Infs3017/Infs3018

Web Development Project

Due: 9pm, Fri 28 Oct 2022

This project is to build a website for the fictitious **Western Inn** that serves the visitors to WSU, as well as the public. This website should allow guests to search and book rooms, and allow administrators to offer discounts to university visitors and view statistics, etc.

You are suggested to read the entire specification first, and then start with the tasks that are already covered by our lectures.

1 Creating project in Visual Studio (VS)

Name this project "WesternInn_XX_YY_ZZ", where XX, YY, ZZ are the initials of your group members.

Since the project needs to support authentication for guests and administrators, it needs the ASP.NET Identity package in the very beginning. To include this package in your project, you should carefully follow the slides in our Lecture 9, which is released on vUWS already.

During the above process, you should name the SQLite database file "WesternInn.db". In this project, you are going to use it to store data not only for authentication, but also for the Western Inn.

Moreover, you should follow the slides in Lecture 9 to scaffold the source code of the Identity package into your project.

Finally, when you run this project, you should see the Register and Login links appear in the right of the navigation bar.

2 Two roles of administrators and guests

Besides maintaining users' credentials, the Identity package can also assign users with different roles. You are required to use this package to divide the users of this website into two roles: **administrators** and **guests**. You should create only one user with the '**administrators**' role, and set up his/her username to be `'admin@westerninn.com'` and his/her password to be `'P@ssw0rd'` when the web application starts. On the other hand, all users registering into the website by clicking the 'Register' link mentioned in the Section 3 below should be assigned the '**guests**' role.

Moreover, source code should be added to the Pages/Shared/_Layout.cshtml file such that, after logging in, administrators and guests should only see links that are accessible to them in the navigation bar. (see the details in the Section 3 below)

NB: We will discuss the above in our Lecture 9. You are welcome to study Lecture 9 in advance.

3 Navigation and layout

All pages in your website should share a consistent layout, which has a navigation bar in the top. This should be achieved by using _Layout.cshtml and Bootstrap.

The links contained in the navigation bar should be dynamic. The detailed requirements are as follows:

- The links for non-logged-in users should include the following: Home, Privacy, Register, and Login. Note that the Login link here is used by both guests and administrators.
- The links for logged-in guests should include the following: Home, Privacy, My Details, Search Rooms, Book A Room, My Bookings and Logout.

- The links for logged-in administrators should include the following: Home, Privacy, Manage Bookings, Statistics and Logout.

Note: How to make the links dynamic are discussed in Lecture 9. You can start with having all the links in the navigation bar, and make them dynamic after Lecture 9.

4 Home page

The Home link in the navigation bar leads to this page, which should display:

- A carousel below the navigation bar. This carousel should rotate five pictures about the Western Inn. The pictures can be about buildings, rooms, dining, etc. and can be downloaded from Internet and then modified. When searching images on the Internet, you should use royalty-free image websites such as Pixabay (www.pixabay.com) and Unsplash (www.unsplash.com) to avoid copyright issues. Each picture needs to have a caption when being displayed by the carousel.
- Two columns below the carousel. The left column should consist of 7 basic columns, and display a welcome message and a brief introduction to the hotel. The right column should consist of 5 basic columns, and display a list of links about our university (e.g., a link to our uni home page, a link to Parramatta City Council, etc.). The two columns should stack up when the viewport width is less than 768px (NB: choose the right column class to use in Bootstrap).

5 Models

This website should use the following three Model classes. Each class has certain properties. The requirements on these properties are also described below. You should apply appropriate data types and data annotations to fulfil these requirements.

You should fully create these Model classes first, and then scaffold them one by one, and then migrate them to database together.

NB: During scaffolding, you should choose the ApplicationDbContext used by Identity as the DbContext (i.e., select the database used by Identity). During migration, in case you see the complaint that SQLite does not support certain operation, you should comment out the migration code in the Up() and Down() methods related to that operation. These two methods can be found in the .cs file under the 'Migrations' folder.

5.1 Room.cs

The Room class models the rooms in Western Inn. It should have the following properties.

Property	Data Type	Requirements
ID	int	Primary key;
Level	string	Meaning the level of this room; Exactly one character of ‘G’, ‘1’, ‘2’, or ‘3’. Required.
BedCount	int	Meaning the number of beds in the room; can only be 1, 2, or 3.
Price	decimal	Meaning the price per night; Between $50 and $300.
TheBookings	ICollection<Booking>	This is a navigation property (Note: will be discussed in Lecture 8)

After scaffolding and migrating this class into database, you should use the https://localhost:xxx/Rooms web interface to manually populate the Room table with the following data. (NB: For the convenience of marking, you must use the data below.)

ID	Level	BedCount	Price
1	1	1	146.30
2	1	2	156.30
3	1	3	166.80
4	1	2	150.80
5	2	1	156.30
6	2	2	166.30
7	2	3	176.80
8	2	2	160.80
9	3	1	136.30
10	3	2	146.30
11	3	3	156.80
12	3	2	140.80
13	G	1	126.30
14	G	2	136.30
15	G	3	146.80
16	G	2	130.80

5.2 Guest.cs

The Guest class models all guests who have ever registered with Western Inn. It should have the following properties.

Property	Data Type	Requirements
Email	string	Primary key; Required; Valid email address; Since its name doesn't follow the convention to be the primary key, you should add the [Key] annotation to indicate that this property is the primary key, and also add the [DatabaseGenerated(DatabaseGeneratedOption.None)] annotation to prevent its value from being automatically generated by database.
Surname	string	Required; Length ranges between 2 and 20 characters inclusive; Can only consist of English letters, hyphen and apostrophe.
GivenName	string	Same as above.
Postcode	string	Required; exactly 4 digits.
TheBookings	ICollection<Booking>	This is a navigation property (Note: will be discussed in lecture 8)

Notes:

- This class has no property for password hashes. Guests' password hashes will be automatically managed by the ASP.NET Identity package, and will not be stored here.
- You should use the 'Register' link in the website to register a guest into the website, and then use the 'My Details' link (described in Section 6) to enter the details for this guest. Do this for at least 10 guests with some common postcodes to allow the 'Statistics' link to show results.

5.3 Booking.cs

The Booking class models all bookings ever made by guests with Western Inn. It should have the following properties.

Property	Data Type	Requirements
ID	int	Primary key;
RoomID	int	Foreign Key (NB: since the name follows convention, this will be automatically figured out by EF Core)
GuestEmail	string	Same as above;
CheckIn	DateTime	Meaning the checking-in date; Only Date part is needed.
CheckOut	DateTime	Meaning the checking-out date; Only Date part is needed.
Cost	decimal	The total cost of this booking. Range from $0.00 to $10,000.00 inclusive.
TheRoom	Room	This is a navigation property (Note: will be discussed in lecture 8)
TheGuest	Guest	Same as above

Notes:

- Each booking can only consist of one room. If multiple rooms are needed by a guest, the guest needs to make multiple bookings.
- You should add booking records into the Booking table by using the 'Book A Room' link. However, before the website is fully functioning, you can also use the SQlite DB Browser to manually add records into this table.
- For properties related to finance, the data type ‘decimal’ is preferred to ‘float’ or ‘double’. See: https://stackoverflow.com/questions/618535/difference-between-decimal-float-and-double-in-net .

6 The links for logged-in guests

This section gives the requirements for the links in the navigation bar for logged-in guests. You should figure out the implementation of the corresponding Razor page for each link based on the requirements.

Moreover, you should use the [Authorize] annotation to only allow the users with the role of 'Guests' to access these four links below.

6.1 The 'My Details' link

This link should allow a registered/logged-in guest to enter/modify the following details: Surname, Given Name, and Postcode. Entering details is the first thing a newly-registered guest should do. A newly-registered guest should be redirected to this 'My Details' page upon successful registration.

If this is the first time that a guest enters his/her details, a new record should be created in the Guest table with the guest's Email property taken from the Identity package. If the guest's details already exist in the database, this page should allow the guest's details to be modified.

The guest should be acknowledged with his/her latest details upon successful entering or modification to his/her details.

Hint: you can refer to Lecture 9 slides on how to implement this page.

6.2 The 'Search rooms' link

This link should allow a logged-in guest to search available hotel rooms by displaying a web form

with the following input devices, all of which are mandatory:

1. The number of beds (Should be implemented with a select tag helper, consisting of options for 1 bed, 2 beds and 3 beds. You can either use static options of 1, 2 and 3 beds within the select tag helper, or use dynamic options by querying the database to obtain possible bed counts. If you choose to use dynamic options, you need to explore how to obtain 'distinct' bed counts by using SelectList() method.)
2. CheckIn date (Only Date part is needed; Should be displayed by an input tag helper)
3. CheckOut date (Only Date part is needed; Should be displayed by an input tag helper)
4. Submit button

You should use a View Model (refer to Lecture 7) to specify the annotations and validations for the input devices in this web form. Once the form submission is successfully validated, you should construct a raw SQL query (discussed in Lecture 12) to search the available rooms. Note that you should use a single query, not multiple queries. While this query is a little complex, we provide following hints:

- Suppose there are two bookings of the same room: bookingA and bookingB. The exact condition for these two bookings to have no overlap can be expressed as follows:

  ```
  bookingA.Checkout <= bookingB.Checkin OR bookingB.Checkout <=
                                  bookingA.Checkin
  ```

  Then, it is easy to understand that the equivalent condition for these two bookings to have overlap is:

  ```
  bookingA.Checkin < bookingB.Checkout AND bookingB.Checkin <
                                  bookingA.Checkout
  ```

- The structure of your single SQL query should be as follows: first, use a main query to find all rooms satisfying the bed count requirement; then, use a subquery to find all rooms that have overlaps with the guest′s intended period; finally, use the NOT IN operator to exclude those rooms found by the subquery from the main query. For how to construct a single query consisting of a main query and a subquery, refer to Lecture 12.

After the SQL query returns, you should display the results in a table underneath the ‘Submit’ button. This table should contain the following properties from the **Room** class: ID, Level, BedCount, and Price.

The Room ID displayed in the above table should be a hyperlink, which points to the ‘Book a room’ page below and is appended with a query string comprising the following values: Room ID, CheckIn date, and CheckOut date. If this hyperlink is clicked, the ‘Book a room’ page will be opened with the values in the query string displayed in the web form.

Hint: First, be clear on the format of the query string. Second, the names in the query string should exactly use those of the input devices in the web form in Subsection 6.3. You can view the page source of the web form in 6.3 via a web browser to be sure of them. The values in the query string are just strings with no quotes needed. The Model Binding in Razor pages will ensure they are interpreted correctly.

6.3 The 'Book a room' link

This link should allow a logged-in guest to book a hotel room by displaying a web form with the following input devices, all of which are mandatory:

1. The room ID (should range between 1 and 16).
2. CheckIn date (Only Date part is needed; Should be displayed by an input tag helper)
3. CheckOut date (Only Date part is needed; Should be displayed by an input tag helper)
4. Submit button

A guest can book a room by filling the above form. The form should be submitted by POST method. After the user inputs are successfully validated, it is still needed to check whether the submitted booking is available. Please construct a raw SQL query to do this based on the hints given in Subsection 6.2.

If this booking is NOT available, you should display the above form again and show a message for the guest to advise this unavailability. If available, you should insert a record about this booking into the **Booking** table. Note that, the cost of the booking should equal the room's price per night times the number of nights (you should figure out how to calculate this by C#).

If the 'insert' is successful, you should inform the guest with the following info below the web form above: the room booked, the level, the CheckIn and CheckOut dates, and the cost of this booking.

Moreover, this web form should be able to receive the query string values composed in Subsection 6.2 and display them in the form. Then, the guest can further modify the inputs if he/she wants, otherwise he/she can click the submit button.

Hint: To enable the reception of query string values, the Model variable for the above web form inside the PageModel class should have the annotation `[BindProperty(SupportsGet = true)]`.

6.4 The 'My bookings' link

This link should allow a logged-in guest to see all his/her bookings. After clicking this link, a table with the following columns should be displayed to the guest: Surname, given name, Room ID, Check-in date, Check-out date, and cost.

Note that only the bookings by this guest should be displayed.

Moreover, the order for displaying the records can be adjusted by clicking the following two column headings of the table: Check-in date and cost. That is, clicking the 'check-in date' heading will toggle the display order of the bookings between ascending and descending in terms of check-in date, and similarly, clicking the 'cost' heading will toggle the display order of the bookings between ascending and descending in terms of cost.

Hint: you can refer to Lecture 11 slides on how to implement this page.

7 The links for logged-in administrators

This section gives the requirements for the links in the navigation bar for logged-in administrators. You should figure out the implementation of the corresponding Razor page for each link based on the requirements.

Moreover, you should use the [Authorize] annotation to only allow the users with the role of 'Administrators' to access these two links below.

7.1 The 'Manage Bookings' link

This link should lead to a web interface that allows a logged-in administrator to manually enter a new booking, edit an existing booking, and delete a booking. (Hint: the implementation of this interface can reuse some of the scaffolded source code from the Booking model.)

Specifically, after clicking this link, a web page with the following contents should be displayed:

1. A 'Create' link for creating a new booking in the upper part of the page.
2. Below this link, a table of all existing bookings. The headings of this table should include Room ID, Guest Surname, Guest Given Name, Checking in Date, Checking out Date, and Cost. Note that the Surname and Given Name should use two columns here, which is different from the default scaffolded code.
3. At the end of each table entry, two links ('Edit' and 'Delete') for editing and deleting this table

entry respectively should be displayed. Note that we don't require the 'Details' link here, which is different from the default scaffolded code.

When the 'Create' link in the top or the 'Edit' link in a table entry is clicked, the administrator should be led to a new page with a web form, which contains the following input devices for creating/modifying the booking:

1. The room ID (should range between 1 and 16).
2. A dropdown list consisting of current guests' full names (Hints: option value = a guest's email, option text = a guest's full name; you need to add a [NotMapped] 'FullName' property to the Guest model; you need to retrieve this list of guests from the database dynamically by calling 'SelectList()'; refer to our sample projects)
3. Checking in date
4. Checking out date
5. Cost (NB: here we don't require the Cost should equal price * number of nights; we give administrators' the privilege to apply a discount to the cost, i.e., reduce the cost at their discretions.)
6. Submit button

All of the above input devices are mandatory.

After the form submission is successfully validated, you also need to check whether the submitted booking is available. Please construct a raw SQL query to do this (similar to the one in Subsection 6.3).

If this booking is unavailable, you should display the above form again and also show a message for the administrator to advise this unavailability. If available, you should insert/update about this booking into the **Booking** table, and then redirect the administrator to the 'Manage Bookings' page.

When the 'Delete' link for a table entry is clicked, the administrator should be led to a new page which:

1. displays the details of this booking. The details should include: Room ID, Room Level, Bed Count, Guest Surname, Guest Given Name, Checking in date, Checking out date, cost.
2. displays a 'Delete Confirmation' button beneath the details.

After the 'Delete Confirmation' button is clicked, this booking should be removed from the Booking table and the administrator should be redirected back to the initial 'Manage Bookings' page.

7.2 The 'Statistics' link

This link allows a logged-in administrator to view:

- How many guests are from each postcode.
- How many bookings have been made for each room.

After this link is clicked, the administrator should see a page that displays the following contents.

1. A <h2> heading "Guest distribution with respect to postcodes".
2. Beneath this heading, a table with two columns ('Postcode' and 'Number of Guests').
3. A <h2> heading "Booking distribution with respect to rooms".
4. Beneath this heading, a table with two columns ('Room ID' and 'Number of Bookings').

Hints:

- See Lecture 11 slides for how to calculate statistics.
- See the PeopleDiff example from Lecture 12 for more clues.

8 Project Administration and Marking

This group project is introduced since the curriculum of your degree requires certain portion of group work. More importantly, if you look at the selection criteria of any advertised jobs, there is at least one criterion about teamwork. Hence, we strongly urge you to apply your collaboration skills into this group project.

8.1 Project group formation and responsibilities

Each project group should consist of 2-3 students. For more detailed group formation rules, please refer to the Lecture 1A slides.

Your group should meet on a regular basis (at least once a week). Each member of the group should contribute equally to the project. It is the group's responsibility to allocate tasks to each member evenly, monitor the progress of each member, and make the collaboration successful.

8.2 Group meetings with tutor

Each group is required to have two meetings with your tutor before project completion. The purpose of these meetings is to discuss the group and individual progress on the project and to obtain feedback from the tutor. The tutor meetings are conducted in your practical classes (see the Section 3 of the Learning Guide for the schedule). All group members should be present for the meeting and be ready by the time the tutor calls your group. If a group member is absent, he/she will receive zero mark for the group meeting.

8.2.1 The meeting 1 with tutor

In this meeting, each group should first let your tutor know the members in this group. Your tutor will then register your group on our subject's vUWS website. This registration is a must and a convenience for you because:

- It enables your group to submit the project, otherwise your group cannot.
- When submitting the project, only one group member needs to submit, and then all group members can see the submission and receive the project mark and feedback via rubric.
- It enables you to use group collaboration tools such as 'Discussion Board', 'File sharing', etc. on vUWS. The way of accessing these tools is to click the "My Groups" link located in the bottom left corner.

Besides informing your group formation to your tutor, your group should provide the following items for discussion and marking:

- Task distribution among group members. Each group should write this down in a 1-page document, and discuss it with your tutor. Tutor should check on the equal allocation. (No Marks)
- Section 1 should be completed: (1 mark)
    - the website should run with 'Register' and 'Login' links displayed;
    - both 'Register' and 'Login' links function correctly;
    - the newly registered user can be found in the 'AspNetUsers' table in the Sqlite DB file.

    NB: all these three bullet points together account for the 1 mark of this group meeting. Any incompletion will result in a zero mark.

8.2.2 The meeting 2 with tutor

Items to be provided at this meeting for discussion and marking with your tutor include:

- Progress on the project. Each group should write down the progress of each member in a 1-page

document, and discuss it with your tutor. Tutor should check whether the progress can ensure the timely completion of the project. (No marks)

- The populated database: (1 Mark)
    - All the three Model classes mentioned in Section 5 should be migrated into database correctly.
    - The Room table is populated as required in Section 5.
    - There should be at least 10 guests and 15 bookings added into the database. This is to ensure that there are sufficient numbers of records for testing and marking purposes. Note that you are allowed to use the SQLite DB Browser to manually add records into database. However, all the records added must satisfy the foreign key constraint and the checkin and checkout dates must be feasible.

NB: All the three bullet points together account for the 1 mark of this group meeting. Any incompletion will result in a zero mark.

8.3 Confidential self and peer assessment

To urge each group member to contribute equally toward the project, each student is asked to submit a confidential peer assessment form, in which you can rate each group member's efforts by a percentage. When we give project mark to a group member, we multiply the group project mark with the average of the percentages he/she received (see the Subsection 8.5 for details). We maintain the right to adjust the percentages received by a group member based on our observations. This is to prevent dishonest percentages from being given toward a group member.

8.4 Items to submit for the Project

Each student should submit the **confidential self and peer assessment form** through the submission link in vUWS.

Each group should submit:

- A zip file of your ASP.NET Core Project.
- Readme.txt. This plain text file should provide the following info to markers:
    - What are the usernames and passwords for the guests and administrators respectively to access your website?
    - Who are the group members?
    - Special notes to tutors (optional)
- Completed declaration document (download at vuws).

Note that only one member of the group needs to do the submission for the above items. The submission steps are as follows:

1. Place the Readme.txt and the completed Declaration Document in the root folder of your project.
2. Zip the entire project folder into a zip file, and submit it by following the procedure below:
    - In vUWS, click Project in the left column, and then Project Submission, and attach your zip file and submit.
    - Download your submitted zip file from vUWS to a different location in your computer. Check if you can run it as expected in Visual Studio.

Note: You can submit the project multiple times until the due time, and your latest submission will be marked. If the submission is late, the late penalty will be applied to all members of the group.

8.5 Project Marking

The marking rubric will be available on vuws via My Grades by Week 9.

Out of the 25 marks for the project, 2 marks are for the two group meetings, and 23 marks are for the project. The mark obtained for the project will then be adjusted for each group member based upon the average rating this member receives from all members. The application of this average rating is demonstrated in the following example:

Suppose Group S has 3 members (Student 1, Student 2, Student 3), and the following marks are achieved:

Group Project mark = 20 out of 23,

Student 1 tutor meeting mark = 1 out of 2

Student 2 tutor meeting mark = 2 out of 2

Student 3 tutor meeting mark = 1 out of 2

Student l provided the following confidential assessments on members' individual contributions:

Student 1 (self) 90/100,

Student 2 100/100,

Student 3 60/100.

Student 2 provided the following confidential assessments on members' individual contributions:

Student 1 80/100,

Student 2 (self) 100/100,

Student 3 50/100.

Student 3 provided the following confidential assessments on members' individual contributions:

Student 1 80/100,

Student 2 100/100,

Student 3 (self) 60/100.

The average ratings derived from the above confidential peer assessments are therefore:

Student 1 = (90+80+80)/3=83.3

Student 2 = 100

Student 3 = 56.6

Allocated individual marks for the project are therefore derived as:

Final Project Mark = Average Rating × Project Mark + Individual tutor meeting mark

For each of the students in Group S, the Final Project Mark is calculated as below:

Student 1, 83.3/100 x 20 + 1

Student 2, 100/100 x 20 + 2

Student 3, 56.6/100 x 20 + 1