Bookings Assignment

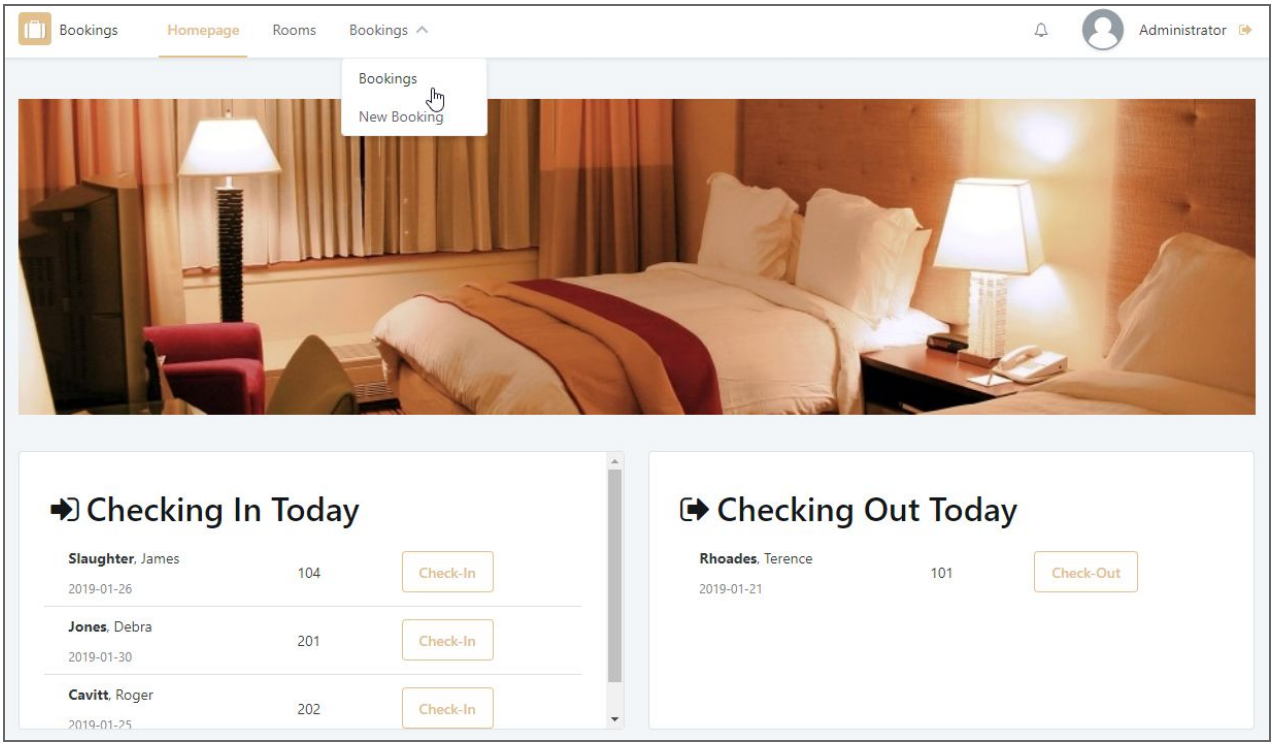


Table of Contents

Introduction

The goal for this Project is to implement the **Bookings** application to be used by the **hotel staff** to book rooms for hotel guests. Each hotel room accommodates a number of adults and a number of children, includes some amenities (Television, Internet Access, etc.) and has a price per night.

A hotel guest calls in to make a room reservation and the clerk at the reception checks for available rooms. If there is a room available, the clerk registers the reservation in the system with the guest's name, the arrival and leaving dates, alongside the number of adults and children staying in the room.

Upon arrival, the guest confirms the booking at the reception and is checked in. Upon departure, the guest checks out at the reception.

Part 1: Create the Bookings application

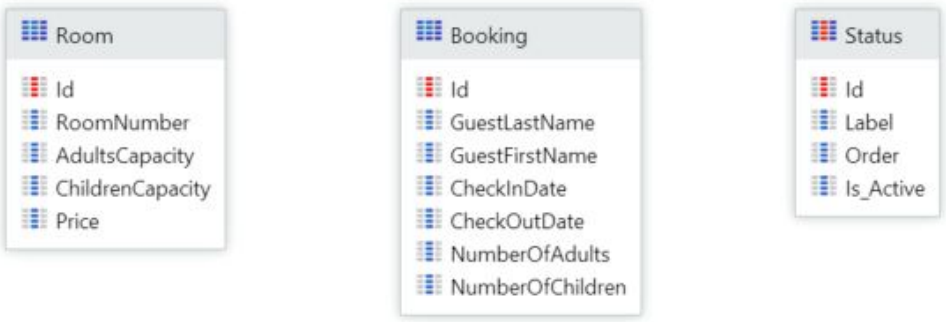
To start this assignment, the first step is to create the Bookings web application. This exercise will assume an application with a Top Menu.

In the resources of the exercise, there is an icon to be used for the application.

The application should have two modules: one Core, for the Data Model, and one for all the UI and business logic. Don’t forget that the names of the modules must be unique in the environment.

Part 2: Data Model

Now that we have the application created, we can start defining the data model of the application. The data model will be created in the core module of the application, and it will start by having three Entities:



The *Room* Entity will represent the rooms of the hotel, with:

- An unique *Id;*
- A mandatory *RoomNumber*, represented by a **Text** with **three characters**;
- Mandatory **integer** attributes for the *AdultsCapacity* and *ChildrenCapacity* of the Room;
- A mandatory *Price*.

The *Booking* Entity will represent a Booking made by a customer of the hotel, with:

- An unique *Id;*
- Mandatory **Text** attributes for the *GuestLastName* and *GuestFirstName*;
- The mandatory Booking's *CheckInDate* and *CheckOutDate*, of type **Date**;
- A mandatory **Integer** attribute for the *NumberOfAdults* and a non-mandatory **Integer** attribute for the *NumberOfChildren* included in the Booking.

The *Status* Static Entity should have the following four statuses for a Booking:

- Booked
- CheckedIn
- CheckedOut
- Canceled

The Entities should be set to Public and with write permissions.

Publish the module to the server 1

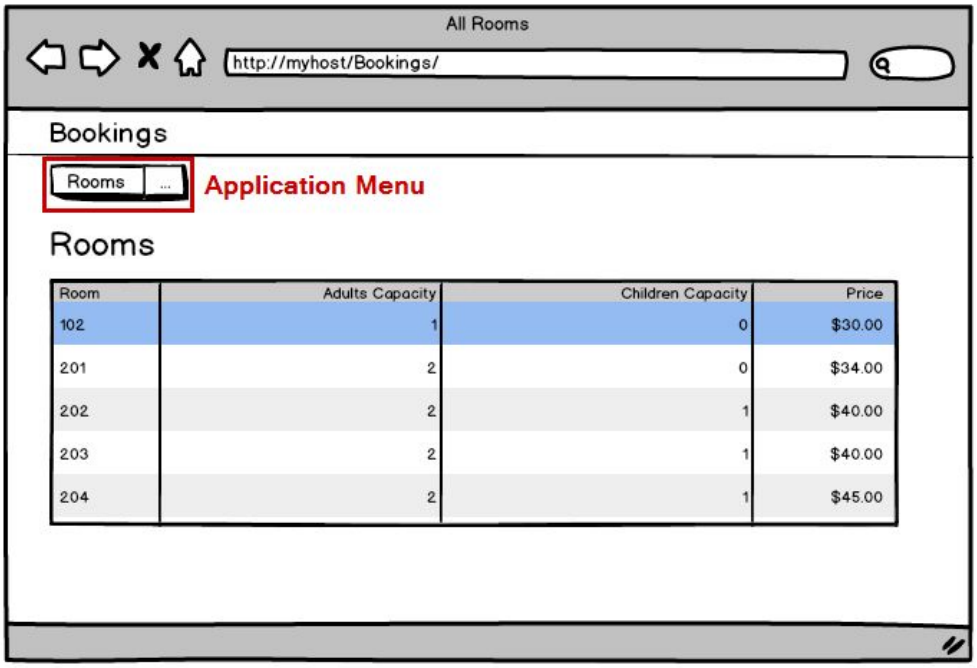
Part 3: List and Detail Screens for Rooms

In this part of the assignment, we will create the list and detail Screens for displaying all the Rooms and editing existing Rooms. When creating the Screens during this whole assignment, it is important to create everything necessary for the Screens to work, and not just only the UI.

Before we start, we should reference the Entities created in the Core module.

Rooms Screen

The **Rooms** Screen will list all the available Rooms in the Database, with the following information displayed in four columns: RoomNumber, AdultsCapacity, ChildrenCapacity and Price. The Screen should look like this



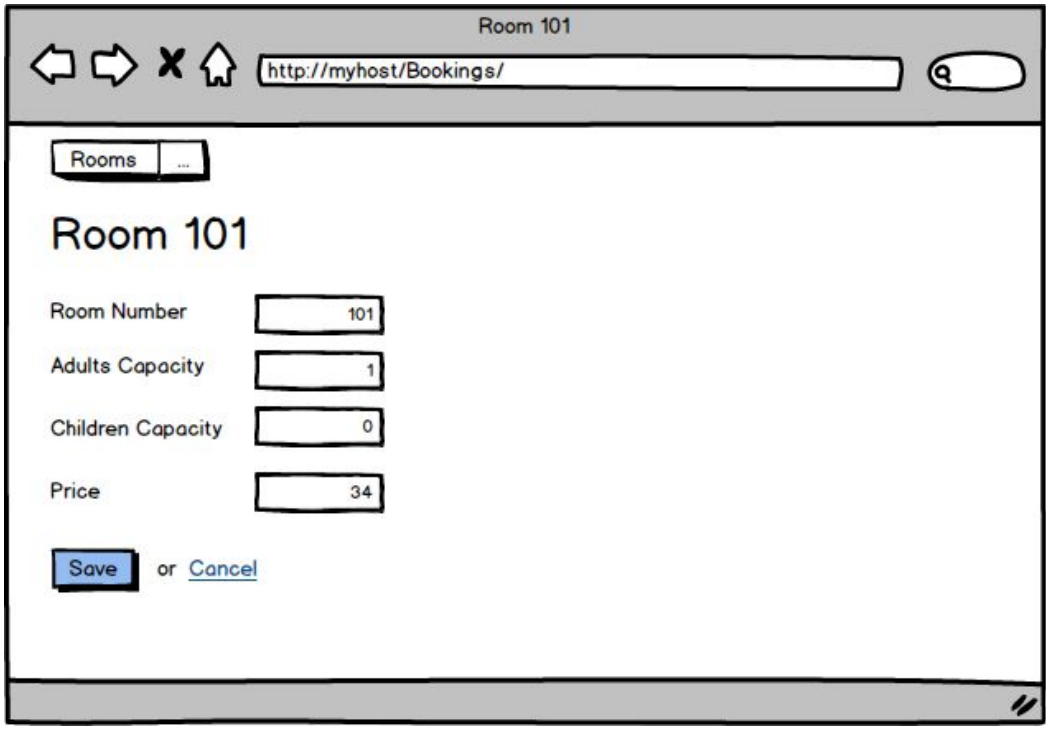
This mockup shows the application menu (with a Link to the Rooms Screen), the Screen title (Rooms) and a Table with the relevant information about the Rooms listed.

The Screen should be accessible by everyone.

NOTE: Do not use Scaffolding patterns or Screen Templates. This Assignment will also be helpful to gain experience on developing the application UI, without using accelerators.

RoomDetail Screen

The second Screen to be created is the **RoomDetail** Screen, to enable editing information about the Rooms, or if desired, to create new Rooms in the database. For that, this Screen would require an Input Parameter that identifies the Room being displayed, *RoomId*. The Screen should enable editing the same four attributes displayed in the previous in the list Screen. The RoomDetail Screen should look like the following mockup.



This Screen has a **Save** button and a **Cancel** link. The first one will save the data on the Room Form to the database, while the link will go back to the Rooms Screen.

The Screen should also be accessible by everyone.

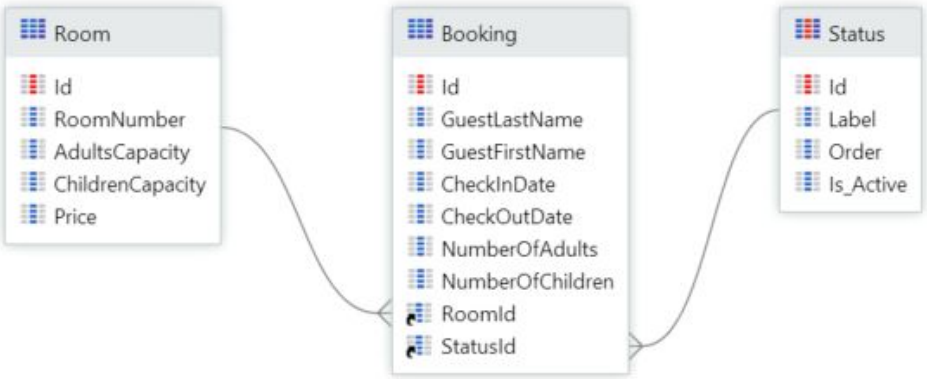
When it is done, do not forget to link the **Rooms** Screen to the **RoomDetail** Screen, meaning that every room listed in the Rooms Screen, should have a link to the RoomDetail Screen, to enable editing their information. If desired, there should also be a new Link in the Rooms Screen to allow creating new Rooms.

Publish the module (1) and test the application in the browser

Part 4: Revisiting the Data Model

In this part of the exercise, we go back to the data model in the Core module, and create relationships between the Entities. Until now, we still don't have any way to relate the **Booking** Entity with the **Room** or the **Status** Entities.

However, a Booking **must** have a Status and a Room. So, the Booking Entity should have two reference attributes, to create one-to-many relationships between the Entities: a Room Identifier and a Status Identifier.



Publish the module (1) Don't forget to refresh the dependencies in the UI module.

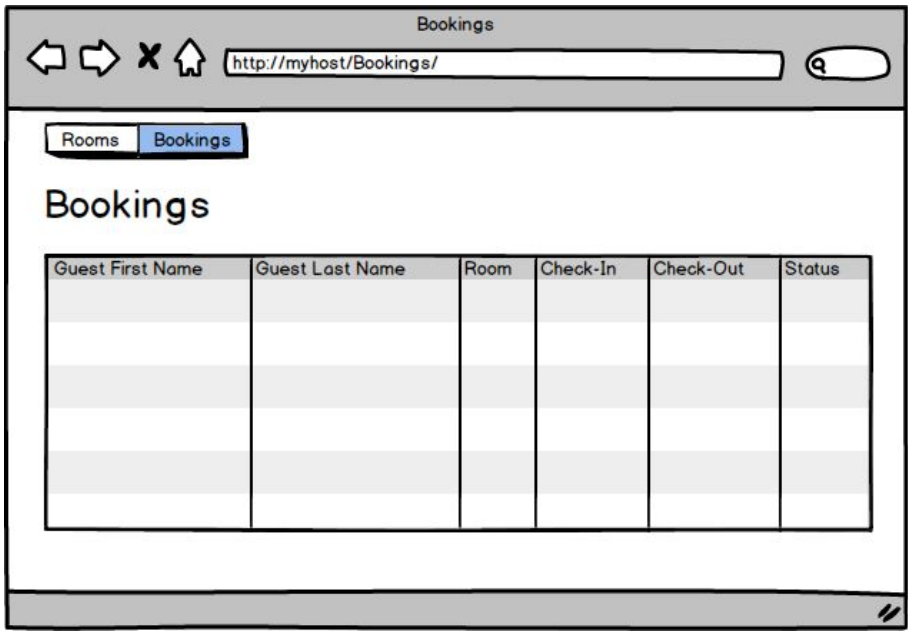
Part 5: List and Detail Screen for Bookings

The most important part of the application is to allow a hotel clerk to create Bookings for guests. For that, it is important to list the existing Bookings and to create / modify Bookings in the database.

To book a room, the clerk at the reception fills in the guest's name, the check-in and check-out dates, the number of adults and children, and then checks for available rooms that match these criteria. The application automatically assigns the cheapest room option and then the clerk confirms the booking.

Bookings Screen

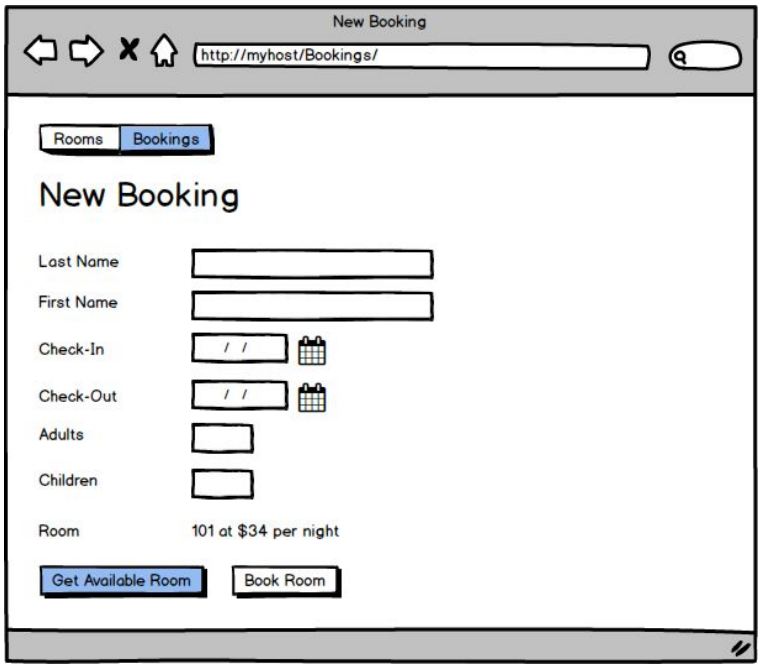
First, we will create the **Bookings** Screen based on the following mockup.



Don't forget to add a Link to the **Bookings** Screen to the Menu of the application and make it Anonymous. For now, the list of Bookings should be empty, since there are no Bookings in the database.

BookingDetail Screen

To add / edit Bookings, we will create the **BookingDetail** Screen. As before, don't forget to add an Input Parameter to the Screen, to allow editing an existing Booking. The Screen should look like the following mockup



The Screen should have a **Form** with the all the inputs in the mockup from the Last Name of the Guest to the number of adult and children guests. Since the Booking has a Room associated, don't forget to add the Room to the Aggregate in the Preparation. The **Title** of the Screen, when editing the Screen should display the **Guest Last Name**.

The final field on the Form is not an input, but an informative field. This should display the Room Number alongside its price per night. This will only be calculated when the clerk selects an available room that suits this booking.

On that note, at the end of the Screen, we have two Buttons: **Get Available Room** and **Book Room**. The first Button will trigger an Action that will get the cheaper available room, for the dates and number of adults and children selected in the Form. The second Button will also trigger an Action, to confirm the Booking and adding it to the database.

As a summary, the **GetAvailableRoom** finds the Room and saves it in memory on the Screen. The **BookRoom** saves the Booking in the database with the Room associated to it. To find the cheapest Room available, we need a SQL Query:

```
SELECT {Room}.* FROM {Room} WHERE @NumberOfAdults > 0
AND {Room}.[AdultsCapacity] >= @NumberOfAdults
AND {Room}.[AdultsCapacity]+{Room}.[ChildrenCapacity]>=@NumberOfAdults+@NumberOfChildren
AND NOT EXISTS
(SELECT 1 FROM {Booking} WHERE {Booking}.[RoomId] = {Room}.[Id]
        AND (@CheckInDate BETWEEN {Booking}.[CheckInDate] AND {Booking}.[CheckOutDate] - 1
        OR  @CheckOutDate BETWEEN {Booking}.[CheckInDate] + 1 AND {Booking}.[CheckOutDate])
       AND {Booking}.[StatusId] <> @CanceledStatus)
ORDER BY {Room}.[Price] ASC
```

Create the Links in the **Bookings** Screen to navigate to the **BookingDetail** Screen, to create new Bookings and to edit existing ones.

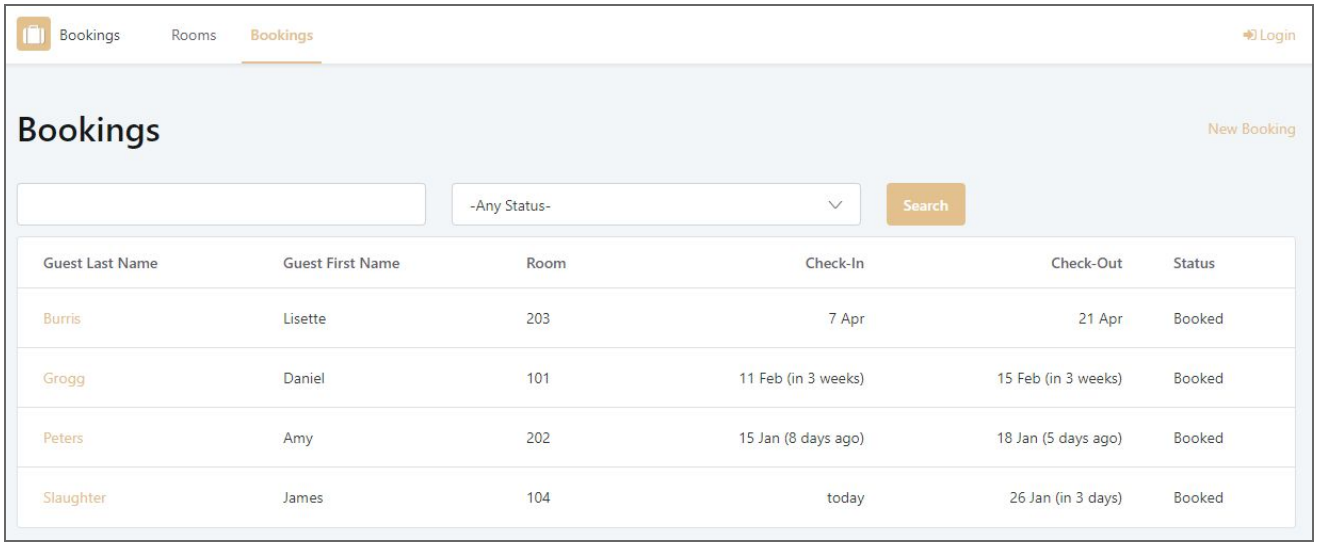
Publish the module  and test the application in the browser

Part 6: Extend the functionality

On this section of the assignment, we will extend the functionality of the application with three new features: search filters by guest last name and booking status, control the visibility of elements in the BookingDetail Screen and add amenities to hotel Rooms.

Search Filter on Bookings

In the Bookings Screen, we should allow searching by guest's last name and Booking status. When there are no filters applied, all Bookings should be displayed in the Table Records. At the end, we should have something like the following screenshot.



Publish the module 1 and test the application in the browser

Control the visibility of elements in the BookingDetail Screen

In the **BookingDetail** Screen, we have the opportunity to control the visibility of some elements on the page. In particular, when creating a new Booking, we should only display some of the content when it is relevant.

1) We need to make sure that the line that displays the Room Number and the Room Price in the Form, is only displayed after clicking on the **Get Available Room** Button and successfully finding the cheapest Room available.
2) The Book Room Button should only be displayed after clicking on the Get Available Room Button.

Publish the module (1) and test the application in the browser

Add Amenities to Hotel Rooms

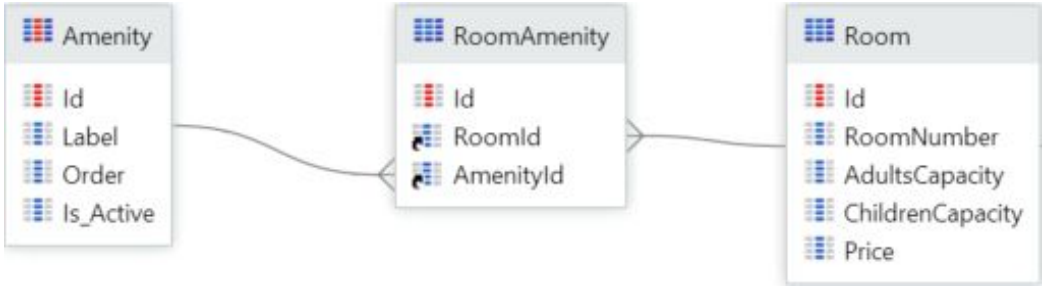
In this Section of the Assignment, we will create the logic to associate **Amenities** to the Rooms. We will be able to add multiple amenities to a Room.

To do this, first we need to, go back to the **BookingsCore** module and create a new Static Entity called *Amenity*. The following Amenities should be represented:

- Television;
- Internet Access;
- Hair dryer;
- Premium Towels;
- Crib;
- Safe.

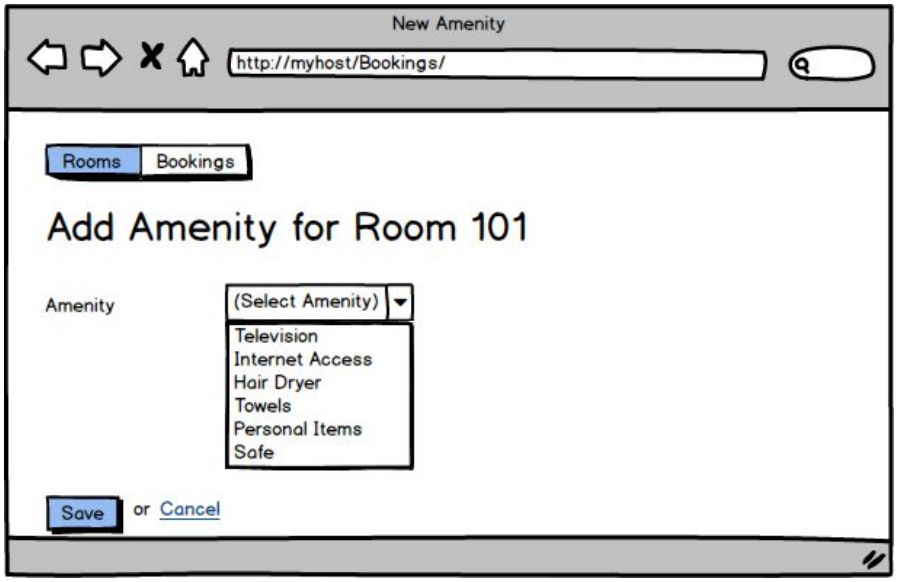
Note: Don't forget to define the **Label** property of each one of the Records.

After having this Static Entity created, we now need to relate the Amenities with a given Room. For that, we need to create a new Junction Entity (*RoomAmenity*) and define a many-to-many relationship. Also, we need to make sure that there cannot be two RoomAmenity records with the same Room and Amenity, through a Unique Index.



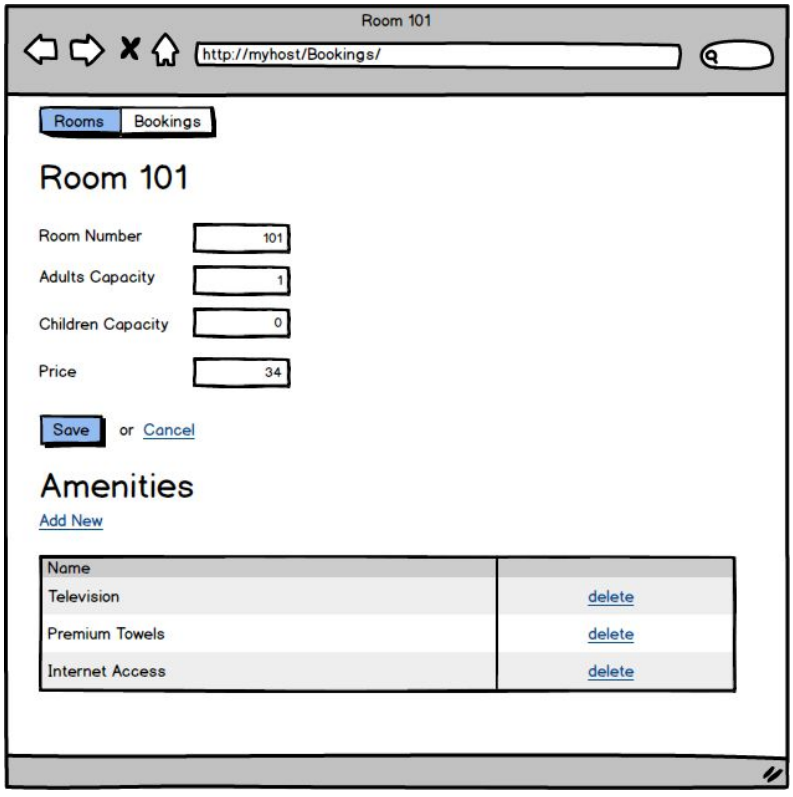
Publish the module (1) and reference the new Entities in the Responsive module.

Now, we need to create a new Screen for the Amenities (*RoomAmenities*). In the RoomAmenities Screen, implement the behavior that allows adding Amenities to a specified Room, according to the following mockup:



In the **RoomDetail** Screen, add a new Link to the **RoomAmenities** Screen.

In the original **RoomDetail** Screen, add a Table to list all the Amenities for the current Room, below the Form. Also add a Link to delete one Amenity. An example of this scenario can be found in the following mockup, with a Delete Link in every row to allow removing an Amenity at any time.



Note: When adding a new Room, instead of editing one, make sure that the Add Amenities option is not available.

Extra Challenge: To add several Amenities to a Room we're forced to navigate multiple times between the RoomAmenities Screen and the RoomDetail Screen. To change this, let's create an additional **Save & New** Button in the **RoomAmenities** Screen. This way, instead of going back to the RoomDetail Screen, simply clear the input to allow you to add a new Amenity to the Room, without closing the Screen.

Publish the module  and confirm that the app works properly in the browser.

Part 7: Check-In, Check-Out and Cancel Bookings

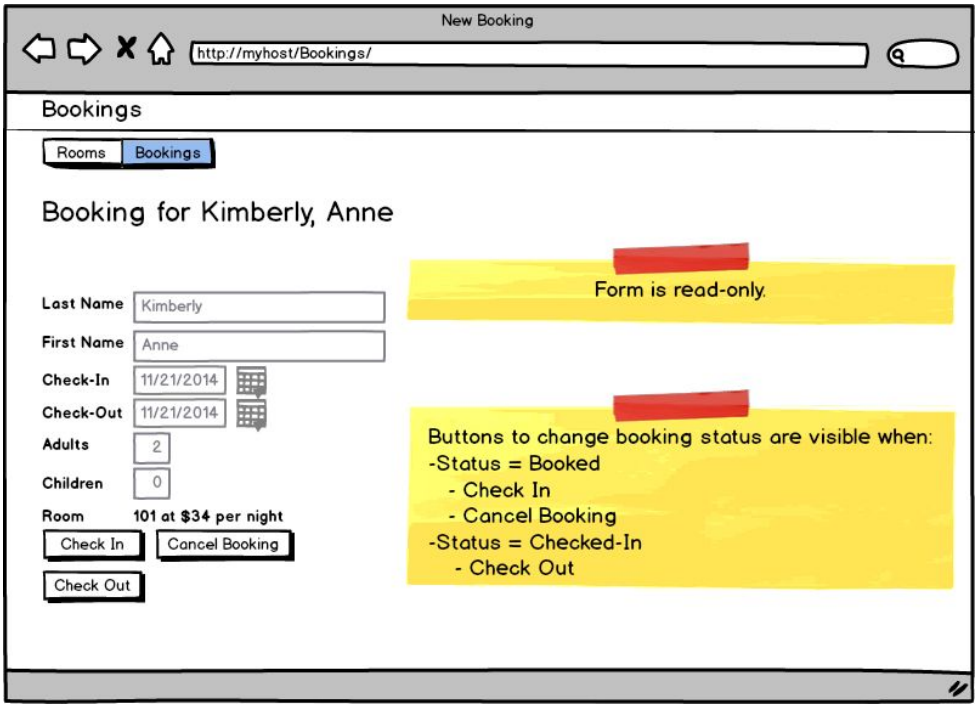
In this part of the assignment, we will create UI and logic to allow the hotel clerk to **check-in** and **check-out** guests, and also to **cancel** a Booking.

This will be done in several parts. First, we will change the functionality of the **BookingDetail** Screen, when editing an existing Booking, to allow checking-in, checking-out or cancel. Then, we will create a **Homepage**, where it will be visible the Check-ins and Check-outs of the day, with a Button to directly check-in or check-out.

This will lead to some extra changes as well. We will change the Form on the BookingDetail to only allow edition while the Booking is not confirmed. Also, we will create a reusable Action to change the status of a Booking.

BookingDetail Screen

1) Change the **BookingDetail** Screen to make it look just like the following mockup



Don't forget to check the visibility of the Buttons and when the Form is editable or not on the yellow boxes in the mock-up.

2) Create a generic Server Action to change the status of a Booking. This Action should be generic and should be able to change the Booking to any possible status.

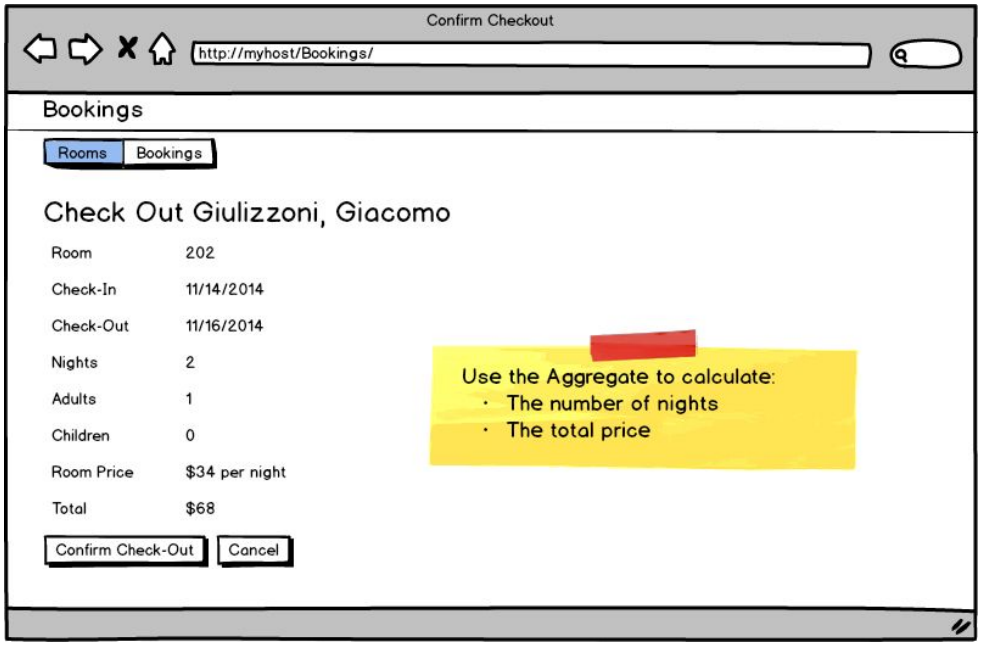
3) Implement the logic of the three new buttons, to change the status of the Booking accordingly, reusing the Action created in the previous step.

Publish the module (1) and confirm that the app works properly in the browser. Follow the next steps to test if your application is behaving properly.

1) Add three Bookings for the guests Jack Daniels, John Doe and Mary Jane.
2) Check in Jack Daniels and John Doe.
3) Did it work as expected? Both Bookings should be checked in.
4) Cancel Mary Jane's Booking.
5) Can you check in Mary Jane now?
6) Try to cancel John Doe's Booking. Since John Doe has already checked in, the **Cancel** Button should not be visible, thus you should not be able to cancel the Booking.
7) Check out Jack Daniels.

Confirmation Message before Checking Out

In the **BookingDetail** Screen, we will now change the Check-Out Action to go to a new **BookingCheckout** Screen, where the number of nights and the total payment of the Booking are calculated and displayed to the user.

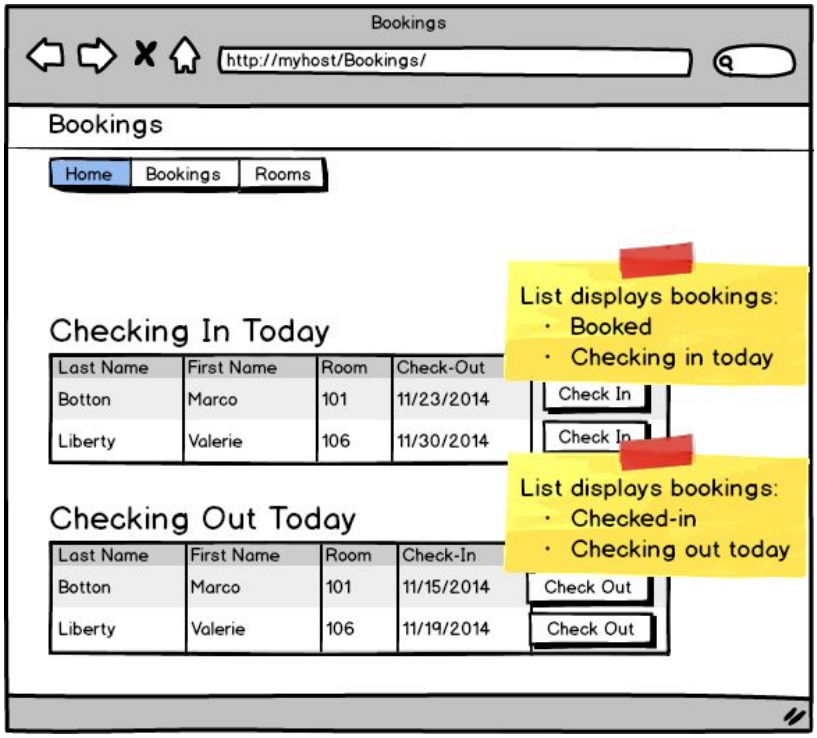


The **number of nights** is calculated using the CheckOutDate and the CheckInDate. The **total price** considers the number of nights and the price of the room per night.

The Confirm Check-Out Button actually performs the Check-out.

Create a Homepage

Create the **Homepage** Screen like the following mockup. Don't forget to add the Homepage to the Menu of the application and make it the default entry Screen.



Make sure that only the Bookings that will check-in and check-out today appear. Also, when the Check-in / Check-out is made, the Booking should disappear from the list.

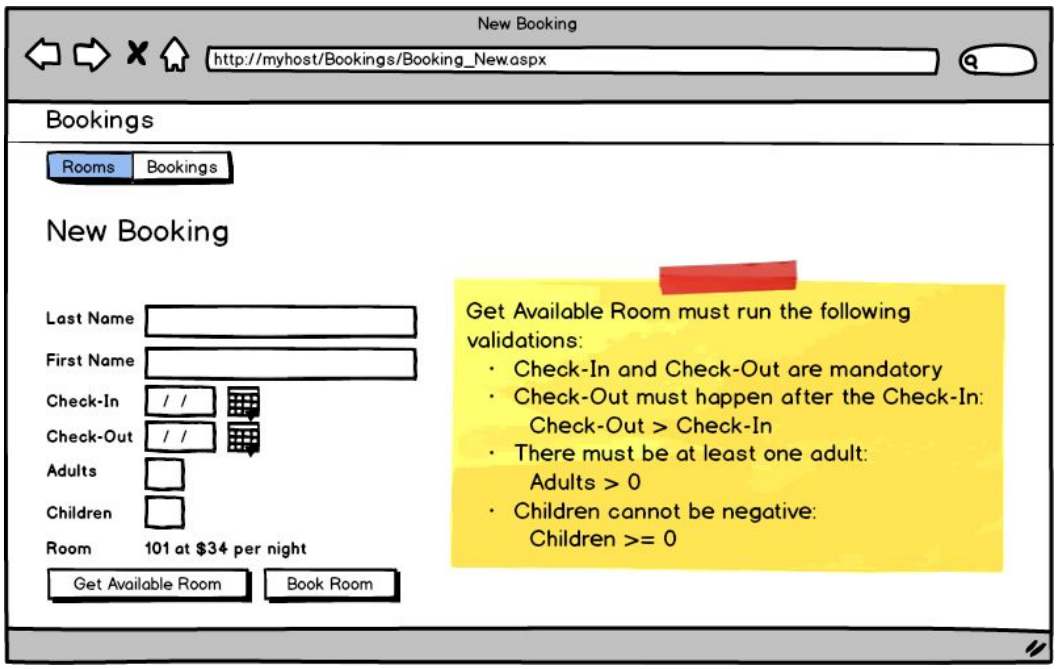
Publish the module (1) and confirm that the app works properly in the browser.

Part 8: Bookings Input Validation

In this section of the assignment, we will avoid sending incorrect data to the Database, when filling information for new Bookings. We need to ensure that when checking for an available room, the user provides valid information according with some specific business rules.

Validate User Inputs using Server-side Validations

The following mockup specifies some validations that the application needs to check. Make sure that the scenarios in the yellow boxes are applied. Note: It is possible to create Bookings in the past.



Publish the module (1) and confirm that the app works properly in the browser. Follow the next steps to test if your application is behaving properly.

1) Test the validations following the next steps.
    a) Try to Book a Room for a Guest:
        - **Check-in**: today
        - **Check-out:** yesterday
        - **Adults**: 1
    b) Are the validations working? Great!
    c) Try to book a room:

- **Adults**: 0
- **Children**: 3

d) Are the validations working? Great!

2) Now, let's try the Client & Server Validations

a) Change your validations to Client & Server.

b) Publish the module  and open again the application in the browser.

c) Try to book a room:

- Don't fill-in the Check In field
- Don't fill-in the Check Out field

d) Are client-side validations working? Great!

e) Change the Check In date to "Jack Daniels".

f) Are client-side validations working? Great!

3) Follow the next steps to create some Bookings in the Database.

a) Book a room for John Doe:

- **Check-in**: today
- **Check-out**: a week from today
- **Adults**: 2

b) Was the booking successful? Great!

c) Were you directed to the **Bookings** Screen after creating the booking? Great!

d) Book a room for Mary Jane:

- **Check-in**: tomorrow
- **Check-out**: the day after tomorrow
- **Adults**: 2

e) What has happened? Mary Jane had a new Room assigned to its Booking, different from the one assigned to John Doe.

f) Book a room for Jack Daniels:

- **Check-in**: today
- **Check-out**: tomorrow
- **Adults**: 1
- **Children**: 2

Clean Invalid Bookings

At this point of the exercise, and since we just added the input validations, some data that was previously created may be incorrect or doesn't fulfill all the necessary conditions. To overcome this problem, a new functionality to delete **Booking** should be added:

1) In the **BookingDetail** Screen, add a new **Delete** Link to the Actions placeholder on the top right of the Screen.
2) Add the following Confirmation Message to this Link, to make sure the User is certain of what will happen: "Are you sure you want to permanently delete this Booking"?
3) In the Screen Action associated with this Link, simply delete the current Booking from the Database.

Part 9: Add Ajax to the Bookings

At this point, if the Bookings application already submits data to the server using the Ajax Submit method, this section can be skipped.

We should add Ajax behavior to the following parts of the application:

1) In the **BookingDetail** Screen, refresh the room information with Ajax, when the **Get Available Room** Button is pressed. Use the *Highlight* **Animation** effect.
2) In the **Homepage** Screen, review the logic of the **Check In** Button to only remove the respective line from the Table Records. Use the *Highlight* **Animation** effect.
3) In the **RoomDetail** Screen, change the logic of the **Delete** Link to use Ajax.

Publish the module (1) and open the application in the browser.

To test if this works, we can follow the next steps:

1) Find a Room for Lewis Nash:
   - **Check-in**: today
   - **Check-out:** the day after tomorrow
   - **Adults**: 2
   - **Children**: 1
2) Only the room information was refreshed? Good!
3) Did the **Book Room** Button appear? Why not? Go back and fix it.
4) Go to the **Homepage** and check in Lewis Nash.
5) Was he removed from the list with Ajax? Nice!
6) Find a Room for Amelia Jacobson:
   - **Check-in:** tomorrow
   - **Check-out**: today
   - **Adults**: 1
   - **Children**: -1
7) Did the validations appear on the page? What happened for this to change? Go back and fix it.

Part 10: Add a Web Block to Display the Guest Name

In this section of the assignment, we will create a Web Block to display the Guest Name in a different way.

This Web Block should contribute to standardize the way the guest name is displayed throughout the application, using the following look and feel

Montgomery, Jane

Millard, Anthony

Stuart, Pamela

Baker, John

This Web Block should be used in the Tables where the guest name is displayed. After that change, we can reduce the number of columns of the Table Records.

Publish the module (1) and open the application in the browser.

Part 11: Secure Bookings

At this point of the assignment, we will add some Role-based Security to the application.

At the end of this section, the Bookings app should meet the following requirements:

1) Restrict the access to all Screens to just users with the **Bookings_User** Role. This Role need to be created first.
2) Create a new User for the application in the *http://<your_server>/Users/ app.*
    a) Login with the user. Still have no access. Why?
    b) Grant the new user with the **Bookings_User** Role.
    c) Can the user access the application now? Great!
3) Create the **BookingsResponsible** Role and make sure that only users with this Role can do the following:
    a) Access the **BookingDetail** Screen.
    b) See the **New Bookings** Link in the **Bookings** Screen.
    c) Publish the module  and open the application in the browser. Make sure that the application works as expected.
4) Give back the access to users with the **Bookings_User** Role to the **BookingDetail** Screen. However, make sure that only **BookingsResponsible** users can delete and cancel Bookings.

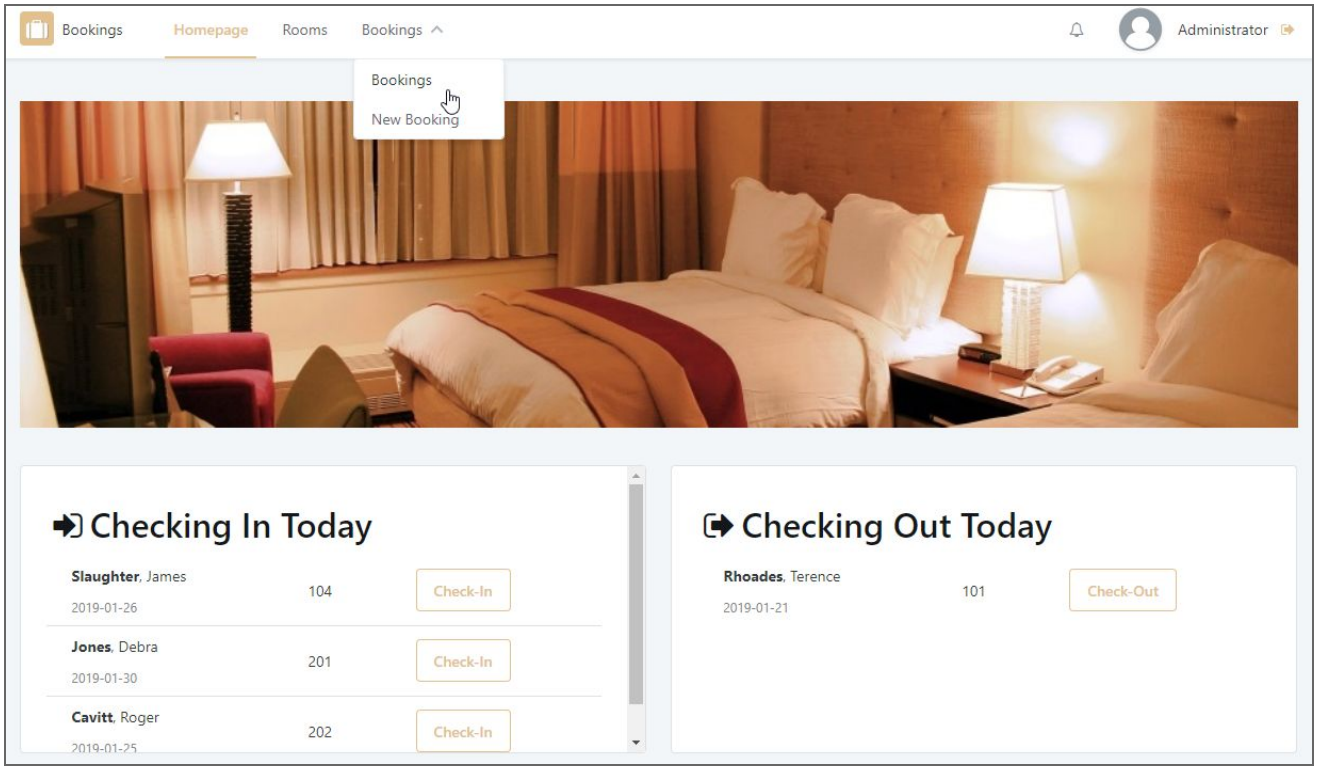
Publish the module  and open the application in the browser. Make sure that the application works as expected.

Before we move on, adjust the search filters to use **Session Variables.**

Publish the module  and open the application in the browser. Make sure that the application works as expected.

Part 12: Improve Bookings UI

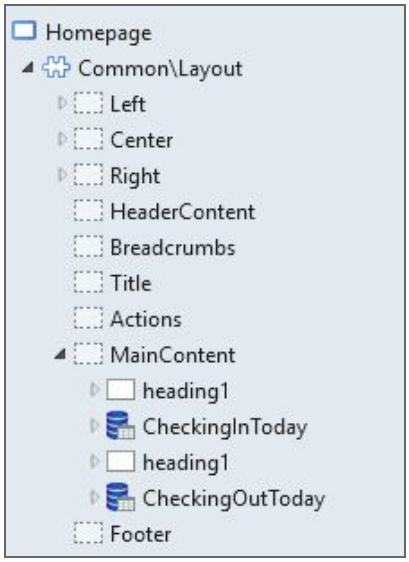
On this section of the assignment, we will do several tweaks to the look and feel of the application, to make it look nicer. At the end of this section, the application should look a bit like the following screenshot



Display the Check-In and Check-Out Lists Side by Side

First, we will change the Homepage Screen to display the Check-In and Check-Out Table Records side by side.

Before we start, we need to make sure that the Homepage widget tree looks like the following screenshot



To make the Table Records appear side by side, we need to enclose each combination of heading1 + Table Records in Containers with 6 columns.

Then, we want to limit the height of both lists. To help us with that, we should use the following CSS Style Class, and apply it properly:

```
.GuestList{
    height: 270px;
    overflow: auto;
}
```

Publish the module (1) and open the application in the browser. Make sure that the application works as expected and the lists appear side by side.

Adjust the Homepage Headings

Now that the Table Records appear side by side, let's improve the Headings for the Tables, adding some icons.

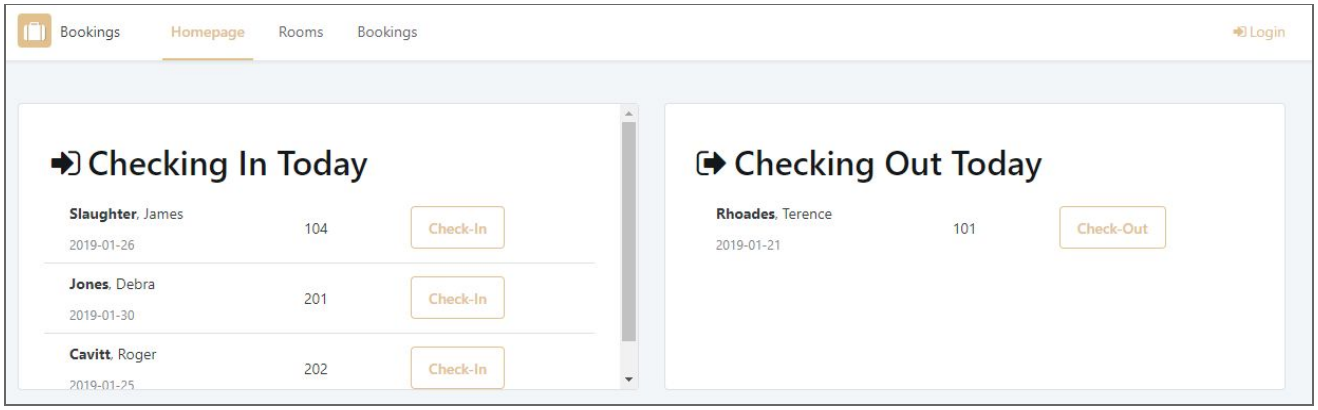
For the icons, we can use the *sign_in* for check-in and the *sign_out* for check-out.

Create Contrast and Improve Readability

Now, we want the columns to stand out. For that, we should apply the Card style to the Containers, to make sure they appear with a white background.

Also, make sure that the Check-Out Date (for the Checking In Today) and the Check-In Date (for the Checking Out Today), appear below with the Guest Name with the following Style:

```
.TextNote{
    font: 12px;
    color: #838689;
}
```



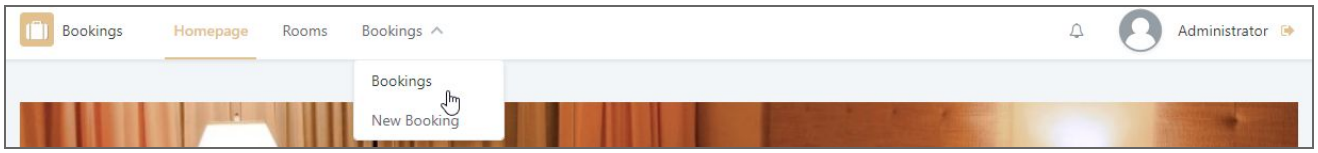
Use an Image

Add an Image above the Checking In Today and the Checking Out Today lists, and make sure that its width takes 100% of the available Screen width.

Add some spacing between the image and the tables.

Promote the Most Common Operations in the Menu

Finally, we will promote some common operations to the menu, as submenus. The Bookings option should open submenus with Links to the List of Bookings (**Bookings** Screen) and to create a new Booking (**BookingDetail** Screen).



Publish the module 1 and open the application in the browser. Make sure that the application works as expected and the lists appear side by side.

Part 13: Extend the Functionality with RichWidgets

In this section, we will add extra functionality using some **RichWidgets** and **Charts**. We will start by adding pagination and dynamic sorting to the Bookings page. Then, we will add some Feedback Messages to the users. Finally, we will display in the Homepage the occupancy of the hotel for the next 7 days.

Pagination and Dynamic Sorting on the Bookings Screen

Add the **List_Navigation** to the **Bookings** Screen. When the search is applied, we need to make sure that the navigation is reset to the first page.

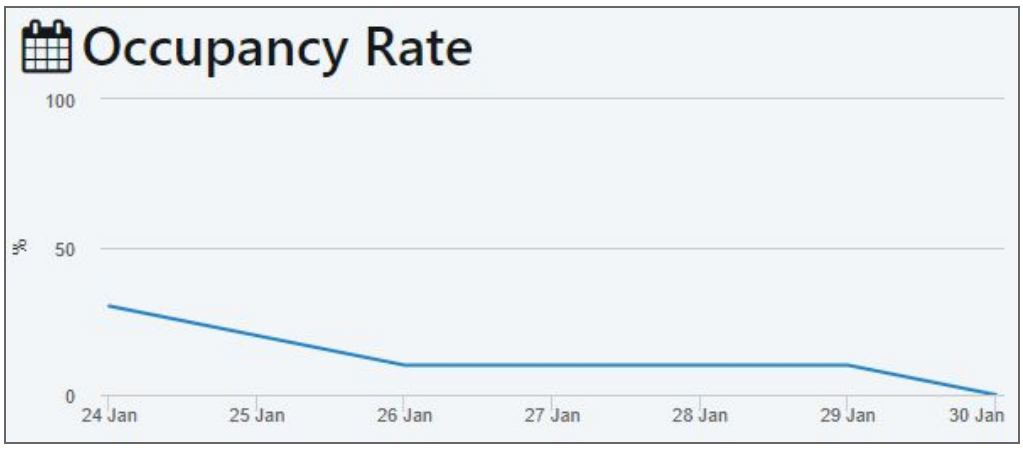
Also, add some dynamic sorting to the Bookings Screen. Make sure that the user can sort by Guest Name, Check-In Date and Check-Out Date. Adjust the Aggregate accordingly.

Publish the module 1 and open the application in the browser.

Occupancy for the Next 7 Days

In the **Homepage** Screen, display the hotel occupancy. This will allow the hotel clerk to plan the next days, and better allocate the available resources. The Chart should display **the hotel occupancy,** in percentage (Y-axis), **for the next 7 days** (X-axis).

At the end of this section, we should have a Chart looking like this:



1) Create the layout for the **Occupancy Rate** title.
2) Use a **Line Chart**, and set its height to *200*.
3) In the Preparation, create the logic to build the data to be plotted on the chart:

a) Get all Rooms in the hotel, to calculate the percentage of occupancy.

b) Use a **Local Variable** to hold the Date to which the occupancy is calculated. The Variable should start on today's date.

c) If the Date in the Local Variable is within the next 7 days:

   i) Get all Rooms Booked or Checked In on the date.

   ii) Build a **DataPoint** with the percentage of occupancy for the date.

   iii) Append the **DataPoint** to a List of Data Points.

   iv) Add one day to the date on the Local Variable.

   v) Define the flow to return to the If.

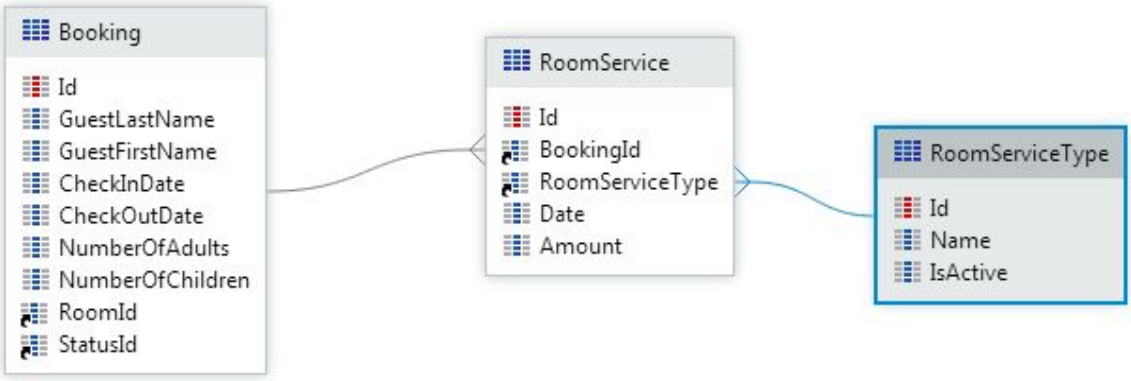
d) Pass the List of Data Points to the Chart.

e) Change the look of your Chart so that the Y-axis always displays values between 0 and 100 and has its labels suffixed with '%'. You can use the **YAxisFormat** property of the chart. To learn how to specify the **YAxisFormat**, checkout the **YAxisFormat_Init** Action online documentation here.

Publish the module  and open the application in the browser. Make sure the Chart works fine.

Part 14 (Extra): Room Service

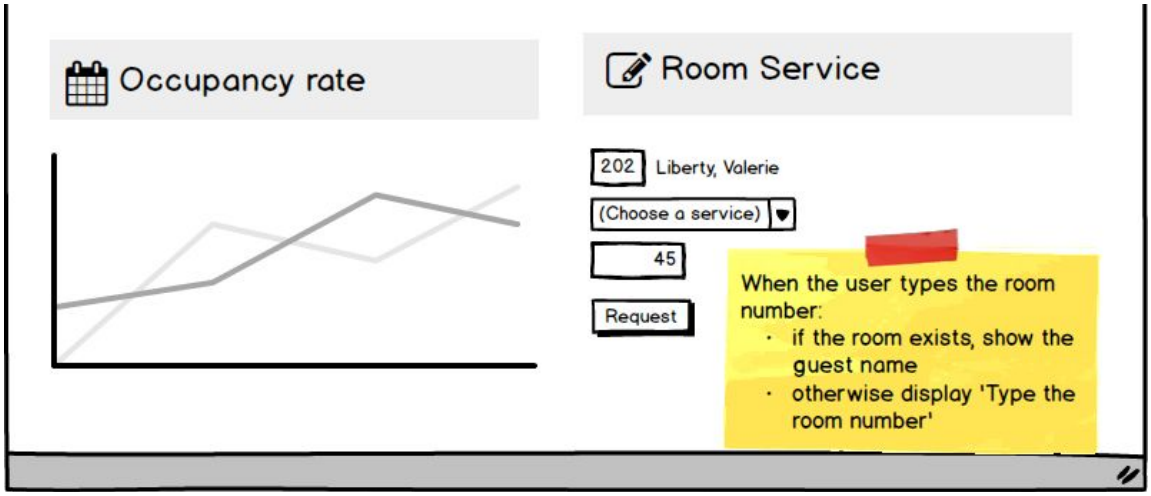
In this extra section of the assignment, we will create a Room Service functionality. In the Homepage of the application, we will have a section to request a service to a Room. However, before we move to the UI, we need to change the data model.



All attributes are mandatory. These changes should be made in the **Bookings_Core** module.

There is a **RoomServiceType** excel sheet with data that can be bootstrapped.

With the data model defined, we need to define the UI in the Homepage, just like the following Screenshot.



This Room Service section will ask for the Room Number. If the Room has someone checked-in, the guest name should appear in front. Then, the service can be selected, as well as the amount and the request is made. The OnChange Event of the inputs can be useful, to have an immediate reaction to what the users select and type.

At the end, in the Booking Checkout, the Room Service charges should appear and should be added to the Total Price of the Room.