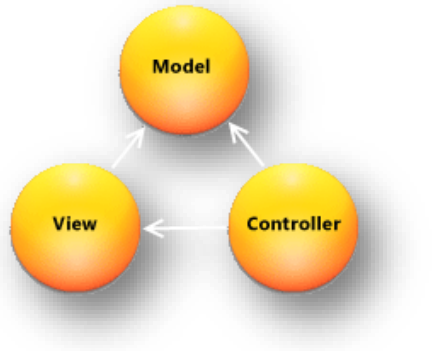
Тема 13

Изпитна тема: ИНТЕРНЕТ ПРОГРАМИРАНЕ

1 точка. Обяснява и представя графично MVC модела.

- Model–view–controller (MVC) е шаблон за архитектура
- Позволява преизползване и разделение на кода
- Първоначално е разработен за настолни приложения, но после идеята е прехвърлена и в уеб приложенията



Модел

- Модела в MVC представлява:
    - Сет който представлява данните, с които работим
    - Правила за това как да манипулираме данните
    - Може да съдържа валидация на данните
    - Често данните са капсулирани
    - Прилича на Data Access Layer
    - Няма значение в рамката, идеята е да предоставя обекти с данни

Изглед

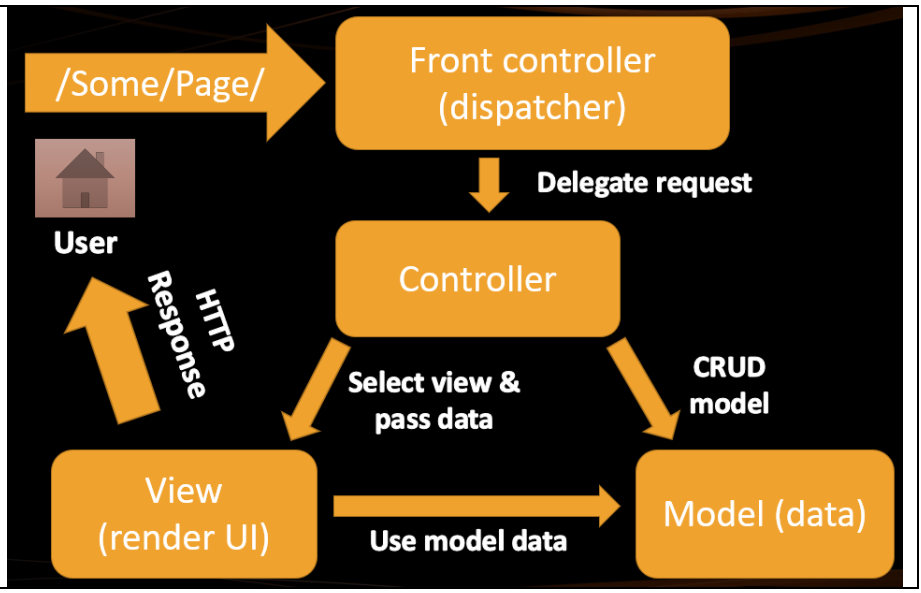
- Изглед в **MVC**:
    - Определя как ще се показва потребителският интерфейс на приложението (UI)
    - Може да поддържа главни изгледи (оформления) и под-изгледи (частични изгледи или контроли)
    - При уеб приложения предоставя шаблон за динамично генериране на HTML

Контролер

- Контролер в **MVC** представлява:
    - Основният MVC компонент
    - Обработват се заявките с помощта на изгледи и модели
    - Набор от класове, който се грижат за
        - Комуникация от потребителя
        - Общ поток на приложение
        - Логика на приложението

- Всеки контролер има едно или повече "действия"

- Входяща заявка се пренасочва към контролер
  - За уеб: HTTP Заявка
- Контролерът обработва заявка и създава презентационен модел
  - Controller също избира подходящ резултат (например: View)
- Моделът се предава на изгледа
- Изгледът преобразува Модела в подходящ изходен формат (HTML)
- **Отговорът е предоставен (HTTP отговор**)



2 точка. Описва същността и демонстрира употребата на ORM технологиите.

Обектно-релационно картографиране или ORM (Object–relational mapping), това е техника за програмиране, която служи за трансформиране на данни, използвани в обектно-ориентирания програмен език или програма и релационната база данни (тип SQL) като персистенция.

- Стандартната рамка за ORM(Object–relational mapping) за .NET и .NET Core е **Entity Framework Core.**

Entity Framework Core

- Предоставя LINQ-базирани заявки за данни и CRUD операции
- Автоматично проследяване на промяната на обекти в паметта
- Работи с много релационни бази данни (с различни доставчици)

Основен Работен Процес

1. Определете модела на данни (Code First / Scaffold from DB)
2. Писане и изпълняване на заявки върху IQueryable
3. EF генерира и изпълнява SQL заявка в БД
4. EF преобразува резултатите от заявката в .NET обекти
5. Промяна на данните със C#
6. EF генерира и изпълнява SQL команда за промяна на БД

За да добавите поддръжка на EF Core към проект във Visual Studio:

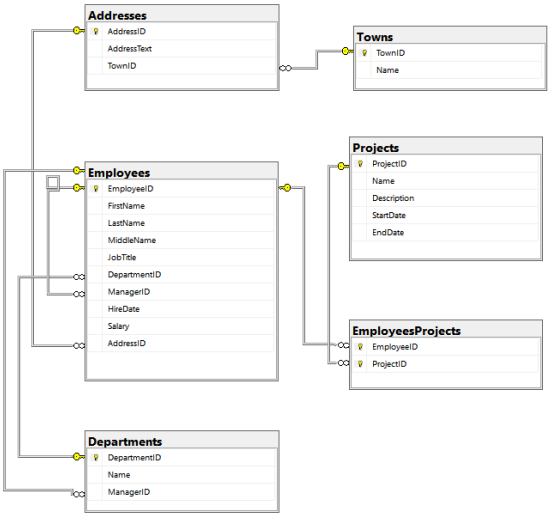
1. Инсталирайте го от **Package Manager Console**

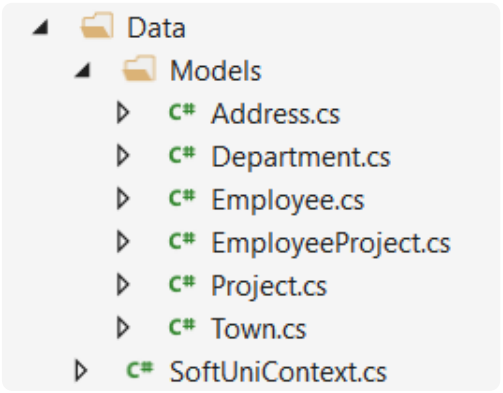
```
Install-Package Microsoft.EntityFrameworkCore
```

2. EF Core е модулен - различни допълнителни пакети могат да бъдат инсталирани:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Моделът "**Database First**" моделира класовете субекта след като базата данни е създадена.





EF Компоненти

- Класът DbContext:
  - Съдържа връзката към базата данни и преобразуваните класове
  - Осигурява достъп до данни, базиран на LINQ
  - Осигурява проследяване на идентичността, проследяване на промените и API за CRUD операции
- Entity classes
  - Всяка таблица от база данни се свежда до C# клас

- Асоциации (връзки между таблиците)
  - Асоциацията е базирана на първичен ключ / чужд ключ между два класа
  - Позволява навигация от един обкет към друг

Четене на Данни - Заявки към БД с помощта на EF Core

Класът DbContext

- DbContext предоставя:
  - CRUD Операции
    - Начин за достъпване на записите
    - Метод за добавяне на нови записи (методът Add())
    - Възможност за манипулиране на данни от база данни чрез промяна на обекти
- Изпълнение на LINQ заявки като SQL заявки
- Управление на база данни създаване/изтриване/миграция

Пример:

Използване на Класът DbContext

- Първо създайте инстанци на класа DbContext:

```
var context = new SoftUniEntities()
```

- В конструктора може да бъде подаден низ за свързване към БД

Свойствата на класа DbContext

- Database - Създава екземпляр на база данни за този контекст, който позволява проверки за създаване/изтриване/съществуване за основната база данни.
- ChangeTracker - Предоставя достъп до функции на контекста, които се занимават с проследяване на промените на обекти.
- Всички таблици са изредени като свойства в следния формат:

```
DbSet<Employee> Employees { get; set; }
```

Четене на данни с LINQ заявки:

1. Изпълненние на LINQ заявка:

```
using (var context = new SoftUniEntities())
{
 var employees = context.Employees
   .Where(e => e.JobTitle == "Design Engineer")
   .ToArray();
}
```

Entity Framework превежда това в SQL заявка

2. Employees е свойство към класа DbContext:

```
public partial class SoftUniEntities : DbContext
{
 public DbSet<Employee> Employees { get; set; }
 public DbSet<Project> Projects { get; set; }
```

```
  public DbSet<Department> Departments { get; set; }
}
```

3. Може да се използват и extension методи в заявката

```
using (var context = new SoftUniEntities())
  var employees = context.Employees
     .Where(c => c.JobTitle == "Design Engineering")
     .Select(c => c.FirstName)
     .ToList();
```

4. Намиране на запис по ID

```
using (var context = new SoftUniEntities())
{
  var project = context.Projects.Find(2);
  Console.WriteLine(project.Name);
}
```

3 точка: Реализира CRUD операции.

Когато изграждаме различни приложения, искаме моделите които прилагаме, без значение от техните различия да осигуряват наличието на четири основни функционалности. Всеки модел трябва да може да създава, прочита, обновява и изтрива ресурси. Съответно тези четири функционалности се наричат "Create, read, update и Delete" (*за по-кратко "CRUD", абревиатура от първите букви на четирите думи*) и те са основни при устойчивото съхранение на данни. CRUD парадигмата е обичайно явление при създаването на уеб приложения, защото осигурява начин, по който да напомни на разработчиците, как да конструират пълноценни и полезни използваеми модели.

CRUD в контекста на бази данни. В този случай CRUD функциите обхващат всички основни функции, които трябва да са имплементирани в приложенията, базирани на релационна база данни. Всяка една от буквите на акронима е свързана със Structured Query Language (SQL) изявление - "C" с "Insert", "R" със "Select", "U" с "Update" и "D" с "Delete".

CRUD в контекста на приложение с Entity Framework.

Изпълнението на CRUD функционалностите с EF се осъществява в DbContext. За да създадете нов ред в БД се използва метода Add(...) на съответния DbSet.

DbContext позволява промяна на свойствата на обект и запазване на промяната в базата данни.

- Просто заредете записа, променете го и извикайте SaveChanges()

```
var project = new Project()
{
  Name = "Judge System",
  StartDate = new DateTime(2015, 4, 15),
};
context.Projects.Add(project);
context.SaveChanges();
```

Изтриването се извършва чрез Remove() върху зададена колекция.

- Методът SaveChanges() извършва изтриване в базата данни

```
Employees employee =softUniEntities.Employees.First();

softUniEntities.Employees.Remove(employee);

softUniEntities.SaveChanges();
```

4 точка: Дефинира и създава шаблонен изглед.

Основни изгледи. - **View Engine Essentials**

- Изгледите в ASP.NET Core MVC използват Razor View Engine, за да вграждат .NET код в HTML маркиране.
- Обикновено те съдържат минимална логика, свързана само с представянето на данни
- Данните могат да бъдат предадени на изглед с помощта на ViewData, ViewBag или чрез ViewModel (силно типизиран изглед).

Returning Views - Класът **Base Controller** осигурява много функционалност

- Метод View () - Един от най-често използваните членове на клас Controller

Предаване на данни към изглед

- С ViewBag (динамичен тип):
  - Действие: ViewBag.Message = "Hello World!";
  - Изглед: @ViewBag.Message
- С ViewData (dictionary)
  - Действие : ViewData["message"] = "Hello World!";
  - Изглед : @ViewData["message"]
- Със силно въведени изгледи:

- Действие : return View(model);
- Изглед : @model ModelDataType;

Пример:

Контролер

```
public IActionResult Index()
{
    var user = new UserModel
    {
        Username = "TestUser",
        FullName = "Test User",
        Age  = 24
    };

    return View(user);
}
```

Модел на изгледа

```
public class UserModel
{
    public string Username { get; set; }

    public string FullName { get; set; }

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

Контролер

```
public IActionResult Index()
{
    var user = new UserModel
    {
        Username = "TestUser",
        FullName = "Test User",
        Age  = 24
    };

    return View(user);
}
```

5 точка: Различава автентикацията от авторизацията

- **Автентикация**
  - Процесът на проверка на самоличността на потребител или компютър
  - Въпроси: Кой си ти? Как го доказваш?
  - Поверителните данни могат да бъдат парола, смарт карта, външен маркер и т.н.
- **Авторизация**
  - Процесът на определяне на това, което на потребителя е разрешено да прави на компютър или мрежа
  - Въпроси: Какво можете да правите? Можете ли да видите тази страница?

6 точка: Диференцира сесиите от бисквитките.

Бисквитките са малък файл с обикновен текст без изпълним код, този файл се изпраща от сървъра до браузъра на клиента. Съхранява се от браузъра на устройството на клиента, като съхранява малка част данни за конкретен клиент и уеб сайт.

Бисквитките се използват за:

- Управление на сесиите – като вход, колички за пазаруване, резултати от игри или нещо друго, което сървърът трябва да запомни
- Персонализация – като потребителски предпочитания, теми и персонализирани настройки
- Проследяване – извършва се записване и анализ на поведението на потребителя

При управление на сесиите HTTP обекта е без състояние, което значи, че не съхранява информация за заявката(GET, POST).

Сесиите са начин за съхраняване на информация за потребител. Механизмът за обмен се използва между потребителя и уеб приложението.

При първо влизане на потребителя в уеб приложението, се изискват поверителни данни на потребителя, като уеб приложението изпраща **ID на сесията** към потребителя. При преглед на страницата , заедно с **ID на сесията** се изпраща на потребителя и изискваните данни. Тъй като са е съхранила информация за потребителя, при преглеждане на страницата след рестарт на сървъра, изискваните данни са вече налични.

7 точка: Обяснява и дава пример за различните уязвимости в сигурността на уеб приложенията. Прави изводи за предотвратяването им.

Видове уязвимости по категории:

Категория	**Атаки**
Валидиране на входа	Преливане на буфер, скриптове, SQL инжекция, канонизация
Подправяне на параметри	Манипулиране на низове за заявки, манипулация на полето на формуляра, манипулиране на бисквитки, манипулация на HTTP хедъри
Управление на сесиите	Откradване на сесия, session replay, man-in-the-middle
Криптография	Лошо генериране на ключове или управление на ключове, слабо или персонализирано криптиране
Чувствителна информация	Достъп до чувствителен код или данни в хранилището, подслушване на мрежата, подправяне на код / данни, администраторска парола
Управление на изключенията	Разкриване на информация, отказ на услугата

Пример: Cross-site scripting (XSS) е често срещана уязвимост в уеб приложенията, като уеб приложенията показват JavaScript код. Този код може да се изпълнява в браузъра на клиента, като чрез него хакерите могат да поемат контрол над сесиите, бисквитките, паролите и др.

За предотвратяване на от XSS е необходимо да проверявате потребителския вход (вградено в ASP.NET Core), също така да се изпълнява HTML escaping при показване на текстови данни.

Пример: Cross-Site Request Forgery - това е атака на уеб сигурност над HTTP протокола:

- Позволява изпълнението на неоторизирани команди от името на някой потребител
- Потребителят има валидни разрешения за изпълнение на заявената команда

- Нападателят използва тези разрешения злонамерено.

Винаги проверявайте данните от страна на сървъра при този вид атаки.

Други заплахи за сигурността:

- Man in the Middle (Винаги ползвайте SSL)
- DoS и DDoS и Brute Force attacks (CAPTCHA и Firewall)
- Пропуски в сигурността на други софтуери
  (Използвайте последните версии)

8 точка: Обяснява основните принципи на REST API.

- API е интерфейс за програмиране на приложения
  - Използван от Web Browser (SPA), Mobile Applications, Games, Desktop Applications, Web Server
- Състои се от публично изложени крайни точки
  - Крайните точки съответстват на дефинирана система за заявка-отговор
  - Комуникацията обикновено се изразява във формат JSON или XML
  - Комуникацията обикновено се осъществява чрез уеб протокол
    - Най-често HTTP - чрез уеб сървър, базиран на HTTP

ASP.NET Core Web API

- Няма нищо различно от уеб приложение
- Вие изграждате контролери с действия
- В този случай обаче действията са в ролята на крайни точки
- Контролерите трябва да се анотират с ApiController и Route

```
[Route("api/[controller]")]
[ApiController]
public class ProductsController :
ControllerBase
{
   ...
}
```

Път, използван за достъп до крайни точки от този ApiController

- Анотацията [ApiController] предоставя удобни функции
  - Автоматични HTTP 400 отговор (за грешки в състоянието на модела)
  - Обвързване на изходния параметър на източника
  - Изисквания за Атрибутно рутиране
  - Подробни отговори за кодове за състояние на грешка

```
{
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.4",
    title: "Not Found",
    status: 404,
    traceId: "0HLHLV31KRN83:00000001"
}
```

Задача 1: Предоставеното приложение зарежда GitHub хранилище с AJAX, поправете грешките в кода, така че да получите работеща програма. Обяснете кода за използването на AJAX в приложението. Модифицирайте задачата като използвате fetch() метода.

Програмен фрагмент:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Load GitHub Repos</title>
   <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<body>
GitHub username:
<input type="text" id="username" value="testnakov" />
<button onclick="FuncRepos()">Load Repos</button>
<ul id="repos"></ul>
<script src=" "></script>
</body>
</html>
```

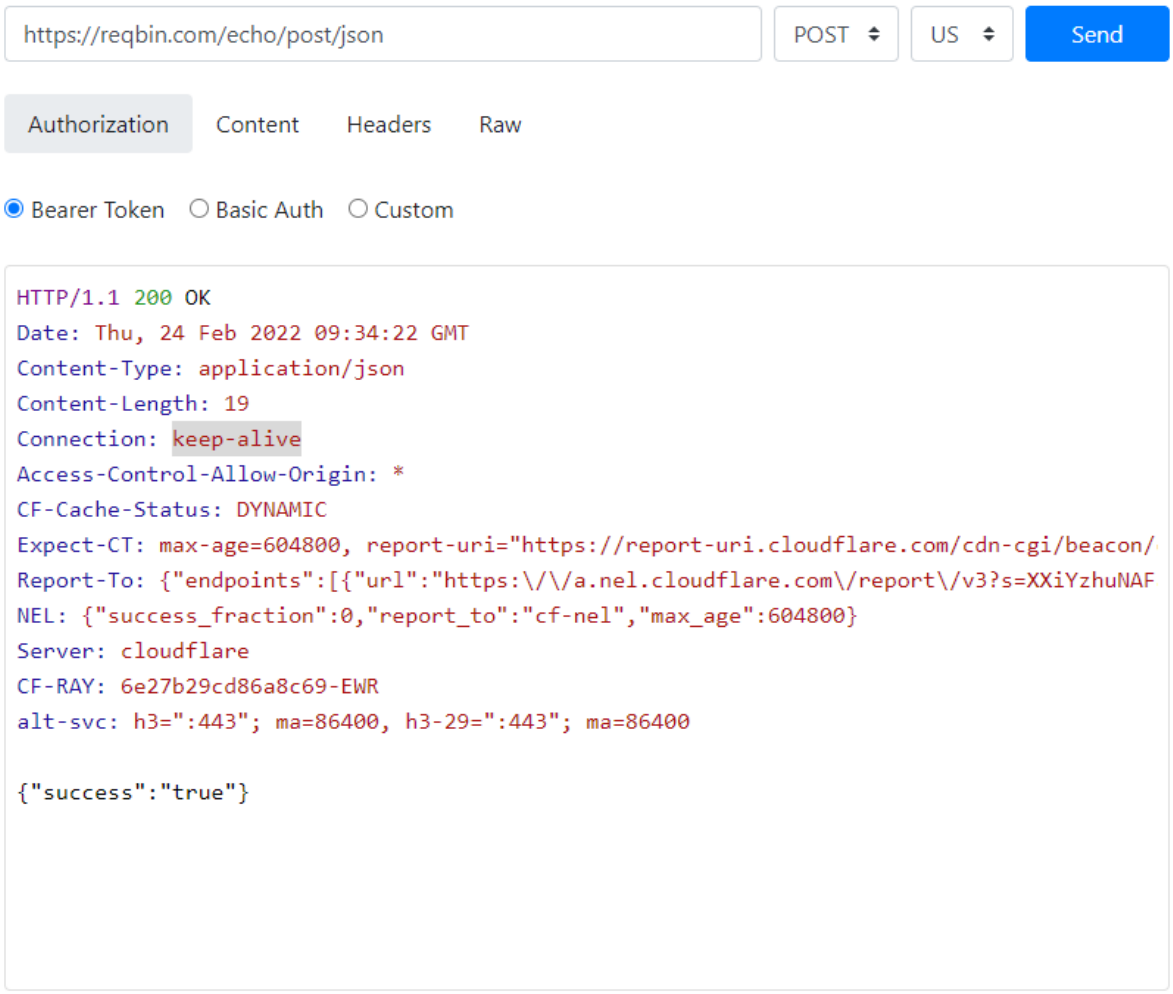
```
function loadRepos() {
   $('#repos').empty();
   let url = "https://api.github.com/users/" + $("#username").val() + "/repos";
   return $.ajax({
      url: url,
      success: displayRepos,
      error: displayError
   });

   function displayRepos(repos) {
      for(let repo of repos){
         let link = $('<a>').text(repo.full_name).attr('href', repo.html_url);
```

```
        $('#repos').append($('<li>').append(link));
      }
    }

    function displayError(err) {
      $('#repos').append($('<li>').text('Error'));
    }
}
```

Задача 2: Обяснете кода за работа с REST API в JSON формат.



```
var url = "https://reqbin.com/echo/post/json";

var httpRequest = (HttpWebRequest)WebRequest.Create(url);
httpRequest.Method = "POST";

httpRequest.Accept = "application/json";
httpRequest.ContentType = "application/json";

var data = @"{
  ""Id"": 78912,
  ""Customer"": ""Jason Sweet"",
  ""Quantity"": 1,
  ""Price"": 18.00
}";

using (var streamWriter = new StreamWriter(httpRequest.GetRequestStream()))
{
   streamWriter.Write(data);
}

var httpResponse = (HttpWebResponse)httpRequest.GetResponse();
using (var streamReader = new StreamReader(httpResponse.GetResponseStream()))
{
   var result = streamReader.ReadToEnd();
}

Console.WriteLine(httpResponse.StatusCode);
```