

# SeatBooking KT Document

## Server Url:

Dev URL: <https://sequaredevsaas.sequare.app/space-management/>

Test URL: <https://securetestsaas.sequare.app/space-management/>

Stage URL: <https://sequrestagesaaS.sequare.app/space-management/>

UAT URL: <https://secureuatsaaS.sequare.app/space-management/>

Live URL: <https://booking-management.sequare.app/space-management/>

## IP Address:

Live: 65.1.31.56

Test/Dev/Stage/UAT: 152.67.13.206

## Jenkins Details:

### Image Build:

Dev Build:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-MSS-Backend/job/develop/>

Test Build:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-MSS-Backend/job/test/>

Stage Build:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-MSS-Backend/job/stage/>

UAT Build:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-MSS-Backend/job/uat/>

Live Build:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-MSS-Backend/job/live/>

### Image Pull Build:

Test/Dev/Stage/UAT:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-Dev-Test-Stage-UAT-Image-Pull/>

Live:

<https://deployment.heptagon.tech/view/Sequare-SpaceManagement/job/Sequare-SpaceManagement-Live-Image-Pull/>

## Docker Details:

### Docker Bash Shell Details:

Test/Dev/Stage/UAT:

Host: 152.67.13.206

Cmd: docker exec -it Sequare-SM-BK-(server name) bash

Live:

Host: 65.1.31.56

Cmd: docker exec -it Sequare-SM-BK-Live bash

### Explanation of All Menu:

#### Overview:

##### 1. Analytics

###### 1. Employees:

- **Count:** Displays the total number of active employees in that location.
- **Purpose:** Helps in understanding the workforce size.

###### 2. Expected Arrivals:

- **Count:** Shows the number of employees expected to arrive today (Total Approved booking for that day).
- **Purpose:** Provides insight into the day's anticipated attendance.

###### 3. Actual Check-Ins:

- **Count:** Indicates the number of employees who have checked in today.
- **Purpose:** Tracks actual attendance and helps in managing occupancy.

###### 4. Pending Requests:

- **Count:** Lists the number of pending bookings.
- **Purpose:** Highlights requests that need attention and action.

###### 5. Occupancy:

- **Count:** Displays the current occupancy level.
- **Purpose:** Helps in monitoring the usage of spaces and managing crowding.

##### 2. Today's Statistics

- **Details:** Shows the number of bookings made for today (Approved, Pending, Rejected).
- **Purpose:** Provides a quick snapshot of today's activity.

##### 3. Weekly Statistics

- **Graph:** Displays a graphical representation of booking statuses (Approved, Pending, Rejected) over a week.
- **Date Range:** Allows selection of date range to view statistics.
- **Purpose:** Helps in analyzing trends and patterns in bookings over the week.

##### 4. Occupancy Overview

- **Map:** Visual representation of the floor plan with indications of available and occupied seats.
- **Available/Occupied Count:** Shows the count of available and occupied seats.
- **Purpose:** Provides a detailed view of space utilization in real-time.

## 5. Department Wise Occupancy

- **Bar Graph:** Displays occupancy percentage for each department.
- **Department List:** Lists various departments with their occupancy statistics.
- **Purpose:** Helps in understanding space usage by different departments and aids in efficient space management.

### Configuration:

#### 1. Company Configuration

- **Request Types:**
  - **Daily, Weekly, Monthly, Custom Date [Checkboxes]:**
    - **Purpose:** Allows users to specify the frequency and type of requests that can be made (e.g., daily booking, weekly bookings).
    - **Usage:** Check the boxes to enable or disable request types according to company policy.

#### 2. Space Management Configuration

- **Seat Colors:**
  - **Available Seat Color:** Color picker
    - **Purpose:** Indicates seats that are available for booking.
  - **Booked Seat Color:** Color picker
    - **Purpose:** Indicates seats that have been booked.
  - **Selected Seat Color:** Color picker
    - **Purpose:** Indicates the seat currently selected by a user.
  - **Non-Available Seat Color:** Color picker
    - **Purpose:** Indicates seats that are not available for booking.
  - **Booked but Unconfirmed Seat Color:**
    - **Option:** Yes/No
    - **Purpose:** Option to differentiate between booked and booked but unconfirmed seats with a different color.

#### 3. Seat Management Configuration

- **Location Wise Capacity (%):**
  - **Purpose:** Set the maximum booking capacity for each location as a percentage.
- **Seat Number Prefix:** Enter Prefix
  - **Purpose:** Prefix to be used for seat numbers.
- **Seat Number Suffix:** Enter Suffix
  - **Purpose:** Suffix to be used for seat numbers.
- **Override Prefix/Suffix:**
  - **Option:** Yes/No
  - **Purpose:** Allow locations to override the default seat number prefix and suffix.

#### 4. Approval Process Configuration

- **Auto-Approve Seat Booking Requests:**
  - **Option:** Yes/No
  - **Purpose:** Enable or disable automatic approval of seat booking requests.
- **Health Declaration:**
  - **Option:** Yes/No
  - **Purpose:** Require employees to complete a health declaration when booking a seat.

## **5. Seat Management Configuration**

- **Display Other Employee Details:**
  - **Option:** Yes/No
  - **Purpose:** Show details of other employees when booking a seat.
- **Seat Availability if Employee is Absent:**
  - **Option:** Yes/No
  - **Purpose:** Make seats available for other bookings if the original booker is absent.
  - **Waiting Time:** User can set waiting time before auto cancelling booked seat
  - **Purpose:** Automatically cancel the seat booking if not used within the specified time after the shift starts.
- **Option to Cancel Seat Booking Request:**
  - **Option:** Yes/No
  - **Purpose:** Allow users to cancel their seat booking requests.
- **Option to Reschedule Seat Booking Request:**
  - **Option:** Yes/No
  - **Purpose:** Allow users to reschedule their seat booking requests.
  - **Rescheduling Time Frame:** User can set time for rescheduling
  - **Purpose:** Specify the minimum number of hours before the scheduled time that a booking can be rescheduled.
- **Advance Booking Period:** Enter number of days
  - **Purpose:** Specify how many days in advance a booking can be made.

## **Request Type Management:**

### **1. Assign/Edit Request Types Section**

- **Employee Role:** Lists the roles of employees within the organization (e.g., Senior Executive - HR, Technical Manager, Solutions Architect).
- **Request Type:** Displays the types of requests assigned to each role. In this case, the types include Daily, Weekly, Monthly, and Custom Date.
- **Action:** Provides an option to edit the request types for each specific role. Clicking the pencil icon allows administrators to modify the request types assigned to that role.

### **2. Edit All Button**

- **Purpose:** Allows administrators to batch edit request types for multiple roles at once, streamlining the process of assigning request types across different roles.

- **Usage:** Click on "Edit All" to make changes to request types for all listed roles simultaneously.

### 3. Search by Employee Role

- **Purpose:** A search bar to quickly find specific employee roles.
- **Usage:** Type in the role name to filter the list and easily locate the role you want to manage.

### 4. Pagination Controls

- **Rows per page:** A dropdown menu to select the number of rows displayed per page (e.g., 10, 25, 50).
- **Page Navigation:** Buttons to navigate between pages if the list of roles spans multiple pages.

## Location Management:

The **Location Management** menu in the SEQUE system allows administrators to manage the assignment and editing of SPOC (Single Point of Contact) and capacity settings for different locations. Below is an explanation of the elements in this menu:

### Overview

- The screen provides an interface to manage location-specific configurations, including seat number prefixes and suffixes, and department-wise capacity and SPOC assignments.

### *Elements in the Location Management Menu*

#### 1. Assign/Edit SPOC and Capacity Section

- **Location:** Lists the locations within the organization (e.g., Coimbatore - Root8).
- **City:** Specifies the city where the location is situated (e.g., Coimbatore, Chennai).
- **Company:** Indicates the company name (e.g., Conneqt Digital).
- **Seat Prefix:** Shows the prefix assigned to seat numbers in that location (e.g., S1).
- **Seat Suffix:** Shows the suffix assigned to seat numbers in that location (e.g., S2).
- **Action:** Provides an option to edit the location settings. Clicking the pencil icon allows administrators to modify the seat prefixes, suffixes, and capacity settings for the location.

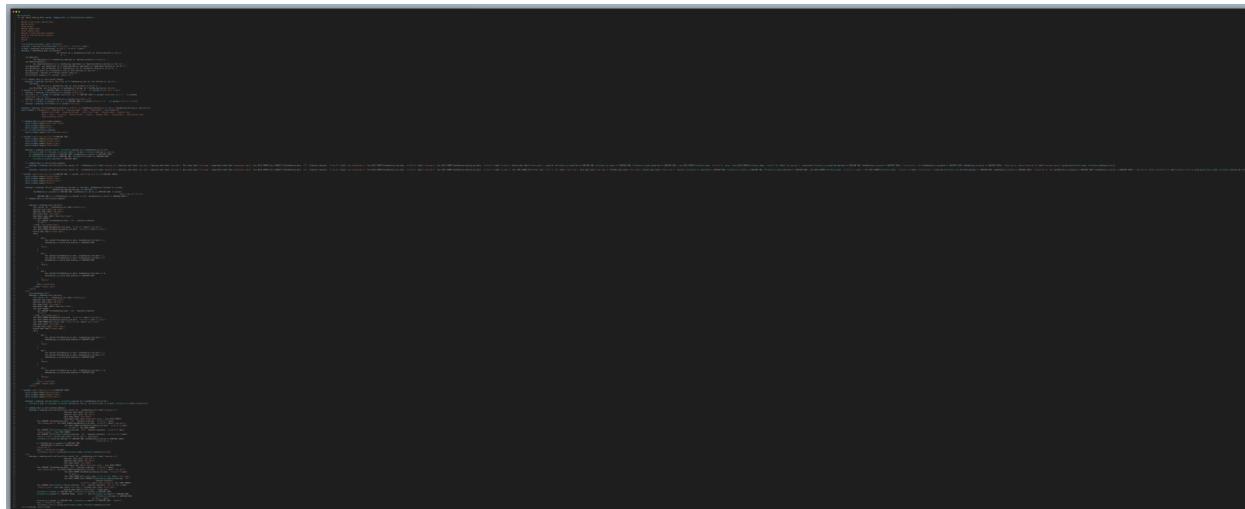
#### 2. Location Configuration Screen

- **Seat Number Prefix/Suffix:** Fields to enter and edit the seat number prefix and suffix for the selected location.
- **Department Section:** Lists the departments within the selected location, showing:
  - **Department Name:** Name of the department.
  - **Capacity (%):** The capacity percentage assigned to the department, which sets the maximum booking capacity for each department as a percentage.

- **Count of SPOC(s)**: Number of Single Points of Contact assigned to the department. Department SPOCs can accept, reject, and view the bookings of that department.
- **Action**: Edit button to modify department settings.
- **Roles Section**: Lists the roles within the selected location, showing:
  - **Role Name**: Name of the role.
  - **Department**: The department to which the role belongs.
  - **Count of SPOC(s)**: Number of Single Points of Contact assigned to the role. Role SPOCs can accept, reject, and view the bookings of that role.
  - **Action**: Edit button to modify role settings.
- **Pagination Controls**: Buttons to navigate between pages if the list of departments or roles spans multiple pages.
- **Search by Location/Department/Role**: Search bars to quickly find specific locations, departments, or roles.

## All Important Query Explanation:

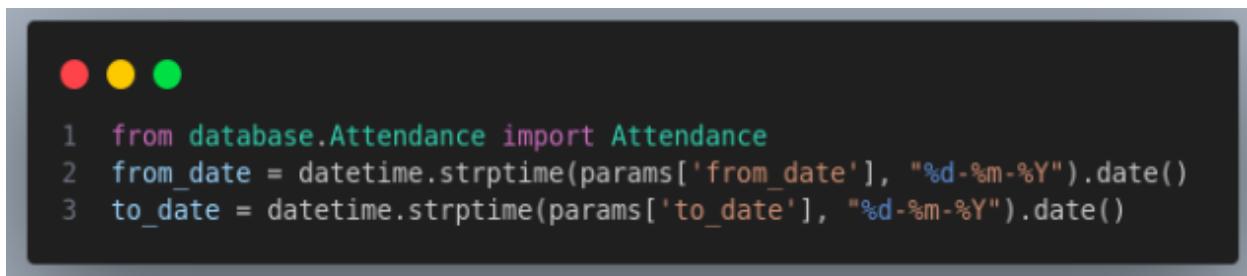
### 1. Booking Report query:



A screenshot of a terminal window displaying a large volume of text, likely a booking report. The text is too dense to read individually but represents a significant amount of data output from a query.

### Query Breakdown

#### 1. Imports and Date Conversion:



```

● ● ●

1 from database.Attendance import Attendance
2 from_date = datetime.strptime(params['from_date'], "%d-%m-%Y").date()
3 to_date = datetime.strptime(params['to_date'], "%d-%m-%Y").date()

```

#### 2. Initial Query Setup:

```

1 bookings = SeatBooking.query.join(Branch,
2                                     and_(Branch.id == SeatBooking.branch_id, Branch.deleted.in_((0,1)),
3                                           )). \
4         join(Employee,
5               and_(Employee.id == SeatBooking.employee_id, Employee.deleted.in_((0,1))). \
6         join(DepartmentDetail,
7               and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted.in_((0,1))). \
8         join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted.in_((0,1))). \
9         join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted.in_((0,1))). \
10        join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted.in_((0,1))). \
11        join(Timezone, Timezone.id == Branch.country_code_id)). \
12        filter(Branch.company_id == params['company_id'])

```

### Joins Explained:

- **Branch**: Joins SeatBooking with Branch on branch\_id, ensuring that the branch is not deleted.
- **Employee**: Joins SeatBooking with Employee on employee\_id, ensuring that the employee is not deleted.
- **DepartmentDetail**: Joins SeatBooking with DepartmentDetail on department\_id, ensuring that the department detail is not deleted.
- **Department**: Joins DepartmentDetail with Department on department\_id, ensuring that the department is not deleted.
- **RoleDetail**: Joins SeatBooking with RoleDetail on role\_id, ensuring that the role detail is not deleted.
- **Role**: Joins RoleDetail with Role on role\_id, ensuring that the role is not deleted.
- **Timezone**: Joins SeatBooking with Timezone using country\_code\_id from Branch.

### 3. Initial Query Setup:

```

1 if not company_data.is_oracle_based_company:
2     bookings = bookings.join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted.in_((0,1))). \
3                               join(Seat,
4                                     and_(Seat.id == SeatBooking.seat_id, Seat.deleted.in_((0,1))). \
5                               join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_id, FloorMap.deleted.in_((0,1)))

```

### Joins Explained:

- **Slot**: Joins SeatBooking with Slot on slot\_id, ensuring that the slot is not deleted.
- **Seat**: Joins SeatBooking with Seat on seat\_id, ensuring that the seat is not deleted.
- **FloorMap**: Joins SeatBooking with FloorMap on floormap\_id, ensuring that the floor map is not deleted.

### 4. Filters Based on Parameters:

```

1 if params['branch_id'] != CONSTANT.ZERO and params['branch_id'] != '' and params['branch_id'] != None:
2     bookings = bookings.filter(Branch.id == params['branch_id'])
3 if 'department_id' in params and params['department_id'] != CONSTANT.ZERO and params['department_id'] != '' and params[
4     'department_id'] != None:
5     bookings = bookings.filter(Department.id == params['department_id'])
6 if 'role_id' in params and params['role_id'] != CONSTANT.ZERO and params['role_id'] != '' and params['role_id'] != None:
7     bookings = bookings.filter(Role.id == params['role_id'])

```

## Filters Explained:

- **Branch ID:** Filters bookings based on branch\_id if it is provided and valid.
- **Department ID:** Filters bookings based on department\_id if it is provided and valid.
- **Role ID:** Filters bookings based on role\_id if it is provided and valid.

## 5. General Filters for Bookings:

```
1 bookings = bookings.filter(SeatBooking.deleted.in_((0,1)), or_(SeatBooking.deleted_at.is_(None), SeatBooking.deleted_at.isnot(None)))
```

## Filters Explained:

- Ensures that bookings are either not deleted or have a valid deleted\_at timestamp.

## 6. Excel Columns Configuration:

```
1 excel_columns = ['Booking Id', 'Employee Id', 'Employee Name', 'Role', 'Department', 'Seat Booked On',
2 'Booking Start Date', 'Booking End Date', 'Shift Start Time', 'CheckIn Date', 'CheckIn Time',
3 'Seat', 'Floor', 'Location', 'Rejected Date', 'Status', 'Request Type', 'Custom Dates', 'Declaration Form',
4 'Food Preference Form']
```

- Initial list of columns for the Excel report.

## 7. Adjusting Columns for Oracle-based Companies:

```
1 if company_data.is_oracle_based_company:
2     excel_columns.remove('Shift Start Time')
3     excel_columns.remove('Seat')
4     excel_columns.remove('Floor')
5     if not is_food_preference_enabled:
6         excel_columns.remove('Food Preference Form')
```

- Ensures that bookings are either not deleted or have a valid deleted\_at timestamp.

## 8. Filters and Query Adjustments Based on Report Type:

### Rejected or Cancelled Bookings Report (Type 2):

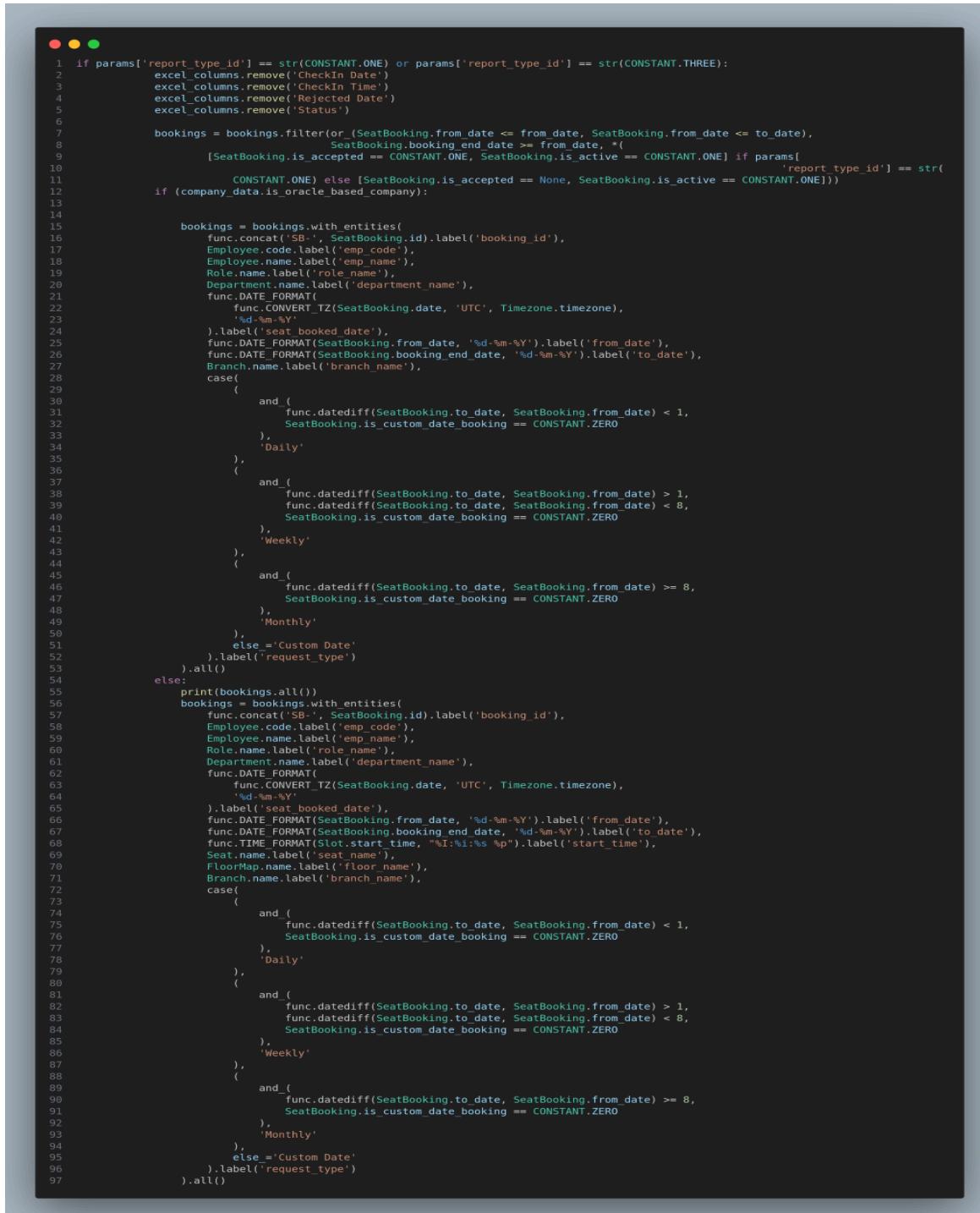
```
1 #!/usr/bin/python
2 # coding: utf-8
3 
4 # This script performs a query on the database to retrieve rejected or cancelled bookings for a specific date range.
5 # The results are then converted into an Excel file for reporting purposes.
6 
7 import sys
8 from datetime import datetime, timedelta
9 import pandas as pd
10 from openpyxl import Workbook
11 
12 # Set the connection string for the database
13 connection_string = "DRIVER={ODBC Driver 17 for SQL Server};Server=DESKTOP-4I9K4D8;Database=AdventureworksLT;Trusted_Connection=yes;Encrypt=0;TrustServerCertificate=1"
14 
15 # Set the date range for the report
16 start_date = "2023-01-01"
17 end_date = "2023-01-31"
18 
19 # Create a new DataFrame to hold the results
20 df = pd.DataFrame()
21 
22 # Create a connection to the database
23 conn = pyodbc.connect(connection_string)
24 
25 # Execute the query
26 cursor = conn.cursor()
27 cursor.execute("SELECT * FROM [AdventureworksLT].[dbo].[SeatBooking] WHERE [Status] IN ('Rejected', 'Cancelled') AND [RejectedOn] BETWEEN ? AND ? ORDER BY [RejectedOn] DESC", (start_date, end_date))
28 
29 # Fetch all rows from the cursor
30 rows = cursor.fetchall()
31 
32 # Convert the rows into a DataFrame
33 df = pd.DataFrame(rows, columns=[column[0] for column in cursor.description])
34 
35 # Save the DataFrame to an Excel file
36 df.to_excel("RejectedBookingsReport.xlsx", index=False)
37 
38 # Close the connection
39 conn.close()
```

## Explanation:

- Columns related to check-in date, check-in time, request type, and custom dates are removed from the Excel columns list.

- The bookings table is joined with the Attendance table based on the condition Attendance.booking\_id == SeatBooking.id.
- Filters are applied to bookings based on dates, attendance deletion status, booking acceptance status, and attendance cancellation/rejection status.
- Depending on whether the company is Oracle-based or not, different entities are selected for the report:
  - For Oracle-based companies: The selected entities include booking ID, employee code, name, role, department, seat booking date, from date, to date, branch name, and booking status (canceled or rejected).
  - For non-Oracle based companies: Additional information such as start time, seat name, and floor name are included.

### Accepted or Pending Bookings Report (Type 1 or 3):



```

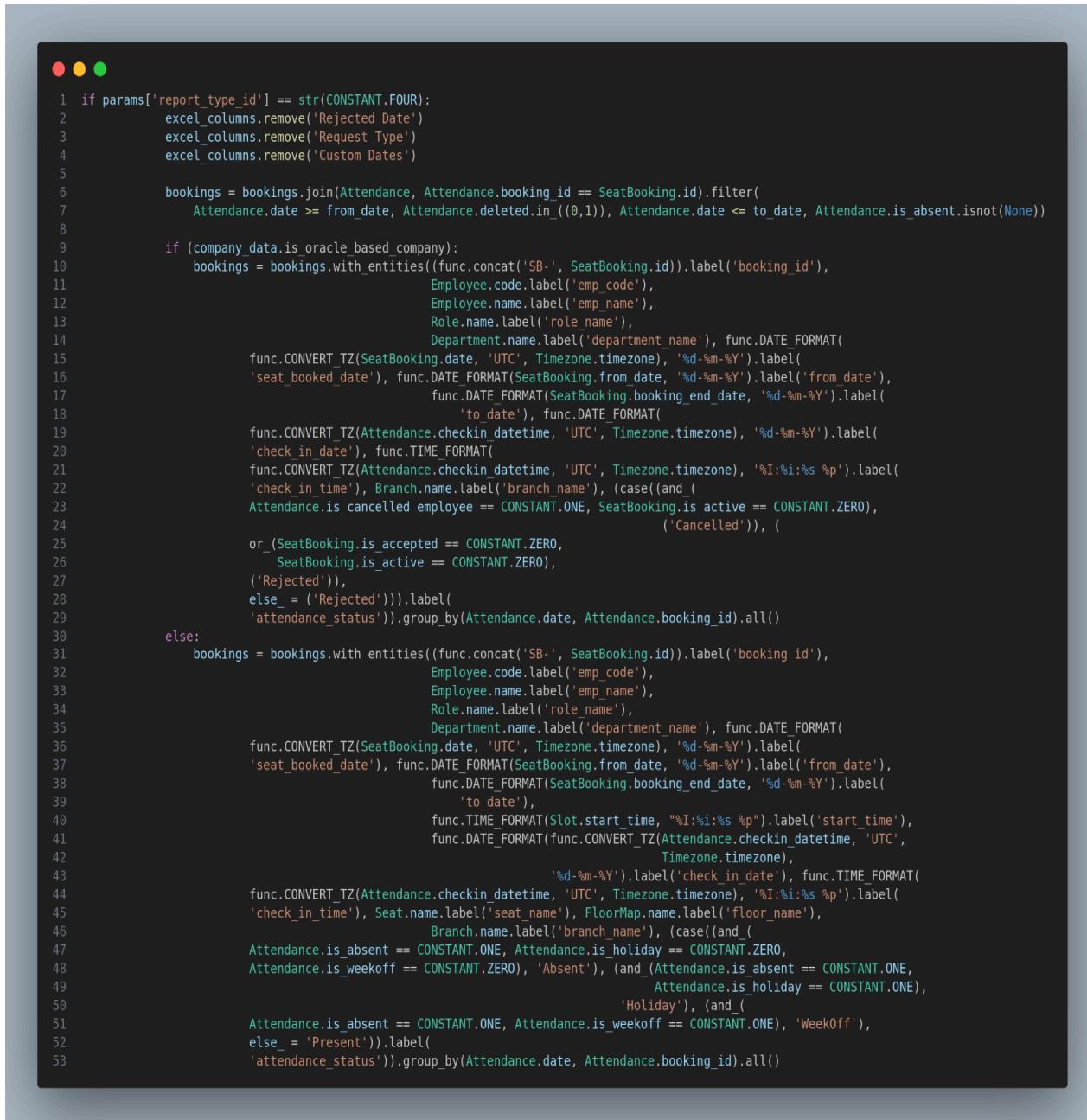
1  if params['report_type_id'] == str(CONSTANT.ONE) or params['report_type_id'] == str(CONSTANT.THREE):
2      excel_columns.remove('CheckIn Date')
3      excel_columns.remove('CheckIn Time')
4      excel_columns.remove('Rejected Date')
5      excel_columns.remove('Status')
6
7      bookings = bookings.filter(or_(SeatBooking.from_date <= from_date, SeatBooking.from_date <= to_date),
8                               SeatBooking.booking_end_date >= from_date, *(
9                                 [SeatBooking.is_accepted == CONSTANT.ONE, SeatBooking.is_active == CONSTANT.ONE] if params[
10                                'report_type_id'] == str(
11                                   CONSTANT.ONE) else [SeatBooking.is_accepted == None, SeatBooking.is_active == CONSTANT.ONE]))
12     if (company_data.is_oracle_based_company):
13
14         bookings = bookings.with_entities(
15             func.concat('SB-', SeatBooking.id).label('booking_id'),
16             Employee.code.label('emp_code'),
17             Employee.name.label('emp_name'),
18             Role.name.label('role_name'),
19             Department.name.label('department_name'),
20             func.DATE_FORMAT(
21                 func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone),
22                 'dd-%m-%Y'
23             ).label('seat_booked_date'),
24             func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
25             func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label('to_date'),
26             Branch.name.label('branch_name'),
27             case(
28                 (
29                     and_(
30                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 1,
31                         SeatBooking.is_custom_date_booking == CONSTANT.ZERO
32                     ),
33                     'Daily'
34                 ),
35                 (
36                     and_(
37                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) > 1,
38                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 8,
39                         SeatBooking.is_custom_date_booking == CONSTANT.ZERO
40                     ),
41                     'Weekly'
42                 ),
43                 (
44                     and_(
45                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) >= 8,
46                         SeatBooking.is_custom_date_booking == CONSTANT.ZERO
47                     ),
48                     'Monthly'
49                 ),
50                 else_= 'Custom Date'
51             ).label('request_type')
52         ).all()
53     else:
54         print(bookings.all())
55         bookings = bookings.with_entities(
56             func.concat('SB-', SeatBooking.id).label('booking_id'),
57             Employee.code.label('emp_code'),
58             Employee.name.label('emp_name'),
59             Role.name.label('role_name'),
60             Department.name.label('department_name'),
61             func.DATE_FORMAT(
62                 func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone),
63                 'dd-%m-%Y'
64             ).label('seat_booked_date'),
65             func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
66             func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label('to_date'),
67             func.TIME_FORMAT(Slot.start_time, "%I:%M:%S %p").label('start_time'),
68             Seat.name.label('seat_name'),
69             FloorMap.name.label('floor_name'),
70             Branch.name.label('branch_name'),
71             case(
72                 (
73                     and_(
74                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 1,
75                         SeatBooking.is_custom_date_booking == CONSTANT.ZERO
76                     ),
77                     'Daily'
78                 ),
79                 (
80                     and_(
81                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) > 1,
82                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 8,
83                         SeatBooking.is_custom_date_booking == CONSTANT.ZERO
84                     ),
85                     'Weekly'
86                 ),
87                 (
88                     and_(
89                         func.datediff(SeatBooking.to_date, SeatBooking.from_date) >= 8,
90                         SeatBooking.is_custom_date_booking == CONSTANT.ZERO
91                     ),
92                     'Monthly'
93                 ),
94                 else_= 'Custom Date'
95             ).label('request_type')
96         ).all()

```

## Explanation:

- Similar to Report Type Two, specific columns are removed from the Excel columns list.
- Bookings are filtered based on the report type (ONE or THREE) and the specified date range.
- Depending on the company's Oracle-based status, different queries will be executed. We'll detail these in the Oracle and non-Oracle sections below.
- For Oracle-based companies: The selected entities include booking ID, employee code, name, role, department, seat booking date, from date, to date, branch name, and request\_type (Daily or Weekly or Custom Date).
- For non-Oracle based companies: Additional information such as start time, seat name, and floor name are included.

## Attendance Report (Type 4):



```
1 if params['report_type_id'] == str(CONSTANT.FOUR):
2     excel_columns.remove('Rejected Date')
3     excel_columns.remove('Request Type')
4     excel_columns.remove('Custom Dates')
5
6     bookings = bookings.join(Attendance, Attendance.booking_id == SeatBooking.id).filter(
7         Attendance.date >= from_date, Attendance.deleted.in_((0,1)), Attendance.date <= to_date, Attendance.is_absent.isnot(None))
8
9     if (company_data.is_oracle_based_company):
10         bookings = bookings.with_entities(func.concat('SB-', SeatBooking.id).label('booking_id'),
11                                         Employee.code.label('emp_code'),
12                                         Employee.name.label('emp_name'),
13                                         Role.name.label('role_name'),
14                                         Department.name.label('department_name'), func.DATE_FORMAT(
15                                         func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone), '%d-%m-%Y').label(
16                                         'seat_booked_date'), func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
17                                         func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label(
18                                         'to_date'), func.DATE_FORMAT(
19                                         func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC', Timezone.timezone), '%d-%m-%Y').label(
20                                         'check_in_date'), func.TIME_FORMAT(
21                                         func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC', Timezone.timezone), '%I:%i:%s %p').label(
22                                         'check_in_time'), Branch.name.label('branch_name'), (case((and_(
23                                         Attendance.is_cancelled_employee == CONSTANT.ONE, SeatBooking.is_active == CONSTANT.ZERO),
24                                         ('Cancelled')), (
25                                         or_(SeatBooking.is_accepted == CONSTANT.ZERO,
26                                             SeatBooking.is_active == CONSTANT.ZERO),
27                                         ('Rejected'))),
28                                         else_ = ('Rejected')).label(
29                                         'attendance_status')).group_by(Attendance.date, Attendance.booking_id).all()
30     else:
31         bookings = bookings.with_entities(func.concat('SB-', SeatBooking.id).label('booking_id'),
32                                         Employee.code.label('emp_code'),
33                                         Employee.name.label('emp_name'),
34                                         Role.name.label('role_name'),
35                                         Department.name.label('department_name'), func.DATE_FORMAT(
36                                         func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone), '%d-%m-%Y').label(
37                                         'seat_booked_date'), func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
38                                         func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label(
39                                         'to_date'),
40                                         func.TIME_FORMAT(Slot.start_time, "%I:%i:%s %p").label('start_time'),
41                                         func.DATE_FORMAT(func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC',
42                                         Timezone.timezone),
43                                         '%d-%m-%Y').label('check_in_date'), func.TIME_FORMAT(
44                                         func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC', Timezone.timezone), '%I:%i:%s %p').label(
45                                         'check_in_time'), Seat.name.label('seat_name'), FloorMap.name.label('floor_name'),
46                                         Branch.name.label('branch_name'), (case((and_(
47                                         Attendance.is_absent == CONSTANT.ONE, Attendance.is_holiday == CONSTANT.ZERO,
48                                         Attendance.is_weekoff == CONSTANT.ZERO), 'Absent'), (and_(Attendance.is_absent == CONSTANT.ONE,
49                                         Attendance.is_holiday == CONSTANT.ONE),
50                                         ('Holiday')), (and_(
51                                         Attendance.is_absent == CONSTANT.ONE, Attendance.is_weekoff == CONSTANT.ONE), 'WeekOff'),
52                                         else_ = 'Present')).label(
53                                         'attendance_status')).group_by(Attendance.date, Attendance.booking_id).all()
```

## Explanation:

- Specific columns related to rejected date, request type, and custom dates are removed from the Excel columns list.
- The bookings table is joined with the Attendance table based on the condition Attendance.booking\_id == SeatBooking.id.

- Filters are applied to bookings based on dates and attendance absence status.
- Depending on the company's Oracle-based status, different queries will be executed. We'll detail these in the Oracle and non-Oracle sections below.
- For Oracle-based companies: The selected entities include booking ID, employee code, name, role, department, seat booking date, from date, to date, branch name, and r attendance\_status (Absent or Holiday or WeekOff).
- For non-Oracle based companies: Additional information such as start time, seat name, and floor name are included.

## 2. Department Wise Occupancy query:

```

1  @staticmethod
2  def get_department_wise_booking_counts( branch_id, date, page, per_page ):
3      """
4          Method to get_department_wise_booking_counts
5          @param branch_id:
6              @type branch_id:
7          @param date:
8              @type date:
9          @param page:
10             @type page:
11         @param per_page:
12             @type per_page:
13         @return:
14             @rtype:
15         """
16     total_occupied = (func.round((func.count(distinct(SeatBooking.id)) * 100) /
17                                     func.count(
18                                         distinct(
19                                             Employee.id))) + func.round(
20                                         (func.count(distinct(CustomDateBooking.id)) * 100) /
21                                         func.count(
22                                             distinct(
23                                                 Employee.id))).label('total_occupied')
24     return Department.query.join(DepartmentDetail, DepartmentDetail.department_id == Department.id). \
25         join(Employee, Employee.department_id == Department.department_id). \
26         join(SeatBooking, and_(SeatBooking.employee_id == Employee.id,
27                               SeatBooking.branch_id == branch_id,
28                               SeatBooking.is_active == CONSTANT.ONE,
29                               SeatBooking.is_accepted == CONSTANT.ONE,
30                               SeatBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
31                               SeatBooking.from_date <= date, SeatBooking.booking_end_date >= date,
32                               SeatBooking.is_custom_date_booking == CONSTANT.ZERO,
33                               SeatBooking.deleted == CONSTANT.ZERO, SeatBooking.deleted_at == None),
34                               isouter = True). \
35         join(CustomDateBooking, and_(CustomDateBooking.employee_id == Employee.id,
36                                     CustomDateBooking.branch_id == branch_id,
37                                     CustomDateBooking.is_active == CONSTANT.ACTIVE,
38                                     CustomDateBooking.from_date == date,
39                                     CustomDateBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
40                                     CustomDateBooking.deleted == CONSTANT.ZERO,
41                                     CustomDateBooking.deleted_at == None
42                               ), isouter = True). \
43         with_entities(Department.id, Department.name, total_occupied
44                         ).filter(
45                             DepartmentDetail.branch_id == branch_id, Employee.branch_id == branch_id,
46                             DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None,
47                             Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None).order_by(
48                                 total_occupied.desc(), Department.name.asc()).group_by(Department.id).paginate(page, per_page,
49                                         error_out = False)
50

```

## Query Explanation:



```
1 total_occupied = (func.round((func.count(distinct(SeatBooking.id)) * 100) /
2                         func.count(
3                             distinct(
4                                 Employee.id))) + func.round(
5                         (func.count(distinct(CustomDateBooking.id)) * 100) /
6                             func.count(
7                                 distinct(
8                                     Employee.id))).label('total_occupied')
```

## Components of the Expression

1. **Distinct Seat Bookings:**
  - `func.count(distinct(SeatBooking.id))`: This part counts the distinct SeatBooking IDs, representing the total number of unique seat bookings.
2. **Distinct Custom Date Bookings:**
  - `func.count(distinct(CustomDateBooking.id))`: This part counts the distinct CustomDateBooking IDs, representing the total number of unique custom date bookings.
3. **Distinct Employees:**
  - `func.count(distinct(Employee.id))`: This part counts the distinct Employee IDs, representing the total number of unique employees.
4. **Calculating the Occupancy Percentage for Seat Bookings:**
  - `(func.count(distinct(SeatBooking.id)) * 100) / func.count(distinct(Employee.id))`: This calculates the percentage of seat bookings in relation to the total number of employees.
5. **Calculating the Occupancy Percentage for Custom Date Bookings:**
  - `(func.count(distinct(CustomDateBooking.id)) * 100) / func.count(distinct(Employee.id))`: This calculates the percentage of custom date bookings in relation to the total number of employees.
6. **Rounding the Percentages:**
  - `func.round(...)`: Each percentage is rounded to the nearest whole number.
7. **Summing the Percentages:**
  - The rounded percentages from seat bookings and custom date bookings are summed together.
8. **Labeling the Result:**
  - `.label('total_occupied')`: The result is labeled as `total_occupied` for use in the query results.

```

1  return Department.query.join(DepartmentDetail, DepartmentDetail.department_id == Department.id). \
2      join(Employee, Employee.department_id == Department.department_id). \
3      join(SeatBooking, and_(SeatBooking.employee_id == Employee.id,
4                          SeatBooking.branch_id == branch_id,
5                          SeatBooking.is_active == CONSTANT.ONE,
6                          SeatBooking.is_accepted == CONSTANT.ONE,
7                          SeatBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
8                          SeatBooking.from_date <= date, SeatBooking.booking_end_date >= date,
9                          SeatBooking.is_custom_date_booking == CONSTANT.ZERO,
10                         SeatBooking.deleted == CONSTANT.ZERO, SeatBooking.deleted_at == None),
11                         isouter = True). \
12     join(CustomDateBooking, and_(CustomDateBooking.employee_id == Employee.id,
13                                 CustomDateBooking.branch_id == branch_id,
14                                 CustomDateBooking.is_active == CONSTANT.ACTIVE,
15                                 CustomDateBooking.from_date == date,
16                                 CustomDateBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
17                                 CustomDateBooking.deleted == CONSTANT.ZERO,
18                                 CustomDateBooking.deleted_at == None
19                         ), isouter = True), \
20     with_entities(Department.id, Department.name, total_occupied
21                   ).filter(
22         DepartmentDetail.branch_id == branch_id, Employee.branch_id == branch_id,
23         DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None,
24         Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None).order_by(
25             total_occupied.desc(), Department.name.asc()).group_by(Department.id).paginate(page, per_page,
26                                         error_out = False)

```

**1. Department.query:** Starts building a query on the Department table.

**2. join(DepartmentDetail, DepartmentDetail.department\_id == Department.id):**

- Joins the DepartmentDetail table on the department\_id field.

**3. join(Employee, Employee.department\_id == Department.department\_id):**

- Joins the Employee table on the department\_id field.

**4. join(SeatBooking, ... , isouter=True):**

- Left outer join with the SeatBooking table on multiple conditions:
  - employee\_id matches Employee.id.
  - branch\_id matches the input branch\_id.
  - SeatBooking is active, accepted, and not deleted.
  - The booking date range includes the input date.
  - The booking is not a custom date booking.

**5. join(CustomDateBooking, ... , isouter=True):**

- Left outer join with the CustomDateBooking table on multiple conditions:
  - employee\_id matches Employee.id.
  - branch\_id matches the input branch\_id.
  - CustomDateBooking is active and for the exact date.
  - The custom date booking is not deleted.

**6. with\_entities(Department.id, Department.name, total\_occupied):**

- Selects Department.id, Department.name, and the calculated total\_occupied fields.

**7. filter(...):**

- Filters results to ensure records are relevant to the input branch\_id and not deleted.

**8. order\_by(total\_occupied.desc(), Department.name.asc()):**

- Orders the results by total\_occupied in descending order and then by Department.name in ascending order.

**9. group\_by(Department.id):**

- Groups the results by Department.id.

**10. paginate(page, per\_page, error\_out=False):**

- Paginates the results based on the input page and per\_page parameters.
- error\_out=False ensures no error is thrown if the page number is out of range; instead, an empty result set is returned.

### 3. Mobile seat list api queries and Custom format function:

#### 1.Query to get booked seat ids

```

1  @staticmethod
2  def get_booked_seat_ids(branch_details, slot_details, from_date, to_date, is_custom_date_booking, custom_dates):
3      """
4          Method to get booked seat IDs.
5
6          :param branch_details: Details of the branch.
7          :type branch_details: object
8          :param slot_details: Details of the slot.
9          :type slot_details: object
10         :param from_date: Starting date of the booking.
11         :type from_date: datetime
12         :param to_date: Ending date of the booking.
13         :type to_date: datetime
14         :param is_custom_date_booking: Flag to indicate custom date booking.
15         :type is_custom_date_booking: bool
16         :param custom_dates: List of custom dates.
17         :type custom_dates: list
18         :return: List of booked seat IDs.
19         :rtype: list
20     """
21     from database.CancellationForUnusedBooking import CancellationForUnusedBooking
22
23     seat_ids = []
24     custom_booking_seat_ids = []
25
26     combined_from_datetime = func.concat(
27         func.cast(SeatBooking.from_date, String),
28         func.cast(SeatBooking.start_time, String)
29     )
30     combined_to_datetime = func.concat(
31         func.cast(SeatBooking.booking_end_date, String),
32         func.cast(SeatBooking.end_time, String)
33     )
34
35     booking_end_date = to_date
36     if slot_details.start_time > slot_details.end_time:
37         to_date += timedelta(days=CONSTANT.ONE)
38
39     booking_obj = (
40         SeatBooking.query
41         .join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted_at == CONSTANT.ZERO, Branch.deleted_at == None))
42         .join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None))
43         .join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None))
44         .join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None))
45         .join(Role, and_(Role.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Role.deleted == None))
46         .join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None))
47         .join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None))
48         .join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None))
49         .join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_id, FloorMap.deleted == CONSTANT.ZERO, FloorMap.deleted_at == None))
50         .filter(
51             SeatBooking.is_active == True,
52             SeatBooking.branch_id == branch_details.id,
53             SeatBooking.deleted == CONSTANT.ZERO,
54             SeatBooking.deleted_at == None,
55             or_(
56                 and_(
57                     SeatBooking.from_date <= to_date,
58                     SeatBooking.from_date >= from_date,
59                     SeatBooking.is_night_shift == CONSTANT.ZERO,
60                     or_(
61                         and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
62                         and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
63                         not_(
64                             or_(
65                                 combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
66                                 combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
67                             )
68                         )
69                     ),
70                 and_(
71                     SeatBooking.from_date <= to_date,
72                     SeatBooking.from_date >= from_date,
73                     SeatBooking.is_night_shift == CONSTANT.ONE,
74                     or_(
75                         and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
76                         not_(
77                             or_(
78                                 combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
79                                 combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
80                             )
81                         )
82                     ),
83                     and_(
84                         combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
85                         combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
86                     )
87                 ),
88             ),
89             and_(
90                 SeatBooking.from_date <= from_date,
91                 SeatBooking.to_date >= from_date,
92                 SeatBooking.is_night_shift == CONSTANT.ZERO,
93                 or_(
94                     and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
95                     and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
96                     not_(
97                         or_(
98                             combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
99                             combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
100                         )
101                     )
102                 ),
103             ),
104             and_(
105                 SeatBooking.from_date <= from_date,
106                 SeatBooking.booking_end_date >= from_date,
107                 SeatBooking.is_night_shift == CONSTANT.ONE,
108                 or_(
109                     and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
110                     not_(
111                         or_(
112                             combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
113                             combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
114                         )
115                     ),
116                     and_(
117                         combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
118                         combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
119                     )
120                 )
121             )
122         )
123     )
124 )
125
126 if booking_end_date == from_date or str(is_custom_date_booking) == str(CONSTANT.ONE):
127     auto_cancelled_booking_ids = CancellationForUnusedBooking.get_cancelled_booking_list(
128         date=from_date, branch_id=branch_details.id, only_id=True
129     )
130     booking_obj = booking_obj.filter(SeatBooking.id.not_in(auto_cancelled_booking_ids))
131     custom_booking_seat_ids = CustomDateBooking.get_custom_booking_count(
132         from_date, booking_end_date, branch_id=branch_details.id, custom_dates=custom_dates, only_booking_seat_id=True
133     )
134     booking_obj = booking_obj.filter(SeatBooking.is_custom_date_booking == CONSTANT.ZERO)
135
136     seat_ids = db.session.scalars(booking_obj.with_entities(SeatBooking.seat_id)).all() + custom_booking_seat_ids
137
138 return seat_ids

```

## Method Definition:

```
1 @staticmethod
2 def get_booked_seat_ids(branch_details, slot_details, from_date, to_date, is_custom_date_booking, custom_dates):
3     """
4         Method to get booked seat IDs.
5
6         :param branch_details: Details of the branch.
7         :type branch_details: object
8         :param slot_details: Details of the slot.
9         :type slot_details: object
10        :param from_date: Starting date of the booking.
11        :type from_date: datetime
12        :param to_date: Ending date of the booking.
13        :type to_date: datetime
14        :param is_custom_date_booking: Flag to indicate custom date booking.
15        :type is_custom_date_booking: bool
16        :param custom_dates: List of custom dates.
17        :type custom_dates: list
18        :return: List of booked seat IDs.
19        :rtype: list
20    """
```

**Definition and Parameters:** This method is defined as a `staticmethod`, meaning it can be called on the class itself, not on an instance. The method aims to fetch booked seat IDs based on the parameters provided.

- `branch_details`: Details of the branch.
- `slot_details`: Details of the slot.
- `from_date` and `to_date`: Date range for the bookings.
- `is_custom_date_booking`: Boolean indicating if it's a custom date booking.
- `custom_dates`: List of custom dates.

## Importing and Initializing Variables:

```
1 from database.CancellationForUnusedBooking import CancellationForUnusedBooking
2 seat_ids = []
3 custom_booking_seat_ids = []
```

- **Imports:** Importing a class or function from another module, `CancellationForUnusedBooking`.
- **Initializing Variables:** Two empty lists `seat_ids` and `custom_booking_seat_ids` are initialized to store the IDs of the booked seats.

## Combining Date and Time:

```
1 combined_from_datetime = func.concat(
2     func.cast(SeatBooking.from_date, String), ' ',
3     func.cast(Slot.start_time, String)
4 )
5 combined_to_datetime = func.concat(
6     func.cast(SeatBooking.booking_end_date, String), ' ',
7     func.cast(Slot.end_time, String)
8 )
```

- **Combining Dates and Times:** Using SQLAlchemy functions to concatenate date and time fields into single datetime strings:
  - combined\_from\_datetime: Concatenates from\_date and start\_time.
  - combined\_to\_datetime: Concatenates booking\_end\_date and end\_time.

## Handling Slot Time Overlaps:



```
1 booking_end_date = to_date
2 if slot_details.start_time > slot_details.end_time:
3     to_date += timedelta(days=CONSTANT.ONE)
```

- **Slot Overlaps:** Adjusts the to\_date if the slot starts before and ends after midnight (i.e., start\_time is greater than end\_time).

## Building the Query



```
1 booking_obj = (
2     SeatBooking.query
3     .join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None))
4     .join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None))
5     .join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None))
6     .join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None))
7     .join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None))
8     .join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None))
9     .join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None))
10    .join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None))
11    .join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_id, FloorMap.deleted == CONSTANT.ZERO, FloorMap.deleted_at == None))
12    .filter(
13        or_(
14            and_(
15                SeatBooking.is_active == True,
16                SeatBooking.branch_id == branch_details.id,
17                SeatBooking.deleted == CONSTANT.ZERO,
18                SeatBooking.deleted_at == None,
19                or_(
20                    and_(
21                        SeatBooking.from_date <= to_date,
22                        SeatBooking.from_date >= from_date,
23                        SeatBooking.is_night_shift == CONSTANT.ZERO,
24                        or_(
25                            and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
26                            and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
27                            not_(
28                                or_(
29                                    combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
30                                    combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
31                                )
32                            )
33                        ),
34                        and_(
35                            and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
36                            not_(
37                                or_(
38                                    combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
39                                    combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
40                                )
41                            ),
42                            and_(
43                                and_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
44                                    combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
45                                )
46                            ),
47                            and_(
48                                and_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
49                                    combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
50                                )
51                            ),
52                            and_(
53                                and_(SeatBooking.from_date <= from_date,
54                                    SeatBooking.to_date >= from_date,
55                                    SeatBooking.is_night_shift == CONSTANT.ZERO,
56                                    or_(
57                                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
58                                        and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
59                                        not_(
60                                            or_(
61                                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
62                                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
63                                            )
64                                        )
65                                    ),
66                                    and_(
67                                        and_(SeatBooking.from_date <= from_date,
68                                            SeatBooking.booking_end_date >= from_date,
69                                            SeatBooking.is_night_shift == CONSTANT.ONE,
70                                            or_(
71                                                and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
72                                                not_(
73                                                    or_(
74                                                        combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
75                                                        combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
76                                                    )
77                                                ),
78                                                and_(
79                                                    and_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
80                                                        combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
81                                                    )
82                                                )
83                                            )
84                                        )
85                                    )
86                                )
87                            )
88                        )
89                    )
90                )
91            )
92        )
93    )
94 )
```

## This large query builds a complex SQLAlchemy query object:

- Joins multiple tables (Branch, Employee, DepartmentDetail, RoleDetail, Department, Role, Slot, Seat, FloorMap) with conditions ensuring they are not deleted.
- Filters SeatBooking records based on various conditions, including:
  - The booking must be active and within the specified date range.
  - The slot times must overlap with the provided slot details.

## Detailed Explanation of Filters

### 1. Basic Filters:

- `SeatBooking.is_active == True`: Only consider active bookings.
- `SeatBooking.branch_id == branch_details.id`: Filter bookings by the specified branch.
- `SeatBooking.deleted == CONSTANT.ZERO`: Exclude deleted bookings.
- `SeatBooking.deleted_at == None`: Ensure the `deleted_at` field is `None` (i.e., not marked as deleted).

### 2. Complex OR Conditions:

- The `or_` function wraps multiple conditions. The bookings that satisfy at least one of these conditions will be included.

### 3. First AND Condition (Day Shifts Within the Date Range):

- `SeatBooking.from_date <= to_date`: The booking start date should be on or before the `to_date`.
- `SeatBooking.from_date >= from_date`: The booking start date should be on or after the `from_date`.
- `SeatBooking.is_night_shift == CONSTANT.ZERO`: This condition is for day shifts.
- Nested `or_` for slot overlap:
  - `Slot.end_time >= slot_details.start_time and Slot.start_time <= slot_details.end_time`: The booking slot overlaps with the specified slot.
  - `Slot.end_time <= slot_details.start_time and Slot.start_time >= slot_details.end_time`: Handles cases where the slot times wrap around midnight.
  - `not_(or_(combined_from_datetime >= str(to_date)+' '+str(slot_details.end_time), combined_to_datetime <= str(from_date)+' '+str(slot_details.start_time)))`: Exclude slots that completely fall outside the specified date and time range.

### 4. Second AND Condition (Night Shifts Within the Date Range):

- Similar to the first condition but with `SeatBooking.is_night_shift == CONSTANT.ONE` and slightly different time overlap logic to accommodate overnight shifts.

### 5. Third AND Condition (Day Shifts Spanning the Date Range):

- `SeatBooking.from_date <= from_date and SeatBooking.to_date >= from_date`: The booking period spans the `from_date`.
- `SeatBooking.is_night_shift == CONSTANT.ZERO`: This condition is for day shifts.
- Similar nested `or_` logic for slot overlaps.

### 6. Fourth AND Condition (Night Shifts Spanning the Date Range):

- Similar to the third condition but for night shifts (`SeatBooking.is_night_shift == CONSTANT.ONE`).

## Special Handling for Custom Date Bookings

```
1 if booking_end_date == from_date or str(is_custom_date_booking) == str(CONSTANT.ONE):
2     auto_cancelled_booking_ids = CancellationForUnusedBooking.get_cancelled_booking_list(
3         date=from_date, branch_id=branch_details.id, only_id=True
4     )
5     booking_obj = booking_obj.filter(SeatBooking.id.not_in((auto_cancelled_booking_ids)))
6     custom_booking_seat_ids = CustomDateBooking.get_custom_booking_count(
7         from_date, booking_end_date, branch_id=branch_details.id, custom_dates=custom_dates, only_booking_seat_id=True
8     )
9     booking_obj = booking_obj.filter(SeatBooking.is_custom_date_booking == CONSTANT.ZERO)
```

- **Auto-cancelled Bookings:**
  - Fetch auto-canceled booking IDs for the given date and branch.
  - Exclude these IDs from the main query.
- **Custom Bookings:**
  - Retrieve custom booking seat IDs for the specified custom dates.
  - Exclude bookings marked as custom date bookings (`SeatBooking.is_custom_date_booking == CONSTANT.ZERO`).

## Final Execution

```
1 seat_ids = db.session.scalars(booking_obj.with_entities(SeatBooking.seat_id)).all() + custom_booking_seat_ids
2 return seat_ids
```

- Execute the query to fetch seat IDs from the database.
- Combine these seat IDs with custom booking seat IDs.
- Return the final list of booked seat IDs.

## 2.Query to get\_seat\_list\_mobile

```
1 @staticmethod
2 def get_seat_list_mobile( floor_id ):
3     """
4         Method to get seat_list_mobile
5         @param branch_id:
6         @param floor_id:
7         @type floor_id:
8         @return:
9         @rtype:
10        """
11    from database.SeatDepartment import SeatDepartment
12    from database.SeatRole import SeatRole
13
14    seat_obj = Seat.query. \
15        join(SeatDepartment, SeatDepartment.seat_id == Seat.id, isouter = True). \
16        join(SeatRole, SeatRole.seat_id == Seat.id, isouter = True). \
17        join(Department, and_(Department.id == SeatDepartment.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None), isouter = True). \
18        join(Role, and_(Role.id == SeatRole.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None), isouter = True). \
19        add_columns((func.group_concat(distinct(Role.role_id))).label('role_ids'), (func.group_concat(distinct(Department.department_id))).label('department_ids')). \
20        filter(Seat.image_id == floor_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None, Seat.is_meeting_room == CONSTANT.ZERO).group_by(Seat.id).all()
21
22    return seat_obj
```

## Method Definition and Docstring:

```
1 @staticmethod
2 def get_seat_list_mobile( floor_id ):
3     """
4         Method to get_seat_list_mobile
5         @param branch_id:
6         @param floor_id:
7         @type floor_id:
8         @return:
9         @rtype:
10        """
```

- **@staticmethod:** Indicates that this method does not modify the state of the class and does not require access to instance-specific data.
- **Parameters:**
- **floor\_id:** The ID of the floor for which the seat list is to be retrieved.
- **Return:** A list of seats with their associated department and role IDs.

## Building the Query:

```
1 seat_obj = Seat.query. \
2     join(SeatDepartment, SeatDepartment.seat_id == Seat.id, isouter = True). \
3     join(SeatRole, SeatRole.seat_id == Seat.id, isouter = True). \
4     join(Department, and_(Department.id == SeatDepartment.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None), isouter = True). \
5     join(Role, and_(Role.id == SeatRole.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None), isouter = True). \
6     add_columns((func.group_concat(distinct(Role.role_id))).label('role_ids'), (func.group_concat(distinct(Department.department_id))).label('department_ids')). \
7     filter(Seat.image_id == floor_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None, Seat.is_meeting_room == CONSTANT.ZERO).group_by(Seat.id).all()
8 return seat_obj
```

1. **Seat.query**: Starts building a query on the Seat table.
2. **join(SeatDepartment, SeatDepartment.seat\_id == Seat.id, isouter=True)**:
  - a. Performs a left outer join with the SeatDepartment table on the seat\_id field, linking SeatDepartment to Seat.
3. **join(SeatRole, SeatRole.seat\_id == Seat.id, isouter=True)**:
  - a. Performs a left outer join with the SeatRole table on the seat\_id field, linking SeatRole to Seat.
4. **join(Department, ... , isouter=True)**:
  - a. Performs a left outer join with the Department table using conditions:
    - i. Department.id matches SeatDepartment.department\_id.
    - ii. Department is not deleted (deleted == CONSTANT.ZERO and deleted\_at == None).
5. **join(Role, ... , isouter=True)**:
  - a. Performs a left outer join with the Role table using conditions:
    - i. Role.id matches SeatRole.role\_id.
    - ii. Role is not deleted (deleted == CONSTANT.ZERO and deleted\_at == None).
6. **add\_columns(func.group\_concat(distinct(Role.role\_id)).label('role\_ids'), (func.group\_concat(distinct(Department.department\_id)).label('department\_ids')))**:
  - a. Adds columns to the query results:
    - i. **role\_ids**: Concatenates distinct Role.role\_id values into a single string.
    - ii. **department\_ids**: Concatenates distinct Department.department\_id values into a single string.
7. **filter(...)**:
  - a. Filters the results to include only seats that:
    - i. Match the input floor\_id (Seat.image\_id == floor\_id).
    - ii. Are not deleted (Seat.deleted == CONSTANT.ZERO and Seat.deleted\_at == None).
    - iii. Are not meeting rooms (Seat.is\_meeting\_room == CONSTANT.ZERO).
8. **group\_by(Seat.id)**:
  - a. Groups the results by Seat.id.
9. **all()**:
  - a. Executes the query and retrieves all results.

### 3.get\_formatted\_seat\_list:

```
1 def get_formatted_seat_list(input_data, booked_seat_ids, floor, department_id, role_id, from_date, to_date, is_custom_date_booking, custom_dates, is_show_employee_details, slot):
2     from database.SeatBooking import SeatBooking
3     from database.Department import Department
4     from database.Role import Role
5     data = []
6     is_paginated = False
7     if input_data:
8         original_datas = input_data
9         if ('hasattr(input_data, "page") and hasattr(input_data, "items")) or (
10             'page' in input_data and 'items' in input_data):
11             original_datas = input_data.items
12             is_paginated = True
13     for each_data in original_datas:
14         dict_data = dict()
15         role_ids = each_data['role_ids'].split(',') if each_data['role_ids'] else []
16         department_ids = each_data['department_ids'].split(',') if each_data['department_ids'] else []
17         each_data = each_data[0]
18         dict_data['id'] = str(each_data.id)
19         dict_data['name'] = str(each_data.name)
20         dict_data['htmlId'] = str(each_data.htmlId)
21         dict_data['htmlWidth'] = str(each_data.htmlWidth)
22         dict_data['htmlHeight'] = str(each_data.htmlHeight)
23         dict_data['htmlLeft'] = str(each_data.htmlLeft)
24         dict_data['htmlTop'] = str(each_data.htmlTop)
25         dict_data['initialId'] = str(each_data.initialId)
26         dict_data['image'] = floor.name
27         dict_data['booked_employee_id'] = ''
28         dict_data['booked_employee_name'] = ''
29         dict_data['booked_employee_role'] = ''
30         dict_data['booked_employee_department'] = ''
31         print(each_data.is_common_seat)
32         if is_show_employee_details == str(CONSTANT.ONE):
33             booking_data = SeatBooking.get_booked_employee_data(floor.branch, slot, from_date, to_date, is_custom_date_booking, custom_dates, each_data.id)
34             if booking_data:
35                 dict_data['booked_employee_id'] = str(booking_data.employee.emp_id)
36                 dict_data['booked_employee_name'] = booking_data.employee.emp_name
37                 dict_data['booked_employee_role'] = booking_data.employee.role_id
38                 dict_data['booked_employee_department'] = booking_data.employee.department_id
39                 dict_data['booked_employee_role'] = role.name if role else ''
40                 dict_data['booked_employee_department'] = department.name if department else ''
41             dict_data['is_available'] = str(CONSTANT.ZERO) if (each_data.id in booked_seat_ids or role_id not in role_ids or department_id not in department_ids) else str(CONSTANT.ONE)
42             dict_data['is_secure'] = str(CONSTANT.ONE) if each_data.id in booked_seat_ids else str(CONSTANT.ZERO)
43             dict_data['is_role_available'] = str(CONSTANT.ONE) if role_id in role_ids else str(CONSTANT.ZERO)
44             dict_data['is_dept_available'] = str(CONSTANT.ONE) if department_id in department_ids else str(CONSTANT.ZERO)
45             dict_data['is_accepted'] = str(CONSTANT.ZERO) if each_data.id not in booked_seat_ids else str(CONSTANT.ONE) if SeatBooking.has_bookings(floor.branch_id, seat_id=each_data.id, is_accepted=True) == str(CONSTANT.ZERO) else str(CONSTANT.ONE)
46             dict_data['seat_color'] = CustomFormatter.get_seat_color(floor.branch.company, dict_data['is_available']) and floor.branch.company.spacesManagementConfiguration else str(CONSTANT.ZERO)
47             dict_data['is_accepted'] = str(CONSTANT.ZERO) if each_data.id not in booked_seat_ids else str(CONSTANT.ONE) if SeatBooking.has_bookings(floor.branch_id, seat_id=each_data.id, is_accepted=True) == str(CONSTANT.ZERO) else str(CONSTANT.ONE)
48             data.append(dict_data)
49     if is_paginated:
50         return Util.paginated_format(data, input_data)
51     return data
```

#### Method Definition and Imports:

```
1 @staticmethod
2 def get_formatted_seat_list(input_data, booked_seat_ids, floor, department_id, role_id, from_date, to_date, is_custom_date_booking, custom_dates, is_show_employee_details, slot):
3     from database.SeatBooking import SeatBooking
4     from database.Department import Department
5     from database.Role import Role
```

- Method Definition:** This is a static method, which means it can be called on the class itself rather than an instance of the class.
- Imports:** The method imports the **SeatBooking**, **Department**, and **Role** classes from the database module.

#### Initial Variables:

```
1 data = []
2 is_paginated = False
```

- data:** An empty list to store the formatted seat data.
- is\_paginated:** A flag to check if the input data is paginated.

#### Check for Pagination:

```
1 if input_data:
2     original_datas = input_data
3     if ('hasattr(input_data, "page") and hasattr(input_data, "items")) or ('page' in input_data and 'items' in input_data):
4         original_datas = input_data.items
5         is_paginated = True
```

- if input\_data:** Check if input\_data is not empty.
- original\_datas = input\_data:** Initialize original\_datas with input\_data.
- Pagination Check:** If input\_data has page and items attributes (or keys), set original\_datas to input\_data.items and mark is\_paginated as True.

## Loop through Data and Extract Seat Information:

```
1 for each_data in original_datas:
2     dict_data = dict()
3     role_ids = each_data['role_ids'].split(',') if each_data['role_ids'] else []
4     department_ids = each_data['department_ids'].split(',') if each_data['department_ids'] else []
5     each_data = each_data[0]
```

- **Loop:** Iterate through each item in original\_datas.
- **dict\_data:** Initialize an empty dictionary to store seat information.
- **role\_ids:** Split role\_ids string by commas if it exists, otherwise set it to an empty list.
- **department\_ids:** Split department\_ids string by commas if it exists, otherwise set it to an empty list.
- **each\_data:** Access the first element of each\_data.

## Populate dict\_data with Seat Attributes:

```
1 dict_data['id'] = str(each_data.id)
2 dict_data['name'] = str(each_data.name)
3 dict_data['htmlId'] = each_data.htmlId
4 dict_data['htmlWidth'] = str(each_data.htmlWidth)
5 dict_data['htmlHeight'] = str(each_data.htmlHeight)
6 dict_data['htmlLeft'] = str(each_data.htmlLeft)
7 dict_data['htmlTop'] = str(each_data.htmlTop)
8 dict_data['intHtmlId'] = str(each_data.intHtmlId)
9 dict_data['image'] = floor.name
10 dict_data['booked_employee_id'] = ''
11 dict_data['booked_employee_name'] = ''
12 dict_data['booked_employee_role'] = ''
13 dict_data['booked_employee_department'] = ''
```

- **Set Seat Attributes:** Populate dict\_data with the seat's attributes, converting them to strings where necessary.
- **Employee Booking Details:** Initialize booking-related fields as empty strings.

## Debug Statements and Conditional Check for Employee Details:

```
1 if is_show_employee_details == str(CONSTANT.ONE):
2     booking_data = SeatBooking.get_booked_employee_data(floor.branch, slot, from_date, to_date, is_custom_date_booking, custom_dates, each_data.id)
3     if booking_data:
4         dict_data['booked_employee_id'] = str(booking_data.employee.e_id)
5         dict_data['booked_employee_name'] = booking_data.employee.emp_name
6         role = Role.get_role_data(seque_role_id = booking_data.employee.role_id)
7         department = Department.get_department_data(seque_department_id = booking_data.employee.department_id)
8         dict_data['booked_employee_role'] = role.name if role else ''
9         dict_data['booked_employee_department'] = department.name if department else ''
```

- **Debug Statements:** Print statements for debugging purposes.
- **Check for Employee Details:** If is\_show\_employee\_details equals 1 (constant), fetch booking data for the seat.
- **Populate Employee Details:** If booking data exists, populate the employee's ID, name, role, and department in dict\_data.

#### **Availability and Booking Status Checks:**

- **Availability:** Set `is_available` to 1 if the seat is not booked and is a common seat. Otherwise, set it to 0 if the seat is booked or the role/department does not match. Else, set it to 1.
  - **Booking Status:** Set `is_booked` to 1 if the seat is in `booked_seat_ids`, otherwise 0.
  - **Role Availability:** Set `is_role_available` to 1 if `role_id` is in `role_ids`, otherwise 0.
  - **Department Availability:** Set `is_dept_available` to 1 if `department_id` is in `department_ids`, otherwise 0.

## **Seat Color and Acceptance Status:**

- **Selected Seat Color:** Set the selected seat color based on the company's space management configuration.
  - **Acceptance Status:** Set `is_accepted` to 0 if the seat is not booked. Otherwise, set it to 1 if the seat has accepted bookings, otherwise 0.
  - **Seat Color:** Determine the seat color using `CustomFormatter.get_seat_color` based on availability, booking, and acceptance status.

#### **4.Query to get\_booked\_employee\_data:**

## Combined Date-Time Strings:

```
1 combined_from_datetime = func.concat(
2     func.cast(SeatBooking.from_date, String), ' ',
3     func.cast(Slot.start_time, String)
4 )
5 combined_to_datetime = func.concat(
6     func.cast(SeatBooking.booking_end_date, String), ' '
7     func.cast(Slot.end_time, String)
8 )
```

**combined\_from\_datetime** and **combined\_to\_datetime**: Create combined date-time strings by concatenating the booking dates and slot times. These are used to compare date and time together in the filters.

## Adjust Date if Slot Crosses Midnight:

```
1 booking_end_date = to_date
2 if slot_details.start_time > slot_details.end_time:
3     to_date = to_date + timedelta(days=CONSTANT.ONE)
```

- **booking\_end\_date**: Initialize booking\_end\_date to to\_date.
- **Adjust to\_date**: If the slot crosses midnight (i.e., the start time is later than the end time), increment to\_date by one day to ensure correct date range handling.

## Query Setup:

```
1 bookingObj = SeatBooking.query. \
2     join/Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None). \
3     joinEmployee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None). \
4     join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None)). \
5     join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None)). \
6     join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None)). \
7     join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None)). \
8     join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None)). \
9     join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None)). \
10    join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_id, FloorMap.deleted == CONSTANT.ZERO, FloorMap.deleted_at == None))
```

- **Query Construction**: Build a query using SQLAlchemy's ORM syntax.
- **Joins**: Join multiple tables (Branch, Employee, DepartmentDetail, RoleDetail, Department, Role, Slot, Seat, FloorMap) to the SeatBooking table.
- **Conditions in Joins**: Ensure that the joined entities are not deleted and are active.

## Filters Explain:

### 1. Basic Filters

```
1 .filter(
2     Seat.id == seat_id,
3     SeatBooking.is_active == True,
4     SeatBooking.deleted == CONSTANT.ZERO,
5     SeatBooking.deleted_at == None,
```

- **Seat.id == seat\_id**: This ensures that the query is filtered for the specific seat ID provided.
- **SeatBooking.is\_active == True**: This filter ensures that only active bookings are considered.
- **SeatBooking.deleted == CONSTANT.ZERO**: This ensures that the bookings are not marked as deleted.
- **SeatBooking.deleted\_at == None**: This ensures that the bookings are not marked as deleted (using a timestamp).

### 2. Complex Filters for Overlapping Bookings:

#### a. Day Shift Booking Starting Within Date Range

```
1 or_(
2     and_(
3         SeatBooking.from_date <= to_date,
4         SeatBooking.from_date >= from_date,
5         SeatBooking.is_night_shift == CONSTANT.ZERO,
6         or_(
7             and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
8             and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
9             not_(or_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)))
10        ),
11    ),
12)
```

- **SeatBooking.from\_date >= from\_date**: The booking starts on or after the start date.
- **SeatBooking.from\_date <= to\_date**: The booking starts before or on the end date.
- **SeatBooking.is\_night\_shift == CONSTANT.ZERO**: This condition ensures it is a day shift booking.
- **Time Overlap Conditions (or\_)**:
  - **and\_(Slot.end\_time >= slot\_details.start\_time, Slot.start\_time <= slot\_details.end\_time)**: Slot times overlap directly.
  - **and\_(Slot.end\_time <= slot\_details.start\_time, Slot.start\_time >= slot\_details.end\_time)**: Slot times overlap inversely.
  - **not\_(or\_(combined\_from\_datetime >= str(to\_date) + '' + str(slot\_details.end\_time), combined\_to\_datetime <= str(from\_date) + '' + str(slot\_details.start\_time)))**: Ensure the combined date-time ranges do not completely exclude each other.

#### b. Night Shift Booking Starting Within Date Range

```

● ● ●
1 and_(
2   SeatBooking.from_date <= to_date,
3   SeatBooking.from_date >= from_date,
4   SeatBooking.is_night_shift == CONSTANT.ONE,
5   or_(
6     and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
7     not_(or_(combined_from_datetime >= str(to_date) + '' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + '' + str(slot_details.start_time))),
8     and_(combined_from_datetime >= str(to_date) + '' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + '' + str(slot_details.start_time))
9   )
10 ),

```

- **SeatBooking.from\_date <= to\_date**: The booking starts before or on the end date.
- **SeatBooking.from\_date >= from\_date**: The booking starts on or after the start date.
- **SeatBooking.is\_night\_shift == CONSTANT.ONE**: This condition ensures it is a night shift booking.
- **Time Overlap Conditions (or\_)**:
  - **and\_(Slot.end\_time >= slot\_details.start\_time, Slot.start\_time <= slot\_details.end\_time)**: Slot times overlap directly.
  - **not\_(or\_(combined\_from\_datetime >= str(to\_date) + '' + str(slot\_details.end\_time), combined\_to\_datetime <= str(from\_date) + '' + str(slot\_details.start\_time)))**: Ensure the combined date-time ranges do not completely exclude each other.
  - **and\_(combined\_from\_datetime >= str(to\_date) + '' + str(slot\_details.end\_time), combined\_to\_datetime <= str(from\_date) + '' + str(slot\_details.start\_time))**: Additional condition for specific overlap handling.

#### c. Day Shift Booking Covering Entire Date Range

```

● ● ●
1 and_(
2   SeatBooking.from_date <= from_date,
3   SeatBooking.to_date >= from_date,
4   SeatBooking.is_night_shift == CONSTANT.ZERO,
5   or_(
6     and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
7     and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
8     not_(or_(combined_from_datetime >= str(to_date) + '' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + '' + str(slot_details.start_time)))
9   )
10 ),

```

- **SeatBooking.from\_date <= from\_date**: The booking starts on or before the start date.
- **SeatBooking.to\_date >= from\_date**: The booking ends on or after the start date.
- **SeatBooking.is\_night\_shift == CONSTANT.ZERO**: This condition ensures it is a day shift booking.
- **Time Overlap Conditions (or\_)**:
  - **and\_(Slot.end\_time >= slot\_details.start\_time, Slot.start\_time <= slot\_details.end\_time)**: Slot times overlap directly.

- **and\_(Slot.end\_time <= slot\_details.start\_time, Slot.start\_time >= slot\_details.end\_time):** Slot times overlap inversely.
- **not\_(or\_(combined\_from\_datetime >= str(to\_date) + '' + str(slot\_details.end\_time), combined\_to\_datetime <= str(from\_date) + '' + str(slot\_details.start\_time)))**: Ensure the combined date-time ranges do not completely exclude each other.

#### d. Night Shift Booking Covering Entire Date Range

```

1 and (
2     SeatBooking.from_date <= from_date,
3     SeatBooking.booking_end_date >= from_date,
4     SeatBooking.is_night_shift == CONSTANT.ONE,
5     or_(
6         and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
7         not_(or_(combined_from_datetime >= str(to_date) + '' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + '' + str(slot_details.start_time))),
8         and_(combined_from_datetime >= str(to_date) + '' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + '' + str(slot_details.start_time))
9     )
10 )
11

```

- **SeatBooking.from\_date <= from\_date:** The booking starts on or before the start date.
- **SeatBooking.booking\_end\_date >= from\_date:** The booking ends on or after the start date.
- **SeatBooking.is\_night\_shift == CONSTANT.ONE:** This condition ensures it is a night shift booking.
- **Time Overlap Conditions (or\_):**
  - **and\_(Slot.end\_time >= slot\_details.start\_time, Slot.start\_time <= slot\_details.end\_time):** Slot times overlap directly.
  - **not\_(or\_(combined\_from\_datetime >= str(to\_date) + '' + str(slot\_details.end\_time), combined\_to\_datetime <= str(from\_date) + '' + str(slot\_details.start\_time)))**: Ensure the combined date-time ranges do not completely exclude each other.
  - **and\_(combined\_from\_datetime >= str(to\_date) + '' + str(slot\_details.end\_time), combined\_to\_datetime <= str(from\_date) + '' + str(slot\_details.start\_time))**: Additional condition for specific overlap handling.

### Handling Custom Date Bookings:

```

1 if to_date == from_date or str(is_custom_date_booking) == str(CONSTANT.ONE):
2     if not bookingObj.filter(SeatBooking.is_custom_date_booking == CONSTANT.ZERO).first():
3         custom_booking_ids = CustomDateBooking.get_custom_booking_count(from_date, to_date, branch_id=branch_details.id, custom_dates=custom_dates, only_booking_id=True)
4         bookingObj = bookingObj.filter(SeatBooking.id.in_(custom_booking_ids), SeatBooking.is_custom_date_booking == CONSTANT.ONE)

```

- **Check for Single Day or Custom Date Booking:** If to\_date equals from\_date or is\_custom\_date\_booking is True:
  - **Regular Booking Check:** If no regular bookings are found for the seat (is\_custom\_date\_booking == CONSTANT.ZERO):
    - **Retrieve Custom Bookings:** Get the custom booking IDs for the specified date range.
    - **Filter by Custom Bookings:** Update the query to include only these custom booking IDs and ensure they are marked as custom date bookings (is\_custom\_date\_booking == CONSTANT.ONE).

### Final Query Execution:

```

return bookingObj.group_by(SeatBooking.id).order_by(SeatBooking.from_date.desc()).first()

```

- **Grouping:** Group the results by SeatBooking.id.
- **Ordering:** Order the results by the start date (from\_date) in descending order.
- **Return:** Return the first result of the query, which is the most recent booking matching the criteria.

## 4. Mobile book seat api queries:

### 1.Query to check\_booking\_is\_possible

```
 1  from database import check_booking_is_possible
 2
 3  class BookSeatAPI:
 4      def check_booking_is_possible(self, employee_details, slot, seat, from_date, to_date, is_custom_date_booking, custom_dates, is_oracle_based_company, branch):
 5          # Implementation of the check_booking_is_possible method
 6
 7  if __name__ == "__main__":
 8      # Test cases for the check_booking_is_possible method
 9
10  
```

## Method Definition and Parameters

```
 1  def check_booking_is_possible(employee_details, slot, seat, from_date, to_date, is_custom_date_booking, custom_dates, is_oracle_based_company, branch):
 2      """
 3          Method to check_booking_is_possible
 4      """
 5
 6  if __name__ == "__main__":
 7      # Test cases for the check_booking_is_possible method
 8
 9  
```

This method is defined as a static method, meaning it can be called on the class itself, not on instances of the class. It takes several parameters related to booking details, including employee information, slot times, seat, date range, custom booking flags, company type, and branch.

## Import and Initial Variables

```
 1  from database.CancellationForUnusedBooking import CancellationForUnusedBooking
 2  custom_booking_count = 0
 3  booking_end_date = to_date
 4
 5
 6  
```

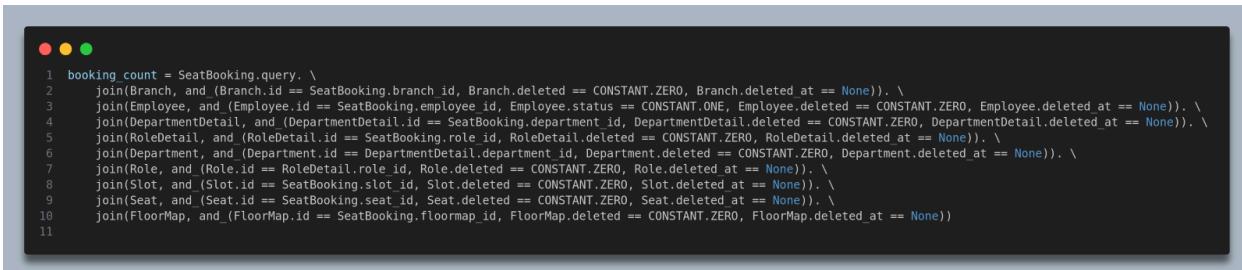
- Import:** Importing a module to handle cancellations of unused bookings.
- custom\_booking\_count:** Initialize custom booking count to zero.
- booking\_end\_date:** Set the booking end date to to\_date.

## Adjust Date if Slot Crosses Midnight

```
 1  if slot.start_time > slot.end_time:
 2      to_date = to_date + timedelta(days=CONSTANT.ONE)
 3
 4
 5  
```

- If the slot's start time is later than its end time (e.g., a night shift that crosses midnight), extend to\_date by one day.

## Query Setup

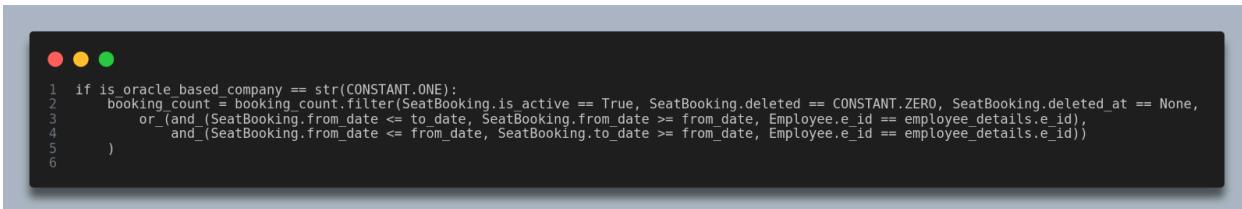


```
1 booking_count = SeatBooking.query. \
2 join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None)). \
3 join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None)). \
4 join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None)). \
5 join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None)). \
6 join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None)). \
7 join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None)). \
8 join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None)). \
9 join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None)). \
10 join(FloorMap, and_(FloorMap.id == SeatBooking.floor_map_id, FloorMap.deleted == CONSTANT.ZERO, FloorMap.deleted_at == None))
```

This chunk constructs a base query joining multiple tables to gather all necessary booking data. The joins ensure that:

- The branch is active and not deleted.
- The employee is active and not deleted.
- The department and role details are active and not deleted.
- The department and role themselves are active and not deleted.
- The slot and seat are active and not deleted.
- The floor map is active and not deleted.

## Oracle-Based Company Filter



```
1 if is oracle based company == str(CONSTANT.ONE):
2     booking_count = booking_count.filter(SeatBooking.is_active == True, SeatBooking.deleted == CONSTANT.ZERO, SeatBooking.deleted_at == None,
3                                         or_(and_(SeatBooking.from_date <= to_date, SeatBooking.from_date >= from_date, Employee.e_id == employee_details.e_id),
4                                             and_(SeatBooking.from_date <= from_date, SeatBooking.to_date >= from_date, Employee.e_id == employee_details.e_id)))
5
6
```

If the company is Oracle-based, the query is filtered to include only active, non-deleted bookings where:

- The booking starts within the date range and is made by the employee.
- The booking overlaps with the date range and is made by the employee.

## Non-Oracle Company Filter



```
1 from datetime import datetime, timedelta
2 from typing import Any, Dict, List, Optional, Union
3 from enum import IntEnum
4 from pydantic import BaseModel, Field, validator
5 from fastapi import APIRouter, Depends, HTTPException, Query
6 from sqlalchemy import func, and_, or_
7 from sqlalchemy.orm import Session
8 from sqlalchemy.exc import IntegrityError
9
10 from app.core import FastAPI, ResponseModel, get_db
11
12 router = APIRouter()
13
14 @router.get("/bookings/filter")
15 def filter_bookings(
16     action: str = "true",
17     booking_start_time: Optional[datetime] = None,
18     booking_end_time: Optional[datetime] = None,
19     seat_start_time: Optional[datetime] = None,
20     seat_end_time: Optional[datetime] = None,
21     slot_start_time: Optional[datetime] = None,
22     slot_end_time: Optional[datetime] = None,
23     employee_id: Optional[int] = None,
24     department_id: Optional[int] = None,
25     role_id: Optional[int] = None,
26     floor_map_id: Optional[int] = None,
27     seat_id: Optional[int] = None,
28     slot_id: Optional[int] = None,
29     start_time: Optional[datetime] = None,
30     end_time: Optional[datetime] = None,
31     is_night_shift: bool = False,
32     is_active: bool = True,
33     deleted: bool = False,
34     deleted_at: Optional[datetime] = None,
35     db: Session = Depends(get_db),
36     user_id: int = Depends(FastAPI.current_user),
37     current_time: datetime = Field(None, alias="now"),
38     offset: int = 0,
39     limit: int = 100,
40     sort_by_start_time: bool = False,
41     sort_by_end_time: bool = False,
42     sort_by_is_night_shift: bool = False,
43     sort_by_is_active: bool = False,
44     sort_by_deleted: bool = False,
45     sort_by_deleted_at: bool = False
46 ) -> ResponseModel:
47     try:
48         bookings = db.query(SeatBooking).filter(
49             and_(
50                 BookingFilter.booking_action == action,
51                 BookingFilter.booking_start_time == booking_start_time,
52                 BookingFilter.booking_end_time == booking_end_time,
53                 BookingFilter.seat_start_time == seat_start_time,
54                 BookingFilter.seat_end_time == seat_end_time,
55                 BookingFilter.slot_start_time == slot_start_time,
56                 BookingFilter.slot_end_time == slot_end_time,
57                 and_(or_(
58                     and_(Employee.e_id == employee_id, Employee.is_active == is_active, Employee.deleted == deleted),
59                     and_(DepartmentDetail.department_id == department_id, DepartmentDetail.is_active == is_active, DepartmentDetail.deleted == deleted),
60                     and_(RoleDetail.role_id == role_id, RoleDetail.is_active == is_active, RoleDetail.deleted == deleted)
61                 ), and_(FloorMap.id == floor_map_id, FloorMap.is_active == is_active, FloorMap.deleted == deleted)),
62                 and_(Seat.id == seat_id, Seat.is_active == is_active, Seat.deleted == deleted),
63                 and_(Slot.id == slot_id, Slot.is_active == is_active, Slot.deleted == deleted),
64                 and_(BookingFilter.start_time == start_time, BookingFilter.end_time == end_time, BookingFilter.is_night_shift == is_night_shift),
65                 and_(BookingFilter.is_active == is_active, BookingFilter.deleted == deleted, BookingFilter.deleted_at == deleted_at)
66             )
67         ).order_by(
68             func.ifnull(BookingFilter.start_time, current_time) if sort_by_start_time else None,
69             func.ifnull(BookingFilter.end_time, current_time) if sort_by_end_time else None,
70             func.ifnull(BookingFilter.is_night_shift, False) if sort_by_is_night_shift else None,
71             func.ifnull(BookingFilter.is_active, True) if sort_by_is_active else None,
72             func.ifnull(BookingFilter.deleted, False) if sort_by_deleted else None,
73             func.ifnull(BookingFilter.deleted_at, None) if sort_by_deleted_at else None
74         ).offset(offset).limit(limit).all()
75         return ResponseModel(bookings=bookings, total=len(bookings))
76     except IntegrityError:
77         raise HTTPException(status_code=400, detail="Integrity error occurred during booking filtering")
78     except Exception as e:
79         raise HTTPException(status_code=500, detail=f"An unexpected error occurred: {str(e)}")
80
81
82 class BookingFilter(BaseModel):
83     booking_action: str
84     booking_start_time: Optional[datetime]
85     booking_end_time: Optional[datetime]
86     seat_start_time: Optional[datetime]
87     seat_end_time: Optional[datetime]
88     slot_start_time: Optional[datetime]
89     slot_end_time: Optional[datetime]
90     employee_id: Optional[int]
91     department_id: Optional[int]
92     role_id: Optional[int]
93     floor_map_id: Optional[int]
94     seat_id: Optional[int]
95     slot_id: Optional[int]
96     start_time: Optional[datetime]
97     end_time: Optional[datetime]
98     is_night_shift: bool
99     is_active: bool
100    deleted: bool
101    deleted_at: Optional[datetime]
```

For non-Oracle-based companies, the query is further filtered with more complex conditions, accounting for time overlaps, seat and slot matches, and night shifts. Here's a breakdown of the filter conditions:

- The booking overlaps with the date range and matches the seat and slot.
- The booking overlaps with the date range and is made by the employee for the slot.
- The booking overlaps with the date range and is made by the employee.
- The booking overlaps with the date range, matches the seat, is not a night shift, and the slot overlaps in time.
- The booking overlaps with the date range, matches the seat, is a night shift, and the slot overlaps in time.
- The booking starts before the date range, ends after the date range starts, matches the seat, is not a night shift, and the slot overlaps in time.

- The booking starts before the date range, ends after the date range starts, matches the seat, is a night shift, and the slot overlaps in time.

## Custom Date Booking and Auto-Cancellation Handling

```
1 if booking_end_date == from_date or str(is_custom_date_booking) == '1':
2     auto_cancelled_booking_ids = CancellationForUnusedBooking.get_cancelled_booking_list(date=from_date, branch_id=branch.id, only_id=True)
3     booking_count = booking_count.filter(~SeatBooking.id.in_(auto_cancelled_booking_ids))
4     custom_booking_count = SeatBooking.get_custom_booking_count(employee_details.id, slot, seat, from_date, booking_end_date, is_custom_date_booking, custom_dates, is_oracle_based_company)
5     booking_count = booking_count.filter(~SeatBooking.is_custom_date_booking == CONSTANT.ZERO)
6
```

If the booking end date equals the start date or it's a custom date booking:

- Fetch and exclude auto-cancelled bookings.
- Get a custom booking count for the specified details.
- Exclude custom date bookings from the count.

## Final Booking Count and Return Value

```
1 booking_count = booking_count.count() + custom_booking_count
2 return False if booking_count > CONSTANT.ZERO else True
```

- Finally, count the bookings and add the custom booking count. Return False if any bookings exist (indicating a conflict), otherwise return True.

