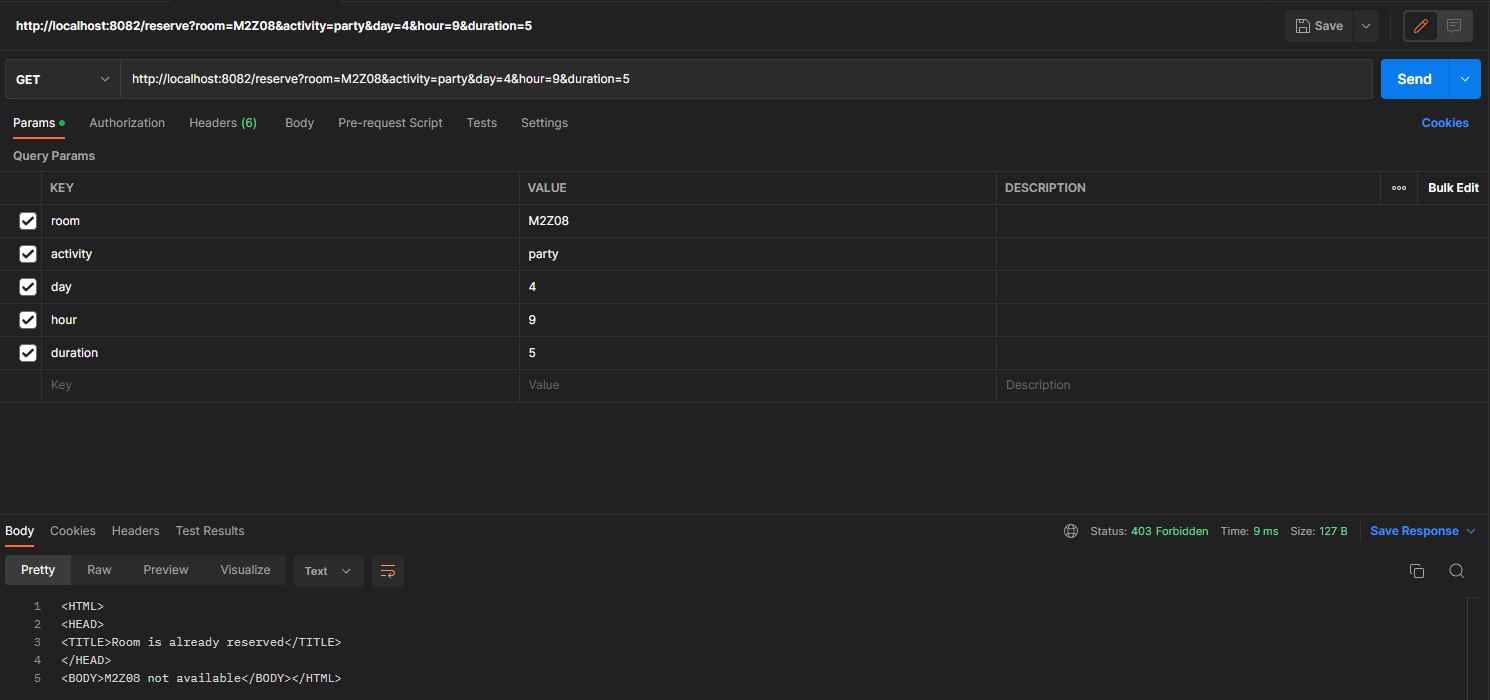
1. If no such activity exists it sends back an HTTP 404 Not Found message.



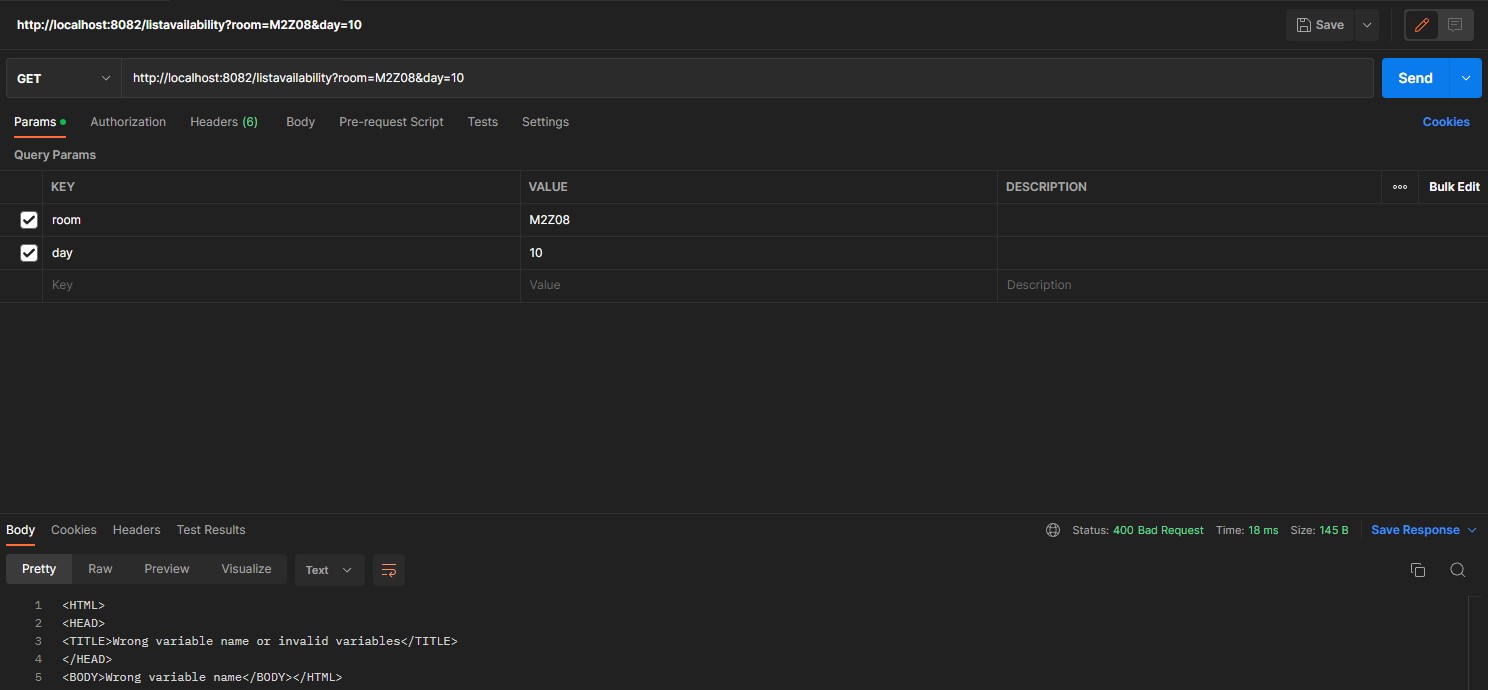
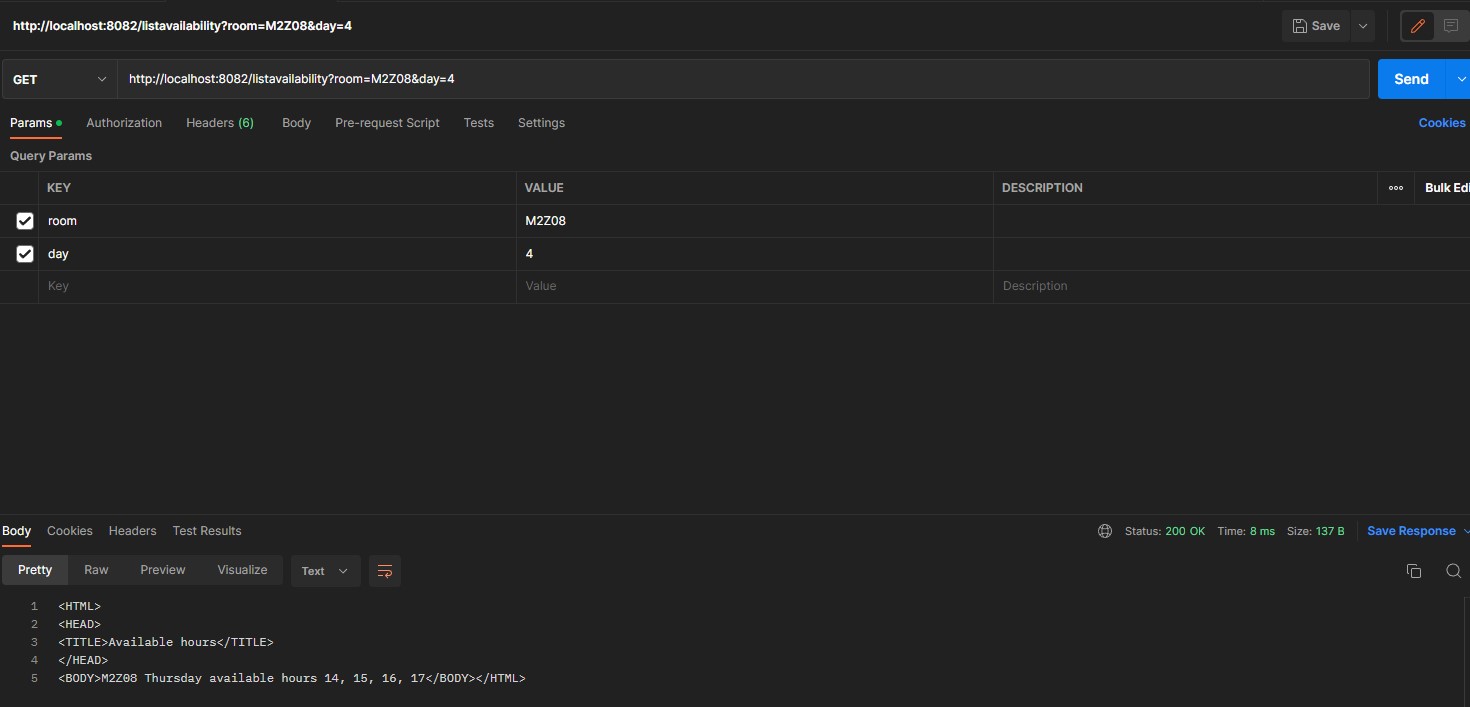
1. If any of the input is invalid, it sends back an HTTP 400 Bad Request message.



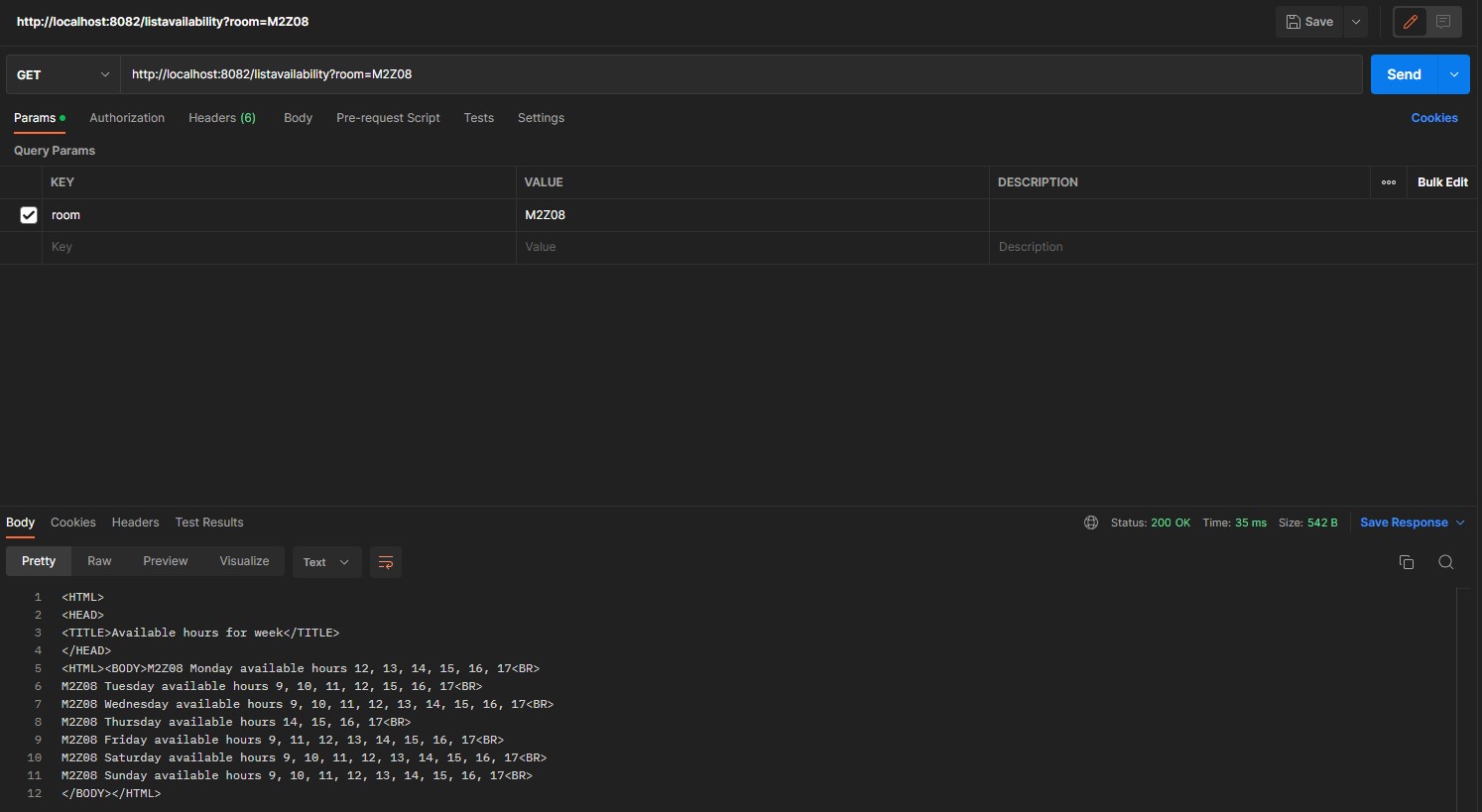
1. If we try to make a reserved day and time



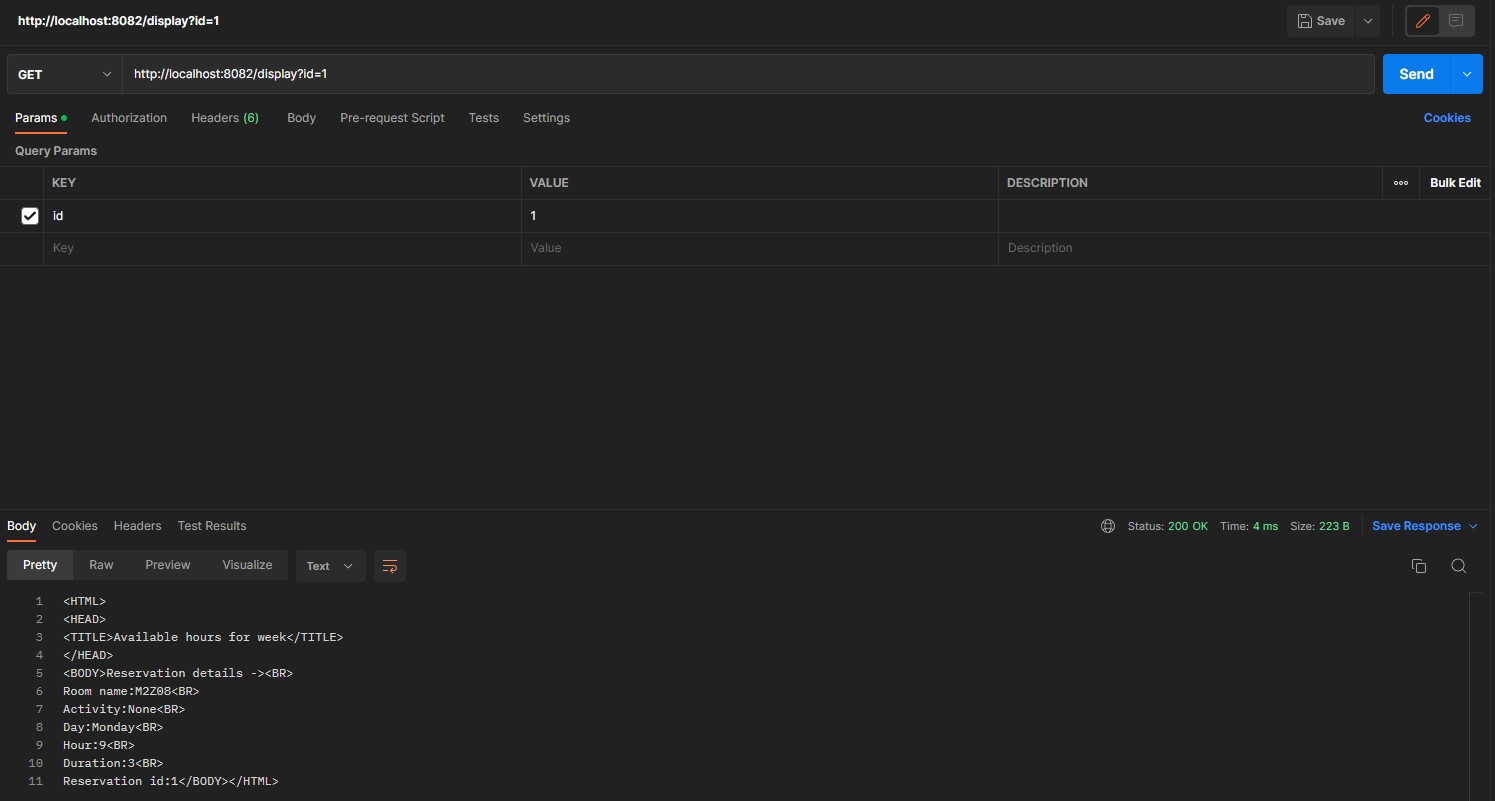
1. Lists all the available hours for the specified day (after contacting the Room Server).

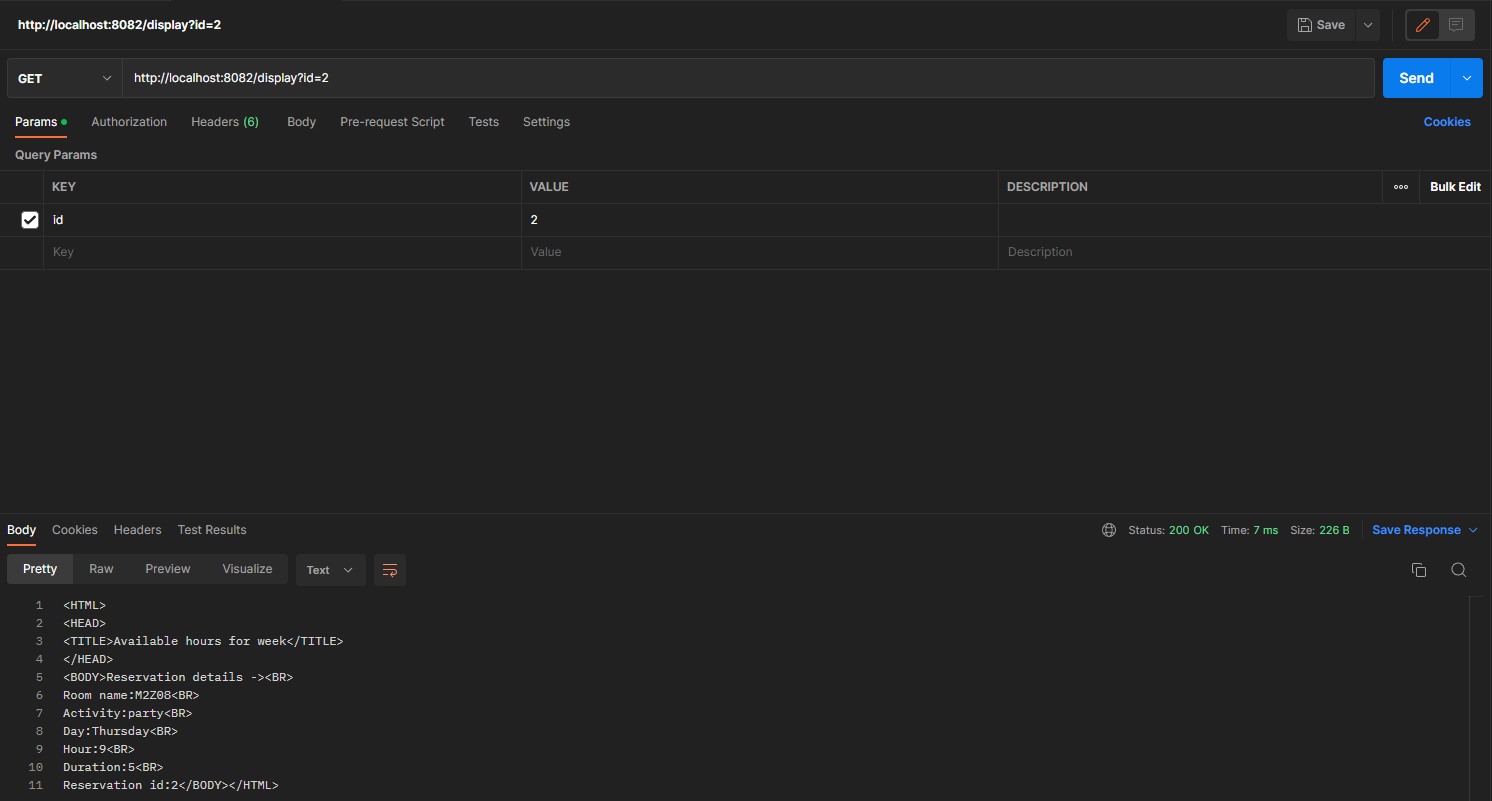


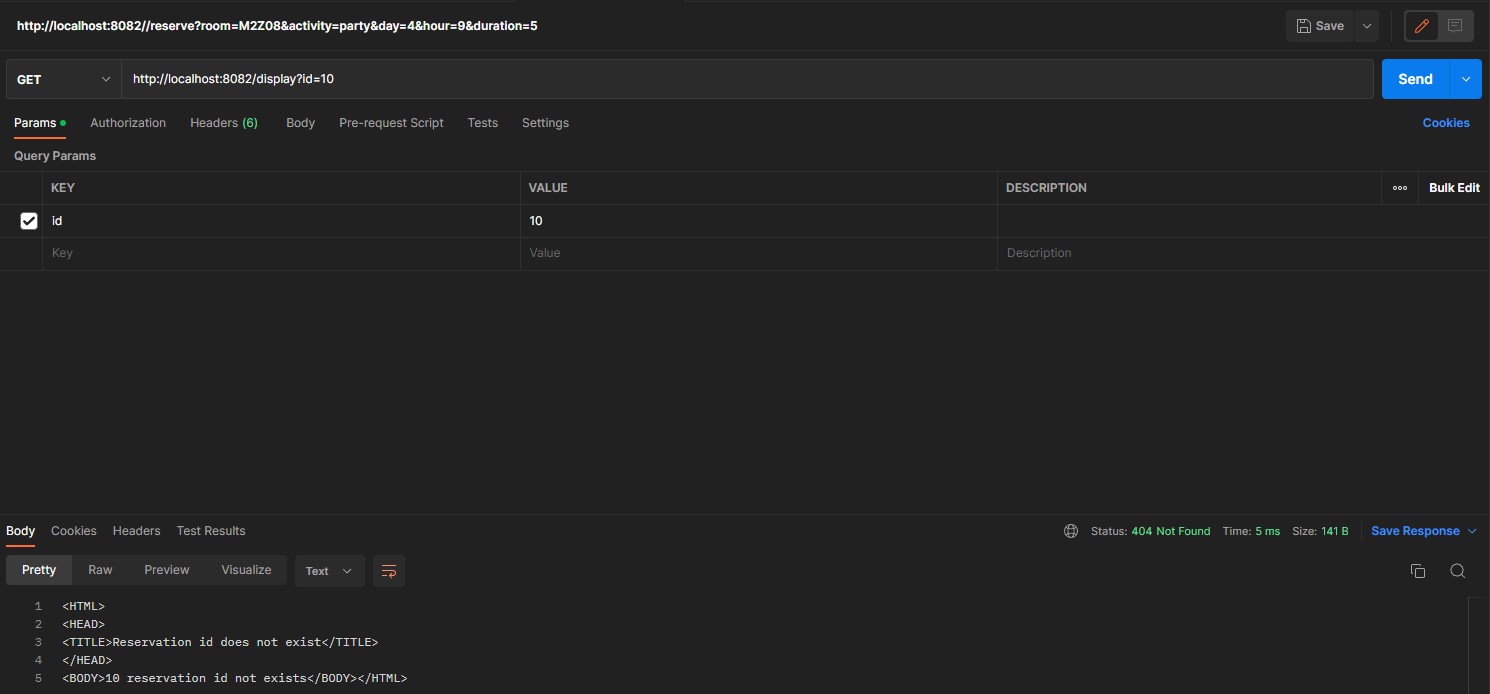
1. Lists all the available hours for all days of the week (after contacting the Room Server probably several times).



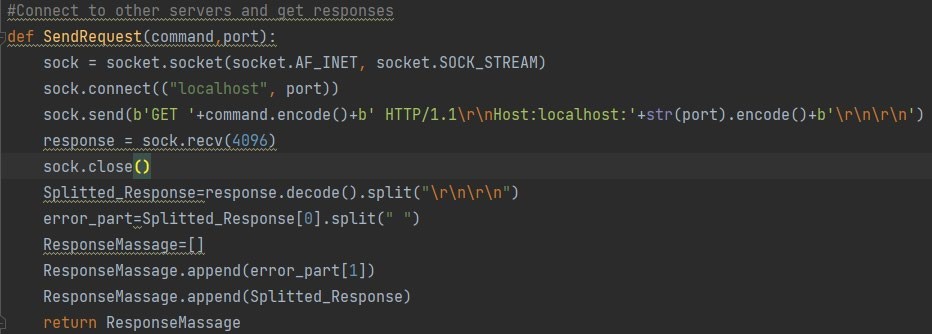
1. Returns the details of the reservation with the specified id. If the reservation\_id does not exist, it returns back an HTTP 404 Not Found message.



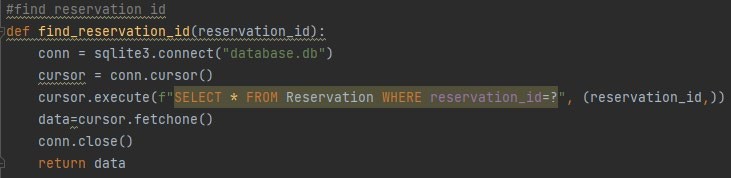


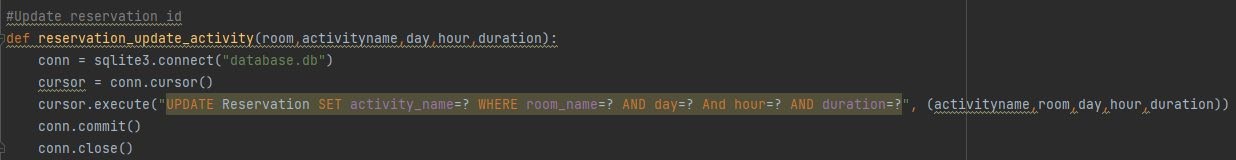


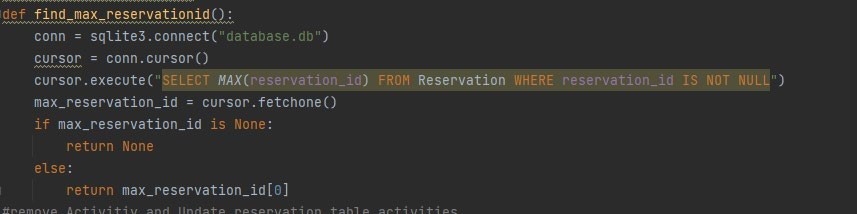
# Functions

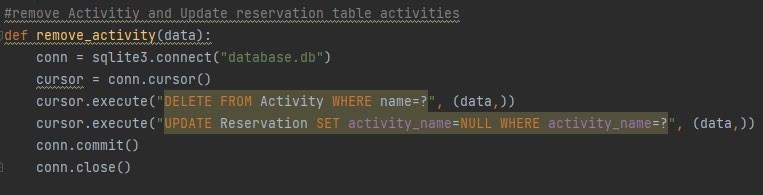


***SendRequest (command, port):*** The SendRequest function takes the port number and the action on the relevant port as a parameter. It connects to the relevant port and sends the command. It splits the status code of the incoming response and the returned response. It returns it as a list***.***

 ***find\_reservation\_id(reservation\_id):*** This function connects to the database. It takes the reservation id value entered as a parameter in the function and finds it in the Reservation table in the database. With the fetchone() method, we get the data we selected and equate it to the data variable. We return the data variable whether it is none or not.

 ***reservation\_update\_activity(room, activityname, day, hour, duration):*** This function connects to the database. In the reservation table in the database, where the room\_name and day and hour and duration values are equal to the room, day, hour, duration values entered as parameters, they are set with the activity\_name in the activityname parameter.

 ***find\_max\_reservationid():*** Returns the largest reservation \_id value in rows where the reservation id value is not NULL in the Reservation table. In here, We return the first element of the max\_reservation\_id list, that is, its largest value. But if this list is None, we return None.



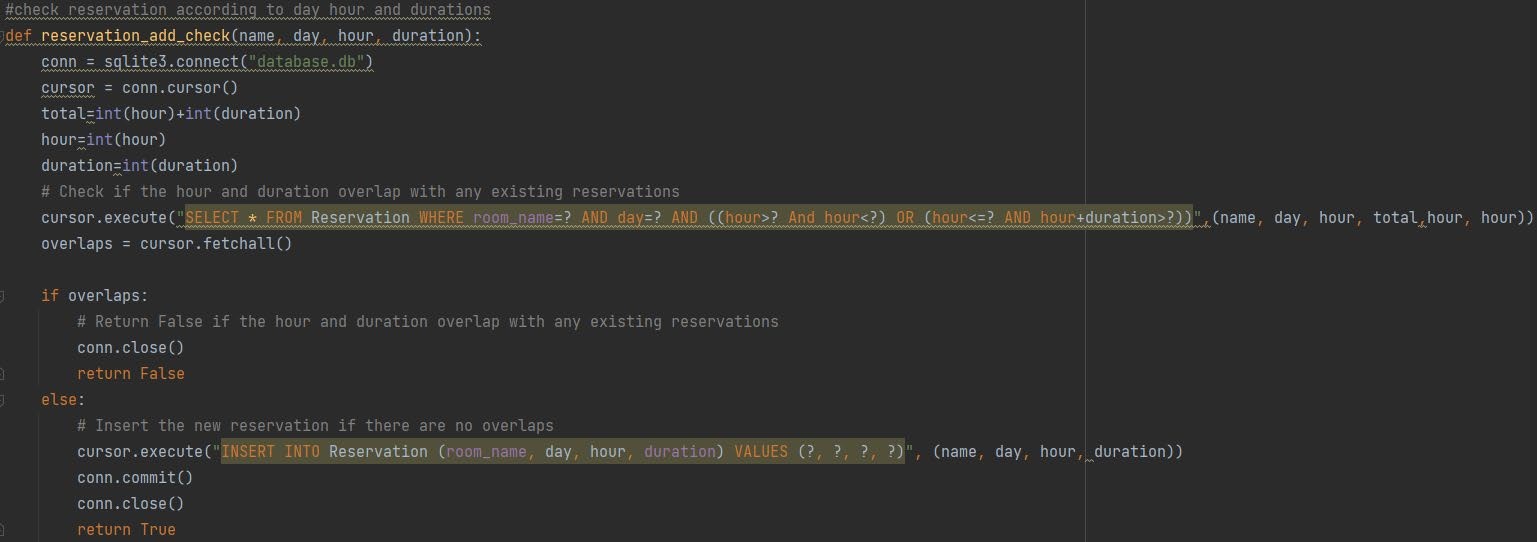
***remove\_activity(data):*** This function deletes the row in the Activity table where the name value is equal to the data value entered as a parameter. After this process, we update the activity\_name value to NULL in the Reservation table where the activity\_name value is equal to the data value entered as a parameter.

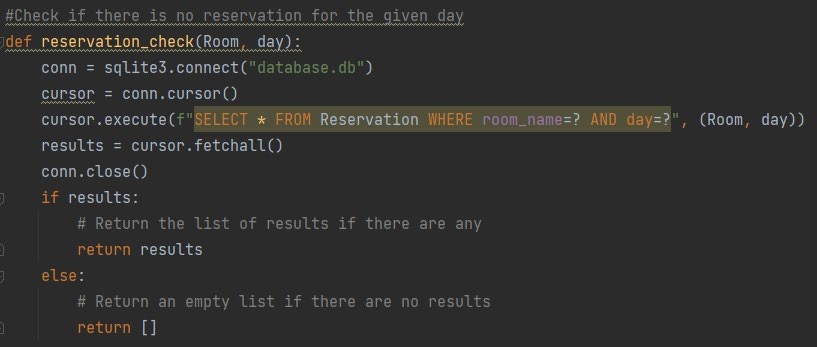
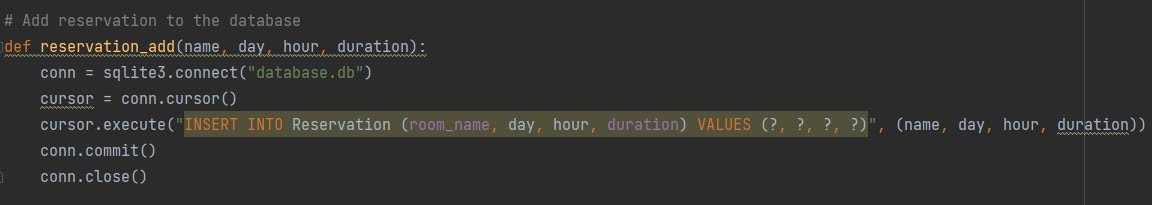


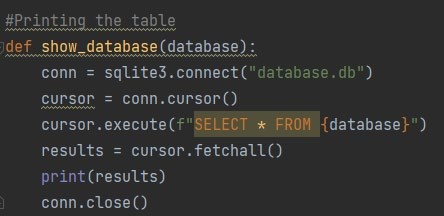
***Add\_Activity(data):*** Adds a new row that takes an activity value to the Activity table.



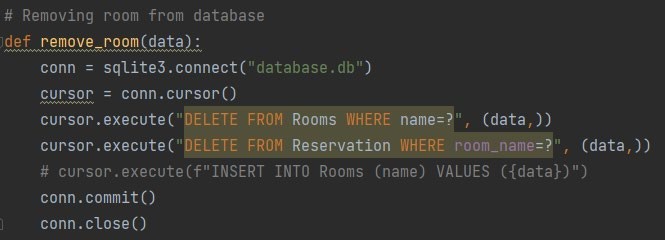
***CheckAvailability(roomname,day):*** This function first increments the values between the startHour and endhour variables (including startHour and endhour) by 1 and adds all the resulting values to the allHours list. In short, it records all hours between 9 and 17 in the allHours list. Returns the data in the Reservation table where the room\_name and day values are equal to the parameter of the function. We equate these returned values to the ‘results’ variable. In the next step we loop over each element of the results list and add the unavailable hours to the not\_available list. In the next step, we reach the available hours by subtracting the hours from being unavailable from all hours. And we add these hours to the avaiable\_hours list.

 ***reservation\_add\_check(name, day, hour, duration):*** In this function, we check whether the time and duration overlap with the existing reservations. If there is a hour and duration overlap, it returns False. Otherwise, we insert a new reservation with no time overlaps an it returns True.

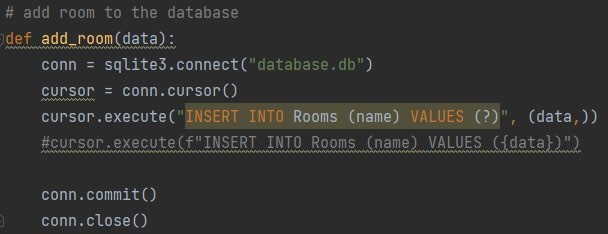
 ***reservation\_check(Room, day):*** We equate it to the results list by bringing the values of the Room value given as the parameter to the day value given as the parameter. If this list has elements, we return the list itself. But if it doesn't have a value, we return an empty list.  ***reservation\_add(name, day, hour, duration):*** A new reservation record is created according to the values given in the parameters of the function.



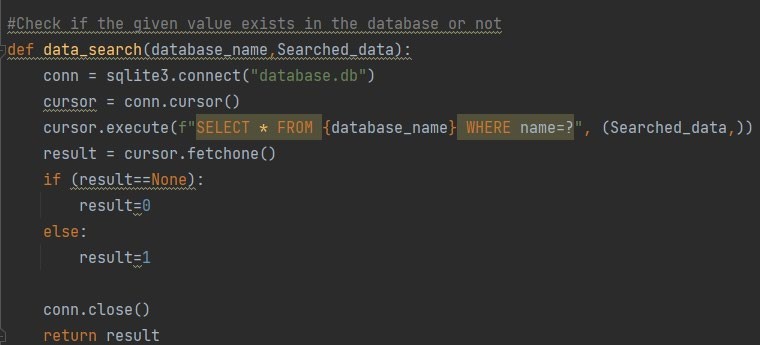
***show\_database(database):*** This function sends all the data in the database to the variable named results in list type. This list of results is then printed.



***remove\_room(data):*** This function first remove the room row whose name is given in the Rooms table. Afterwards, the same action also remove the data of this room in the Reservation table.



***add\_room(data):***This function is created to add a new room to the database.



***data\_search(database\_name, Searched\_data):*** This function checks whether the parameter entered in it is in the database or not.