## Birthday Event Management System

**Overview:**

This application is built to help customers to arrange birthday event online.

## Users of the System:

1. Admin
2. Customer

## Functional Requirements:

* Build an application that customers can book birthday events online.
* The customers can choose themes for the event, select food menu, add add-ons etc.
* The user will have option to edit and cancel the event.
* The admin can add new themes, delete the themes, edit the theme details and view themes
* The admin can also add/edit/view/delete menu
* The admin can also add add/edit/view/delete add-ons.
* Customer can provide reviews.

While the above ones are the basic functional features expected, the below ones can be nice to have add-on features:

* Filters for like Low to High or showcasing based on the customer’s price range, specific etc.
* Email integration for intimating new personalized offers to customers.
* Multi-factor authentication for the sign-in process
* Payment Gateway

## Output/ Post Condition:

* Records Persisted in Success & Failure Collections
* Standalone application / Deployed in an app Container

## Non-Functional Requirements:

|  |  |
| --- | --- |
| **Security** | * App Platform – Username/Password-Based Credentials * Sensitive data has to be categorized and stored in a secure manner * Secure connection for transmission of any data |
| **Performance** | * Peak Load Performance (during Festival days, National holidays etc.) |

|  |  |
| --- | --- |
|  | * Admin application < 2 Sec * Non-Peak Load Performance * Appointment Application< 2 Sec * Admin Application < 2 Sec |
| **Availability** | * 99.99 % Availability |
| **Standard Features** | * Scalability * Maintainability * Usability * Availability * Failover |
| **Logging &**  **Auditing** | * The system should support logging(app/web/DB) & auditing at   all levels |
| **Monitoring** | * Should be able to monitor via as-is enterprise monitoring tools |
| **Cloud** | * The Solution should be made Cloud-ready and should have a   minimum impact when moving away to Cloud infrastructure |
| **Browser Compatible** | * All latest browsers |

**Technology Stack**

|  |  |
| --- | --- |
| Front End | Angular 10  Material Design  Bootstrap / Bulma |
| Server Side | Spring Boot |
| Database | MySQL or Oracle or MSSQL |

## Platform Prerequisites (Do’s and Don’ts):

1. The react app should run in port 8081.
2. Spring boot app should run in port 8080.

## Key points to remember:

1. The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.
2. Remember to check the screenshots provided with the SRS. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.
3. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
4. Adhere strictly to the endpoints given below.

## This is a basic SRS document, so understand them well and please feel free to explore and come with new ideas.

**Application assumptions:**

1. The login page should be the first page rendered when the application loads.
2. Manual routing should be restricted by using Auth Guard by implementing the

Can Activate interface. For example, if the user enters as http://localhost:8080/signup or http://localhost:8080/home the page should not navigate to the corresponding page instead it should redirect to the login page.

1. Unless logged into the system, the user cannot navigate to any other pages.
2. Logging out must again redirect to the login page.
3. To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.
4. Use admin/admin as the username and password to navigate to the admin dashboard.

## Validations:

1. Basic email validation should be performed.
2. Basic mobile number validation should be performed.

## Project Tasks: API Endpoints: Admin Side:

|  |  |  |  |
| --- | --- | --- | --- |
| Action | URL | Method | Response |
| Admin Login | /admin/login | POST-Sends email ID and password | Return True/False |
| Admin SignUp | /admin/signup | POST-Sends Admin Model  data | Admin added |
| Add Theme | /admin/addTheme | POST – Send Theme Model data | Theme Added |
| View Theme | /admin/getTheme | GET – Get Theme data | Retrieve list of Themes |
| Edit Theme | /admin/editTheme/{ themeId} | PUT – Post themeid | Theme edited |
| Delete Theme | /admin/deleteTheme  /{themeId} | DELETE – Delete the themeId | Theme deleted |
| Add Food menu | /admin/addMenu | POST – Send Food Model data | Food Menu Added |
| View Food menu | /admin/getMenu | GET – Get Food  Model | Retrieve all the food menu |
| Edit Food menu | /admin/editMenu/{  menuId} | PUT – Send  FoodMenuId | Food menu edited |
| Delete Food menu | /admin/deleteMenu/  {menuId} | DELETE – Send  FoodMenuId | Food menu edited |
| Add add-on | /admin/addAddon | POST – Sends show model data | Show added |

|  |  |  |  |
| --- | --- | --- | --- |
| View add-on | /admin/getAddon | GET – Get show data | Retrieve all the shows |
| Edit add-on | /admin/editAddon | PUT – send updates show data | Show edited |
| Delete add-on | /admin/deleteAddon | DELETE – send show id | Show deleted |

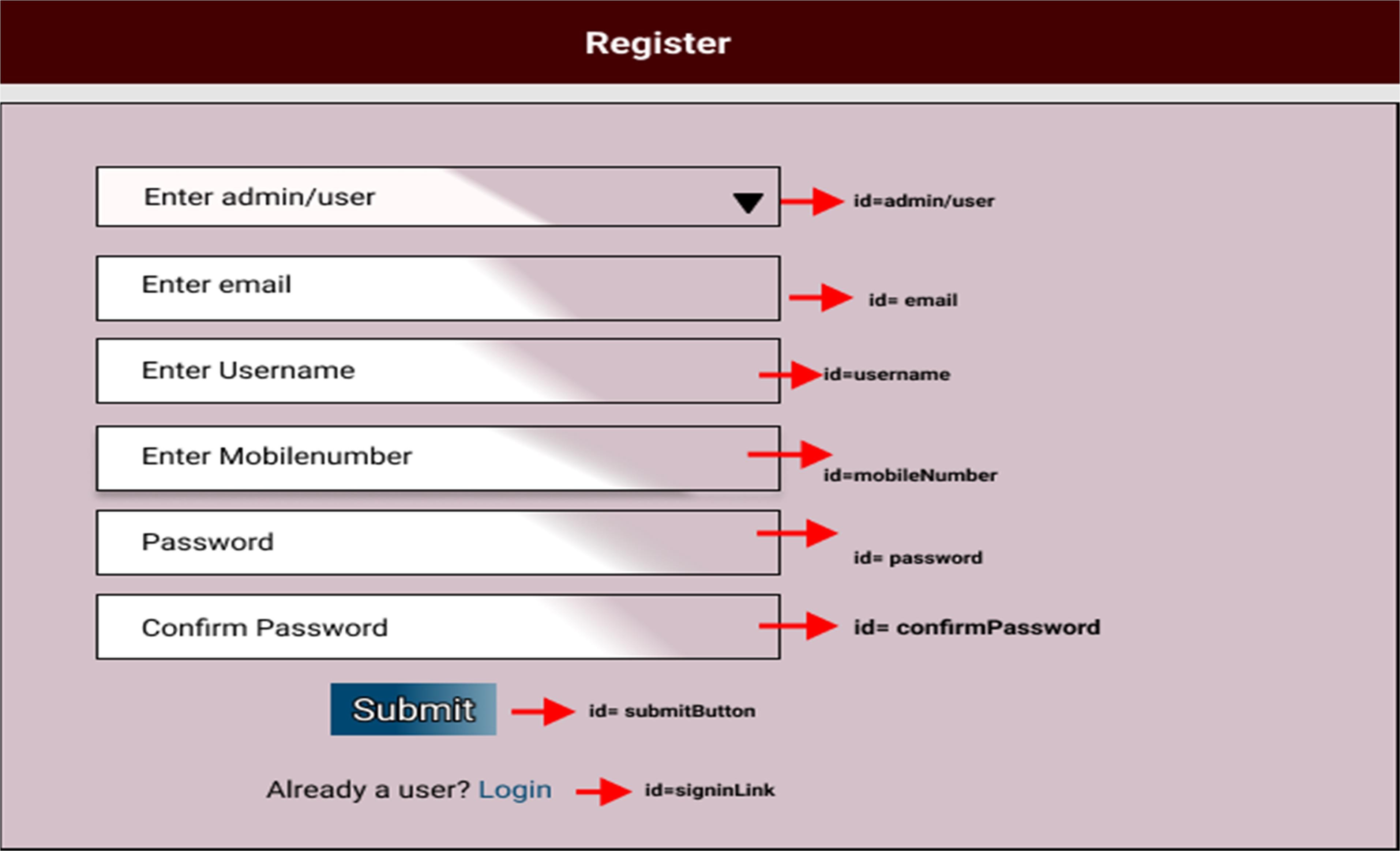
**User Side:**

|  |  |  |  |
| --- | --- | --- | --- |
| Action | URL | Method | Response |
| User Login | /user/login | POST-Sends email ID and  password | Return True/False |
| Admin SignUp | /user/signup | POST-Sends  User Model data | User added |
| Theme | /user/getAllThemes | GET – fetch theme data | Retrieve all themes |
| Book Theme | /user/bookTheme | POST – Send Theme data | Theme booked |
| Edit Theme | /user/editTheme/{themeId  } | PUT – Send theme id | Theme edited |
| Delete Theme | /user/deleteTheme/{them eId} | DELETE – send theme id | Theme deleted |
| View my booked Theme | /user/getBookedTheme | GET – Get theme details | Retrieve the theme that is  booked |

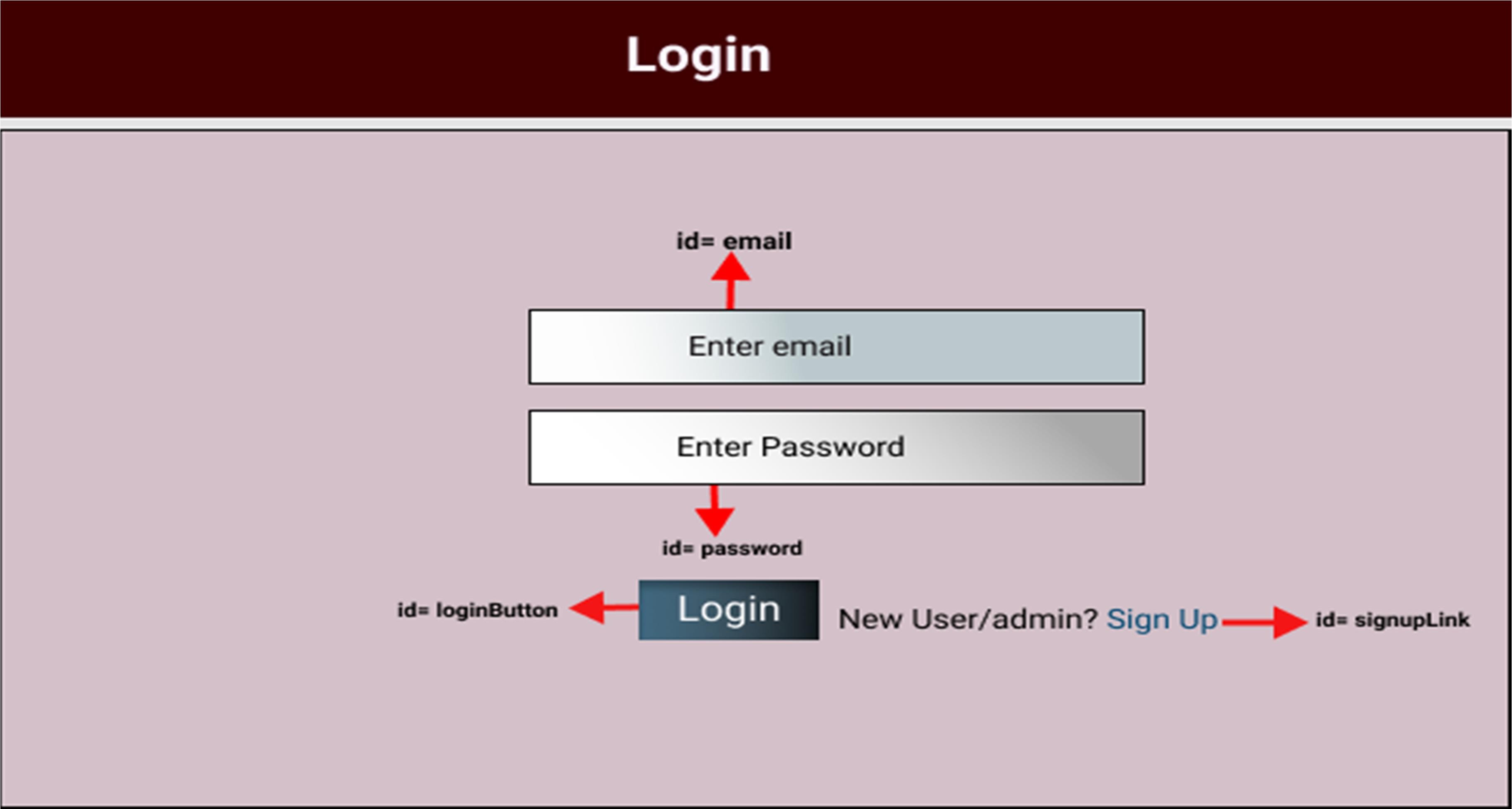
# Frontend:

## Customer:

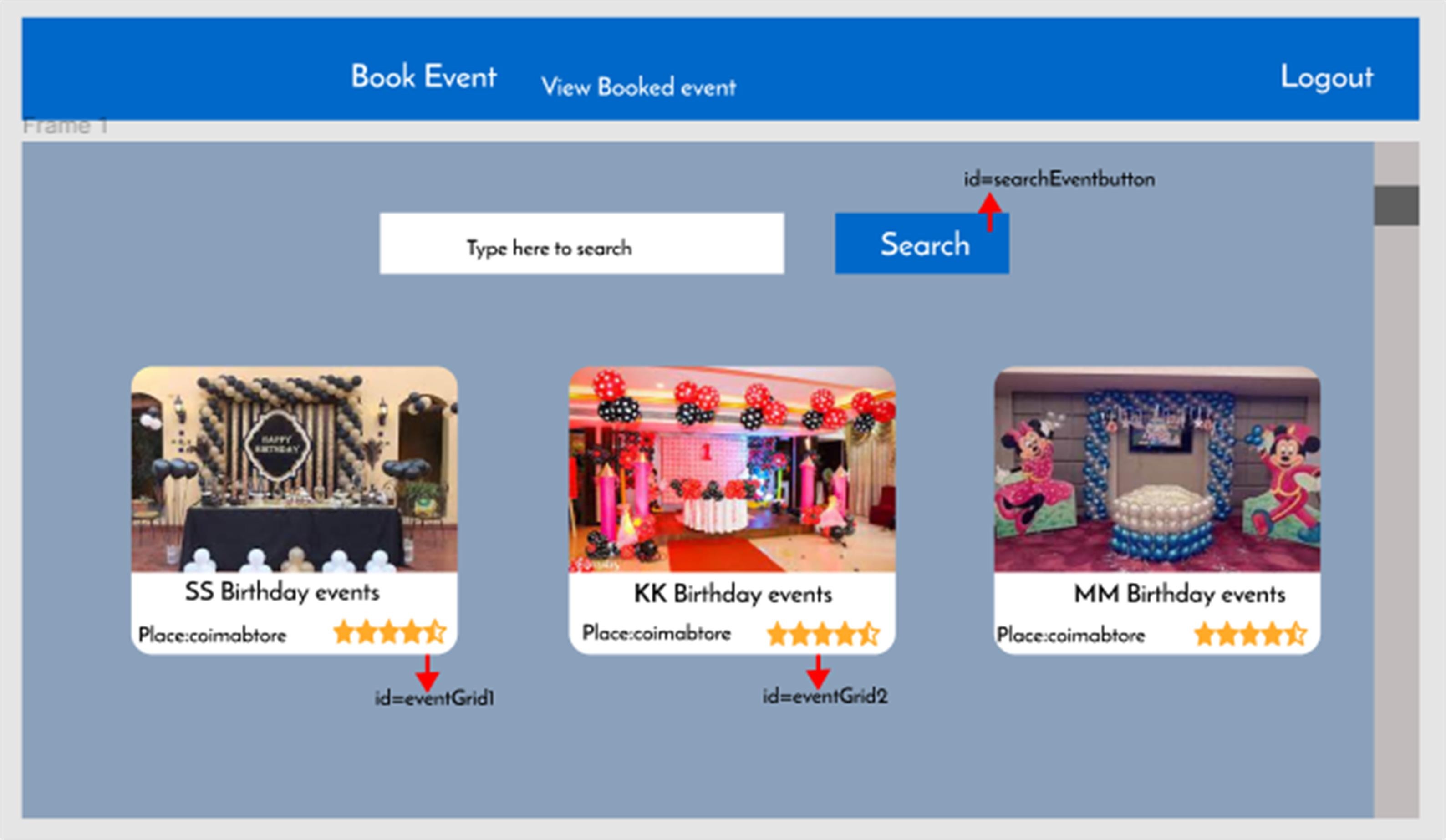
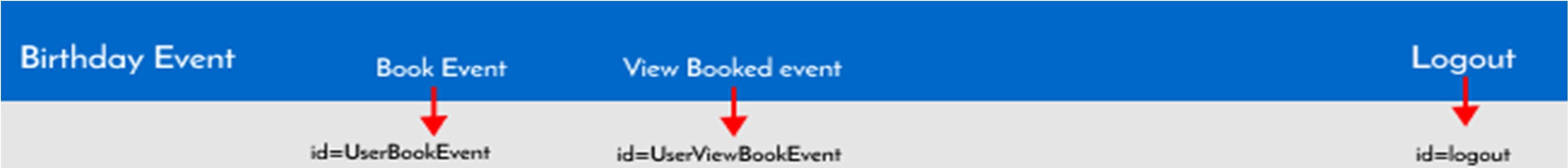
1. Auth: Design an auth component where the customer can authenticate login and signup credentials
2. Signup: Design a signup page component inside the auth where the new customer has options to sign up by providing their basic details.
   1. Ids: Refer to the screenshot below for the id details.
   2. Your frontend should use the ids provided.
   3. Routing URL: http://localhost:8080/user/signup
   4. Output screenshot:



1. Login: Design a login page inside the auth where the existing customer can log in using the registered email id and password.
   1. Ids: Refer to the screenshot below for the id details.
   2. Routing URL: http://localhost:8080/user/login
   3. Output Screenshot:

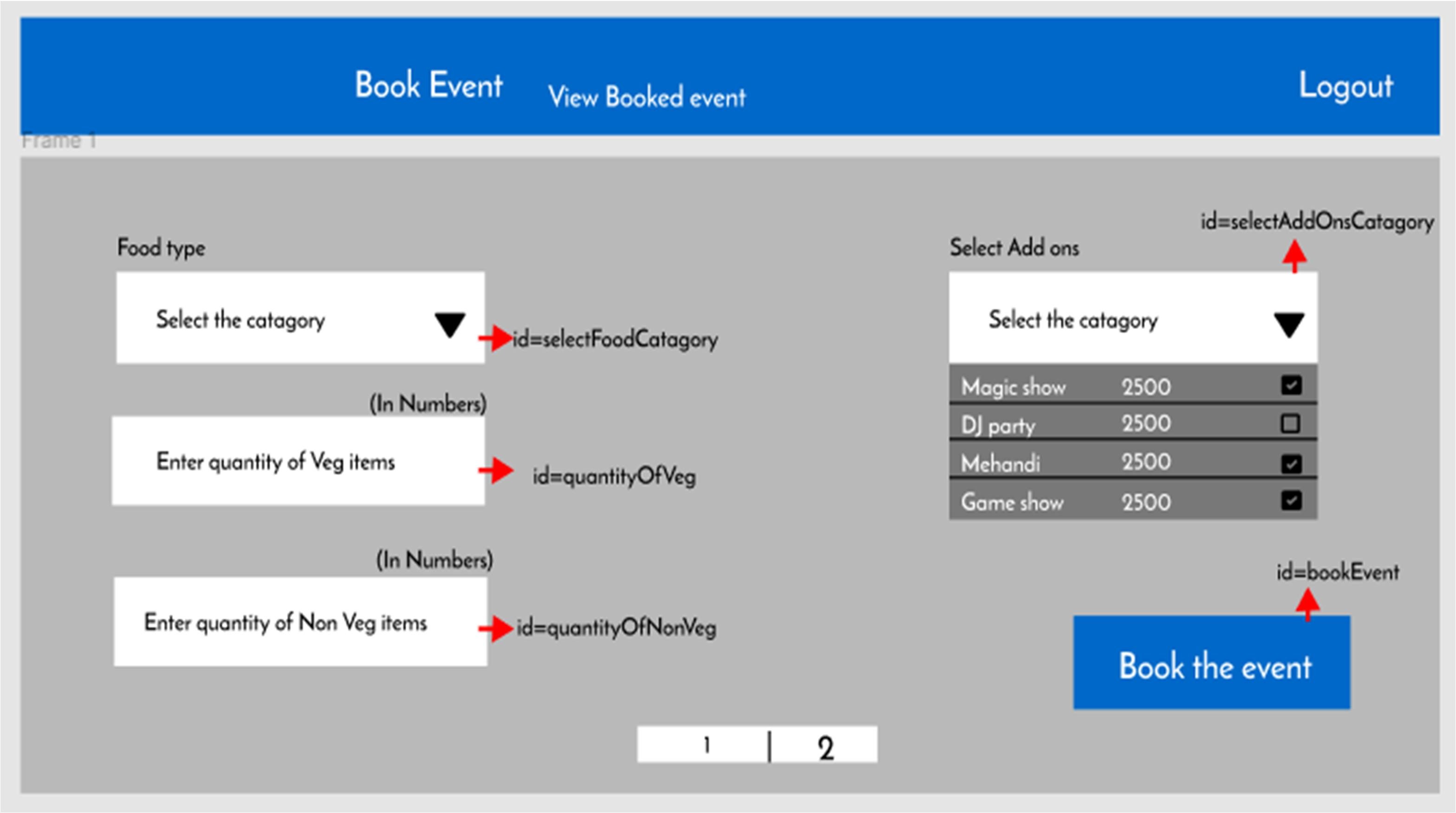
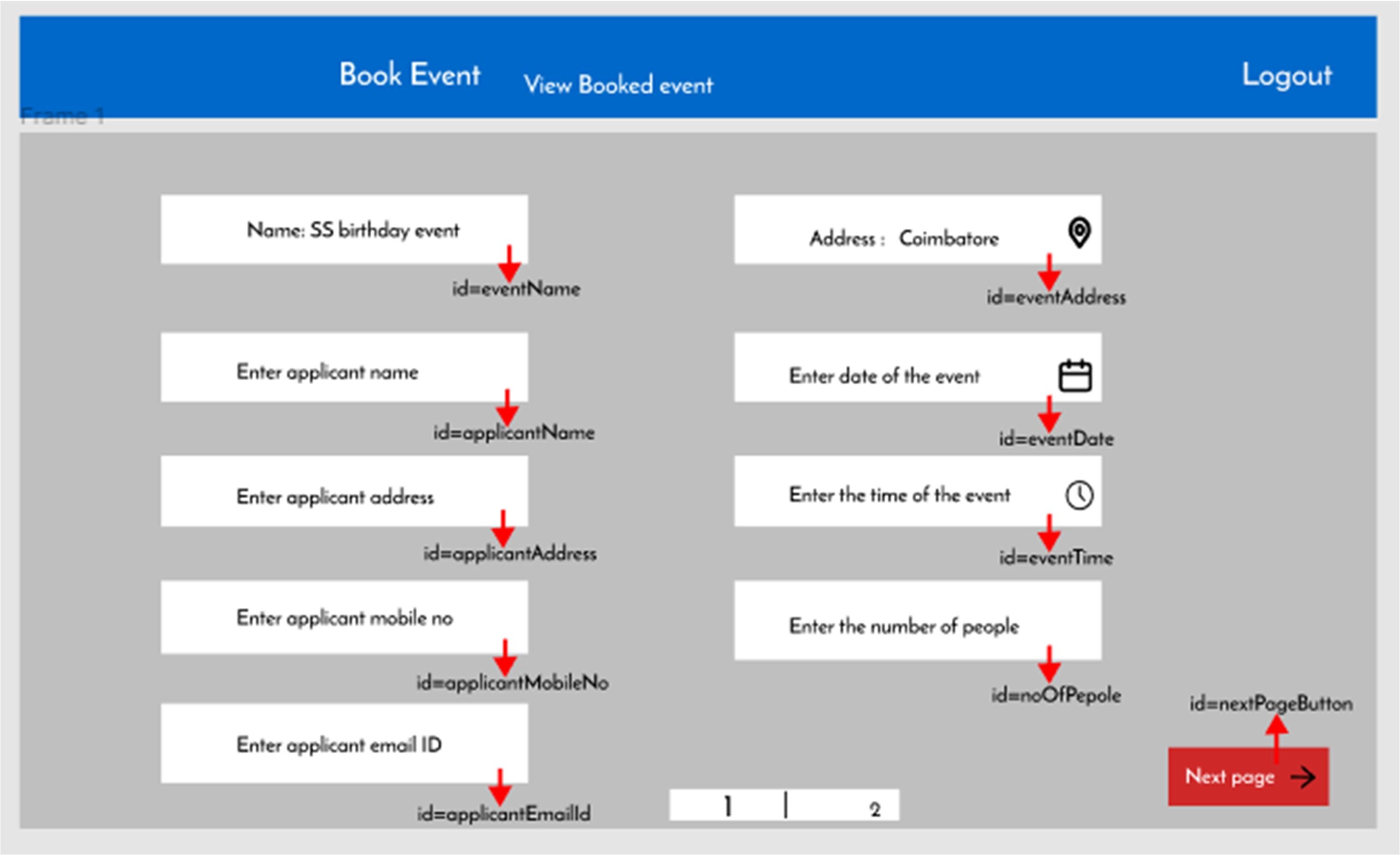


1. Home Page: Design a home page
   1. Ids: Refer to the screenshot below for the id details.
   2. Routing URL: http://localhost:8080/user/getAllThemes
   3. Output Screenshot:

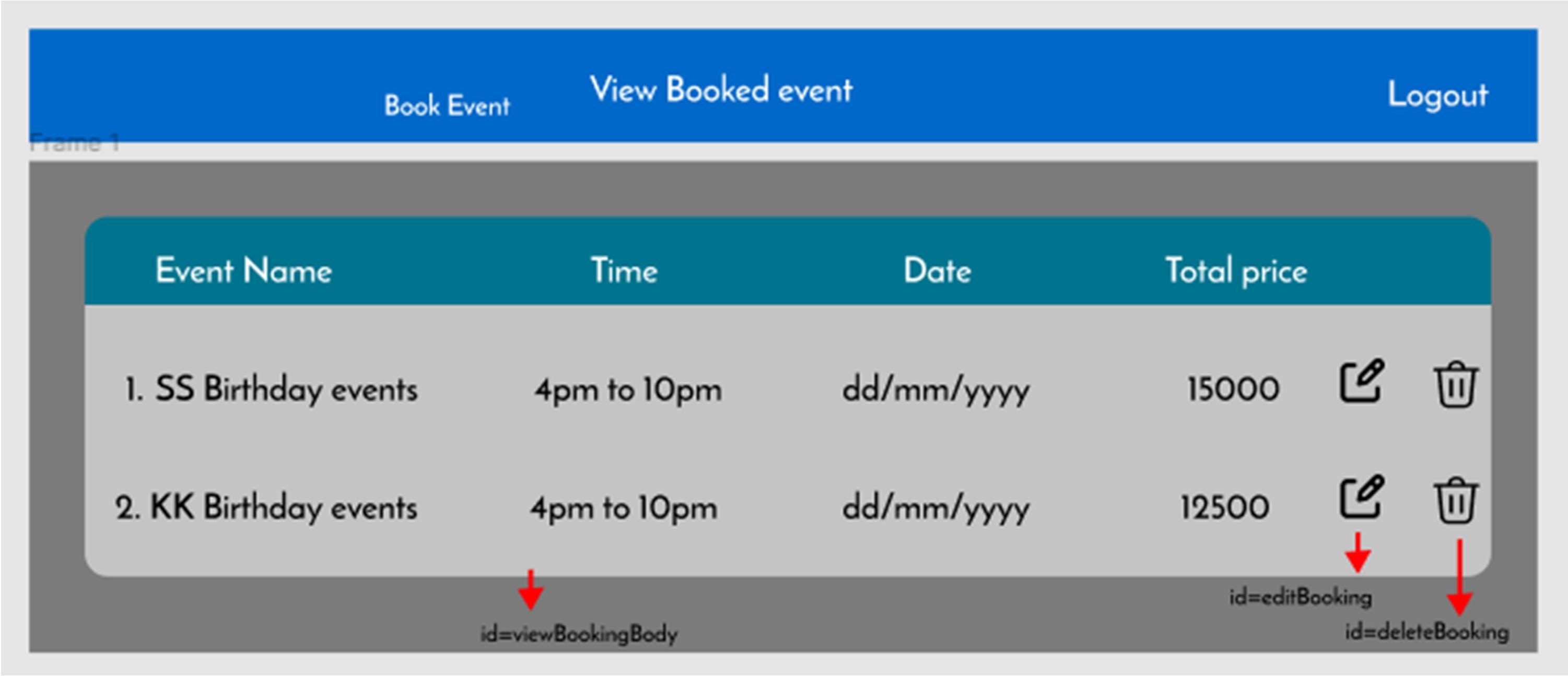


5.Booking: Design an Booking component.

* 1. Ids: Refer to the screenshot below for the id details.
  2. Routing URL: http://localhost:8080/user/bookTheme
  3. Output Screenshot:

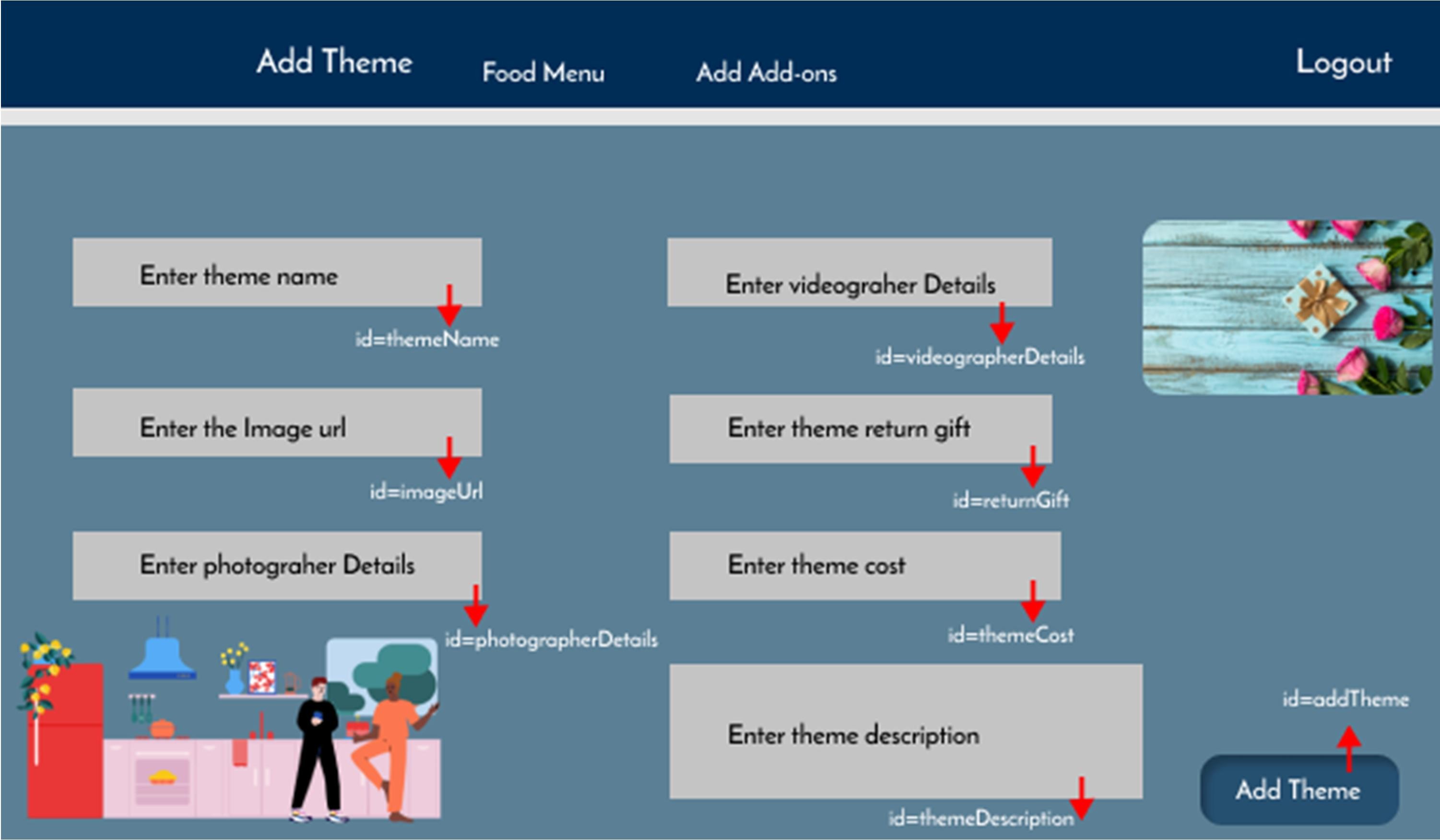


1. View Booking: Design a view booking component.
   1. Ids: Refer to the screenshot below for the id details.
   2. Routing URL: <http://localhost:8080/user/getBookedTheme>
   3. Output Screenshot

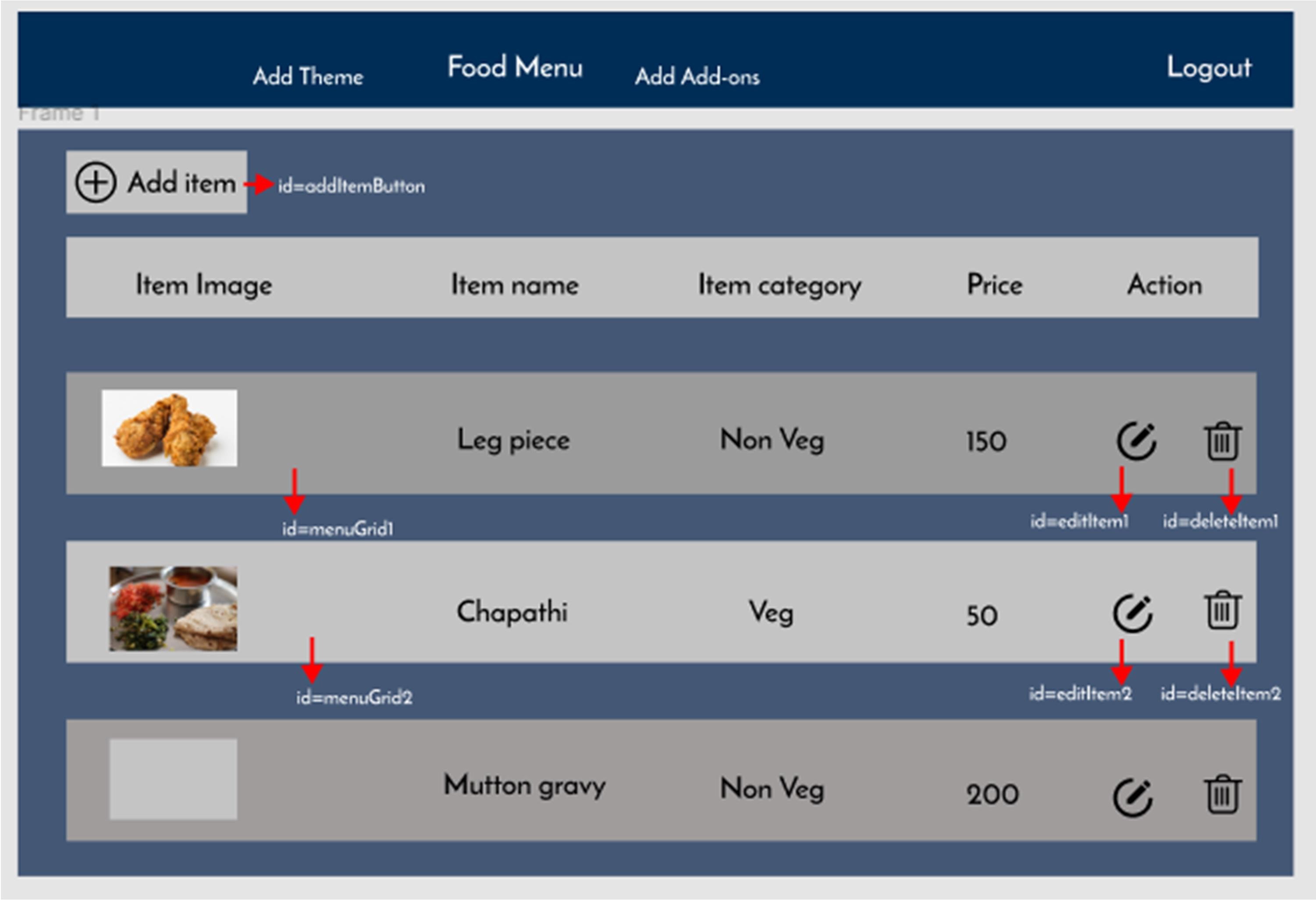


## Admin:

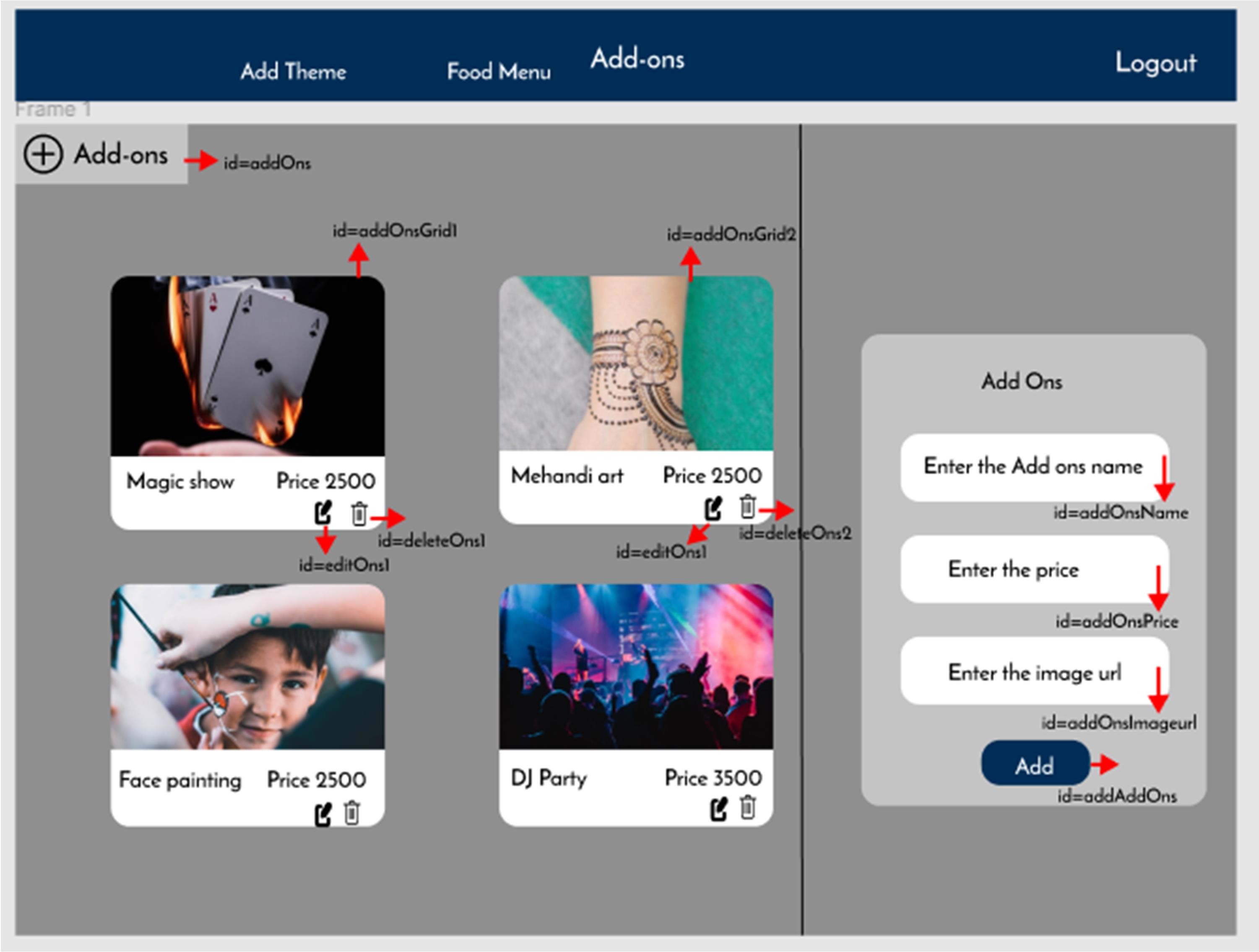
1. Add Theme: Design an add service
   1. Ids: Refer to the screenshot below for the id details.
   2. Routing URL:
      1. http://localhost:8080/admin/addTheme
      2. http://localhost:8080/admin/getTheme
      3. http://localhost:8080/admin/editTheme/{id}
      4. http://localhost:8080/admin/deleteTheme/{id}
   3. Output Screenshot:



1. Add Menu: Design an add service component
   1. Ids: Refer to the screenshot below for the id details.
   2. Routing URL:
      1. http://localhost:8080/admin/addMenu
      2. http://localhost:8080/admin/getMenu
      3. http://localhost:8080/admin/editMenu/{id}
      4. http://localhost:8080/admin/deleteMenu/{id}
   3. Output Screenshot:



1. Add Add-Ons : Design an add on component
2. ds: Refer to the screenshot below for the id details.
3. Routing URL:
   1. http://localhost:8080/admin/addAddon
   2. http://localhost:8080/admin/getAddon
   3. http://localhost:8080/admin/editAddon/{id}
   4. http://localhost:8080/admin/deleteAddon/{id}



Backend:

## Class and Method description:

**Model Layer:**

1. **UserModel**: This class stores the user type (admin or the customer) and all user information.
   1. Attributes:
      1. email: String
      2. password: String
      3. username: String
      4. mobileNumber: String
      5. userRole: String
2. **LoginModel**: This class contains the email and password of the user.
   1. Attributes:
      1. email: String
      2. password: String
3. **AdminModel**: This class stores the details of the admin.
   1. Attributes:
      1. email:String
      2. password:String
      3. mobileNumber:String
      4. userRole:String

## ThemeModel:

* 1. Attributes:
     1. themeId; int
     2. themeName: String
     3. themeImageURL: String
     4. themeDescription: String
     5. themePhotographer: String
     6. themeVideographer: String vii.ThemeReturnGift: String

viii. ThemeCost: Long

1. Menu Model:
   1. Attributes:
      1. foodMenuID: int
      2. foodMenuType: String
      3. foodMenuItems: String
      4. foodMenuCost: String
2. Add-on Model:
   1. Attributes:
      1. addonId: int
      2. addonName: String
      3. addonDescription: String
      4. AddAddonPrice: String
3. Event Model:
   1. Attributes:
      1. eventId: Int
      2. eventName:String
      3. applicantName: String
      4. ApplicantAddress: String
      5. applicantMobile: String
      6. applicantEmail: String vii.eventAddress: String

viii. eventDate: Date

1. eventTime: DateTime
2. eventMenuId: Int
3. addonId: Int xii.EventCost: String

## Controller Layer:

1. **AuthController**: This class control the user /admin signup and signin
   1. Methods:
      1. isUserPresent(LoginModel data): This method helps to check whether the user present or not and check the email and password are correct and return the boolean value.
      2. isAdminPresent(LoginModel data): This method helps to check whether the admin present or not and check the email and password are correct and return the boolean value.
      3. saveUser(UserModel user): This method helps to save the user data in the database.
      4. saveAdmin(UserModel user): This method helps to save the admin data in the database.
2. **UserController**: This class helps to get the users
   1. Methods:
      1. addUser(UserModel data): This method adds user to the application.
      2. getUser(String UserID): This method is used to get the list of users.
      3. editUser(String UserID). This method helps to edit the users.
      4. deleteUser(String UserID): This method helps to delete the users
3. **ThemeController**: This class helps to add/edit/delete/view themes
   1. Methods:
      1. addTheme(ThemeModel data): This method adds theme.
      2. getTheme(Int themeId): This method retrieves the theme.
      3. EditTheme(int themeId); This method edits the theme
      4. ViewTheme(int themeId); This method deletes the theme
4. **FoodMenuController**: This class helps to add/edit/view/delete menu.
   1. Methods:
      1. addMenu(MenuModel data): This method adds Menu
      2. editMenu(int menuId): This method edits the menu.
      3. getMenu(int menuId): This method retrieves all the menu
      4. deleteMenu(int menuId): This method deletes the menu
5. **AddOnController**:
   1. Methods:
      1. addAddon(AddonModel data): This method adds Add-on
      2. editAddon(int addonId): This method edits the Add-on.
      3. getAddon(int addonId): This method retrieves all the Add-on
      4. deleteAddon(int addonId): This method deletes the Add-on
6. **EventBookingController**:
   1. Methods:
      1. bookEvent(EventModel data): This method books an event.
      2. viewEvent(int eventId): This method displays the bookedEvent
      3. editEvent(int eventId): This method edits the event.
      4. deleteEvent(int eventId): This method deletes the event.

How to run the Project

**Back End**

**API endpoint:**

8080

**Platform Guidelines:**

To run the command use **Terminal**in the platform.

**Spring Boot:**

Navigate to the springapp directory => **cd springapp**

To start/run the application '**mvn spring-boot:run**'

**To Connect Database Open Terminal**

**Cmd:mysql -u root –protocol=tcp -p**

**Password: examly  
  
Front End**

**Step 1:**

Open the terminal

Use “nvm use 14” command to change node version to 14

**Step 1:**

Use "cd reactapp" command to go inside the reactapp folder

Install Node Modules **- "**npm install**"**

**Step 2:**

Write the code inside src folder

Create the necessary components

**Step 3:**

Click the run test case button to run the test cases

**Note :**

* Click PORT 8081 to view the result / output
* If any error persists while running the app , delete the node modules and reinstall them