BOOK BOOK GO

CS 348 Final Project Report

Prof. - Xi He

TA - Glaucia Melo dos Santos

Members

Jacky Chen

Komal Sarvani Mullapudi

Nyle Kabiruddin Dharani

Olivier Ambroise

Shichen Hu

Table of Contents

NOTE: Any part of the report from Milestone 2 is under the highlight "Milestone 2 stuff" and any changes included as part of our final report is under the highlight "End of Milestone 2, Start of Changes from last time, if any"

Intended Demographic

Milestone 2 stuff

Our group is working on building a Hotel Booking app (currently nicknamed BookBookGo) where we'll be building our own datasets with inspiration from other hotel or room booking datasets or applications. Our intended users are people who wish to book a room in a hotel either singly, or in a group for travel or vacations and basically, someone who likes to be spoiled with choice for their temporary place of stay!

End of Milestone 2, Start of Changes from last time, if any

Technology Stack Used

Milestone 2 stuff

Our system support consists of Django Framework, Python (using MySql Client to interact with the database), Google Cloud MySQL (as the backend and database support) and JavaScript with React.js for working on the user interface.

Since all our users will be allotted rooms of some form or another, we've designed our schema based on the assumption that the room numbers are unique. We're taking into consideration that the same room might be used in a lifespan multiple times, but we'll be updating the old records by archiving them or handling them appropriately such that at any given time, there's only one valid corresponding record for a room number. We are also operating under the assumption that multiple guests will be part of one room, but there'll be only one primary or one secondary contact who's name the booking is under and the rest of the guests are their "roommates".

End of Milestone 2, Start of Changes from last time, if any

Files Structure

hotelbackend/urls.py : API Endpoints

hotels/views.py : Backend Code / SQL Queries Doc for Endpoints

hotel001.sql : How to create Tables and Indexes for our Backend

hotels/model.py : Tables (Abstracted in Django) hotels/serializers.py : Format to create json serializable objects from each table

test-production.sql/out : executable SQL queries we used

README.txt : How our codebase is executed & how data is generated

hotelfrontend : Built in React (http://localhost:3000)

Database Population Method

Milestone 2 stuff

Now, the way we plan on getting data to populate our database is by using a csv from (personally generated one from https://mockaroo.com/)Uploading and writing data into
MySQL database is done using Django Framework with raw SQL queries. The file is called create_hotel_rooms.py

Some detail into what we're using as a guide is:
Iterating over CSV rows and using the below SQL statement to make sure we meet our constraints and relationships.

```
INSERT INTO table-name (column-names)
SELECT column-names
FROM table-name
WHERE condition
```

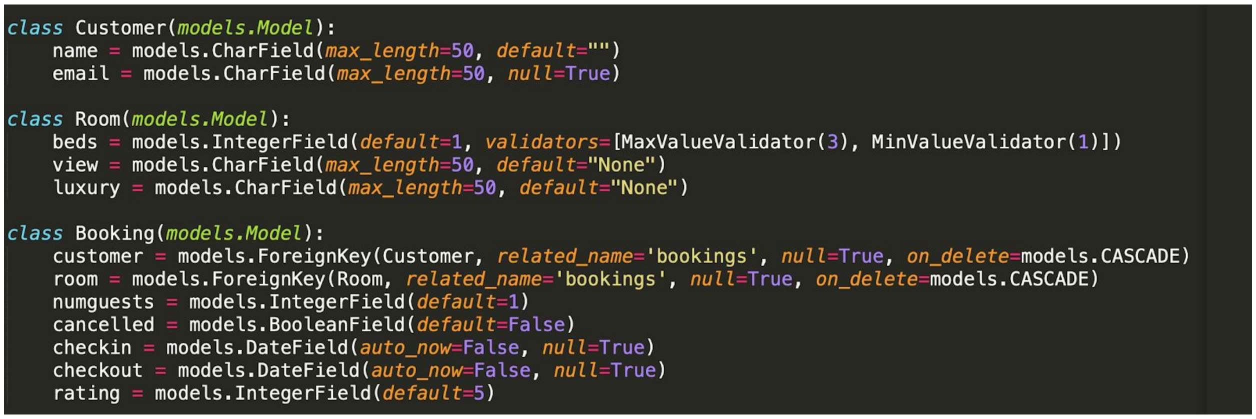
Creating Objects and adding Foriegn keys. Look at create_hotel_rooms.py for more info.

End of Milestone 2, Start of Changes from last time, if any

Updated Database Population Method:

Creating the database schema in the Google Cloud Platform Database:

We used the Django model to create the Tables in the Database



```
class Customer(models.Model):
    name = models.CharField(max_length=50, default="")
    email = models.CharField(max_length=50, null=True)

class Room(models.Model):
    beds = models.IntegerField(default=1, validators=[MaxValueValidator(3), MinValueValidator(1)])
    view = models.CharField(max_length=50, default="None")
    luxury = models.CharField(max_length=50, default="None")

class Booking(models.Model):
    customer = models.ForeignKey(Customer, related_name='bookings', null=True, on_delete=models.CASCADE)
    room = models.ForeignKey(Room, related_name='bookings', null=True, on_delete=models.CASCADE)
    numguests = models.IntegerField(default=1)
    cancelled = models.BooleanField(default=False)
    checkin = models.DateField(auto_now=False, null=True)
    checkout = models.DateField(auto_now=False, null=True)
    rating = models.IntegerField(default=5)
```

The above code is equivalent to:

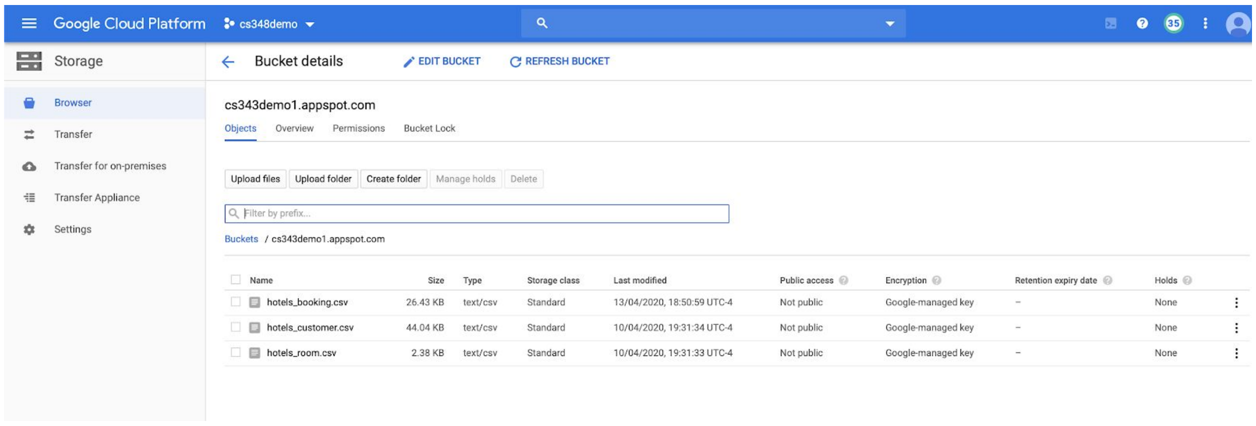
```
CREATE TABLE Customer(
     id int NOT NULL AUTO_INCREMENT,
     name varchar(50) DEFAULT '',
     email varchar(50),
     PRIMARY KEY (id));

CREATE TABLE Room(
     id int NOT NULL AUTO_INCREMENT,
     beds int DEFAULT 1 CHECK (beds <= 3 AND beds >=1 ),
     view varchar(50) DEFAULT 'None',
     luxury varchar(50),
     PRIMARY KEY (id));

CREATE TABLE Booking(
     id int NOT NULL AUTO_INCREMENT,
     customer int,
     room int,
     numguests int DEFAULT 1,
     cancelled int DEFAULT 0,
     checkin date,
     checkout date,
     rating int DEFAULT 5,
     PRIMARY KEY (id),
     FOREIGN KEY (customer) REFERENCES Customer (id) ON DELETE
CASCADE,
    FOREIGN KEY (room) REFERENCES Room (id) ON DELETE CASCADE

);
```

Porting data into the database

Due to the difficulty of finding sets of real-world booking data of a specific hotel, we decided to generate our data set through a data generator: https://www.generatedata.com.

We generated three csv, corresponding to three tables in our schema. After generating the data set, we run the data files through a python script to trim the data, making the data-set comply with our database constraints, for example, no overlap booking.

Then we upload the csv data files to the Google Cloud Platform's Storage Module and manually imported the csv files into our database.

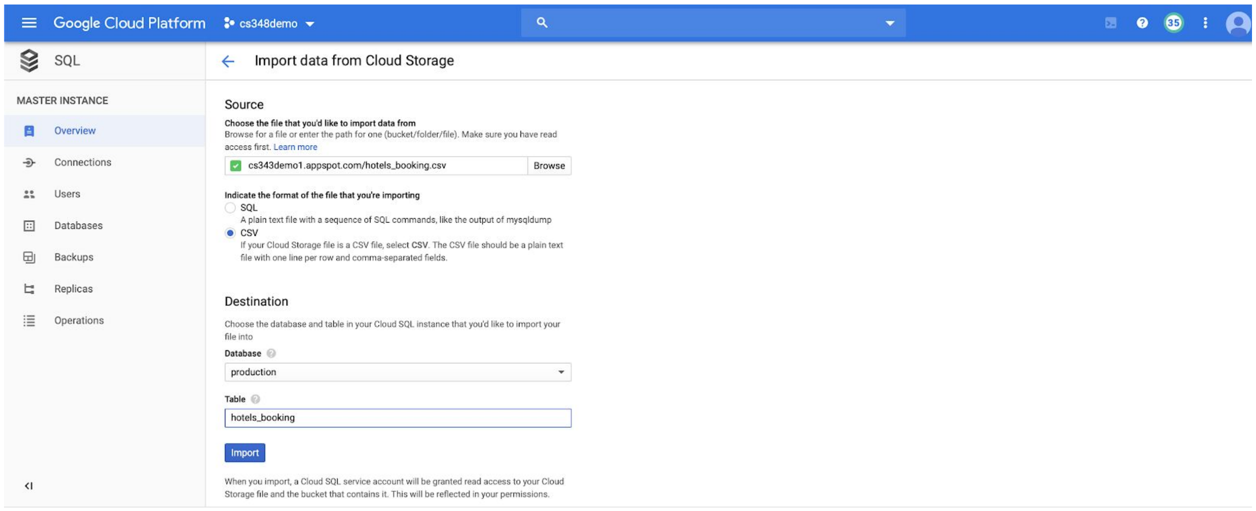


cs343demo1.appspot.com

Buckets / cs343demo1.appspot.com

Name	Size	Type	Storage class	Last modified	Public access	Encryption	Retention expiry date	Holds
hotels_booking.csv	26.43 KB	text/csv	Standard	13/04/2020, 18:50:59 UTC-4	Not public	Google-managed key	–	None
hotels_customer.csv	44.04 KB	text/csv	Standard	10/04/2020, 19:31:34 UTC-4	Not public	Google-managed key	–	None
hotels_room.csv	2.38 KB	text/csv	Standard	10/04/2020, 19:31:33 UTC-4	Not public	Google-managed key	–	None

Figure 1. The Data source files on GCP



Import data from Cloud Storage

Source

Choose the file that you'd like to import data from
Browse for a file or enter the path for one (bucket/folder/file). Make sure you have read access first. Learn more

cs343demo1.appspot.com/hotels_booking.csv Browse

Indicate the format of the file that you're importing
- ○ SQL
  A plain text file with a sequence of SQL commands, like the output of mysqldump
- ● CSV
  If your Cloud Storage file is a CSV file, select **CSV**. The CSV file should be a plain text file with one line per row and comma-separated fields.

Destination

Choose the database and table in your Cloud SQL instance that you'd like to import your file into

Database: production

Table: hotels_booking

Import

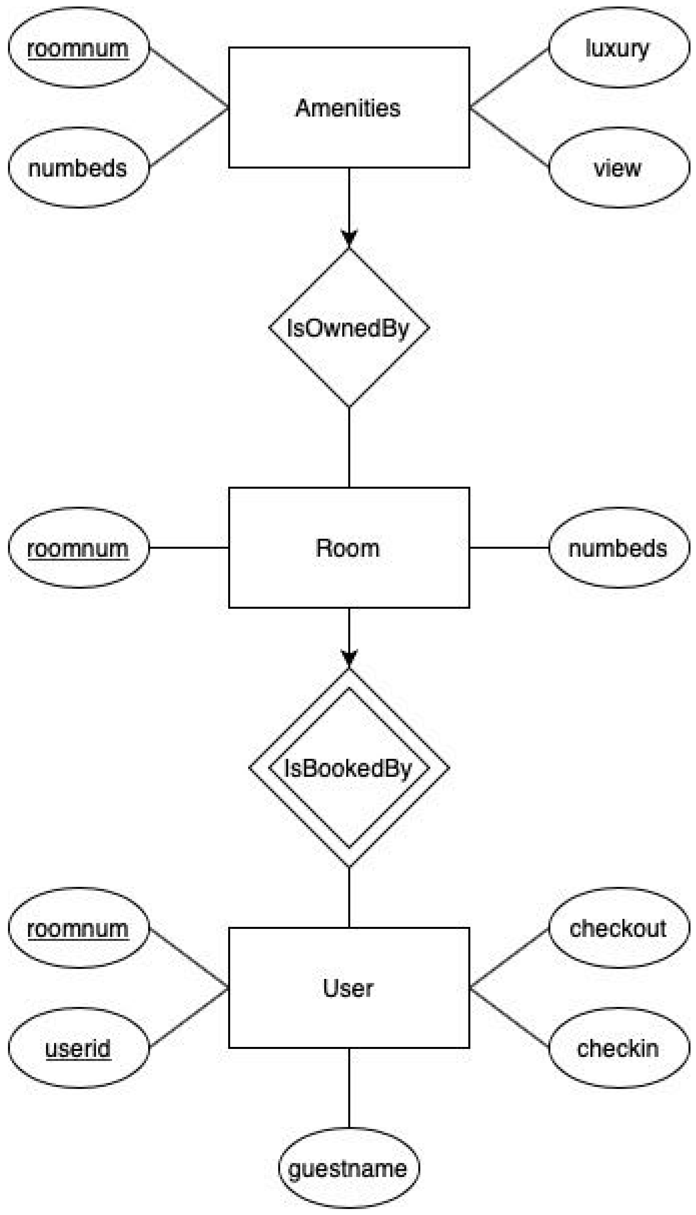
When you import, a Cloud SQL service account will be granted read access to your Cloud Storage file and the bucket that contains it. This will be reflected in your permissions.

Figure 2. Importing the data files into the database tables

E/R Diagram, DB Schema & Design

Milestone 2 stuff

The E/R diagram we are working with currently is : (More updates will be made as our project expands)



Corresponding Database Schema:

Room(<u>roomnum</u>, numbeds)
User(<u>userid, roomnum</u>, guestname, checkin, checkout)
Amenities(<u>roomnum,</u> numbeds, view, luxury)
(Primary keys have been underlined according to the norm)

Design:

The design is based on real world relationships. Our customers are sole entities, our rooms are sole entities and since we can actually move some amenities around they are also sole entities. As we scale some amenities like view and some perks will be owned by the room, then our design will adapt and create double lined relationships so that we know what needs to be cascade deleted or not (for amenities).

Indexes:

hotel001.sql (2 indexes)

Features Implemented:

(views.py) customer.html rooms.html
- Getting customers
- Getting list of hotels (2 features) various variables used
- General hotel view
- Adding a booking

Look at readme for data transfer instructions

End of Milestone 2, Start of Changes from last time, if any

Updated Schema:

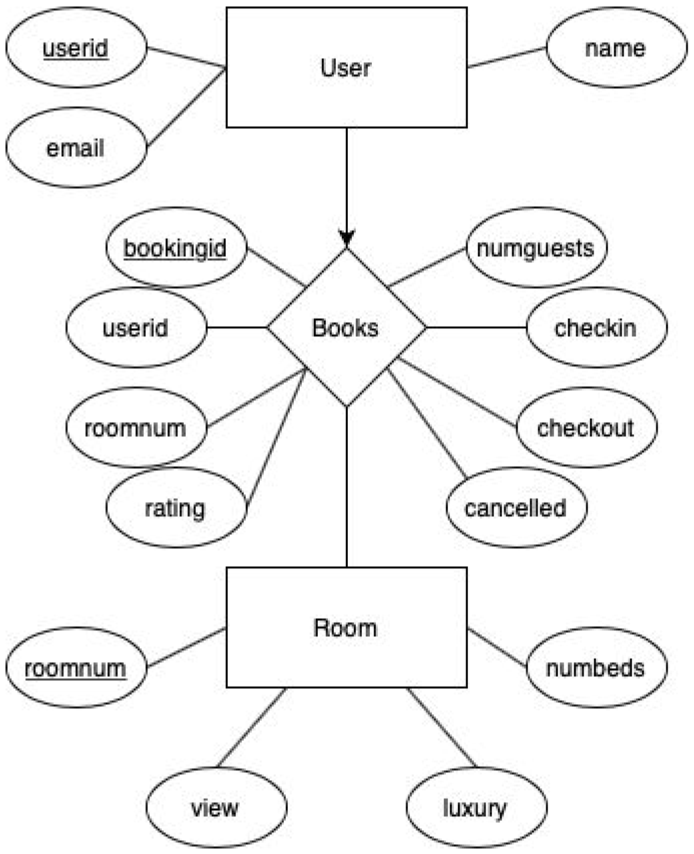
Assumptions:

- Userid, bookid, roomnum are unique.
- A user can have zero bookings.
- Assumed number of beds can only be from 1 to 3
- Rating takes values 1-5
- Checkin date cannot be after the checkout date
- Rooms can have multiple bookings over time.
- No overlap booking for one room at the same time

We removed the Amenities relation as we now track them using the Room relation. We have also added a Booking relation to our schema to better keep track of orders between Users and Rooms over time.

User(userid, name, email)
Books(bookingid, userid, roomnum, numguests, checkin, checkout, cancelled, rating)
Room(roomnum, numbeds, view, luxury)

Updated E/R Diagram:



Attribute mapping

Table 1 below maps the attribute names in the schema to the ones used in our backend database:

Attribute name in schema	Attribute name in database
User.userid	hotels_customer.id
User.name	hotels_customer.name
User.email	hotels_customer.email
Booking.bookingid	hotels_booking.id
Booking.userid	hotels_booking.customer_id
Booking.roomnum	hotels_booking.room_id
Booking.numguests	hotels_booking.numguests
Booking.checkin	hotels_booking.checkin
Booking.checkout	hotels_booking.checkout
Booking.cancelled	hotels_booking.cancelled
Booking.rating	hotels_booking.rating
Room.roomnum	hotels_room.id
Room.numbeds	hotels_room.beds
Room.view	hotels_room.view
Room.luxury	hotels_room.luxury

Table 1. Schema-to-database attribute mappings

Features

Milestone 2 stuff

While building our application, we're taking into account the possible factors that might influence the user's decision to book a room in a hotel. This includes variables (in no particular order) like : number of people, number and type of beds, price of room, photos, dates booked for, view from room, amenities like Internet or T.V or Room Service and such (Details are present in our schema).

These above variables in turn, are what we're considering in the kind of queries we'll generate or support. Some examples include:

● *Add* a new booking with specifications of different variables
Eg: Add an entry for 2 people looking for a king sized bed in a room with a sea-facing view that has unlimited high-speed WiFi, and a T.V for the dates Feb 10th to Feb 12th, 2020.

● *Update* a booking for an existing client of the hotel with the changed variables
Eg: Move Mr.John Doe from a sea-facing room to a room with a yard.
Group guests Alice and Bob in Alice's room and extend the number of days of stay by 2.

● *Cancel* a booking for someone.
Eg: Cancel Ms.Bond's booking and delete or archive her entries as she had to run off on a mission and cut her trip short.

● *Obtain* a list of all the guests who satisfy a given condition.
Eg: For all the guests who are about to check in on the 19th of February, send them a reminder of their upcoming booking with the hotel.

● *Check* the value of a feature and use it to determine the appropriate response to the client.
Eg: If all the rooms with 2 beds are booked, display a greyed out 2 bed option to the client indicating the unavailability to book such rooms.

● *Display Stats* of a particular feature using aggregate functions
Eg: Aggregate functions like count or sum can be used to get an idea of the total number of guests, total number of guests in rooms with a particular view, total number of open rooms etc.

Then, we'll be using the data received through the querying to update the number of rooms left and such while trying to give the users a seamless interface that answers all their queries regarding a BookBookGo room-booking process with Hotel X!

End of Milestone 2, Start of Changes from last time, if any

Final features:

- Filter rooms that match search criteria
- Check these rooms against existing bookings on those rooms and present results that don't overlap with existing bookings
- Favourite / Unfavourite results to save them to book later (This feature uses the browser's local storage and not the prod. DB like the other features)
- If customer enters details, create booking under old customer id if customer exists, if not, create a new customer and then create a new booking
- Update a booking
- Cancel a booking.
- Rate a booking.
- Display the hotel's average rating.
- Display the number of rooms currently available at the hotel.
- Display the number of customers received so far.
- Display the most popular type of view at the hotel.

If you want to see further information on the corresponding CRUD operations, please see below under User Interaction with BookBookGo Interface (the implementation of the queries and features using them is also described in the video demo and is part of our code base).

User Interaction with BookBookGo Interface

Milestone 2 stuff

Speaking of the interface, we've broadly split it into 4 chunks or stages of the happy path that the user will take while interacting with our app:

The initial page that the user will see is a portal that helps them search based on say, location or a couple other features that are important to them. This would ideally be a freeform field where they can search for one location and we'll map it to the closest available hotel we have, but for now we'll be implementing dropdowns that the user can make selections from. Once they click submit, we'll take all those parameters and retrieve the corresponding results from our database that match the given criteria.

On the results page that is displayed, we plan to have a photo or two of the place, list of features using icons to increase the user's sense of familiarity with or understanding of the product and a couple options that they can modify - like number of rooms, the view from the room, number of beds, dates for booking and so on. Once the user has decided that they want to book this particular place, they'll click 'Book'.

Clicking 'Book!' will take the user to the 'Finalize Booking' page where we'll ask the user for their details like Name, E-mail ID, phone number and such and confirm that the details presented to them are the ones they've chosen. Once the user has confirmed their booking, we'll take them to the final screen which confirms the booking as displays this confirmation from BookBookGo's side as well that we've processed everything the user asked for. At the bottom of the screen, there'll be another option for them to return to the main site and book another place or such.

End of Milestone 2, Start of Changes from last time, if any

Homepage

The user is presented with 3 options on the Homepage.

1. Create a booking
2. Manage a booking
3. View hotel statistics

Create a Booking

Under this page, the user is initially presented with a search / filter form and an empty section of Favourites. The search form has various fields: number of guests (which will be used to create a booking and is maxed out around 12 guests if number of beds hits 8), number of beds (lies between 1 and 8), type of view (a view can be one of: Street, Garden, River, City Skyline, No View), type of suite (a suite can be one of: Single, Double, Queen, King, Studio, Master, Presidential). Beds, view and suite will be used in the queries to filter through rooms that match the criteria. Form validation ensures that all these fields follow DB constraints and that all fields need to be filled out inorder to place a booking. The form also takes in check-in and check-out dates. This will be used in the next set of queries to check if any of the filtered rooms have pre-existing bookings that overlap with the user's check-in or check-out dates. Error messages, warning messages and success messages will be displayed throughout to give the user a sense of what is going on.

Once the user is presented with results, they can choose to add them to their favourites to come back and book later or book immediately. Adding and removing to favourites just involves toggling the star button and the appropriate content is displayed under the Favourites section of the app. For booking one of the rooms, the user has to enter their name and email that again go through form validation checks to be a valid e-mail and so on. Then, the query is executed to check whether this customer is already a BookBookGo member - if they are, a new booking under the same customer ID is created through a POST query. If not, a new customer is created in the backend and then a corresponding booking too. A success message is then displayed with all booking relevant details and users can check how their booking affected stats under the stats page and go through their bookings under their MyBookings page!

My Bookings

On the **My Bookings** page, the user will be able to see all of their bookings, including past, present, and future. Currently, there is an input field and accompanying button used to re-query the database for the bookings of the entered user by their ID. This is for demonstration purposes so that we may demo the ability to display and interact with a different user's set of bookings.

The user's bookings will be displayed neatly in a table, showing information including their check-in date, check-out date, room number, and number of guests. Additionally, there is another column to the right, where the user may assign a rating to a past visit using the 5-star system. Users may choose to show or hide inactive bookings by clicking on the corresponding button above the table to toggle this setting (an **active** booking is one that has not yet come to pass and has not been cancelled, and an **inactive** booking is any booking that is not such).

Active bookings can be adjusted by clicking on the edit icon that will appear to the left of that booking's table row. When clicked, a modal will pop up with fields to change the details of the chosen booking. Note that there are a number of front-end validation checks in place to ensure no bad data is put into the booking:

- The number of guests must be non-negative and non-zero
- The check-in date must not be today or earlier (you cannot make a booking for the same day!)
- The check-out date must not be before the check-in date (your visit must be for at least one day!)

Clicking on "Update Booking" will send a request to the database to update the booking with the given parameters.

In addition to "Update Booking", the modal has a "Cancel Booking" button. Clicking this button will send a request to the database to set the status of the chosen booking to be **"cancelled"**. Note that there is no way to un-cancel a booking!

After sending the request to the database, the app waits for a response. Upon success, the modal will close (if it was open) and the table will be updated to reflect the changes made. Upon failure, the modal will close (if it was open) and an error snackbar will pop up at the bottom of the screen indicating a booking conflict was encountered.

Statistics Page

The idea of this page is to provide users with some more information about our hotel that could help them decide to create a booking.

These statistics are queried from whenever the user navigates to this page, which means they are always up to date with the latest changes made to the database.

We present the following 4 statistics:

1. **Hotel average rating**: Represents the average rating compiled from our guests reviews.
   We calculate this by querying the average of all our guests' ratings.

2. **Rooms available**: Represents the number of non-booked rooms at the hotel.
   This query involves two steps.
   Step 1: Obtain the number of active bookings for today.
   Step 2: Subtract this number from the total amount of rooms at the hotel.

3. **Customers served so far**: We proudly display how many customers have chosen to book with us.
   This query sums the list of non-cancelled booking entries in our database.
   In this case, we assume that one booking in our database maps to one customer, regardless of the number of guests listed in their booking.

4. **Most popular view**: This is calculated from the type of view that has the most bookings in our database.
   We retrieve this information by querying all our booking records, grouping them by view. Then, we simply output the view group that contains the most records.

Summary

To summarize, our hotel booking app broadly has three categories that allow users to check various statistics about a hotel, then choose what kind of room they want through searching and filtering. Then they can choose to favourite those results for later or book them immediately. Once booked, it shows up in their bookings where they can view past bookings and update or cancel upcoming ones. This was achieved using GCP Cloud SQL, Django REST API and a React frontend. It was a fun learning endeavor as none of us had experience with this stack before and we used Zoom, Github and Google Docs to the best of our abilities! We came out from this project with the knowledge that there are areas of our app that we can still better with the newfound understanding of databases, like trying a non-relational database to store large JSON objects, setting up rollbacks and other conditions to prevent concurrency issues, create user authentication to let customers create accounts, get images and various locations for the hotels and so on!

~ Thank You! ~