**ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО МЕХАНОЕЛЕКТРОТЕХНИКА**
**"СТОЙЧО И КИЦА МАРЧЕВИ" – ХАСКОВО**

6300 гр. Хасково, бул."Съединение" 46, тел.: 038 622331, e-mail: pgmet_haskovo@abv.bg

# ДИПЛОМЕН ПРОЕКТ

професия код 481030 "Приложен програмист"
специалност код 4810301 "Приложно програмиране"

Тема:

## РАЗРАБОТКА НА МНОГОСЛОЙНО УЕБ ПРИЛОЖЕНИЕ ЗА КИНОЦЕНТЪР

Изготвил:

Ваня Тодорова Ванева

Ръководител: инж. Наталия Пенева

2024 - гр. Хасково

# 1 Увод

В настоящата епоха, пронизана от бързо темпо и постоянна свързаност, преживяването на кинематографични произведения е станало неизменна част от ежедневието ни. В същото време, се усеща нарастващата необходимост от удобство и бързина в решаването на различни аспекти от нашия живот. В този контекст, представям ви **eTickets**, платформа, обогатяваща преживяването ви в света на киното.

**eTickets** е иновативен онлайн портал, който ви предоставя лесен и бърз достъп до билети за филми от различни кина. Съчетавайки съвременни технологии и интуитивен дизайн, моят уебсайт предлага персонализирано потребителско преживяване, което отговаря на вашите индивидуални предпочитания.

Чрез **eTickets** можете да разглеждате актуалната програма на кината във вашия град и да резервирате билети с няколко клика. Моята платформа предоставя детайлна информация за филмите, включително актьорите, кината и продуцентите, за да направи вашето решение по-лесно и информирано.

С **eTickets** вие не само купувате билети за филми, но и влизате в свят на удобство, където преживяването на киното е лесно достъпно и приятно. Съчетавайки страстта към филмите и технологичните възможности, моят уебсайт предоставя новаторски решения за феновете на това изкуство.

**eTickets** - вашата врата към безгранични кинематографични възможности. Добре дошли в бъдещето на онлайн купуване на билети за филми!

# 2 Основни цели и задачи на дипломния проект.

Основната задача на настоящия дипломен проект е създаването на уеб приложение за киноцентър, предлагащ на клиентите си каталог с актуалните филми във всяко от кината си и възможност за закупуване на билети за тях. В резултат на това фирмата ще има възможност да предоставя на клиентите си актуална и изчерпателна информация за прожектираните филми, достъпна по всяко време и за всички потребители.

Софтуерният продукт трябва да предлага възможност за разглеждане на филмите в различните обекти на фирмата (различните киносалони) разположени на територията на цялата страна и да съдържа информация за всеки филм (име, категория, актьори, продуцент, цена на билет, снимка и др.).

Клиентската част трябва да разполага със следните публично достъпни секции:

Начало (списък с филми); детайли на филм; списък с продуценти; детайли на продуцент; списък с актьори; детайли на актьор; списък с кина; детайли на кино; регистрация на акаунт; вход в акаунт; списък на поръчки, поръчани от този потребител; кошница с продукти; завършена покупка.

Дизайна на сайта трябва да е адаптивен и да изглежда еднакво добре на различни устройства.

Потребители на приложението ще бъдат всички клиенти на киноцентъра, като служители на фирмата ще имат възможност да актуализира информацията, използвайки администраторския панел.

За да се постигнат целите и задачите на настоящия дипломен проект е необходимо да се изследват и анализират основните аспекти на:

- ❖ Мрежови протоколи
- ❖ Клиент-сървър комуникация
- ❖ HTTP и видове заявки
- ❖ HTML и основни тагове
- ❖ CSS, селектори и основни правила в CSS
- ❖ Семантични страници
- ❖ Адаптивно (responsive) оформление на страници
- ❖ Сървърни езици
- ❖ Бази данни и СУРБД
- ❖ UML и E/R диаграми.
- ❖ Многослойна архитектура
- ❖ Трислойни модели
- ❖ Обектно-релационно свързване (ORM frameworks)
- ❖ Упълномощаване (authentication) и удостоверяване (authorization)

# 3 Теория

## 3.1 HTTP и HTTPS

HTTP (HyperText Transfer Protocol) и HTTPS (HyperText Transfer Protocol Secure) са протоколи, които се ползват комуникация между клиент (обикновено уеб браузър) и сървър през интернет. Те определят как съобщенията са форматирани, предавани и как уеб сървърите и браузърите трябва да реагират на различни команди.

### 3.1.1 HTTP

**Hypertext Transfer Protocol:**

- **Употреба:** HTTP е в основата на комуникацията на данни в World Wide Web. Той се използва за прехвърляне на хипертекст (текст с връзки) и други мултимедийни файлове, като изображения, видеоклипове и скриптове, между уеб сървъри и браузъри.
- **Функционалност:** Той работи като протокол за заявка-отговор. Клиентът (обикновено уеб браузър) изпраща HTTP заявка до сървъра, а сървърът отговаря с искания ресурс или със съобщение за грешка.

### 3.1.2 HTTPS

**Hypertext Transfer Protocol Secure:**

- **Употреба:** HTTPS е защитената версия на HTTP. Тя е предназначена да осигури сигурна връзка чрез криптиране на данните, обменяни между клиента и сървъра.
- **Функционалност:** HTTPS използва TLS (Transport Layer Security) или неговия предшественик SSL (Secure Sockets Layer), за да криптира данните, предавани между клиента и сървъра. Това криптиране спомага за защита на поверителна информация, като данни за вход и лични данни, от неразрешен достъп или прихващане от трети страни.

### 3.1.3 Разлики

- **Защита:** Основната разлика между HTTP и HTTPS е нивото на сигурност. Докато HTTP изпраща данни в обикновен текст, HTTPS криптира данните, което ги прави

по-сигурни.

- **Порт:** HTTP обикновено използва порт 80 за комуникация, докато HTTPS използва порт 443.
- **Протокол:** HTTP и HTTPS имат един и същ основен протокол за комуникация, но допълнителните функции за сигурност правят HTTPS по-сигурен избор, особено за уебсайтове, които обработват чувствителна информация.

## 3.2 Клиент-сървър комуникация

### 3.2.1 Клиент-сървър

Терминът "клиент/сървър" описва взаимоотношенията между две компютърни програми, от които едната програма – клиент, прави заявка за услуга към другата програма - сървър, който изпълнява заявката

### 3.2.2 Клиент

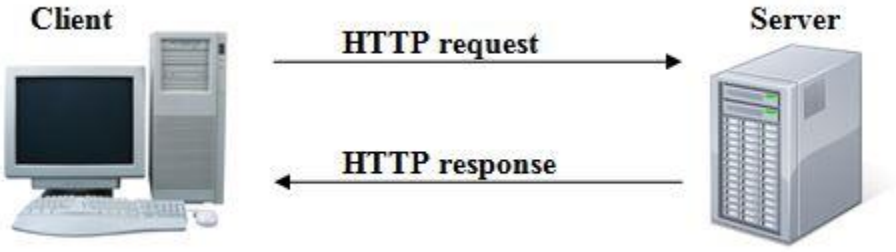
Клиентът е програмата, очакваща да получи услуга от друга програма. Взаимодейства с потребителя чрез клавиатурата, дисплея или друго входно/изходно устройство. Клиентът няма директни отговорности към достъпа до данни. Той само изпраща заявки до сървъра.

### 3.2.3 Сървър

Сървърът е системата, която предоставя набор от услуги на клиента. Той има за задача да приема и обработва клиентските заявки и да връща отговора обратно.

### 3.2.4 Комуникация между клиента и сървъра

Комуникацията между клиент и сървър е процесът на обмен на информация между двата компонента посредством HTTP протокола. Това е основният протокол за изпращане на заявки и получаване на отговори в интернет.




**Връзката между клиента и сървъра**

Клиентът и сървърът могат да са на един и същ компютър или на различни компютри, свързани в мрежа. Мрежата прави възможна отдалечената клиент/сървър комуникация.

**Комуникацията между клиент и сървър се извършва в следния ред:**

#### 3.2.4.1 Изпращане на заявка

- Клиентът генерира и изпраща заявка към сървъра.
- Заявката съдържа необходимата информация, като например типа на услугата, която се иска, и параметрите за изпълнение.

#### 3.2.4.2 Обработка на заявката

- Сървърът приема заявката и я обработва.
- Обработката може да включва достъп до база данни, изпълнение на определени операции или други дейности.

#### 3.2.4.3 Генериране на отговор

- Сървърът генерира отговор, който съдържа резултатите от обработката на заявката.
- Отговорът също така може да включва код на състоянието/код със статус(статусен код), който указва дали заявката е била успешна или има грешка.

#### 3.2.4.4 Изпращане на отговор

- Сървърът изпраща отговора към клиента.
- Отговорът съдържа необходимата информация, която клиентът трябва да обработи.

## 3.3 HTTP requests

### 3.3.1 HTTP request

В HTTP, заявките (requests) са начинът, по който браузърът или клиентската програма взаимодейства с уеб сървъра, за да изпраща или получава данни.

**Една правилна съставена/формулирана HTTP заявка съдържа следните елементи**:

1. Ред за заявка.
2. Поредица от HTTP заглавия или заглавни полета.
3. Съдържание/тяло на съобщението, ако е необходимо.

### 3.3.2 Request Line

HTTP request line е част от HTTP заявката и съдържа основната информация за това каква операция (метод) се извършва и към кой ресурс се отправя заявката.

Състои се от поне три елемента:

1. Метод. Методът е команда от една дума, която казва на сървъра какво трябва да направи с ресурса Например, от сървъра може да бъде поискано да се изпрати ресурса на клиента.
2. Компонентът на маршрута/пътя в URL адреа на заявката. Маршрута/пътя идентифицира ресурса на сървъра.
3. Номерът на версията на HTTP, показващ спецификацията на HTTP, на която клиентът се е опитал да придаде съответствие на съобщението.

Пример

Получаване на информация за времето. Клиентът (напр. приложение за времето) изпраща заявка до сървъра за текущото време на определено място. Сървърът обработва заявката, извлича съответните данни за времето и ги изпраща обратно на клиента. След това клиентът показва на потребителя получената информация за времето.

Редът на заявката може да съдържа някои допълнителни елементи:

- Заявка тип низ (string). Тя предоставя низ от информация, която ресурсът може да използва за дадена цел. Тя следва път/маршрут и е предшествана от въпросителен знак.

### 3.3.3 Request Methods

Протоколът HTTP (Hypertext Transfer Protocol - Протокол за трансфер на хипертекст) определя набор от методи за заявка, с които се указва какво действие трябва да се извърши с даден ресурс. Най-често използваните методи за заявки по HTTP са GET, POST, PUT, PATCH и DELETE. Те са еквивалентни на операциите CRUD (създаване, четене, актуализиране и изтриване).

#### 3.3.3.1 GET

**GET (Получаване на ресурси):**

- Основно предназначение: Заявката GET се използва за извличане на информация от уеб сървъра .
- Данни в заявката: Данните се добавят към URL-то (query string).
- Сигурност: GET заявките са обикновено по-леки и се използват за четене на информация, като например четене на HTML страници, изображения или други ресурси.

#### 3.3.3.2 HEAD

**HEAD (Получаване на заглавна част):**

- Основно предназначение: HEAD заявката е подобна на GET, но с една съществена разлика: сървърът отговаря само с заглавната част на HTTP отговора, без да предоставя съдържанието на ресурса.
- Данни в заявката: Както и при GET, данните се добавят към URL-то.
- Сигурност: Използва се, когато клиентът иска да получи само заглавната част на отговора, без фактическото съдържание. Това може да бъде полезно, когато клиентът иска да провери заглавната част, преди да изтегли цялото съдържание.

#### 3.3.3.3 POST

**POST (Създаване на ресурси или изпращане на данни към сървъра):**

- Основно предназначение: POST се използва, когато искате да изпратите данни към сървъра, обикновено за създаване на нов ресурс.
- Данни в заявката: Данните се предават в тялото на заявката и не са видими в URL-то.
- Сигурност: POST заявките се използват, когато искате да изпратите чувствителна информация и обикновено се използват за създаване на ресурси, като например когато изпращате формуляри.

#### 3.3.3.4 PUT

**PUT (Актуализиране на ресурси или създаване, ако не съществуват):**

- Основно предназначение: Използва се, за да изпратите данни, които ще заменят съдържанието на сървъра или да създадат нов ресурс, ако не съществува.
- Данни в заявката: Данните се предават в тялото на заявката.
- Сигурност: Подобно на POST, използва се, когато се изпращат данни за актуализация на ресурси.

#### 3.3.3.5 PATCH

**PATCH (Частична актуализация на ресурси):**

- Основно предназначение: Използва се за частична актуализация на ресурс, като се предоставят само промените.
- Данни в заявката: Данните за актуализация се предават в тялото на заявката.
- Сигурност: Подобно на PUT, но с по-фин контрол върху това, кои части от ресурса се променят.

#### 3.3.3.6 DELETE

**DELETE (Изтриване на ресурси):**

- Основно предназначение: Използва се за изтриване на ресурс от сървъра.
- Данни в заявката: Обикновено не се предават данни в тялото на DELETE заявката.
- Сигурност: Използва се, когато искате да изтриете ресурс от сървъра.

## 3.4 HTTP Response

Сървърът изпраща HTTP отговор на клиента. Целта на отговора е да предостави на клиента ресурса, който е поискал, или да го информира, че действието, което е поискал, е извършено; или да го информира, че при обработката на заявката му е възникнала грешка.

HTTP отговор/отзив съдържа:

1. Ред за състоянието.
2. Поредица от HTTP заглавия или заглавни полета.
3. Тяло на съобщението, което обикновено е необходимо.

### 3.4.1 Status Line

Линията за статуса е първата линия в съобщението за отговор. Тя се състои от три елемента:

- Номерът на версията на HTTP, показващ спецификацията на HTTP, на която сървърът се е опитал да придаде съответствие на съобщението.
- Код на състоянието, което е трицифрено число, показващо резултата от заявката.
- Фраза за причината, известна също като текст на състоянието, която представлява текст, който може да бъде прочетен от човек и който обобщава значението на кода на състоянието.

### 3.4.2 HTTP headers

HTTP заглавията на отговора на сървъра съдържат информация, която клиентът може да използва, за да разбере повече за отговора и за сървъра, който го е изпратил. Тази информация може да помогне на клиента при показването на отговора на потребителя, при съхраняването (или кеширането) на отговора за бъдеща употреба и при извършването на допълнителни заявки към сървъра сега или в бъдеще.

### 3.4.3 Message Body

Тялото на съобщението на отговора може да се нарича за удобство "тяло на отговора". Телата на съобщенията се използват за повечето отговори. Изключение

правят случаите, когато сървърът отговаря на клиентска заявка, използваща метода HEAD.

## 3.5 HTML и основни тагове

HTML е съкращение за "Hypertext Markup Language" (Хипертекстов Език за Отбелязване). Това е стандартен език за маркиране, използван за създаване и структуриране на уеб страници и уеб приложения. HTML използва тагове за дефиниране на различни елементи

### 3.5.1 Основни HTML тагове

**<!DOCTYPE>** - Дефинира типа на документа.

**<html>** - Дефинира html документ.

**<head>** - Съдържа метаданни/информация за документа.

**<title>** - Дефинира заглавие на документа.

**<body>** - Дефинира съдържанието на документа.

**<h1>** до **<h6>** - Дефинира заглавия в HTML.

**<p>** - Дефинира параграф.

**<br>** - Вмъква единично прекъсване на реда.

**<hr>** - Дефинира тематична промяна в съдържанието.

**<!--…-->** - Дефинира коментар.

### 3.5.2 Тагове за форматиране

**<b>** - Дефинира удебелен текст.

**** - Дефинира подчертан текст.

**<mark>** -Дефинира маркиран/подчертан текст.

**** - Дефинира важен текст.

**<sub>** - Дефинира подписан текст.

**<sup>** - Дефинира текст с надпис.

### 3.5.3 Вход от потребителя - Формуляри и елементи на формуляр

**<form>** - Дефинира html формуляр за въвеждане от потребителя.

**<input>** - Дефинира поле за въвеждане.

**<textarea>** - Дефинира много редово поле за въвеждане.

**<button>** - Дефинира бутон.

**<select>** - Дефинира падащ списък.

**<option>** - Дефинира опция в падащ списък.

**<label>** - Дефинира етикет за елемент **<input>**.

**<output>** - Дефинира резултата от изчисление.

### 3.5.4 Изображения, аудио, видео и връзки

**<img>** - Дефинира изображение.

**<map>** - Дефинира карта.

**<area>** - Дефинира област.

**<audio>** - Дефинира звуково съдържание.

**<source>** - Дефинира множество медийни ресурси за медийни елементи.

**<video>** - Дефинира видео или филм.

**<a>** - Дефинира хипервръзка.

**<link>** - Дефинира връзката между документа и външен ресурс.

**<nav>** - Дефинира навигационни връзки.

### 3.5.5 Списъци, таблици, стилове и семантика

**<ul>** - Дефинира неподреден списък.

**<ol>** - Дефинира подреден списък.

**<li>** - Дефинира елемент от списък.

**<dl>** - Дефинира списък с описание.

**<dt>** - Дефинира термин/име в списък с описание.

**<dd>** - Дефинира описание на термин/име в списък с описание.

**<table>** - Дефинира таблица.

**<caption>** - Дефинира заглавие в таблица.

**<th>** - Дефинира заглавна клетка в таблица.

**<tr>** - Дефинира ред в таблица.

**<td>** - Дефинира клетка в таблица.

**<thead>** - Групира съдържанието на заглавието в таблица.

**<tbody>** - Групира основното съдържание в таблица.

**<tfoot>** - Групира съдържанието на колонтитула в таблицата.

**<col>** - Дефинира свойствата на колоните за всяка колона.

**<style>** - Дефинира информация за стил на документ.

** ** - Дефинира раздел в документ.

**** - Дефинира раздел в документ.

**<header>** - Дефинира горен колонтитул за документ или раздел.

**<footer>** - Дефинира долен колонтитул за документ или раздел.

**<main>** - Дефинира основното съдържание на документа.

**<section>** - Дефинира раздел в документ.

**<article>** - Дефинира статия.

**<script>** - Дефинира скрипт/код.

## 3.6 CSS, селектори и основни правила в CSS

CSS или Cascading Style Sheets е стилов език, използван за описание на представянето на документ, написан на HTML или XML. CSS описва как елементите трябва да бъдат изобразени на екрана, на хартия или на други медии.

Той контролира цветовете, шрифтовете и оформлението на елементите на уебсайт. CSS отговорен за това как ще изглеждат уеб страници. Този стилов език също ви позволява да добавяте ефекти или анимации към уебсайт.

Има три различни метода за стилизиране в CSS - **Inline (Local style), Internal (Embedded, Page-Level style)** и **External Styles**.

- **Външни стилове (External Style Sheet)**

Външен стилов лист е файл с **.css** разширение, което съдържа дефиниции на каскаден стилов лист (CSS) за уеб страница(и). Той напълно разделя CSS стиловете от HTML документа, което го прави лесен за повторна употреба и поддръжка.

- **Вътрешни стилове (Internal / Embedded Style Sheet)**

  Вътрешен CSS се използва за дефиниране на стил за отделна HTML страница. Вътрешен CSS е дефиниран в секцията <head> на HTML страница, в елемент <style>.

- **Inline CSS / Local Styles (Вградени в самите HTML елементи**)

  Вграден стил може да се използва за прилагане на уникален стил за отделен елемент.

  За да се използват вградени стилове, се добавя атрибута style към съответния елемент. Атрибутът style може да съдържа всяко CSS свойство.

При наличие на повече от един стил, специфициран за HTML елемент, ще се използва този, който има най-висок приоритет, както следва:

На всяко ниво на стилизиране се дава различен йерархичен приоритет (кога да се приложи) и се използва по различни причини.

1. Стиловете, вградените в самите HTML елементи.
2. Вътрешните стилове.
3. Външните стилове.

### 3.6.1 Основна структура на CSS

CSS е базиран на "блокове" от правила, които дефинират как определени елементи на уеб страницата трябва да се представят. Тези правила могат да се прилагат на конкретни елементи или класове на елементи.

CSS правилата се състоят от две основни части - селектор и блок за декларация. Селекторът определя кой HTML елемент се прилага правилото, а декларацията задава стила за съответния елемент.

Пример за CSS правило:

```
selector {
  property: value;
}
```

- **Selector:** Избира HTML елемента(ите), който искате да стилизирате.

- **Property:** Указва стила, който искате да приложите.
- **Value:** Указва стойността на свойството стил.

### 3.6.2 CSS селектори

Селекторите се използват за насочване към конкретни HTML елементи за стилизиране. Ето някои често срещани селектори:

**Селектор на елементи**

```
p {
  color: blue;
}
```

Това избира всички <p> (параграф) елементи и прави текста син.

**Селектор на ID**

```
#myId {
  font-size: 16px;
}
```

Избира елемента с посочения идентификатор.

**Селектор на клас**

```
.myClass {
  Background-color: yellow;
}
```

Избира всички елементи с посочения клас.

**Селектор на потомък**

```
div p {
  font-style: italic;
}
```

Избира всички <p> елементи, които са наследници на .

**Селектор на атрибути**

```
input [type=”text”] {
```

```
  Border: 1px solid #ccc;
}
```

Избира всички <input> елементи с type="text".

**Псевдо-клас селектор**

```
a:hover {
  text-decorator: underline;
}
```

Избира връзки, върху които се задържа курсора на мишката.

### 3.6.3 Основни CSS правила

1. **Цвят:** определя цвета на текста в HTML елемент.
2. **Шрифт:** font в CSS е общ термин, който обхваща няколко свойства, които контролират различни аспекти на шрифтовете в HTML елементите.
   - **font-family:** Указва списък от шрифтови семейства или конкретни имена на шрифтове. Браузърът ще използва първия шрифт в списъка, който е наличен на компютъра на потребителя.
   - **font-size:** Указва размера на шрифта. Може да се изрази в различни единици като пиксели, емове, проценти и други.
   - **font-weight:** Указва дебелината на шрифта. Може да приема стойности като "normal" или "bold", или числови стойности от 100 до 900.
   - **font-style:** Указва стила на шрифта, като "normal", "italic" или "oblique".
3. **Външен отстъп (Margin)** се използва за контролиране на пространството около елемента.
   - **Единична стойност**: margin: 10px; - Указва еднакви отстояния от всички страни на елемента.
   - **Двойка стойности**: margin: 10px 20px; - Първата стойност е горе и долу, втората е отляво и отдясно.
   - **Три стойности**: margin: 10px 20px 30px; - Първата стойност е горе, втората е отляво и отдясно, третата е долу.
   - **Четири стойности**: margin: 10px 20px 30px 40px; - Първата стойност е горе, втората е отляво, третата е долу, четвъртата е отдясно.

4. **Вътрешен отстъп (padding):** се използва за контролиране на пространството вътре в елемента, между границата на елемента и неговото съдържание.

   - **Стойностите са както при външния отстъп (margin)**

5. **Фон (Background)**

   - **background-color**: Указва цвета на фона. Можете да използвате ключови думи (като "red", "blue") или цветови кодове (като "#ff0000").
   - **background-image**: Указва изображението, което да се използва като фон.
   - **background-repeat**: Указва как фоновото изображение се повтаря. Възможни стойности са repeat, repeat-x, repeat-y и no-repeat.
   - **background-position**: Указва позицията на фоновото изображение.
   - **background-size**: Указва размера на фоновото изображение.
   - **background-attachment**: Указва дали фонът трябва да е фиксиран или да следва превъртането на страницата. Стойности са scroll или fixed.

6. **Граница (Border)** се използва за задаване на стил, цвят и ширина на границата на елемент.

   - **border-width**: Указва ширината на границата и може да приема стойности като thin, medium, thick, или конкретни стойности в пиксели, em, и други мерни единици.
   - **border-style**: Указва стила на границата, като например solid (плавен), dashed (пунктиран), dotted (точков), и други.
   - **border-color**: Указва цвета на границата.

7. **Текст -** text-align и text-decoration са две от свойствата в CSS, които контролират изгледа на текста в елементите на уеб страница.

   - **text-align**: Свойството text-align се използва, за да управлява подравняването на текста в елемента. Възможни стойности включват:
     - **left:** Подравнява текста отляво.
     - **right:** Подравнява текста отдясно.
     - **center:** Центрира текста.
     - **justify:** Подравнява текста и разтяга интервалите между думите, така че линиите да станат равномерни по ширина.

- **text-decoration:** Свойството text-decoration се използва за управление на декорациите на текста, като подчертаване, зачертаване и линия над текста. Възможни стойности включват:
  - **none:** Без декорации (по подразбиране).
  - **underline:** Подчертава текста.
  - **overline:** Показва линия над текста.
  - **line-through:** Показва линия през текста.
  - **blink:** Мигащ текст (най-често не се поддържа и не се препоръчва използването му).

## 3.7 Семантични страници

Една страница се счита за семантична, когато HTML кодът й използва семантични елементи съгласно техните истински значения и смисъл. Това означава, че различните части на уеб страницата са маркирани с елементи, които отразяват съдържанието и функционалността им. Използването на семантични елементи подпомага по-доброто разбиране на структурата на страницата от браузърите, търсачките и други инструменти.

### 3.7.1 Семантични HTML елементи

Семантичните HTML елементи са тези, които имат значително значение и предназначение за структурата и съдържанието на уеб страницата, а не само за стилизацията или визуалния ефект. Тези елементи помагат на браузърите и другите инструменти за разбиране на смисъла на различните части на уеб страницата, което може да подобри SEO (оптимизацията за търсачки) и достъпността. Някои от семантичните HTML елементи включват:

Ето някои от по-важните, HTML5 семантични елементи и тяхното значение по отношение на SEO оптимизацията:

**`<header> и <footer>:`** Търсачките разпознават заглавията и футерите на страниците като ключови части, които съдържат важна информация за контекста на страницата, авторството, и връзките с други страници.

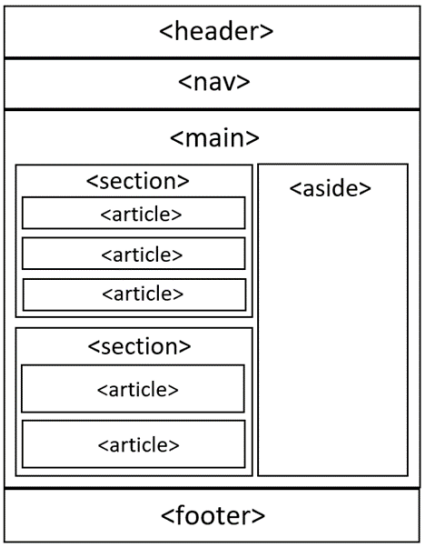
**`<nav>:`** помага на търсачките да разберат връзките между различните страници и разделите на уеб сайта, което подпомага индексирането на съдържанието.

**<article> и <main>:** Съдържанието се разпознава като основно и важно. Това подобрява видимостта на съдържанието в резултатите от търсенето, особено ако търсачката го счита за ключово.

**<section> и <aside>:** помага за структуриране на страницата и подчертаване на важни секции. Търсачките извличат значима информация от тези секции.

**<video> и <figure>:** Когато вграждате медийни елементи с тези тагове, търсачките по-лесно разпознават и индексират мултимедийното съдържание на страницата.

### 3.7.2 Примерна структура на семантична HTML страница



```
<!DOCTYPE html>
<html>
    <body>
        <header>Placeholder</header>
        <nav>Content</nav>
        <aside>
            Aside menu
        </aside>
        <main>
            <section id="courses">
                <article>
                    course 01 info content
                </article>
                <article>
                    course 02 info content
                </article>
                <article>
                    course 03 info content
                </article>
            </section>
            <section id="teachers">
                <article>
                    teacher 01 info content
                </article>
                <article>
                    teacher 02 info content
                </article>
            </section>
        </main>
        <footer>

        </footer>
    </body>
</html>
```

## 3.8 Адаптивно (responsive) оформление на страници

Адаптивен дизайн на уеб страница означава създаване на дизайн, който автоматично се приспособява към различните размери на екраните, осигурявайки удобно и качествено изглеждане на устройства с различни разделителни способности. Това е важно, за да уебсайтът ви бъде удобен за използване на компютри, таблети и мобилни устройства.

### 3.8.1 Ключови принципи на адаптивния дизайн

1. **Гъвкава мрежа (Flexible Grid):** Използвайте относителни единици за измерване, като проценти, вместо фиксирани пиксели за разполагане на елементите на страницата. Това позволява те да се мащабират в зависимост от размера на екрана.
2. **Гъвкави изображения и медийни елементи (Flexible Images and Media):** Използвайте CSS свойства като max-width: 100% за изображения, така че те да не излизат извън своите контейнери при промяна на размера на екрана.
3. **Медиазапроси (Media Queries):** Вграждайте медийно запитване в CSS стиловете, за да определите различни стилове в зависимост от параметрите на устройството, като ширината на екрана, височината и ориентацията.
4. **Относителни размери на шрифтовете (Relative Font Sizes):** Използвайте относителни единици за измерване, като em или rem, за размерите на шрифтовете, за да се мащабират заедно с другите елементи.
5. **Тестване на различни устройства:** Проверете как изглежда и се държи вашият уебсайт на различни устройства, за да се уверите, че той е отзивчив и удобен за използване.

## 3.9 JavaScript

JavaScript (JS) е лек интерпретиран (или компилиран в момента) език за програмиране с първокласни функции. Той е най-известен като скриптов език за уеб страници.

JavaScript (JS) is a lightweight interpreted (or just-in-time compiled) programming language with first-class functions. It is most well-known as the scripting language for Web pages.

https://developer.mozilla.org/en-US/docs/Web/JavaScript

JavaScript е доминиращият език за скриптиране от страна на клиента в уеб. Скриптовете се вграждат в HTML документи или се включват в тях и взаимодействат с DOM. Всички основни уеб браузъри имат вграден JavaScript енджин, който изпълнява кода на устройството на потребителя.

JavaScript is the dominant client-side scripting language of the Web. Scripts are embedded in or included from HTML documents and interact with the DOM. All major web browsers have a built-in JavaScript engine that executes the code on the user's device.

https://en.wikipedia.org/wiki/JavaScript

Моделът на обекта на документа (DOM) е междуплатформен и независим от езика интерфейс, който разглежда документа HTML като дървовидна структура, в която всеки възел е обект, представляващ част от документа. DOM представя документа с логическо дърво. Всяко разклонение на дървото завършва с възел, а всеки възел съдържа обекти.

The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects.

В HTML DOM (Document Object Model) всеки елемент е възел:

- Документът е възел на документ.
- Всички HTML елементи са елементни възли.
- Всички HTML атрибути са възли на атрибути.
- Текстът, вмъкнат в HTML елементи, е текстов възел.
- Коментарите са възли за коментари.

In HTML DOM (Document Object Model), every element is a node:

- A document is a document node.
- All HTML elements are element nodes.
- All HTML attributes are attribute nodes.
- Text inserted into HTML elements are text nodes.
- Comments are comment nodes.

При зареждане на уеб страница браузърът създава модел на обекта на страницата, който представлява обектно-ориентирано представяне на HTML документ, служещ като интерфейс между JavaScript и самия документ. Това позволява създаването на динамични уеб страници, тъй като в рамките на една страница JavaScript може:

- добавя, променя и премахва всички елементи и атрибути на HTML.
- да променя стиловете на CSS
- да реагира на всички съществуващи събития
- да създава нови събития

When a web page is loaded, the browser creates a Document Object Model of the page, which is an object oriented representation of an HTML document that acts as an interface between JavaScript and the document itself. This allows the creation of dynamic web pages, because within a page JavaScript can:

- add, change, and remove any of the HTML elements and attributes
- change any of the CSS styles
- react to all the existing events
- create new events

https://en.wikipedia.org/wiki/Document_Object_Model

**JavaScript type conversions**

| left operand | operator | right operand | result |
|---|---|---|---|
| `[]` (empty array) | `+` | `[]` (empty array) | `""` (empty string) |
| `[]` (empty array) | `+` | `{}` (empty object) | `"[object Object]"` (string) |
| `false` (boolean) | `+` | `[]` (empty array) | `"false"` (string) |
| `"123"` (string) | `+` | `1` (number) | `"1231"` (string) |
| `"123"` (string) | `-` | `1` (number) | `122` (number) |
| `"123"` (string) | `-` | `"abc"` (string) | `NaN` (number) |

Променливите в JavaScript могат да бъдат дефинирани с помощта на ключовите думи var, let или const. Променливите, дефинирани без ключови думи, ще бъдат дефинирани в глобалния обхват.

Variables in JavaScript can be defined using either the var, let or const keywords. Variables defined without keywords will be defined at the global scope.

```
var x;
```

Декларира променлива с обхват на функцията, наречена ‘x’, и ѝ присвоява специалната стойност ‘undefined’. Променливите без стойност автоматично се задават като undefined.

Declares a function-scoped variable named `x`, and implicitly assigns the special value `undefined` to it. Variables without value are automatically set to undefined.

```
let x2 = undefined;
```

Променливите могат да бъдат ръчно зададени като `неопределени`.

Variables can be manually set to `undefined`

```
const z = "this value cannot be reassigned!";
```

Ключовата дума `const` означава константа, следователно променливата не може да бъде повторно зададена, тъй като нейната стойност е `константа`.

The keyword `const` means constant, hence the variable cannot be reassigned as the value is `constant`.

Функции: Блокове от код за многократна употреба.

Functions: Blocks of reusable code.

```
function greet(name) {
    console.log("Hello, " + name + "!");
}
greet("Alice");
```

Условни твърдения: Като if, else if и else.

Conditional Statements: Like if, else if, and else.

```
var grade = 85;
if (grade >= 90) {
    console.log("A");
} else if (grade >= 80) {
    console.log("B");
} else {
    console.log("C");
}
```

Цикли: за повтарящи се задачи.

Loops: for repetitive tasks.

```
for (var i = 0; i < 5; i++) {
    console.log(i);
}
```

Обекти: Използват се за групиране на свързани данни и функции.

Objects: Used to group related data and functions.

```
var person = {
    name: "Alice",
    age: 30,
    greet: function() {
        console.log("Hello, " + this.name + "!");
    }
};
person.greet();
```

Минималистична програма Hello World на JavaScript в среда за изпълнение с конзолен обект:

A minimalist Hello World program in JavaScript in a runtime environment with a console object:

```
console.log("Hello, World!");
```

Елементите могат да бъдат взети задължително с querySelector за един елемент или querySelectorAll за няколко елемента, които могат да бъдат зациклени с forEach:

Elements can be imperatively grabbed with querySelector for one element, or querySelectorAll for multiple elements that can be looped with forEach.

`document.querySelector('.class');` // Избира първия елемент с клас "class"

`document.querySelector('#id');` // Избира първия елемент с идентификационне номер "id"

`document.querySelector('[data-other]');` // Избира първия елемент с атрибут "data-other"

`document.querySelectorAll('.multiple');` // Връща списък от възли, подобен на масив, на всички елементи с клас "multiple"

**document.querySelector('.class');** // Selects the first element with the "class" class

**document.querySelector('#id');** // Selects the first element with an `id` of "id"

**document.querySelector('[data-other]');** // Selects the first element with the "data-other" attribute

**document.querySelectorAll('.multiple');** // Returns an Array-like NodeList of all elements with the "multiple" class

Обработка на взаимодействията с потребителите

Handling user interactions

```
document.getElementById("myButton").addEventListener("click", function() {
    alert("Button clicked!");
});
```

// Използва се за добавяне на събитие (event listener) към HTML елемент с идентификатор "myButton", което се изпълнява, когато този елемент бъде кликнат.

## 3.10 Bootstrap

Bootstrap е безплатен front-end framework за по-бързо и лесно разработване на уеб сайтове. Включва шаблони за дизайн, базирани на HTML и CSS, за типография (typography), форми, бутони, таблици, навигация, модали (modals), въртележки (carousels) с изображения, както и допълнителни plugin-и за JavaScript. Bootstrap също така дава възможност за лесно създаване на адаптивни дизайни.

Два начина за използване на Bootstrap:

- ❖ Изтегляне на Bootstrap от getbootstrap.com.
- ❖ Включване на Bootstrap от CDN.

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
```

Пример:

```
<!DOCTYPE html>
    <html lang="en">
    <head>
        <title>Bootstrap Example</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/
bootstrap.min.css">
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jq
uery.min.js"></script>
        <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bo
otstrap.min.js"></script>
    </head>
    <body>
        <div class="container-fluid">
            <h1>My First Bootstrap Page</h1>
            <p>This is some text.</p>
        </div>
    </body>
</html>
```

**Bootstrap Grid система**

Системата позволява до 12 колони на страница. Могат да се ползват заедно колоните или поотделно:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 | span 1 |
| span 4 | | | span 4 | | | span 4 | | | | | |
| span 4 | | | span 8 | | | | | | | | |
| span 6 | | | | | | span 6 | | | | | |
| span 12 | | | | | | | | | | | |

Системата е адаптивна и колоните се подреждат в зависимост от размера на екрана

**Grid класове**

(Решетка)Grid системата за Bootstrap има четири класа:

- **xs** - за телефони с екрани < 768px ширина
- **sm** - за таблети с екрани с ≥ 768px ширина
- **md** – за малки лаптопи с екрани с ≥ 992px ширина
- **lg** – за латпопи и настолни компютри с екрани с ≥ 1200px ширина

Класовете могат да се комбинират, за да се създадат по-динамични гъвкави оформления. Всеки клас се мащабира нагоре, така че ако искате да зададете еднакви ширини за xs и sm, трчбва да посочите само xs

**Основни стилове в Bootstrap**

1. **Типография (Typography):** Определени стилове за заглавия (h1-h6). Различни размери на шрифтове. Контекстни стилове като маркирани текстове и абзаци.
2. **Цветова палитра:** Предварително дефинирани цветове за фона, текста и бутоните. Използване на контекстни цветове като "success," "warning," и "danger."
3. **Бутони (Buttons):** Стилизирани бутони с различни размери. Вградени стилове за важни действия, като например "primary" и "danger."
4. **Навигация (Navigation):** Навигационни барове със стилове за хоризонтална и вертикална навигация. Опции за създаване на навигационни менюта.
5. **Формуляри (Forms):** Стилизирани формуляри с различни полета, като текстови полета, пароли, и т.н. Помощни текстове и стилове за валидация.
6. **Списъци (Lists):** Стилизирани неномерирани и номерирани списъци. Възможности за вграждане на иконки в списъците.
7. **Медиа обекти (Media Objects):** Възможности за стилове на медийни обекти като изображения и текст. Форматиране на изображения и видеа.
8. **Контейнери и решетки (Containers and Grids):** Система от решетки за лесно разпределение на съдържанието в контейнери. Адаптивни дизайни за различни устройства.
9. **Модални прозорци (Modal Windows):** Предварително стилзирани модални прозорци за показване на допълнително съдържание.
10. **Карти (Cards):** Компоненти за съдържанието, стилозирано в карти с изображения, заглавия и текст.
11. **Панели (Panels):** Стилове за панели, които могат да бъдат използвани за разпределяне на съдържание.

## 3.11 Сървърни езици

Сървърните езици са програмни езици, които се използват за разработка на софтуер, който се изпълнява на сървър, обработва заявки от клиентски устройства и предоставя търсените от тях ресурси или услуги. Тези езици се различават от клиентските езици, които се използват в браузърите на потребителите.

Ето някои от популярните сървърни езици:

1. **PHP (Hypertext Preprocessor):** PHP е език за програмиране, който се използва широко за уеб разработка. Той често се вгражда директно в HTML кода и се използва за генериране на динамично съдържание на уеб страниците.
2. **Node.js (JavaScript на сървъра):** Node.js позволява използването на JavaScript за сървърна разработка. Той е известен с асинхронното си програмиране, което го прави подходящ за обработка на голям брой едновременни заявки.
3. **Python:** Python също се използва за сървърна разработка. Различни framework-ове като Django и Flask предоставят инструменти за улесняване на сървърната разработка с Python.
4. **Ruby:** Ruby, заедно със своя framework Ruby on Rails, е популярен избор за разработка на уеб приложения.
5. **Java:** Java е общопризнат за своята устойчивост и е използван за голям брой сървърни приложения, включително уеб услуги и приложения за обработка на данни.
6. **C#:** C# е език, който се използва често за разработка на сървърни приложения в средата на Microsoft с помощта на платформата .NET.

## 3.12 Бази данни и СУРБД

База данни (БД) и Система за Управление на Бази от Данни (СУБД) са ключови концепции в областта на информационните технологии.

### 3.12.1 Бази данни

Базата данни представлява организирана колекция от информация, която е структурирана така, че да може лесно да се съхранява, обработва и извлича. Тя съдържа данни, които са организирани по определен начин.

Основната цел на базата данни е да предоставя ефективен и удобен начин за съхранение и управление на информацията, като осигурява лесен достъп до данните и поддържа тяхната цялостност.

#### 3.12.1.1 Релационни бази данни

Релационна база данни организира данните в редове и колони, които заедно образуват таблица. Таблиците се използват за съхраняване на информация за обектите, които трябва да бъдат представени в базата данни. Всяка колона в таблица съдържа определен вид данни, а полето съхранява действителната стойност на атрибут. Редовете в таблицата представляват колекция от свързани стойности на един обект. Всеки ред в таблица може да бъде маркиран с уникален идентификатор, наречен първичен ключ, а редовете между множество таблици могат да бъдат свързани с помощта на външни ключове. Тези данни могат да бъдат достъпни по много различни начини, без да се реорганизират самите таблици на базата данни.

#### 3.12.1.2 Не-релационни бази данни

Нерелационните бази данни понякога се наричат „NoSQL“, което означава Not Only SQL. Основната разлика между тях е как съхраняват информацията си.

Нерелационната база данни съхранява данни в нетаблична форма и има тенденция да бъде по-гъвкава от традиционните, базирани на SQL структури на релационни бази данни. Той не следва релационния модел, предоставен от традиционните системи за управление на релационни бази данни.

Вместо типичната таблична структура на релационна база данни, NoSQL базите данни съдържат данни в една структура от данни, като например JSON документ. Тъй като този дизайн на нерелационна база данни не изисква схема, той предлага бърза мащабност за управление на големи и обикновено неструктурирани набори от данни.

### 3.12.2 СУБД

СУБД е софтуерен продукт, който управлява създаването, манипулирането и администрирането на база данни. Той предоставя интерфейс между потребителите и самата база данни, позволявайки им да изпълняват заявки и операции с данни.

Основната цел на СУБД е да улесни работата с данни и да гарантира съхранението и достъпа до тях по ефективен и сигурен начин.

Често използвани **СУРБД (Системи за управление на релационни бази данни)** са:

- MySQL - система за управление на релационни бази данни (СУРБД) с отворен код, която използва език за структурирани заявки (SQL) за управление и манипулиране на данни. Тя се използва широко за изграждане и управление на бази данни в различни приложения и уебсайтове.
- MariaDB - система за управление на релационни бази данни (СУРБД) с отворен код, която е разклонение (fork) на MySQL и е проектирана така, че да бъде изключително съвместима с MySQL, като същевременно предлага допълнителни функции и подобрения. Тя се използва широко за управление и съхранение на данни в различни приложения и уебсайтове.
- Postgree - система за управление на релационни бази данни с отворен код, известна със своята гъвкавост, съответствие със стандартите и усъвършенствани функции, като поддръжка на сложни типове данни, мощно индексиране и надеждно управление на транзакции. Обикновено се използва за обработка на големи обеми от данни в различни приложения.
- **MS SQL Server** - система за управление на релационни бази данни (СУРБД), разработена от Microsoft. Осигурява цялостна и мащабна платформа за управление и анализ на структурирани данни, като предлага функции като обработка на транзакции, бизнес анализ и интеграция с други технологии на Microsoft.
- Oracle - широко използвана система за управление на релационни бази данни (СУРБД), разработена от Oracle Corporation. Известна със своята надеждност, мащабност и усъвършенствани функции, Oracle Database често се използва в корпоративни среди за управление и обработка на големи обеми от данни, за поддръжка на сложни транзакции и за осигуряване на надеждна основа за бизнес приложения.

### 3.12.3 Нормализация

**Нормализацията** е процес на организиране на данни в база данни с цел минимизиране на излишествата и предотвратяване на аномалиите при манипулирането на данните.

**Главната цел** на нормализацията е да осигури, че данните са структурирани така, че да не съществуват излишества и зависимости, които биха могли да доведат до проблеми при обработката и поддръжката на базата данни.

**Нормалната форма** (NF) е концепция в теорията на базите данни, която се използва за структуриране на данни в база данни с цел предотвратяване на излишества и аномалии при манипулиране на данни.

1. **Първа нормална форма (1NF):** Основната идея на 1NF е, че всеки атрибут трябва да бъде атомарен, т.е., да не може да съдържа повече от една стойност. Всички стойности в даден атрибут трябва да са от същия тип.

2. **Втора нормална форма (2NF):** 2NF се прилага, когато базата данни е в 1NF, и изисква, че всеки неключов атрибут зависи само от целия ключ (пълен ключ), а не от част от него.

3. **Трета нормална форма (3NF):** 3NF се прилага, когато базата данни е в 2NF, и изисква, че неключовите атрибути не зависят функционално от други неключови атрибути. Това означава, че атрибутите, които не са част от ключа, не трябва да зависят  един от друг.

### 3.12.4 Типове връзки

Връзките между таблиците в база данни се определят от външни ключове (foreign keys) и връзки между тях. Те са от ключово значение за поддържане на целостта на данните.

Три основни типа връзки са:

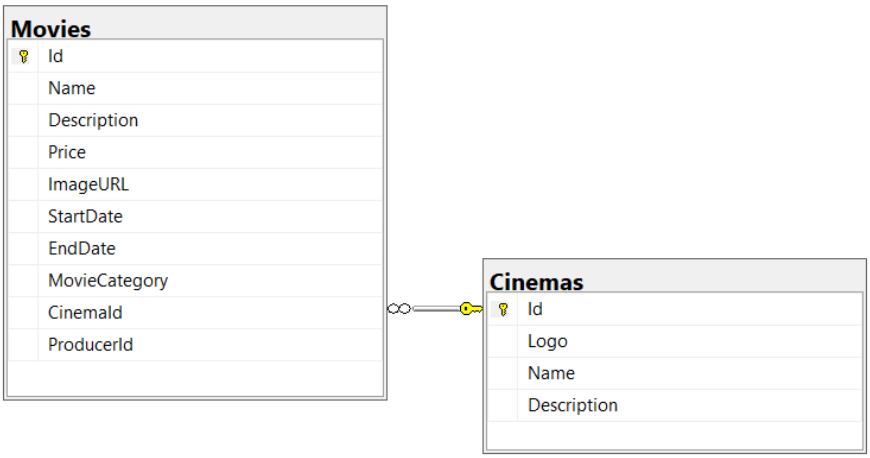
- Едно към едно (**One-to-One**): един запис в таблица е свързан с един и само един запис в друга таблица.

| People | |
|---|---|
| **ID** | **Name** |
| 1 | Alice |
| 2 | Bob |
| 3 | Cathy |

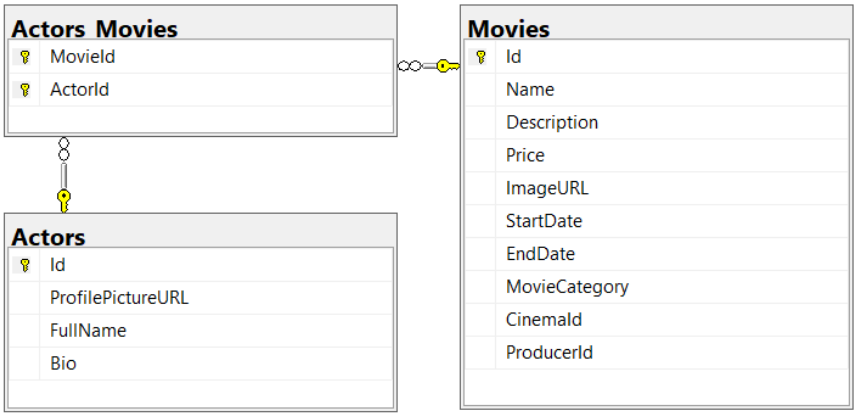
One-to-One

| Drivers Licenses | |
|---|---|
| **UserID** | **Number** |
| 1 | F25532 |
| 2 | S43212 |
| 3 | B98364 |

❖ Едно към много (**One-to-Many**): един запис в таблица е свързан с един или повече записи в друга таблица.




❖ Много към много (**Many-to-Many**): множество записи в една таблица са свързани с множество записи в друга таблица.




### 3.12.5 Ключове

В контекста на релационни бази данни, ключовете са уникални идентификатори, които се използват за идентификация и свързване на записи в различни таблици.

#### 3.12.5.1 Primary key

Уникален идентификатор за всеки запис в таблицата, който гарантира уникалността на всеки ред.

#### 3.12.5.2 Foreign key

Поле в една таблица, което е свързано с основния ключ на друга таблица и установява връзката между тях.

#### 3.12.5.3 Unique

Гарантира уникалността на стойностите в определено поле, но разрешава на полето да бъде празно (null).

#### 3.12.5.4 Index

Описва колона или група от колони, върху които е създаден индекс.

### 3.12.6 Code-first database

Подходът "първо кодът" се използва основно при проектирането, основано на домейна. При този подход се разглежда областта на приложение. Класовете се създават в съответствие с домейна, а не с базата данни. След това се правят проучвания, които съответстват на дизайна на базата данни.

#### 3.12.6.1 Code First Approach

Framework-а за entity-та създава или актуализира базата данни в зависимост от класовете на домейна. Следователно потребителят трябва първо да кодира, а след това entity framework-а ще създаде базата данни, като използва кода. За това той се нарича подход "първо код".

#### 3.12.6.2 Code First Workflow

Подходът "код първо" в MVC следва работния процес с тези стъпки:

- Създаване на класовете, управлявани от домейна
- Конфигуриране на създадените класове в домейна
- Актуализиране или създаване на база данни за класовете от домейна.
- Конфигурирането на класовете на домейна става с помощта на Fluent API. Актуализирането на базата данни се извършва с помощта на автоматизираната миграция.

#### 3.12.6.3 Steps to use code-first

1. Създаване на празна база данни
2. Създаване на проект MVC
3. Създаване на проект за библиотека от класове
4. Добавете Entity Framework към проекта, създаден в предишната стъпка.
5. Имплементация на подхода:

- Създаване на класа Employee

```
public Клас Employee {
public int EmpId { get; set; }
public string EmpName { get; set; }
public float Age { get; set; }
public DateTime DateOfJoining { get; set; }
public float ExpInYears { get; set; }
public Department Department { get; set; }
}
```

- Създаване на класа Department

```
Public class Department {
Public int DeptId { get; set; }
Public string DeptName { get; set; }
Public ICollection<Employee> Employee { get; set; }
}
```

- Кодът първо използва класа DbContext, за да извлече класа context. Класът context излага DBSet, който е колекцията от класове на същности. Кодът за създаване на контекстния клас:

```
Namespace EF6Console {
Public class OfficeContext: DbContext {
Public OfficeContext(): base(){}
Public DbSet<Employee> Employee { get; set; }
Public DbSet<Department> Department { get; set; }
}
```

```
}
```

- Сега, след като е създаден контекстният клас, добавете служител, като го използвате като:

```
Namespace EF6Console {
Клас ExProgram {
Статичен void main(string[], args) {
using(var obj = newOfficeContext())
{
Var emp = new Employee() { EmployeeName = "Peter" };
Obj.Employee.Add(emp);
Obj.SaveChanges();
}
}
}
}
```

6. Препратка към проекта DAL към проекта UI: Добавете препратка, като щракнете с десния бутон на мишката върху References на UI Project.
7. Разрешете миграцията: Отидете до Tools > Package Manager > Manage NuGet Packages for Solution и изпълнете тези команди:

```
Enable-Migrations
Add-migration Initial Create (Добавяне на миграция)
Update-database
```

8. Добавяне на контролер: Отидете до Controller > Add > New Controller и изберете MVC 5 Controller с изгледи, използвайки Entity Framework. Изберете класа на модела, класа на контекста и страницата на оформлението.

Използвайки подхода "първо кодът", можете да създадете класовете и същностите и след това да актуализирате базата данни.

1. Create blank database
2. Create MVC project
3. Create the class library project
4. Add Entity Framework to the project created in the previous step.

5. Code First Approach Implementation:

- Create the Employee class

```
public Клас Employee {
public int EmpId { get; set; }
public string EmpName { get; set; }
public float Age { get; set; }
public DateTime DateOfJoining { get; set; }
public float ExpInYears { get; set; }
public Department Department { get; set; }
}
```

- Create the department class

```
Public class Department {
Public int DeptId { get; set; }
Public string DeptName { get; set; }
Public ICollection<Employee> Employee { get; set; }
}
```

- Code first uses the DbContext class to derive the context class. The context class exposes the DBSet, which is the collection of entity classes. The code for the creation of context class:

```
Namespace EF6Console {
Public class OfficeContext: DbContext {
Public OfficeContext(): base(){}
Public DbSet<Employee> Employee { get; set; }
Public DbSet<Department> Department { get; set; }
}
}
```

- Now as context class is created, add employee using it as:

```
Namespace EF6Console {
Клас ExProgram {
Статичен void main(string[], args) {
```

```
using(var obj = newOfficeContext())
{
Var emp = new Employee() { EmployeeName = "Peter" };
Obj.Employee.Add(emp);
Obj.SaveChanges();
}
}
}
}
```

6. Reference DAL Project to UI Project: Add reference by right-clicking the References of UI Project.
7. Enable Migration: Navigate to Tools > Package Manager > Manage NuGet Packages for Solution and run these commands:

```
Enable-Migrations
Add-migration Initial Create
Update-database
```

8. Add Controller: Navigate to Controller > Add > New Controller and select the MVC 5 Controller with views, using Entity Framework. Select the model class, context class, and layout page.

Using the code first approach, you can create the classes and entities and then update the database.

https://www.upgrad.com/blog/code-first-approach-in-mvc/

## 3.13 UML и E/R диаграми.

UML (Unified Modeling Language) и E/R (Entity-Relationship) диаграмите са инструменти за моделиране, използвани в различни области на разработването на софтуер и бази данни

### 3.13.1 UML диаграми

**UML (Unified Modeling Language):** стандартен език за моделиране, който предоставя нотация за създаване на визуални модели на системи. Той се използва за описание на структурата и поведението на дадена система. В UML има няколко вида диаграми, всяка от които е предназначена за моделиране на определени аспекти на дадена

система.

**Някои от основните видове диаграми на UML включват:**

- **Диаграма на класовете (Class Diagram):** се използва за моделиране на структурата на една система чрез описание на класове, техните атрибути, методи и връзки между класовете.
- **Диаграма на случаите на употреба (Use Case Diagram):** се използва за описване на функционалността на дадена система от гледна точка на нейните потребители.
- **Диаграма на последователността (Sequence Diagram):** Диаграмата на последователността се състои от две части: Използва се за моделиране на взаимодействието между обекти или компоненти на системата във времето.
- **Диаграма на дейностите (Activity Diagram):** Осигурява графично представяне на процесите в системата.
- **Диаграма на състоянието (State Diagram):** Използва се за моделиране на различните състояния, в които може да се намира даден обект или система.

### 3.13.2 E/R (Entity-Relationship) диаграми

E/R диаграмите се използват широко за моделиране на структурата на базите данни. Те описват entity-тата и връзките между тях. Някои от основните елементи на E/R диаграмите са:

- **Субекти:** Представляват обекти или концепции, които могат да се съхраняват в базата данни. Например: Клиент, Продукт и т.н.
- **Връзки:** Определят връзките между entity-тата. Например, връзка "Поръчки" между "Клиент" и "Продукт".
- **Атрибути:** Представляват характеристики на entity-тата. Например, "Име" и "Адрес" могат да бъдат атрибути на същността "Клиент".
- **Ключове:** Определят уникални идентификатори за entity-тата, например първични ключове.

## 3.14 Многослойна архитектура

Многослойната архитектура е основен шаблон за дизайн, който е изиграл ключова роля при оформянето на структурата на софтуерните системи.

Това е модулен подход, който организира система в отделни слоеве, всеки от които отговаря за специфична функционалност.

В многослойната архитектура основния модел на проектиране организира приложението в отделни хоризонтални слоеве, всеки от които служи за специфична цел.

Въпреки че точният брой и видовете слоеве могат да варират, стандартната многослойна архитектура често включва презентационни, бизнес, постоянни и база данни слоеве. Основната цел е да се подобри модулността, поддръжката и мащабността чрез изолиране на различни проблеми в отделни слоеве.

### 3.14.1 Presentation Layer

**Роля**

Презентационният слой, известен още като слой на потребителския интерфейс (UI), е най-горният слой, който взаимодейства директно с крайните потребители. Основната му роля е да представя информация на потребителите по разбираем и визуално привлекателен начин.

**Отговорности**

- **Изобразяване на потребителски интерфейс**: Управлява изобразяването на елементите на потребителския интерфейс като формуляри, бутони и интерактивни компоненти.
- **Обработка на потребителско въвеждане**: Улавя въведеното от потребителя чрез различни устройства и канали, улеснявайки взаимодействието с приложението.
- **Логика на дисплея**: Управлява логиката на дисплея, осигурявайки правилното представяне на данните и отговорите на действията на потребителя.
- **Комуникация с други слоеве**: Комуникира със слоевете отдолу, често чрез добре дефинирани интерфейси или API, за извличане или изпращане на подходящи данни.

### 3.14.2 Business Layer (Application layer, Logic Layer)

**Роля**

Бизнес слоят, известен също като логически слой или приложен слой, е ядрото на приложението, което съдържа бизнес логиката и способностите за обработка.

**Отговорности**

- **Внедряване на бизнес логика**: Внедрява основните бизнес правила и логика, които определят как функционира приложението.
- **Потвърждаване на данни**: Валидира и обработва данните, получени от презентационния слой преди по-нататъшно действие.
- **Управление на работния процес**: Управлява потока от операции и процеси в приложението.
- **Междуслойна комуникация**: Действа като посредник между слоя за представяне и слоя за постоянство, като осигурява безпроблемна комуникация.

### 3.14.3 Persistence Layer

**Роля**

Persistance слоят, често наричан слой за достъп до данни, отговаря за управлението на съхранението и извличането на данни.

**Отговорности**

- **Съхранение и извличане на данни**: Управлява съхранението и извличането на данни, взаимодействайки с бази данни или други системи за съхранение на данни.
- **Изпълнение на заявка**: Изпълнява заявки към база данни въз основа на заявки от бизнес слоя.
- **Управление на транзакциите**: Гарантира целостта на данните чрез управление на транзакции, включително операции за извършване и връщане назад.
- **Data mapping / Съпоставяне на данни**: Съпоставя данните между обектно-ориентирания модел на приложението и модела на релационна база данни.

### 3.14.4 Database Layer

**Роля**

Слоят на базата данни представлява физическата система за съхранение на данни, която може да бъде релационна база данни, NoSQL база данни или друг механизъм за съхранение на данни.

**Отговорности**

- **Хранилище за данни**: Съхранява и организира данните по структуриран начин.
- **Извличане на данни**: Осигурява механизми за ефективно извличане на

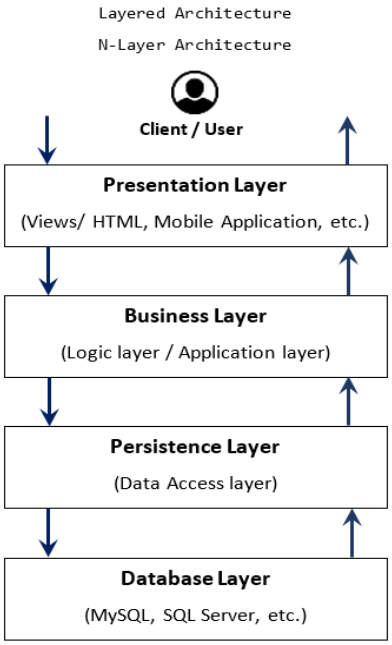
съхранени данни въз основа на заявки.

- **Сигурност на данните**: Прилага мерки за сигурност за защита на съхраняваните данни, включително контрол над достъпа и криптиране.
- **Управление на бази данни**: Управлява операции с бази данни, включително архивиране, възстановяване и оптимизация.

### 3.14.5 Взаимодействие между слоевете

1. **Presentation layer към Business layer:** Въведеното от потребителя се обработва от презентационния слой и се предава на бизнес слоя за валидиране и обработка.
2. **Business layer към Persistence layer:** След обработката бизнес слоят взаимодейства със слоя за постоянство, за да съхранява или извлича данни.
3. **Persistence layer към Database layer:** Слоят за постоянство комуникира със слоя база данни, за да изпълнява заявки и да управлява транзакции с данни.

Данните протичат вертикално през слоевете, като всеки слой изпълнява специфични операции и предава данните на следващия слой в йерархията - фиг. 03.08-1




фиг. 03.12-1

### 3.14.6 Основни характеристики

1. **Модулност и разделяне на отговорностите:** Всеки слой има специфична роля и отговорност, насърчавайки ясното разделение на проблемите. Например, презентационният слой се фокусира върху потребителския интерфейс, бизнес слоят обработва логиката, постоянният слой управлява съхранението на данни, а нивото на базата данни съхранява действителните данни.
2. **Гъвкавост и повторно използване:** Модулният характер на многослойната архитектура улеснява гъвкавостта и повторното използване. Разработчиците могат да променят или заменят отделни слоеве, без да засягат цялата система, насърчавайки по-лесна поддръжка и актуализации.
3. **Мащабност:** Мащабността е присъща на модела на многослойната архитектура. С нарастването на приложението могат да се добавят или модифицират допълнителни слоеве, за да се приспособят към повишена сложност или промени в изискванията.
4. **Лесно развитие:** Разделянето на проблемите опростява развитието, като позволява на различни екипи или лица да се съсредоточат върху конкретни слоеве. Този подход на паралелно развитие може да доведе до по-бързо и по-ефективно изпълнение на проекта.

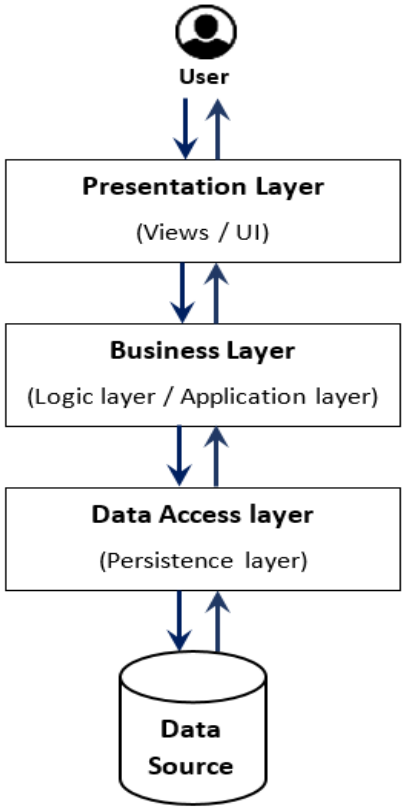
### 3.14.7 Предимства

1. **Поддържане** - Ясното разделение на отговорностите улеснява намирането и отстраняването на проблеми. Задачите по поддръжката са по-лесни, тъй като промените в един слой не се отразяват непременно на други.
2. **Четимост и разбираемост** - Многослойната архитектура подобрява четливостта и разбираемостта на кода. Разработчиците могат да разберат и да работят върху конкретни слоеве, без да бъдат претоварени от сложността на цялата система.
3. **Гъвкавост и адаптивност** - Модулният дизайн позволява гъвкавост и адаптивност към променящите се изисквания. Нови функции или технологии могат да бъдат интегрирани в конкретни слоеве, без да се нарушава цялостната система.
4. **Капсулация** - Всеки слой капсулира своята функционалност, осигурявайки ниво на абстракция. Това капсулиране защитава вътрешната работа на слоя.

## 3.15 Трислойни модели

### 3.15.1 Линейна 3-слойна архитектура

Трислойната архитектура е софтуерен модел клиент-сървър, където приложението е разделено на три основни компонента или слоя: presentation слой, business слой и слой data access.




фиг. 03.13-1

1. **Presentation layer:** Най-горният слой, отговорен за представянето на потребителския интерфейс.

   **Функции:**

- ❖ Изобразява потребителския интерфейс (UI) и взаимодейства с крайните потребители.
- ❖ Използва уеб технологии като HTML, JavaScript, CSS или рамки за уеб разработка

2. **Business layer:** Средният слой, съдържащ функционалната/оперативната логика на процеса

   **Функции**:

   - ❖ Управлява основните възможности на приложението.
   - ❖ Извършва подробна обработка въз основа на бизнес правила

3. **Data Access Layer:** Слоят, отговорен за съхранението и достъпа до данни.

   **Функции**:

   - ❖ Управлява съхранението и извличането на данни от приложението.
   - ❖ Използва системи за бази данни като MySQL, Oracle, MongoDB и др.

## Взаимодействие между слоевете

1. **Презентационен слой към приложен слой:**

   - ❖ Презентационният слой изпраща потребителски вход и заявки към приложния слой.
   - ❖ Използва API извиквания за комуникация с приложния слой.
   - ❖ Получава обработени данни от приложния слой за представяне.

2. **Приложен слой към слой данни:**

   - ❖ Приложният слой взаимодейства със слоя данни за извличане и съхранение на данни.
   - ❖ Използва API извиквания за комуникация със слоя данни.
   - ❖ Гарантира прилагането на бизнес логиката преди данните да бъдат съхранени или извлечени.

### 3.15.2 MVC архитектура

**Model-View-Controller** (MVC) стои като крайъгълен камък в съвременната разработка на софтуер, предоставяйки структуриран и ефективен подход за изграждане на уеб приложения. Този архитектурен модел разделя приложението на три основни логически компонента: модел (model), изглед (view) и контролер (controller).

**MVC компоненти:**

#### 3.15.2.1 Controller

Контролерът действа като посредник между изгледите и модела. Той обработва бизнес логиката, обработва входящи заявки, манипулира данни с помощта на компонента Model и взаимодейства с View, за да изобрази крайния изход.

**Отговорности:**

- Обработка на бизнес логика: Контролерът обработва цялата бизнес логика, като взема решения въз основа на данните, получени от модела.
- Обработка на заявка: Той обработва входящи заявки от потребителския интерфейс и предприема подходящи действия.
- Манипулиране на данни: Контролерът манипулира данни с помощта на компонента Model, за да ги подготви за представяне.

#### 3.15.2.2 View

Компонентът View отговаря за логиката на потребителския интерфейс на приложението. Той генерира потребителския интерфейс за крайния потребител въз основа на данните, предоставени от модела, но взаимодейства с контролера.

**Отговорности:**

- Изобразяване на потребителски интерфейс: Изгледът изобразява елементите на потребителския интерфейс, представяйки информацията на крайния потребител.
- Взаимодействие с контролера: Той взаимодейства с контролера за получаване на потребителски вход и препращането му за обработка.
- Показване на данни: Изгледът показва данни на потребителя, като осигурява визуално привлекателно и разбираемо представяне.

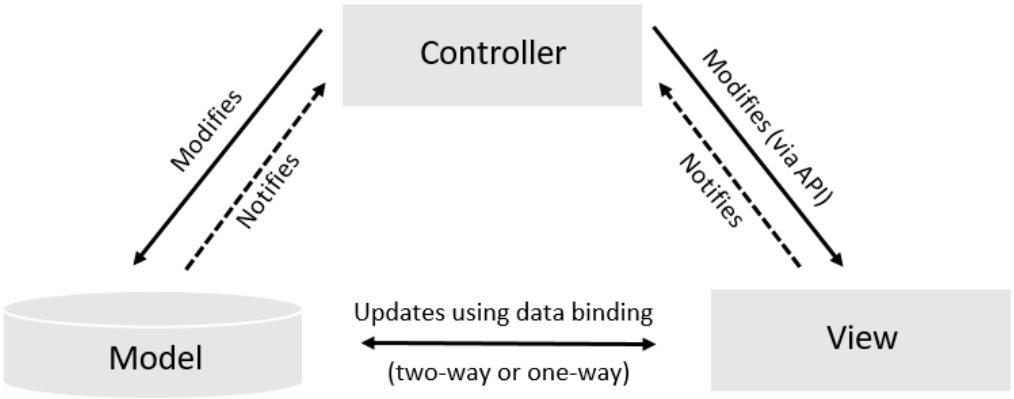
### 3.15.2.3 Model

Моделът съответства на свързаната с данните логика на приложението. Той управлява данните, които се прехвърлят между компонентите View и Controller и обработва данни, свързани с бизнес логиката.

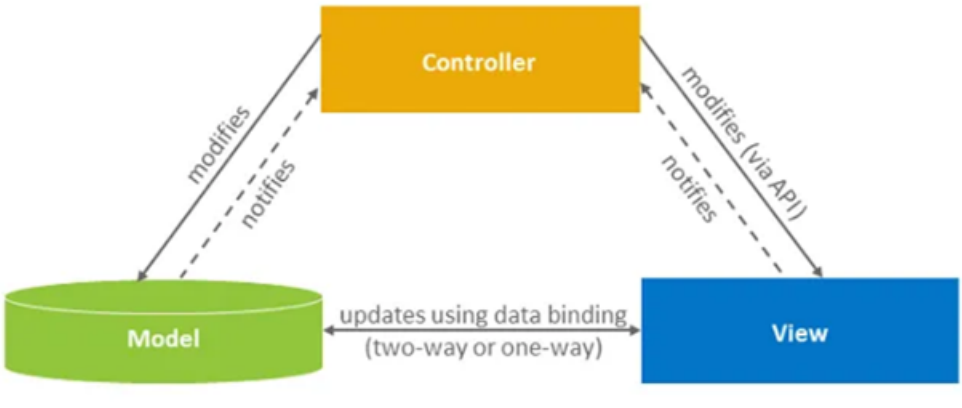
**Отговорности:**

- Управление на данни: Управлява данните, използвани в приложението, включително съхранение, извличане и манипулиране.
- Внедряване на бизнес логика: Внедрява основните бизнес правила и логика, които определят функционалността на приложението.
- Взаимодействие с контролера: Отговаря на заявки от администратора, като предоставя необходимите данни за представяне.

### 3.15.2.4 Графично представяне на MVC




Долната схема да се изтрие




фиг. 03.13-2

### 3.15.2.5 Взаимодействие на MVC компонентите

1. **Взаимодействие модел-изглед:**

**Модел към изглед:**

- Разпространение на данни: Моделът, отговорен за управлението на данните на приложението, уведомява View за всякакви промени в състоянието на данните.
- Механизъм за наблюдение: Изгледите често използват механизъм за наблюдение, където регистрират интерес към конкретни елементи от данни с модела. При промени моделът сигнализира тези регистрирани изгледи.
- Изобразяване на данни: Изгледите получават актуализации на данни от модела и динамично изобразяват актуализираната информация в потребителския интерфейс.

**Изглед към модел**

- Запаметяване/записване (capture) на потребителско въвеждане: Изгледите улавят въведените от потребителя данни, като щраквания върху бутони или изпращане на формуляр, инициирайки действие.
- Заявка до контролера: Вместо директно да манипулира модела, изгледът изпраща намерението на потребителя до контролера за подходяща обработка.
- Обработка на потребителското взаимодействие: Изгледите обработват потребителските взаимодействия и предават подходяща информация на контролера, като гарантират разделяне на проблемите.

2. **Взаимодействие контролер-модел:**

- Приемане на потребителски данни: Контролерите получават информация от потребителя от изгледите, интерпретирайки намерението зад действието.
- Изпълнение на бизнес логиката: Въз основа на получените входни данни контролерът извиква подходяща бизнес логика в рамките на модела, за да

обработи заявката.

- Актуализация на състоянието на модела: Моделът, след извършване на необходимите операции, актуализира своето състояние, отразявайки промените в данните на приложението.

3. **Взаимодействие контролер-изглед:**

- Известие за актуализация на модела: След обработка на въведеното от потребителя и актуализиране на модела, контролерът уведомява регистрираните изгледи за промени в модела.
- Преглед на тригера за изобразяване: Изгледите, при получаване на известия, задействат процес на повторно изобразяване, за да отразят актуализираните данни в потребителския интерфейс.
- Комуникационна абстракция: Контролерите предпазват Views от сложността на бизнес логиката, осигурявайки ниво на абстракция за безпроблемна комуникация.

### 3.15.3 MVVM

Подобно на много други шаблони за дизайн, MVVM помага за организирането на кода и разделянето на програмите на модули, за да направи разработката, актуализирането и повторната употреба на кода по-прости и по-бързи.

Шаблонът MVVM помага за чистото отделяне на бизнес логиката и логиката на представяне на приложението от неговия потребителски интерфейс (UI).

Има три основни компонента в модела MVVM: моделът, изгледът и моделът на изгледа. Всеки има различно предназначение.

**Компоненти на MVVM**

#### 3.15.3.1 Model

Models (класове на моделите) са невизуални класове, които капсулират данните на приложението. Следователно моделът може да се разглежда като представящ модела на домейна на приложението, който обикновено включва модел на данни заедно с бизнес и логика за валидиране.

Този слой отговаря за абстракцията на източниците на данни. Моделът и ViewModel работят заедно, за да получат и запазят данните.

Моделът съхранява данните и да изпълнява бизнес логиката. Не трябва да се грижи за това как информацията се представя.

#### 3.15.3.2 View

Изгледът (View) представлява потребителския интерфейс на приложението. Това частта, с което потребителят взаимодейства. Това е презентационната част.

Изгледът е колекцията от видими елементи, която също получава въведени от потребителя данни. Това включва потребителски интерфейси (UI), анимации и текст.

Целта на този слой е да информира ViewModel за действието на потребителя. Този слой наблюдава ViewModel и не съдържа никаква логика на приложението, логиката е делегирана на ViewModel

#### 3.15.3.3 ViewModel

ViewModel се намира между слоевете View и Model.

Това е логиката на изгледа (View). ViewModel се нарича още презентационна логика. Изгледът (View) и ViewModel комуникират помежду си. Заявката от ViewModel се препраща към модела (слой на бизнес логика/ слой за достъп до данни). ViewModel позволява споделяне на резултата / данните от изчисленията с изгледа.

ViewModel е мястото, където се намират контролите за взаимодействие с View

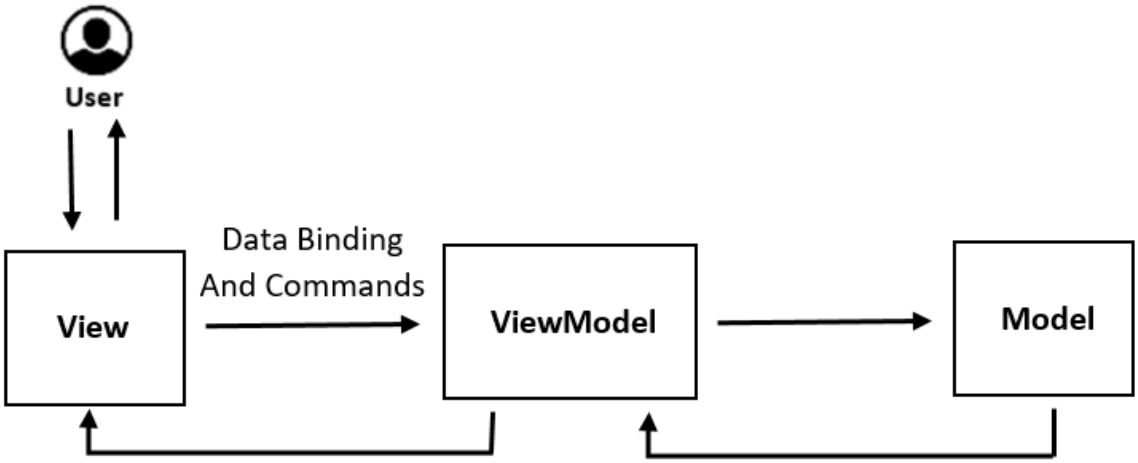
Моделът на изгледа (ViewModel) имплементира свойства и команди, към които изгледът може да се обвърже. Обвързването се използва за свързване на елементите на потребителския интерфейс в View към контролите ( командите и свойствата ) в ViewModel.

ViewModel уведомява изгледа за всякакви промени в състоянието чрез събития за известяване за промяна.

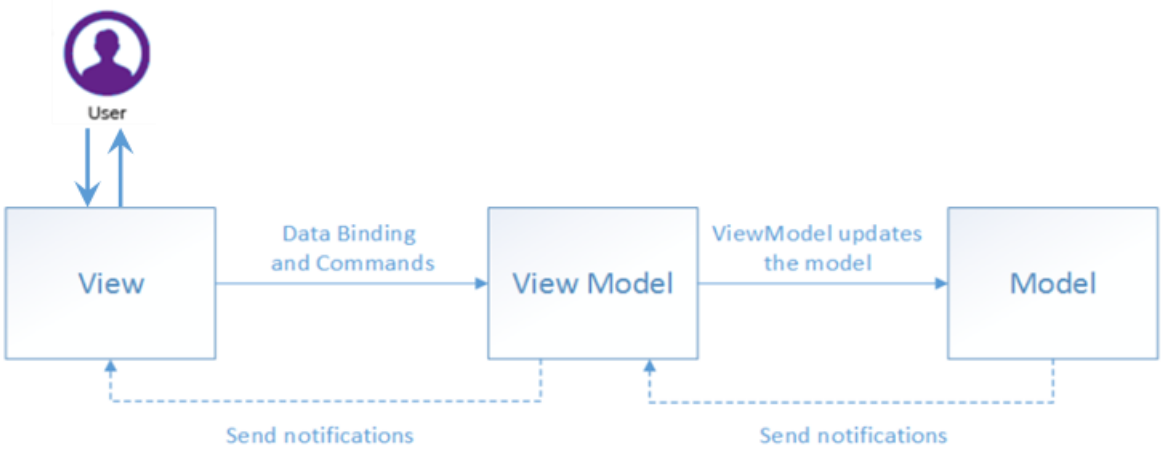
Свойствата и командите, които ViewModel предоставя, определят функционалността, която да се предлага от потребителския интерфейс, но изгледът определя как тази функционалност да бъде показана.

ViewModel не трябва да има преки зависимости от изгледа. Това го прави лесен за тестване и независим от конкретния интерфейс.

### 3.15.3.4 Графично представяне на MVVM




Долната схема да се изтрие




фиг. 03.12-3

### 3.15.3.5 Характеристики на MVVM

1. **Ясно разделяне на дейностите/функциите:** MVVM налага ясно разграничение между логиката на потребителския интерфейс (View), данните и логиката на приложението (Model) и логиката за управление на представянето и състоянието (ViewModel).
2. **Двупосочно обвързване на данни:** MVVM улеснява безпроблемната комуникация между View и ViewModel чрез двупосочно обвързване на данни. Промените в единия компонент автоматично се отразяват в другия

3. **Тестване:** Компонентите в MVVM, особено ViewModel, са много лесни за тестване.
4. **Повторна употреба на компонентите:** MVVM улеснява създаването на компоненти за многократна употреба. ViewModel, по-специално, често може да се използва повторно в различни изгледи.

#### 3.15.3.6 Взаимодействие на MVVM компонентите

1. **Model към ViewModel**
   - ❖ Моделът уведомява ViewModel за всякакви промени в данните на приложението.
   - ❖ ViewModel обработва актуализираните данни и ги подготвя за представяне на View.
2. **ViewModel към View:**
   - ❖ ViewModel актуализира изглед чрез механизми за свързване на данни.
   - ❖ Изгледът, наблюдавайки промените в ViewModel, изобразява динамично актуализираните данни в потребителския интерфейс.
3. **Взаимодействие с потребителя (View(изглед) към ViewModel):**
   - ❖ Потребителските взаимодействия, като щраквания върху бутони или изпращане на формуляр, се улавят от изгледа.
   - ❖ Изгледът препраща тези взаимодействия към ViewModel за подходяща обработка.
4. **Изпълнение на бизнес логика (ViewModel към Model):**
   - ❖ ViewModel, при получаване на потребителски вход, извиква необходимата бизнес логика в модела.
   - ❖ Моделът обработва данните, актуализирайки съответно състоянието си.

### 3.15.4 Съпоставка между моделите

#### 3.15.4.1 Линейна трислойна vs MVC

**Между Линейна 3-слойна архитектура и MVC**

1. **Структурни разлики:**

- **Линейна 3-слойна:** Линейна организация с отделни слоеве (представяне, приложение, данни).
- **MVC:** Триъгълна организация с три взаимосвързани компонента (модел, изглед, контролер).

2. **Модели на взаимодействие:**
   - **Линейна 3-слойна:** Линейна комуникация между съседни слоеве. Строго взаимодействие, базирано на API от слой до слой.
   - **MVC:** Триъгълен комуникационен модел. Изгледите взаимодействат с контролерите, контролерите актуализират моделите, а изгледите наблюдават промените в модела.
3. **Гъвкавост и повторно използване:**
   - **Линейна 3-слойна:** Компонентите в слоевете могат да се използват повторно, но им липсва модулността на MVC.
   - **MVC:** Висока модулност и многократна употреба, тъй като всеки компонент (модел, изглед, контролер) има специфична отговорност.
4. **Управление на зависимостите:**
   - **Линейна 3-слойна:** Слоевете са линейно зависими един от друг.
   - **MVC:** Отделени компоненти, намаляващи зависимостите между модел, изглед и контролер.
5. **Мащабност:**
   - **Линейна 3-слойна:** Мащабността може да е ограничена поради линейни зависимости.
   - **MVC:** Компонентите могат да бъдат индивидуално мащабирани, подобрявайки мащабността.

#### 3.15.4.2 MVC vs MVVM

**Между MVC и MVVM:**

1. **Роля на ViewModel/Controller:**
   - **MVC:** Контролерът обработва въвеждането от потребителя и актуализира модела.

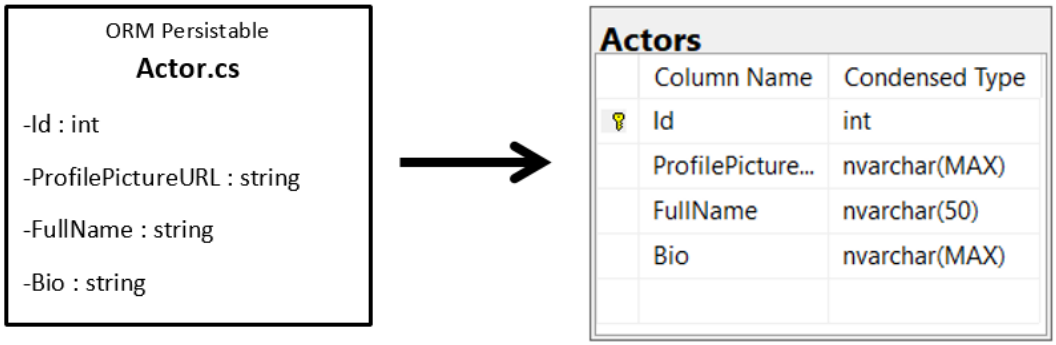
- **MVVM:** ViewModel управлява въведеното от потребителя, актуализира модела и комуникира с изгледа.

2. **Обвързване на данни:**
   - **MVC:** Обикновено разчита на ръчни актуализации между изгледа и модела.
   - **MVVM:** Използва двупосочно обвързване на данни, автоматизирайки синхронизирането на View и ViewModel.
3. **Управление на зависимостите:**
   - **MVC:** Изгледите са по-тясно свързани с контролерите.
   - **MVVM:** Изгледите са слабо свързани с ViewModels, което спомага за по-лесната поддръжка.
4. **Компонентно тестване (Unit Testing)**
   - **MVC:** Контролерите може да са по-трудни за независимо тестване.
   - **MVVM:** ViewModels са предназначени за по-лесно тестване на отделни компоненти поради техния отделен характер.
5. **Повторна употреба на кода:**
   - **MVC:** Повторната употреба може да бъде ограничена поради тясно свързване.
   - **MVVM:** Висока възможност за повторно използване на кода, особено в компонентите на ViewModel.

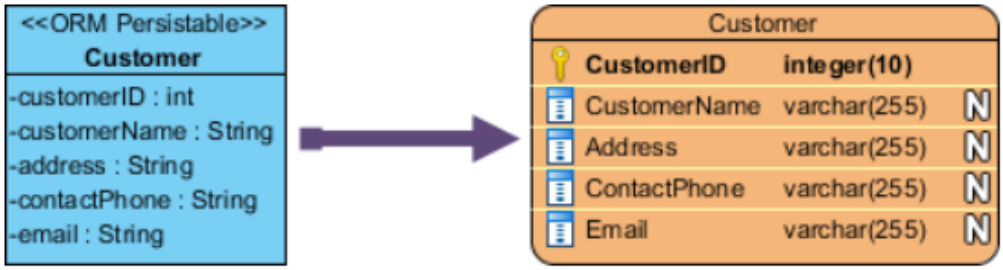
## 3.16 Обектно-релационно свързване (ORM frameworks)

**ORM (Object-Relational Mapping)** е технология, която служи за **съпоставяне на обекти в програмен код с редове в релационни бази данни**. Това улеснява работата на програмистите, като им позволява да работят с обекти и класове, вместо да се занимават директно със заявки към базата данни. ORM frameworks са библиотеки или инструменти, които предоставят този тип функционалност.

На примерната схема по-долу е изобразена съпоставка между един клас (обектен модел) и съответстващата му таблица (релационен модел)




Долната схема да се изтрие




Фиг.03.10-1
Примерна съпоставка между клас и таблица

**Ето основния начин, по който работи ORM:**

1. **Обектно-ориентиран модел:** В програмния код се използват класове и обекти, които представляват **entities / ентитетите** и техните взаимоотношения.
2. **Релационна база данни:** В базата данни се използват таблиците за съхранение на данни, като връзките между тези таблиците отразяват връзките между обектите в обектно-ориентирания модел.
3. **ORM Mapping:** ORM **framework-ът** извършва мапиране (съпоставяне) между обектите и таблиците в базата данни. Това включва съпоставяне на полета и методи на класовете с колони и операции в таблиците.
4. **Автоматизация на заявките:** ORM framework-ът предоставя методи и инструменти, които позволяват на програмистите да извършват операции като създаване, четене, обновяване и изтриване (CRUD) на данни, като използват обектно-ориентиран синтаксис, а не SQL заявки.
5. **Съхранение на обектното състояние в базата данни:** Промените в обектите се

отразяват автоматично в базата данни и обратно, без да е необходима директна манипулация на SQL от страна на програмиста.

**Някои широко използвани работни рамки за съпоставка/ ORM framework-ове са:**

1. **Entity Framework (.NET):** Entity Framework е ORM framework, използван в платформата на Microsoft .NET. Той позволява разработчиците да работят с бази данни като обекти в .NET приложенията си.

2. **Hibernate (Java):** Hibernate е един от най-популярните ORM инструменти за Java приложения. Той предоставя мощен начин за мапиране / съпоставяне на Java обекти към релационни таблици в база данни.

3. **Spring Data JPA (Java):** Това е част от Spring Framework за Java и предоставя лесен начин за използване на JPA (Java Persistence API) за работа с бази данни.

4. **Django ORM (Python):** В света на Python, Django ORM предоставя лесен начин за мапиране на модели на данни към релационни таблици в бази данни.

5. **SQLAlchemy (Python):** SQLAlchemy предоставя гъвкав и мощен инструмент за ORM в Python. Той предоставя също така и "Core" част, която позволява използването на SQL езика направо, ако е необходимо.

6. **Rails ActiveRecord (Ruby on Rails):** В рамките на Ruby on Rails, ActiveRecord предоставя ORM функционалност, която позволява лесно взаимодействие с бази данни.

7. **Sequelize (Node.js):** Ако работите с Node.js, Sequelize е ORM framework, което поддържа различни релационни бази данни като MySQL, PostgreSQL и други.

## 3.17 Упълномощаване (authentication) и удостоверяване (authorization)

"Упълномощаване" и "удостоверяване" са два термина, които се използват в областта на сигурността на информацията, особено в контекста на достъпа до ресурси или системи. Въпреки че често се използват заедно, те представляват различни аспекти на сигурността.

### 3.17.1 Authentication

Удостоверяването се отнася до процеса на потвърждение на идентичността на потребител, система или приложение. Това може да бъде посредством

предоставяне на потребителско име и парола, използване на биометрични данни, използване на ключове и сертификати и др.

Основната цел на удостоверяването е да провери дали субектът (потребителят, системата и т.н.) е този, който твърди, че е.

### 3.17.2 Authorization

Упълномощаването се отнася до процеса на предоставяне на права и разрешения на потребителя след успешно удостоверяване. Този процес определя какви действия и ресурси са разрешени за конкретния потребител или система.

Целта на упълномощаването е да определи какво е разрешено и какво не е разрешено за конкретния идентифициран субект.

### 3.17.3 Съпоставка

| | **Фокус** | **Цел** |
|---|---|---|
| **Удостоверяване** | Фокусира се върху идентичността на потребителя или субекта. | Целта е потвърждение на идентичността. |
| **Упълномощаване** | Фокусира се върху предоставянето на права и разрешения след успешно удостоверяване. | Целта е управление на достъпа и определяне на права. |

# 4 Използвани технологии и софтуер

След направеното изследване на мрежови протоколи, http заявки, езиците програмиране, СУРБД, архитектурните модели за изграждане на приложения, изграждане на семантични страници, адаптивен (responsive) дизайн, може да се определят следните уеб технологии необходими за изпълнението на настоящия дипломен проект

- ❖ За изгледи / визуализация и адаптивен дизайн HTML, CSS, Bootstrap
- ❖ Програмен език C#
- ❖ СУБД - SQL Server
- ❖ Платформа - .NET

Приложението ще бъде разработено като уеб базирана система на принципа на MVVM модела, ще има добра функционалност и лесен за използване адаптивен (responsive) интерфейс, подходящ за различни устройства (компютри, таблети и смартфони).

Предложеното в дипломния проект многослойно уеб приложение не претендира за краен и завършен продукт, но той ще бъде изцяло работещо уеб приложение, което може постоянно да се развива и усъвършенства, като се добавят нови функционалности.

- **Entity Framework (.NET):** Entity Framework е ORM framework, използван в платформата на Microsoft .NET. Той позволява разработчиците да работят с бази данни като обекти в .NET приложенията си.
- **MS SQl Ser:** Entity Framework е ORM framework, използван в платформата на Microsoft .NET. Той позволява разработчиците да работят с бази данни като обекти в .NET приложенията си.

**Visual Studio:**

- **Поддръжка на няколко езика:** Visual Studio поддържа множество езици за програмиране, включително C#, VB.NET, F#, C++, Python, JavaScript и др. Това го прави универсално за широк спектър от сценарии за разработка.
- **Редактор на код с IntelliSense:** Редакторът на код във Visual Studio предоставя функции като подчертаване на синтаксиса, завършване на кода и IntelliSense, които помагат на разработчиците да пишат код по-ефективно и с по-малко грешки.
- **Дебъгер:** Visual Studio включва мощен дебъгер с функции като точки на прекъсване, поетапно дебъгване, прозорци за наблюдение и други, които улесняват идентифицирането и отстраняването на проблеми в кода.
- **NuGet Package Manager:** Visual Studio се интегрира с NuGet, мениджър на пакети за .NET, което позволява на разработчиците лесно да управляват и инсталират библиотеки и зависимости от трети страни в своите проекти.

**SQL Server Management Studio:**

- **Графичен потребителски интерфейс (GUI):** SSMS предоставя удобен за потребителя графичен интерфейс за управление и взаимодействие с базите данни на SQL Server. Той позволява на потребителите да изпълняват различни задачи, без да се налага да пишат сложни SQL заявки.
- **Диаграми на бази данни:** SSMS включва инструмент за визуална диаграма на базата данни, който позволява на потребителите да създават, променят и визуализират връзките между таблиците в базата данни.

**C#:**

- **Опростен и разбираем синтаксис:** C# е проектиран с изчистен и лесен за четене синтаксис, което го прави достъпен за начинаещи и приятен за опитни разработчици.
- **Обектно-ориентирано програмиране (ООП):** C# е изцяло обектно-ориентиран език за програмиране, който насърчава организацията на кода, повторната му използваемост и модулността чрез концепции като класове и наследяване.
- **Безопасност на типовете:** C# е статично типизиран, което означава, че типовете променливи се проверяват по време на компилация. Това помага за откриване на грешки в началото на процеса на разработка и повишава надеждността на кода.
- **Обширна стандартна библиотека (.NET Framework/.NET Core/.NET 5+):** C# е част от екосистемата на .NET, която включва обширна стандартна библиотека с богат набор от вградени класове и API за общи задачи на програмирането, като например въвеждане/извеждане на файлове, работа в мрежа и др.

**HTML:**

- **Структура и семантика:** HTML предоставя структуриран начин за организиране на съдържанието в уеб. Той използва тагове, за да определи структурата на документа, като например заглавия, параграфи, списъци и др.
- **Съвместимост с различни браузъри:** HTML е проектиран така, че да е съвместим с различни уеб браузъри, като осигурява последователно изживяване за потребителите на различни платформи.
- **Адаптивен дизайн:** HTML включва функции, които поддържат адаптивен уеб дизайн, позволявайки на разработчиците да създават уебсайтове, които се адаптират към различни размери на екрана и устройствата.

**CSS:**

- **Разделяне на грижите:** CSS позволява разделянето на стила от съдържанието. Това означава, че промените във външния вид на уебсайта могат да се правят, без да се променя основната структура на HTML.
- **Последователен стил:** CSS дава възможност за последователно стилизиране на уеб страниците, като осигурява еднакъв вид и усещане за целия уебсайт.

- **Адаптивен дизайн:** CSS играе ключова роля в адаптивния уеб дизайн. Запитванията за медии и техниките за гъвкаво оформление позволяват на разработчиците да създават уебсайтове, които се адаптират към различни устройства и размери на екрана.

**JavaSript:**

- **Инструменти за разработчици на браузъри:** Съвременните уеб браузъри разполагат с мощни инструменти за разработчици, които позволяват отстраняване на грешки, профилиране и анализ на производителността на JavaScript кода директно в браузъра.
- **Динамично типизиране:** JavaScript е динамично типизиран, което означава, че типовете на променливите се определят по време на изпълнение. Тази гъвкавост позволява по-бърза разработка и по-лесно създаване на прототипи.
- **Скриптиране от страна на клиента:** JavaScript е известен предимно с ролята си в разработването на уеб страници от страна на клиента. Той позволява на разработчиците да създават динамични и интерактивни потребителски интерфейси, които реагират на действията на потребителя, без да изискват презареждане на страницата.

**Microsoft.AspNetCore.Identity.EntityFrameworkCore:**

- **Управление на потребители и роли:** Пакетът предоставя цялостен набор от приложни програмни интерфейси за управление на потребители и роли. Това включва създаване, актуализиране и изтриване на потребителски акаунти, както и присвояване на роли на потребителите.
- **Покриване на пароли и сигурност:** ASP.NET Core Identity автоматично хешира паролите, като добавя ниво на сигурност към потребителските акаунти. Той използва стандартни за индустрията алгоритми за хеширане на пароли, за да защити потребителските идентификационни данни.
- **Удостоверяване и оторизация:** Пакетът се интегрира безпроблемно с механизмите за удостоверяване на ASP.NET Core, като ви позволява лесно да удостоверявате потребителите и да разрешавате достъп до различни части на вашето приложение въз основа на роли и претенции.

**Microsoft.EntityFrameworkCore:**

- **Миграции:** EF Core включва система за миграция, която опростява промените в схемата на базата данни с течение на времето. Миграциите позволяват на разработчиците да

развиват схемата на базата данни с развитието на приложението, като гарантират, че базата данни остава в синхрон с модела на данните на приложението.

- **Запитвания LINQ:** EF Core позволява на разработчиците да пишат LINQ (Language Integrated Query) заявки към базата данни, което осигурява силно типизиран и интуитивен начин за извличане и манипулиране на данни.
- **Интеграция с ASP.NET Core:** EF Core се интегрира безпроблемно с ASP.NET Core, осигурявайки възможности за достъп до данни за уеб приложения, изградени на базата на рамката ASP.NET Core.

**Microsoft.EntityFrameworkCore.SqlServer:**

- **Съвместимост със SQL Server:** Пакетът е специално разработен, за да осигури безпроблемна интеграция между Entity Framework Core и Microsoft SQL Server - една от най-широко използваните системи за управление на релационни бази данни.
- **Интеграция на Entity Framework Core:** EFCore.SqlServer е част от по-широката екосистема Entity Framework Core. Тя разширява основната функционалност на EF Core, за да поддържа специфични за SQL Server функции и оптимизации.
- **Безпроблемна миграция и еволюция на схемите:** Разработчиците могат да използват миграция с Entity Framework Core, за да развиват схемата на базата данни с течение на времето.

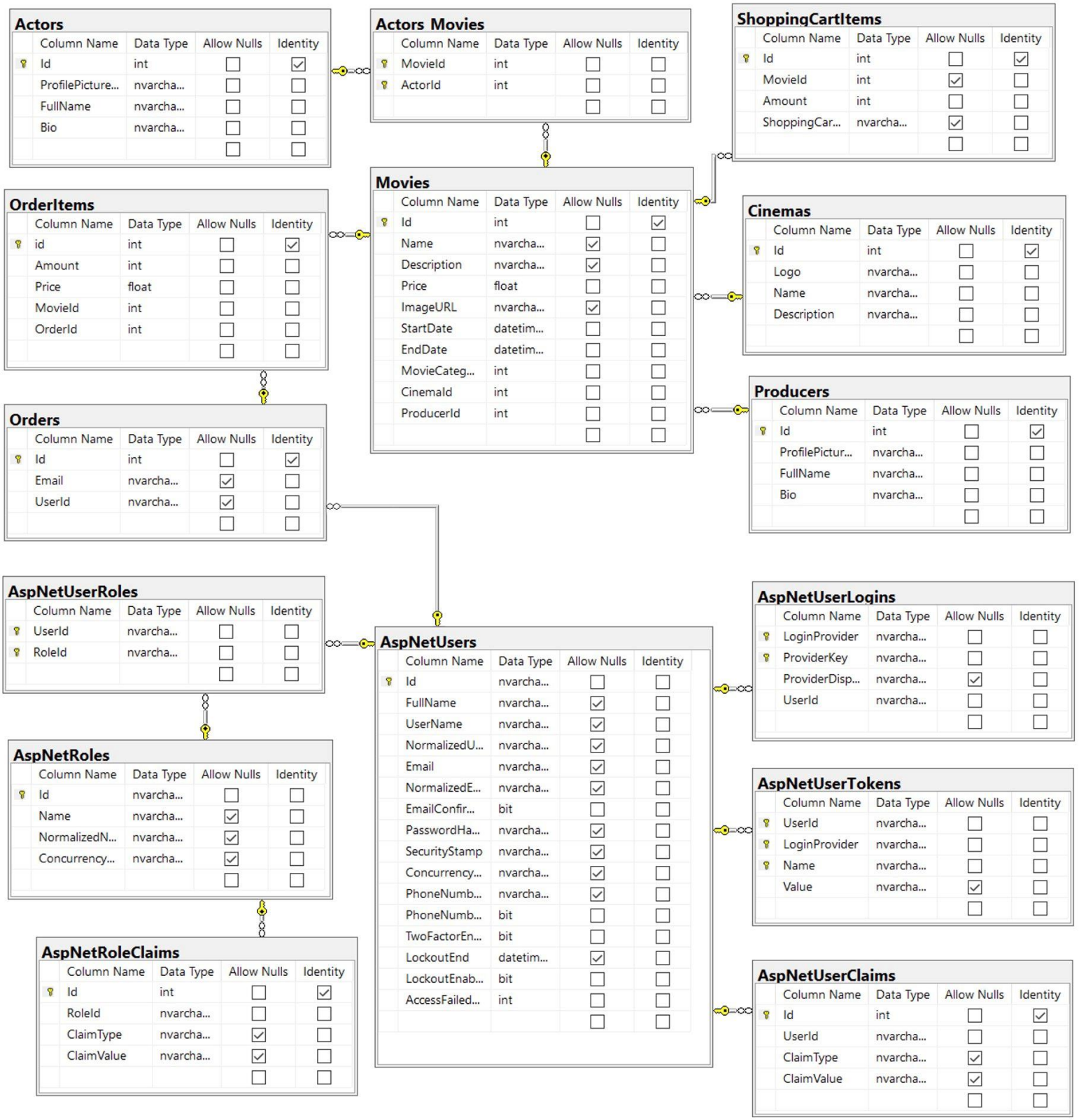
**Microsoft.Entity.FrameworkCore.Tools:**

- **Миграции:** Основното предназначение на EFCore.Tools е да позволява миграция. Миграциите са начин за развитие на схемата на базата данни с течение на времето чрез прилагане на промени в структурата на базата данни по версията.
- **Актуализиране на базата данни:** CLI предоставя команди за актуализиране на схемата на базата данни въз основа на дефинираните миграции. Командата dotnet ef database update (актуализация на базата данни) прилага всички висящи миграции към базата данни, като гарантира, че схемата на базата данни е в синхрон с модела на данните на приложението.

# 5 Структура на приложението

## 5.1 База данни ER диаграма

Приложението съхранява данните в база със следната структура


Actors
Column Name Data Type Allow Nulls Identity
Id int
ProfilePicture... nvarcha...
FullName nvarcha...
Bio nvarcha...
Actors Movies
Column Name Data Type Allow Nulls Identity
MovieId int
ActorId int
ShoppingCartItems
Column Name Data Type Allow Nulls Identity
Id int
MovieId int
Amount int
ShoppingCar... nvarcha...
Movies
Column Name Data Type Allow Nulls Identity
Id int
Name nvarcha...
Description nvarcha...
Price float
ImageURL nvarcha...
StartDate datetim...
EndDate datetim...
MovieCateg... int
CinemaId int
ProducerId int
OrderItems
Column Name Data Type Allow Nulls Identity
id int
Amount int
Price float
MovieId int
OrderId int
Cinemas
Column Name Data Type Allow Nulls Identity
Id int
Logo nvarcha...
Name nvarcha...
Description nvarcha...
Producers
Column Name Data Type Allow Nulls Identity
Id int
ProfilePictur... nvarcha...
FullName nvarcha...
Bio nvarcha...
Orders
Column Name Data Type Allow Nulls Identity
Id int
Email nvarcha...
UserId nvarcha...
AspNetUserRoles
Column Name Data Type Allow Nulls Identity
UserId nvarcha...
RoleId nvarcha...
AspNetUsers
Column Name Data Type Allow Nulls Identity
Id nvarcha...
FullName nvarcha...
UserName nvarcha...
NormalizedU... nvarcha...
Email nvarcha...
NormalizedE... nvarcha...
EmailConfir... bit
PasswordHa... nvarcha...
SecurityStamp nvarcha...
Concurrency... nvarcha...
PhoneNumb... nvarcha...
PhoneNumb... bit
TwoFactorEn... bit
LockoutEnd datetim...
LockoutEnab... bit
AccessFailed... int
AspNetUserLogins
Column Name Data Type Allow Nulls Identity
LoginProvider nvarcha...
ProviderKey nvarcha...
ProviderDisp... nvarcha...
UserId nvarcha...
AspNetRoles
Column Name Data Type Allow Nulls Identity
Id nvarcha...
Name nvarcha...
NormalizedN... nvarcha...
Concurrency... nvarcha...
AspNetUserTokens
Column Name Data Type Allow Nulls Identity
UserId nvarcha...
LoginProvider nvarcha...
Name nvarcha...
Value nvarcha...
AspNetRoleClaims
Column Name Data Type Allow Nulls Identity
Id int
RoleId nvarcha...
ClaimType nvarcha...
ClaimValue nvarcha...
AspNetUserClaims
Column Name Data Type Allow Nulls Identity
Id int
UserId nvarcha...
ClaimType nvarcha...
ClaimValue nvarcha...

## Описание на таблици

### Таблица Producers

**Producers**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | int | ☐ | ☑ |
| | ProfilePictur... | nvarcha... | ☐ | ☐ |
| | FullName | nvarcha... | ☐ | ☐ |
| | Bio | nvarcha... | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на продуцента,
профилна снимка на продуцента,
цялото име на продуцента,
биография на продуцента.

### Таблица Cinemas

**Cinemas**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | int | ☐ | ☑ |
| | Logo | nvarcha... | ☐ | ☐ |
| | Name | nvarcha... | ☐ | ☐ |
| | Description | nvarcha... | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на кино,
лого на кино,
име на кино,
описание на кино.

### Таблица Movies

**Movies**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | int | ☐ | ☑ |
| | Name | nvarcha... | ☑ | ☐ |
| | Description | nvarcha... | ☑ | ☐ |
| | Price | float | ☐ | ☐ |
| | ImageURL | nvarcha... | ☑ | ☐ |
| | StartDate | datetim... | ☐ | ☐ |
| | EndDate | datetim... | ☐ | ☐ |
| | MovieCateg... | int | ☐ | ☐ |
| | CinemaId | int | ☐ | ☐ |
| | ProducerId | int | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на филма,
име на филма,
описание на филма,
цена на филма,
снимка на филма,
дата започване на прожекция,
дата спиране на прожекция,
жанр на филма,
уникален идентификатор на киното,
уникален идентификатор на продуцента.

**Таблица Actors**

**Actors**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | int | ☐ | ☑ |
| | ProfilePictur... | nvarcha... | ☐ | ☐ |
| | FullName | nvarcha... | ☐ | ☐ |
| | Bio | nvarcha... | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на актьор,
профилна снимка на актьор,
пълно име на актьор,
биография.

**Таблица Actors_Movies**

**Actors Movies**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | MovieId | int | ☐ | ☐ |
| 🔑 | ActorId | int | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на филм,
уникален идентификатор на актьор.

**Таблица ShoppingCartItems**

**ShoppingCartItems**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | int | ☐ | ☑ |
| | MovieId | int | ☑ | ☐ |
| | Amount | int | ☐ | ☐ |
| | ShoppingCar... | nvarcha... | ☑ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на елемента,
уникален идентификатор на филма,
количество,
уникален идентификатор на количката.

**Таблица AspNetUsers**

**AspNetUsers**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | nvarcha... | ☐ | ☐ |
| | FullName | nvarcha... | ☑ | ☐ |
| | UserName | nvarcha... | ☑ | ☐ |
| | NormalizedU... | nvarcha... | ☑ | ☐ |
| | Email | nvarcha... | ☑ | ☐ |
| | NormalizedE... | nvarcha... | ☑ | ☐ |
| | EmailConfir... | bit | ☐ | ☐ |
| | PasswordHa... | nvarcha... | ☑ | ☐ |
| | SecurityStamp | nvarcha... | ☑ | ☐ |
| | Concurrency... | nvarcha... | ☑ | ☐ |
| | PhoneNumb... | nvarcha... | ☑ | ☐ |
| | PhoneNumb... | bit | ☐ | ☐ |
| | TwoFactorEn... | bit | ☐ | ☐ |
| | LockoutEnd | datetim... | ☑ | ☐ |
| | LockoutEnab... | bit | ☐ | ☐ |
| | AccessFailed... | int | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на потребителя,
пълно име,
потребителско име,
нормализирано на потребителско име,
имейл,
нормализиран имейл,
хеш парола,
печат за сигурност,
печат за съгласуваност (concurrency),
телефонен номер,
потвърден тел. номер,
активиран двуфакторен достъп,
край на блокировката,
активирана блокировка,
брой неуспешни опити за достъп.

**Таблица Orders**

**Orders**

| Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|
| Id | int | ☐ | ☑ |
| Email | nvarcha... | ☑ | ☐ |
| UserId | nvarcha... | ☑ | ☐ |
| | | ☐ | ☐ |

уникален идентификатор на поръчка,
имейл на купувача,
уникален идентификатор на купувача.

**Таблица OrderItems**

**OrderItems**

| Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|
| id | int | ☐ | ☑ |
| Amount | int | ☐ | ☐ |
| Price | float | ☐ | ☐ |
| MovieId | int | ☐ | ☐ |
| OrderId | int | ☐ | ☐ |
| | | ☐ | ☐ |

уникален идентификатор на елемента,
количество на този елемент,
крайна цена (цена на филма * количество),
уникален идентификатор на филма,
уникален идентификатор на поръчка.

**Таблица AspNetUserLogins**

**AspNetUserLogins**

| Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|
| LoginProvider | nvarcha... | ☐ | ☐ |
| ProviderKey | nvarcha... | ☐ | ☐ |
| ProviderDisp... | nvarcha... | ☑ | ☐ |
| UserId | nvarcha... | ☐ | ☐ |
| | | ☐ | ☐ |

доставчик на вход,
ключ за доставчик,
display-но име на доставчик,
уникален идентификатор на потребител.

**Таблица AspNetUserTokens**

**AspNetUserTokens**

| Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|
| UserId | nvarcha... | ☐ | ☐ |
| LoginProvider | nvarcha... | ☐ | ☐ |
| Name | nvarcha... | ☐ | ☐ |
| Value | nvarcha... | ☑ | ☐ |
| | | ☐ | ☐ |

уникален идентификатор на потребителя,
доставчик на вход,
име на жетон,
стойност.

**Таблица AspNetUserClaims**

**AspNetUserClaims**

| Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|
| Id | int | ☐ | ☑ |
| UserId | nvarcha... | ☐ | ☐ |
| ClaimType | nvarcha... | ☑ | ☐ |
| ClaimValue | nvarcha... | ☑ | ☐ |
| | | ☐ | ☐ |

уникален идентификатор на претенция,
уникален идентификатор на потребителя,
тип претенция,
стойност на претенцията.

**Таблица AspNetRoles**

**AspNetRoles**

| Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|
| Id | nvarcha... | ☐ | ☐ |
| Name | nvarcha... | ☑ | ☐ |
| NormalizedN... | nvarcha... | ☑ | ☐ |
| Concurrency... | nvarcha... | ☑ | ☐ |
| | | ☐ | ☐ |

уникален идентификатор на ролята,
име на ролята,
нормализирано име,
печат съвместимост.

**Таблица AspNetRoleClaims**

**AspNetRoleClaims**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | Id | int | ☐ | ☑ |
| | RoleId | nvarcha... | ☐ | ☐ |
| | ClaimType | nvarcha... | ☑ | ☐ |
| | ClaimValue | nvarcha... | ☑ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на претенция,
уникален идентификатор на роля,
тип претенция,
стойност на претенцията.

**Таблица AspNetUserRoles**

**AspNetUserRoles**

| | Column Name | Data Type | Allow Nulls | Identity |
|---|---|---|---|---|
| 🔑 | UserId | nvarcha... | ☐ | ☐ |
| 🔑 | RoleId | nvarcha... | ☐ | ☐ |
| | | | ☐ | ☐ |

уникален идентификатор на потребителя,
уникален идентификатор на роля.

## 5.2 Структура на кода

**Controllers:**

- **`AccountController`** - използва се за управление на акаунти и автентикация на потребителите. Този контролер извършва операции, свързани с акаунти и автентикация, като вход, регистрация, изход и управление на правата за достъп. Освен това, той използва Entity Framework за връзка с базата данни и Identity системата за управление на потребителите и техните роли.
- **`ActorController`** - служи за управление на данните относно актьорите в системата. Този контролер предоставя пълен CRUD (Create, Read, Update, Delete) функционалности за управление на данните за актьорите, като осигурява правилно управление на правата за достъп и валидация на данните.
- **`CinemasController`** - инициализира контролера с услуга за управление на данните за кината (ICinemasService). Този контролер предоставя основни операции за управление на данните за кината във вашия система за електронни билети, включително добавяне, редактиране и изтриване, и контролира достъпа до тях съгласно ролевите права.
- **`MoviesController`** - управлява операциите за управление на данните за филмите. Контролерът предоставя основни операции за управление на данните за

филмите и се грижи за обработка на данните и осигуряване на стойности за изгледите.

- **`OrdersController`** - управлява операциите, свързани с поръчките и пазарската кошница (shopping cart). Контролерът предоставя функционалности за добавяне и премахване на елементи от пазарската кошница, преглеждане на съдържанието на кошницата и завършване на поръчката. Всички тези операции са защитени чрез атрибута [Authorize], за да бъдат достъпни само за автентифицирани потребители.
- **`ProducersController`** - управлява операциите за управление на данните за продуцентите. Контролерът предоставя основни операции за управление на данните за продуцентите и се грижи за обработка на данните и потвърждаване преди изтриване.

## Data

### Base

- **`EntityBaseRepository`** - представлява обобщен клас за извършване на базови операции с обекти от тип, който имплементира интерфейса IEntityBase. Този клас предоставя основните CRUD (Create, Read, Update, Delete) операции за обекти от тип T в контекста на базата данни. Методите са асинхронни и се грижат за манипулацията с данните в базата.
- **`IEntityBase`** - дефинира едно свойство: **Id**, типът на свойството е int, задава се и се получава/връща.
- **`IEntityBaseRepository`** - задава контракти за класове, които оперират с обекти от тип T. Дефинира основни CRUD (Create, Read, Update, Delete) операции, а също така и методи за извличане на обекти с включени свойства. Този интерфейс служи като контракт, който трябва да бъде изпълнен от класове (репозитории), отговарящи за взаимодействие с базата данни за обекти от тип T.

### Cart

- **`ShoppingCart`** - представлява модел на кошница за пазаруване.

### Enums:

- **`MovieCategory`** - класът представлява enum (преизчисление) с име

MovieCategory. Преизчисленията в C# представляват типове данни, които позволяват на програмистите да декларират именувани константи.

## Models

- **Actor** – представлява модел за актьор в системата.
- **Movies** – модел за филм в системата.
- **Actor_Movies** – представлява връзка между актьор и филм в системата. Този клас улеснява установяването на връзката между актьор и филм във връзката "много към много".
- **Cinema** – модел за кино в системата.
- **Producer** – модел за продуцент в приложението.
- **ApplicationUser** – модел, който наследява стандартния клас IdentityUser от ASP.NET Core Identity.
- **ErrorViewModel** – модел за грешка в приложението.
- **Order** – модел за поръчките в приложението.
- **OrderItem** – модел за отделен елемент от поръчка в приложението.
- **ShoppingCartItem** – модел за елемент от кошницата за пазаруване в приложението.

## Services

- **IActorsService** - наследява интерфейса IEntityBaseRepository<Actor>. Проверява дали даденият актьор има свързани филми.
- **ActorsService** - служебен клас за управление на данни, свързани с актьори. Този клас използва базовия клас EntityBaseRepository<Actor> и реализира интерфейса IActorsService.
- **ICinemasService** - наследява интерфейса IEntityBaseRepository<Cinema>. Проверява дали даденото кино има свързани филми.
- **CinemasService** - служебен клас за управление на данни, свързани с кина. Този клас използва базовия клас EntityBaseRepository<Cinema> и реализира интерфейса ICinemasService.
- **IProducersService** - наследява интерфейса IEntityBaseRepository<Producer>. Проверява дали даденият продуцент има свързани филми.

- **ProducersService** - реализира интерфейса IProducersService и предоставя конкретна логика за управление на данните за продуцентите (модели от тип Producer) в системата. Наследява EntityBaseRepository<Producer>.
- **IMoviesService** - дефинира контракт за управление на данни за филмите (модели от тип Movie). Наследява IEntityBaseRepository<Movie>.
- **MoviesService** - представлява реализация на интерфейса IMoviesService и наследява базовия клас EntityBaseRepository<Movie>.
- **IOrdersService** - дефинира два метода, свързани с обработката на поръчки в контекста на системата.
- **OrdersService** - предоставя конкретна реализация на интерфейса IOrdersService и предоставя функционалности за обработка на поръчки.

## Static

- **UserRoles** – статичен клас, който дефинира константи за ролите в системата.

## ViewComponents

- **ShoppingCartSummary** – представлява ASP.NET Core View Component, който се използва за генериране на сумирано представяне на броя елементи в пазарната количка.

## ViewModels

- **LoginVM** – модел (ViewModel) за данните, свързани с входа в системата (login)
- **RegisterVM** – слузи за регистрация на потребител. Съдържа атрибути, които се ползват за представяне на информация, необходима при регистрация на потребител.
- **ShoppingCartVM** – ViewModel за кошницата. Ползва се за представяне на информация свързана със съдържанието на пазарската кошница в контекста на уеб приложението.
- **NewMovieVM** – служи за представяне на данните, свързани със създаването или редактирането на нов филм.

- **NewMovieDropdownsVM** – съдържащ списъци от обекти от типове Producer, Sinema, Actor.

## Views

### Account

- **AccessDenied** – използва се за показване на съобщение за недопустим достъп. В този случай съобщението е "You are not allowed to access this page". Потребителят се насочва към началната страница чрез бутона "Home Page".
- **Login** – елементарен интерфейс за влизане в системата със стандартни полета за електронна поща и парола, както и възможност за отказ и препращане към началната страница.
- **Register** – страница за регистрация на нов потребителски профил. Има полета за име, имейл, парола и потвърждаване на парола.
- **RegisterCompleted** – информационна страница, която се показва на потребителя след успешно създаване на акаунт. Съдържа заглавие, което съобщава на потребителя, че акаунтът е създаден успешно.
- **Users** – страница, която показва списък от всички потребители в табличен формат. Съдържа таблица с пълно име, потребителско име и имейл на всеки потребител, регистриран в системата.

### Actors

- **Create** – страница за добавяне на нов актьор в системата.
- **Delete** – страница за изтриване на актьор от системата.
- **DeleteError** – страница за грешка при триене на актьор (когато е свързан с филм).
- **Details** – страница, която показва информация за актьор (име, биография, …).
- **Edit** – страница за редактиране на актьор.
- **Index** – показва лист с всички актьори в системата.

### Cinemas

- **Create** – страница за добавяне на ново кино в системата.
- **Delete** – страница за изтриване на кино от системата.
- **DeleteError** – страница за грешка при триене на кино (когато е свързано с филм).
- **Details** – страница, която показва информация за кино (име, описание, …).
- **Edit** – страница за редактиране на кино.
- **Index** – показва лист с всички кина в системата.

**Movies**

- **Create** – страница за добавяне на нов филм в системата.
- **Details** – страница, която показва информация за филм (актьори, кино, …).
- **Edit** – страница за редактиране на филм.
- **Index** – начало на сайта; лист с всички филми в системата.

**Orders**

- **Index** –показва лист с всички поръчки в панела на администратора; всички поръчки направени от този профил на потребителя.
- **OrderCompleted** – страница, която уведомява потребителя, че поръчката е завършена.
- **ShoppingCart** – страница, която показва количката и нейното съдържание.

**Producers**

- **Create** – страница за добавяне на нов продуцент в системата.
- **Delete** – страница за изтриване на продуцент от системата.
- **DeleteError** – страница за грешка при триене на продуцент (когато е свързан с филм).
- **Details** – страница, която показва информация за продуцент (име, биография, …).
- **Edit** – страница за редактиране на продуцент.

- **Index** – показва лист с всички продуценти в системата.

**Shared**

- **Components/ShoppingCartSummary/Default** – част от навигационното меню; показва броя на елементите в кошницата за пазаруване.
- **_CreateItem** – създава бутон "Add New", който се показва само ако потребителят е аутентикиран и има ролята "Admin". Бутона е позициониран в долния десен ъгъл на страницата и има зелен стил..
- **_Identity** – част от навигационното меню; показва бутони "login" и "register" или "hello-user" и "logout" спрямо дали потребителят е влезнал в профил или не.
- **_Layout** – съдържа общи елементи на уеб страница, като навигационна лента, заглавие, footer и включва стилове и скриптове от wwwroot/lib и wwwroot/js.
- **_ValidationScriptsPartial** – включва две библиотеки за валидация в JavaScript: jQuery Validation и jQuery Unobtrusive Validation.
- **Error** – използва се за представяне на информацията за грешка в уеб приложение, когато се получи HTTP грешка.
- **NotFound** – показва съобщение със заглавие "You've reached the end of the internet", следвано от хоризонтална линия и бутон "Home Page", който връща потребителя към началната страница на контролера "Movies"..

**_ViewImports** – блок от директиви във вю файла на ASP.NET MVC приложение. Добавя някои пространства от имена (namespaces) и таг помощници, които се използват в проекта.

**_ViewStart** – директива в ASP.NET MVC вю файла служи за определяне на макета (layout), който да бъде използван за този конкретен изглед.

**AppDbContext** – контекстът на базата данни за приложението, който наследява IdentityDbContext<ApplicationUser> и съдържа свойства за моделите, които са представени в базата данни.

**AppDbInitializer** – служи за инициализация на базата данни с начални данни и потребители при стартирането на приложението.

**appsettings** – Конфигурационният файл; съдържа настройки, които приложението използва. Съдържа настройки за конфигурирането на връзката с база данни и нивата на логване.

**Program** – основния клас Program в ASP.NET Core приложение. Настройва и стартира всички основни компоненти на приложението.

**Startup** – конфигурационната информация за приложението. В конкретния код са конфигурирани различни аспекти, като базата данни, услуги, аутентикация и авторизация. Настройва основните компоненти и конфигурации за работата на приложението.

## 5.3 Функционалност

**Клиентската част предоставя следните функционалности:**

Преглед на всички филми, преглед на детайли на филми, потребителска кошница, поръчка на билети(след регистрация и автентикация), преглед на поръчки.

**Административния панел** е достъпен с потребителско име и парола. Предлага следните възможности на администратора:

Добавяне на нов филм, продуцент, актьор и кино; редактиране на филм, продуцент, актьор и кино; изтриване на продуцент, актьор и кино; преглед на всички акаунти в системата и на всички поръчки в системата.

# 6 Заключение

В заключение, създаването и представянето на **eTickets** за купуване на билети за филми представлява иновативен подход към обогатяване на кинематографичното преживяване на потребителите. Проектът не само адресира нарастващата потребност от удобство и бързина във връзка с покупката на билети, но и интегрира технологии и функционалности, които подобряват общия потребителски опит.

Със своите функционалности за резервации и известия за предстоящи събития, **eTickets** не само опростява процеса на купуване на билети, но и подчертава ангажимента си към усъвършенстването на кинематографичния опит на своите потребители.

Накрая, **eTickets** не е просто платформа за онлайн продажба на билети, а по-скоро виртуално пространство, което свързва любителите на филмите и ги води във вълнуващо и персонализирано пътуване през света на изкуството. Този проект представлява иновация в

сферата на киното, като отваря вратата към по-удобен и вълнуващ начин за изживяване на любимите филми.

С цел непрекъснато усъвършенстване и отговаряне на нарастващите изисквания на потребителите, **eTickets** може да бъде значително подобрен чрез добавянето на следните функционалности в бъдещи ъпдейти:

- **Разнообразни начини на плащане:** Добавянето на възможност за плащане с различни видове дебитни и кредитни карти ще предостави на потребителите по-голяма гъвкавост и удобство при избора на начин на плащане. Интегрирането на сигурни и утвърдени платежни системи ще подобри цялостното потребителско изживяване и ще улесни процеса на закупуване на билети

- **Добавяне на час на прожекция: Включването на функционалност за часове на прожекция е предназначено да подобри информативността и лесното достъпване на данни за нашите потребители. Това нововъведение предоставя по-детайлна и точна информация за времето на прожекцията, предоставяйки допълнителна яснота и удобство при избора на подходящия момент за гледане на филма.**

- **Интерактивно избиране място и ред в залата:** Допълнително подобрение може да бъде постигнато чрез въвеждането на функционалност, която позволява на потребителите да избират конкретни места и редове в залата при резервацията на билети. Това ще предостави персонализирано преживяване и ще отговори на предпочитанията на потребителите, които предпочитат конкретни места в кинозалата

- **Прикачване на снимки вместо линк към снимка в админ панела**: Добавянето на тази функционалност не само улеснява администрирането, но и подобрява бързината и ефективността на добавянето на снимки към събития, филми или други части от сайта. Сега администраторите могат да качват и актуализират съдържание с по-малко стъпки, като по този начин предоставят на потребителите още по-бърз и персонализиран достъп до визуална информация.

- **Прикачване на снимки от потребителите в сайта:** Интегрирането на възможност за качване на снимки от устройството на потребителя може да направи **eTickets** още по-личен и интерактивен. Потребителите могат да споделят моменти от своето кинематографично преживяване, като качват снимки от събитията, които посещават. Това ще подпомогне за по-силно ангажиране на общността и създаване на по-близки връзки с потребителите.

С тези допълнителни функционалности **eTickets** не само ще подобри своята функционалност, но и ще отговори на разнообразните нужди и предпочитания на широка аудитория от любители

на филмите. Това ще го направи не просто за място за покупка на билети, а за център на иновации в областта на онлайн киното.

# 7 Ресурси и използвана литература


https://www.w3schools.in/types-of-network-protocols-and-their-uses

https://www.w3schools.com

(https://www.w3schools.com/tags/ref_httpmethods.asp

https://www.w3schools.com/tags/ref_byfunc.asp

https://www.w3schools.com/css/css_intro.asp

https://www.w3schools.com/tags/ref_byfunc.asp

https://www.w3schools.com/html/html_responsive.asp

https://www.w3schools.com/bootstrap/bootstrap_get_started.asp

https://www.w3schools.com/html/html5_semantic_elements.asp)

https://developer.mozilla.org

(https://developer.mozilla.org/en-US/docs/Web/HTTP

https://developer.mozilla.org/en-US/docs/Web/CSS

https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design

https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

https://developer.mozilla.org/en-US/docs/Web/JavaScript)

https://www.ssl.com/faqs/what-is-https/

https://www.techtarget.com

(https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol

https://www.techtarget.com/searchsoftwarequality/definition/HTTPS

https://www.techtarget.com/whatis/definition/Model-View-ViewModel#:~:text=MVVM%20is%20also%20known%20as,of%20code%20simpler%20and%20faster)

https://www.freecodecamp.org

(https://www.freecodecamp.org/news/what-is-https-http-vs-https-meaning-and-how-it-works/

https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/)

https://www.keyfactor.com/blog/http-vs-https-whats-the-difference/

https://www.geeksforgeeks.org

(https://www.geeksforgeeks.org/mvc-framework-introduction/

https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

https://www.geeksforgeeks.org/mvc-design-pattern/

https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/)

https://www.ibm.com

(https://www.ibm.com/topics/three-tier-architecture

https://www.ibm.com/topics/three-tier-architecture

https://www.ibm.com/docs/en/i/7.3?topic=programming-server-side-scripting-languages

https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-responses

https://www.ibm.com/docs/en/zos/2.1.0?topic=internets-typical-client-server-program-flow-chart

https://www.ibm.com/docs/en/cics-ts/6.1?topic=programs-clientserver-model)

https://en.wikipedia.org

(https://en.wikipedia.org/wiki/Client_(computing)

https://en.wikipedia.org/wiki/Server_(computing)

https://en.wikipedia.org/wiki/Client%E2%80%93server_model

https://en.wikipedia.org/wiki/CSS

https://en.wikipedia.org/wiki/Responsive_web_design

https://en.wikipedia.org/wiki/Unified_Modeling_Language

https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

https://en.wikipedia.org/wiki/Document_Object_Model

https://en.wikipedia.org/wiki/JavaScript)

https://bg.wikipedia.org

(https://bg.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D1%87%D0%B5%D0%BD_HTML

https://bg.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8

https://bg.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%B7%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%BE%D1%82_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8

https://bg.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%81%D0%BB%D0%BE%D0%B9%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0

https://bg.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8)

https://www.quora.com/How-does-client-server-communication-work-in-web-applications

https://kmk.fmi-plovdiv.org/InternetTechnology.pdf

https://learn.fmi.uni-sofia.bg/pluginfile.php/392713/mod_resource/content/2/Web%20%D1%83%D1%81%D0%BB%D1%83%D0%B3%D0%B8.%20GET%20%D0%B8%20POST%20%D0%B7%D0%B0%D1%8F%D0%B2%D0%BA%D0%B8..pdf

https://medium.com

(https://medium.com/before-semicolon/10-html-semantic-tags-and-when-to-use-them-5ae7d7d0b0f2

https://medium.com/@hussainabbas365/mvvm-with-clean-architecture-in-react-native-a-detailed-guide-3ff387944692

https://medium.com/@charithavenkataraju/model-view-controller-mvc-ba6c07dff565

https://medium.com/geekculture/what-is-a-layered-architecture-in-software-design-22152fc06822

https://medium.com/@syantien/clean-architecture-5574cd731e8d

https://medium.com/@deanrubin/the-three-layered-architecture-fe30cb0e4a6

https://medium.com/@9cv9official/what-are-get-post-put-patch-delete-a-walkthrough-with-javascripts-fetch-api-17be31755d28)

https://stackoverflow.com

(https://stackoverflow.com/questions/9563845/semantic-use-of-nav-in-html5-with-search-form-element

https://stackoverflow.com/questions/16634501/in-proper-mvc-does-everything-have-to-be-a-model-view-or-controller

https://stackoverflow.com/questions/8336449/architecture-principles-for-how-to-create-multiple-layers

https://stackoverflow.com/questions/27030649/explain-and-example-about-get-delete-post-put-options-patch-h)

https://dev.to

(https://dev.to/mochafreddo/mastering-mvvm-a-comprehensive-guide-to-the-model-view-viewmodel-architecture-221g

https://dev.to/sardarmudassaralikhan/clean-architecture-used-in-software-development-5gpc

https://dev.to/sardarmudassaralikhan/three-layer-architecture-used-in-software-development-57ji

https://dev.to/qbentil/http-methods-get-post-put-patch-delete-1fhi)

https://www.w3.org/TR/2018/REC-selectors-3-20181106/

https://savovdesign.com/html5-semantichni-elementi-i-polzata-im-za-seo

https://webfe.wordpress.com/html/%D1%81%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D1%87%D0%B5%D0%BD-html/

https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/#:~:text=Responsive%20Web%20design%20is%20the,use%20of%20CSS%20media%20queries.

https://www.qulix.com/about/blog/the-best-server-side-language/

https://www.oracle.com/database/what-is-database/#:~:text=Is%20a%20Database%3F-,Database%20defined,database%20management%20system%20(DBMS).

https://obuch.info

(https://obuch.info/lekcii-po-bazi-danni.html?page=2

https://obuch.info/bazi-danni-v2.html)

http://tru.uni-sz.bg/virtru/Relations.htm

https://codegym.cc/bg/quests/lectures/bg.questhibernate.level17.lecture02

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

https://www.lucidchart.com/pages/er-diagrams

https://www.oreilly.com

(https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html)

https://niteco.com/articles/5-principles-for-software-architecture/

https://www.sciencedirect.com/topics/computer-science/tier-architecture

https://learn.microsoft.com

(https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles)

https://www.codemag.com/article/0705071/Layered-Architecture-Dependency-Injection-and-Dependency-Inversion

https://cw.fel.cvut.cz/old/_media/courses/b6b33ear/lecture-04-architectures-h.pdf

https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1

https://www.toptal.com/software/single-responsibility-principle

https://hackernoon.com/model-view-controller-architecture-pattern-usage-advantages-examples

https://crimsonpublishers.com/prsp/fulltext/PRSP.000505.php

https://www.ramotion.com/blog/what-is-mvvm/

https://www.netguru.com/blog/mvvm-architecture

https://builtin.com/software-engineering-perspectives/mvvm-architecture

https://copyprogramming.com/howto/mvvm-ddd-and-wpf-layered-application-project-structure-guidance

https://www.brcline.com

(https://www.brcline.com/blog/introduction-to-the-layered-architecture-n-tier-architecture

https://www.brcline.com/blog/introduction-to-the-layered-architecture-n-tier-architecture)

https://www.linkedin.com/pulse/decoding-few-technical-jargons-recruiters-frontend-backend-pandey

https://www.tutorialspoint.com

(https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

https://www.tutorialspoint.com/mvvm/index.htm)

https://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained

https://www.guru99.com/mvc-vs-mvvm.html

https://in.indeed.com/career-advice/career-development/difference-between-mvvm-and-mvc

https://startup-house.com/blog/mvvm-vs-mvc-comparison

https://www.hwlibre.com/bg/orm-object-relational-mapping/

https://indico.cern.ch/event/63130/contributions/1225280/attachments/1007198/1432928/BG_AA-bg-1.pdf

https://www.mtc.government.bg/bg/node/4751

https://bg.surveillancepackages.com/difference-between-authentication-and-authorization-4b26

https://www.onelogin.com/learn/authentication-vs-authorization#:~:text=Authentication%20and%20authorization%20are%20two,authorization%20determines%20their%20access%20rights.

https://help.claris.com/archive/help/18/fmp/en/index.html#page/FMP_Help/one-to-one-relationships.html

https://blog.devart.com/types-of-relationships-in-sql-server-database.html#:~:text=So%2C%20what%20is%20one-to,one%20record%20in%20table%201.

https://www.upgrad.com/blog/types-of-keys-in-dbms/

https://www.upgrad.com/

(https://www.upgrad.com/blog/types-of-keys-in-dbms/

https://www.upgrad.com/blog/code-first-approach-in-mvc/)

https://www.upgrad.com/blog/code-first-approach-in-mvc/

https://www.semrush.com/blog/semantic-html5-guide/

https://primeinspire.com/blog/new-semantic-elements-in-html5-with-examples

https://getbootstrap.com

(https://getbootstrap.com/docs/3.4/css/

https://getbootstrap.com/2.3.2/base-css.html

https://getbootstrap.com/docs/4.0/content/typography/)

# 8 Приложения

## 8.1 Source Code на проекта

https://github.com/vaniatodorova/Diploma-Project/tree/main/Source%20code/eTickets

## 8.2 Линк към dump-а на базата данни

https://github.com/vaniatodorova/Diploma-Project/tree/main/Data%20base%20dump

## 8.3 Теоретична разработка – pdf и word файл

https://github.com/vaniatodorova/Diploma-Project/tree/main/Theory

## 8.4 Линк към пълния списък с ресурси организирани по точки

https://github.com/vaniatodorova/Diploma-Project/tree/main/Full%20list%20of%20links

## 8.5 Екрани (screenshots)

https://github.com/vaniatodorova/Diploma-Project/tree/main/Screenshots

## 8.6 Линк към видео презентация на проекта

https://github.com/vaniatodorova/Diploma-Project/tree/main/Link%20to%20video