SeatBooking KT Document

Server Url:

Dev URL: https://sequredevsaas.sequre.app/space-management/

Test URL: https://sequretestsaas.sequre.app/space-management/

Stage URL: https://sequrestagesaas.sequre.app/space-management/

UAT URL: https://sequreuatsaas.sequre.app/space-management/

Live URL: https://booking-management.sequre.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/Sequre-SpaceManagement/job/Sequre-SpaceManagement-MSS-Backend/job/develop/

Test Build:
https://deployment.heptagon.tech/view/Sequre-SpaceManagement/job/Sequre-SpaceManagement-MSS-Backend/job/test/

Stage Build:
https://deployment.heptagon.tech/view/Sequre-SpaceManagement/job/Sequre-SpaceManagement-MSS-Backend/job/stage/

UAT Build:
https://deployment.heptagon.tech/view/Sequre-SpaceManagement/job/Sequre-SpaceManagement-MSS-Backend/job/uat/

Live Build:
https://deployment.heptagon.tech/view/Sequre-SpaceManagement/job/Sequre-SpaceManagement-MSS-Backend/job/live/

Image Pull Build:

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

Live:
https://deployment.heptagon.tech/view/Sequre-SpaceManagement/job/Sequre-SpaceManagement-Live-Image-Pull/

Docker Details:

Docker Bash Shell Details:

Test/Dev/Stage/UAT:
Host: 152.67.13.206
Cmd: docker exec -it Sequre-SM-BK-(server name) bash

Live:
Host: 65.1.31.56
Cmd: docker exec -it Sequre-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 SEQURE 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.

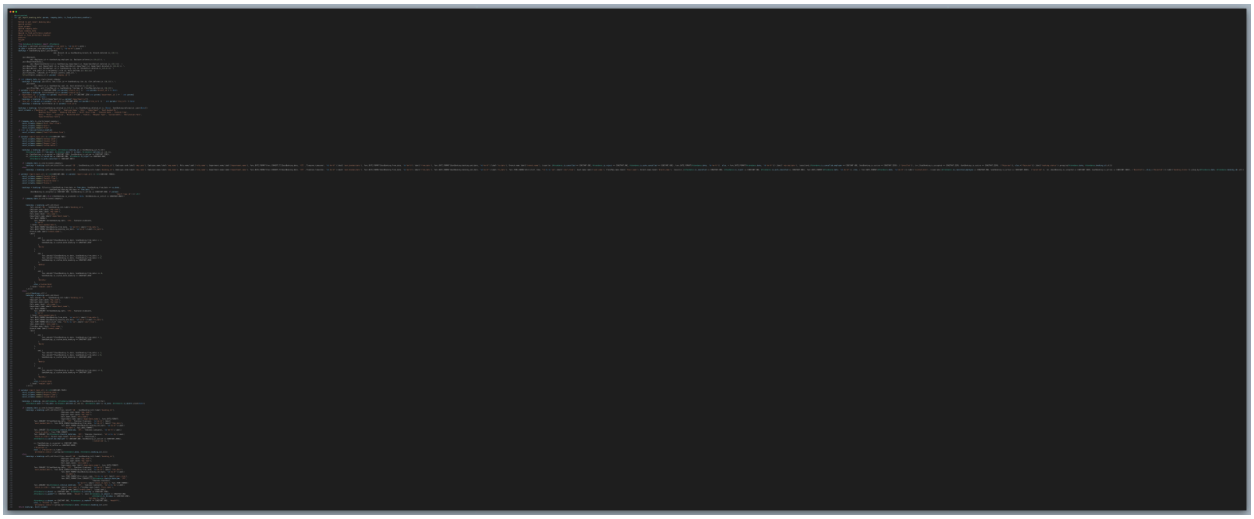
Elements in the Location Management Menu

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:



Query Breakdown

1. **Imports and Date Conversion**:

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

2. **Initial Query Setup**:

```
bookings = SeatBooking.query.join(Branch,
                                  and_(Branch.id == SeatBooking.branch_id, Branch.deleted.in_((0,1)),
                                       )). \
           join(Employee,
                and_(Employee.id == SeatBooking.employee_id, Employee.deleted.in_((0,1)))). \
           join(DepartmentDetail,
                and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted.in_((0,1)))). \
           join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted.in_((0,1)))). \
           join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted.in_((0,1)))). \
           join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted.in_((0,1)))). \
           join(Timezone, Timezone.id == Branch.country_code_id). \
           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**:

```
if not company_data.is_oracle_based_company:
    bookings = bookings.join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted.in_((0,1)))). \
        join(Seat,
                and_(Seat.id == SeatBooking.seat_id, Seat.deleted.in_((0,1)))). \
        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**:

```
if params['branch_id'] != CONSTANT.ZERO and params['branch_id'] != '' and params['branch_id'] != None:
            bookings = bookings.filter(Branch.id == params['branch_id'])
if 'department_id' in params and params['department_id'] != CONSTANT.ZERO and params['department_id'] != '' and params[
    'department_id'] != None:
    bookings = bookings.filter(Department.id == params['department_id'])
if 'role_id' in params and params['role_id'] != CONSTANT.ZERO and params['role_id'] != '' and params['role_id'] != None:
    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**:

```
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**:

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

• Initial list of columns for the Excel report.

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

```
if (company_data.is_oracle_based_company):
    excel_columns.remove('Shift Start Time')
    excel_columns.remove('Seat')
    excel_columns.remove('Floor')
if (not is_food_perference_enabled):
    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):

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):

```
if params['report_type_id'] == str(CONSTANT.ONE) or params['report_type_id'] == str(CONSTANT.THREE):
            excel_columns.remove('CheckIn Date')
            excel_columns.remove('CheckIn Time')
            excel_columns.remove('Rejected Date')
            excel_columns.remove('Status')

            bookings = bookings.filter(or_(SeatBooking.from_date <= from_date, SeatBooking.from_date <= to_date),
                                       SeatBooking.booking_end_date >= from_date, *(
                    [SeatBooking.is_accepted == CONSTANT.ONE, SeatBooking.is_active == CONSTANT.ONE] if params[
                                                                                                           'report_type_id'] == str(
                        CONSTANT.ONE) else [SeatBooking.is_accepted == None, SeatBooking.is_active == CONSTANT.ONE]))
            if (company_data.is_oracle_based_company):


                bookings = bookings.with_entities(
                    func.concat('SB-', SeatBooking.id).label('booking_id'),
                    Employee.code.label('emp_code'),
                    Employee.name.label('emp_name'),
                    Role.name.label('role_name'),
                    Department.name.label('department_name'),
                    func.DATE_FORMAT(
                        func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone),
                        '%d-%m-%Y'
                    ).label('seat_booked_date'),
                    func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
                    func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label('to_date'),
                    Branch.name.label('branch_name'),
                    case(
                        (
                            and_(
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 1,
                                SeatBooking.is_custom_date_booking == CONSTANT.ZERO
                            ),
                            'Daily'
                        ),
                        (
                            and_(
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) > 1,
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 8,
                                SeatBooking.is_custom_date_booking == CONSTANT.ZERO
                            ),
                            'Weekly'
                        ),
                        (
                            and_(
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) >= 8,
                                SeatBooking.is_custom_date_booking == CONSTANT.ZERO
                            ),
                            'Monthly'
                        ),
                        else_='Custom Date'
                    ).label('request_type')
                ).all()
            else:
                print(bookings.all())
                bookings = bookings.with_entities(
                    func.concat('SB-', SeatBooking.id).label('booking_id'),
                    Employee.code.label('emp_code'),
                    Employee.name.label('emp_name'),
                    Role.name.label('role_name'),
                    Department.name.label('department_name'),
                    func.DATE_FORMAT(
                        func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone),
                        '%d-%m-%Y'
                    ).label('seat_booked_date'),
                    func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
                    func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label('to_date'),
                    func.TIME_FORMAT(Slot.start_time, "%I:%i:%s %p").label('start_time'),
                    Seat.name.label('seat_name'),
                    FloorMap.name.label('floor_name'),
                    Branch.name.label('branch_name'),
                    case(
                        (
                            and_(
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 1,
                                SeatBooking.is_custom_date_booking == CONSTANT.ZERO
                            ),
                            'Daily'
                        ),
                        (
                            and_(
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) > 1,
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) < 8,
                                SeatBooking.is_custom_date_booking == CONSTANT.ZERO
                            ),
                            'Weekly'
                        ),
                        (
                            and_(
                                func.datediff(SeatBooking.to_date, SeatBooking.from_date) >= 8,
                                SeatBooking.is_custom_date_booking == CONSTANT.ZERO
                            ),
                            'Monthly'
                        ),
                        else_='Custom Date'
                    ).label('request_type')
                ).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):

```
if params['report_type_id'] == str(CONSTANT.FOUR):
            excel_columns.remove('Rejected Date')
            excel_columns.remove('Request Type')
            excel_columns.remove('Custom Dates')

            bookings = bookings.join(Attendance, Attendance.booking_id == SeatBooking.id).filter(
                Attendance.date >= from_date, Attendance.deleted.in_((0,1)), Attendance.date <= to_date, Attendance.is_absent.isnot(None))

            if (company_data.is_oracle_based_company):
                bookings = bookings.with_entities((func.concat('SB-', SeatBooking.id)).label('booking_id'),
                                                  Employee.code.label('emp_code'),
                                                  Employee.name.label('emp_name'),
                                                  Role.name.label('role_name'),
                                                  Department.name.label('department_name'), func.DATE_FORMAT(
                        func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone), '%d-%m-%Y').label(
                        'seat_booked_date'), func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
                                                  func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label(
                                                      'to_date'), func.DATE_FORMAT(
                        func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC', Timezone.timezone), '%d-%m-%Y').label(
                        'check_in_date'), func.TIME_FORMAT(
                        func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC', Timezone.timezone), '%I:%i:%s %p').label(
                        'check_in_time'), Branch.name.label('branch_name'), (case((and_(
                        Attendance.is_cancelled_employee == CONSTANT.ONE, SeatBooking.is_active == CONSTANT.ZERO),
                                                                                  ('Cancelled')), (
                        or_(SeatBooking.is_accepted == CONSTANT.ZERO,
                            SeatBooking.is_active == CONSTANT.ZERO),
                        ('Rejected')),
                        else_ = ('Rejected'))).label(
                        'attendance_status')).group_by(Attendance.date, Attendance.booking_id).all()
            else:
                bookings = bookings.with_entities((func.concat('SB-', SeatBooking.id)).label('booking_id'),
                                                  Employee.code.label('emp_code'),
                                                  Employee.name.label('emp_name'),
                                                  Role.name.label('role_name'),
                                                  Department.name.label('department_name'), func.DATE_FORMAT(
                        func.CONVERT_TZ(SeatBooking.date, 'UTC', Timezone.timezone), '%d-%m-%Y').label(
                        'seat_booked_date'), func.DATE_FORMAT(SeatBooking.from_date, '%d-%m-%Y').label('from_date'),
                                                  func.DATE_FORMAT(SeatBooking.booking_end_date, '%d-%m-%Y').label(
                                                      'to_date'),
                                                  func.TIME_FORMAT(Slot.start_time, "%I:%i:%s %p").label('start_time'),
                                                  func.DATE_FORMAT(func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC',
                                                                                   Timezone.timezone),
                                                                   '%d-%m-%Y').label('check_in_date'), func.TIME_FORMAT(
                        func.CONVERT_TZ(Attendance.checkin_datetime, 'UTC', Timezone.timezone), '%I:%i:%s %p').label(
                        'check_in_time'), Seat.name.label('seat_name'), FloorMap.name.label('floor_name'),
                                                  Branch.name.label('branch_name'), (case((and_(
                        Attendance.is_absent == CONSTANT.ONE, Attendance.is_holiday == CONSTANT.ZERO,
                        Attendance.is_weekoff == CONSTANT.ZERO), 'Absent'), (and_(Attendance.is_absent == CONSTANT.ONE,
                                                                                  Attendance.is_holiday == CONSTANT.ONE),
                                                                             'Holiday'), (and_(
                        Attendance.is_absent == CONSTANT.ONE, Attendance.is_weekoff == CONSTANT.ONE), 'WeekOff'),
                        else_ = 'Present')).label(
                        'attendance_status')).group_by(Attendance.date, Attendance.booking_id).all()
```

Explanation:

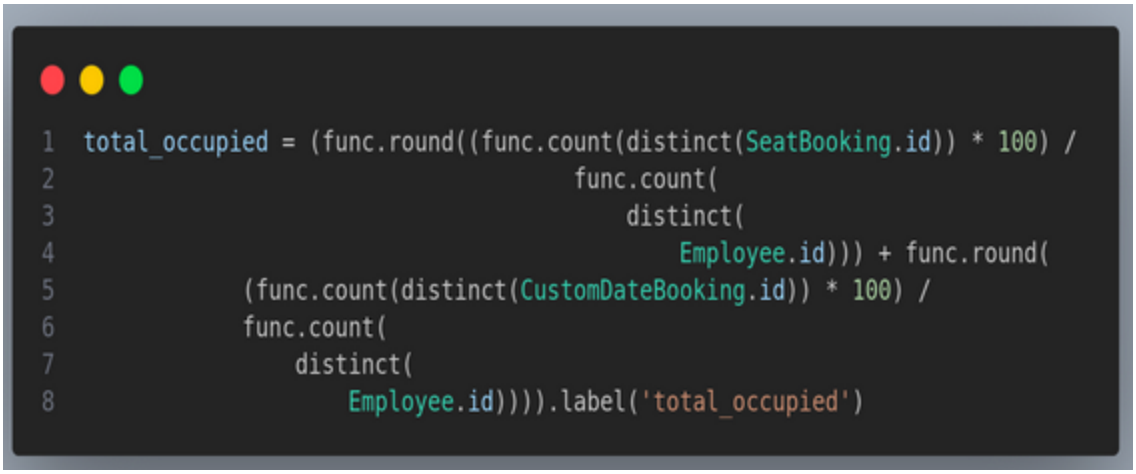
- Specific columns related to rejected date, request type, and custom dates are removed from the Excel columns list.
- The bookings table is joined with the Attendance table based on the condition Attendance.booking_id == SeatBooking.id.

- 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:

```python
@staticmethod
    def get_department_wise_booking_counts( branch_id, date, page, per_page ):
        """
        Method to get_department_wise_booking_counts
        @param branch_id:
        @type branch_id:
        @param date:
        @type date:
        @param page:
        @type page:
        @param per_page:
        @type per_page:
        @return:
        @rtype:
        """
        total_occupied = (func.round((func.count(distinct(SeatBooking.id)) * 100) /
                                     func.count(
                                         distinct(
                                             Employee.id))) + func.round(
            (func.count(distinct(CustomDateBooking.id)) * 100) /
            func.count(
                distinct(
                    Employee.id)))).label('total_occupied')
        return Department.query.join(DepartmentDetail, DepartmentDetail.department_id == Department.id). \
            join(Employee, Employee.department_id == Department.department_id). \
            join(SeatBooking, and_(SeatBooking.employee_id == Employee.id,
                                   SeatBooking.branch_id == branch_id,
                                   SeatBooking.is_active == CONSTANT.ONE,
                                   SeatBooking.is_accepted == CONSTANT.ONE,
                                   SeatBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
                                   SeatBooking.from_date <= date, SeatBooking.booking_end_date >= date,
                                   SeatBooking.is_custom_date_booking == CONSTANT.ZERO,
                                   SeatBooking.deleted == CONSTANT.ZERO, SeatBooking.deleted_at == None),
                 isouter = True). \
            join(CustomDateBooking, and_(CustomDateBooking.employee_id == Employee.id,
                                         CustomDateBooking.branch_id == branch_id,
                                         CustomDateBooking.is_active == CONSTANT.ACTIVE,
                                         CustomDateBooking.from_date == date,
                                         CustomDateBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
                                         CustomDateBooking.deleted == CONSTANT.ZERO,
                                         CustomDateBooking.deleted_at == None
                                         ), isouter = True). \
            with_entities(Department.id, Department.name, total_occupied
                          ).filter(
            DepartmentDetail.branch_id == branch_id, Employee.branch_id == branch_id,
            DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None,
            Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None).order_by(
            total_occupied.desc(), Department.name.asc()).group_by(Department.id).paginate(page, per_page,
                                                                                           error_out = False)

```

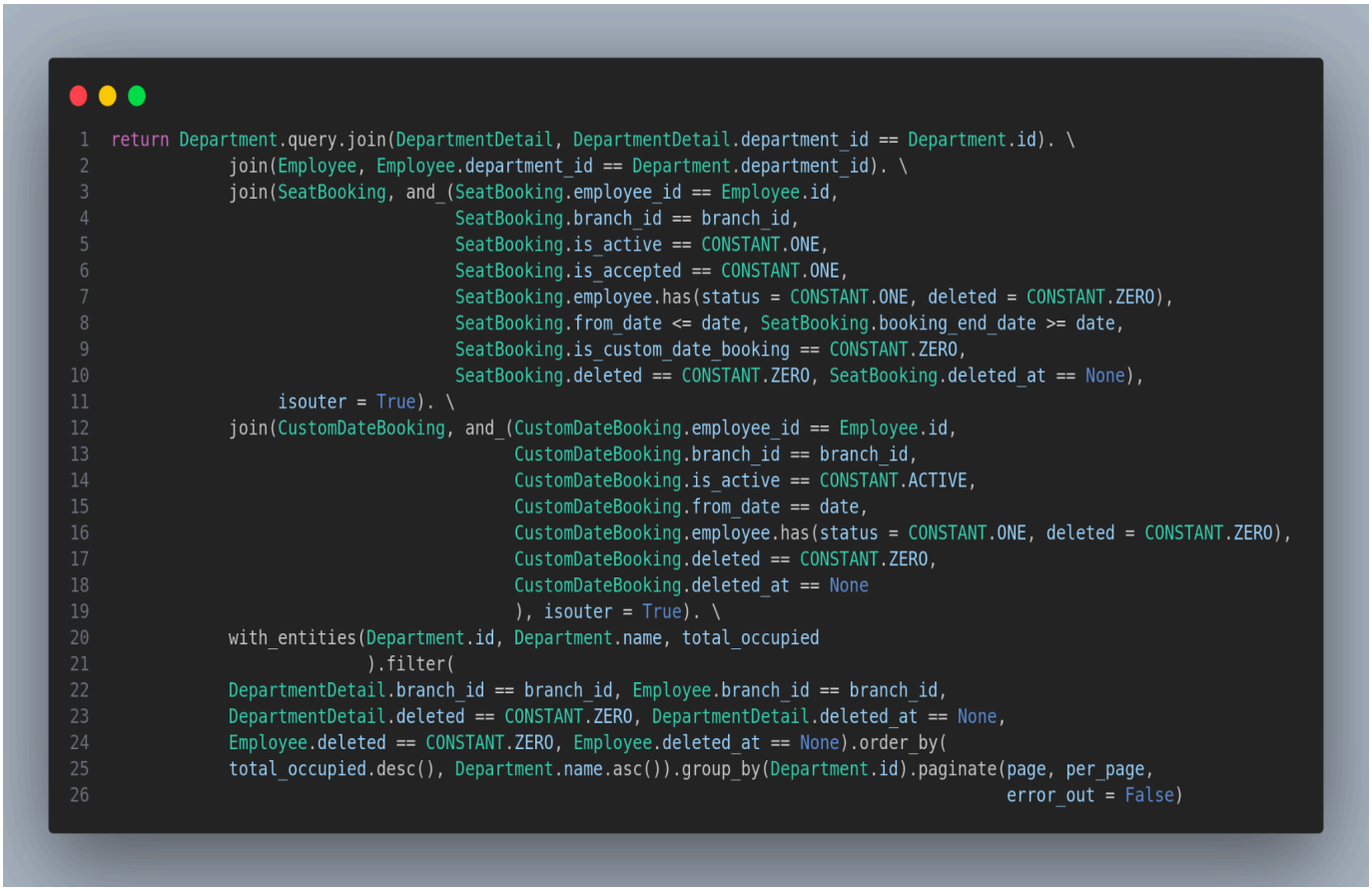
Query Explanation:



```
total_occupied = (func.round((func.count(distinct(SeatBooking.id)) * 100) /
                             func.count(
                                 distinct(
                                     Employee.id))) + func.round(
        (func.count(distinct(CustomDateBooking.id)) * 100) /
        func.count(
            distinct(
                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.



```
return Department.query.join(DepartmentDetail, DepartmentDetail.department_id == Department.id). \
            join(Employee, Employee.department_id == Department.department_id). \
            join(SeatBooking, and_(SeatBooking.employee_id == Employee.id,
                                   SeatBooking.branch_id == branch_id,
                                   SeatBooking.is_active == CONSTANT.ONE,
                                   SeatBooking.is_accepted == CONSTANT.ONE,
                                   SeatBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
                                   SeatBooking.from_date <= date, SeatBooking.booking_end_date >= date,
                                   SeatBooking.is_custom_date_booking == CONSTANT.ZERO,
                                   SeatBooking.deleted == CONSTANT.ZERO, SeatBooking.deleted_at == None),
                 isouter = True). \
            join(CustomDateBooking, and_(CustomDateBooking.employee_id == Employee.id,
                                         CustomDateBooking.branch_id == branch_id,
                                         CustomDateBooking.is_active == CONSTANT.ACTIVE,
                                         CustomDateBooking.from_date == date,
                                         CustomDateBooking.employee.has(status = CONSTANT.ONE, deleted = CONSTANT.ZERO),
                                         CustomDateBooking.deleted == CONSTANT.ZERO,
                                         CustomDateBooking.deleted_at == None
                                         ), isouter = True). \
            with_entities(Department.id, Department.name, total_occupied
                          ).filter(
            DepartmentDetail.branch_id == branch_id, Employee.branch_id == branch_id,
            DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None,
            Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None).order_by(
            total_occupied.desc(), Department.name.asc()).group_by(Department.id).paginate(page, per_page,
                                                                                           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

```python
@staticmethod
def get_booked_seat_ids(branch_details, slot_details, from_date, to_date, is_custom_date_booking, custom_dates):
    """
    Method to get booked seat IDs.

    :param branch_details: Details of the branch.
    :type branch_details: object
    :param slot_details: Details of the slot.
    :type slot_details: object
    :param from_date: Starting date of the booking.
    :type from_date: datetime
    :param to_date: Ending date of the booking.
    :type to_date: datetime
    :param is_custom_date_booking: Flag to indicate custom date booking.
    :type is_custom_date_booking: bool
    :param custom_dates: List of custom dates.
    :type custom_dates: list
    :return: List of booked seat IDs.
    :rtype: list
    """
    from database.CancellationForUnusedBooking import CancellationForUnusedBooking

    seat_ids = []
    custom_booking_seat_ids = []

    combined_from_datetime = func.concat(
        func.cast(SeatBooking.from_date, String), ' ',
        func.cast(Slot.start_time, String)
    )
    combined_to_datetime = func.concat(
        func.cast(SeatBooking.booking_end_date, String), ' ',
        func.cast(Slot.end_time, String)
    )

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

    booking_obj = (
        SeatBooking.query
        .join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None))
        .join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None))
        .join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None))
        .join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None))
        .join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None))
        .join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None))
        .join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None))
        .join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None))
        .join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_id, FloorMap.deleted == CONSTANT.ZERO, FloorMap.deleted_at == None))
        .filter(
            SeatBooking.is_active == True,
            SeatBooking.branch_id == branch_details.id,
            SeatBooking.deleted == CONSTANT.ZERO,
            SeatBooking.deleted_at == None,
            or_(
                and_(
                    SeatBooking.from_date <= to_date,
                    SeatBooking.from_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ZERO,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        )
                    )
                ),
                and_(
                    SeatBooking.from_date <= to_date,
                    SeatBooking.from_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ONE,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        ),
                        and_(
                            combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                            combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                        )
                    )
                ),
                and_(
                    SeatBooking.from_date <= from_date,
                    SeatBooking.to_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ZERO,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        )
                    )
                ),
                and_(
                    SeatBooking.from_date <= from_date,
                    SeatBooking.booking_end_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ONE,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        ),
                        and_(
                            combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                            combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                        )
                    )
                )
            )
        )
    )

    if booking_end_date == from_date or str(is_custom_date_booking) == str(CONSTANT.ONE):
        auto_cancelled_booking_ids = CancellationForUnusedBooking.get_cancelled_booking_list(
            date=from_date, branch_id=branch_details.id, only_id=True
        )
        booking_obj = booking_obj.filter(SeatBooking.id.not_in((auto_cancelled_booking_ids)))
        custom_booking_seat_ids = CustomDateBooking.get_custom_booking_count(
            from_date, booking_end_date, branch_id=branch_details.id, custom_dates=custom_dates, only_booking_seat_id=True
        )
        booking_obj = booking_obj.filter(SeatBooking.is_custom_date_booking == CONSTANT.ZERO)

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

Method Definition:

```python
@staticmethod
def get_booked_seat_ids(branch_details, slot_details, from_date, to_date, is_custom_date_booking, custom_dates):
    """
    Method to get booked seat IDs.

    :param branch_details: Details of the branch.
    :type branch_details: object
    :param slot_details: Details of the slot.
    :type slot_details: object
    :param from_date: Starting date of the booking.
    :type from_date: datetime
    :param to_date: Ending date of the booking.
    :type to_date: datetime
    :param is_custom_date_booking: Flag to indicate custom date booking.
    :type is_custom_date_booking: bool
    :param custom_dates: List of custom dates.
    :type custom_dates: list
    :return: List of booked seat IDs.
    :rtype: list
    """
```

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:

```python
from database.CancellationForUnusedBooking import CancellationForUnusedBooking
seat_ids = []
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:

```python
combined_from_datetime = func.concat(
    func.cast(SeatBooking.from_date, String), ' ',
    func.cast(Slot.start_time, String)
)
combined_to_datetime = func.concat(
    func.cast(SeatBooking.booking_end_date, String), ' ',
    func.cast(Slot.end_time, String)
)
```

- **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:

```
booking_end_date = to_date
if slot_details.start_time > slot_details.end_time:
    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

```
booking_obj = (
        SeatBooking.query
        .join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None))
        .join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None))
        .join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None))
        .join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None))
        .join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None))
        .join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None))
        .join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None))
        .join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None))
        .join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_id, FloorMap.deleted == CONSTANT.ZERO, FloorMap.deleted_at == None))
        .filter(
            SeatBooking.is_active == True,
            SeatBooking.branch_id == branch_details.id,
            SeatBooking.deleted == CONSTANT.ZERO,
            SeatBooking.deleted_at == None,
            or_(
                and_(
                    SeatBooking.from_date <= to_date,
                    SeatBooking.from_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ZERO,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        )
                    )
                ),
                and_(
                    SeatBooking.from_date <= to_date,
                    SeatBooking.from_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ONE,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        ),
                        and_(
                            combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                            combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                        )
                    )
                ),
                and_(
                    SeatBooking.from_date <= from_date,
                    SeatBooking.to_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ZERO,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        )
                    )
                ),
                and_(
                    SeatBooking.from_date <= from_date,
                    SeatBooking.booking_end_date >= from_date,
                    SeatBooking.is_night_shift == CONSTANT.ONE,
                    or_(
                        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
                        not_(
                            or_(
                                combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                                combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                            )
                        ),
                        and_(
                            combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time),
                            combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)
                        )
                    )
                )
            )
        )
    )
```

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

```python
if booking_end_date == from_date or str(is_custom_date_booking) == str(CONSTANT.ONE):
    auto_cancelled_booking_ids = CancellationForUnusedBooking.get_cancelled_booking_list(
        date=from_date, branch_id=branch_details.id, only_id=True
    )
    booking_obj = booking_obj.filter(SeatBooking.id.not_in((auto_cancelled_booking_ids)))
    custom_booking_seat_ids = CustomDateBooking.get_custom_booking_count(
        from_date, booking_end_date, branch_id=branch_details.id, custom_dates=custom_dates, only_booking_seat_id=True
    )
    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

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

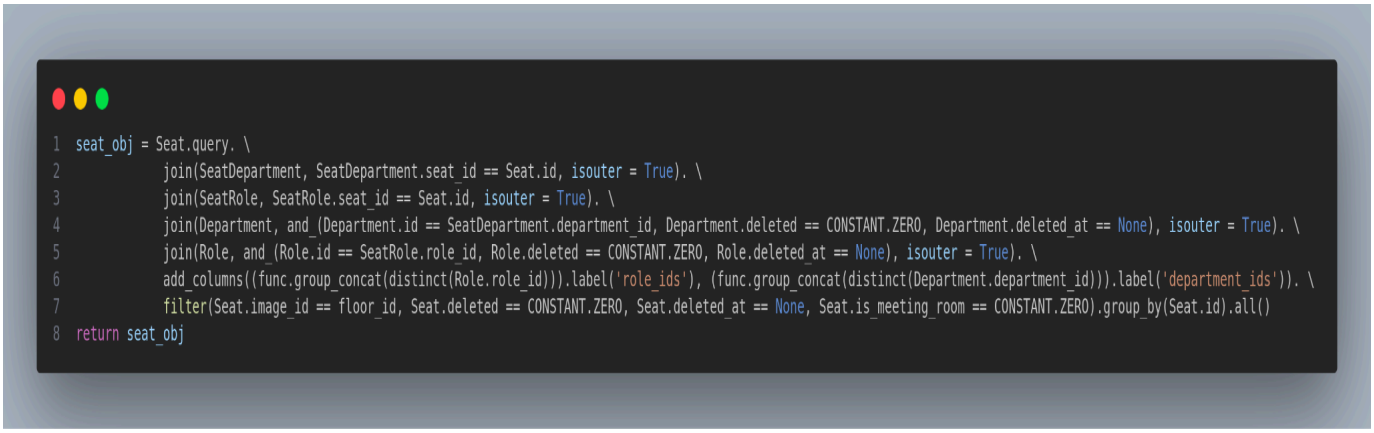
```
@staticmethod
def get_seat_list_mobile( floor_id ):
    """
    Method to get_seat_list_mobile
    @param branch_id:
    @param floor_id:
    @type floor_id:
    @return:
    @rtype:
    """
    from database.SeatDepartment import SeatDepartment
    from database.SeatRole import SeatRole

    seat_obj = Seat.query. \
                join(SeatDepartment, SeatDepartment.seat_id == Seat.id, isouter = True). \
                join(SeatRole, SeatRole.seat_id == Seat.id, isouter = True). \
                join(Department, and_(Department.id == SeatDepartment.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None), isouter = True). \
                join(Role, and_(Role.id == SeatRole.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None), isouter = True). \
                add_columns((func.group_concat(distinct(Role.role_id))).label('role_ids'), (func.group_concat(distinct(Department.department_id))).label('department_ids')). \
                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()
    return seat_obj
```

Method Definition and Docstring:

```
@staticmethod
def get_seat_list_mobile( floor_id ):
    """
    Method to get_seat_list_mobile
    @param branch_id:
    @param floor_id:
    @type floor_id:
    @return:
    @rtype:
    """
```

- **@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:



```
seat_obj = Seat.query. \
            join(SeatDepartment, SeatDepartment.seat_id == Seat.id, isouter = True). \
            join(SeatRole, SeatRole.seat_id == Seat.id, isouter = True). \
            join(Department, and_(Department.id == SeatDepartment.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None), isouter = True). \
            join(Role, and_(Role.id == SeatRole.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None), isouter = True). \
            add_columns((func.group_concat(distinct(Role.role_id))).label('role_ids'), (func.group_concat(distinct(Department.department_id))).label('department_ids')). \
            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()
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:

```
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 ):
    from database.SeatBooking import SeatBooking
    from database.Department import Department
    from database.Role import Role
    data = []
    is_paginated = False
    if input_data:
        original_datas = input_data
        if (hasattr(input_data, 'page') and hasattr(input_data, 'items')) or (
                'page' in input_data and 'items' in input_data):
            original_datas = input_data.items
            is_paginated = True
        for each_data in original_datas:
            dict_data = dict()
            role_ids = each_data['role_ids'].split(',') if each_data['role_ids'] else []
            department_ids = each_data['department_ids'].split(',') if each_data['department_ids'] else []
            each_data = each_data[0]
            dict_data['id'] = str(each_data.id)
            dict_data['name'] = str(each_data.name)
            dict_data['htmlId'] = each_data.htmlId
            dict_data['htmlWidth'] = str(each_data.htmlWidth)
            dict_data['htmlHeight'] = str(each_data.htmlHeight)
            dict_data['htmlLeft'] = str(each_data.htmlLeft)
            dict_data['htmlTop'] = str(each_data.htmlTop)
            dict_data['intHtmlId'] = str(each_data.intHtmlId)
            dict_data['image'] = floor.name
            dict_data['booked_employee_id'] = ''
            dict_data['booked_employee_name'] = ''
            dict_data['booked_employee_role'] = ''
            dict_data['booked_employee_department'] = ''
            print("SELaaaaaaaaaaaaVAMAI")
            print(each_data.is_common_seat)
            if is_show_employee_details == str(CONSTANT.ONE):
                booking_data = SeatBooking.get_booked_employee_data(floor.branch, slot, from_date, to_date, is_custom_date_booking, custom_dates, each_data.id)
                if booking_data:
                    dict_data['booked_employee_id'] = str(booking_data.employee.e_id)
                    dict_data['booked_employee_name'] = booking_data.employee.emp_name
                    role = Role.get_role_data(sequre_role_id = booking_data.employee.role_id)
                    department = Department.get_department_data(sequre_department_id = booking_data.employee.department_id)
                    dict_data['booked_employee_role'] = role.name if role else ''
                    dict_data['booked_employee_department'] = department.name if department else ''
            dict_data['is_available'] = str(CONSTANT.ONE) if (each_data.id not in booked_seat_ids and each_data.is_common_seat) else 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)
            dict_data['is_booked'] = str(CONSTANT.ONE) if each_data.id in booked_seat_ids else str(CONSTANT.ZERO)
            dict_data['is_role_available'] = str(CONSTANT.ONE) if role_id in role_ids else str(CONSTANT.ZERO)
            dict_data['is_dept_available'] = str(CONSTANT.ONE) if department_id in department_ids else str(CONSTANT.ZERO)
            dict_data['selected_seat_color'] = floor.branch.company.spacemanagementconfiguration.selected_color if hasattr(floor.branch.company,'spacemanagementconfiguration') and floor.branch.company.spacemanagementconfiguration else str(CONSTANT.ZERO)
            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.ZERO)
            dict_data['seat_color'] = CustomFormatter.get_seat_color(floor.branch.company, dict_data['is_available'], dict_data['is_booked'], dict_data['is_accepted'])
            data.append(dict_data)
        if is_paginated:
            data = util.paginated_format(data, input_data)
    return data
```

Method Definition and Imports:

```
@staticmethod
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):
    from database.SeatBooking import SeatBooking
    from database.Department import Department
    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:

```
data = []
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:

```
if input_data:
    original_datas = input_data
    if (hasattr(input_data, 'page') and hasattr(input_data, 'items')) or ('page' in input_data and 'items' in input_data):
        original_datas = input_data.items
        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:

```
for each_data in original_datas:
    dict_data = dict()
    role_ids = each_data['role_ids'].split(',') if each_data['role_ids'] else []
    department_ids = each_data['department_ids'].split(',') if each_data['department_ids'] else []
    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:

```
dict_data['id'] = str(each_data.id)
dict_data['name'] = str(each_data.name)
dict_data['htmlId'] = each_data.htmlId
dict_data['htmlWidth'] = str(each_data.htmlWidth)
dict_data['htmlHeight'] = str(each_data.htmlHeight)
dict_data['htmlLeft'] = str(each_data.htmlLeft)
dict_data['htmlTop'] = str(each_data.htmlTop)
dict_data['intHtmlId'] = str(each_data.intHtmlId)
dict_data['image'] = floor.name
dict_data['booked_employee_id'] = ''
dict_data['booked_employee_name'] = ''
dict_data['booked_employee_role'] = ''
dict_data['booked_employee_department'] = ''
```

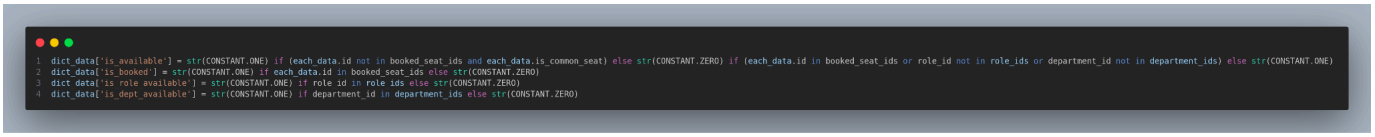
- **Set Seat Attributes:** Populate dict_data with the seat's attributes, converting them to strings where necessary.
- **Employee Booking Details:** Initialize booking-related fields as empty strings.

Debug Statements and Conditional Check for Employee Details:

```
if is_show_employee_details == str(CONSTANT.ONE):
    booking_data = SeatBooking.get_booked_employee_data(floor.branch, slot, from_date, to_date, is_custom_date_booking, custom_dates, each_data.id)
    if booking_data:
        dict_data['booked_employee_id'] = str(booking_data.employee.e_id)
        dict_data['booked_employee_name'] = booking_data.employee.emp_name
        role = Role.get_role_data(sequre_role_id = booking_data.employee.role_id)
        department = Department.get_department_data(sequre_department_id = booking_data.employee.department_id)
        dict_data['booked_employee_role'] = role.name if role else ''
        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:



```
dict_data['is_available'] = str(CONSTANT.ONE) if (each_data.id not in booked_seat_ids and each_data.is_common_seat) else 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)
dict_data['is_booked'] = str(CONSTANT.ONE) if each_data.id in booked_seat_ids else str(CONSTANT.ZERO)
dict_data['is_role_available'] = str(CONSTANT.ONE) if role_id in role_ids else str(CONSTANT.ZERO)
dict_data['is_dept_available'] = str(CONSTANT.ONE) if department_id in department_ids else str(CONSTANT.ZERO)
```

- **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.

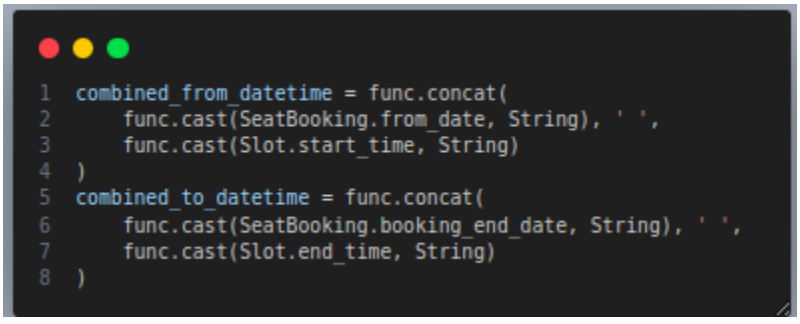
Seat Color and Acceptance Status:

```
dict_data['selected_seat_color'] = floor.branch.company.spacemanagementconfiguration.selected_color if hasattr(floor.branch.company,'spacemanagementconfiguration') and floor.branch.company.spacemanagementconfiguration else str(CONSTANT.ZERO);
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.ZERO)
dict_data['seat_color'] = CustomFormatter.get_seat_colot(floor.branch.company, dict_data['is_available'], dict_data['is_booked'], dict_data['is_accepted'])
```

- **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_colot based on availability, booking, and acceptance status.

4.Query to get_booked_employee_data:

Combined Date-Time Strings:



```
combined_from_datetime = func.concat(
    func.cast(SeatBooking.from_date, String), ' ',
    func.cast(Slot.start_time, String)
)
combined_to_datetime = func.concat(
    func.cast(SeatBooking.booking_end_date, String), ' ',
    func.cast(Slot.end_time, String)
)
```

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:

```
booking_end_date = to_date
if slot_details.start_time > slot_details.end_time:
    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:

```
bookingObj = SeatBooking.query. \
    join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None)). \
    join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None)). \
    join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None)). \
    join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None)). \
    join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None)). \
    join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None)). \
    join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None)). \
    join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None)). \
    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**

```
.filter(
    Seat.id == seat_id,
    SeatBooking.is_active == True,
    SeatBooking.deleted == CONSTANT.ZERO,
    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**

```
or_(
    and_(
        SeatBooking.from_date <= to_date,
        SeatBooking.from_date >= from_date,
        SeatBooking.is_night_shift == CONSTANT.ZERO,
        or_(
            and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
            and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
            not_(or_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)))
        )
    ),

```

- **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

```
and_(
    SeatBooking.from_date <= to_date,
    SeatBooking.from_date >= from_date,
    SeatBooking.is_night_shift == CONSTANT.ONE,
    or_(
        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
        not_(or_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time))),
        and_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time))
    )
),
```

- **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.

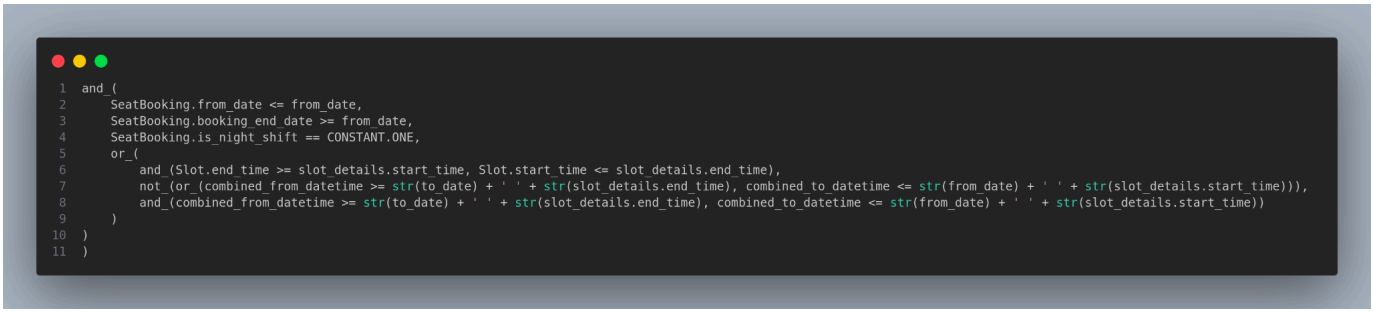
c. Day Shift Booking Covering Entire Date Range

```
and_(
    SeatBooking.from_date <= from_date,
    SeatBooking.to_date >= from_date,
    SeatBooking.is_night_shift == CONSTANT.ZERO,
    or_(
        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
        and_(Slot.end_time <= slot_details.start_time, Slot.start_time >= slot_details.end_time),
        not_(or_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time)))
    )
),
```

- **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.

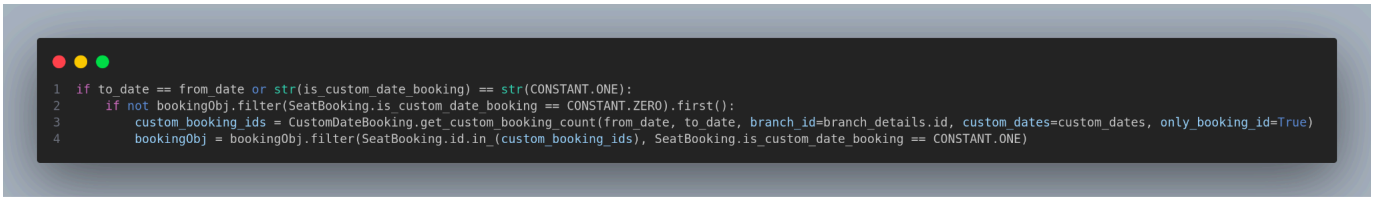
d. Night Shift Booking Covering Entire Date Range



```
and_(
    SeatBooking.from_date <= from_date,
    SeatBooking.booking_end_date >= from_date,
    SeatBooking.is_night_shift == CONSTANT.ONE,
    or_(
        and_(Slot.end_time >= slot_details.start_time, Slot.start_time <= slot_details.end_time),
        not_(or_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time))),
        and_(combined_from_datetime >= str(to_date) + ' ' + str(slot_details.end_time), combined_to_datetime <= str(from_date) + ' ' + str(slot_details.start_time))
    )
)
)
```

- **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:



```
if to_date == from_date or str(is_custom_date_booking) == str(CONSTANT.ONE):
    if not bookingObj.filter(SeatBooking.is_custom_date_booking == CONSTANT.ZERO).first():
        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)
        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).

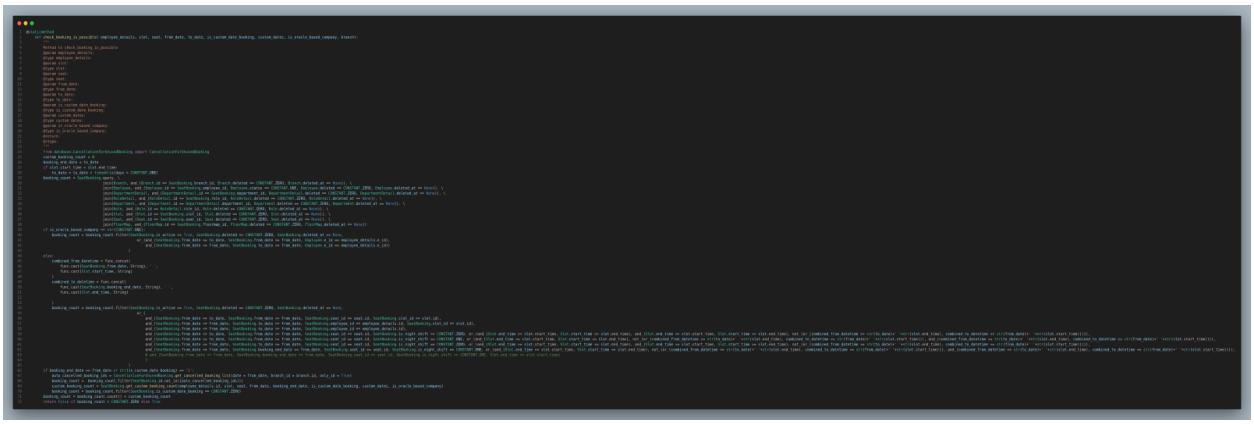
Final Query Execution:

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



Method Definition and Parameters

```
def check_booking_is_possible(employee_details, slot, seat, from_date, to_date, is_custom_date_booking, custom_dates, is_oracle_based_company, branch):
    """
    Method to check_booking_is_possible
    """
```

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

```
from database.CancellationForUnusedBooking import CancellationForUnusedBooking
custom_booking_count = 0
booking_end_date = to_date
```

- **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

```
if slot.start_time > slot.end_time:
    to_date = to_date + timedelta(days=CONSTANT.ONE)
```

- 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

```
booking_count = SeatBooking.query. \
    join(Branch, and_(Branch.id == SeatBooking.branch_id, Branch.deleted == CONSTANT.ZERO, Branch.deleted_at == None)). \
    join(Employee, and_(Employee.id == SeatBooking.employee_id, Employee.status == CONSTANT.ONE, Employee.deleted == CONSTANT.ZERO, Employee.deleted_at == None)). \
    join(DepartmentDetail, and_(DepartmentDetail.id == SeatBooking.department_id, DepartmentDetail.deleted == CONSTANT.ZERO, DepartmentDetail.deleted_at == None)). \
    join(RoleDetail, and_(RoleDetail.id == SeatBooking.role_id, RoleDetail.deleted == CONSTANT.ZERO, RoleDetail.deleted_at == None)). \
    join(Department, and_(Department.id == DepartmentDetail.department_id, Department.deleted == CONSTANT.ZERO, Department.deleted_at == None)). \
    join(Role, and_(Role.id == RoleDetail.role_id, Role.deleted == CONSTANT.ZERO, Role.deleted_at == None)). \
    join(Slot, and_(Slot.id == SeatBooking.slot_id, Slot.deleted == CONSTANT.ZERO, Slot.deleted_at == None)). \
    join(Seat, and_(Seat.id == SeatBooking.seat_id, Seat.deleted == CONSTANT.ZERO, Seat.deleted_at == None)). \
    join(FloorMap, and_(FloorMap.id == SeatBooking.floormap_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

```
if is_oracle_based_company == str(CONSTANT.ONE):
    booking_count = booking_count.filter(SeatBooking.is_active == True, SeatBooking.deleted == CONSTANT.ZERO, SeatBooking.deleted_at == None,
        or_(and_(SeatBooking.from_date <= to_date, SeatBooking.from_date >= from_date, Employee.e_id == employee_details.e_id),
            and_(SeatBooking.from_date <= from_date, SeatBooking.to_date >= from_date, Employee.e_id == employee_details.e_id))
    )

```

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

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.

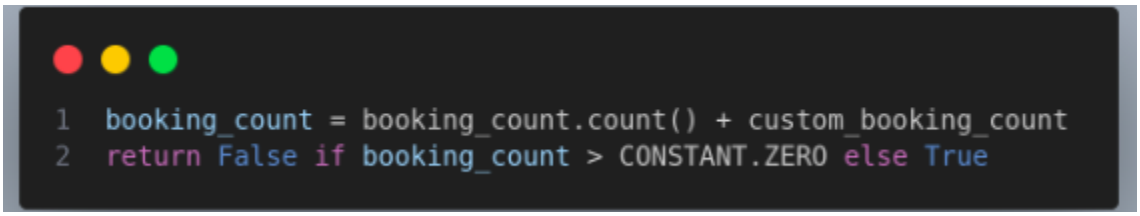
Custom Date Booking and Auto-Cancellation Handling

```
if booking_end_date == from_date or str(is_custom_date_booking) == '1':
    auto_cancelled_booking_ids = CancellationForUnusedBooking.get_cancelled_booking_list(date=from_date, branch_id=branch.id, only_id=True)
    booking_count = booking_count.filter(SeatBooking.id.not_in((auto_cancelled_booking_ids)))
    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)
    booking_count = booking_count.filter(SeatBooking.is_custom_date_booking == CONSTANT.ZERO)

```

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



```
booking_count = booking_count.count() + custom_booking_count
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.