Technical University of Cluj-Napoca

Travel Planner

Industrial Informatics Project

Sem. II, Year 2022-2023

Team Members

Apetrei Andrei-Octavian: **Team Leader**
Bokotey Monika-Imola: **Software Developer**
Ciurila Andrei-Victor: **Software Tester**
Dafinoiu Andreea-Raluca: **Database Developer**

Coordinating Teacher

Dr. Ing. Teodora Sanislav

Contents

1 Introduction

1.1 Overview

Have you ever been that one friend in the friend group that carries the big responsibility of organising each and every trip? It becomes really exhausting to do that over and over again sometimes, right?

In this case, we have come up with a solution for you. Its name is "Travel Planner", an app that combines every aspect of a successful trip in one easy to use tool.

We have created a MVC web app that helps you plan a trip from A to Z by offering you the chance to book accommodations, choose different types of activities that you wish to try in a specific area and so on!

1.2 Goal

The main goal of our application is to facilitate the access of the users to information about different locations, accommodations and activities, as well as giving them the ability to book everything mentioned before, without the hassle of navigating through lots of websites in order to gather all the necessary information for their desired trip.

1.3 Functions

As mentioned above, this is an ASP.NET MVC application and we chose to implement it this way as it offers a clearer visualisation of the code and a better structure for implementing new functionalities.

You'll be greeted by our main page that offers you a short overview about who we are and what are we here for and from there you can use the navigation bar to create an account, or enter your already existing one as well as searching for accommodations and destinations.

One more important aspect to be mentioned is that the navigation bar is present in every view of the application for quick actions.

2 Application Design

2.1 Use Cases and Class Diagram

As shown in **Figure** 1, three types of users are used: Customer, Manager and Local. Each type of user has its own functionalities and permissions. When a user logs in, they are granted access to the features associated with their user type.

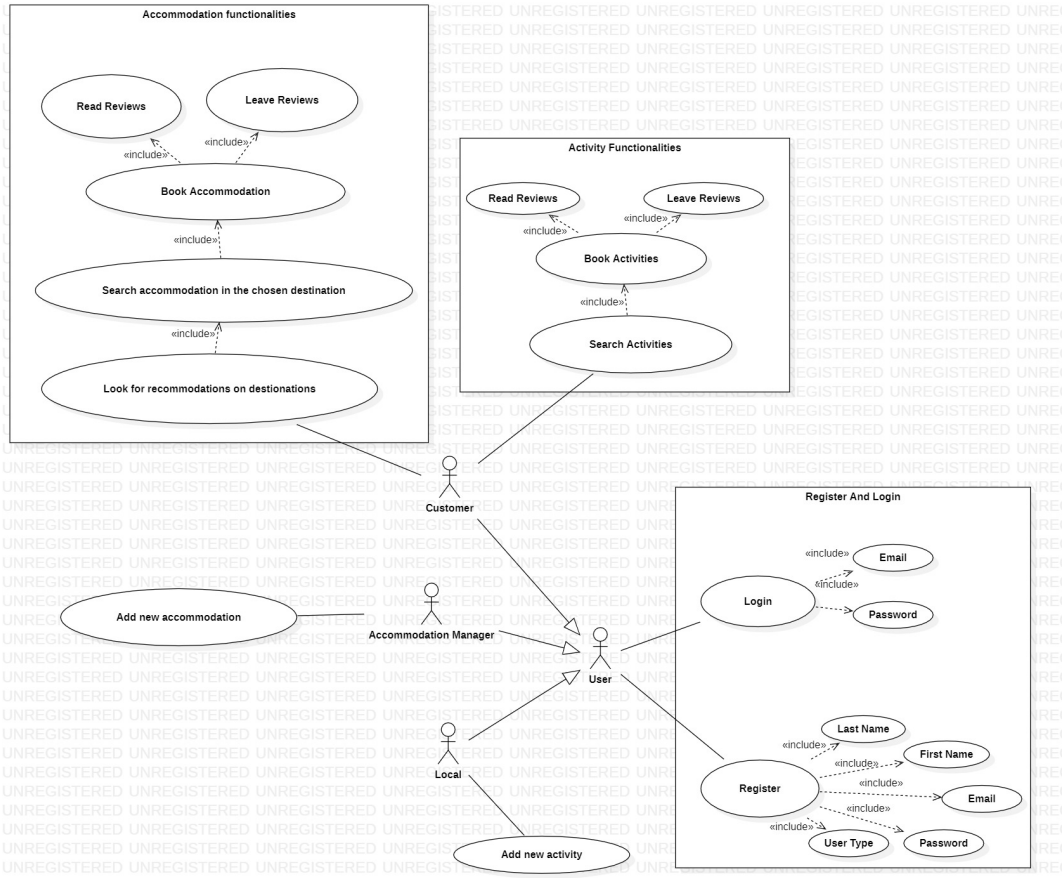

Figure 1: Use Case Diagram

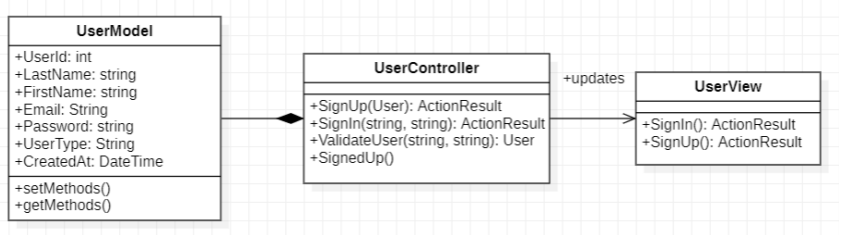

Figure 2: User Class Diagram

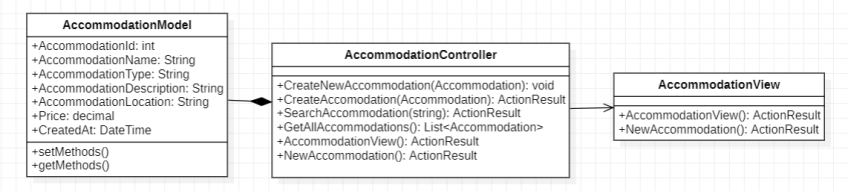

Figure 3: Accommodation Class Diagram

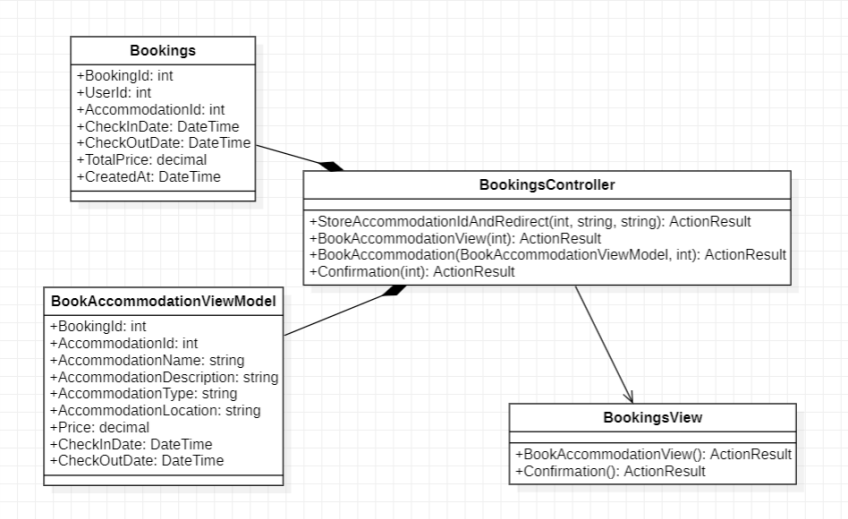

Figure 4: Book Accommodation Class Diagram

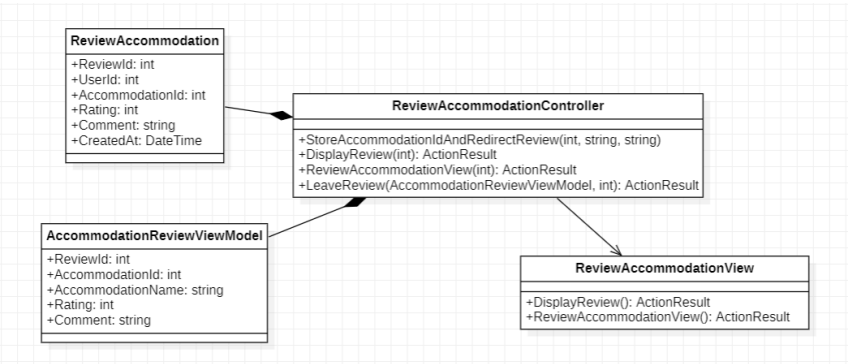

Figure 5: Review Accommodation Class Diagram

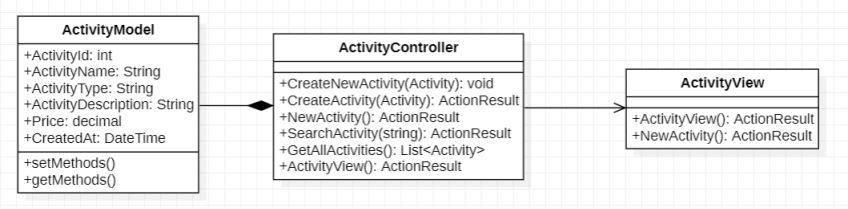

Figure 6: Activity Class Diagram

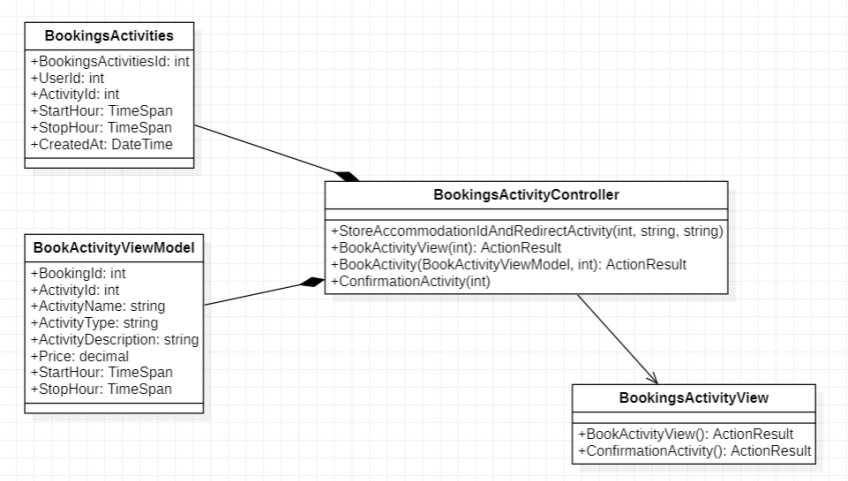

Figure 7: Booking Activity Class Diagram

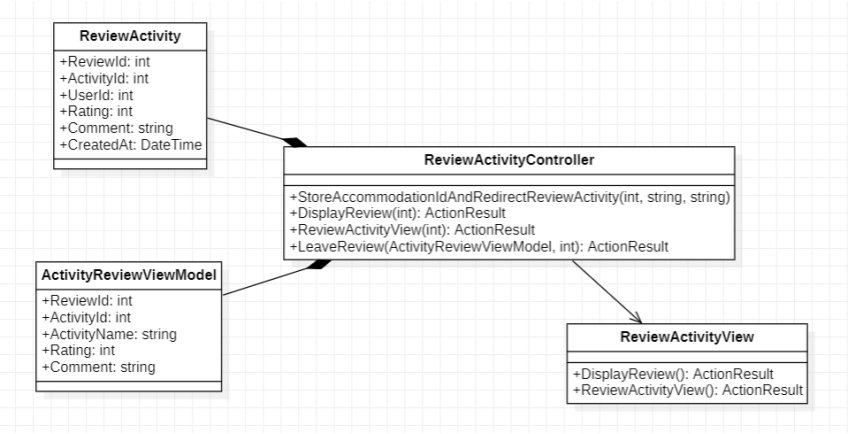

Figure 8: Review Activity Class Diagram

In addition, we have created two models to provide users with an overview of possible destinations they could visit: **DestiantionRecommandation** and **DestiantionRecommandationDefinition**. The *DestinationRecommadation* model includes two fields, namely *CityName* and *CityDescription*. On the other hand, *DestiantionRecommandationDefinition* model is used to initialize *DestiantionRecommandation* objects. A corresponding view has been developed as a part of **Frontend**, which displays all the objects from the *DestiantionRecommandationDefinition*. This implementation approach was chosen to maintain a simple database structure, as opposed to directly add the destination recommedations into the database.

Figure 9: Destination Class Diagram

2.2 Database Design

In order to implement our database, we have used SQL Server, ensuring that each column has an appropriate data type and maximum character length. To minimize the occurrence of duplicate or conflicting primary keys, a unique identifier is generated for every entry in the table through automatic ID incrementation.

The Travel Planner database contains 7 tables, as shown in the **Figure10**.

Users table has all the information about a user which has created an account in the application, such as name, email, password, user type etc.

Accommodations table and **Activities** table contains all the relevant information on accommodations and activities that will be displayed for customer, such as location, description, price etc.

Bookings table and **BookingsActivities** table stores the booked accommodations, respectively the booked activities.

ReviewsAccommodations table and **ReviewsActivities** table contain all the reviews submitted by past users for specific accommodations and activities, respectively

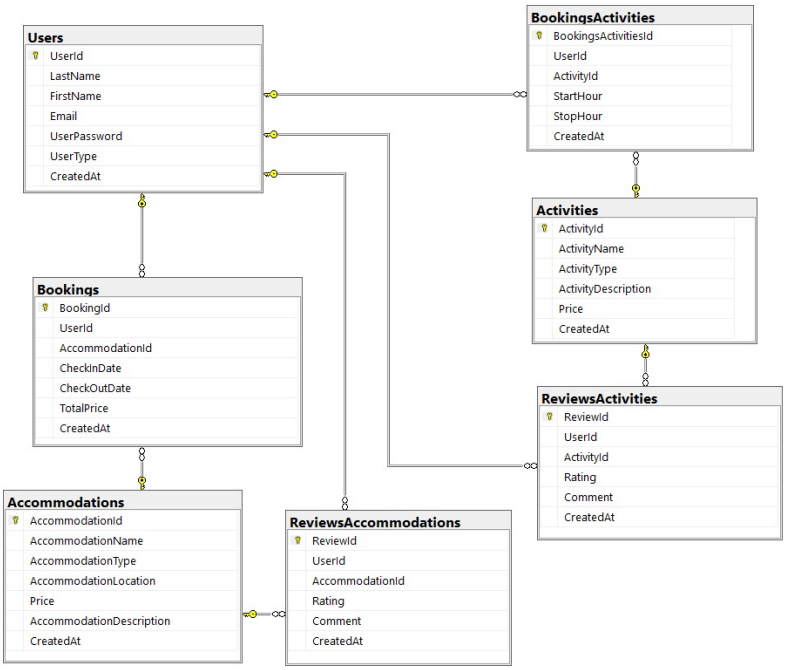

Figure 10: Database Diagram

2.3 GUI Design

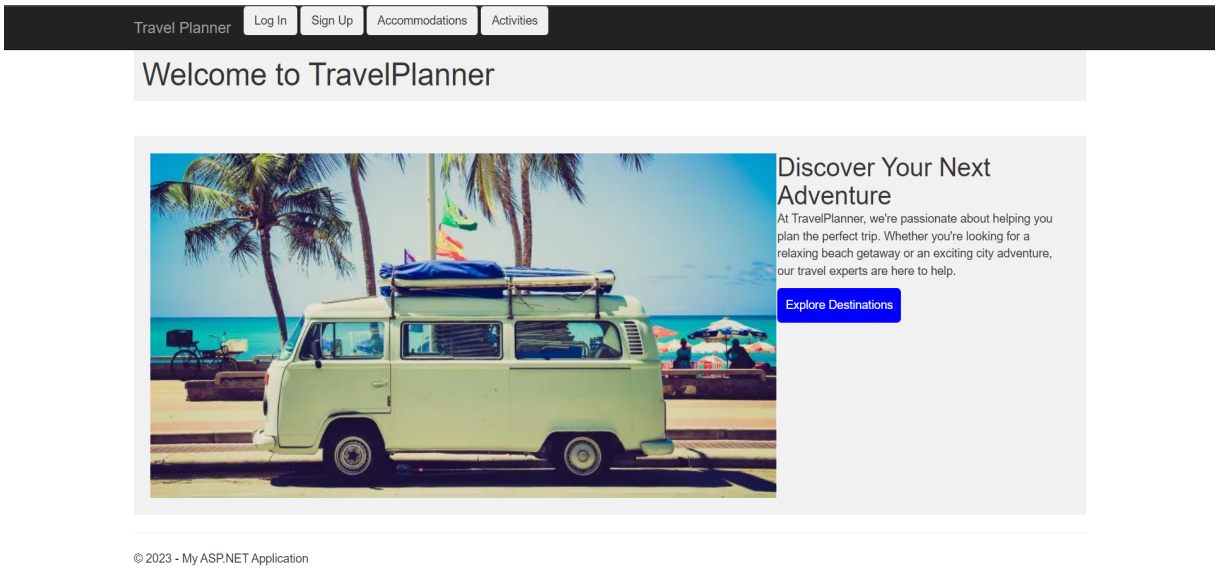

Figure 11: Home Page

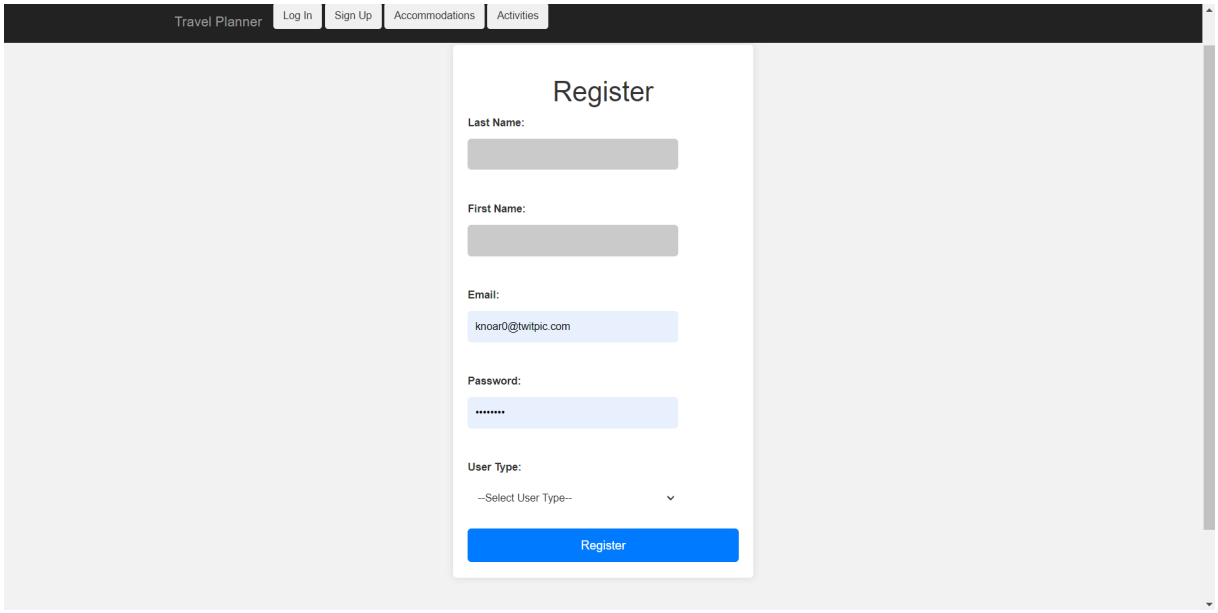

Figure 12: Register Page

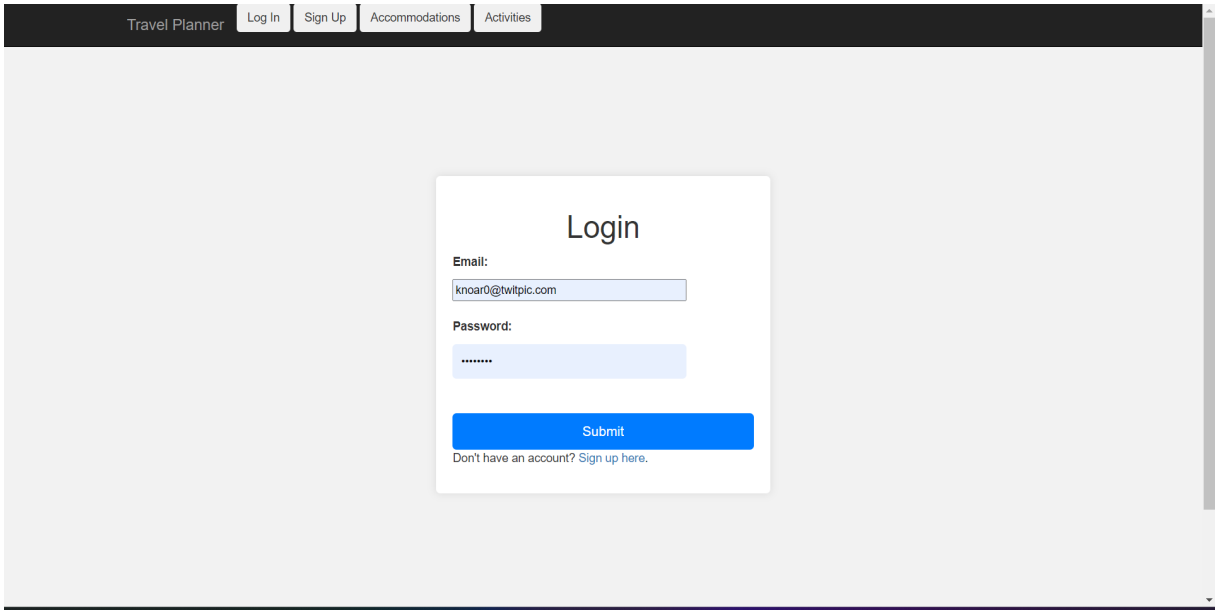

Figure 13: Login Page

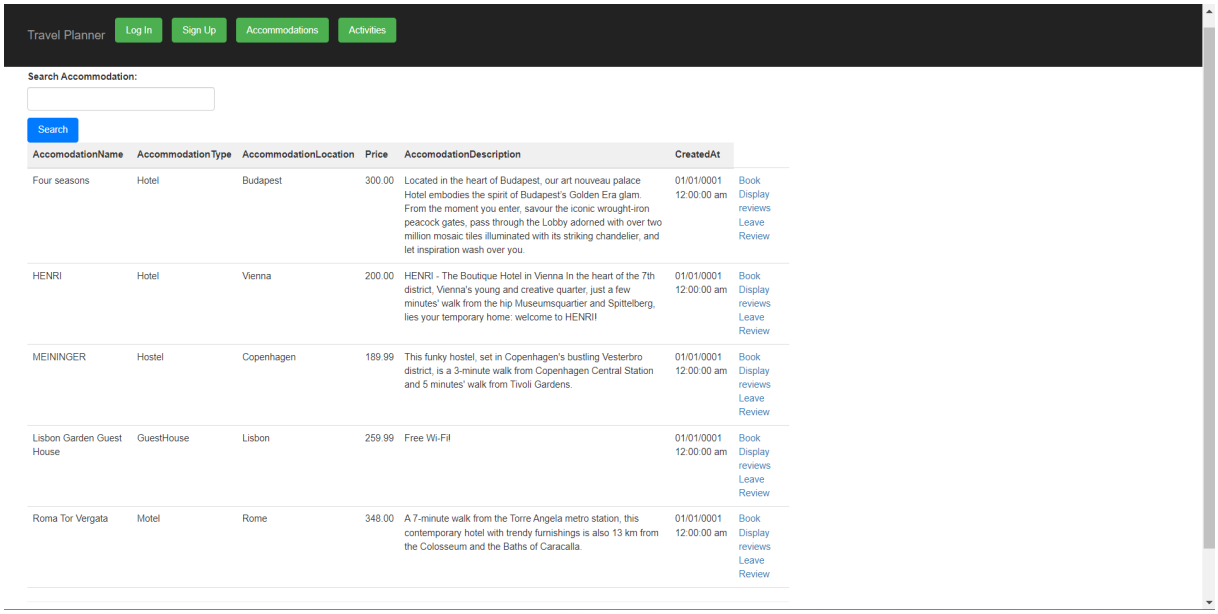

Figure 14: Accommodation Page

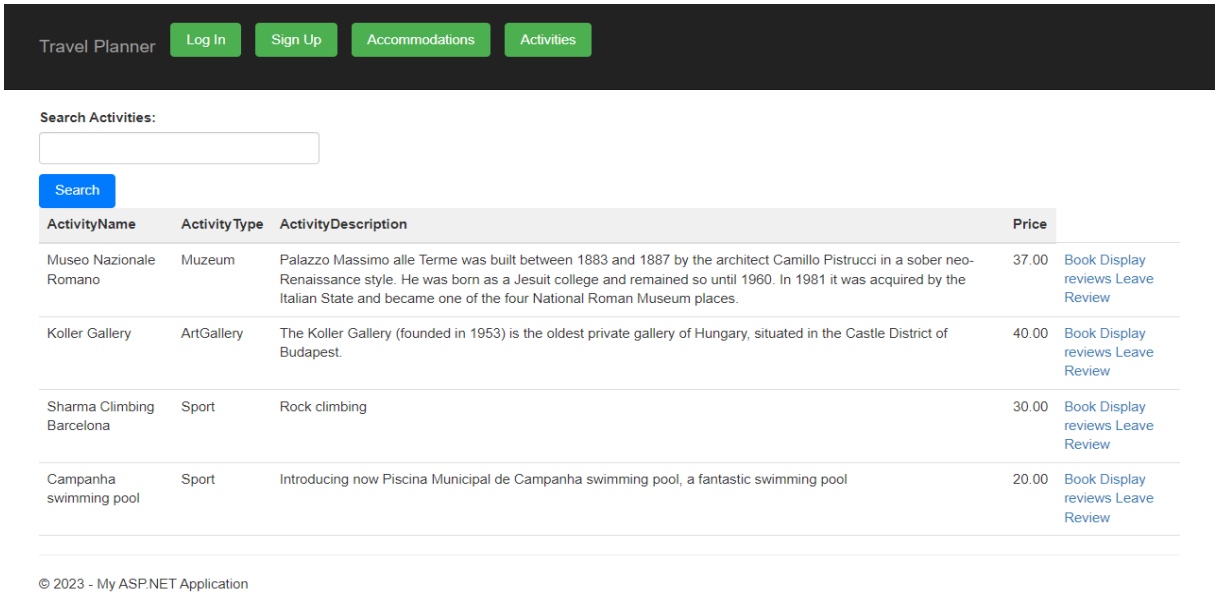

Figure 15: Activity Page

3 Application Implementation

3.1 Backend

The application was developed in C#with ASP.NET Web Application using the MVC framework. The controllers we used are the following:

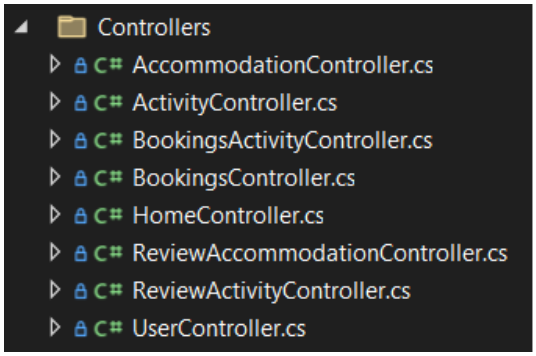

Figure 16: Controllers

Since we used a database-first implementation, every controller is responsible for fetching data from the database and storing incoming data from the view into the database.

3.1.1 User Controller

This subchapter provides an overview of the User Controller, focusing on the sign-up and sign-in methods.

a) **Sign-Up:** When a user submits the sign-up form, the data is sent as a POST request to the server. The server-side code captures the user object, containing the sign-up details, as a parameter of the SignUp method. The method then takes the user object as input, containing details like last name, first name, email, password, user type and adds the user's information to an SQL query using parameterized values.

```
[HttpPost]
public ActionResult SignUp(User user)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        string insertQuery = "INSERT INTO Users(LastName, FirstName, Email,
↪  UserPassword, UserType) VALUES (@LastName, @FirstName, @Email, @Password,
↪  @UserType)";

        using (SqlCommand command = new SqlCommand(insertQuery, connection))
        {
            command.Parameters.AddWithValue("@LastName", user.LastName);
            command.Parameters.AddWithValue("@FirstName", user.FirstName);
            command.Parameters.AddWithValue("@Email", user.Email);
            command.Parameters.AddWithValue("@Password", user.Password);
```

```
            string userType = Request.Form["user-type"];
            if (string.IsNullOrEmpty(user.UserType))
            {
                command.Parameters.AddWithValue("@UserType", "Customer");
            }
            else
            {
                command.Parameters.AddWithValue("@UserType", user.UserType);
            }
            command.ExecuteNonQuery();
            }
        }
    return RedirectToAction("SignedUp");
}
```

b) **Sign-In:** This method facilitates user authentication and access to the application. Users provide their login credentials, which consists of an email address and password. The sign-in method employs the ValidateUser method to verify these credentials against the stored user data. Upon successful validation, the user is directed to a specific page based on their user type.

```
[HttpPost]
public ActionResult SignIn(string email, string password)
{
    User user = ValidateUser(email, password);
    if (user == null)
    {
        return Content("Wrong email or password! Please try again.");
    }
    else
    {
        FormsAuthentication.SetAuthCookie(user.UserId.ToString(), false);

        var ticket = new FormsAuthenticationTicket(1, user.UserId.ToString(),
↪   DateTime.Now, DateTime.Now.AddMinutes(20), false, user.UserType);
        string encryptedTicket = FormsAuthentication.Encrypt(ticket);

        var authCookie = new HttpCookie(FormsAuthentication.FormsCookieName,
↪   encryptedTicket);
        Response.Cookies.Add(authCookie);

        if (user.UserType == "Customer")
        {
            return RedirectToAction("AccommodationView", "Accommodation");
        }
        else if (user.UserType == "Manager")
        {
            return RedirectToAction("NewAccomodation", "Accommodation");
        }
        else if (user.UserType == "Local")
        {
```

```
            return RedirectToAction("NewActivity", "Activity");
        }
        else
        {
            return RedirectToAction("AccommodationView", "Accommodation");
        }
    }
```

```
private User ValidateUser(string email, string password)
{
    User user = null;
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        string selectQuery = "SELECT * FROM Users WHERE Email = @Email AND
↪   UserPassword = @Password";

        using (SqlCommand command = new SqlCommand(selectQuery, connection))
        {
            command.Parameters.AddWithValue("@Email", email);
            command.Parameters.AddWithValue("@Password", password);

            using (SqlDataReader reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                     user = new User
                    {
                        UserId = (int)reader["UserId"],
                        LastName = (string)reader["LastName"],
                        FirstName = (string)reader["FirstName"],
                        Email = (string)reader["Email"],
                        Password = (string)reader["UserPassword"],
                        UserType = (string)reader["UserType"],
                    };
                }
            }
        }
    }
    return user;
```

A FormsAuthenticationTicket is created with the user's identifier, expiration time, and user type. This ticket is then encrypted using FormsAuthentication.Encrypt.

An HttpCookie object is created to store the encrypted ticket, and it is added to the response cookies using Response.Cookies.Add(authCookie). This ensures that the client's browser receives and stores the authentication cookie for subsequent requests.

3.1.2 Accommodation Controller

This section highlights the accommodation functionalities, including displaying all the accommodation, searching for an accommodation based on location and adding a new accommodation to the database.

a) **GetAllAccommodations:** The GetAllAccommodations method retrieves all the accommodations from the database and returns them as a list of Accommodations objects. This method is called within the AccommodationView method, which has a corresponding view (**Section 3.4.2.a**) responsible for displaying all the accommodations to the user.

```
public List<Accommodation> GetAllAccommodations()
{
    List<Accommodation> accommodations = new List<Accommodation>();

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        string selectQuery = "SELECT * FROM Accommodations";

        using (SqlCommand command = new SqlCommand(selectQuery, connection))
        {
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                Accommodation accommodation = new Accommodation();

                accommodation.AccommodationId = (int)reader["AccommodationId"];
                accommodation.AccomodationName = (string)reader["AccommodationName"];
                accommodation.AccommodationType = (string)reader["AccommodationType"];
                accommodation.AccommodationLocation =
↪   (string)reader["AccommodationLocation"];
                accommodation.AccomodationDescription =
↪   (string)reader["AccommodationDescription"];
                accommodation.Price = (decimal)reader["Price"];

                accommodations.Add(accommodation);
            }
        }
    }

    return accommodations;
}
```

b) **AccommodationView:**

```
[HttpGet]
public ActionResult AccommodationView()
{
    List<Accommodation> accommodations = GetAllAccommodations();
    return View(accommodations);
```

c) **SearchAccommodation:** This search functionality enhances the user experience by allowing them to find based on a specific location. It retrieves the matching accommodations from the database and returns them as a list of Accommodations objects.

```
public ActionResult SearchAccommodation(string searchAccommodation)
{
    List<Accommodation> accommodations = new List<Accommodation>();
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();
        string queryToSearch = "SELECT * FROM Accommodations WHERE
↪   AccommodationLocation LIKE @searchAccommodation";
        using (SqlCommand command = new SqlCommand(queryToSearch, conn))
        {
            command.Parameters.AddWithValue("@searchAccommodation", "%" +
↪   searchAccommodation + "%");
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                Accommodation accommodation = new Accommodation
                {
                    AccommodationId = (int)reader["AccommodationId"],
                    AccomodationName = reader["AccommodationName"].ToString(),
                    AccommodationType = reader["AccommodationType"].ToString(),
                    AccommodationLocation =
↪   reader["AccommodationLocation"].ToString(),
                    Price = (decimal)reader["Price"],
                    AccomodationDescription =
↪   reader["AccommodationDescription"].ToString(),
                    CreatedAt = (DateTime)reader["CreatedAt"],
                };
                accommodations.Add(accommodation);
            }
        }
    }
    return View("AccommodationView", accommodations);
```

d) **NewAccommodation:**

```
[HttpGet]
public ActionResult NewAccomodation()
{
    return View();
```

e) **CreateAccommodation:** This method is called when the user submits the form to create a new accommodation. Inside this method, the CreateNewAccommodation is called which performs the actual insertion of the new accommodation in the database. After successful creation, it returns a content result to indicate that the accommodation has been added.

```
[HttpPost]
public ActionResult CreateAccomodation(Accommodation accommodations)
{
    CreateNewAccommodation(accommodations);

    return Content("Added accomodation"); ;
```

f) **CreateNewAccommodation:**

```
public void CreateNewAccommodation(Accommodation accommodations)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        string insertQuery = "INSERT INTO Accommodations (AccommodationName,
↪  AccommodationType, AccommodationLocation, AccommodationDescription, Price,
↪  CreatedAt) VALUES (@AccommodationName, @AccommodationType, @AccommodationLocation,
↪  @AccommodationDescription, @Price, @CreatedAt)";

        using (SqlCommand command = new SqlCommand(insertQuery, connection))
        {
            command.Parameters.AddWithValue("@AccommodationName",
↪  accommodations.AccomodationName);
            command.Parameters.AddWithValue("@AccommodationType",
↪  accommodations.AccommodationType);
            command.Parameters.AddWithValue("@AccommodationLocation",
↪  accommodations.AccommodationLocation);
            command.Parameters.AddWithValue("@AccommodationDescription",
↪  accommodations.AccomodationDescription);
            command.Parameters.AddWithValue("@Price", accommodations.Price);
            command.Parameters.AddWithValue("@CreatedAt", DateTime.Now);

            command.ExecuteNonQuery();
        }
    }
```

3.1.3 Bookings Controller

This controller includes the following methods for handling the accommodation booking.

a) **StoreAccommodationAndRedirect:** This method stores the selected accommodation ID in a session variable and redirects the user to a specified action in a specified controller .

```
public ActionResult StoreAccommodationIdAndRedirect(int accommodationId, string
↪  actionName, string controllerName)
{
    Session["AccommodationId"] = accommodationId;
    return RedirectToAction(actionName, controllerName);
```

b) **BookAccommodation:** This method handles the booking of an accommodation. It receives the booking details through the BookAccommodationViewModel and accommodation ID as parameters. It performs the following:

- Validates the ModelState to ensure that the submitted data is valid.
- Retrieves the user ID from the authentication cookie.
- Retrieves the accommodation ID from the session variable.
- If the user ID and accommodation ID are valid, creates a new Bookings object with the booking details.
- Inserts the booking in the database.
- Retrieves the generated booking ID.
- Redirects the user to a Confirmation page with the generated booking ID as parameter.

```
[HttpPost]
public ActionResult BookAccommodation(BookAccommodationViewModel model, int
↪  accommodationId)
{
    int bookingId = 0;
    if (ModelState.IsValid)
    {
        int userId = 0;
        var authCookieUser = Request.Cookies[FormsAuthentication.FormsCookieName];
        if (authCookieUser != null)
        {
            var ticket = FormsAuthentication.Decrypt(authCookieUser.Value);
            int.TryParse(ticket.Name, out userId);
        }

        if (Session["AccommodationId"] != null)
        {
            accommodationId = (int)Session["AccommodationId"];
```

```
        }

        if (userId != 0 && accommodationId != 0)
        {
            Bookings bookings = new Bookings
            {
                UserId = userId,
                AccommodationId = accommodationId,
                CheckInDate = model.CheckInDate,
                CheckOutDate = model.CheckOutDate,
            };

            TimeSpan stayDuration =
↪   bookings.CheckOutDate.Date.Subtract(bookings.CheckInDate.Date);
            int numberOfNights = stayDuration.Days;
            decimal totalPrice = model.Price * numberOfNights;

            bookings.TotalPrice = totalPrice;
            bookings.CreatedAt = DateTime.Now;

            using (SqlConnection conn = new SqlConnection(connectionString))
            {
                conn.Open();

                string queryInsertBooking = "INSERT INTO Bookings (UserId,
↪   AccommodationId, CheckInDate, CheckOutDate, TotalPrice, CreatedAt) VALUES
↪   (@UserId, @AccommodationId, @CheckInDate, @CheckOutDate, @TotalPrice,
↪   @CreatedAt)";

                using (SqlCommand insertCommand = new SqlCommand(queryInsertBooking,
↪   conn))
                {
                    insertCommand.Parameters.AddWithValue("@UserId", bookings.UserId);
                    insertCommand.Parameters.AddWithValue("@AccommodationId",
↪   bookings.AccommodationId);
                    insertCommand.Parameters.AddWithValue("@CheckInDate",
↪   bookings.CheckInDate);
                    insertCommand.Parameters.AddWithValue("@CheckOutDate",
↪   bookings.CheckOutDate);
                    insertCommand.Parameters.AddWithValue("@TotalPrice",
↪   bookings.TotalPrice);
                    insertCommand.Parameters.AddWithValue("@CreatedAt",
↪   bookings.CreatedAt);

                    insertCommand.ExecuteScalar();
                }
            }

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();

                string queryGetBookingId = "SELECT TOP 1 BookingId FROM Bookings ORDER
↪   BY BookingId DESC";
                using (SqlCommand command = new SqlCommand(queryGetBookingId,
↪   connection))
```

```
                {
                    object result = command.ExecuteScalar();
                    if (result != null)
                    {
                        bookingId = (int)result;
                    }
                }

                string queryGetBooking = "SELECT BookingId, UserId, AccommodationId,
↪  CheckInDate, CheckOutDate, TotalPrice, CreatedAt FROM Bookings WHERE BookingId =
↪  @BookingId";

                using (SqlCommand getBookingCommand = new SqlCommand(queryGetBooking,
↪  connection))
                {
                    getBookingCommand.Parameters.AddWithValue("@BookingId",
↪  bookingId);

                    using (SqlDataReader reader = getBookingCommand.ExecuteReader())
                    {
                        if (reader.Read())
                        {
                            bookings = new Bookings
                            {
                                BookingId = (int)reader["BookingId"],
                                UserId = (int)reader["UserId"],
                                AccommodationId = (int)reader["AccommodationId"],
                                CheckInDate = (DateTime)reader["CheckInDate"],
                                CheckOutDate = (DateTime)reader["CheckOutDate"],
                                TotalPrice = (decimal)reader["TotalPrice"],
                                CreatedAt = (DateTime)reader["CreatedAt"]
                            };
                            return RedirectToAction("Confirmation", new { bookingId =
↪  bookings.BookingId });
                        }
                    }
                }
            }
        }
    }
    return RedirectToAction("Confirmation", new { id = model.AccommodationId });
```

c) **BookAccommodationView:** This method renders a view for booking an accommodation. It takes the accommodation ID as parameter and retrieves the details from the database to populate the BookAccommodationViewModel model. It also has a corresponding view (**Section 3.4.4.a**).

```
public ActionResult BookAccommodationView(int id)
{
    BookAccommodationViewModel model = new BookAccommodationViewModel();
```

```
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();
        string query = "SELECT AccommodationName, AccommodationDescription,
↪   AccommodationType, AccommodationLocation, Price FROM Accommodations WHERE
↪   AccommodationId = @Id";
        using (SqlCommand command = new SqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@Id", id);
            using (SqlDataReader reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                    model.AccommodationName = (string)reader["AccommodationName"];
                    model.AccommodationDescription =
↪   (string)reader["AccommodationDescription"];
                    model.AccommodationType = (string)reader["AccommodationType"];
                    model.AccommodationLocation =
↪   (string)reader["AccommodationLocation"];
                    model.Price = (decimal)reader["Price"];
                }
            }
        }
    }
    return View(model);
```

d) **Confirmation:** This method displays the confirmation page for a successful booking. It takes the booking ID as a parameter and retrieves the booking details from the database to display them on the confirmation page (**Section 3.4.4.b**).

```
public ActionResult Confirmation(int bookingId)
{
    Bookings booking;
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();

        string query = "SELECT BookingId, UserId, AccommodationId, CheckInDate,
↪   CheckOutDate, TotalPrice, CreatedAt FROM Bookings WHERE BookingId = @BookingId";

        using (SqlCommand command = new SqlCommand(query, conn))
        {
            command.Parameters.AddWithValue("@BookingId", bookingId);

            using (SqlDataReader reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                    booking = new Bookings
                    {
                        BookingId = (int)reader["BookingId"],
                        UserId = (int)reader["UserId"],
```

```
                        AccommodationId = (int)reader["AccommodationId"],
                        CheckInDate = (DateTime)reader["CheckInDate"],
                        CheckOutDate = (DateTime)reader["CheckOutDate"],
                        TotalPrice = (decimal)reader["TotalPrice"],
                        CreatedAt = (DateTime)reader["CreatedAt"]
                    };
                }
                else
                {
                    return RedirectToAction("BookingNotFound");
                }
            }
        }
    }
    return View(booking);
```

3.1.4 Review Accommodation Controller

This controller handles the functionality related to displaying and leaving reviews for accommodations.

a) **StoreAccommodationIdAndRedirectReview:** As in the BookingsController, this method stores the accommodation ID in a session and redirects to another action in a different controller.

```
public ActionResult StoreAccommodationIdAndRedirectReview(int accommodationId, string
↪  actionName, string controllerName)
{
    Session["AccommodationId"] = accommodationId;
    return RedirectToAction(actionName, controllerName);
```

b) **DisplayReview:** It starts by creating an empty list of ReviewAccommodation objects. If the accommodation ID stored in the session exists, then it populates the ReviewAccommodation model with the data from the reader. After reading all the reviews, it returns a view called 'DisplayReviewAccommodation' (**Section 3.4.7.a**) and passes the list of reviews as the model. In case of any exceptions during the process, it returns an error view.

```
[HttpGet]
public ActionResult DisplayReview(int accommodationId)
{
    List<ReviewAccommodation> reviewAccommodations = new List<ReviewAccommodation>();
    if (Session["AccommodationId"] != null)
    {
        accommodationId = (int)Session["AccommodationId"];
    }
    try
```

```
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open();
            string queryToDisplayReview = "SELECT * FROM ReviewsAccommodations WHERE
↪   AccommodationId = @AccommodationId";
            using (SqlCommand command = new SqlCommand(queryToDisplayReview, conn))
            {
                command.Parameters.AddWithValue("@AccommodationId", accommodationId);
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        ReviewAccommodation review = new ReviewAccommodation
                        {
                            Rating = (int)reader["Rating"],
                            Comment = reader["Comment"].ToString(),
                            CreatedAt = (DateTime)reader["CreatedAt"]
                        };
                        reviewAccommodations.Add(review);
                    }
                }
            }
        }
        return View("DisplayReviewAccommodation", reviewAccommodations);
    }
    catch (Exception e)
    {
        return View("Error" + e);
    }
```

c) **LeaveReview:** This is responsible for handling the submission of a review for an accommodation. This method was built in the same manner as 'BookAccommodation' from BookingsController:

- First, it checks if the ModelState is valid.
- Retrieves the user ID from the authentication cookie.
- Retrieves the accommodation ID from the session variable.
- If the user ID and accommodation ID are valid, creates a new ReviewAccommodation object and populates its properties with the provided data from the 'model'.
- Inserts the review in the database.
- If the review submission is successful, it returns a content result with the message "Review submitted".
- If the model state is not valid or if any error occurs, it returns a content result with the message "Failed to submit review".

```
[HttpPost]
public ActionResult LeaveReview(AccommodationReviewViewModel model, int
↪  accommodationId)
{
    if (ModelState.IsValid)
    {
        int userId = 0;
        var authCookieUser = Request.Cookies[FormsAuthentication.FormsCookieName];
        if (authCookieUser != null)
        {
            var ticket = FormsAuthentication.Decrypt(authCookieUser.Value);
            int.TryParse(ticket.Name, out userId);
        }

        if (Session["AccommodationId"] != null)
        {
            accommodationId = (int)Session["AccommodationId"];
        }

        if (userId != 0 && accommodationId != 0)
        {
            ReviewAccommodation review = new ReviewAccommodation
            {
                UserId = userId,
                AccommodationId = accommodationId,
                Rating = model.Rating,
                Comment = model.Comment
            };
            review.CreatedAt = DateTime.Now;

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                connection.Open();
                string queryInsert = "INSERT INTO ReviewsAccommodations(UserId,
↪  AccommodationId, Rating, Comment, CreatedAt) VALUES (@UserId, @AccommodationId,
↪  @Rating, @Comment, @CreatedAt)";

                using (SqlCommand command = new SqlCommand(queryInsert, connection))
                {
                    command.Parameters.AddWithValue("@UserId", review.UserId);
                    command.Parameters.AddWithValue("@AccommodationId",
↪  review.AccommodationId);
                    command.Parameters.AddWithValue("@Rating", review.Rating);
                    command.Parameters.AddWithValue("@Comment", review.Comment);
                    command.Parameters.AddWithValue("@CreatedAt", review.CreatedAt);

                    command.ExecuteScalar();
                }
            }
        }
    }
    else
    {
        return Content("Failed to submit review");
    }
```

```
    return Content("Review submitted");
```

d) **ReviewAccommodationView:** As in 'BookAccommodationView' from BookingsController, this method is responsible for displaying the view (**Section 3.4.7.b**) for leaving a review for an accommodation. It retrives the accommodation name based on the provided ID and sets the necessary properties in the 'AccommodationReviewViewModel' object. The view is then rendered with the populated model.

```
public ActionResult ReviewAccommodationView(int id)
{
    AccommodationReviewViewModel review = new AccommodationReviewViewModel();

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();
        string query = "SELECT AccommodationName FROM Accommodations WHERE
↪   AccommodationId = @Id";
        using (SqlCommand command = new SqlCommand(query, conn))
        {
            command.Parameters.AddWithValue("@Id", id);
            using (SqlDataReader reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                    review.AccommodationId = id;
                    review.AccommodationName = (string)reader["AccommodationName"];
                }
            }
        }
    }
    return View(review);
```

3.1.5 Activity Controller

This controller manages the activities functionality, which is implemented in the same way as the ones in *Accommodation Controller*. It contains the following methods: *GetAllActivities*, *ActivityView*, *SearchActivity*, *NewActivity*, *CreateNewActivity*, *CreateActivity*.

3.1.6 BookingsActivity Controller

The BookingsActivity Controller handles the activities booking and is implemented similarly to the *BookingAccommodation Controller*. It includes the following methods: *StoreAccommodationIdAndRedirectActivity*, *BookActivity*, *BookActivityView*, *ConfirmationActivity*.

3.1.7 ReviewsActivity Controller

This controller oversees the activities reviews functionality, which is implemented in a parallel manner to the ones in the *ReviewsAccommodation Controller*. The methods implemented are: *StoreActivityIdAndRedirectReviewActivity*, *DisplayReview*, *LeaveReview*, *ReviewActivityView*.

3.2 Frontend

The frontend of the application is developed using HTML and CSS. The views are responsible for rendering the user interface by using the data provided by the controllers presented in **Backend**. These views define how data should be presented and create the visual output. Now, let's take a look at the main views in the application:

3.2.1 User View

a) **Sign Up:** Displays a form that enables the user to create a new account. Also, it is associated with the *SignUp* method from *UserController*.

```
<!DOCTYPE html>
<html>
<head>
    <title>Registration Page</title>
    <link href="~/Register.css" rel="stylesheet" type="text/
    css"/>
</head>
<body>
    <div class="register-container">
        <h1>Register</h1>
        <form method="post" action="@Url.Action("SignUp", "
    User")">
            <label for="lastname">Last Name:</label>
            <input type="text" id="lastname" name="lastname"
    required>
            <br />

            <label for="firstname">First Name:</label>
            <input type="text" id="firstname" name="firstname"
     required>
            <br />

            <label for="email">Email:</label>
            <input type="email" id="email" name="email"
    required>
            <br />

            <label for="password">Password:</label>
```

```
            <input type="password" id="password" name="
    password" required>
            <br />

            <label for="user-type">User Type:</label>
            <select id="user-type" name="UserType" required>
                <option value="">--Select User Type--</option>
                <option value="Customer">Customer</option>
                <option value="Manager">Manager</option>
                <option value="Local">Local</option>

            </select>
            <button type="submit">Register</button>
        </form>
    </div>
</body>
</html>
```

b) **Sign In:** Displays a form that enables the user to sign in into the application. Also, it is associated with the *SignIn* method from *UserController*.

```
@model TravelPlanner.Models.User

<!DOCTYPE html>
<html>
<head>
    <title>Login Page</title>
    <link href="~/Login.css" rel="stylesheet" type="text/css">
</head>
<body>
    <div class="login-container">
        <h1>Login</h1>
        <form method="post" action="@Url.Action("SignIn", "
    User")">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email"
    required>
            <br />

            <label for="password">Password:</label>
            <input type="password" id="password" name="
    password" required>
            <br />

```

```
            <button type="submit">Submit</button>
            <p>Don't have an account? <a href="@Url.Action("
    SignedUp","User")">
            Sign up here</a>.</p>
        </form>
    </div>
</body>
</html>
```

3.2.2 Accommodation View

a) **AccommodationView:** Displays all the accommodations from the database to the user and enables the search functionality. Also, in the last column of each row, there are three action links:

- The first link redirects to the *StoreAccommodationIdAndRedirect* from *Bookings Controller*.
- The second link redirects to the *DisplayReview* action from the *ReviewAccommodation Controller*.
- The third link redirects to the *ReviewAccommodationView* action from the *ReviewAccommodation Controller*.

```
@model IEnumerable<TravelPlanner.Models.Accommodation>

<!DOCTYPE html>
<html>
<head>
    <title>TravelPlanner - Home</title>
    <link rel="stylesheet" type="text/css" href="~/
    AccommodationActivity.css">
</head>

<br />
<br />

@using (Html.BeginForm("SearchAccommodation", "Accommodation",
     FormMethod.Get))
{
    <div class="form-group">
        @Html.Label("Search Accommodation:")
        @Html.TextBox("searchAccommodation", null, new {
    @class = "form-control" })
    </div>
```

```
19      <button type="submit" class="btn btn-primary">Search</
    button>
20  }
21
22  <table class="table">
23      <thead>
24          <tr>
25              <th>
26                  @Html.DisplayNameFor(model => model.
    AccomodationName)
27              </th>
28              <th>
29                  @Html.DisplayNameFor(model => model.
    AccommodationType)
30              </th>
31              <th>
32                  @Html.DisplayNameFor(model => model.
    AccommodationLocation)
33              </th>
34              <th>
35                  @Html.DisplayNameFor(model => model.Price)
36              </th>
37              <th>
38                  @Html.DisplayNameFor(model => model.
    AccomodationDescription)
39              </th>
40              <th>
41                  @Html.DisplayNameFor(model => model.CreatedAt)
42              </th>
43          </tr>
44      </thead>
45      <tbody>
46          @foreach (var item in Model)
47          {
48              <tr>
49                  <td>
50                      @Html.DisplayFor(modelItem => item.
    AccomodationName)
51                  </td>
52                  <td>
53                      @Html.DisplayFor(modelItem => item.
    AccommodationType)
54                  </td>
```

```
55                  <td>
56                      @Html.DisplayFor(modelItem => item.
    AccommodationLocation)
57                  </td>
58                  <td>
59                      @Html.DisplayFor(modelItem => item.Price)
60                  </td>
61                  <td>
62                      @Html.DisplayFor(modelItem => item.
    AccomodationDescription)
63                  </td>
64                  <td>
65                      @Html.DisplayFor(modelItem => item.
    CreatedAt)
66                  </td>
67                  <td>
68                      @Html.ActionLink("Book", "
    StoreAccommodationIdAndRedirect",
69                      "Bookings", new { accommodationId = item.
    AccommodationId,
70                      actionName = "BookAccommodationView",
    controllerName = "Bookings",
71                      id = item.AccommodationId }, null)
72                      @Html.ActionLink("Display reviews", "
    DisplayReview",
73                      "ReviewAccommodation", new {
    accommodationId =
74                      item.AccommodationId }, null)
75                      @Html.ActionLink("Leave Review", "
    ReviewAccommodationView",
76                      "ReviewAccommodation", new { id = item.
    AccommodationId }, null)
77                  </td>
78              </tr>
79          }
80      </tbody>
81  </table>
```

b) **NewAccommodation:** Displays a form to let the user add a new accommodation to the database.

```
1  <div class="text-center">
2  <h1 class="display-4">Add accommodation</h1>
3  <link href="~/Accommodation.css" rel="stylesheet" type="text/
```

```
    css">
<form method="post" action="@Url.Action("CreateAccomodation",
    "Accommodation")">
    <label for="AccommodationName">Accommodation Name:</label>
    <input type="text" id="AccommodationName" name="
    AccomodationName" required>

    <label for="AccommodationType">Accommodation Type:</label>
    <input type="text" id="AccommodationType" name="
    AccommodationType"
    required>

    <label for="AccommodationLocation">Accommodation Location:
    </label>
    <input type="text" id="AccommodationLocation" name="
    AccommodationLocation" required>

    <label for="AccommodationDescription">Accommodation
    Description:</label>
    <input type="text" id="AccommodationDescription" name="
    AccomodationDescription"
    required>

    <label for="Price">Price:</label>
    <input type="text" id="Price" name="Price" required>

    <button type="submit">Submit</button>
</form>
</div>
```

3.2.3 Activity View:

It contains two views, *ActivityView* and *NewActivity*, built in the same manner as *AccommodationView* and *NewAccommodation*.

3.2.4 Bookings View

a) **BookAccommodationView:** The View for the *BookAccommodationView* method from *Bookings Controller*.

```
@model TravelPlanner.Models.BookAccommodationViewModel

@{
    ViewData["Title"] = "Book Accommodation";
```

```
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<head>
    <title>Book Accommodation</title>
    <link rel="stylesheet" type="text/css" href="~/BookingsView.css">
</head>


<h2>Book Accommodation</h2>

<h3>@Model.AccommodationName</h3>
<p>@Model.AccommodationDescription</p>
<p>@Model.AccommodationType</p>
<p>@Model.Price</p>

<form action="@Url.Action("BookAccommodation", "Bookings")" method="post">
    @Html.HiddenFor(model => model.AccommodationId)
    @Html.HiddenFor(model => model.AccommodationName)
    @Html.HiddenFor(model => model.AccommodationDescription)
    @Html.HiddenFor(model => model.AccommodationType)
    @Html.HiddenFor(model => model.Price)

    <label for="checkInDate">Select Check-In Date:</label>
    <input type="date" id="checkInDate" name="CheckInDate">
    <label for="checkOutDate">Select Check-Out Date:</label>
    <input type="date" id="checkOutDate" name="CheckOutDate">
    <br>

    <img src="~/Images/random-photo.jpg" alt="Random Photo">
    <br>

    <button type="submit" onclick="redirectToConfirmation()">Book</button>
</form>

<script>
    function redirectToConfirmation() {
        document.getElementById("bookForm").submit();
        var bookingId = @Model.BookingId;
```

```
        window.location.href = '@Url.Action("Confirmation", "Bookings")' +
        '?bookingId=' + bookingId;
    }
</script>
```

b) **Confirmation:** This is the corresponding View for the *Confirmation* method from *Bookings Controller*.

```
@model TravelPlanner.Models.Bookings

@{
    ViewBag.Title = "Confirmation";
}

<h2 style="color: #333; font-size: 24px; margin-bottom: 10px;">Confirmation</h2>

<p style="color: #666; font-size: 16px; margin-bottom: 10px;">Check-in Date:
<span style="font-size: 14px; color: #4CAF50;">@Model.CheckInDate</span></p>
<p style="color: #666; font-size: 16px; margin-bottom: 10px;">Check-out Date:
<span style="font-size: 14px; color: #4CAF50;">@Model.CheckOutDate</span></p>
<p style="color: #666; font-size: 16px; margin-bottom: 10px;">Total Price:
<span style="font-size: 14px; color: #4CAF50;">@Model.TotalPrice</span></p>
<p style="color: #666; font-size: 16px; margin-bottom: 10px;">Created At:
<span style="font-size: 14px; color: #4CAF50;">@Model.CreatedAt</span></p>
```

3.2.5 Bookings Activity View:

It contains two views, *BookActivityView* and *ConfirmationActivity*, built in the same manner as *BookAccommodationView* and *Confirmation*.

3.2.6 Home View

a) **Index:** Our startup page, where the user has the option to sign up, log in (if they already have an account) or explore our list of recommended destinations.

```
<!DOCTYPE html>
<html>
<head>
    <base href="@Url.Action("Start", "Home")" />
    <title>TravelPlanner - Home</title>
    <link rel="stylesheet" type="text/css" href="~/HomePage.
    css">
</head>
<body>
    <header class="header" style="display: flex; justify-
    content: space-between;">
        <h1>Welcome to TravelPlanner</h1>
    </header>
    <br /><br />
    <div class="Message" style="display: flex; justify-content
    : flex-end;">
        <div style="flex: 1;">
            <img src="~/bus.png" alt="Travel Photo" style="
    float: left">
            <h2>Discover Your Next Adventure</h2>
            <p>At TravelPlanner, we're passionate about
    helping you plan the perfect
            trip. Whether you're looking for a relaxing beach
    getaway or an exciting
            city adventure, our travel experts are here to
    help.</p>
            <a href="@Url.Action("About", "Home")" class="btn"
    >Explore Destinations</a>
        </div>
    </div>
</body>
</html>
```

b) **About:** Displays the recommended destinations mentioned in **Section2.1**,**Figure9**.

```
@model TravelPlanner.Models.
    DestinationRecommandationDefinition

<!DOCTYPE html>
<html>
<head>
    <base href="@Url.Action("About", "Home")" />
    <title>Our recommandations on destionations</title>
    <style>
```

```
        .city-image {
            width: 500px;
            height: 200px;
            object-fit: cover;
        }
    </style>
</head>
<body class="u-body u-xl-mode" data-style="blank" data-posts="
    " data-global-section-properties="{&quot;colorings&quot
    ;:{&quot;light&quot;:[&quot;clean&quot;],&quot;colored&
    quot;:[&quot;clean&quot;],&quot;dark&quot;:[&quot;dark&
    quot;]}}" data-source="" data-lang="en" data-page-sections
    -style="[{&quot;name&quot;:&quot;blank&quot;}]" data-page-
    coloring-types="{&quot;light&quot;:[&quot;clean&quot;],&
    quot;colored&quot;:[&quot;clean&quot;],&quot;dark&quot;:[&
    quot;dark&quot;]}" data-page-category="&quot;Basic&quot;">
     <section class="u-clearfix u-block-a769-1" custom-posts-
    hash="[]" data-style="blank" data-section-properties="{&
    quot;margin&quot;:&quot;none&quot;,&quot;stretch&quot;:
    true}" id="sec-75f4" data-id="a769">
        <div class="u-clearfix u-sheet u-block-a769-2">
            <div class="u-clearfix u-layout-wrap u-block-a769
    -3">
                <div class="u-layout">
                    @for (int i = 0; i < TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations.Count;
     i += 2)
                    {
                        <div class="u-layout-row">
                            <div class="u-container-style u-
    layout-cell u-size-30 u-block-a769-4">
                                <div class="u-container-layout
     u-block-a769-5">
                                    <h2>@TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations[i].
    CityName</h2>
                                    <img class="city-image"
    src="@Url.Content("~/Images/" + TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations[i].
    CityName + ".jpg")" alt="@TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations[i].
    CityName" />
                                    <p>@TravelPlanner.Models.
```

```
    DestinationRecommandationDefinition.recommandations[i].
    CityDescription</p>
29                                  </div>
30                              </div>
31                              @if (i + 1 < TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations.Count)
32                              {
33                                  <div class="u-container-style
    u-layout-cell u-size-30 u-block-a769-6">
34                                      <div class="u-container-
    layout u-block-a769-7">
35                                          <h2>@TravelPlanner.
    Models.DestinationRecommandationDefinition.recommandations
    [i + 1].CityName</h2>
36                                          <img class="city-image
    " src="@Url.Content("~/Images/" + TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations[i+1].
    CityName + ".jpg")" alt="@TravelPlanner.Models.
    DestinationRecommandationDefinition.recommandations[i+1].
    CityName" />
37                                          <p>@TravelPlanner.
    Models.DestinationRecommandationDefinition.recommandations
    [i + 1].CityDescription</p>
38                                      </div>
39                                  </div>
40                              }
41                          </div>
42                      }
43                  </div>
44              </div>
45          </div>
46
47      </section>
48  </body>
49  </html>
```

3.2.7 Review Accommodation View

a) **DisplayReviewAccommodation:** The corresponding View for the *DisplayReview* method from *ReviewAccommodation Controller*.

```
1  @model List<TravelPlanner.Models.ReviewAccommodation>
2
```

```
@{
    ViewBag.Title = "DisplayReview";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<!DOCTYPE html>
<html>
<head>
    <title>TravelPlanner - Home</title>
    <link rel="stylesheet" type="text/css" href="~/
   DisplayReview.css">
</head>
<br />
<br />


<h2>Accommodation Review</h2>

@foreach (var review in Model)
{
    <div>
        <strong>Rating: </strong> @review.Rating
    </div>
    <div>
        <strong>Comment: </strong> @review.Comment
    </div>
    <div>
        <p>Created At: </p> @review.CreatedAt
    </div>
}
```

b) **ReviewAccommodationView:** Is linked with the *ReviewAccommodationView* method from the *ReviewAccommodation Controller*.

```
@model TravelPlanner.Models.AccommodationReviewViewModel
@{
    ViewBag.Title = "Review Accommodation";
    Layout = "~/Views/Shared/_Layout.cshtml";
}


<!DOCTYPE html>
<html>
<head>
```

```
    <title>TravelPlanner - Home</title>
    <link rel="stylesheet" type="text/css" href="~/Reviews.css
    ">
</head>

<br />
<br />



<h2>Review Accommodation</h2>
<h3>@Model.AccommodationName</h3>

<form action="@Url.Action("LeaveReview", "ReviewAccommodation"
    )" method="post">
    @Html.HiddenFor(model => model.AccommodationId)
    @Html.HiddenFor(model => model.AccommodationName)

    <div class="form-group">
        @Html.LabelFor(model => model.Rating)
        @Html.TextBoxFor(model => model.Rating, new { @class =
    "form-control" })
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Comment)
        @Html.TextBoxFor(model => model.Comment, new { @class
    = "form-control" })
    </div>

    <button type="submit" class="btn btn-primary">Submit
    Review</button>
</form>
```

3.2.8 Review Activity View

It contains two views, *DisplayReviewActivity* and *ReviewActivityView*, built in the same manner as *DisplayReviewAccommodation* and *ReviewAccommodationView*.

4 Application Testing

4.1 Introduction to Testing

Software testing plays a crucial role in ensuring the reliability and functionality of applications, including complex ones like travel planning applications. Thus, with the potential for unexpected failures when using an application, software testing becomes essential before releasing an application to the public. This necessity is emphasized by a quote that states: "As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications." (Dave Parnas)

The realm of automated testing offers various techniques tailored to software applications. The most straightforward approach is unit testing, followed by integration and functional testing at a higher level. Additionally, there are load tests, stress tests, and smoke tests, which are employed in more advanced stages of testing, but were not necessary for our project.

Testing can be performed either manually or through automation, with each method having its advantages. While manual testing is often perceived as slower, there are instances where it can be a viable option.

4.2 Testing Procedure for Our Application

In the case of our travel planning application, we opted for manual testing and employed an organized approach by utilizing an Excel table to document and track every aspect of the testing process. This method allowed us to efficiently gather and organize all testing data, as well as record the corresponding results.

Within the Excel table, we created different sections to categorize the various aspects of our testing. This included separate columns to capture test cases IDs, test cases descriptions, test cases steps, test data, actual results, and test results tagged as pass, fail, not executed or suspended. By structuring the table in this manner, a comprehensive overview of the testing process was ensured.

Moreover, we employed a step-by-step approach so as to identify any inconsistencies, issues, or bugs in the application. The Excel table allowed for tracking the progress of testing by highlighting completed test cases and pending ones. Henceforth, the status of each test was identified and thus, any outstanding tasks were prioritized.

Overall, the testing procedure served as a reliable and efficient method for capturing, analyzing, and documenting the testing data, and ensuring a thorough evaluation of the travel planning application.

5 Conclusion

We have reached the end of our project and in the end is fair to say that we have reached our goals in terms of implementing the functionalities. For sure, it can use some touch-ups here and there from a design standpoint but it is a good start for an app that we think will surely be a top seller.