

# HOUSERENTING HRS

Web application for house renting system created with C# programming language and .NET core 6 framework some provides a secure, efficient, and user-friendly online environment

ITPE3200-1 23H

# The Next Strongest Competitor for Airbnb

Username for Admin: <a href="mailto:admin@hrs.com">admin@hrs.com</a> password: Aa.1234
--

**Group: Group 5**

Members:

**Student-1** Kenan Hasan Abou Shakra **s345510**

**Student-2** Shatha Mashhour Amer **s362064**

**Student-3** Hiba Abdalaziz Taha Mohmmed **s351932**

**Student-4** Duaa Salah Krenbeh **s362058**

## Table of Contents

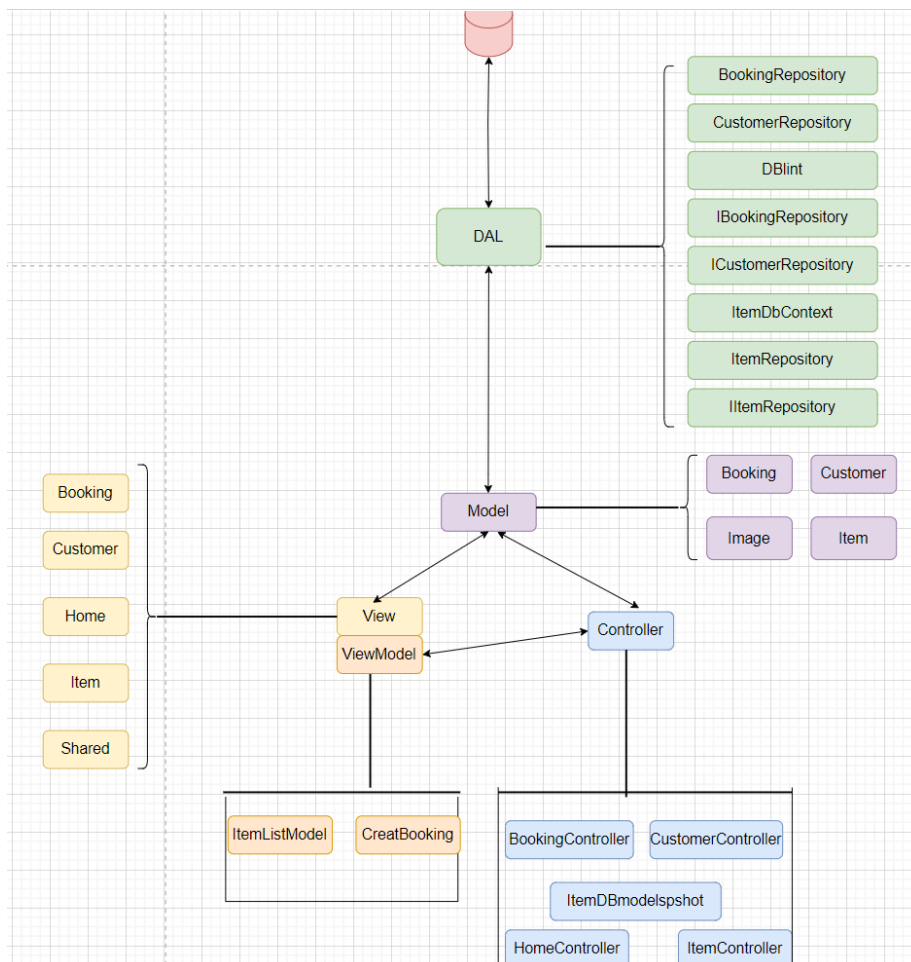
<b><u>1 INTRODUCTION</u></b> .....	<b><u>2</u></b>
<b><u>2 ARCHITECTURE</u></b> .....	<b><u>2</u></b>
<b><u>2.1 MODEL</u></b> .....	<b><u>3</u></b>
<b><u>2.2 VIEWS</u></b> .....	<b><u>4</u></b>
<b><u>2.3 CONTROLLERS</u></b> .....	<b><u>10</u></b>
<b><u>2.4 DATA ACCESS LAYER</u></b> .....	<b><u>11</u></b>
<b><u>3 DATABASE</u></b> .....	<b><u>12</u></b>
<b><u>4 CODING DETAILS</u></b> .....	<b><u>13</u></b>
<b><u>5 CONCLUSION</u></b> .....	<b><u>14</u></b>
<b><u>6 DESIG</u></b> .....	<b><u>15</u></b>
<b><u>REFERENCE LIST</u></b> .....	<b><u>16</u></b>

# 1 Introduction

Our project is a website of House renting management system, similar to Airbnb

The code project is based on .Net 6.0 and the Model-View-Controller framework (MVC), our project developed a web application that simplifies a house renting processes. The project leverages the power of C# and the development of Visual Studio to create an intuitive platform that addresses the challenges associated with property search, rental management, and user experience.

## 2 Architecture



## 2.1 Models

We have four data models

### 2.1.1 Booking.cs:

C# class refers three (Id)s as integer for Booking, customer and item which including the Booking information as well we write BookingData class to present the data of the booking , in addition we have Customer and items classes as navigation properties

### 2.1.2 Customer.cs:

C# class defines the structure of customer information and enforces validation rules for the customer's name, email, and phone number. It also provides a way to associate customers with their bookings.

For example this code line here

```
[RegularExpression(@"[0-9a-zA-ZæøåÆØÅ. \-]{2,20}", ErrorMessage = "The Name must be numbers or letters and between 2 to 20 characters.")  
[Display(Name = "Customer.Name")]
```

ensures that the Name must consist of 2 to 20 characters, which can be letters, numbers, or specific special characters like ".", "-", and the Norwegian characters æ, ø, å, Æ, Ø, Å. (same for the Email and let different for the phone number).

When we come to this code line which is property that represents a collection of bookings associated with the customer. In this part The List<Booking>? indicates that a customer may have zero or more bookings

```
public virtual List<Booking>? Bookings {get; set;}
```

### 2.1.3 Image.cs:

A C# class that represents images used in the items with in context of a houses. In the code we define **public class** for (declares the class), **public int** (for the image Id) **public virtual** to connect the image with specific item and **public String** for image URL (file path for the image)

### 2.1.3 Item.cs:

This class defines the properties and validation rules for items in Our project It includes information such as category, location, room count, area, rent, description, and associated images and bookings.

We start the code by declaring the class then define the item Id using integer for identify the item after that we used **public string Category { get; set; } = string.Empty** to represent he category of the item such as apartment or house It has a default value of an empty string in addition we insert an error message for the rang of room number, area of the property and rental prices,

```
[Range(1,24, ErrorMessage = "The numbers of rooms must be greater than 0.and less then 20")]
```

```
public int Rooms { get; set; }
```

```
[Range(5, int.MaxValue, ErrorMessage = "The Area must be greater than 5m.")]
```

```
public int Area {get; set; }
```

```
[Range(500, int.MaxValue, ErrorMessage = "The Renting must be greater than 500 kr.")]
```

## 2.2 Views

Here we present a description of layout, body, and navigation in five different views as following.

### 2.2.1 Booking

Contin of 4 different razor views

-**AllBookingsTable.cshtml** Here we have a HTML tag that includes a partial view named "\_BookingTable" and assigns the "Model" to it.

- **CreateBookingItem.cshtml**

We have here a razor view code which creates a form for users to book Houses. Most important fetcher in this file are displaying validation messages and it's achieved by this code line **<div asp-validation-summary="All" class="text-danger">** in addition we used these two lines of codes which retrieves the user's name and email from the claims **@User.FindFirstValue(ClaimTypes.Name)**  
**@User.FindFirstValue(ClaimTypes.Email)**

At the end we create a button to submit the form.

```
<button type="submit" asp-route-itemId="@Model.Item.ItemId" class="btn btn-primary mt-3">Create Booking</button>
```

```
<a asp-controller="Item"
```

```
asp-action="Grid" class="btn btn-warning mt-3">Cancel</a>
```

- **Delete.cshtml**

Within this razor view file, we used two headings in html, one for the title and the second for confirming the delete action. We also have a div element which displays booking details for example represent definition for terms, description and list at last we create <form> for submit deletion request and button to proceed it.

#### - Receipt.cshtml

this code generates an HTML page for displaying a booking receipt We create <html>, <head>, <body> These tags define the basic structure of the page. Two div elements that displays the booking table message and the booking confirmation message. They check if the respective messages are not empty and, if so, display them using @Html.Raw to render HTML markup within the message.

#### - Table.cshtml

<partial name="\_BookingTable" model="Model" /> We create this line to includes the content of the "\_BookingTable" partial view in this view.

### 2.2.2 Customer

#### - AllCustomersTable.cshtml

Includes code of a Razor view which deals with rendering or calling a list of customers and providing an option to delete all customers we specifies the model for the view is of type IEnumerable<Customer> in the top and after that we call the partial view "\_CustomerTable" by this line <partial name="\_CustomerTable" model="Model" /> because it is responsible for displaying the list of customers in a tabular format. Ends the codes with form tag to Send the HTTP Get request and button to submit.

#### - DeleteAllCustomersConfirmation.cshtml

this view is to confirm the deletion of all customers. It displays a confirmation message and allows the user to proceed with the deletion or cancel the operation. In this codes lines we create div which includes alert box to confirm the deletion action as well as we Html form which used to send the HTTP post request. And button to allow the user to confirm the deletion

```
<form asp-controller="Customer" asp-action="DeleteAllCustomersConfirmed"
method="post">
```

```
<button type="submit" class="btn btn-outline-danger">
```

#### - DeleteCustomer.cshtml.

In this razor view we displays a confirmation message and the details for just one customer to be deleted, allowing the user to proceed with the deletion or cancel the operation. The actual

deletion logic is typically implemented in the "DeleteCustomerConfirmed" action in the associated controller.

- **Table.cshtml.**

`<partial name="_CustomerTable" model="Model" />` in this razor view we use this line to rendering a Razor partial view `"_CustomerTable"` and the model `"Model"`.

### 2.2.3 Home

-**Index.cshtml**

Here we combine various sections of the home web page, including details about the HRS, our team members, and contact information. It also incorporates images, links, and an embedded map to provide a comprehensive view of the company's services and how to get in touch with us. Three sections are included:

**1-About section** contain information about the renting company as general and its includes image and description on the rithg and of course the image src is set to "House1.jpg," and the description provides details about the company's responsibilities, such as lease agreements, rent collection, and property maintenance.

**2- Team section** with id = team introduces the team members.

It displays team members' names, images, and email addresses.

**3- Contact section** with the `id="contact," provides contact information for the company.

It includes the company's address, email, phone number, and opening hours.

### 2.2.4 Item

- **Create.cshtml**

In this razor view provide html from to create new house item with help of asp-action set to 'create' we can specify where the data should submit. In this form we create a number of input field for example "Category," "Location," "Rooms," "Area," "Renting," "Description," and "Image URL." This field allows the user to insert information about the house they wont to create. The form includes validation for each input field **asp-validation-for** if there are validation errors, we have error messages that will be displayed.

- **Delete.cshtml**

This razor view code represents a confirmation page for deleting a "House" item and we used two html heading tags for the titles followed by details display of the house such as Category, Rooms, Area and Renting.

`<dl class="row">`

`<dt class="col-sm-2">Category</dt>`

```
<dd class="col-sm-10">@Model.Category</dd>
```

```
</dl>
```

Additionally, we create form element for confirm the deletion and two button to delete and cancel.

#### - Details.cshtml

The code starts with a Bootstrap carousel to display images of the house.

```
<div id="carouselExampleIndicators" class="carousel slide">
```

after that we displays the house information in this class as table:

```
class="portfolio-info">
```

```
<h3>Item @Model.ItemId</h3>
```

```
<ul>
```

```
<li><strong>Category</strong>: @Model.Category</li>
```

```
<li><strong>Renting</strong>: @Model.Renting.ToString("0.00 NOK")</li>
```

```
<li><strong>House Location</strong>: @Model.Location</li>
```

```
</ul>
```

Beside the navigation links we insert that only the admin user some can delete or update in the houses list. Another important fetcher is that booking in known period and provide an error message if this period not valid or shown Booked sign if its valid.

#### - Grid.cshtml

Here we display alist of houses in table or graid format

```
<h1>List of House Items (@Model.CurrentViewName View)</h1>
```

```
<a href="/Item/Table">Table View</a>
```

We uses a partial view "\_ItemCard" to recall a card displaying the house item's details. This is done with `<partial name="_ItemCard" model="item" />`.

At the end we create new house button `<a class="btn btn-outline-primary" asp-controller="Item" asp-action="Create">New House</a>`.

#### - HouseSerach.cshtml

Here we create a fetcher of displaying a list of house as result of search, we used grid format to display the images of the houses and there categories



#### - Table.cshtml

Here we display the items (houses) in table format and for provide dynamic display we write this line **@Model.CurrentViewName**. Additionally to recall the partial view we 'ItemTable'

```
<partial name="_ItemTable" model="Model.Items" />
```

#### - update.cshtml

Razer view used for updating the information for house item

```
<form asp-action="Update" class="col-md-7 col-lg-5 offset-md-3 offset-lg-4">
```

We defined for different properties of the house item such as Category, Location, Rooms, Area, Renting, Description, and Image Url).

We provide for each form a label, an input element, and a span element to display validation errors.

```
<div class="form-group">
```

```
    <label asp-for="Location">Location</label> <span class="text-bg-danger"></span>
```

```
    <input asp-for="Location" class="form-control bg-light" />
```

```
    <span asp-validation-for="Location" class="text-danger"></span>
```

```
</div>
```

### 2.2.5 Shared

#### ❖ Layout

##### - \_BookingTable.cshtml

Here we have the list of booking records, its include the booking object and their details  
Each booking record includes a "Delete this Booking"

##### - \_Carousel.cshtml

Within this lines of codes we create the container for customer list in addition to html table that list customer information .

```
<table class="table table-striped">
```

```
    <tr>
```

```
        <th>Name</th>
```

```
        <th>Email</th>
```

```
        <th>Phone</th>
```

```
        <th>Booking</th>
```

```
        <th>Date</th>
```

```
        <th>Action</th>
```

```
    </tr>
```

##### - \_CustomerTable.cshtml

Contain of Html tags and elements that create for the table and column header in addition to button to delete the booking .

- **\_ItemCard.cshtml**

In this section we designed a card layout for display the house's details including the image, category, renting price and booked label if the item That already reserved.

- **\_ItemTable.cshtml**

In this razor view we Creates a container to hold the content. After the heading tag we sets up an HTML table with Bootstrap classes for styling **<table class="table table-hover">** rest of the code create tables for house information and we finish with the button of create new house

- **\_Layout.cshtml**

This file contain a structure for the website with a navigation bar, a hero section, main content, and a footer. It also includes links to external CSS and JavaScript files to provide styling and interactivity to the site.

- **\_LoginPartial.cshtml**

In this file we create codes for dynamic display for the Hello word ?+ the user name or email and log out button when the user is signed inn

```
<li class="nav-item">
    <a id="manage" class="nav-link text-white active" asp-area="Identity" asp-
page="/Account/Manage/Index" title="Manage">Hello @UserManager.GetUserName(User)!</a>
```

We add another fetcher of "Register" and "Login" links when the user is not authenticated

```
<li class="nav-item">
    <form id="logoutForm" class="form-inline" asp-area="Identity" asp-
page="/Account/Logout" asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })">
        <button id="logout" type="submit" class="nav-link btn btn-link text-white border-
0 active">Logout</button>
    </form>
</li>
}
```

- **\_ValidationScriptsPartial.cshtml**

Javascript code for present client-side allow as to define validation rules for form fields and provides user-friendly error messages

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```

#### - **\_ViewImports.cshtml@using HouseRenting**

```
@using HouseRenting
@using HouseRenting.Models
@using HouseRenting.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Includes several tag and refence.

#### - **\_\_ViewStart.cshtml**

```
@{
    Layout = "_Layout";
}
```

specifies the layout that the current view should use.

## 2.4 ViewModles

### 2.4.1 CreateBookingViewModel.cs

is designed to hold data related to both an Item and a Booking, which can then be passed to a view to facilitate the creation of a booking for a specific item. This helps in organizing and presenting the data for the view. Including Item class which represent an item, and booking represent booking in application

### 2.4.2 ItemListViewModel.cs

Is designed to hold a collection of Item objects along with information about the current view. This ViewModel is used to pass data from a controller to a view when rendering a list of items IEnumerable<Item>. This property is used to store a list of items that will be presented in the view.

## 2.5 Controllers

It consists of five controllers classes

### 2.5.1 BookingController.cs:

it is a class that inherits from controller who mange the booking and used to handle Http request. Its take three parameters ItemBbContext, IBookingRepository and

`ILogger<BookingController>` is provided when an instance of the controller is created. It's controller handles booking, including viewing creating deleting and checking booking availability. It also interacts with the database through `ItemDbContext` and `IBookingRepository`. By logging keep track of events and potential problem.

Action Method Like (Table, AllBookingTable, CreateBookingItem, BookingPeriodOverlaps, BookingPeriods, Delete, DeleteConfirmed) the Action Method are responsible for processing Http request. It's can handle different Http operation, such GET, Post, DELETE, PUT.

Attributes like `[Authorize]` use to restrict access "like Delete and Update in our project" to certain action to users. `[Authorize (Role = "Admin")]` Restricts access to the AllBookingsTable action to users ]

We used a logger (`_logger`) to Log errors, warning and viewing.

### **2.5.2 CustomerControllers**

### **2.5.3 HomeController-cs**

It defines MVC controller ( `HomeController`) with a single method (Index) that handle request for the index page. When a user access this page "Index" returns a result of "ViewResult" by calling view. View will be display in the browser to user.

### **2.5.4 ItemController:**

This controller manages CRUD (Create, Read, Update, Delete) The operation for item it connected with an item repository "`ItemRepository`" to access and manipulate item data and uses View to present data. `ItemController` takes two parameters: "`ItemRepository`" and "`ILogger<ItemController>`" It include Action Method like Table that retrieves a list of items from the `_ItemRepository` using "GetAll" if item is found then constructs a view model "`itemListViewModel`" and return view with the list. Details, Grid, Create (Get,Post), Update Delete (GET, Post) also Action method.

### **2.5.5 ItemDBmodelsnapshot.**

This class represented a database model snapshot and is generated by Entity Framework Core when we create/apply migration. Entity Framework Core uses this snapshot to generate SQL scripts that create or update the database schema to match your data model.

Every method has functionality in this class for example The `BuildModel` method is an overridden method from the `ModelSnapshot`, It's used to define the database model as it should be. It configurations for various entity types in our project there are configurations for entities like Booking," "Customer," "Image," and "Item." These configurations define properties like data types, constraints (e.x, required fields), and relationships between entities (e.x, foreign keys).

## 2.4 Data access layer

It consists of DbContext, DbInit, etc

## 2.5. DAL folder

### 2.5.1 BookingRepository.cs

### 2.5.2 CustomerRepository.cs

### 2.5.4 DBInit.cs

### 2.5.5 IbookingRepository.cs

### 2.5.6 IcustomerRepository.cs

### 2.5.7 IitemRepository.cs

### 2.5.8 ItemDbContext

## 2-6 M -igrations folder

This migration file database tables required for identity management in our application. It's a crucial step in enabling user registration, authentication, and authorization within the application.

- 20231022020619\_IdentityAdded
- 20231023011942\_InitDb

## 2.7 LOGs folder

these logs provide information about the application's startup, listening endpoints, environment, and details about incoming HTTP requests.

- app\_20231028\_214840.log
- app\_20231028\_215426.log
- app\_20231028\_215430.log

## 3 Database

We utilize four tables in our system: "Item," "Customer," "Booking", and "Images."

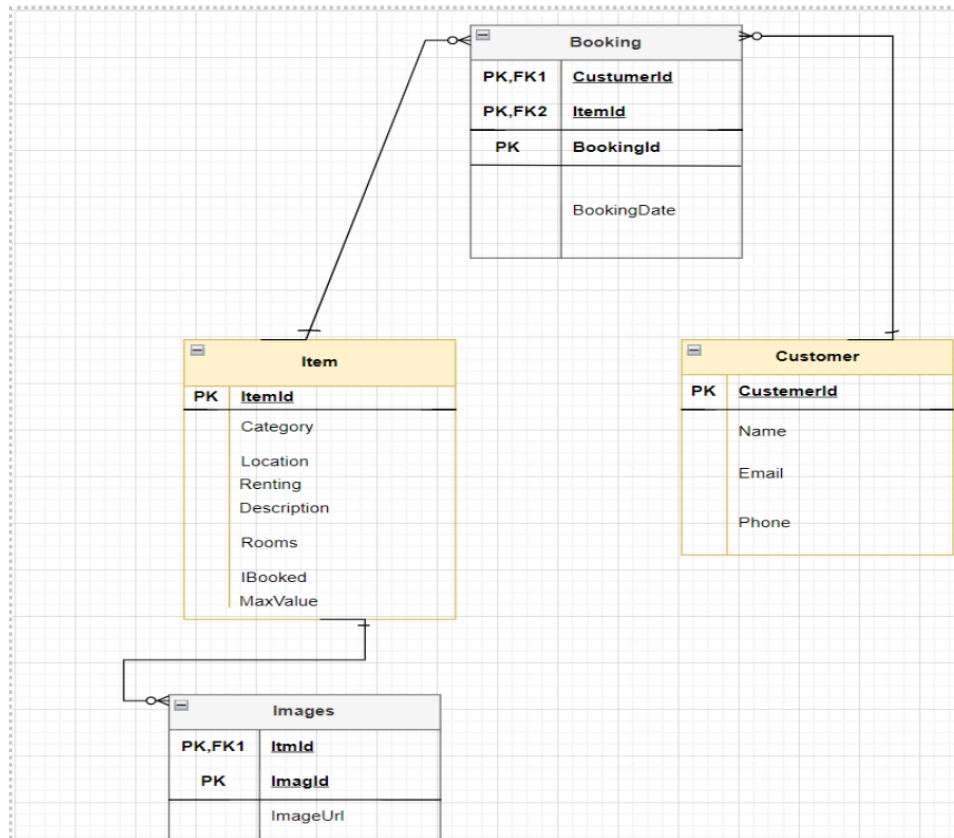
"Customer" and "Item" represent entities with a many-to-many relationship, and to manage this relationship, we introduce an intermediary table called "Booking."

Furthermore, "Image" is another entity that is related to the "Item" table, allowing each item to be associated with multiple images.

An "Item" can be linked to zero or more bookings.

A "Customer" has the potential to have multiple bookings.

An "Item" can be linked to zero or more images.



Etc.

## 4 Conclusion

By combining the .NET 6.0 and the Model-View-Controller (MVC) framework, we've developed a web application that redefines the house rental System project which provides us with many different information from the basics details of .Net core till advanced programming language and frame work . in addition, we learn how to use the data and connect with project and how used the crud technique for the web side and

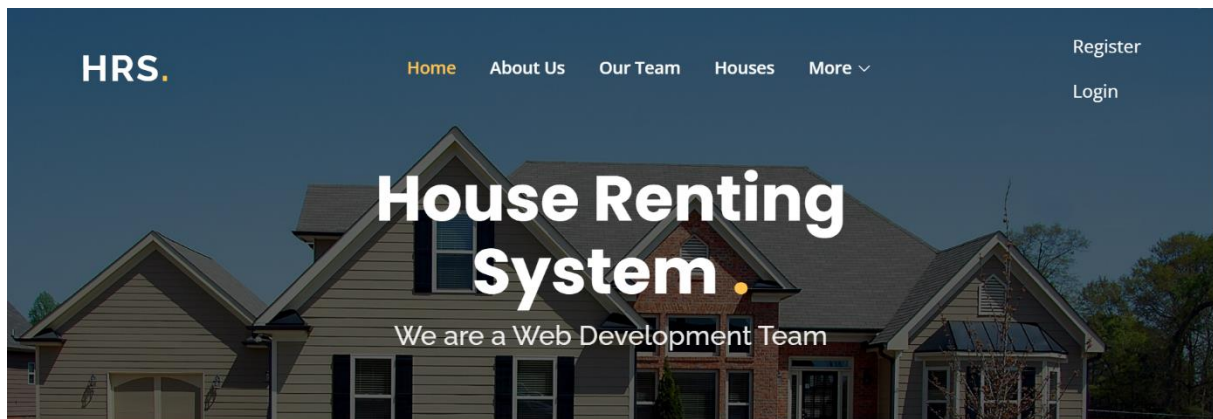
## 5 Contribution List

Student 1 & student 3 back-end codes.

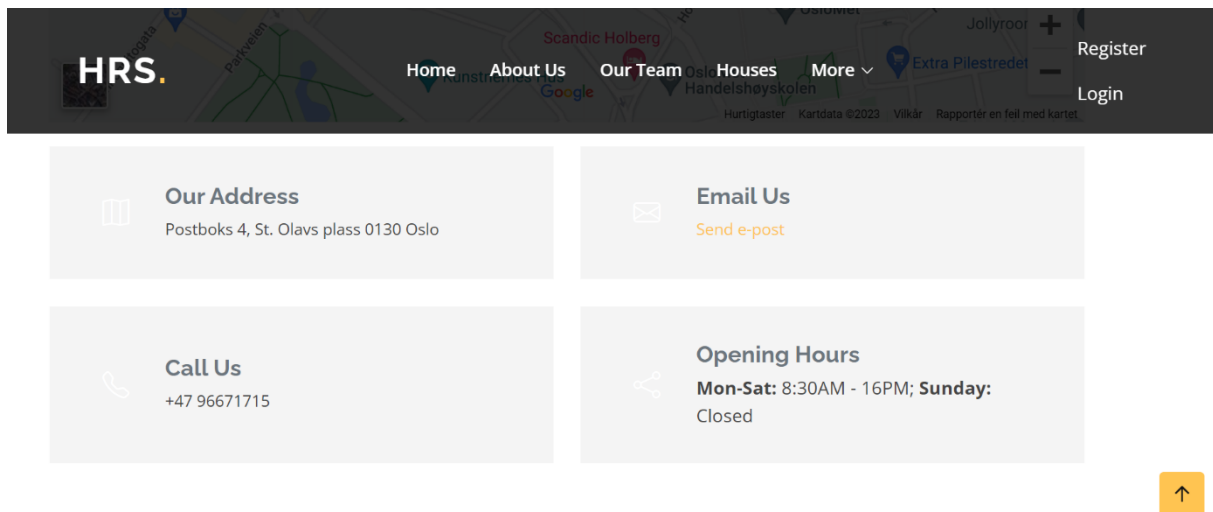
Student 2 & Student4 did design and front-end.

Student 1&2&3&4 did documentation.

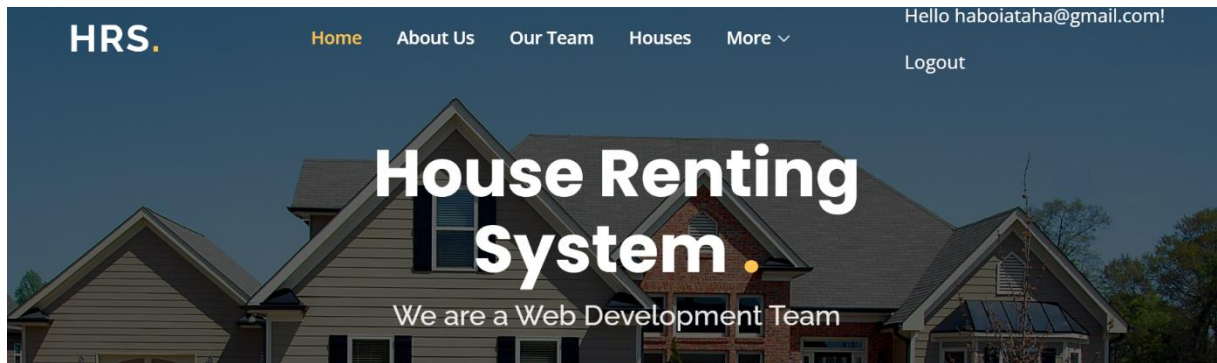
## 6.Design



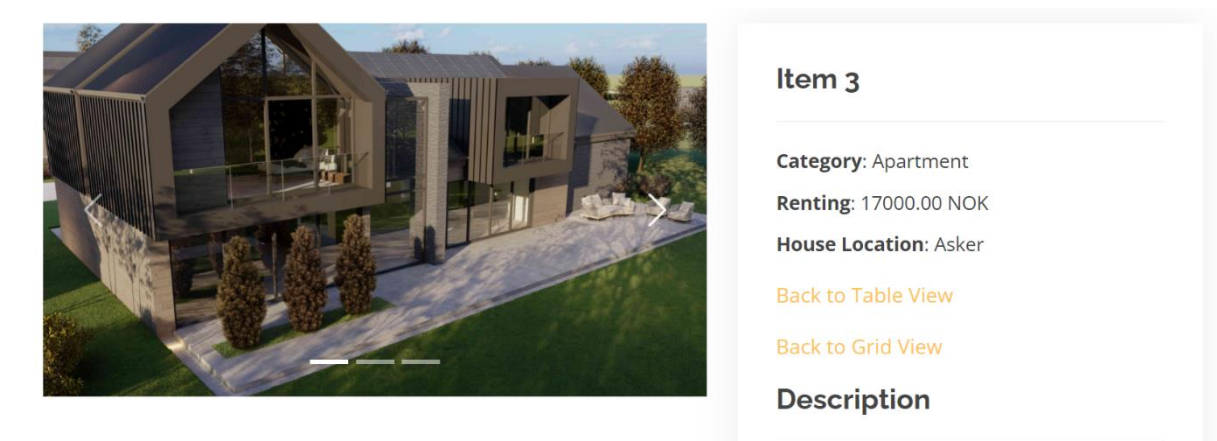
Home page



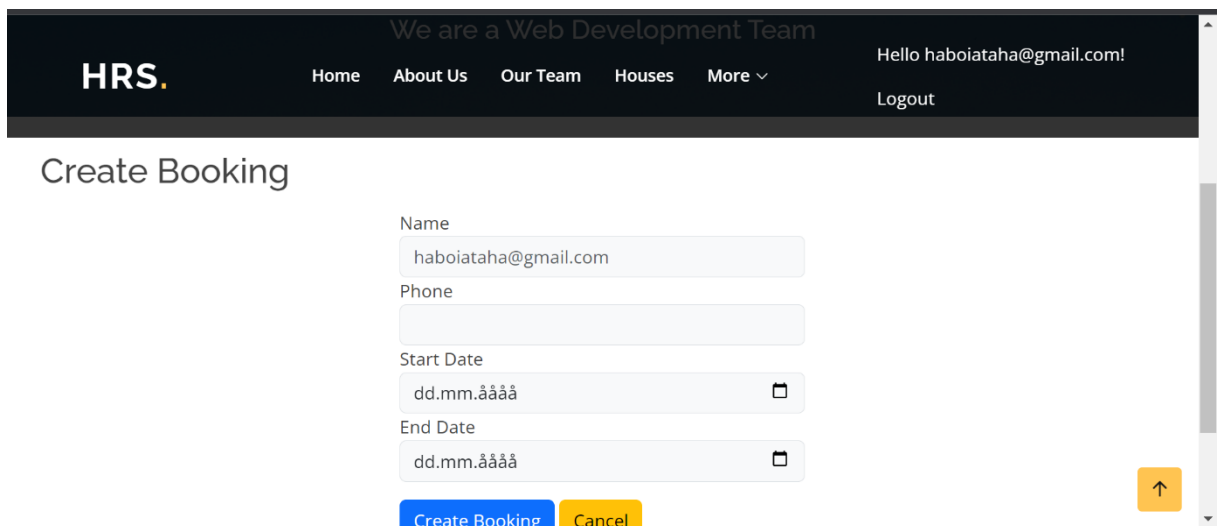
Footer



Hello + username or email



No Delete or update button for vanlig users.



If the user inserts valid date a Booked sing will appear in particular item if not he will get an error message.



På localhost:7274 står det

Booking Confirmation:

We confirm that the house number 3 are booked to Hiba in the date:2023-10-31 in the period[31/10/2023 ,10/11/2023]!

OK




**HRS**

HomeAbout UsOur TeamHousesMore ▾

Hello admin@hrs.com!  
Logout

List of House Items (Table View)

Grid View

Id	Category	Rooms	Area	Renting	Images	Actions
1	Semi-detached house	8	150	19500.00 NOK		<button>Update</button> <button>Delete</button>
2	Apartment	6	160	17500.00 NOK		<button>Update</button> <button>Delete</button>
3	Apartment	6	160	17000.00 NOK		<button>Update</button> <button>Delete</button>

↑

Admin user have er facilities such as authority to update and delete within the house list

## Reference List

Bootstrap design

<https://bootstrapmade.com/license/>

Emoji item

<https://anyconv.com/no/konvertere-image-til-jpg/>

<https://oslomet.instructure.com/courses/26679/modules>