**ONLINE RAILWAY RESERVATION SYSTEM**

**Synopsis:**

This system is designed for the online reservation of railway tickets for travel between any destinations. This system enables the Advance booking in any class, against general and ladies quota, on payment of fare in full for adults and children, a maximum of six berths/seats at a time, for journey between any two stations served by a train.

It also provides details about

1. Timetable

2. Train fares

3. Current status of reservation position  
4.Train available between a pair of stations  
5.Accommodation available for a train/date combination

The user of this system should first register for any interaction with the system. Once registered, he/she will be provided with a username and password for the user to log in. After logging in the user should select the kind of activity, he would like to perform like booking a ticket, canceling a ticket, look out for help and so on. A person can check the train timings, train fares and other trains information without login.

**Requirements:**

***RESERVATION***

The customer can reserve ticket by filling the reservation form present in the website. The reservation form present contain passenger name, gender, age, address, credit no, bank name, class through which the passenger is to travel etc. The online manager will verify the detail and provide PNR number to the customers who reserve the ticket.

***CANCELLATION***

The user can also perform cancellation of the ticket which he\she had reserved earlier by entering PNR no. This PNR no will be checked with the PNR no in the database. If it exists, then it will be cancelled. After cancellation process the conformation message will be send by the server.



**Functional Requirements Specification**

This section outlines the use cases for each of the actors separately.

**Passenger**

Use cases: reserve ticket, cancel ticket, view ticket fare, view train details.



If passenger needed reserves his ticket before and after reserving if he wants to cancel his ticket, then he can move to cancel ticket option and amount of ticket is display along with the train details….

1: passenger first checks for the train details.

2: passenger by viewing ticket amount reserves his ticket

3: if he wants to cancel his ticket, he does it

**Administrator**

Usecase: update train details



Administrator simply updates all the information such as the train arrival and departure details, ticket fare amounted…

1: first administrator updates all details.

2: passenger goes through then and acts accordingly.

## User Characteristics

A passenger can reserve ticket, view ticket fare and view train details by searching from the database. A user can also cancel ticket which is reserved by him. And the administrator simply updates all the information such as the train arrival and departure details, ticket fare amounted and the tickets which are bought and then blocks the seats which are reserved.

The railway database is used to know the total details and information about the train and the tickets reserved by the passengers. So, if the user wants to reserve the train ticket then he should first fill the form and select the dates and timing of the train along with the place. So, if the seats are available then the administrator will issue ticket and the bill should be paid by using credit/debit card. So, the passenger can also cancel the reserved ticket whenever he wants.

**Class diagram for Online Railway Ticket Reservation**

**Sequence diagram for Online Railway Ticket Reservation**



Creating a simplified version of an Online Railway Reservation System involves several components, including models, controllers, and views. Below is an outline of how you can implement this system using ASP.NET MVC. For brevity, I'll focus on the core functionality: user registration, ticket reservation, and cancellation.

1. **Step 1: Set Up the Project**
2. Create a new ASP.NET MVC project in Visual Studio.
3. Add necessary packages (like Entity Framework for database interactions).
4. **Step 2: Define Models**

csharp

Copy code

// Models/User.cs

public class User

{

public int Id { get; set; }

public string Username { get; set; }

public string Password { get; set; }

public string UserType { get; set; } // Passenger or Admin

}

// Models/Train.cs

public class Train

{

public int Id { get; set; }

public string TrainNumber { get; set; }

public string DepartureTime { get; set; }

public string ArrivalTime { get; set; }

public string Source { get; set; }

public string Destination { get; set; }

public decimal Fare { get; set; }

}

// Models/Reservation.cs

public class Reservation

{

public int Id { get; set; }

public string PNR { get; set; }

public int UserId { get; set; }

public int TrainId { get; set; }

public string PassengerName { get; set; }

public string Gender { get; set; }

public int Age { get; set; }

}

1. **Step 3: Create the Database Context**

csharp

Copy code

// Data/ApplicationDbContext.cs

public class ApplicationDbContext : DbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Train> Trains { get; set; }

public DbSet<Reservation> Reservations { get; set; }

// Configure the connection string in Startup.cs

}

1. **Step 4: Create Controllers**
2. **UserController**

csharp

Copy code

// Controllers/UserController.cs

public class UserController : Controller

{

private readonly ApplicationDbContext \_context;

public UserController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public IActionResult Register(User user)

{

if (ModelState.IsValid)

{

\_context.Users.Add(user);

\_context.SaveChanges();

return RedirectToAction("Login");

}

return View(user);

}

[HttpGet]

public IActionResult Login()

{

return View();

}

[HttpPost]

public IActionResult Login(string username, string password)

{

var user = \_context.Users.SingleOrDefault(u => u.Username == username && u.Password == password);

if (user != null)

{

// Set authentication cookie (not shown here)

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError("", "Invalid username or password");

return View();

}

}

1. **ReservationController**

csharp

Copy code

// Controllers/ReservationController.cs

public class ReservationController : Controller

{

private readonly ApplicationDbContext \_context;

public ReservationController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Reserve()

{

var trains = \_context.Trains.ToList();

return View(trains);

}

[HttpPost]

public IActionResult Reserve(Reservation reservation)

{

if (ModelState.IsValid)

{

reservation.PNR = Guid.NewGuid().ToString(); // Generate PNR

\_context.Reservations.Add(reservation);

\_context.SaveChanges();

return RedirectToAction("Confirmation", new { pnr = reservation.PNR });

}

return View(reservation);

}

[HttpGet]

public IActionResult Cancel()

{

return View();

}

[HttpPost]

public IActionResult Cancel(string pnr)

{

var reservation = \_context.Reservations.SingleOrDefault(r => r.PNR == pnr);

if (reservation != null)

{

\_context.Reservations.Remove(reservation);

\_context.SaveChanges();

ViewBag.Message = "Reservation canceled successfully.";

}

else

{

ViewBag.Message = "Invalid PNR.";

}

return View();

}

}

1. **Step 5: Create Views**

**Register View**

html

Copy code

@model YourNamespace.Models.User

<form asp-action="Register" method="post">

<input asp-for="Username" placeholder="Username" required />

<input asp-for="Password" type="password" placeholder="Password" required />

<button type="submit">Register</button>

</form>

**Login View**

html

Copy code

<form asp-action="Login" method="post">

<input name="username" placeholder="Username" required />

<input name="password" type="password" placeholder="Password" required />

<button type="submit">Login</button>

</form>

**Reserve View**

html

Copy code

@model IEnumerable<YourNamespace.Models.Train>

<form asp-action="Reserve" method="post">

<select name="TrainId" required>

@foreach (var train in Model)

{

<option value="@train.Id">@train.TrainNumber - @train.Source to @train.Destination</option>

}

</select>

<input name="PassengerName" placeholder="Passenger Name" required />

<input name="Gender" placeholder="Gender" required />

<input name="Age" type="number" placeholder="Age" required />

<button type="submit">Reserve</button>

</form>

**Cancel View**

html

Copy code

<form asp-action="Cancel" method="post">

<input name="pnr" placeholder="Enter PNR to cancel" required />

<button type="submit">Cancel</button>

</form>

@if (ViewBag.Message != null)

{

<p>@ViewBag.Message</p>

}

1. **Step 6: Configure Routing and Database**
2. Set up the routing in Startup.cs for MVC.
3. Configure your database connection string in appsettings.json.
4. Run migrations to create the database.
5. **Conclusion**

This implementation covers the essential functionalities of an Online Railway Reservation System, including user registration, login, ticket reservation, and cancellation. You can extend it further with features like email notifications, payment processing, and more advanced user management.

4o mini

……………………code for first…………………..

1. **Step 1: Set Up the Database**
2. **Create the Database: Create a database for the system (e.g., RailwayReservationDB).**
3. **Define Tables: You need to create tables for Users, Trains, Reservations, etc. Below are sample SQL statements to create these tables.**

**sql**

**Copy code**

**CREATE TABLE Users (**

**UserId INT PRIMARY KEY IDENTITY,**

**Username NVARCHAR(50) NOT NULL UNIQUE,**

**PasswordHash NVARCHAR(255) NOT NULL,**

**Email NVARCHAR(100),**

**CreatedAt DATETIME DEFAULT GETDATE()**

**);**

**CREATE TABLE Trains (**

**TrainId INT PRIMARY KEY IDENTITY,**

**TrainNumber NVARCHAR(20) NOT NULL,**

**DepartureTime DATETIME,**

**ArrivalTime DATETIME,**

**Source NVARCHAR(100),**

**Destination NVARCHAR(100),**

**Fare DECIMAL(10, 2),**

**TotalSeats INT,**

**AvailableSeats INT**

**);**

**CREATE TABLE Reservations (**

**ReservationId INT PRIMARY KEY IDENTITY,**

**UserId INT FOREIGN KEY REFERENCES Users(UserId),**

**TrainId INT FOREIGN KEY REFERENCES Trains(TrainId),**

**PassengerName NVARCHAR(100),**

**Gender NVARCHAR(10),**

**Age INT,**

**Address NVARCHAR(255),**

**CreditCardNumber NVARCHAR(20),**

**BankName NVARCHAR(100),**

**TravelClass NVARCHAR(50),**

**PNR NVARCHAR(20) NOT NULL UNIQUE,**

**CreatedAt DATETIME DEFAULT GETDATE()**

**);**

1. **Step 2: Create Models**

**In your ASP.NET MVC project, create models for User, Train, and Reservation.**

**csharp**

**Copy code**

**// Models/User.cs**

**public class User**

**{**

**public int UserId { get; set; }**

**public string Username { get; set; }**

**public string PasswordHash { get; set; }**

**public string Email { get; set; }**

**public DateTime CreatedAt { get; set; }**

**}**

**// Models/Train.cs**

**public class Train**

**{**

**public int TrainId { get; set; }**

**public string TrainNumber { get; set; }**

**public DateTime DepartureTime { get; set; }**

**public DateTime ArrivalTime { get; set; }**

**public string Source { get; set; }**

**public string Destination { get; set; }**

**public decimal Fare { get; set; }**

**public int TotalSeats { get; set; }**

**public int AvailableSeats { get; set; }**

**}**

**// Models/Reservation.cs**

**public class Reservation**

**{**

**public int ReservationId { get; set; }**

**public int UserId { get; set; }**

**public int TrainId { get; set; }**

**public string PassengerName { get; set; }**

**public string Gender { get; set; }**

**public int Age { get; set; }**

**public string Address { get; set; }**

**public string CreditCardNumber { get; set; }**

**public string BankName { get; set; }**

**public string TravelClass { get; set; }**

**public string PNR { get; set; }**

**public DateTime CreatedAt { get; set; }**

**}**

1. **Step 3: Create the Database Context**

**Create a database context class to manage the entity models.**

**csharp**

**Copy code**

**// Data/RailwayDbContext.cs**

**using Microsoft.EntityFrameworkCore;**

**public class RailwayDbContext : DbContext**

**{**

**public RailwayDbContext(DbContextOptions<RailwayDbContext> options)**

**: base(options) { }**

**public DbSet<User> Users { get; set; }**

**public DbSet<Train> Trains { get; set; }**

**public DbSet<Reservation> Reservations { get; set; }**

**}**

1. **Step 4: Configure the Application**

**Update Startup.cs to include Entity Framework and your database context.**

**csharp**

**Copy code**

**public void ConfigureServices(IServiceCollection services)**

**{**

**services.AddDbContext<RailwayDbContext>(options =>**

**options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));**

**services.AddControllersWithViews();**

**}**

1. **Step 5: Create the Registration Controller**

**Create a controller to handle user registration and login.**

**csharp**

**Copy code**

**// Controllers/UserController.cs**

**using Microsoft.AspNetCore.Mvc;**

**using System.Security.Cryptography;**

**using System.Text;**

**public class UserController : Controller**

**{**

**private readonly RailwayDbContext \_context;**

**public UserController(RailwayDbContext context)**

**{**

**\_context = context;**

**}**

**[HttpGet]**

**public IActionResult Register() => View();**

**[HttpPost]**

**public IActionResult Register(User user)**

**{**

**if (ModelState.IsValid)**

**{**

**user.PasswordHash = HashPassword(user.PasswordHash); // Hashing password for security**

**\_context.Users.Add(user);**

**\_context.SaveChanges();**

**return RedirectToAction("Login");**

**}**

**return View(user);**

**}**

**[HttpGet]**

**public IActionResult Login() => View();**

**[HttpPost]**

**public IActionResult Login(string username, string password)**

**{**

**var user = \_context.Users.FirstOrDefault(u => u.Username == username);**

**if (user != null && VerifyPassword(password, user.PasswordHash))**

**{**

**// Set session or authentication cookie**

**return RedirectToAction("Index", "Home");**

**}**

**ModelState.AddModelError("", "Invalid login attempt.");**

**return View();**

**}**

**private string HashPassword(string password)**

**{**

**using (var sha256 = SHA256.Create())**

**{**

**var bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));**

**return Convert.ToBase64String(bytes);**

**}**

**}**

**private bool VerifyPassword(string password, string hashedPassword)**

**{**

**var hashedInput = HashPassword(password);**

**return hashedInput == hashedPassword;**

**}**

**}**

1. **Step 6: Create Views for Registration and Login**

**Register.cshtml:**

**html**

**Copy code**

**@model YourNamespace.Models.User**

**<h2>Register</h2>**

**<form asp-action="Register" method="post">**

**<input name="Username" placeholder="Username" required />**

**<input name="PasswordHash" type="password" placeholder="Password" required />**

**<input name="Email" placeholder="Email" required />**

**<button type="submit">Register</button>**

**</form>**

**Login.cshtml:**

**html**

**Copy code**

**<h2>Login</h2>**

**<form asp-action="Login" method="post">**

**<input name="username" placeholder="Username" required />**

**<input name="password" type="password" placeholder="Password" required />**

**<button type="submit">Login</button>**

**</form>**

1. **Step 7: Set Up Navigation and Authentication**

**Make sure your views include navigation links to the registration and login pages. Set up session or cookie-based authentication as needed.**

1. **Conclusion**

**With this setup, you have established the foundational process of the Online Railway Reservation System, including user registration and login functionality. From here, you can continue to implement features like ticket reservation, cancellation, and train management, gradually expanding the application’s capabilities.**

**4o mini**

**Top of Form**

**Bottom of Form**

**Ch**