NÁSKOK
DÍKY
ZNALOSTEM

PROFINIT

NSWI026, 5. cvičení

Představení frameworků a kostry aplikace

Jonáš Klimeš, Dalibor Zeman

25. března 2020

Prerequisities

- .NET Core 3.1 - https://dotnet.microsoft.com/download
  - Runtime: https://dotnet.microsoft.com/download/dotnet-core/current/runtime
  - SDK

- IDE (jedno z následujících):
  - Visual Studio 2019 – preferované (stačí i community edice)
  - Visual Studio Core: https://code.visualstudio.com/

- Git

Cíl následujících 2 až 3 cvičení

- Seznámit se s kostrou aplikace
- Představit si technologie

PROFINIT

Průběh

- Představení kostry aplikace a architektury

- Bude se opakovat
  - Představení technologie
  - Samostatný úkol – TODOs v kódu
  - Nápovědy a ukázka řešení

PROFINIT

Agenda

1. Použité technologie
2. Úkol 1: Příprava prostředí a spuštění aplikace
3. Zvolená architektura a datový model
4. Úkol 2: Repository
5. Úkol 3: MVC Controller
6. Úkol 5: Export do CSV
7. Úkol 6: Integrační testy
8. Úkol 7: Unit testy
9. Úkol 8: Napojení na externí službu

PROFINIT

1

Základ aplikace a technologie

Technologie

- Server
  - C# 8, .NET Core 3.1
  - ASP.NET Core Web Application
  - ASP.NET Core MVC, resp. API (Views nepotřebujeme)
  - Entity Framework Core, SQLite EF Core Database Provider
  - RestSharp – zjednodušení integrace na externí služby
  - xUnit – testovací framework
  - SQLite – one-file based DB
  - IIS Express

- Klient – nebudeme upravovat
  - Aurelia.js

Úkol 1: Příprava prostředí a spuštění aplikace

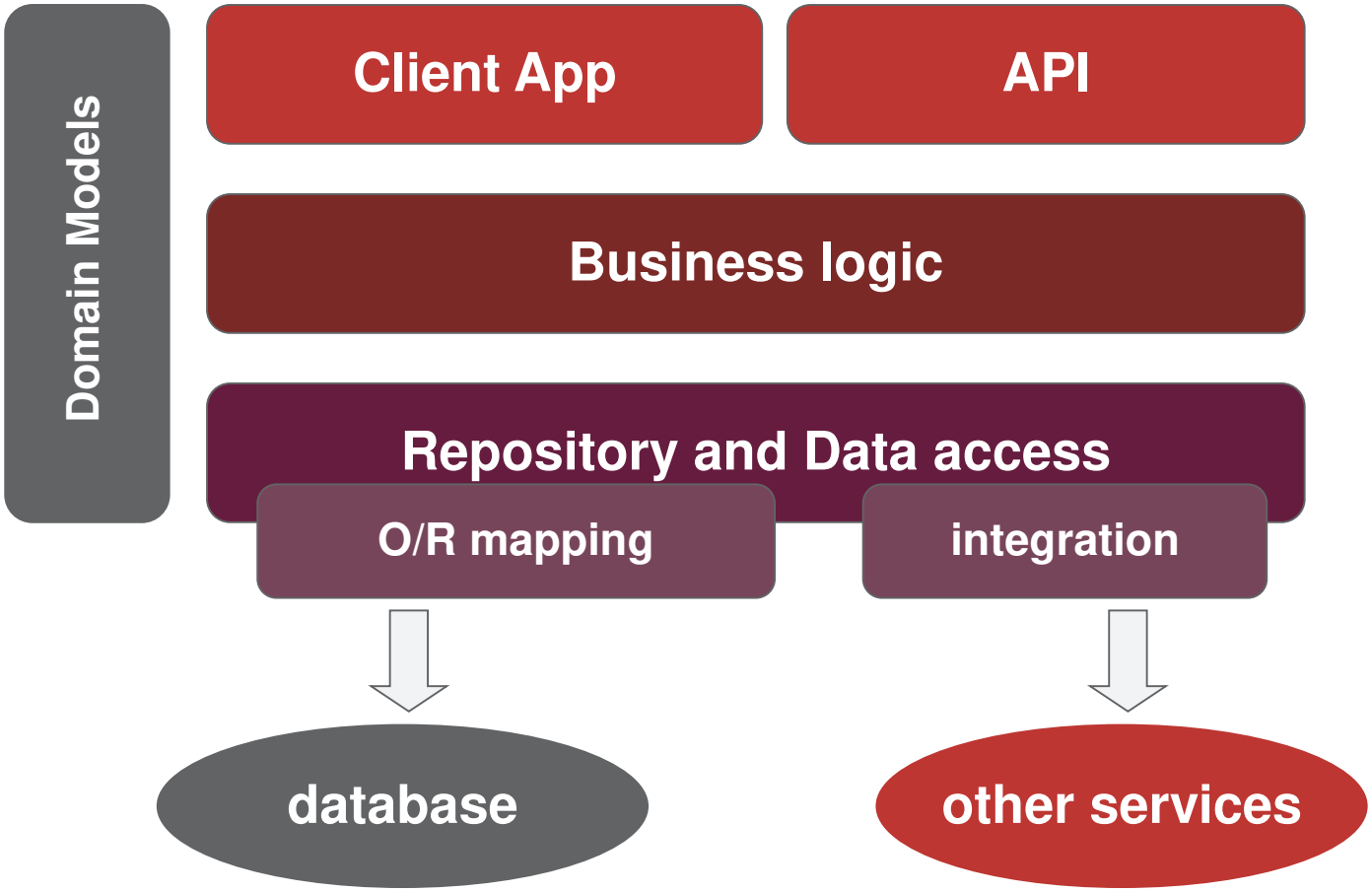
- Měli byste mít nainstalováno
  - IDE (VS 2019, nebo VS Code)

- Naklonujte si repozitář
  - https://github.com/profinit/FlightLogDotNet

- Otevřete solution, nebo složku v IDE
- Obnovte nuget balíčky
- Zkompilujte a spusťte aplikaci
- Aplikace běží na https://localhost:44313/

PROFINIT

Schéma architektury



PROFINIT

Architektura a moduly

- wwwroot
  - statický obsah pro frontend
- Controllers – vystavuje REST API pro frontend
- Facades – definuje rozhraní pro funkcionalitu
- Operations – obsahuje byznys logiku
- Models – obsahuje doménové modely
- Integration – obsahuje integrace na externí služby
- Repositories – obsahuje integraci na databázi
  - Entities – obsahuje databázové entity

PROFINIT

Datový model

FlightStart
+ Id: long
+ Towplane: Flight
+ Glider: Flight

Flight
+ Id: long
+ Type: FlightType
+ Note: string
+ Task: string
+ Copilot: Person
+ Pilot: Person
+ Airplane: Airplane
+ LandingTime: DateTime?
+ TakeoffTime: DateTime

Person
+ Id: long
+ PersonType: PersonType
+ FirstName: string
+ LastName: string
+ Address: Address
+ MemberId: long

Address
+ Id: long
+ Street: string
+ City: string
+ PostalCode: string
+ Country: string

Airplane
+ Id: long
+ ClubAirplane: ClubAirplane
+ GuestAirplaneImmatriculation: string
+ GuestAirplaneType: string

ClubAirplane
+ Id: long
+ Immatriculation: string
+ AirplaneType: AirplaneType
+ Archive: bool

AirplaneType
+ Id: long
+ Type: string
+ MaxCapacity: int

PROFINIT

2a

Objektově-relační mapování

ORM – Entity Framework Core

- Co typicky chceme pro každý objekt
  - Vkládání
  - Editaci, mazání
  - Načtení podle kritérií (select ... where ...)

2b

AutoMapper

AutoMapper

- Aplikace typicky obsahuje velké množství objektů
  - Databázové entity,
  - Bussiness Modely,
  - View Modely,
  - …
- Chceme mapovat jednotlivé objekty na sebe
  - Psát všechny mapování ručně je spoustu práve
  - Na které straně mapování uchovávat (DB, Bussines, View)
- AutoMapper umí:
  - Mapování stejně se jmenujících položek
  - Zploštění entity (Adress.City -> AdressCity)
  - Rekurnizvní mapování složitější položek
  - Zvalidování mapování všech položek (defaultně výchozích, nebo i zdrojových)

Úkol 2: Repository

- Pro každý krok najdete v kódu TODO komentář

- Otevřete aplikaci na homepage - nejsou zde zobrazeny žádné lety
- Rozehřívací úkol
  - Ve třídě FlightRepository vytvořte metodu pro načtení všech letů podle typu
  - Ve třídě FlightRepositoryTest dokončete test metody pro načtení letů kluzáků
- Implementace používané metody
  - Ve třídě FlightRepository vytvořte metodu pro načtení letů
  - Ve třídě FlightRepositoryTest dokončete test metody
  - Ve třídě FlightFacade použijte vytvořenou metodu pro načtení dat
- Zkontrolujte, že na homepage jsou zobrazeny lety

Nápověda 1 – první metoda Repository

- FlightRepository

```
public IList<FlightModel> GetFlightsOfType(FlightType type)
{
        using var dbContext = new LocalDatabaseContext();

        var flights = dbContext.Flights
                .Where(flight => flight.Type == type);

        return mapper.ProjectTo<FlightModel>(flights).ToList();
}
```

- FlightRepositoryTest

```
var result = flightRepository
        .GetFlightsOfType(FlightType.Glider);
```

PROFINIT

Nápověda 2 – druhá metoda repository

- Načtení všech letadel ve vzduchu

```
public IList<FlightModel> GetAirplanesInAir()
{
        using var dbContext = new LocalDatabaseContext();

        var flights = dbContext.Flights
        .Include(flight => flight.Airplane)
        .Include(flight => flight.Copilot)
        .Include(flight => flight.Pilot)
        .Where(flight => flight.LandingTime == null);

        return mapper.ProjectTo<FlightModel>(flights).ToList();
}
```

PROFINIT

Nápověda 3 – druhá metoda repository

- Načtení všech letadel ve vzduchu v požadovaném pořadí

```
public IList<FlightModel> GetAirplanesInAir()
{
        using var dbContext = new LocalDatabaseContext();

        var flights = dbContext.Flights
        .Include(flight => flight.Airplane)
        .Include(flight => flight.Copilot)
        .Include(flight => flight.Pilot)
        .Where(flight => flight.LandingTime == null)
        .OrderBy(flight => flight.TakeoffTime)
        .ThenBy(flight => flight.Type);

        return mapper.ProjectTo<FlightModel>(flights).ToList();
}
```

PROFINIT

Nápověda 4 – Facade

- Stačí metodu pouze provolat

```
internal IEnumerable<FlightModel> GetAirplanesInAir()
{
        return flightRepository.GetAirplanesInAir();
}
```

PROFINIT

3

Vystavení REST API

REST controller

- Metoda + anotace
- ASP.NET MVC
- Url se skládá ze jména Controlleru a attributu („Flight/InAir“)

```
[ApiController]
[Route("[controller]")]
public class FlightController : ControllerBase
{
        private readonly ILogger<FlightController> logger;
        private readonly FlightFacade flightFacade;

        public FlightController(ILogger<FlightController> logger,
                                FlightFacade flightFacade)
        {
                this.logger = logger;
                this.flightFacade = flightFacade;
        }

        [HttpGet("InAir")]
        public IEnumerable<FlightModel> GetPlanesInAir()
        {
                logger.LogDebug("Get airplanes in Air.");
                return flightFacade.GetAirplanesInAir();
        }
```

Úkol 3: ASP.NET Core MVC – REST endpoint

- Otevřete obrazovku pro zadání nového letu – mělo by vyskočit chybová zpráva, že se nepodařilo načíst seznam letadel
- Ze třídy AirplaneController vytvořte funkční Controller, který vrátí seznam letadel
- Zrestartujte aplikaci a zadejte nový let

- Inspirace: ostatní *Controller třídy

PROFINIT

Nápověda 1 – prázdná odpověď

```
[ApiController]
[Route("[controller]")]
public class AirplaneController : ControllerBase
{
        [HttpGet]
        public IEnumerable<AirplaneModel> Get()
        {
                return new List<AirplaneModel>();
        }
}
```

PROFINIT

Nápověda 2 – napojení na facade

```
[ApiController]
[Route("[controller]")]
public class AirplaneController : ControllerBase
{
        private readonly AirplaneFacade airplaneFacade;

        public AirplaneController(
                AirplaneFacade airplaneFacade)
        {
                this.airplaneFacade = airplaneFacade;
        }

        [HttpGet]
        public IEnumerable<AirplaneModel> Get()
        {
                return airplaneFacade.GetClubAirplanes();
        }
}
```

Nápověda 3 – i s logováním

```
[ApiController]
[Route("[controller]")]
public class AirplaneController : ControllerBase
{
        private readonly ILogger<AirplaneController> logger;
        private readonly AirplaneFacade airplaneFacade;

        public AirplaneController(
                ILogger<AirplaneController> logger,
                AirplaneFacade airplaneFacade)
        {
                this.logger = logger;
                this.airplaneFacade = airplaneFacade;
        }

        [HttpGet]
        public IEnumerable<AirplaneModel> Get()
        {
                logger.LogDebug("Get airplanes.");
                return airplaneFacade.GetClubAirplanes();
        }
}
```

PROFINIT

4

Depencency Injection

Kde se vezme AirplaneFacade

- ASP.NET Core má defaultní Dependency Injection
  - V Startup je nakonfigurován, aby injektoval správnou třídu

```
services.AddScoped<AirplaneFacade, AirplaneFacade>();
services.AddScoped<IAirplaneRepository, AirplaneRepository>();
```

  - Dá se nastavit životnost objektů (Transient, Scoped, Singleton)
  - Pozor na hierarchii tříd a dlouhotrvající životnost

5

Export do CSV

Úkol 5: Implementujte export do CSV

- Vyzkoušejte export do CSV v GUI a podívejte se, co se stane
- Ve třídě GetExportToCsvOperation implementujte metodu Execute
- Vyzkoušejte soubor stáhnout z GUI a zkontrolujte jeho správnost

- Jak má CSV vypadat? Na co byste se zeptali zákazníka při analýze?

PROFINIT

6

Automatické testy

Unit vs. Integrační testy

- Unit (jednotkový) test
  - Testuje jednu jednotku funkcionality
  - Minimalizuje počet závislostí
  - Izolovaně od prostředí (externí služby, DB...)
  - Používá zjednodušené náhrady svých závislostí
    - Mocky a/nebo stuby

- Integrační (systémové) testy
  - Testují, jak více jednotek funguje společně
  - Testují společně s infrastrukturou
    - Může být zjednodušená (in memory DB místo klasické)
  - Podpora Springu – sestavení aplikačního kontextu v testu – bude dále

- Každé se hodí na různé situace, používá se obojí
  - Můžeme mixovat – otestovat hlavní scénář společně se závislostmi (integrační) a otestovat okrajové podmínky izolovaně (unit test)

Unit a integrační testy

- Využíváme xUnit
- Je potřeba stejně jako u serveru zajistit nastavení DI
- Již máme testovací DB

PROFINIT

Úkol 6 – integrační test CSV exportu

- Naimplementujte test v GetExportToCsvOperationTests, aby otestoval váš export do CSV
- Zkontrolujte, že test prochází

- Proč je to integrační test?
  - Testuje nejen Operaci, ale i Repository a data v DB
  - V tomto případě se dal použít i unit test

PROFINIT

Nápověda 1: Porovnání se souborem

- Soubor s očekávaným výstupem
  - Do solution si můžeme přidat soubor csv, který má být vyprodukován a porovnávat výsledek s ním

Moq (mockovací framework)

- Nahrazuje závislosti tříd za běhu vytvořenými mocky (implementují stejné rozhraní)
- Postup testování:
  1. Použijte mockovací Framework pro vytvoření mocků
  2. Nadefinujte, jak se má mock chovat v daném testovacím scénáři
     - Jaká metoda bude zavolaná
     - Co má vrátit
  3. Vykonejte testovací scénář
  4. Ověřte, že se mockované metody opravdu zavolali
     - Toto je přidaná hodnota mockování – pokud byste použili reálný objekt, tak nezjistíte, jestli se volal.

Úkol 7 – unit test PersonService

- Otevřete si PersonServiceTest, obsahuje 2 hotové testy
- Naimplementujte třetí test podle TODO
- Zkontrolujte, že testy prochází

Mockito – příklad z PersonServiceTest

```
[Fact]
public void Execute_ShouldReturnExistingClubMember()
{
        // Arrange
        var createPersonOperation = CreateCreatePersonOperation();
        PersonModel personModel = new PersonModel
        {
                FirstName = "Jan",
                LastName = "Novák",
                MemberId = 3
        };
        long id = 333;
        mockPersonRepository.Setup(repository =>
                repository.TryGetPerson(personModel, out id))
                .Returns(true);

        // Act
        var result = createPersonOperation.Execute(personModel);

        // Assert
        Assert.Equal(id, result);
        mockRepository.VerifyAll();
}
```

PROFINIT

Nápověda 1: Osoba není v lokální DB

```
mockPersonRepository.Setup(repository =>
        repository.TryGetPerson(personModel, out id))
        .Returns(false);
```

PROFINIT

Nápověda 2: Osoba je ve klubové databázi

- Umožníme uložit jen osobu, kterou jsme vrátili z databáze

```
PersonModel clubUser = new PersonModel
{
        FirstName = "Karel",
        LastName = "Lucemburský",
        MemberId = 444
};

mockClubUserDatabase.Setup(repository =>
        repository.TryGetClubUser(444, out clubUser))
        .Returns(true);
mockPersonRepository.Setup(repository =>
        repository.CreateClubMember(clubUser))
        .Returns(4);
```

PROFINIT

8

Integrace na REST API

HTTP GET v C#

```
try
{
    string url = "http://vyuka.profinit.eu/club/airplane";
    WebClient webClient = new WebClient();
    string response = webClient.DownloadString(url);

    // convert from string to Json

    // convert from Json to object
}
catch (WebException exception)
{
    System.Console.WriteLine(exception);
    // Handle Exception
}
```

PROFINIT

RestSharp

- Usnadnění volání REST služeb
- Způsob použití:

```
using RestSharp;

var client = new RestClient("https://api.twitter.com/1.1");

var request = new RestRequest("statuses/home_timeline.json",
DataFormat.Json);

var response = client.Get(request);
```

- RestSharp umí i generickou variantu s automatickou deserializací:

```
var stucturedData = client.Get<SuperClass>(request).Data;
```

PROFINIT

Úkol 5: Napojení na externí službu

- V InjectConfiguration změntě injektování z ClubUserDatabaseStub na ClubUserDatabase a otevřete si aplikaci na obrazovce pro zadání letu. Co se stane?
- Seznam členů klubu se načítá přes REST službu. Zkuste nejprve získat seznam v prohlížeči:
  - http://vyuka.profinit.eu:8080/club/user
- Ve třídě ClubUserDatabase postupujte podle pokynů a naimplementujte načtení dat z REST služby
- Vyzkoušejte zadat nový let s výběrem členů klubu

PROFINIT

Nápověda 1: Načtení URL z appsettings.json

- Dependency injection přes konstruktor

```
private readonly IConfiguration configuration;

public ClubUserDatabase(IConfiguration configuration)
{
        this.configuration = configuration;
}


…

string baseUrl = configuration["ClubUsersApi"];
```

PROFINIT

Nápověda 2: Request pomocí RestSharp

- Request
- URL
- Návratový typ

```
private List<ClubUser> ReceiveClubUsers()
{
    var client = new RestClient(configuration["ClubUsersApi"]);
    var request = new RestRequest("club/user", DataFormat.Json);

    var response = client.Get<List<ClubUser>>(request);
    return response.Data;
}
```

PROFINIT

Závěr

Aplikace Flight Log

- Funkce
  - Přehled letů ve vzduchu, zadání času přistání
  - Zadání letu
  - Historie letů
  - Export do CSV

- Integrace na REST API pilotů

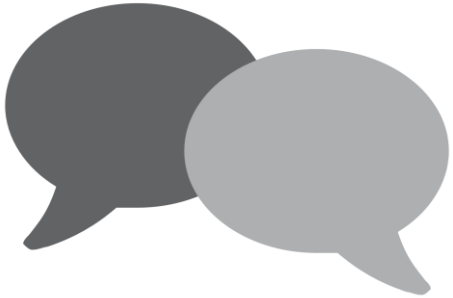
PROFINIT

Úkol

- Specifikace
  - Odevzdání do 19.4. EOD

- Aplikace musí odpovídat specifikaci
  - Napsat specifikaci tak, aby jí aplikace splňovala
  - Vymezit se vůči původní nabídce, která měla větší scope

PROFINIT



Diskuze

Děkujeme
za pozornost

PROFINIT
NÁSKOK DÍKY ZNALOSTEM

Profinit EU, s.r.o.
Tychonova 2, 160 00 Praha 6 | Telefon + 420 224 316 016

Web
www.profinit.eu

LinkedIn
linkedin.com/company/profinit

Twitter
twitter.com/Profinit_EU

Facebook
facebook.com/Profinit.EU

Youtube
Profinit EU