# Процес за решаване на задачата SharedTrip

## Имплементиране на базата данни

Създаваме папка Models в папката Data. Създаваме клас за всеки един от моделите описани в заданието. Всички валидационни константи ги поставяме в папка Common/GlobalConstants.cs.

Максимална дължина на Guid – 36, на хешираната парола – 64.

### User

* Has an Id – a **string, Primary Key**
* Has a Username – a string with **min length** **5** and **max length 20** (**required**)
* Has an Email - a string (**required**)
* Has a Password – a string with **min length** **6** and **max length 20** - hashed in the database (**required**)

Has **UserTrips** collection – a **UserTrip** type

using SharedTrip.Common;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace SharedTrip.Data.Models

{

public class User

{

public User()

{

UserTrips = new HashSet<UserTrip>();

Id = Guid.NewGuid().ToString();

}

[Key]

[StringLength(GlobalConstants.GuidMaxLength)]

public string Id { get; set; }

[Required]

[StringLength(GlobalConstants.UserUsernameMaxLength)]

public string Username { get; set; }

[Required]

[StringLength(GlobalConstants.UserEmailMaxLength)]

public string Email { get; set; }

[Required]

[StringLength(GlobalConstants.UserPasswordHashlMaxLength)]

public string Password { get; set; }

public ICollection<UserTrip> UserTrips { get; set; }

}

}

### Trip

* Has an Id – a **string, Primary Key**
* Has a StartPoint – a string (**required**)
* Has a EndPoint – a string (**required**)
* Has a DepartureTime – a datetime (**required**)
* Has a Seats – an integer with **min value 2** and **max value 6** (**required**)
* Has a Description – a string with **max length 80** (**required**)
* Has a ImagePath – a string
* Has **UserTrips** collection – a **UserTrip** type

using SharedTrip.Common;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace SharedTrip.Data.Models

{

public class Trip

{

public Trip()

{

UserTrips = new HashSet<UserTrip>();

Id = Guid.NewGuid().ToString();

}

[Key]

[StringLength(GlobalConstants.GuidMaxLength)]

public string Id { get; set; }

[Required]

[StringLength(GlobalConstants.TripStartEndPointMaxLength)]

public string StartPoint { get; set; }

[Required]

[StringLength(GlobalConstants.TripStartEndPointMaxLength)]

public string EndPoint { get; set; }

[Required]

public DateTime DepartureTime { get; set; }

[Required]

[Range(GlobalConstants.TripSeatsMinValue, GlobalConstants.TripSeatsMaxValue)]

public int Seats { get; set; }

[Required]

[StringLength(GlobalConstants.TripDescriptionMaxLength)]

public string Description { get; set; }

[StringLength(GlobalConstants.TripImagePathMaxLength)]

public string ImagePath { get; set; }

public ICollection<UserTrip> UserTrips { get; set; }

}

}

### UserTrip

* Has UserId – a **string**
* Has User – a User object
* Has **TripId**– a **string**
* Has Trip – a Trip object

using System.ComponentModel.DataAnnotations.Schema;

namespace SharedTrip.Data.Models

{

public class UserTrip

{

[ForeignKey(nameof(User))]

public string UserId { get; set; }

public User User { get; set; }

[ForeignKey(nameof(Trip))]

public string TripId { get; set; }

public Trip Trip { get; set; }

}

}

### Сменяме ConnectionString в DatabaseConfiguration:

namespace SharedTrip.Data

{

public class DatabaseConfiguration

{

public const string ConnectionString =

@"Server=RAPTOR\SQLEXPRESS;Database=SharedTrip;Trusted\_Connection=True;Integrated Security=True;";

//@"Server=.;Database=SharedTrip;Trusted\_Connection=True;Integrated Security=True;";

}

}

### Добявяме DbSet и композитен ключ в ApplicationDbContext

namespace SharedTrip.Data

{

using Microsoft.EntityFrameworkCore;

using SharedTrip.Data.Models;

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext()

{

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlServer(DatabaseConfiguration.ConnectionString);

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<UserTrip>()

.HasKey(ut => new { ut.UserId, ut.TripId });

}

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

public DbSet<Trip> Trips { get; set; }

public DbSet<UserTrip> UserTrips { get; set; }

}

}

### Създаване на миграция през Package Manager Console

Задаваме стартовия проект да бъде SharedTrip. В конзолата изписваме командата: **add-migration InitialMigration**. След това **update-database**.

## Коригиране на изгледите

В файла [Views/Layout.cshtml](https://github.com/beshkata/SoftUni/commit/a08b889e852c1393b8af8a33e2eca93dc62b91c2#diff-bf5d547d421c46d77ba7f1a87c73ad09294c947d4de53345b6ea145ecca9fcca):

<head>

<title>Shared Trip Platform</title>

<meta charset="UTF-8">

<link rel="stylesheet" href="/css/bootstrap.css" />

</head>

В файла [/Views/Error.cshtml](https://github.com/beshkata/SoftUni/commit/a08b889e852c1393b8af8a33e2eca93dc62b91c2#diff-f1ce1e7be4fa883620029c26105f6c0215527e5f8553517d6c0b516a4e45018f):

<section class="py-5">

<div class="container">

<h1>Your request is not valid. Errors:</h1>

<ul>

{{foreach}}

{

<li>{{ErrorMessage}}</li>

}

</ul>

</div>

</section>

## Имплементация на Изгледите

Създаваме Контролер класове за всяка една от папките в Veiws. Във всеки един контролер правим методи за всеки един от изгледите за съответния контролер. Всички контролери наследяват класа Controller.

public class UsersController : Controller

{

public UsersController(Request request)

: base(request)

{

}

public Response Login()

{

if (User.IsAuthenticated)

{

return Redirect("/Trips/All");

}

return View();

}

public Response Register()

{

if (User.IsAuthenticated)

{

return Redirect("/Trips/All");

}

return View();

}

}

Проверяваме дали имаме логнат потребител, тъй като в страницата Login и Register са достъпни само при не логнат потребител. Ако има потребител го препращаме към Trips/All.

public class TripsController : Controller

{

public TripsController(Request request)

: base(request)

{

}

[Authorize]

public Response Add() => View();

[Authorize]

public Response All()

{

return View();

}

[Authorize]

public Response Details() => View();

}

Атрибута [Authorize] означава, че трябва да има логнат потребител, за да се достъпят изгледите

## Имплементиране на Repository pattern

За индиректна работа с базата данни имплементираме Repository pattern. В папка Data създаваме папка Common. В нея създаваме интерфейс IRepository

public interface IRepository

{

void Add<T>(T entity) where T : class;

IQueryable<T> All<T>() where T : class;

int SaveChanges();

}

Създаваме клас Repository за имплементиране на интерфейса:

using Microsoft.EntityFrameworkCore;

using System.Linq;

namespace SharedTrip.Data.Common

{

public class Repository

{

private readonly DbContext dbContext;

public Repository(ApplicationDbContext context)

{

dbContext = context;

}

public void Add<T>(T entity) where T : class

{

DbSet<T>().Add(entity);

}

public IQueryable<T> All<T>() where T : class

{

return DbSet<T>().AsQueryable();

}

public int SaveChanges()

{

return dbContext.SaveChanges();

}

private DbSet<T> DbSet<T>() where T : class

{

return dbContext.Set<T>();

}

}

}

Работата с базата вече ще става чрез инстанция на този клас.

## Създаване на Services

Създаваме папка Contracts и в нея интерфейс за потребителите – IUserService

public interface IUserService

{

}

Създаваме папка Services. В нея създаваме клас UserService, който наследява IUserService.

Регистрираме сървиза в Startup класа:

server.ServiceCollection

.Add<IUserService, UserService>();

В класа UsersController създаваме поле с инстанция на UserService, а самата инстанция я инжектираме през конструктора на UsersController

private readonly UserService \_userService;

public UsersController(Request request, UserService userService)

: base(request)

{

\_userService = userService;

}

Първия сървиз за потребителя е регистрация на потребител. От изгледа Register.cshtml се вижда, какви свойства трябва да има модела(гледаме атрибута name на всяка една input форма): username, email, password, confirmPassword. Коригираме имената да са с голяма буква в Register.cshtml за да съвпадат с конвенцията за именуване на свойства на класовете: Username, Email, Password, ConfirmPassword.

В папка Models проекта(не в папката Data) създаваме папка UserViewModels. Създаваме UserRegistrationViewModel клас.

public class UserRegistrationViewModel

{

public string Username { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string ConfirmPassword { get; set; }

}

Създаване на Action Register() в UserController класа, който се изпълнява при Post заявка на Register изгледа:

public Response Register(UserRegistrationViewModel model)

{

var (isValid, errors) = \_userService.ValidateModel(model);

if (!isValid)

{

return View(errors, "/Error");

}

try

{

\_userService.RegisterUser(model);

}

catch (ArgumentException aex)

{

return View(new List<ErrorViewModel>() { new ErrorViewModel(aex.Message) }, "/Error");

}

catch (Exception)

{

return View(new List<ErrorViewModel>() { new ErrorViewModel("Unexpected Error") }, "/Error");

}

return Redirect("/Users/Login");

}

Методът ValidateModel на UserService класа прави проверка дали въведените данните са валидни и в променливата isValid връща дали е валиден UserRegistrationViewModel, а в променливата errors връща списък с ErrorViewModel обекти с информация за невалидните данни. Класа ErrorViewModel създаваме в папката Models:

public class ErrorViewModel

{

public string ErrorMessage { get; init; }

public ErrorViewModel(string message)

{

ErrorMessage = message;

}

}

Ако модела е невалиден, връщаме страницата "/Error", със списъка на грешките. Ако обекта е валиден в try – catch блок добавяме потребителя в базата данни чрез метода RegisterUser на UserService.

В интерфейса IUserService регистрираме методите

(bool isValid, IEnumerable<ErrorViewModel> errors) ValidateModel(UserRegistrationViewModel model);

void RegisterUser(UserRegistrationViewModel model);

Имплементираме ги в класа UserService като в конструктора инжектираме инстанция на IRepository

private readonly IRepository \_repo;

public UserService(IRepository repo)

{

\_repo = repo;

}

public (bool isValid, IEnumerable<ErrorViewModel> errors)

ValidateModel(UserRegistrationViewModel model)

{

List<ErrorViewModel> errors = new List<ErrorViewModel>();

bool isValid = true;

if (model.Username == null ||

model.Username.Length < 5 ||

model.Username.Length > 20)

{

isValid = false;

errors.Add(new ErrorViewModel("Username is requred and must be between 5 and 20 characters"));

}

if (string.IsNullOrWhiteSpace(model.Email))

{

isValid = false;

errors.Add(new ErrorViewModel("Email is required"));

}

if (model.Password == null ||

model.Password.Length < 6 ||

model.Password.Length > 20)

{

isValid = false;

errors.Add(new ErrorViewModel("Password is requred and must be between 6 and 20 characters"));

}

if (model.Password != model.ConfirmPassword)

{

isValid = false;

errors.Add(new ErrorViewModel("Password and ConfirmPasswor are not the same"));

}

return (isValid, errors);

}

public void RegisterUser(UserRegistrationViewModel model)

{

var userExists = GetUserByUsername(model.Username) != null;

if (userExists)

{

throw new ArgumentException("Registration failed");

}

User user = new User()

{

Email = model.Email,

Username = model.Username

};

user.Password = HashPassword(model.Password);

\_repo.Add(user);

\_repo.SaveChanges();

}

private User GetUserByUsername(string username)

{

return \_repo.All<User>()

.FirstOrDefault( u => u.Username == username);

}

private string HashPassword(string password)

{

byte[] passwordArray = Encoding.UTF8.GetBytes(password);

using(SHA256 sha256 = SHA256.Create())

{

return Convert.ToBase64String(sha256.ComputeHash(passwordArray));

}

}

}

Чрез метода GetUserByUsername проверяваме дали няма вече такъв потребител регистриран, а чрез метода HashPassword хешираме паролата преди да я запишем в базата данни

В Startup класа добавяме IRepository и ApplicationDbContext в ServiceCollection

server.ServiceCollection

.Add<IUserService, UserService>()

.Add<IRepository, Repository>()

.Add<ApplicationDbContext>();

Създаване на Action Login()

Създаваме модела за логване. В папката Models/UserViewModels създаваме класа

public class UserLoginViewModel

{

public string Username { get; set; }

public string Password { get; set; }

}

В UserController добавяме метода

[HttpPost]

public Response Login(UserLoginViewModel model)

{

Request.Session.Clear();

(string userId, bool isCorrect) = \_userService.IsLoginCorrect(model);

if (isCorrect)

{

SignIn(userId);

CookieCollection cookies = new CookieCollection();

cookies.Add(Session.SessionCookieName,

Request.Session.Id);

return Redirect("/Trips/All");

}

return View(new List<ErrorViewModel>() { new ErrorViewModel("Login incorrect") }, "/Error");

}

Метода изчиства текущата сесия. Метода IsLoginCorrect логва потребителя, като връща id на потребителя и дали е логнат успешно. Ако е логнат успешно го регистрира като логнат (SignIn) и добавя сесията към бисквитките.

Добавяме метода IsLoginCorrect към IUserService и го имплементираме в UserService:

public (string userId, bool isCorrect) IsLoginCorrect(UserLoginViewModel model)

{

bool isCorrect = false;

string userId = String.Empty;

var user = GetUserByUsername(model.Username);

if (user != null)

{

isCorrect = user.Password == HashPassword(model.Password);

}

if (isCorrect)

{

userId = user.Id;

}

return (userId, isCorrect);

}

В него проверяваме дали потребител с такова име съществува, ако съществува проверяваме паролата дали е същата ако да връщаме id на потребителя и дали е валиден логина.