### Booking-clone.com API Documentation

### 1- Create a new .Net Core Web API project

2- Install the following packgaes

<PackageReference Include="AutoMapper" Version="12.0.1" />

<PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.1" />

<PackageReference Include="BCrypt.Net-Next" Version="4.0.3" />

<PackageReference Include="MailKit" Version="4.4.0" />

<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="7.0.14" />

<PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="7.0.14" />

<PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.0" />

<PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.14" />

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.14" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="7.0.14">

<PrivateAssets>all</PrivateAssets>

<IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

</PackageReference>

<PackageReference Include="Microsoft.IdentityModel.Tokens" Version="7.0.3" />

<PackageReference Include="MimeKit" Version="4.4.0" />

<PackageReference Include="Serilog" Version="3.1.1" />

<PackageReference Include="Serilog.AspNetCore" Version="8.0.0" />

<PackageReference Include="Serilog.Sinks.Console" Version="5.0.1" />

<PackageReference Include="Serilog.Sinks.File" Version="5.0.0" />

<PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />

<PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="7.0.3" />

3- Create new folder named Data and add new ApplicationDbContext.cs file

using Booking.Com\_Clone\_API.Models.Domain;

using Microsoft.EntityFrameworkCore;

using NZWalks.API.Models.Domain;

namespace NZWalks.API.Data

{

public class NZWalksDbContext:DbContext

{

public NZWalksDbContext(DbContextOptions<NZWalksDbContext> dbContextOptions): base(dbContextOptions)

{

}

public DbSet<Region> Regions { get; set; }

public DbSet<Image> Images { get; set; }

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

// Only for manual entry to database

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

// Seed the data for Region

// Easy Medium, hard

var regions = new List<Region>

{

new Region()

{

Id = Guid.Parse("89c18c38-1825-4373-9cc2-8f87056db7f0"),

Name = "Auckland",

Code="AKL",

RegionImageUrl="Ak.jpg"

},

new Region()

{

Id = Guid.Parse("1526dc5d-9bdd-4ac7-aad4-20e49b9eccef"),

Name = "Wellington",

Code = "WL",

RegionImageUrl = "WL.jpg"

}

};

// Seed difficulties to database

modelBuilder.Entity<Region>().HasData(regions);

}

}

}

4- Create new folder named Models and create new folder Domain folder and Add the following class named as User.cs

using System.ComponentModel.DataAnnotations;

namespace Booking.Com\_Clone\_API.Models.Domain

{

public class User

{

[Key]

public int Id { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string ConfirmPassword { get; set; }

public string VerificationToken { get; set; }

public bool IsVerified { get; set; }

public string ResetPasswordToken { get; set; }

public DateTime ResetPasswordTokenExpiresAt { get; set; }

}

}

5- Now add database connection string name in appsettings.json file refere the highlighted lines

{

"ConnectionStrings": {

"BookingConnectionString": "Server=AZKBAN\\MSSQLSERVER01;Database=Booking\_Clone;Trusted\_Connection=True;TrustServerCertificate=True"

}

}

5- Now Configure the ApplicationDbContext connection string in Program.cs file refer the highlighted lines

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using NZWalks.API.Data;

using NZWalks.API.Mappings;

using NZWalks.API.Middlewares;

using NZWalks.API.Repositories;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add DbContext class to connect with database and reate connectin string

builder.Services.AddDbContext<NZWalksDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("BookingConnectionString")));

5- Now initiate the migration by running the following command

add-migration “Initiated migration”

After successfully running the above command the following command update-database

6-Now add new folder inside Models folder named as DTOs and create new file inside named as UserDto.cs

using System.ComponentModel.DataAnnotations;

namespace Booking.Com\_Clone\_API.Models.DTO

{

public class UserDto

{

public string Email { get; set; }

public string Password { get; set; }

public string ConfirmPassword { get; set; }

public string VerificationToken { get; set; }

public bool IsVerified { get; set; }

public string ResetPasswordToken { get; set; }

public DateTime ResetPasswordTokenExpiresAt { get; set; }

}

}

7-Now Create new folder named as Mappings and create new file inside AutoMapperProfiles.cs

using AutoMapper;

using Booking.Com\_Clone\_API.Models.Domain;

using Booking.Com\_Clone\_API.Models.DTO;

using NZWalks.API.Models.Domain;

using NZWalks.API.Models.DTO;

namespace NZWalks.API.Mappings

{

public class AutoMapperProfiles:Profile

{

// Insatll AutoMapper Nuget named AutoMapper and inject into Program.cs file

public AutoMapperProfiles()

{

CreateMap<UserDto, User>().ReverseMap();

CreateMap<VerificationDto, User>().ReverseMap();

CreateMap<LoginDto, User>().ReverseMap();

}

}

}

8- Now add new folder named as Middleware and create new file inside named as ExceptionHandlerMiddleware

using System.Net;

namespace NZWalks.API.Middlewares

{

public class ExceptionHandlerMiddleware

{

private readonly ILogger<ExceptionHandlerMiddleware> logger;

private readonly RequestDelegate next;

public ExceptionHandlerMiddleware(ILogger<ExceptionHandlerMiddleware> logger,

RequestDelegate next)

{

this.logger = logger;

this.next = next;

}

public async Task InvokeAsync(HttpContext httpContext)

{

try

{

await next(httpContext);

}

catch ( Exception ex)

{

var errorId = Guid.NewGuid();

//Log this Exception

logger.LogError(ex, $"{errorId} : {ex.Message}");

httpContext.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

httpContext.Response.ContentType = "application.json";

var error = new

{

Id = errorId,

ErrorMessage = "Something went wrong! we are looking into resolving this."

};

await httpContext.Response.WriteAsJsonAsync(error);

}

}

}

}

9- Now configure ExceptionHandlerMiddleware inside Program.cs and add Automapper configuration refer highlighted lines

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using NZWalks.API.Data;

using NZWalks.API.Mappings;

using NZWalks.API.Middlewares;

using NZWalks.API.Repositories;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Injectd AutoMapper

builder.Services.AddAutoMapper(typeof(AutoMapperProfiles));

// Global exception handler

app.UseMiddleware<ExceptionHandlerMiddleware>();

10- Now configure the Serilog inside Program.cs file to logs the result refer highlighted lines

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using NZWalks.API.Data;

using NZWalks.API.Mappings;

using NZWalks.API.Middlewares;

using NZWalks.API.Repositories;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Serilog Console to get information on console window

var logger = new LoggerConfiguration()

.WriteTo.Console()

// To get log in text file opion above line sufficient ot handle the log

.WriteTo.File("Logs/Booking\_logs.txt",rollingInterval: RollingInterval.Minute)

.MinimumLevel.Information()

.CreateLogger();

builder.Logging.ClearProviders();

builder.Logging.AddSerilog(logger);

**User Registration Implementation**

11- Now let’s start working on User registration.So now create new folder named Repository and create new file named as IUserRepository.cs refere the highlighted code for creating the user registration

using Booking.Com\_Clone\_API.Models.Domain;

namespace Booking.Com\_Clone\_API.Repositories

{

public interface IUserRepository

{

// Registration

///<summary>

///method to create new registration

/// </summary>

Task<User> CreateUserAsync(User users);

}

}

11- Now create new file inside repository folder named as UserRepository.cs and the following code refer the highlighted lines first configure the Repository.cs class

private readonly IConfiguration \_configuration;

private readonly NZWalksDbContext dbContext;

private readonly ILogger<UserRepository> logger;

public UserRepository(NZWalksDbContext dbContext, ILogger<UserRepository> logger,IConfiguration configuration)

{

this.dbContext = dbContext;

this.logger = logger;

\_configuration = configuration;

}

12- Now add the logic for user registration refer the below code inside UserRepository.cs file

///<summary>

///method to create new registration

/// </summary>

public async Task<User> CreateUserAsync(User user)

{

if (await EmailExistAsync(user.Email))

{

throw new Exception("Email already exists");

}

// Generate verification token

var token = Guid.NewGuid().ToString();

// Hash the password

user.Password = BCrypt.Net.BCrypt.HashPassword(user.Password);

user.ConfirmPassword = BCrypt.Net.BCrypt.HashPassword(user.ConfirmPassword);

user.Password = user.Password;

user.ConfirmPassword = user.ConfirmPassword;

// Set verification token and isVerified status

user.VerificationToken = token;

user.IsVerified = false;

// Add user to database

await dbContext.Users.AddAsync(user);

await dbContext.SaveChangesAsync();

// Send verification email

await SendVerificationEmailAsync(user.Email, token);

logger.LogInformation($"Registration successful. Please verify your email.: {JsonSerializer.Serialize(user)}");

return user;

}

13- Now add logic to check if user already exists, add following code inside UserRepository.cs file along with point number 12

///<summary>

///method to check if user already exists

/// </summary>

public async Task<bool> EmailExistAsync(string email)

{

return await dbContext.Users.AnyAsync(x => x.Email == email);

}

14- Now add method inside IUserRepository.cs file refer the highlighted line

using Booking.Com\_Clone\_API.Models.Domain;

namespace Booking.Com\_Clone\_API.Repositories

{

public interface IUserRepository

{

///<summary>

///method to check if user already exists

/// </summary>

Task<bool> EmailExistAsync(string email);

}

}

15- Now add the repository inside the Program.cs file refer the highlighted code

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using NZWalks.API.Data;

using NZWalks.API.Mappings;

using NZWalks.API.Middlewares;

using NZWalks.API.Repositories;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Email notification

builder.Services.Configure<Booking.Com\_Clone\_API.Models.Domain.EmailConfiguration>(builder.Configuration.GetSection("EmailConfiguration"));

// Injected Repository

// Use nuge package AutoMapper Injection

builder.Services.AddScoped<IUserRepository, UserRepository>();

16- Add new class inside Domain folder inside Models folder and named as SendVerificationMail.cs

namespace Booking.Com\_Clone\_API.Models.Domain

{

public class SendVerificationMail

{

public string ToEmail { get; set; }

public string Subject { get; set; }

public string Body { get; set; }

}

}

16.1 Add new class inside Domain folder inside Models folder named as EmailConfiguration.cs add the following property

namespace Booking.Com\_Clone\_API.Models.Domain

{

public class EmailConfiguration

{

public string SmtpServer { get; set; }

public int Port { get; set; }

public string FromAddress { get; set; }

public string FromName { get; set; }

public string UserName { get; set; }

public string Password { get; set; }

public string VerificationLink { get; set; }

}

}

17- Add new file inside DTOs inside Model folder named as VerificationDto.cs

namespace Booking.Com\_Clone\_API.Models.DTO

{

public class VerificationDto

{

public string Email { get; set; }

public string VerificationToken { get; set; }

}

}

18- Now Create new method inside UserRepository.cs file to send the Verification link along with unique code to verify user email address

///<summary>

///method to send mail for verifying user

/// </summary>

public async Task SendVerificationEmailAsync(string email, string token)

{

var smtpSettings = \_configuration.GetSection("SmtpSettings");

var fromAddress = smtpSettings["FromAddress"];

var fromName = smtpSettings["FromName"];

if (string.IsNullOrEmpty(fromAddress))

{

throw new ArgumentNullException(nameof(fromAddress), "From address is missing or null in SMTP settings.");

}

var emailMessage = new MailMessage

{

From = new MailAddress(fromAddress, fromName),

Subject = "Verify your email",

//https://localhost:7151/api/User/verify-email?email=danishakhtar539%40gmail.com&token=fc60a64e-c2cb-4824-913d-0764040c1c1d

//Body = $"Please click the following link to verify your email: https://localhost:7151/api/User/verify-email?email={email}&token={token}",

IsBodyHtml = true

};

emailMessage.To.Add(email);

// HTML email template

string htmlBody = @"

<!DOCTYPE html>

<html lang=""en"">

<head>

<meta charset=""UTF-8"">

<meta name=""viewport"" content=""width=device-width, initial-scale=1.0"">

<title>Booking.com Clone</title>

</head>

<body>

<div>

<h1 style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px; margin-top:10px; margin-bottom:15px;width:200px;"">Booking-clone.com</h1>

<h2>Verify your email address</h2>

<p>""Your created an account with the email address:" +email+@""".<br/> Click Confirm to verify the email address and unlock your full account.<br/> We'll also import any bookings you've made with that address""</p>

<a href=""https://localhost:7151/api/User/verify-email?email=" + email + "&token=" + token + @""" style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px;"">Confirm</a>

</div>

</body>

</html>";

emailMessage.Body = htmlBody;

using (var client = new SmtpClient(smtpSettings["SmtpServer"], int.Parse(smtpSettings["Port"])))

{

client.UseDefaultCredentials = false;

client.Credentials = new NetworkCredential(smtpSettings["Username"], smtpSettings["Password"]);

client.EnableSsl = true;

try

{

await client.SendMailAsync(emailMessage);

logger.LogInformation($"Verification email sent successfully to {email}.");

}

catch (Exception ex)

{

logger.LogError($"Failed to send verification email to {email}. Error: {ex.Message}");

throw; // Rethrow the exception to propagate it up the call stack

}

}

}

19- Now add method inside IUserRepository.cs file refre highlighted lines

using Booking.Com\_Clone\_API.Models.Domain;

namespace Booking.Com\_Clone\_API.Repositories

{

public interface IUserRepository

{

///<summary>

///method to send mail for verifying user

/// </summary>

Task SendVerificationEmailAsync(string email, string verificationToken);

}

}

20- Now Configure the SMTP server inside appsettings.json refer highlighted lines

{

"SmtpSettings": {

"SmtpServer": "smtp.gmail.com",

"Port": "587",

"Username": "mddanish867@gmail.com",

"Password": "dyptkwuddyrzdvps",

"FromAddress": "bookingclone@booking.com", // Make sure this is properly configured

"FromName": "BookingClone.Com",

"VerificationLink": "https://bookingclone.com/verify"

}

}

21- Now Configure the setting in Program.cs file to send mail refer highlighted lines

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using NZWalks.API.Data;

using NZWalks.API.Mappings;

using NZWalks.API.Middlewares;

using NZWalks.API.Repositories;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Email notification

builder.Services.Configure<Booking.Com\_Clone\_API.Models.Domain.EmailConfiguration>(builder.Configuration.GetSection("EmailConfiguration"));

app.MapControllerRoute(

name: "verify-email",

pattern: "verify-email/{email}/{token}",

defaults: new { controller = "User", action = "VerifyEmailAsync" });

22- Now add new API Controller inside Controller folder named as UserController.cs and add the following code refer the highlighted lines now we are able to create user and send mail to the registered mail

using AutoMapper;

using Booking.Com\_Clone\_API.Models.Domain;

using Booking.Com\_Clone\_API.Models.DTO;

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using NZWalks.API.Controllers;

using NZWalks.API.CustomeActionFilter;

using NZWalks.API.Data;

using NZWalks.API.Repositories;

using System.Data;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using System.Text.Json;

namespace Booking.Com\_Clone\_API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class UserController : ControllerBase

{

private readonly NZWalksDbContext dbContext;

private readonly IUserRepository userRepository;

private readonly IMapper mapper;

private readonly ILogger<RegionsController> logger;

private readonly IConfiguration configuration;

public UserController(NZWalksDbContext dbContext,

IUserRepository userRepository,

IMapper mapper,

ILogger<RegionsController> logger,

IConfiguration configuration)

{

this.dbContext = dbContext;

this.userRepository = userRepository;

this.mapper = mapper;

this.logger = logger;

this.configuration = configuration;

}

///<summary>

///method to create new registration

/// </summary>

[HttpPost]

[ValidateModel]

//[Authorize(Roles = "Writer")]

public async Task<IActionResult> CreateUser([FromBody] UserDto userDto)

{

try

{

// Using Automapper makes much clear code as compare to above method

var regionDomainModel = mapper.Map<User>(userDto);

regionDomainModel = await userRepository.CreateUserAsync(regionDomainModel);

logger.LogInformation($"Finished User created request data: {JsonSerializer.Serialize(regionDomainModel)}");

// Using Automapper makes much clear code as compare to above method

var regionDto = mapper.Map<UserDto>(regionDomainModel);

return Ok(regionDto);

}

catch (Exception ex)

{

logger.LogError(ex, ex.Message);

throw;

}

}

**User Verifications on Clicking the Link**

23- User is successfully created and email sent to the registered mail now once user click on the link we should verify the mail for that we implement following logic Create new method inside IUserRepository.cs refer the higlighted lines

using Booking.Com\_Clone\_API.Models.Domain;

namespace Booking.Com\_Clone\_API.Repositories

{

public interface IUserRepository

{

///<summary>

///method to verify mail

/// </summary>

Task<bool> VerifyEmailAsync(string email, string token);

}

}

24- Now create new method inside UserRepository.cs file refer the following lines of code

///<summary>

///method to verify mail

/// </summary>

public async Task<bool> VerifyEmailAsync(string email, string token)

{

var user = await dbContext.Users.FirstOrDefaultAsync(x => x.Email == email);

if (user == null)

{

return false;

}

// Compare the verification token with the token stored in the user object

if (user.VerificationToken == token)

{

// Mark the email as verified

user.IsVerified = true;

//user.VerificationToken = null; // Remove the token after verification

await dbContext.SaveChangesAsync();

return true;

}

return false;

}

25- Now create new method inside UserController.cs to verify the user. So Registration is successfully created

///<summary>

///method to verify mail

/// </summary>

[HttpGet("verify-email")]

public async Task<IActionResult> VerifyEmailAsync(string email, string token)

{

if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(token))

return BadRequest("Email and token are required");

var isVerified = await userRepository.VerifyEmailAsync(email, token);

if (isVerified)

{

logger.LogInformation($"Verification Successfull: {JsonSerializer.Serialize("Email verified successfully")}");

return Ok("Email verified successfully");

}

else

{

logger.LogInformation($"Verification failed: {JsonSerializer.Serialize("Invalid email or token")}");

return BadRequest("Invalid email or token");

}

}

**Login Implementation Through JWT**

24- Now lets work on login functionality and generate JWT token. So lets first completed the configuration to generate the JWT token. Add the following lines of code inside appsettings.json refer the highlighted lines

{

"Jwt": {

"SecretKey": "bbuosyJSSGPOosflUSJ75JJHst6yjjjST5rt655Y77uhSYSko098HHhgst",

"Issuer": "https://localhost:7151",

"Audience": "https://localhost:7151"

},

}

26- Now lets configure inside Program.cs file refer the highlighted lines

//============================= Start Jwt Authentication =================

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(builder.Configuration["Jwt:SecretKey"]))

});

//============================= End Jwt Authentication =================

27- Now create new method inside IUserRepository.cs refer the following highlighted lines of code

using Booking.Com\_Clone\_API.Models.Domain;

namespace Booking.Com\_Clone\_API.Repositories

{

public interface IUserRepository

{

// Login

///<summary>

///method to generate JWT token

/// </summary>

string GenerateJwtToken(User user);

}

}

28- Now and now implement inside the UserRepository.cs file refere the following code

///<summary>

///method to generate JWT token

/// </summary>

public string GenerateJwtToken(User user)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_configuration["Jwt:SecretKey"]);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Email, user.Email),

// Add additional claims as needed

}),

Expires = DateTime.UtcNow.AddHours(1), // Token expiration time

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

29- Now implement the login endpoint inside UserController.cs refer the highlighted code

///<summary>

///method to login user and generate JWT token

/// </summary>

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginDto loginDto)

{

try

{

// Validate user credentials

var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == loginDto.Email);

if (user != null)

{

var checkpasswordResult = await dbContext.Users.FirstOrDefaultAsync(u => u.Password == loginDto.Password);

if (checkpasswordResult != null)

{

// Create Token

var jwtToken = userRepository.GenerateJwtToken(user);

var response = new LoginResponseDto

{

JwtToken = jwtToken

};

return Ok(response);

}

}

return BadRequest("Username or password incorrect");

}

catch (Exception ex)

{

// Log the exception

logger.LogError(ex, "An error occurred during login.");

// Return a generic error response

return StatusCode(500, new { message = "An error occurred during login. Please try again later." });

}

}

**Forgot Password & Reset Password Implementation**

30- Now lets start working on forgot password functionality

Create new method inside IUserRespository.cs file refere the following code

///<summary>

///method to generate JWT token

/// </summary>

Task<string> GenerateResetPasswordToken(User user);

31- Now Impement the logic in UserRepository for token generation add the following code

///<summary>

///method to generate reset password token

/// </summary>

public async Task<string> GenerateResetPasswordToken(User user)

{

var token = Guid.NewGuid().ToString();

user.ResetPasswordToken = token;

user.ResetPasswordTokenExpiresAt = DateTime.UtcNow.AddHours(1); // Set token expiration time

await dbContext.SaveChangesAsync();

return token;

}

32- Now add the method to send the reset token to the mail for that add the following code inisde IUserRepository.cs

///<summary>

///method to send mail for verifying user

/// </summary>

Task SendPasswordResetEmailAsync(string email, string token);

33- Now add the logic implementation for sending the reset token through mail add the following code

///<summary>

///method to send mail for verifying user

/// </summary>

public async Task SendPasswordResetEmailAsync(string email, string token)

{

var smtpSettings = \_configuration.GetSection("SmtpSettings");

var fromAddress = smtpSettings["FromAddress"];

var fromName = smtpSettings["FromName"];

if (string.IsNullOrEmpty(fromAddress))

{

throw new ArgumentNullException(nameof(fromAddress), "From address is missing or null in SMTP settings.");

}

var emailMessage = new MailMessage

{

From = new MailAddress(fromAddress, fromName),

Subject = "Reset your password",

//Body = $"Please click the following link to verify your email: https://localhost:7151/api/User/verify-email?email={email}&token={token}",

IsBodyHtml = true

};

emailMessage.To.Add(email);

// HTML email template

string htmlBody = @"

<!DOCTYPE html>

<html lang=""en"">

<head>

<meta charset=""UTF-8"">

<meta name=""viewport"" content=""width=device-width, initial-scale=1.0"">

<title>Booking.com Clone</title>

</head>

<body>

<div>

<h1 style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px; margin-top:10px; margin-bottom:15px;width:200px;"">Booking-clone.com</h1>

<h2>Reset your password</h2>

<p>No worries – it happens!.<br/>Simply click on the button below to choose.<br/>a new one. It’s as easy as that</p>

<a href=""https://localhost:7151/api/User/reset-password?email=" + email + "&token=" + token + @""" style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px;"">Reset Password</a>

</div>

</body>

</html>";

emailMessage.Body = htmlBody;

using (var client = new SmtpClient(smtpSettings["SmtpServer"], int.Parse(smtpSettings["Port"])))

{

client.UseDefaultCredentials = false;

client.Credentials = new NetworkCredential(smtpSettings["Username"], smtpSettings["Password"]);

client.EnableSsl = true;

try

{

await client.SendMailAsync(emailMessage);

logger.LogInformation($"Verification email sent successfully to {email}.");

}

catch (Exception ex)

{

logger.LogError($"Failed to send verification email to {email}. Error: {ex.Message}");

throw; // Rethrow the exception to propagate it up the call stack

}

}

}

34- Now add the forgot password endpoint inside UserController.cs file add the following code

[HttpPost("forgot-password")]

public async Task<IActionResult> ForgotPassword(string email)

{

var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == email);

if (user == null)

{

return BadRequest("User with the provided email does not exist.");

}

// Generate a password reset token

var token = userRepository.GenerateResetPasswordToken(user).ToString();

// Send password reset email

await userRepository.SendPasswordResetEmailAsync(user.Email, token);

return Ok("Password reset email sent successfully.");

}

35- Now reset mail token send successfully through the mail now we implement the rest logic for that we will create the reset-password endpoint for that add the following code

[HttpPost("reset-password")]

public async Task<IActionResult> ResetPassword(string email, string token, string newPassword)

{

var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == email);

if (user == null || user.ResetPasswordToken != token || user.ResetPasswordTokenExpiresAt < DateTime.UtcNow)

{

return BadRequest("Invalid or expired password reset token.");

}

// Reset the user's password

user.Password = BCrypt.Net.BCrypt.HashPassword(newPassword);

await dbContext.SaveChangesAsync();

return Ok("Password reset successfully.");

}

36- Here is the full implementation of IUserRepository.cs

using Booking.Com\_Clone\_API.Models.Domain;

namespace Booking.Com\_Clone\_API.Repositories

{

public interface IUserRepository

{

// Registration

///<summary>

///method to create new registration

/// </summary>

Task<User> CreateUserAsync(User users);

///<summary>

///method to check if user already exists

/// </summary>

Task<bool> EmailExistAsync(string email);

///<summary>

///method to send mail for verifying user

/// </summary>

Task SendVerificationEmailAsync(string email, string verificationToken);

///<summary>

///method to verify mail

/// </summary>

Task<bool> VerifyEmailAsync(string email, string token);

// Login

///<summary>

///method to generate JWT token

/// </summary>

string GenerateJwtToken(User user);

///<summary>

///method to generate JWT token

/// </summary>

Task<string> GenerateResetPasswordToken(User user);

///<summary>

///method to send mail for verifying user

/// </summary>

Task SendPasswordResetEmailAsync(string email, string token);

}

}

37- Here is the full implementation for the UserRepository.cs

using Booking.Com\_Clone\_API.Models.Domain;

using Booking.Com\_Clone\_API.Models.DTO;

using MailKit.Security;

using Microsoft.AspNetCore.Http.HttpResults;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using MimeKit;

using NZWalks.API.Controllers;

using NZWalks.API.Data;

using System.IdentityModel.Tokens.Jwt;

using System.Net;

using System.Net.Mail;

using System.Security.Claims;

using System.Text;

using System.Text.Json;

using static System.Net.WebRequestMethods;

namespace Booking.Com\_Clone\_API.Repositories

{

public class UserRepository : IUserRepository

{

private readonly IConfiguration \_configuration;

private readonly NZWalksDbContext dbContext;

private readonly ILogger<UserRepository> logger;

public UserRepository(NZWalksDbContext dbContext, ILogger<UserRepository> logger,IConfiguration configuration)

{

this.dbContext = dbContext;

this.logger = logger;

\_configuration = configuration;

}

///<summary>

///method to create new registration

/// </summary>

public async Task<User> CreateUserAsync(User user)

{

if (await EmailExistAsync(user.Email))

{

throw new Exception("Email already exists");

}

// Generate verification token

var token = Guid.NewGuid().ToString();

// Hash the password

user.Password = BCrypt.Net.BCrypt.HashPassword(user.Password);

user.ConfirmPassword = BCrypt.Net.BCrypt.HashPassword(user.ConfirmPassword);

user.Password = user.Password;

user.ConfirmPassword = user.ConfirmPassword;

// Set verification token and isVerified status

user.VerificationToken = token;

user.IsVerified = false;

// Add user to database

await dbContext.Users.AddAsync(user);

await dbContext.SaveChangesAsync();

// Send verification email

await SendVerificationEmailAsync(user.Email, token);

logger.LogInformation($"Registration successful. Please verify your email.: {JsonSerializer.Serialize(user)}");

return user;

}

///<summary>

///method to check if user already exists

/// </summary>

public async Task<bool> EmailExistAsync(string email)

{

return await dbContext.Users.AnyAsync(x => x.Email == email);

}

///<summary>

///method to send mail for verifying user

/// </summary>

public async Task SendVerificationEmailAsync(string email, string token)

{

var smtpSettings = \_configuration.GetSection("SmtpSettings");

var fromAddress = smtpSettings["FromAddress"];

var fromName = smtpSettings["FromName"];

if (string.IsNullOrEmpty(fromAddress))

{

throw new ArgumentNullException(nameof(fromAddress), "From address is missing or null in SMTP settings.");

}

var emailMessage = new MailMessage

{

From = new MailAddress(fromAddress, fromName),

Subject = "Verify your email",

//https://localhost:7151/api/User/verify-email?email=danishakhtar539%40gmail.com&token=fc60a64e-c2cb-4824-913d-0764040c1c1d

//Body = $"Please click the following link to verify your email: https://localhost:7151/api/User/verify-email?email={email}&token={token}",

IsBodyHtml = true

};

emailMessage.To.Add(email);

// HTML email template

string htmlBody = @"

<!DOCTYPE html>

<html lang=""en"">

<head>

<meta charset=""UTF-8"">

<meta name=""viewport"" content=""width=device-width, initial-scale=1.0"">

<title>Booking.com Clone</title>

</head>

<body>

<div>

<h1 style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px; margin-top:10px; margin-bottom:15px;width:200px;"">Booking-clone.com</h1>

<h2>Verify your email address</h2>

<p>""Your created an account with the email address:" +email+@""".<br/> Click Confirm to verify the email address and unlock your full account.<br/> We'll also import any bookings you've made with that address""</p>

<a href=""https://localhost:7151/api/User/verify-email?email=" + email + "&token=" + token + @""" style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px;"">Confirm</a>

</div>

</body>

</html>";

emailMessage.Body = htmlBody;

using (var client = new SmtpClient(smtpSettings["SmtpServer"], int.Parse(smtpSettings["Port"])))

{

client.UseDefaultCredentials = false;

client.Credentials = new NetworkCredential(smtpSettings["Username"], smtpSettings["Password"]);

client.EnableSsl = true;

try

{

await client.SendMailAsync(emailMessage);

logger.LogInformation($"Verification email sent successfully to {email}.");

}

catch (Exception ex)

{

logger.LogError($"Failed to send verification email to {email}. Error: {ex.Message}");

throw; // Rethrow the exception to propagate it up the call stack

}

}

}

///<summary>

///method to verify mail

/// </summary>

public async Task<bool> VerifyEmailAsync(string email, string token)

{

var user = await dbContext.Users.FirstOrDefaultAsync(x => x.Email == email);

if (user == null)

{

return false;

}

// Compare the verification token with the token stored in the user object

if (user.VerificationToken == token)

{

// Mark the email as verified

user.IsVerified = true;

//user.VerificationToken = null; // Remove the token after verification

await dbContext.SaveChangesAsync();

return true;

}

return false;

}

///<summary>

///method to generate JWT token

/// </summary>

public string GenerateJwtToken(User user)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_configuration["Jwt:SecretKey"]);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Email, user.Email),

// Add additional claims as needed

}),

Expires = DateTime.UtcNow.AddHours(1), // Token expiration time

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

///<summary>

///method to send mail for verifying user

/// </summary>

public async Task SendPasswordResetEmailAsync(string email, string token)

{

var smtpSettings = \_configuration.GetSection("SmtpSettings");

var fromAddress = smtpSettings["FromAddress"];

var fromName = smtpSettings["FromName"];

if (string.IsNullOrEmpty(fromAddress))

{

throw new ArgumentNullException(nameof(fromAddress), "From address is missing or null in SMTP settings.");

}

var emailMessage = new MailMessage

{

From = new MailAddress(fromAddress, fromName),

Subject = "Reset your password",

//Body = $"Please click the following link to verify your email: https://localhost:7151/api/User/verify-email?email={email}&token={token}",

IsBodyHtml = true

};

emailMessage.To.Add(email);

// HTML email template

string htmlBody = @"

<!DOCTYPE html>

<html lang=""en"">

<head>

<meta charset=""UTF-8"">

<meta name=""viewport"" content=""width=device-width, initial-scale=1.0"">

<title>Booking.com Clone</title>

</head>

<body>

<div>

<h1 style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px; margin-top:10px; margin-bottom:15px;width:200px;"">Booking-clone.com</h1>

<h2>Forgot your password</h2>

<p>No worries – it happense!.<br/>Simply click on the button below to choose.<br/>a new one. It’s as easy as that</p>

<a href=""https://localhost:7151/api/User/reset-password?email=" + email + "&token=" + token + @""" style=""background-color: #007bff; color: #fff; padding: 10px 20px; text-decoration: none; border-radius: 5px;"">Reset Password</a>

</div>

</body>

</html>";

emailMessage.Body = htmlBody;

using (var client = new SmtpClient(smtpSettings["SmtpServer"], int.Parse(smtpSettings["Port"])))

{

client.UseDefaultCredentials = false;

client.Credentials = new NetworkCredential(smtpSettings["Username"], smtpSettings["Password"]);

client.EnableSsl = true;

try

{

await client.SendMailAsync(emailMessage);

logger.LogInformation($"Verification email sent successfully to {email}.");

}

catch (Exception ex)

{

logger.LogError($"Failed to send verification email to {email}. Error: {ex.Message}");

throw; // Rethrow the exception to propagate it up the call stack

}

}

}

///<summary>

///method to generate reset password token

/// </summary>

public async Task<string> GenerateResetPasswordToken(User user)

{

var token = Guid.NewGuid().ToString();

user.ResetPasswordToken = token;

user.ResetPasswordTokenExpiresAt = DateTime.UtcNow.AddHours(1); // Set token expiration time

await dbContext.SaveChangesAsync();

return token;

}

}

}

38- Here is the full controller implementation

using AutoMapper;

using Booking.Com\_Clone\_API.Models.Domain;

using Booking.Com\_Clone\_API.Models.DTO;

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.IdentityModel.Tokens;

using NZWalks.API.Controllers;

using NZWalks.API.CustomeActionFilter;

using NZWalks.API.Data;

using NZWalks.API.Repositories;

using System.Data;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using System.Text.Json;

namespace Booking.Com\_Clone\_API.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class UserController : ControllerBase

{

private readonly NZWalksDbContext dbContext;

private readonly IUserRepository userRepository;

private readonly IMapper mapper;

private readonly ILogger<RegionsController> logger;

private readonly IConfiguration configuration;

public UserController(NZWalksDbContext dbContext,

IUserRepository userRepository,

IMapper mapper,

ILogger<RegionsController> logger,

IConfiguration configuration)

{

this.dbContext = dbContext;

this.userRepository = userRepository;

this.mapper = mapper;

this.logger = logger;

this.configuration = configuration;

}

///<summary>

///method to create new registration

/// </summary>

[HttpPost]

[ValidateModel]

//[Authorize(Roles = "Writer")]

public async Task<IActionResult> CreateUser([FromBody] UserDto userDto)

{

try

{

// Using Automapper makes much clear code as compare to above method

var regionDomainModel = mapper.Map<User>(userDto);

regionDomainModel = await userRepository.CreateUserAsync(regionDomainModel);

logger.LogInformation($"Finished User created request data: {JsonSerializer.Serialize(regionDomainModel)}");

// Using Automapper makes much clear code as compare to above method

var regionDto = mapper.Map<UserDto>(regionDomainModel);

return Ok(regionDto);

}

catch (Exception ex)

{

logger.LogError(ex, ex.Message);

throw;

}

}

///<summary>

///method to verify mail

/// </summary>

[HttpGet("verify-email")]

public async Task<IActionResult> VerifyEmailAsync(string email, string token)

{

if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(token))

return BadRequest("Email and token are required");

var isVerified = await userRepository.VerifyEmailAsync(email, token);

if (isVerified)

{

logger.LogInformation($"Verification Successfull: {JsonSerializer.Serialize("Email verified successfully")}");

return Ok("Email verified successfully");

}

else

{

logger.LogInformation($"Verification failed: {JsonSerializer.Serialize("Invalid email or token")}");

return BadRequest("Invalid email or token");

}

}

///<summary>

///method to login user and generate JWT token

/// </summary>

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginDto loginDto)

{

try

{

// Validate user credentials

var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == loginDto.Email);

if (user != null)

{

var checkpasswordResult = await dbContext.Users.FirstOrDefaultAsync(u => u.Password == loginDto.Password);

if (checkpasswordResult != null)

{

// Create Token

var jwtToken = userRepository.GenerateJwtToken(user);

var response = new LoginResponseDto

{

JwtToken = jwtToken

};

return Ok(response);

}

}

return BadRequest("Username or password incorrect");

}

catch (Exception ex)

{

// Log the exception

logger.LogError(ex, "An error occurred during login.");

// Return a generic error response

return StatusCode(500, new { message = "An error occurred during login. Please try again later." });

}

}

[HttpPost("forgot-password")]

public async Task<IActionResult> ForgotPassword(string email)

{

var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == email);

if (user == null)

{

return BadRequest("User with the provided email does not exist.");

}

// Generate a password reset token

var token = userRepository.GenerateResetPasswordToken(user).ToString();

// Send password reset email

await userRepository.SendPasswordResetEmailAsync(user.Email, token);

return Ok("Password reset email sent successfully.");

}

[HttpPost("reset-password")]

public async Task<IActionResult> ResetPassword(string email, string token, string newPassword)

{

var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == email);

if (user == null || user.ResetPasswordToken != token || user.ResetPasswordTokenExpiresAt < DateTime.UtcNow)

{

return BadRequest("Invalid or expired password reset token.");

}

// Reset the user's password

user.Password = BCrypt.Net.BCrypt.HashPassword(newPassword);

await dbContext.SaveChangesAsync();

return Ok("Password reset successfully.");

}

}

}

39- Here is the full implementation of Program.cs file

using Booking.Com\_Clone\_API.Repositories;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.FileProviders;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using NZWalks.API.Data;

using NZWalks.API.Mappings;

using NZWalks.API.Middlewares;

using NZWalks.API.Repositories;

using Serilog;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Serilog Console to get information on console window

var logger = new LoggerConfiguration()

.WriteTo.Console()

// To get log in text file opion above line sufficient ot handle the log

.WriteTo.File("Logs/Booking\_logs.txt",rollingInterval: RollingInterval.Minute)

.MinimumLevel.Information()

.CreateLogger();

builder.Logging.ClearProviders();

builder.Logging.AddSerilog(logger);

// Add services to the container.

builder.Services.AddControllers();

// to upload image on server

builder.Services.AddHttpContextAccessor();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

// Open the Authorization Bearer Token in Swagger

builder.Services.AddSwaggerGen(options =>

{

options.SwaggerDoc("v1", new OpenApiInfo { Title = "Booking.com API", Version = "v1" });

options.AddSecurityDefinition(JwtBearerDefaults.AuthenticationScheme, new OpenApiSecurityScheme

{

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.ApiKey,

Scheme = JwtBearerDefaults.AuthenticationScheme

});

options.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = JwtBearerDefaults.AuthenticationScheme

},

Scheme = "Oauth2",

Name = JwtBearerDefaults.AuthenticationScheme,

In = ParameterLocation.Header

},

new List<string>()

}

});

});

// Add DbContext class to connect with database and reate connectin string

builder.Services.AddDbContext<NZWalksDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("BookingConnectionString")));

// Email notification

builder.Services.Configure<Booking.Com\_Clone\_API.Models.Domain.EmailConfiguration>(builder.Configuration.GetSection("EmailConfiguration"));

// Injected Repository

// Use nuge package AutoMapper Injection

builder.Services.AddScoped<IRegionRepository, SQLRegionRepository>();

builder.Services.AddScoped<ITokenRepository, TokenRepositoryToken>();

builder.Services.AddScoped<IImageRepository, ImageRepository>();

builder.Services.AddScoped<IUserRepository, UserRepository>();

// Injectd AutoMapper

builder.Services.AddAutoMapper(typeof(AutoMapperProfiles));

//============================= Start Jwt Authentication =================

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(builder.Configuration["Jwt:SecretKey"]))

});

//============================= End Jwt Authentication =================

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

// Global exception handler

app.UseMiddleware<ExceptionHandlerMiddleware>();

app.UseHttpsRedirection();

// Authentication

app.UseAuthentication();

app.UseAuthorization();

// to access the local file path using URL

app.UseStaticFiles(new StaticFileOptions

{

FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "Images")),

RequestPath = "/Images"

});

app.MapControllerRoute(

name: "verify-email",

pattern: "verify-email/{email}/{token}",

defaults: new { controller = "User", action = "VerifyEmailAsync" });

app.MapControllers();

app.Run();

40- Here the full implementation of appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"BookingConnectionString": "Server=AZKBAN\\MSSQLSERVER01;Database=Booking\_Clone;Trusted\_Connection=True;TrustServerCertificate=True"

},

"Jwt": {

"SecretKey": "bbuosyJSSGPOosflUSJ75JJHst6yjjjST5rt655Y77uhSYSko098HHhgst",

"Issuer": "https://localhost:7151",

"Audience": "https://localhost:7151"

},

"SmtpSettings": {

"SmtpServer": "smtp.gmail.com",

"Port": "587",

"Username": "mddanish867@gmail.com",

"Password": "dyptkwuddyrzdvps",

"FromAddress": "bookingclone@booking.com", // Make sure this is properly configured

"FromName": "BookingClone.Com",

"VerificationLink": "https://bookingclone.com/verify"

}

}

**Congratulation you are now authenticated**