|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Workspace Booking Management Portal**    **FSE – Business Aligned Project**  **Case Study Specification**    **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Suja Rajasekharan |  |  | | **Role** | Trainer |  |  | | **Signature** | t-suja1 |  |  | | **Date** | 20 October 2022 |  |  | |
|  |

**Table of Contents**

|  |  |  |  |
| --- | --- | --- | --- |
| 1.0 |  | Important Instructions | 1 |
| 2.0 |  | Introduction | 4 - 5 |
|  | 2.1 | Purpose of this document |  |
|  | 2.2 | Project Overview |  |
|  | 2.3 | Scope |  |
| 3.0 |  | Use Case Diagram | 6 |
| 4.0 |  | System Architecture Diagram | 7 |
| 5.0 |  | Development Phase | 8 |
| 6.0 |  | System Requirements | 9 - |
|  | 6.1 | User Service |  |
|  | 6.2 | Admin Service |  |
|  | 6.3 | Room Booking Service |  |
|  | 6.4 | Recurring Room Booking Service |  |
| 7.0 |  | Deployment Requirements | 20 |
| 8.0 |  | Design Consideration | 20 |
|  | 8.1 | Most Important Common Rules |  |
| 9.0 |  | Reference Learning | 22 |
| 10.0 |  | Project Template | 23 |
| 11.0 |  | Change Log | 24 |

1. Important Instructions
2. Associate must adhere to the Design Considerations specific to each Technology Track.
3. Associate must not submit project with compile-time or build-time errors.
4. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development.
5. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Unit testing and Mocking Frameworks wherever applicable.
6. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only.
7. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file.
8. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.
9. (Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)
10. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
11. You are supposed to use an In-memory database or code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.
12. Introduction
    1. Purpose of the Document

The purpose of the software requirement document is to systematically capture requirements for the Meetings. The Digital Workspace Management System provides the employees to book meeting rooms and it enhance the communication among employees to optimize space management. Employees can book meeting rooms internally as well as externally according to the room details provided by the Admin.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specified wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

* 1. Project Overview

Managing employee work areas is more important especially in pandemic situations. Whether it involves saving a spot in the office or at a coworking centre. Workspace Management tool helps to efficiently manage the workspace as well as manage the digital spaces and enhance the communications both internal and external to optimize space management.

The Workspace Management System helps to reserve conference rooms, other spaces within the office or shared workspace.  The application ensures the organizers and attendees get the accommodations for the important meetings and at the sometime administrators can get the updated information of resources being utilized. Through this they can limit the access as well as restrict to ensure the appropriate usage of the resources. Employees can book a meeting room or desk for different types of meetings like team meeting, sales calls, one-on-ones and brainstorming sessions. This tool improves workplace efficiency while preventing double bookings or other situations where meeting spaces and resources are unavailable at critical moments.

* 1. Scope

Below are the modules that needs to be developed part of the Project.

**Application Module**

|  |  |  |
| --- | --- | --- |
| REQ . NO | REQ NAME | REQ DESCRIPTION |
| REQ 1 | Admin Module | * This module will be managed by the admin. * Admin can add new rooms details i.e internal as well as external * Get the status of the rooms and update it as per the requirements * The admin will add the details of the available ammenitites in the meeting room like TV,Projector, Wifi etc.. |
| REQ 2 | Meeting Room Booking | * The details of the venue will be displayed as a list . * The user will select the required venue and book the meeting room. * Once they enter the details then it should display the booking summary details with the total amount. |
| REQ 3 | Recuring Meeting Booking | * The user can choose a plan for daily, weekly, monthly or quarterly. * Once they enter the details like start date and end date, they will be able to see the booking summary of it. * According to the plan what they choose, user will get the bill details. |
| REQ 4 | User Information | * The user has to register to book the room. New user’s need to be registered and existing users can enter the credentials and proceed for the booking. |

 3.0 Use Case Diagram

The following use case diagram shows various users of the system and their responsibilities.

User

Admin

4.0 System Architecture Diagram

Front End

Admin Service

Recurring Room Booking

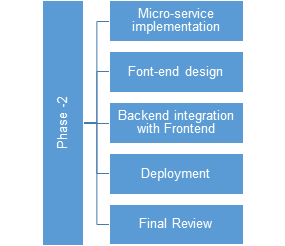
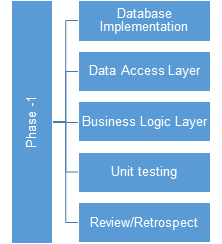
Room Booking Service

User Service

Back End

5.0 Development Phases

* The application will be developed in 2 phases.
* Each phase will have 4 stages followed by a review at the end.
* The phase-1 output will be unit tested core business logic of the application.
* In phase-2 the output will be a functional application with micro-service and the font-end.
* Each stage of the development phase must be completed alongside the learning milestone



1. System Requirements
   1. **Module – User Details**

The User Management module will allow to manage the user details. The module will provide the following features.

1. Creation/Update of new user
2. Display the details

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

UserInfo

|  |  |
| --- | --- |
| Column Name | Type |
| UserID (PK) | Integer (10) |
| UserName | Varchar (25) |
| Password | Varchar (25) |
| FirstName | Varchar (25) |
| LastName | Varchar (25) |
| PhoneNo | Integer (10) |
| EmailId | Varchar (25) |
| CompanyName | Varchar(100) |
| Address | Varchar (100) |

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
2. Username, first name and last name must be minimum 3 characters long

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
4. Add New User
5. Return the details of newly added user
6. Update a newly added user
7. Existing user can login with the credentials

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
5. Add New User
6. Return the details of newly added user
7. Update a newly added user
8. Following business rules must be implemented as part of the business service class

* Phone number should be exactly 10 digits long

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

**Table 1 : User Details - Endpoint - 1**

|  |  |
| --- | --- |
| **URL** | /api/NewUser |
| **Request Type** | Post |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | Endpoint will be responsible to add new user details |
| **Inputs** |  |
| **Outputs** | UserDetails |

**Table 2 : User Details - Endpoint - 2**

|  |  |
| --- | --- |
| **URL** | /api/UserDetails |
| **Request Type** | Get |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | Endpoint will be responsible to get the user details. Once the user submits details, he/she will be able to see the details below with edit option. |
| **Inputs** |  |
| **Outputs** | UserDetails |

**Table 3 : User Details - Endpoint - 3**

|  |  |
| --- | --- |
| **URL** | /api/EditUserDetails |
| **Request Type** | Put |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | Endpoint will be responsible for edit the user details. Once the details display the user will be able to edit his/her details if they need to update it. |
| **Inputs** |  |
| **Outputs** | UserDetails |

**Stage: Front-end design**

Create the following components as per the specification provided below.

1. Create User Component

1. Create a component to add the new user details
2. The component should provide a form for the user
3. Once all the user details are validated, user should be able to submit the form and get an acknowledgement.
4. UserList Component
5. Develop a component which will be used to list the user details. Provide a navigation to the Edit component from the UserList Component
6. The component should display the newly added user details in the form.
7. Upon saving the status successfully display an acknowledgement.
8. EditUser Component
9. Once the user submits as a new user through create user component, the details will be displayed for the user with edit option.
10. User will be able to update the details.
11. Before submitting the form ensure that all fields are validated
12. Once the form is submitted successfully, display an acknowledgement.

4. LoginPage Component

1. User can give the username and password and login to the application to book the rooms.
2. Once they log in with credential, log id should be available till the booking.

**Stage: Integration of Frontend and backend**

1. Create a data service in the front-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

**6.2 Module – Admin Service**

In this module, admin will be entering details about the room. He will get the details of the room status. He will be able to update the details of the room.

1. Create new Room details
2. Update the room details
3. Get the list of room status
4. Add the new ammenities
5. Update the details of ammenities

**Stage: Database Implementation**

Design a data base as per the following ER diagram provided.

Table: RoomDetails

|  |  |
| --- | --- |
| Column Name | Type |
| Room\_Id (PK) | Integer(10) |
| Room\_Name | Varchar(50) |
| Address | Varchar(100) |
| PhoneNo | Integer(10) |
| NoofSeating | Integer(5) |
| Image | VarBinary(max) |
| RatePerHour | Decimal |
| Status | Varchar(15) |

Table: Ammenities

|  |  |
| --- | --- |
| Column Name | Type |
| AmmenityID (PK) | Integer(10) |
| AmmenityType | Varchar(25) |
| Available | Boolean |

**Note :** Initially set the status column as available for RoomDetails and Ammenities table ‘Available’ Col default set as true.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
4. Create new Room details
5. Update the room details by RoomID
6. Get the list of room status
7. Add the new ammenities
8. Update the details of amenities by AmmenityID

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes using the single responsibility principle which perform the given operations as follows
5. Create new Room details
6. Update the room details by RoomID
7. Get the list of room status
8. Add the new ammenities
9. Update the details of amenities by AmmenityID

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

**Table 4 : Admin management - Endpoint - 1**

|  |  |
| --- | --- |
| **URL** | /api/AddRoom |
| **Request Type** | Post |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to provide to add room details in the database |
| **Inputs** | RoomName, Address, PhoneNo, NoofSeats, Image, Status, RateperHour |
| **Outputs** | RoomDetails |

**Table 5 : Admin management - Endpoint - 2**

|  |  |
| --- | --- |
| **URL** | /api/UpdateRoom |
| **Request Type** | Put |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to provide to update room details in the database |
| **Inputs** | RoomId |
| **Outputs** | UpdatedRoomDetails |

**Table 6 : Admin management - Endpoint - 3**

|  |  |
| --- | --- |
| **URL** | /api/GetRoomStatus |
| **Request Type** | Get |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | This endpoint will display the roomdetails with status from the database |
| **Inputs** |  |
| **Outputs** | RoomDetailsList |

**Table 7 : Admin management - Endpoint - 4**

|  |  |
| --- | --- |
| **URL** | /api/AddAmmenities |
| **Request Type** | Post |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to add ammenities details in the database |
| **Inputs** | AmmenityID, AmmenityName, Availability |
| **Outputs** | AmmenityDetails |

**Table 8 : Admin management - Endpoint - 5**

|  |  |
| --- | --- |
| **URL** | /api/UpdateAmmenity |
| **Request Type** | Put |
| **User Role** | Admin |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to update Ammenity details in the database |
| **Inputs** | AmmenityID |
| **Outputs** | UpdatedAmmenityDetails |

**Stage: Front-end design**

Create the following components as per the specification provided below

* 1. AddRoom Component
     1. Create a Room component which can be navigated from the menu of the application when the admin login.
     2. The component will provide a form to accept a new room detail and save it into the system after validating all the details.
  2. RoomList Component

1. This component will display the room details from the database with Edit Button.
2. When admin click on edit link it should navigate to UpdateRoom Component.

3. UpdateRoom Component

1. When admin click on edit link it should navigate from RoomList to UpdateRoom Component where RoomId should be passed and respective details will be filled in the form to edit.
2. Once the admin enters the relevant fields then it will be acknowledged with a message.

4. AddAmmenity Component

1. Create a Ammenity component which can be navigated from the menu of the application when the admin login.
2. The component will provide a form to accept a new Ammenity details with Update and Reset Button.
3. The update button will be enabled once new Ammenity save it to the database after validating all the details.
4. Once the admin clicks on update button it should enable textboxes with the respective values to update the details.

5. UpdateAmmenity Component

* + - Once the admin clicks on update button it should enable textboxes with the respective values to update the details.
    - Once the admin the update then an acknowledgement must be displayed.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

**6.3 Module – Room Booking Service**

This module will provide room booking facility for the users. Users can book the meeting internally or outsourced places. Below are the features provided by this module

1. Create a new booking
2. Update the booking status

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

Table : Room\_Booking

|  |  |
| --- | --- |
| Column Name | Type |
| BookingId(PK) | Integer(10) |
| UserId(FK) | Varchar(25) |
| Start\_date | Date |
| End\_date | Date |
| NoofSeat | Integer(10) |
| NoofHours | Integer(4) |
| TotalAmount | Decimal |
| Status | Varchar(25) |

Table : RoomBooking\_Ammenity

|  |  |
| --- | --- |
| Column Name | Type |
| BookingId(FK) | Integer(10) |
| Ammenity\_ID(FK) | Varchar(25) |

2. Apply the following constraints apart from primary keys and foreign keys on the database

1. Values allowed for the meeting status are – scheduled, completed, cancelled.
2. Meeting end date must be after the Meeting start date
3. The amenities of particular room booking will be added into the RoomBooking\_Amenity table. BookingID should be taken from the RoomBooking table and Ammenity ID from Ammenity table.

**Note**: During the application startup, seed the data into the MeetingsPlatforms table as Zoom, Teams, WebEx, etc.,

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations

* Insert new meeting Room Booking
* Update a Room Booking
* Get the Summary

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows:

* Insert new meeting Room Booking
* Update a Room Booking
* Get the Summary

1. Following business rules must be implemented as part of the business service class

* When the user enters the booking details then it provides the summary with payment details. It should be calculated for 1 hr amount + GST +CST and total will be displayed with Summary.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

**Table 9 : Room Booking Management - End point - 1**

|  |  |
| --- | --- |
| **URL** | /api/RoomDetailList |
| **Request Type** | GET |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | User will be able to see the list of rooms with room name, address, phone, image of the room and list of amenitites available from amenity table based on booking ID and UserID. |
| **Inputs** | RoomId |
| **Outputs** | RoomDetails |

**Table 10 : Room Booking Management - End point - 1**

|  |  |
| --- | --- |
| **URL** | /api/Ammenity |
| **Request Type** | GET |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | It will display the ammenity list from the ammenity table |
| **Inputs** |  |
| **Outputs** | Ammenities |

**Table 11 : Room Booking Management - End point - 2**

|  |  |
| --- | --- |
| **URL** | /api/BookRoom/AddAmmenity |
| **Request Type** | POST |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the users to add ammenity for the new meeting room booking. |
| **Inputs** | AmmenityID, BookingID |
| **Outputs** | Status code |

**Table 12: Room Booking Management - End point - 3**

|  |  |
| --- | --- |
| **URL** | /api/BookRoom/create |
| **Request Type** | POST |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the users to create the new meeting room booking. |
| **Inputs** | UsesID, BookingID,RoomID,Start-date,End\_date,Noofseats,NoofHours,TotalAmount |
| **Outputs** | Status code |

**Table 13 : Room Booking Management - End point - 4**

|  |  |
| --- | --- |
| **URL** | /api/BookRoom/Edit |
| **Request Type** | Put |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the users to edit meeting room booking details. |
| **Inputs** | Start-date,End\_date,Noofseats,NoofHours |
| **Outputs** | Status code |

**Table 14 : Room Booking Management - End point - 5**

|  |  |
| --- | --- |
| **URL** | /api/RoomDetailList/RoomId |
| **Request Type** | GET by RoomID |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | User will be able to see the room with room name, address, phone, image of the room and list of amenitites available. |
| **Inputs** | RoomId |
| **Outputs** | RoomDetails |

**Stage: Front-end design**

Create the following components as per the specification provided below.

1. RoomList Component

* Design a component which can be used by user to view the list of rooms.
* Each row in table must contain a button to view details which should redirect to Room Booking component by passing RoomId as parameter.

2.CreateRoomBooking Component

* Develop a component to be used by user which contains a form to create a new booking. It should be redirected from Roomlist Component.
* When the user selects as particular room, the ammenities will be listed with checkboxes from AddAmmenity Component. The selected Ammenities will be saved into the RoomBooking\_ammenities table with booking ID and ammenityID
* Form should display the Roomname, Image, tariff(i.e per hour charge) and user can select start\_date, end\_date, Hours
* There should be a submit button which should save the room after all the validation are passed.
* UserId, RoomID must be taken from RoomListComponent when it redirected from RoomListComponent for specific room. Start\_date and End\_date should generate through a date calender. Hour should be fetched from the dropdown.
* No of hours should not exceed 2 hours. Validation should be done and no of seats should not exceed 10 members.
* The ammenities will be displayed as Checkbox with Ammenity name. When the user selects the ammenity, it will get added into the RoomBooking Ammenity Table.

1. AddAmmenities Component

* Ammenities will be displayed as Checkboxes and user will select the required ammenities and save the details when user save the booking details in the database.

1. RoomSummary Component

* Create a component which will display the details of booking
* Component should also provide a button to update the summary.

1. Updatesummary Component

* Develop a component to update the booking details. In this only room start\_date, end\_date, no of seats and hours will be able to edit.
* Component should redirect from RoomSummary Component when the user wants to edit the details. Once it is done, do the validation and submit the form. Get it acknowledged.

**Stage: Integration of Frontend and backend**

1. Create a data service in the front-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received from the API

**6.4 Module – Recurring Meeting Booking Service**

This module provides the user to book meeting room daily, weekly, monthly and quarterly.

Below are the features provided by this module

1. Create a new recurring booking
2. Update the booking

**Stage: Database Implementation**

1. Design a data base as per the following ER diagram provided.

Table : Recurring\_Booking

|  |  |
| --- | --- |
| Column Name | Type |
| R\_BookingId(PK) | Integer(10) |
| UserID(FK) | Varchar(25) |
| Start\_date | Date |
| End\_date | Date |
| NoofSeaters | Integer(10) |
| NoofHours | Integer(4) |
| PlanType | Varchar(25) |
| TotalAmount | Decimal |

1. Ensure the following constraints along with primary and foreign keys
2. PlanType values will be daily, weekly, monthly or quarterly.
3. TotalAmount should be calculated according to the plantype.

**Stage: Data Access Layer Design**

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
4. Insert new recurring meeting details
5. Get the details
6. Update the details

**Stage: Business Logic Layer Development**

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows:

a. Insert new recurring meeting details

b. Get the details

c. Update the details

Following business rules must be implemented as part of the business service class

* Daily, weekly, monthly and Quarterly Charges will be supplied to calculate the total.
* When the recurring room details save in the Recurringbooking table then ammenities will be saved in the RoomBooking\_ammenties table with the ammenity ID and BookingID.

**Stage: Unit Testing**

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. Create the following end-points and test them using postman and export the requests into a json file.

**Stage: Unit Testing**

* Create a new Unit test project to test the service classes created in business logic layers
* Mock all the repositories using a mocking framework.

**Stage: Micro-service implementation**

1. Create an API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

**Table 15 : Recurring Booking Management - End point - 1**

|  |  |
| --- | --- |
| **URL** | /api/RoomDetailList |
| **Request Type** | GET |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | User will be able to see the list of rooms with room name, address, phone, image of the room,type and list of amenitites. |
| **Inputs** | RoomId |
| **Outputs** | RoomDetails |

**Table 16 : Recurring Booking Management - End point - 2**

|  |  |
| --- | --- |
| **URL** | /api/Ammenity |
| **Request Type** | GET |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | It will display the ammenity list from the ammenity table |
| **Inputs** |  |
| **Outputs** | Ammenities |

**Table 17 : Recurring Booking Management - End point - 3**

|  |  |
| --- | --- |
| **URL** | /api/BookRoom/create |
| **Request Type** | POST |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the users to create the new meeting room booking. |
| **Inputs** | BookingID, UserID, Start-date,End\_date,Noofseats,NoofHours,Type,TotalAmount with ammenities |
| **Outputs** | Status code |

**Table 18 : Recurring Booking Management - End point - 4**

|  |  |
| --- | --- |
| **URL** | /api/BookRoom/AddAmmenity |
| **Request Type** | POST |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the users to add ammenity for the new meeting room booking. |
| **Inputs** | AmmenityID, BookingID |
| **Outputs** | Status code |

**Table 19 : Recurring Booking Management - End point - 5**

|  |  |
| --- | --- |
| **URL** | /api/BookRoom/Edit |
| **Request Type** | Put |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the users to edit meeting room booking with ammenities |
| **Inputs** | UserID, BookingID, RoomId,Start-date,End\_date,Noofseats,NoofHours,Type,Ammenities |
| **Outputs** | Status code |

**Table 20: Recurring Booking Management - End point - 6**

|  |  |
| --- | --- |
| **URL** | /api/RoomDetailList/RoomId |
| **Request Type** | GET by RoomID |
| **User Role** | User |
| **Trigger** | Front end |
| **Description** | User will be able to see the room with room name, address, phone, image of the room , type and list of amenitites available. |
| **Inputs** | RoomId |
| **Outputs** | RoomDetails |

**Stage: Front-end design**

Create the following components as per the specification provided below.

1. RoomList Component

* Design a component which can be used by user to view the list of rooms.
* Each row in table must contain a button to view details which should redirect to Room Booking component by passing RoomId as parameter.

2.RoomBooking Component

* Develop a component to be used by user which contains a form to create a new booking. It should be redirected from Roomlist Component.
* Form should display the Roomname, Image, tariff(i.e per hour charge for different plan like daily, weekly etc ..) and user can select start\_date, end\_date, Hours, plan .
* When the user selects as particular room, the ammenities will be listed with checkboxes. The selected Ammenities will be saved into the ammenities table with booking ID and ammenityID
* There should be a submit button which should save the room after all the validation are passed.
* UserId, RoomID must be taken from RoomListComponent when it redirected from RoomListComponent for specific room. Start\_date and End\_date should generate through a date calender. Hour should be fetched from the dropdown. Plan type will be displayed as list
* No of hours should not exceed 2 hours. Validation should be done and no of seats should not exceed 10 members.

1. Ammenities Component

* Ammenities will be displayed as Checkboxes and user will select the required ammenities and save the details when user save the booking details in the database.

1. RoomSummary Component

* Create a component which will display the details of booking
* Component should also provide a button to update the summary.

5. Updatesummary Component

* Develop a component to update the booking details. In this only room, start\_date, end\_date, no of seats and hours will be able to edit.
* Component should redirect from RoomSummary Component when the user wants to edit the details. Once it is done, do the validation and submit the form. Get it acknowledged.

**Stage: Integration of Frontend and backend**

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

7.Deployment requirements

1. All the Microservices must be deployed on a local web server like IIS or Apache Tomcat
2. All the Microservices must be independently deployable.
3. These services must be consumed from an front-end app running in a local environment.

8. Design Considerations

Java and Dotnet specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.



Refer this link for the coding standards. <https://cognizantonline.sharepoint.com/:w:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/Coding%20standards/Effective%20coding%20standards.docx?d=w6430574d9db5478bbbe37c25b16e68e2&csf=1&web=1&e=84lTVf>

1. Most Important and Common rules

|  |  |
| --- | --- |
| **Category** | **Rule** |
| Database | Table names in database must be pascal cased and plural. All primary keys must be named as Pk\_<table>. All foreign keys must be named as FK\_<PrimaryKeyTable>\_<ForeignKeyTable> |
| Database | Column names must be pascal cased. Multi-word column must be split using \_ (underscore) |
| Coding | Follow pascal casing for naming classes, interfaces, methods, properties and other public members |
| Coding | Use camel casing for method parameter name, backing fields for properties and private variables. Consts must be capitalized |
| Coding | All exceptions must be handled and logged using a logging library |
| Coding | For communication between micro-services use the HttpClient class available in .Net and Java |
| Unit testing | Each method in services classes in business logic must be unit tested using nUnit/jUnit |
| Unit testing | Use a mocking library to mock the repositories while performing tests for business logic layer |
| Code Coverage | Should be minimum 90% |
| Front-end(Angular/React ONLY) | Use pascal casing for the component names |
| Front-end(Angular/React ONLY) | Create all components and data services in Angular/React project in dedicated folders |
| GitHub | Create ONLY Private Repositories.  No password should be stored.  DO NOT Mention in the Profile that You work for Cognizant |

Any deviation from the high-level design must be approved by trainer, mentor and solutions team

9.Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**Other References:**

|  |  |
| --- | --- |
| Java 8 Parallel Programming | <https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8> |
| Feign client | [https://dzone.com/articles/Microservices-communication-feign-as-rest-client](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger (Optional) | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](https://dzone.com/articles/centralized-documentation-in-microservice-spring-b) |
| ECL Emma Code Coverage | <https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php> |
| Lombok Logging | <https://javabydeveloper.com/lombok-slf4j-examples/> |
| Spring Security | <https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world> |
| H2 In-memory Database | <https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot>  <https://www.baeldung.com/spring-boot-h2-database> |
| AppInsights logging | <https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights> |
| Error response in WebApi | <https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api> |
| Read content from CSV | <https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects> |
| Access app settings key from appSettings.json in .Net core application | <https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/>  <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1> |

1. Project Templates







11.Change Log

|  |  |
| --- | --- |
|  | **Changes Made** |
| V1.0.0 | Initial baseline created on 19-Oct-2022 by Suja Rajasekharan |