IIIT Guest House Management System

By CSE- 2nd Year
B219015 Atrik Ray
B219024 Gayathri MS
B219032 Manas Sahu

The project intends to provide a solution to implement the IIIT Guest House Management System, using a database and web pages to interact with the database.

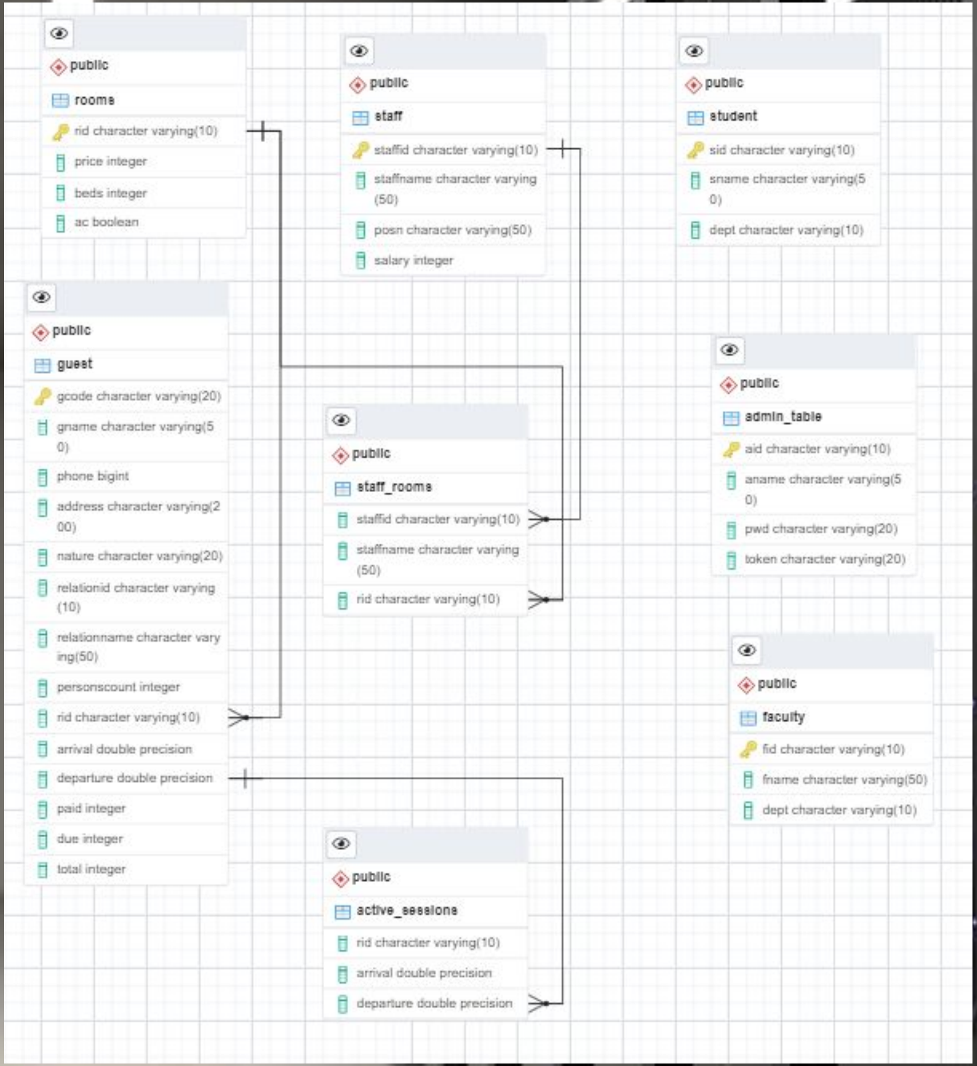
DATABASE

Tools Used

PostgreSQL 10.15

pgAdmin 4.30 (GUI)

SCHEMA DIAGRAM



public
rooms
rid character varying(10)
price integer
beds integer
ac boolean

public
staff
staffid character varying(10)
staffname character varying (50)
posn character varying(50)
salary integer

public
student
sid character varying(10)
sname character varying(5 0)
dept character varying(10)

public
guest
gcode character varying(20)
gname character varying(5 0)
phone bigint
address character varying(2 00)
nature character varying(20)
relationid character varying (10)
relationname character vary ing(50)
personscount integer
rid character varying(10)
arrival double precision
departure double precision
paid integer
due integer
total integer

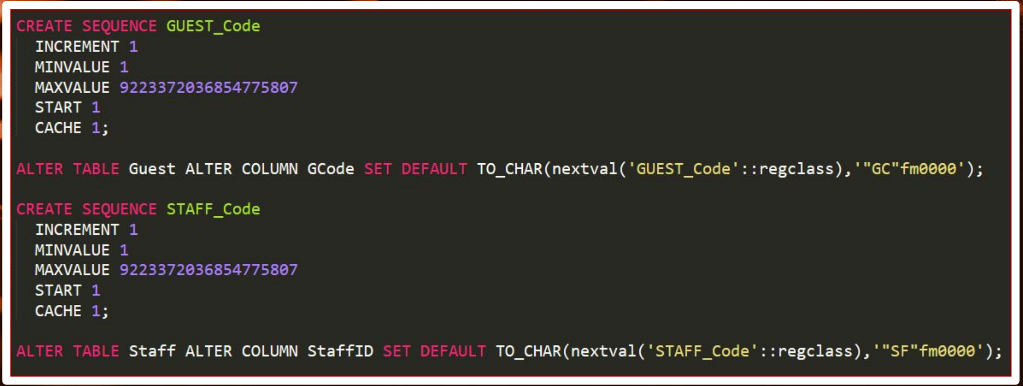
public
staff_rooms
staffid character varying(10)
staffname character varying (50)
rid character varying(10)

public
admin_table
aid character varying(10)
aname character varying(5 0)
pwd character varying(20)
token character varying(20)

public
faculty
fid character varying(10)
fname character varying(50)
dept character varying(10)

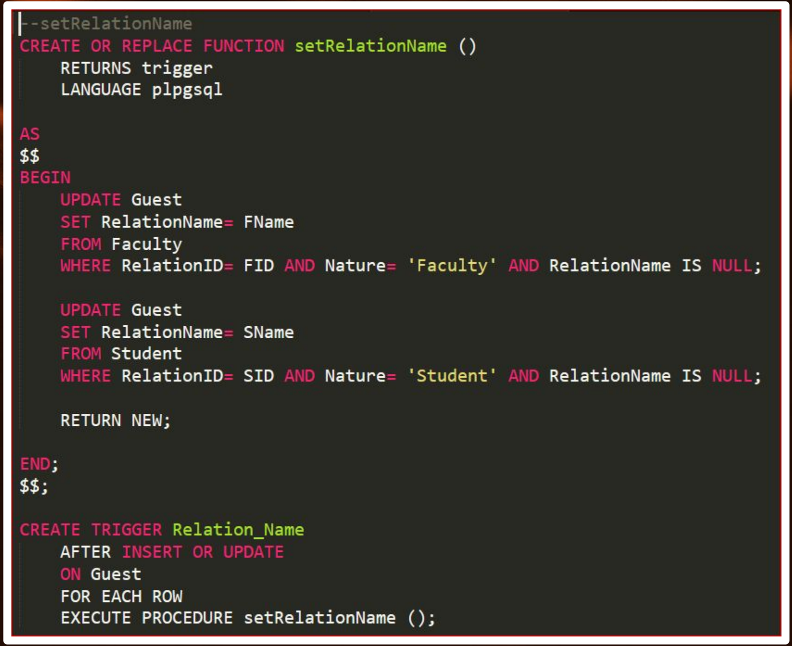
public
active_sessions
rid character varying(10)
arrival double precision
departure double precision

- ❖ A sequence is a user defined schema bound object that generates a sequence of numeric values.
- ❖ Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.



```
CREATE SEQUENCE GUEST_Code
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

ALTER TABLE Guest ALTER COLUMN GCode SET DEFAULT TO_CHAR(nextval('GUEST_Code'::regclass),'"GC"fm0000');

CREATE SEQUENCE STAFF_Code
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

ALTER TABLE Staff ALTER COLUMN StaffID SET DEFAULT TO_CHAR(nextval('STAFF_Code'::regclass),'"SF"fm0000');
```

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.



```
--setRelationName
CREATE OR REPLACE FUNCTION setRelationName ()
    RETURNS trigger
    LANGUAGE plpgsql

AS
$$
BEGIN
    UPDATE Guest
    SET RelationName= FName
    FROM Faculty
    WHERE RelationID= FID AND Nature= 'Faculty' AND RelationName IS NULL;

    UPDATE Guest
    SET RelationName= SName
    FROM Student
    WHERE RelationID= SID AND Nature= 'Student' AND RelationName IS NULL;

    RETURN NEW;

END;
$$;

CREATE TRIGGER Relation_Name
    AFTER INSERT OR UPDATE
    ON Guest
    FOR EACH ROW
    EXECUTE PROCEDURE setRelationName ();
```

FRONT-END

Tools Used

HTML5

Describing the structure of the information in a webpage

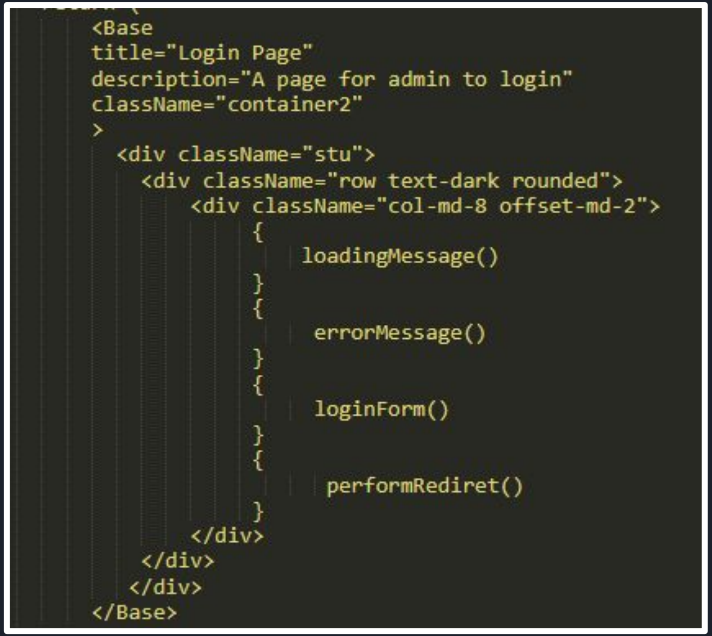
CSS

Deals with elements color, font, layout, etc

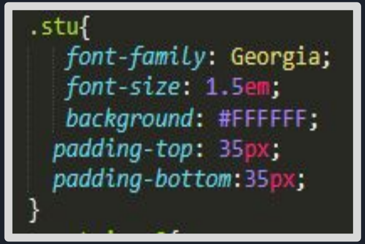
REACT.JS

Allows developers to create large web applications which changes data without reloading

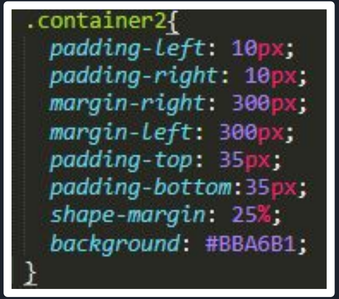
Form and its corresponding HTML and CSS codes



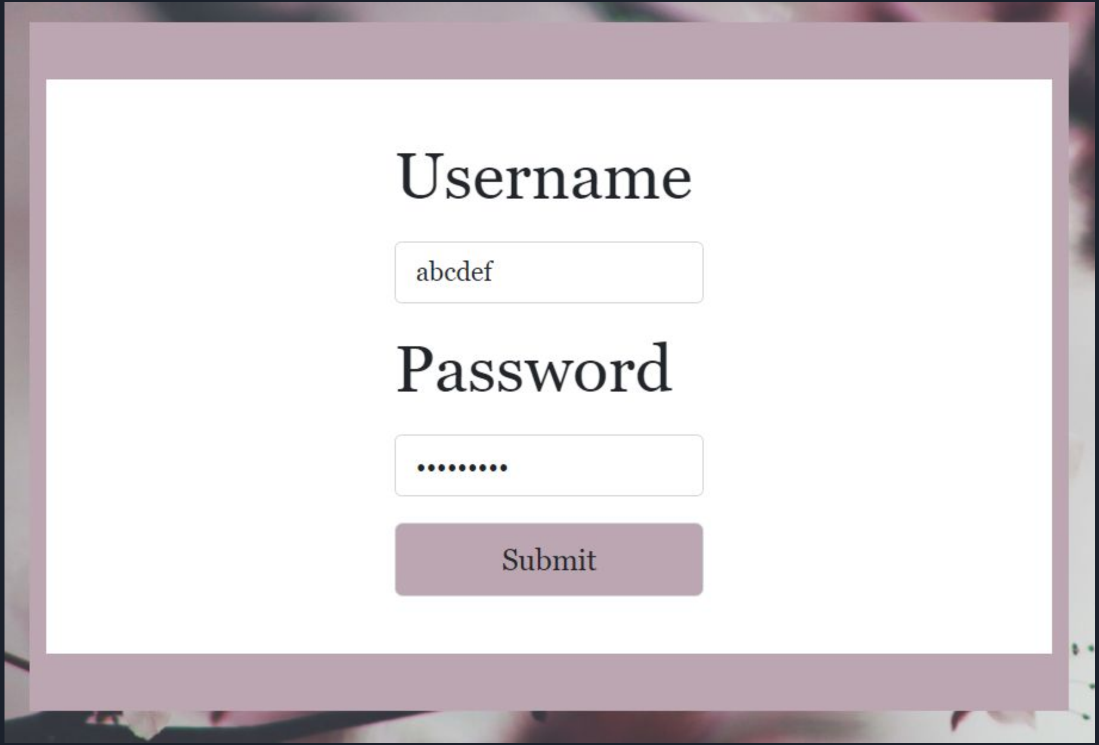
```
        <Base
        title="Login Page"
        description="A page for admin to login"
        className="container2"
        >
          <div className="stu">
            <div className="row text-dark rounded">
                <div className="col-md-8 offset-md-2">
                     {
                         loadingMessage()
                     }
                     {
                          errorMessage()
                     }
                     {
                          loginForm()
                     }
                     {
                           performRediret()
                     }
                </div>
            </div>
           </div>
        </Base>
```



```
.stu{
    font-family: Georgia;
    font-size: 1.5em;
    background: #FFFFFF;
   padding-top: 35px;
   padding-bottom:35px;
}
```



```
.container2{
    padding-left: 10px;
    padding-right: 10px;
    margin-right: 300px;
    margin-left: 300px;
    padding-top: 35px;
    padding-bottom:35px;
    shape-margin: 25%;
    background: #BBA6B1;
}
```



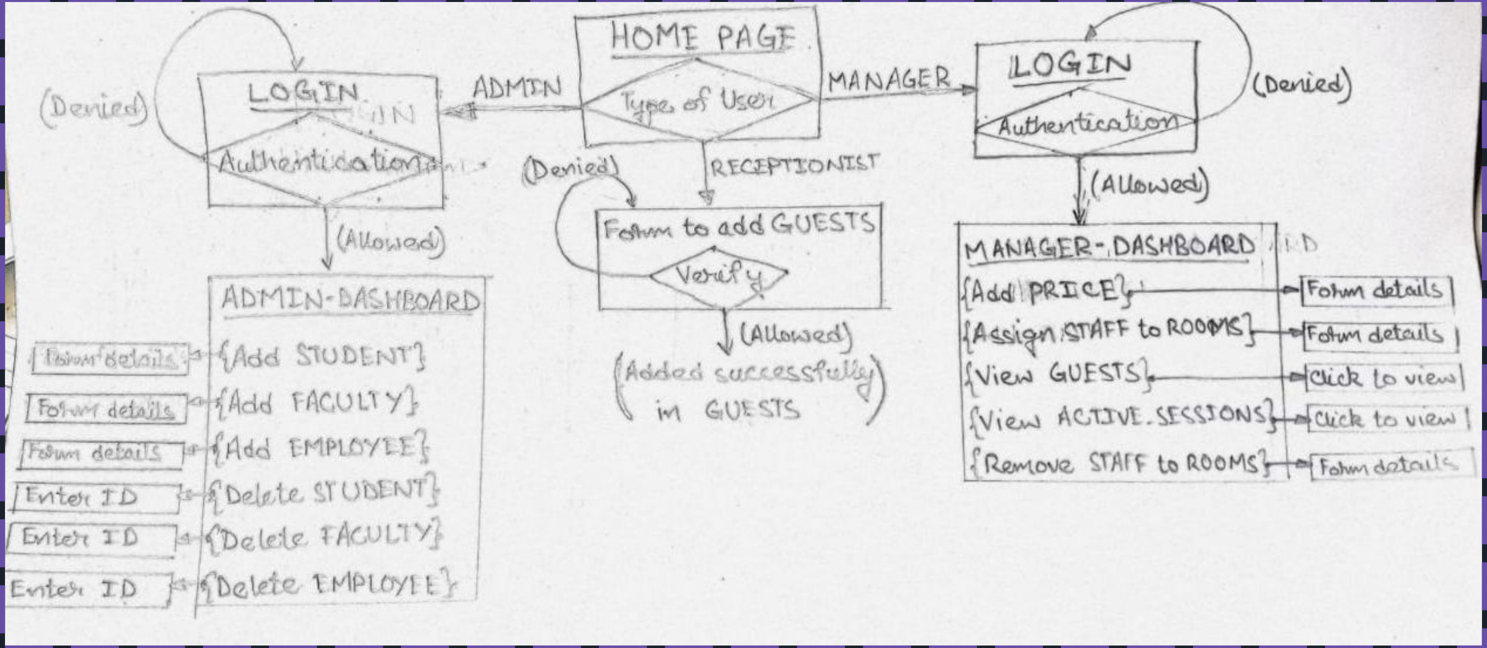
Username

abcdef

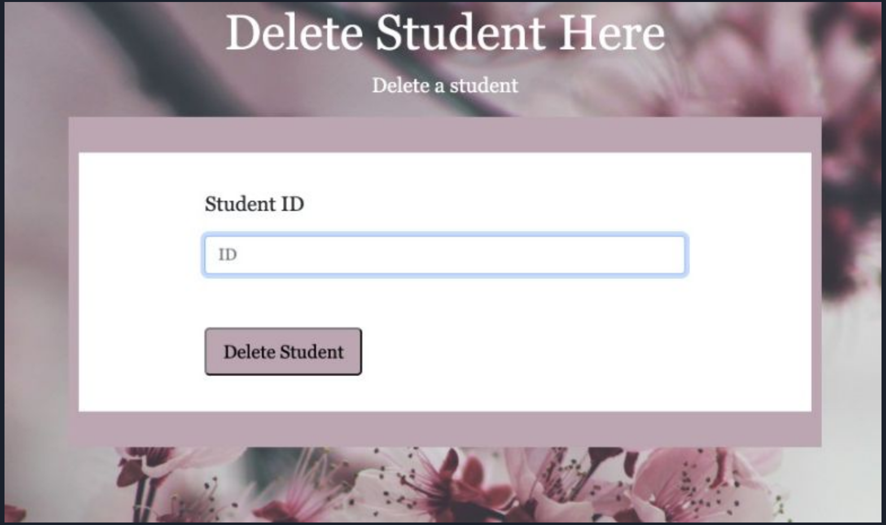
Password

•••••••••

SITE MAPS & ACTIONS



SAMPLE IMAGES

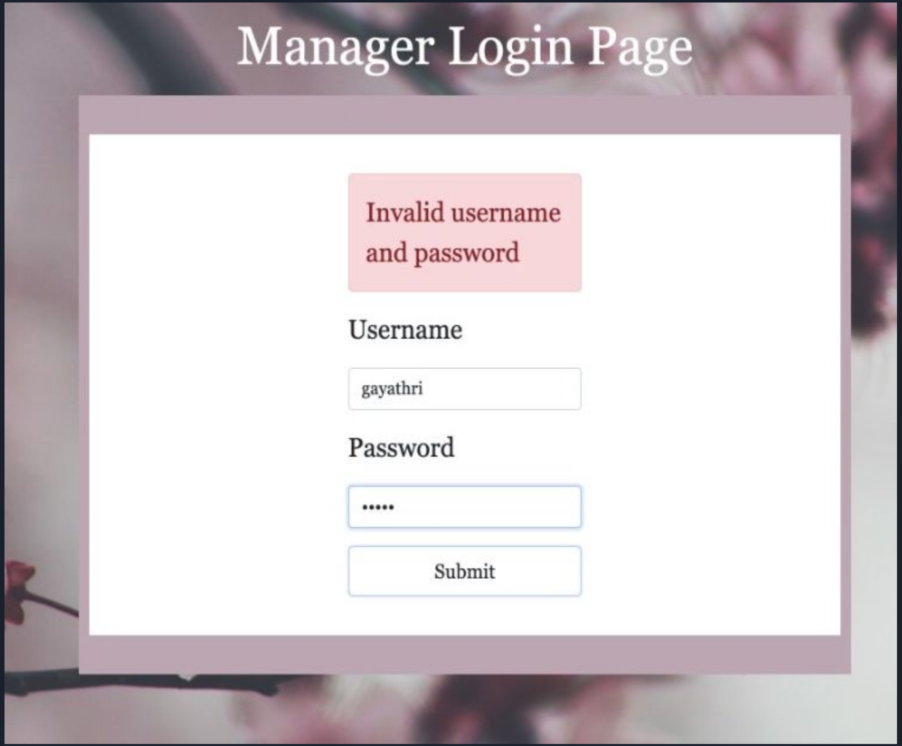


Delete Student Here

Delete a student

Student ID

ID



Manager Login Page

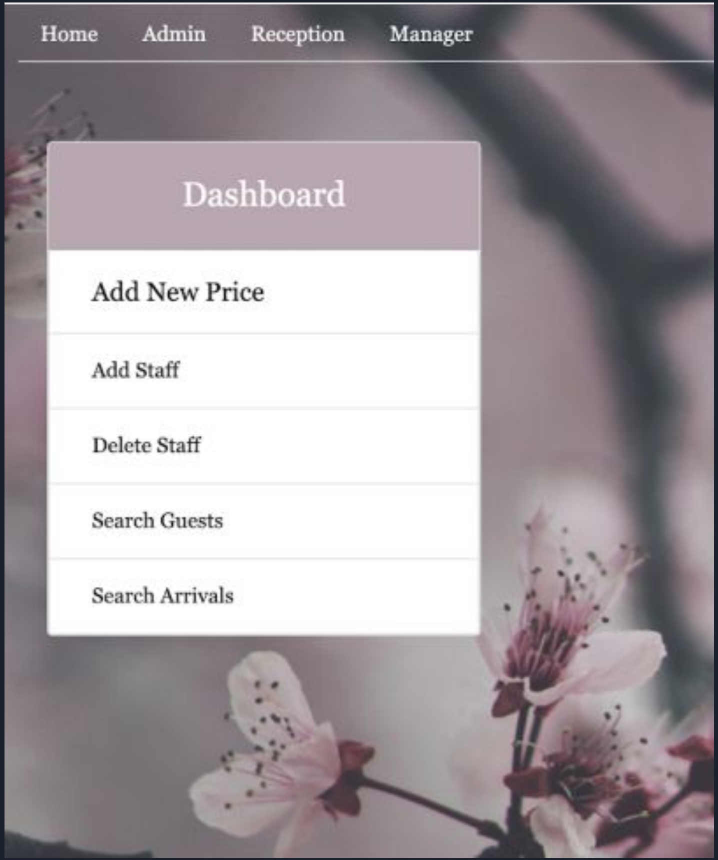
Invalid username and password

Username

gayathri

Password

•••••


Home Admin Reception Manager

Dashboard

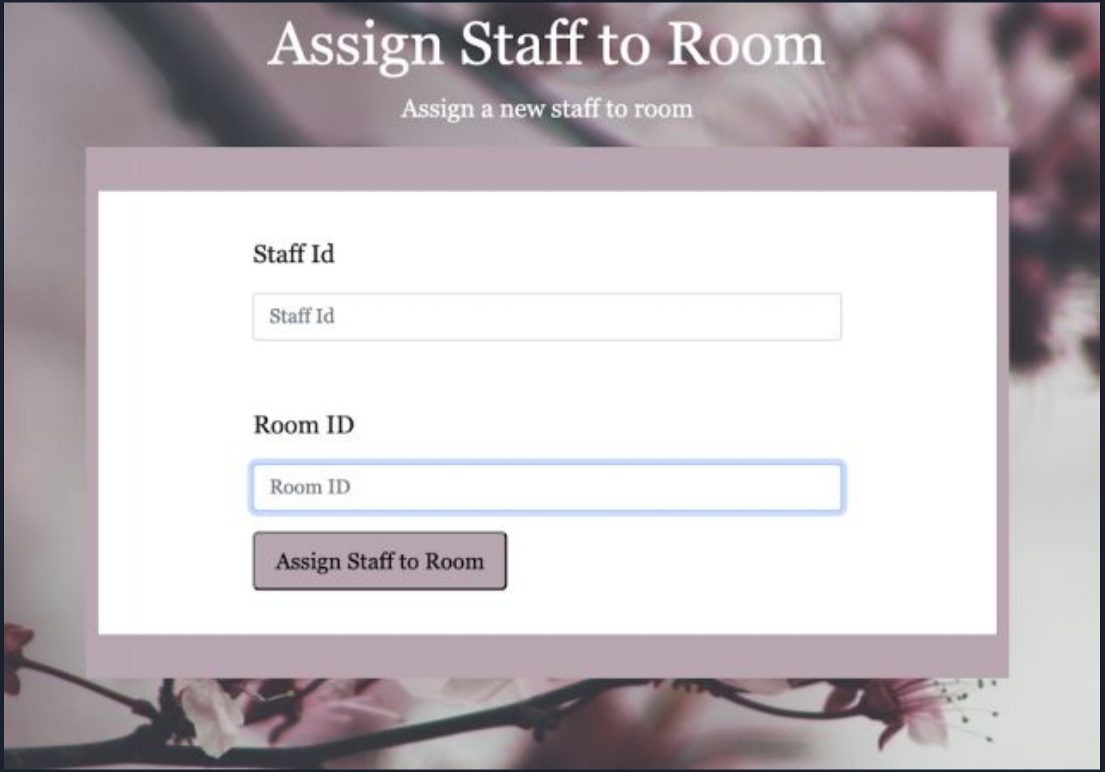
Add New Price

Add Staff

Delete Staff

Search Guests

Search Arrivals


Assign Staff to Room

Assign a new staff to room

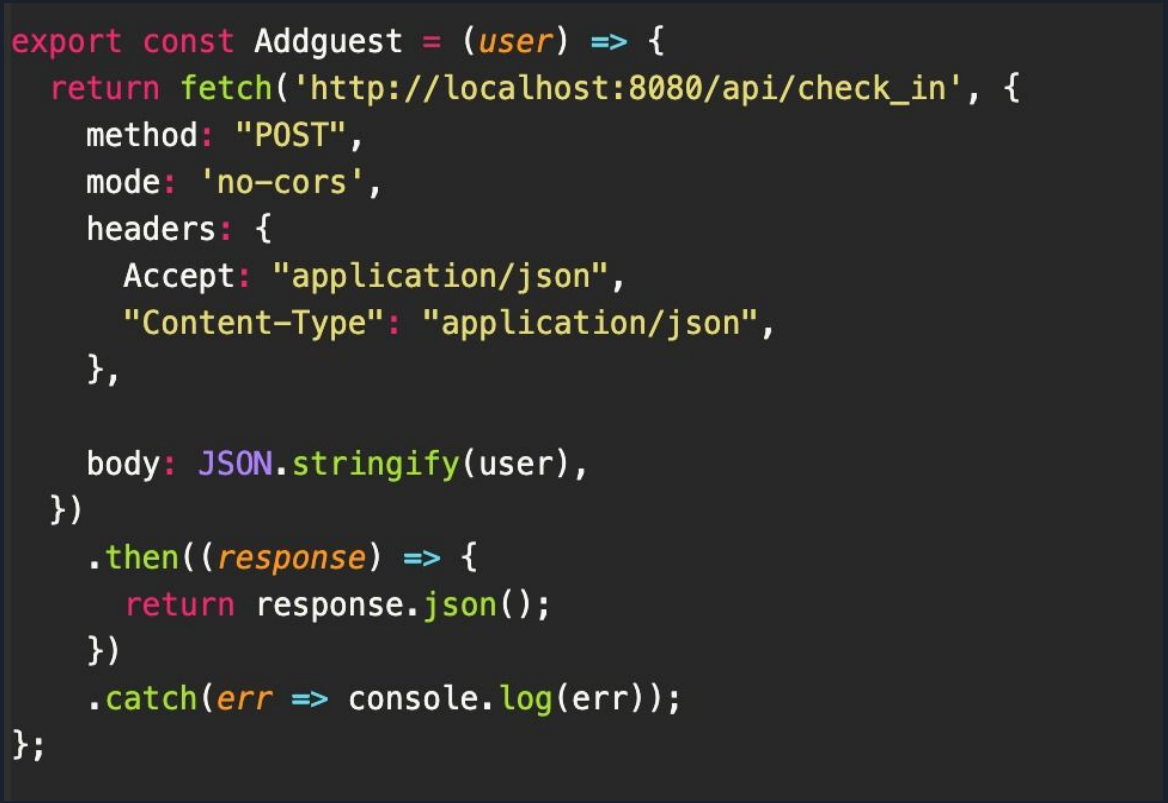
Staff Id

Staff Id

Room ID

Room ID

Assign Staff to Room



```
export const Addguest = (user) => {
  return fetch('http://localhost:8080/api/check_in', {
    method: "POST",
    mode: 'no-cors',
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },

    body: JSON.stringify(user),
  })
    .then((response) => {
      return response.json();
    })
    .catch(err => console.log(err));
};
```

API is being called from frontend to backend using POST request in JSON format

Backend APIs

How it works

A database interface class that builds on functions of psycopg2 and is called upon by various functions to access database

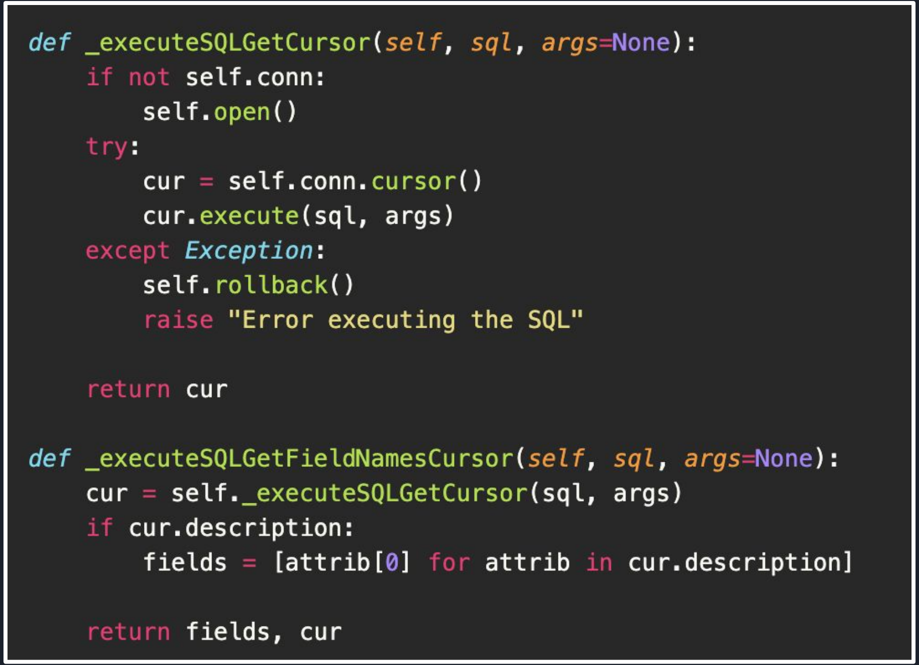
A server.py file that runs a WSGI server and serves requests

Python files containing related functions

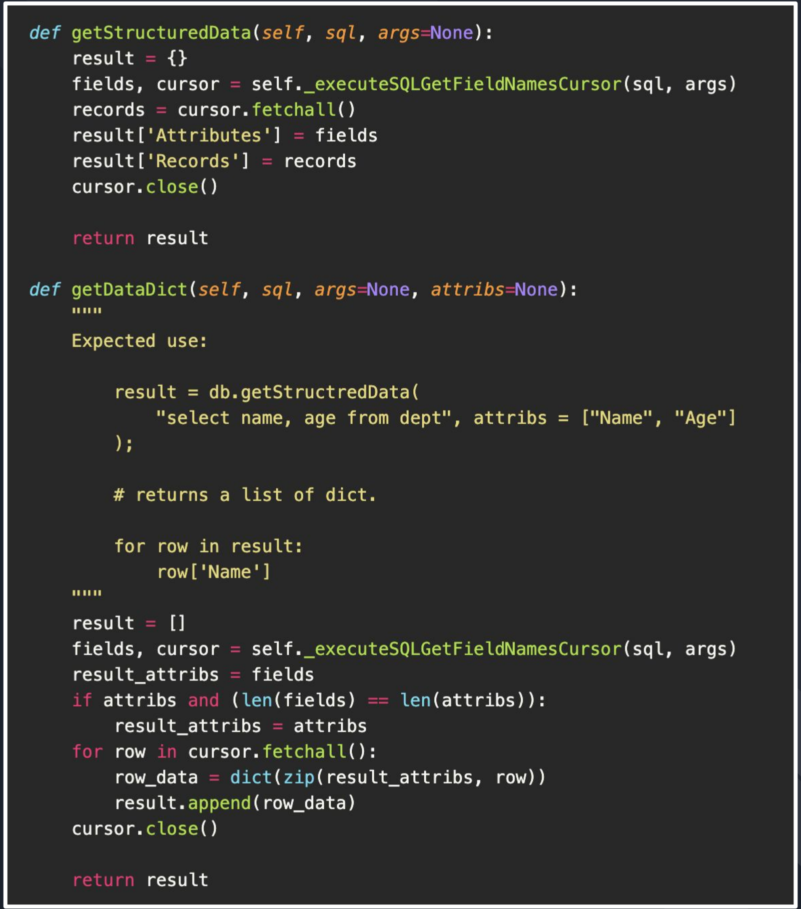
The Database Interface

1. Opens a connection to the database on instantiation and closes the connection when out of scope.
2. Common calls to the database don't need to be typed out everytime.
3. Helper functions in the database execute a SQL and return the cursor to the calling function adding the scope for more functionality
4. Changing databases just requires a few changes in the Database Class instead of all throughout the code.

Example



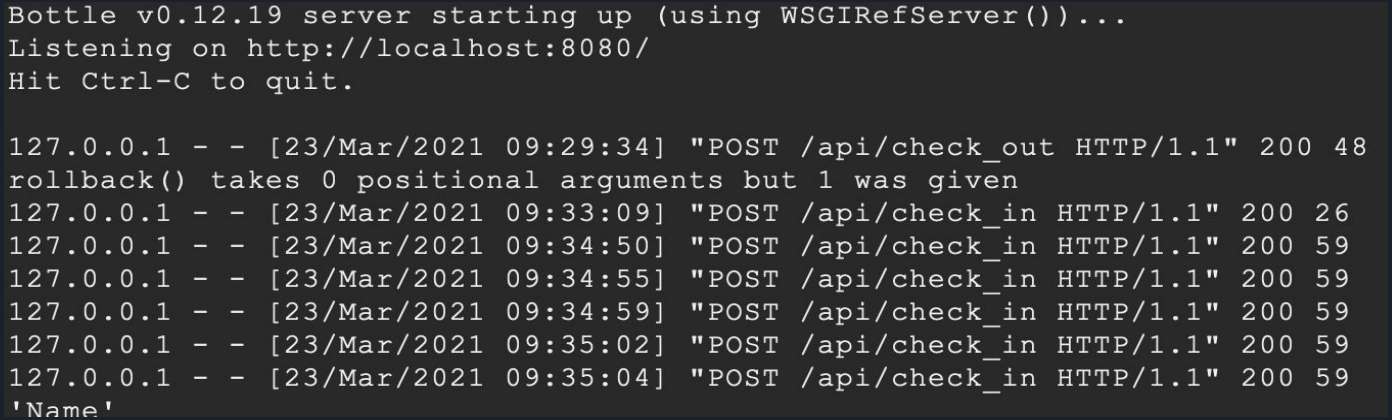
```python
def _executeSQLGetCursor(self, sql, args=None):
    if not self.conn:
        self.open()
    try:
        cur = self.conn.cursor()
        cur.execute(sql, args)
    except Exception:
        self.rollback()
        raise "Error executing the SQL"

    return cur

def _executeSQLGetFieldNamesCursor(self, sql, args=None):
    cur = self._executeSQLGetCursor(sql, args)
    if cur.description:
        fields = [attrib[0] for attrib in cur.description]

    return fields, cur
```



```python
def getStructuredData(self, sql, args=None):
    result = {}
    fields, cursor = self._executeSQLGetFieldNamesCursor(sql, args)
    records = cursor.fetchall()
    result['Attributes'] = fields
    result['Records'] = records
    cursor.close()

    return result

def getDataDict(self, sql, args=None, attribs=None):
    """
    Expected use:

        result = db.getStructredData(
            "select name, age from dept", attribs = ["Name", "Age"]
        );

        # returns a list of dict.

        for row in result:
            row['Name']
    """
    result = []
    fields, cursor = self._executeSQLGetFieldNamesCursor(sql, args)
    result_attribs = fields
    if attribs and (len(fields) == len(attribs)):
        result_attribs = attribs
    for row in cursor.fetchall():
        row_data = dict(zip(result_attribs, row))
        result.append(row_data)
    cursor.close()

    return result
```

Server.py

1. Links the API routes to the handling functions
2. Built on bottle framework. Uses inbuilt functions to parse request JSON and generate response JSON.
3. Listens on localhost:8080 for API requests



```
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

127.0.0.1 - - [23/Mar/2021 09:29:34] "POST /api/check_out HTTP/1.1" 200 48
rollback() takes 0 positional arguments but 1 was given
127.0.0.1 - - [23/Mar/2021 09:33:09] "POST /api/check_in HTTP/1.1" 200 26
127.0.0.1 - - [23/Mar/2021 09:34:50] "POST /api/check_in HTTP/1.1" 200 59
127.0.0.1 - - [23/Mar/2021 09:34:55] "POST /api/check_in HTTP/1.1" 200 59
127.0.0.1 - - [23/Mar/2021 09:34:59] "POST /api/check_in HTTP/1.1" 200 59
127.0.0.1 - - [23/Mar/2021 09:35:02] "POST /api/check_in HTTP/1.1" 200 59
127.0.0.1 - - [23/Mar/2021 09:35:04] "POST /api/check_in HTTP/1.1" 200 59
'Name'
```

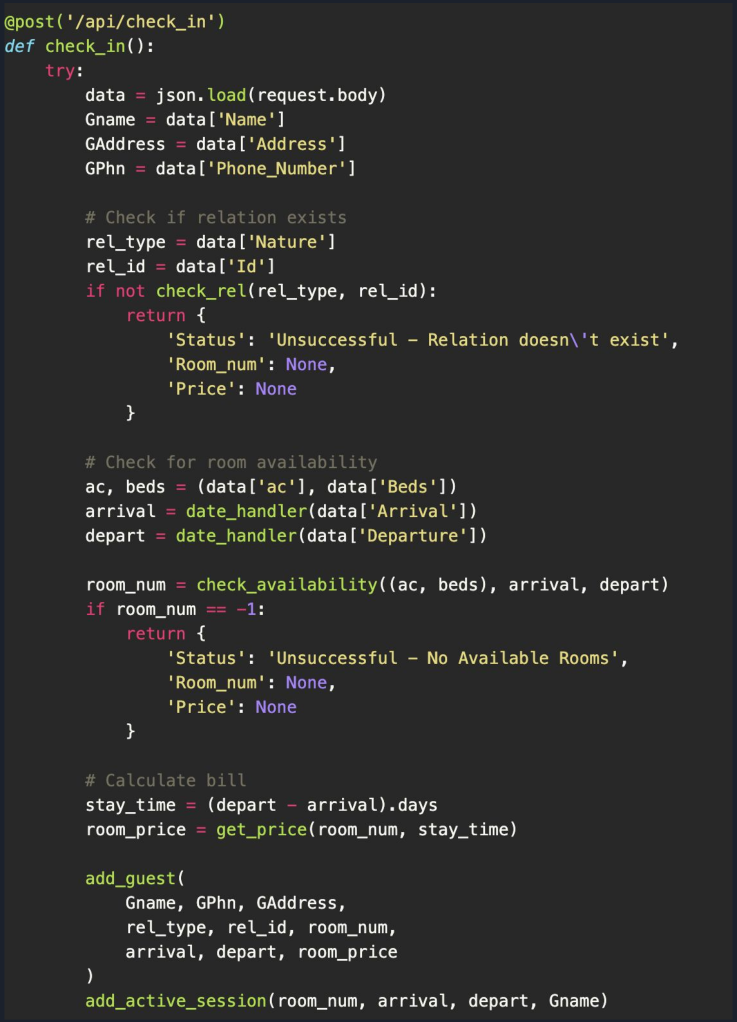
Auxiliary files with related functions

- Are functions called by the API request handling function to retrieve and check for data in the database.

Example:

The /api/check_in handling room booking calls 3 functions:

- check_availability()
- get_price()
- add_guest()
- add_active_session()

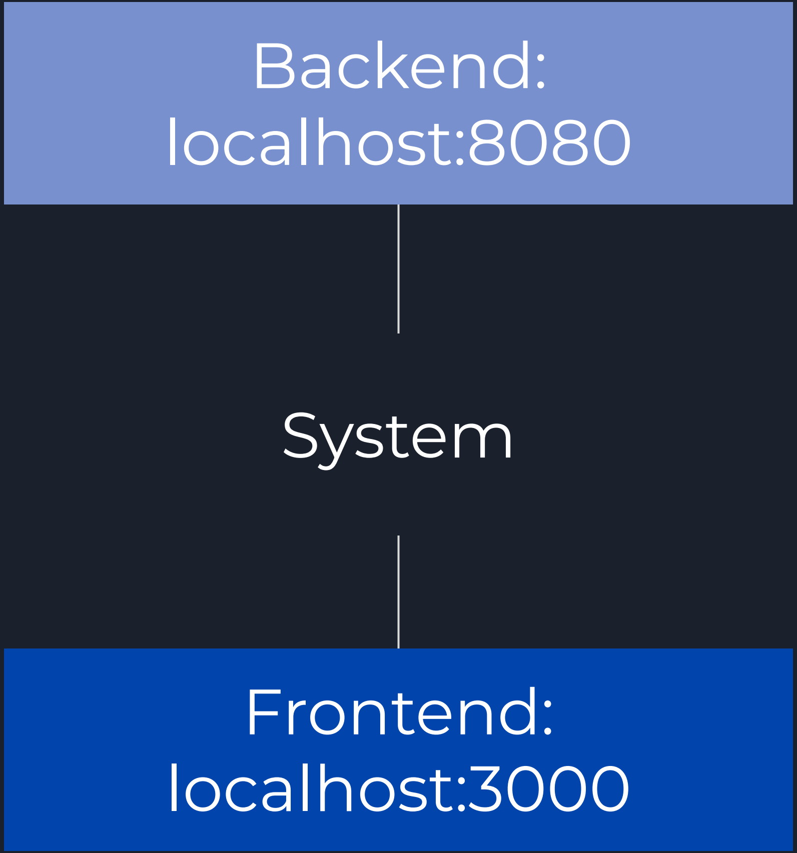


```
@post('/api/check_in')
def check_in():
    try:
        data = json.load(request.body)
        Gname = data['Name']
        GAddress = data['Address']
        GPhn = data['Phone_Number']

        # Check if relation exists
        rel_type = data['Nature']
        rel_id = data['Id']
        if not check_rel(rel_type, rel_id):
            return {
                'Status': 'Unsuccessful - Relation doesn\'t exist',
                'Room_num': None,
                'Price': None
            }

        # Check for room availability
        ac, beds = (data['ac'], data['Beds'])
        arrival = date_handler(data['Arrival'])
        depart = date_handler(data['Departure'])

        room_num = check_availability((ac, beds), arrival, depart)
        if room_num == -1:
            return {
                'Status': 'Unsuccessful - No Available Rooms',
                'Room_num': None,
                'Price': None
            }

        # Calculate bill
        stay_time = (depart - arrival).days
        room_price = get_price(room_num, stay_time)

        add_guest(
            Gname, GPhn, GAddress,
            rel_type, rel_id, room_num,
            arrival, depart, room_price
        )
        add_active_session(room_num, arrival, depart, Gname)
```

How it fits together

The backend runs on port 8080 and listens for requests.

The frontend on an even makes a API request to the backend and data is transferred between them through JSON.

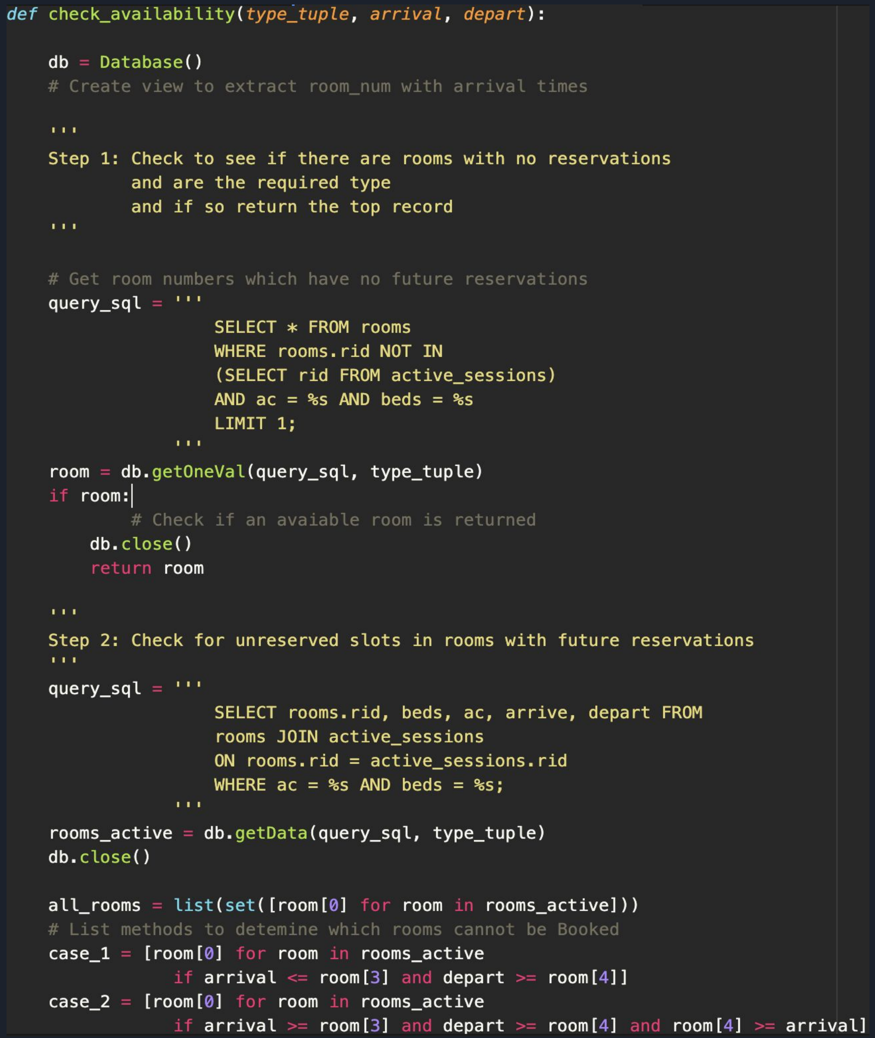


The Case of Using View To query selected data from 2 tables

Approach: Create a view to query data from 2 tables.

Problem: Disconnection or abrupt interruption of function causes view to persist needing to be dropped before the next call.

Solution: Using dynamic views created by “SELECT… AS...” SQL command.



```
def check_availability(type_tuple, arrival, depart):

    db = Database()
    # Create view to extract room_num with arrival times

    '''
    Step 1: Check to see if there are rooms with no reservations
            and are the required type
            and if so return the top record
    '''

    # Get room numbers which have no future reservations
    query_sql = '''
                    SELECT * FROM rooms
                    WHERE rooms.rid NOT IN
                    (SELECT rid FROM active_sessions)
                    AND ac = %s AND beds = %s
                    LIMIT 1;
                '''
    room = db.getOneVal(query_sql, type_tuple)
    if room:
            # Check if an avaiable room is returned
        db.close()
        return room

    '''
    Step 2: Check for unreserved slots in rooms with future reservations
    '''
    query_sql = '''
                    SELECT rooms.rid, beds, ac, arrive, depart FROM
                    rooms JOIN active_sessions
                    ON rooms.rid = active_sessions.rid
                    WHERE ac = %s AND beds = %s;
                '''
    rooms_active = db.getData(query_sql, type_tuple)
    db.close()

    all_rooms = list(set([room[0] for room in rooms_active]))
    # List methods to detemine which rooms cannot be Booked
    case_1 = [room[0] for room in rooms_active
                if arrival <= room[3] and depart >= room[4]]
    case_2 = [room[0] for room in rooms_active
                if arrival >= room[3] and depart >= room[4] and room[4] >= arrival]
```

Thank you