ТЕХНИЧЕСКИ УНИВЕРСИТЕТ-СОФИЯ

ФИЛИАЛ ПЛОВДИВ

ФАКУЛТЕТ „ ЕЛЕКТРОТЕХНИКА И АВТОМАТИКА‘‘

***ДИПЛОМНА РАБОТА***

**Проектиране и реализация на система за онлайн разплащания и достъп до услуги**

Студент:

Иван Иванов Здравков

Фак. № 354827

Научен ръководител:

гл. ас. Васил Стоянов

Пловдив 2014

Съдържание

[Глава 1. Увод 3](#_Toc399715746)

[1.1. Постановка на задачата 3](#_Toc399715747)

[1.2. Архитектура 3](#_Toc399715748)

[1.3. Използвани сървърни технологии (Back End) 4](#_Toc399715749)

[1.4. Използвани клиентски технологии (Front End) 5](#_Toc399715750)

[1.5. Инструменти за разработка 8](#_Toc399715751)

[Глава 2. Проектиране на елементите на визуалния потребителски интерфейс 9](#_Toc399715752)

[2.1. Многоезичен интерфейс, чрез .resx файлове 9](#_Toc399715753)

[2.1. Razor View Engine 12](#_Toc399715754)

[2.3. Проектиране на \_Layout страница 14](#_Toc399715755)

[2.3.1. Навигация 17](#_Toc399715756)

[2.3.2. Тяло 20](#_Toc399715757)

[2.3.3. Footer 21](#_Toc399715758)

[2.4. Използвани стилове 23](#_Toc399715759)

[2.5. Проектиране на Register страницата за регистрации 25](#_Toc399715760)

[2.6. Проектиране на Login страницата за вход в системата 26](#_Toc399715761)

[2.7. Проектиране на Subscription страницата за извършване на плащания 27](#_Toc399715762)

[2.8. Проектиране на Management страницата за управление на потребители 32](#_Toc399715763)

[2.9. Проектиране на Visits и Payments страниците 36](#_Toc399715764)

[Глава 3. Проектиране на релационната база от данни 37](#_Toc399715765)

[3.1. Релационен модел 37](#_Toc399715766)

[3.2. Identity таблици 37](#_Toc399715767)

[3.3. Таблици 42](#_Toc399715768)

[3.4. Entity-relationship диаграма 52](#_Toc399715769)

[Глава 4. Проектиране и реализация на Web услуги 53](#_Toc399715770)

[4.1. BaseController и EntityHelpers класовете 53](#_Toc399715771)

[4.2. Payment Controller 57](#_Toc399715772)

[4.3. Subscription Controller 64](#_Toc399715773)

[4.4. LogController 66](#_Toc399715774)

[Глава 5. Описание на програмното осигуряване 68](#_Toc399715775)

[5.1. Проект InternalDB 69](#_Toc399715776)

[5.2. Проект InternalDAL 72](#_Toc399715777)

[5.3. Проект Resources 75](#_Toc399715778)

[5.4. Проект WebSite 75](#_Toc399715779)

[Global.asax 76](#_Toc399715780)

[5.5. Проект RFID Reader 80](#_Toc399715781)

[5.6. Приложение Mifare1k 90](#_Toc399715782)

[Глава 6. Ръководство на потребителя 91](#_Toc399715783)

[6.1. Клиент 91](#_Toc399715784)

[6.2. Персонал 95](#_Toc399715785)

[Глава 7. Изводи и приложимост 99](#_Toc399715786)

[Глава 8. Приложения 100](#_Toc399715787)

[8.1. Програмен код 100](#_Toc399715788)

[8.2. Използвана литература 100](#_Toc399715789)

# Глава 1. Увод

## 1.1. Постановка на задачата

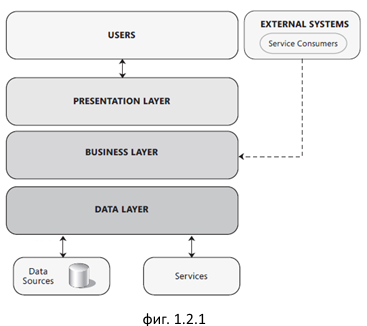
Настоящата дипломна работа има за цел да опише създаването на Web-базирана система за организиране процеса на разплащане и контролиране на достъпа на клиентите на мрежа от спортни зали. Всеки клиент, желаещ да използва системата, след създаване на собствен акаунт и издаване на RFID карта, може да извършва онлайн разплащания, чрез PayPal, закупувайки месечен абонаментен план, осигуряващ му неограничен достъп за периода на ползване или брой кредити, позволяващи му да "заплати" входна такса при всяко влизане в помещенията. Клиентът има право да ползва всяко от помещенията, които са част от системата с равни права.

Самите помещения (Facilities) са самостоятелно обособени и независими спортни зали, всяка от които имат Ръководител (Manager), както и обслужващ персонал, притежаващи акаунти в системата, както и RFID карти с неограничени права на вход.

Входът се осигурява от RFID четец, който управлява механична врата, отваряйки я, ако клиентът има активен месечен план, достатъчно на брой кредити, за да заплати входната такса или се ползва с неограничени права за вход.

Системата пази пълна хронология на събитията, плащанията и посещенията на клиентите.

## 1.2. Архитектура

 Избраната архитектура за разработка е клиент-сървърно MVC (Model View Controller) приложение, с ориентирана към услуги архитектура (фиг. 1.2.1). Това е класическа многослойна архитектура, използвана широко при разработването на уеб приложения. MVC дизайна решава проблема с комуникацията между презентационния слой (View), бизнес логиката (Controller) и слоя за данни (Model). В идеалния случай 3-те слоя трябва да бъдат слабо зависими един от друг, което да позволи лесната замяна на който и да е от компонентите в системата.

Освен класическата MVC архитектура, приложението се възползва от възможностите на услугите (services), за лесна комуникация с 3-ти системи, например PayPal API, осигуряващо ни готова функционалност за извършване на плащания и Yahoo! Finance API, осигуряващо ни актуална информация за валутните курсове, както и за връзка с десктоп конзолно приложение, контролиращо RFID четеца.

Между модела (Data Layer) и базата данни (Data Source) се помества също така междинния слой за достъп до данни (DAL - Data Access Layer). В нашия случай за лесен достъп до данните в базата данни ще използваме ORM модел (Object-Relational Mapping). Работата на ORM модела е да предостави връзка, между иначе несъвместимите модели за описване на данни - релационния и обектния. ORM моделът предоставя интерфейс за работа с данните от базата, като обекти.

1.3. Използвани сървърни технологии (Back End)

Сървърната част от системата е хостната на Web сървър IIS7 и е реализирана чрез Microsoft стека за разработка на уеб приложеня ASP.NET.

* **ASP.NET MVC 5.0** - рамка (framework) за реализирането на MVC частта от многослойната архитектура.
* **ASP.NET Web Api 2.0** - дава възможност за създване на HTTP услуги (services), които могат да бъдат използвани за изнасяне на функционалност към други системи / устройства.
* **ASP.NET Identity 1.0 -** система за управление на потребители (Membership Management System). Осигурява функционалност за управление на потребители, роли, права за достъп и вход в системата, чрез 3-ти системи (Facebook, Twitter и др.)
* **ASP.NET Razor View Engine** - програмен синтаксис, използван за генериране на динамични HTML страници, чрез възможността си да комбинира синтаксиса на HTML и езици от високо ниво (C# или VB), с цел използване на възможностите им за обхождане на колекции, текстообработка и извличане на данни.
* **Entity Framework 6.1.0** - обектно-релационен модел. На база на конкретна инстанция на дадена база данни (най-често MSSQL), създава структура от обекти, които точно отговарят на структурата на базата за лесно използване от останалите приложения. Моделът осигурява още автоматично обновяване на данните, състоящо се от обновяване на модела, при настъпване на структурни промени в базата (Data Base First), или на базата, при настъпване на структурни промени в модела (Code First).
* **MSSQL** - Система за управление на бази от данни (СУБД).
* **C# 5.0** - Си Шарп е типизиран, обектно-ориентиран език за програмиране, изпълняващ се върху виртуална машина (най-често CLR - Common Language Runtime за Windows OS)

## 1.4. Използвани клиентски технологии (Front End)

Клиентската част на системата е разработена с класическия стек за разработна на Web страници и Web приложения (HTML, CSS, JavaScript), както и с използване на jQuery, Modernizr, Bootstrap и Knockout.

* **HTML** **5 (HyperText Markup Language)** - HTML е език за описание на съдържанието и структурата на Web страници. Един HTML документ се състои от елементи, които пък се съставят от отварящ и затварящ таг, като в отварящия таг може да се съдържат атрибути, придаващи някакви свойства на тага.

<tag attribute="стойност"> Съдържание </tag>

Някои HTML елементи нямат затварящи тагове, например **<br/>** (Line break - нов ред ).

В **<head>** елемента се помества мета информацията за страницата, както и зависимостите от CSS и JavaScript файлове и библиотеки. В **<body>** елемента се помества съдържанието на HTML страницата.

Още някои HTML елементи:

<h1> Заглавие </h1>

<p> Параграф </p>

<ol > Подреден списък

<li> Елемент от списъка </li>

<li> Eлемент от списъка </li>

</ol>

<div> Блоков елемент </div>

<span> Елемент, поместен във flow-a на текста </span>

<title> Заглавие на страницата </title>

<a> Линк </a>

<button> Бутон </button>

<style> Елемент, в който поместваме CSS стилове </style>

<script> Елемент, в който поместваме JavaScript код </script>

Пример за HTML документ:

<html>

<head>

<title> Заглавие на страницата </title>

</head>

<body>

</body>

</html>

* **CSS** **(Cascading Style Sheets)** - CSS е език описващ външния вид и форматирането на HTML документ. CSS може да се използва, като вложен (inline) в самия таг на елемента, посредством атрибута **stylе.**

<p style="color: red;">Този текст ще e червен</p>

Може също така да се използва елементът **<style>**, в който да бъдат поместени стиловете, които ще бъдат използвани. Това е полезно, ако например трябва да се опишат стилове за повече от 1 елемент или за елементи, изпълняващи някакво условие.

Пример:

<div>

<p id="paragraph-1" class="red">Параграф 1</p>

<p class="red">Параграф 2</p>

<p class="blue">Параграф 3</p>

</div>

В този случай, ако трябва да се смени цвета на всички <p> елементи, които се намират в <div> елемент на страницата, всичко което е необходимо да се направи е да се добави следния <style> елемент:

<style>

div p {

color: yellow;

}

</style>

Ако трябва да бъде сменен цвета само на елемента с id="paragraph-1" (id атрибута е уникален за цялата HTML страница), то ще бъде използван селектора **#**

<style>

#paragraph-1 {

color: yellow;

}

</style>

Ако трябва да бъдат избрани всички елементи, споделящи даден клас, то ще бъде използван селектора точка.

<style>

.red {

color: red;

}

</style>

Най-добрата практика при използването на CSS е отделянето на стиловете в отделен файл и зaреждането му в <head> елемента на HTML страницата. Това се прави, тъй като CSS стиловете не са динамично съдържание и ако са в отделни файлове могат да бъдат кеширани от браузъра, което подобрява бързодействието на сайтa.

<head>

<link href="path/style.css" rel="stylesheet" type="text/css">

</head>

* **JavaScript** - JavaScript е езикът за програмиране на интернет. Докато HTML се отнася за съдържанието и структурата, а CSS за презентацията на интернет страницата, JavaScript се грижи за поведението. Синтаксисът на JavaScript е много близък до този на C. JavaScript е функционален език, без типов (Typeless), както и прототипно базиран. Задкулисно съществуват 7 вътрешни типа - **number**, **boolean**, **string**, **object**, **array**, **null** и **undifined**, но на практика всички променливи се дефинират с типа **var**. С напредването на технологиите, JavaScript се развива бързо и вече не е само език за DOM манипулация и събития (Events) в една интернет страница. JavaScript е изключително мощен инструмент, с който може да се пише всякакъв вид софтуер. Има начини езикът да се ползва обектно-ориентирано, съществуват и множество библиотеки и рамки (frameworks), които разширяват възможностите му още повече, позволявайки MVC разработки, едностранични приложения (Single Page Applications) и дори игри.
* **jQuery** - jQuery е изключително популярна JavaScript библиотека, целяща да изравни различията в реализацията на JavaScript при различните версии на популярните браузъри. Също се използва широко query селектора му (от където идва и самото име), който използва синтаксис подобен на CSS за да избира елементи от DOM дървото на HTML страницата. jQuery е добър помощник и при AJAX заявки, улеснявайки много процеса, чрез обещания (promises) и улеснен синтаксис.
* **jQuery Validate** - jQuery Validate е добавка (plugin) към jQuery, която се грижи за клиентска форм валидация. Библииотеката е добре интегрирана с ASP.NET MVC5 и посредством валидационни атрибути на моделите и използване на подходящи Razor помощници (helpers), клиентската форм валидация става автоматично. Естествено същите тези атрибути осигуряват валидация на сървъра.
* **Modernizr** - Modernizr е JavaScript библиотека, която при зареждането на страницата, проверява дали браузъра на клиента поддържа определените HTML5/CSS3 пропъртита или функции и ако не ги поддържа, ги добавя. По този начин се решава проблема с "нагаждането" на страниците към стари браузъри.
* **Bootstrap** - Bootstrap e Front End Framework (HTML и CSS, като част от разширенията на Bootstrap използват и JavaScript), осигуряващ така наречения Responsive Design на страницата. С други думи, Bootstrap осигурява на приложението стилистично издържан външен вид, който скалира с намаляне размера на екрана, правейки сайта подходящ и за ползване от мобилни устройства.
* **Knockout** - Knockout е JavaScript библиотека, осигуряваща автоматично свързване на HTML с модел (Declarative Data Binding), което води до автоматично обновяване на модела, гъвкавост при работата с данните и възможност за декларативен HTML.

## 1.5. Инструменти за разработка

* **Visual Studio 2013 -** интегрирана среда за разработка. Съдържа текстов редактор, файлов редактор, builder, debugger, autocomplete за C#, JavaScripт, SQL и други езици, подсказки за пропъртита и методи, тестов интерфейс, NuGet Package Manager (Автоматично управление на библиотеки и пакети), автоматичен publish на сайтове, Source Control система (TFS) и други.
* **SQL Server 2012** - приложение за създаване, конфигуриране и управление на MSSQL бази от данни. Приложението борави както със скриптов редактор, така и с графични инструменти.
* **Fiddler** - Инструмент за прихващане на HTTP и HTTPS заявки. Има възможност да откопирва, променя и изпраща напълно нови заявки.
* **Googe Chrome F12 Developer Tools** - интегрирана в Google Chrome браузъра среда за Front End разработка. Съдържа инспектор на HTML, CSS editor, JavaScript debugger, JavaScript конзола, прозорец с трафик информация и други.

# Глава 2. Проектиране на елементите на визуалния потребителски интерфейс

## 2.1. Многоезичен интерфейс, чрез .resx файлове

Благодарение на Razor View Engine, който ще бъде използван за генериране на HTML страници от даден Model, може много лесно да бъде имплементиран двуезичен или многоезичен интерфейс на разработваното приложение. За целта създаваме проект **Resources** в решението (Solution). Проекта ще служи като хранилище на .resx файлове.

Файловете с разширение resx са ресурсни файлове, които по своята структура са XML файлове, по дизайн са предвидени да са във формат, лесно четим от хора (Human Readable), но все пак използваеми от машините. По конкретно те имплементират Ключ - Стойност принципа, като ключа (data) се явява уникален идентификатор, описващ стойността (value) която трябва да използваме. В допълнение .resx файловете имат и известно количество мета-информация, основно за изобразяване в графични редактори.

Примерна структура на .resx файл:

<data name="Name" xml:space="preserve">

<value>Ivan Zdravkov</value>

</data>

<data name="Country" xml:space="preserve">

<value>Bulgaria</value>

</data>

Удобството при този формат е лесната заменяемост на 1 файл с дадена структура с друг със същата, което може да бъде използвано за осигуряване многоезичност на нашето приложение.

В **Resources** проекта могат да бъдат намерени 2 .resx файла: Resources.resx и Resources.en.resx, съответно за Български (по подразбиране) и Английски (по избор) езици. От тук нататък, всички символни низове (string), които ще бъдат под една или друга форма достъпни за клиента, ще бъдат поместени и достъпвани, чрез тези два файла.

**Resources.resx**

<data name="About" xml:space="preserve">

<value>За нас</value>

</data>

<data name="Account" xml:space="preserve">

<value>Акаунт</value>

</data>

<data name="ApplicationName" xml:space="preserve">

<value>ТУ Фитнес</value>

</data>

**...**

**Resources.en.resx**

<data name="About" xml:space="preserve">

<value>About</value>

</data>

<data name="Account" xml:space="preserve">

<value>Account</value>

</data>

<data name="ApplicationName" xml:space="preserve">

<value>TU Fitness</value>

</data>

...

Избора на това, кой файл да бъде използван при генерирането на HTML страницата, става програмно. Всяка заявка до Web сървъра, преди да бъде обработена минава през обвиващия метод (Wrapper Method) BeginExecutionCore. Това е метод от класа Controller, който се изпълнява преди изпълнението на конкретния метод, рефериран от заявката. Използвайки разширения клас BaseController, наследяваме класа Controller и заместваме (Оverride) функционалността на метода BeginExecutionCore със следната функционалност:

**BeginExecuteCore**

protected override IAsyncResult BeginExecuteCore(AsyncCallback callback, object state)

{

string cultureName = null;

HttpCookie cultureCookie = Request.Cookies["\_culture"];

if (cultureCookie != null)

{

cultureName = cultureCookie.Value;

}

else

{

cultureName = (Request.UserLanguages != null && Request.UserLanguages.Length > 0) ? Request.UserLanguages[0] : null; // obtain it from HTTP header AcceptLanguages

}

cultureName = WebSite.Helpers.CultureHelper.GetImplementedCulture (cultureName);

Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo(cultureName);

Thread.CurrentThread.CurrentUICulture = Thread.CurrentThread.CurrentCulture;

return base.BeginExecuteCore(callback, state);

}

Благодарение на това, всяка заявка към който и да е MVC контролер от тук нататък ще бъде проверявана първо за наличието на ключа "\_cuture" в бисквитката, която бива изпратена със заявката, а след това за AcceptLanguage хедъра на самата HTTP заявка. Използвайки класа CultureHelper, преобразуваме различните формати на културите до един унифициран формат и след това задаваме текущата култура на нишката, която изпълнява заявката на сървъра. Благодарение на този процес и информацията, съхранена в нишката за текущата култура, се избира подходящия .resx файл, отговарящ на текущо избраната ни култура.

**CultureHelper**

public static class CultureHelper

{

private static readonly List<string> ValidCultures = new List<string> { "bg", "en" };

// Include ONLY cultures you are implementing

private static readonly List<string> Cultures = new List<string> { "bg", "en" };

public static string GetImplementedCulture(string name)

{

if (string.IsNullOrEmpty(name))

{

return GetDefaultCulture();

}

if (ValidCultures.Where(c => c.Equals(name, StringComparison.InvariantCultureIgnoreCase)).Count() == 0)

{

return GetDefaultCulture(); // return Default culture if it is invalid

}

if (Cultures.Where(c => c.Equals(name, SringComparison.InvariantCultureIgnoreCase)).Count() > 0)

{

return name;

}

return GetDefaultCulture(); // return Default culture as no match found

}

public static string GetDefaultCulture()

{

return Cultures[0]; // return Default culture

}

public static string GetCurrentCulture()

{

return Thread.CurrentThread.CurrentCulture.Name;

}

public static string GetCurrentNeutralCulture()

{

return GetNeutralCulture(Thread.CurrentThread.CurrentCulture.Name);

}

public static string GetNeutralCulture(string name)

{

if (!name.Contains("-")) return name;

return name.Split('-')[0]; // Read first part only. E.g. "en", "es"

}

}

Тук следва да споменем и екшъна (Action) SetCulture в Home контролера, който бива използван за задаване на желана стойност на културата, която да бъде използвана. Това позволява на клиента да избере езика, който желае, като неговия избор ще бъде запазен, докато бисквитката, издадена от сайта ни е достъпна за ползване от клиентския браузър.

**SetCulture**

public ActionResult SetCulture(string culture)

{

culture = CultureHelper.GetImplementedCulture(culture);

HttpCookie cookie = Request.Cookies["\_culture"];

if (cookie != null)

cookie.Value = culture; // update cookie value

else

{

cookie = new HttpCookie("\_culture");

cookie.Value = culture;

cookie.Expires = DateTime.Now.AddYears(1);

}

Response.Cookies.Add(cookie);

return Redirect(Request.UrlReferrer.AbsoluteUri);

}

## 

## 2.1. Razor View Engine

Както стана ясно, за генериране на HTML страници ще бъде използван Razor View Engine. Този енджин предоставя възможност да бъдат смесвани HTML синтаксис със C# код, като в допълнение, предоставя някои удобни помощни средства (Helpers), като например възможност да бъде използван Html класа и неговите методи за генерираме на HTML код, бързо и лесно, по подобие на създаването на обект и извикването на негов конструктор.

@Html.ActionLink(Resources.VisitsLog, "Visits", "Log")

Този ред например би продуцирал следните HTML конструкции в изходния .html файл:

<a href="/Log/Visits">Visits Log</a>

или

<a href="/Log/Visits">Посещения</a>

Както се вижда, конкретният метод ActionLink на класа Html, приема 3 параметъра:

* Текстът, който ще бъде поставен в тялото на <a> елемента. В конкретния случай, в зависимост от избраната култура, този текст отговаря на VisitLog ключа от ресурсния проект (Resources).
* Екшъна (Action), който да бъде извикан от контролера
* Контролера, към който да се обърнем

Тези 3 параметъра формират href атрибута на <a> елемента, който отговаря на релативния път до екшъна на конкретния контролер. Естествено следва въпросът, дали това не е прекалено сложно и кому е нобходим този синтаксис, но именно генерирането на динамичен HTML, чрез програмна логика позволява да имаме специфичен HTML документ, който да удовлетворява дадени условия.

Например:

@if (Request.IsAuthenticated)

{

<li>@Html.ActionLink(Resources.VisitsLog, "Visits", "Log")</li>

<li>@Html.ActionLink(Resources.PaymentsLog, "Payments", "Log")</li>

}

Елементите в кода в къдравите скоби ще се изпълни, само ако заявката е изпратена от ауторизиран потребител. Двата <a> елемента, които този код би генерирал, няма да съществуват на HTML страницата на потребител, който не е ауторизиран с потребителско име и парола.

Може да прави впечатление привидно невъзможното смесване на HTML елементи с програмна логика. Тук се намесва "@" ситаксисът. Всеки израз, предшестван непосредствено от символа @ се интерпретира като C# код. Един израз може да бъде например '@DateTime.Today.ToString()', ако кодът е непрекъснат от интервали, '@(string text = "Hello")', код заграден със скоби, предшестван от @, ако кодът е прекъснат от интервали или блок, заграден с { }, например следващ @if или @for. View Enginе-a е достатъчно добре разработен, за да различава успешно HTML елементите и да ги комбинира с програмната логика.

Razor предоставя и някои удобни помощни класове (Helpers), които помагат при генерирането на HTML страници. Освен вградените, могат да бъдат дефинирани и собствени. Някои от по известните помощни класове са:

* Html Helper
* Web Grid
* Web Graphics
* Google Analytics
* Facebook Integration
* Twitter Integration
* Sending Email
* Validation

В текущата дипломна работа използваме основно Html хелпъра, косвено и Validation, осигуряващ валидация на входните данни, за която удобно се грижат валидационните атрибути на моделите, комбинирани с jQuery Validate библиотеката, чрез DisplayFor() методите на Html помощника.

## 2.3. Проектиране на \_Layout страница

За да се говори за \_Layout страницата и нейното значение за проектирането на дизайна, първо трябва да се разясни идеологията на частичните изгледи (Partial View).

Partial View е частичен HTML. Той няма <html>, често и <body> елементи. Най-често частичния изглед е просто някаква идеологично цяла HTML структура, която в последствие бива използвана за да се "инжектира" в други страници, като готов шаблон (Template). На този принцип има изградени цели рамки (Frameworks), като например Asp.Net Web Forms, чиито форми, могат да бъдат разглеждани като своеобразни частични изгледи.

Позовавайки се на този принцип ще бъде изграден така наречения главен изглед (Layout или Master Page). Главния изглед представлява HTML документ, в който има оставено специално обозначено място (Place Holder), най-често <div> елемент, където останалите частични изгледи да поместят своето съдържание. По този начин ще съществува едно централизирано място, където ще се помещава основния външен вид, като според извикваната страница единствено ще бъде променяно съдържанието му.

Тъй като ще бъде използван Razor View Engine съществува лесен механизъм, по който да бъде реализирана тази идея. Всеки Razor View Engine документ е файл с .cshtml разширение (cs - oт разширението, което CSharp файловете имат и html - от разширението на HTML документите) има достъп до някои свойства, които могат да бъдат използвани директно. Пример за такова свойство е **Layout** пропъртито. Това е текстово свойство, стойността на което показва кой файл да бъде използван като главен изглед (Layout).

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

По конвенция, файл с наименование \_ViewStart.cshtml в главната директория с всички изгледи, добавя съдържанието си преди тяхното, при всяко тяхно извикване. Само горните 3-реда, добавени в \_ViewStart.cshtml са всичко от необходимо, за да се осигури, че файлът \_Layout.cshtml в Shared директорията от тук нататък ще бъде използван като главен изглед.. Естествено този изглед може да бъде променян програмно по всяко време, например за мобилен изглед, ако екранът е прекалено малък или за различни теми, ако сайтът поддържа такава функционалост.

Преди да се фокусираме върху \_Layout файла, за да бъде проследено неговото изграждане и как бързо, благодарение на Razor и Bootstrap библиотеката, може да се изгражда откликващ (Responsive) дизайн, трябва да се обърне внимание на следния клас, единствения метод на който бива извикван при самото стартиране на приложението от Application\_Start() метода. Този клас има за цел да групира CSS и JS библиотеките, които ще бъдат използвани в така наречените бъндъли (bundles), които по-нататък ще се използват в .cshtml файловете за зареждане на библиотеките.

**BundleConfig**

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js",

"~/Scripts/respond.js"));

bundles.Add(new ScriptBundle("~/bundles/knockout").Include(

"~/Scripts/knockout-3.2.0.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css",

"~/Content/social-buttons-3.css"));

}

След като вече са подготвени бъндълите за работа и е настроен \_ViewStart.cshtml файлът е време да бъде създаден \_Layout.cshtml файла. Първоначално бива изградена общата структура на документа, както и зареждането на необходимите CSS и JavaScript библиотеки.

@using Microsoft.AspNet.Identity

@using WebSite.Models.Subscription

@using WebSite.Controllers

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>@ViewBag.Title - @Resources.ApplicationName</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

...

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

Кодът, изгждащ страницата има класическа HTML5 структура. Обръщаме внимание на

@using Microsoft.AspNet.Identity

@using WebSite.Models.Subscription

@using WebSite.Controllers

Тези 3 реда показват кои именувани колекции класове (namespaces) ще бъдат използвани в страницата. <head> елементът е изграден по малко по-особен начин. Обръщаме внимание първо на <title> елемента. @ViewBag обектът е общ обект от тип KеyValuePair за контролера и неговите изгледи (Views). Този обект е заобиколен начин да се предават данни между изгледа и контролера, но не е основния. Подаването на модел от контролера към изгледа е предпочитания начин по който се осъществява комуникацията на данни мъжду контролера и изгледа. Вече беше описан и @Resources проектът, който се грижи за двуезиочността на приложението. В случая @Resources.ApplicationName ще бъде заместено при Build на HTML страницата със "ТУ Фитнес" или "TU Fitness", съответно за избрана българска или английска култура.

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

Останалата част на <head> елемента сa @Styles и @Scripts помощниците (Helpers), които респективно се грижат за зареждане на CSS стиловете и JavaScript библиотеките, които вече са "пакетирани" в бъндъл от BundleConfig класа.

Следва <body> елементът. Неговата структура ще бъде разгледана след малко. За сега е важно да се отбележи зареждането на jquery бъндъла, bootstrap бъндъла,

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

както и зареждането на секцията

@RenderSection("scripts", required: false)

със скриптове от изгледа, което ще използва изграждания Layout. Зареждането на JavaScript библиотеките и допълнителните скриптове става в края на <body> елемента, след като останалата част от HTML-a вече е заредена.

Съдържанието на <body> елементът може да бъде структурно разделено на 3 части:

* Навигация - Вертикална лента в най-горната част на екрана, заемаща цялата ширина и поместваща в себе си навигацията на сайта, както и контролите за потребителски вход.
* Тяло (Container) - Контейнерът е центрирано бяло платно с определена ширина, в което се поместват всички елементи на страницата, заредени в Layout от изгледа (View), който го използва.
* Footer - На практика част от Container елемента, футърът е бяла хоризонтална лента, с ширината на Container, която помества в себе си информация за сайта, контролите за избор на език, както и за регистрирани потребители, показателите за статуса на техния абонамент и количеството кредити с които разполагат.

### 2.3.1. Навигация

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink(Resources.ApplicationName, "Index", "Home", null, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

@if (this.User.IsInRole("Client"))

{

<li>@Html.ActionLink(Resources.About, "About", "Home")</li>

<li>@Html.ActionLink(Resources.Contact, "Contact", "Home")</li>

@if (Request.IsAuthenticated)

{

if (this.User.IsInRole("Client"))

{

<li>@Html.ActionLink(Resources.VisitsLog, "Visits", "Log")</li>

<li>@Html.ActionLink(Resources.PaymentsLog, "Payments", "Log")</li>

}

if(this.User.IsInRole("Administrator"))

{

<li>@Html.ActionLink(Resources.AdminPanel, "Index", "Admin")</li>

}

if (this.User.IsInRole("Administrator") || this.User.IsInRole("Facility Manager") || this.User.IsInRole("Receptionist"))

{

<li>@Html.ActionLink(Resources.ManagerPanel,

"Index", "Manage")</li>

}

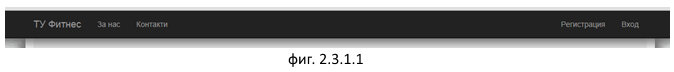
}

</ul>

@Html.Partial("\_LoginPartial")

</div>

Класовете, които са окичили елементите в тази секция са предефинирани от bootstrap библиотеката. Те имат за цел да оформят както външния вид на навигационния бар, така и неговата откликност (responsiveness). Навигационният бар при широк екран изпълва цялата ширина на екрана. (фиг. 2.3.1.1)



Ако обаче сайта бива достъпен през телефон или друг по-малък екран, навигацията автоматично се презарежда, за да задоволи нуждите на устройството. (фиг. 2.3.1.2)



<div> елементът, окичен с navbar-header класа, се явява горната част на разпъващото се меню при мобилния изглед, като неговите елементи са, съответно бутонът за разтваряне на менюто, 3-те <span> елемента, които изрисуват бутона, както и <a> елементът, който се явява Home бутон за цялото приложение, сочейки към Index екшъна на Home контролера, и според езика се нарича "Ту Фитнес" или "Tu Fitness". Вижда се, че на широкоекранния изглед, бутонът не се изобразява, но Home линкът го има, изобразен като обикновен линк от навигацията.

Останалият код е обясним сам по себе си. Имаме 2 допълнителни линка за 2 страници, както и 2 линка, които обаче са видими само, ако потребителят е аутентикиран пред системата с потребителското си име и парола и е от роля Client. Тяхната роля е да го навигират до страниците с информацията за неговите посещения и плащания. Ако потребителят е от роли Administrator и Manager, за него съответно има линкове към страниците за администриране и управление на системата. Благодарение на @ синтаксиса на Razor View Engine, може да бъде използвана прогмната логика, която да проверява Request хедъра за аутентикационните данни и да изобрази линковете, ако данните са налични.

@Html.Partial("\_LoginPartial")

Този ред зарежда допълнителен частичен изглед, който предстои да бъде разгледан:

**\_LoginPartial.cshtml**

@using Microsoft.AspNet.Identity

@if (Request.IsAuthenticated)

{

using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm", @class = "navbar-right" }))

{

@Html.AntiForgeryToken()

<ul class="nav navbar-nav navbar-right">

<li>

@Html.ActionLink(Resources.Hello + User.Identity.GetUserName() + "!", "Manage", "Account", routeValues: null, htmlAttributes: new { title = "Manage" })

</li>

<li>

<a href="javascript:document.getElementById('logoutForm').submit()"> @Resources.LogOff</a>

</li>

</ul>

}

}

else

{

<ul class="nav navbar-nav navbar-right">

<li>@Html.ActionLink(Resources.Register, "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })

</li>

<li>@Html.ActionLink(Resources.LogIn, "Login", "Account", routeValues: null, htmlAttributes: new { id = "loginLink" })

</li>

</ul>

}

Кодът в този изглед се грижи за дясната част на навигацията, а именно да изобрази Register/Регистрация и Log In/Вход бутоните, ако потребителят не се е оторизирал пред системата или да изпише съобщение от вида "Здравейте, [потребителско име]"/"Hello, [username]" (В зависимост от избрания език), както и да изобрази бутонът Log Out/Изход.

### 2.3.2. Тяло

Тялото на страницата е изградено изключително просто:

<div class="container body-content shadow">

@RenderBody()

<div class="footer-wrapper">

<hr />

<footer class="centercontents">

</footer>

</div>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</div>

Прави впечатление, че съдържанието на страницата е оставено за всяка индивидуално страница, която го използва. Помощникът @RenderBody() указва къде ще бъде поместен частичния HTML на страницата, използваща шаблона.

Освен това, тук бива поместен и footer-a, който ще бъде разгледан в следващата подглава. След него, чрез помощниците @Scripts.Render(), биват заредени предварително подготвените ни bundle файлове със съответно jQuery файловете и boostrap файловете. Мястото на тези скриптове е отдолу на страницата, след като всички други HTML елементи са се заредили. Най-отдолу са поместени клиентските скриптове, тези които са дефинирани в секцията със скриптове на частичната HTML страница, която използва шаблона. Това са скриптове, които се изпълняват последни и най-често имат презентационна или бизнес функция.

### 2.3.3. Footer

Тази част от страницата помества в себе си информацията за сайта, контролите за избор на език, както и информация с връзки за оставащото време от абонамента и броя оставащи кредити. (фиг. 2.3.3.1)

<p class="pull-left">&copy; @DateTime.Now.Year - @Resources.ApplicationName </p>

@if (Request.IsAuthenticated)

{

SubscriptionCreditsModel model = (new SubscriptionController()). GetSubscriptionCredits(User.Identity.GetUserId());

if (model != null && this.ViewContext.RouteData.

GetRequiredString("controller") != "Subscription")

{

<span class="span5 center">

@if (!model.UnlimitedAccess)

{

if (model.ActiveUntil.HasValue &&

model.ActiveUntil.Value > DateTime.Today)

{

@Html.ActionLink(Resources.ActiveUntil + " " + (model.ActiveUntil.HasValue ? model.ActiveUntil.Value.ToString("dd/MM/yyyy") : "--/--/----"), "Index", "Subscription")

}

else

{

@Html.ActionLink(Resources.ActiveUntil + " " + (model.ActiveUntil.HasValue ? model.ActiveUntil.Value.ToString("dd/MM/yyyy") : "--/--/----"), "Index", "Subscription", new { }, new { style = "color: red;" })

}

<span>|</span>

if (model.Credits > 0)

{

@Html.ActionLink(Resources.Credits + " " + model.Credits, "Index", "Subscription")

}

else

{

@Html.ActionLink(Resources.Credits + " " + model.Credits, "Index", "Subscription", new { }, new { style = "color: red;" })

}

}

else

{

@Html.ActionLink(Resources.UnlimitedAccess, "Index", "Subscription")

}

</span>

}

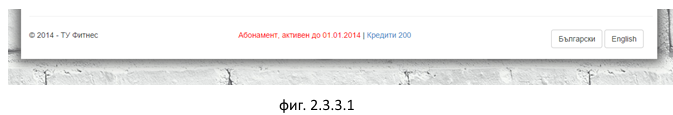
}

<span class="language-control pull-right">

@Html.ActionLink(Resources.Bulgarian, "SetCulture", "Home", new { culture = "bg" }, new { @class = "btn btn-default" })

@Html.ActionLink(Resources.English, "SetCulture", "Home", new { culture = "en" }, new { @class = "btn btn-default" })

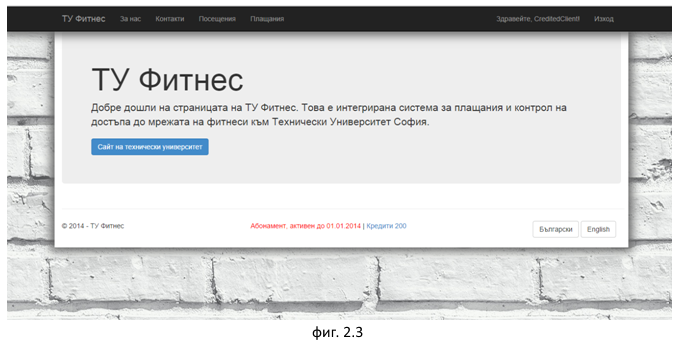
</span>



Първият параграф елемент <p>, окичен с класът pull-left, който го позиционира в лявата част на футъра има за цел да изпише на футъра "© 2014 - ТУ Фитнес" или "© 2014 - TU Fitness", според избраната култура.

След това бива поместена логиката, която изписва информацията за оставащия абонамент на клиента или наличните му кредити, като оцветява в червено съответния линк, ако времето е изтекло или кредитите са свършили.

<span> елементът с клас language-control се грижи да изобрази бутоните, които сменят езика.

В крайна сметка, след проектирането на \_Layout страницата, всички страници в сайта ще изглеждат по подобен начин (фиг. 2.3), като част от елементите на \_Layout страницата зависят от това, дали потребителят е аутентикиран или състоянието на неговия абонамент, докато индивидуалното съдържание на всяка страница се определя еднозначно от самата страница. 

## 2.4. Използвани стилове

CSS стиловете, използвани за конструирането на \_Layout страницата се намират в "Content/Site.css" файла във WebSite проекта. Файлът има следното съдържание:

html {

margin: 0;

padding: 0;

}

body {

background: url(/content/images/carbon.jpg) top left repeat;

padding-top: 50px;

padding-bottom: 20px;

margin: 0;

height: 100%;

}

/\* Set padding to keep content from hitting the edges \*/

.body-content {

padding-left: 15px;

padding-right: 15px;

padding-bottom: 100px;

background-color: white;

position: relative;

height: 100%;

}

/\* Set width on the form input elements since they're 100% wide by default \*/

input,

select,

textarea {

max-width: 280px;

}

/\* styles for validation helpers \*/

.field-validation-error {

color: #b94a48;

}

.field-validation-valid {

display: none;

}

input.input-validation-error {

border: 1px solid #b94a48;

}

input[type="checkbox"].input-validation-error {

border: 0 none;

}

.validation-summary-errors {

color: #b94a48;

}

.validation-summary-valid {

display: none;

}

.centercontents {

text-align: center !important;

}

.center {

float: none;

margin-left: auto;

margin-right: auto;

}

.underline {

text-decoration: underline;

}

.very-small-margin {

margin: 4px;

}

.footer-wrapper {

position: absolute;

width: 100%;

bottom: 0px;

}

hr {

width: 98%;

margin: 0px;

margin-bottom: 24px;

padding: 0px;

}

.language-control.pull-right {

margin-bottom: 20px;

margin-right: 40px;

}

.shadow {

-moz-box-shadow: 0 0 30px 8px #666;

-webkit-box-shadow: 0 0 30px 8px #666;

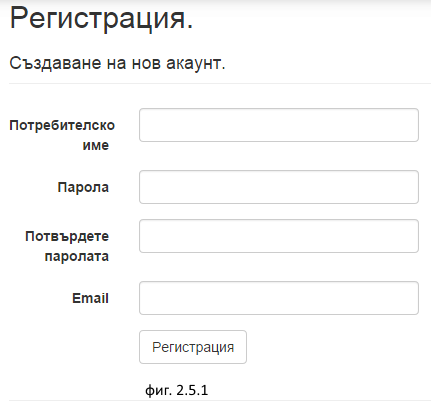
}

Останалите CSS класове, които биват използвани са предефинирани в "Content/bootstrap.css" файла във WebSite проекта. Използваната версия на bootstrap е 3.0.0, която също може да бъде намерена на следния адрес: <http://bootstrapdocs.com/v3.0.0/docs/getting-started/>

## 2.5. Проектиране на Register страницата за регистрации

http://83.148.87.159/TuFitness/Account/Register

Страницата за регистрация (фиг.2.5.2) е стандартна форма с 4 полета. (фиг. 2.5.1)

* Потребителско име - задължително поле,

описващо еднозначно името, с което потребителя се аутентикира пред системата

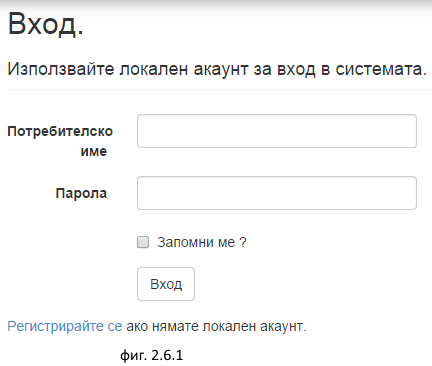
* Парола - задължително поле, паролата на потребителя
* Потвърдете паролата - задължително поле, задължително еднаква с "Парола"
* Email - задължително поле. Електронната поща на потребителя.

## 

## 

## 2.6. Проектиране на Login страницата за вход в системата

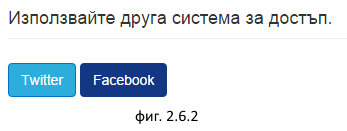
http://83.148.87.159/TuFitness/Account/Login



Login страницата (фиг. 2.6.3) е съставена от 2 части.

Стандартна "Вход" форма, с 2 полета (фиг. 2.6.1) -

* "Потребителско име" - уникалният идентификатор, чрез който потребителя се аутентикира пред системата.
* "Парола" - паролата за достъп на потребителя
* "Запомни ме" - checkbox, указващ дали сесията на потребителя да се пази за следващи влизания, без да има нужда от нов Вход в системата.

Освен стандартната форма за Вход, има и втора, осигуряваща достъп до системата, чрез аутентикация, посредством 3-ти системи, в случая Twitter и Facebook. (фиг. 2.6.2)

## 

## 2.7. Проектиране на Subscription страницата за извършване на плащания

http://83.148.87.159/TuFitness/Subscription

Абонаментната страница(фиг 2.7.1) предоставя стандартни "Buy Now" бутони, използващи PayPal API, за извършване на плащане. Callback функцията, която те извикват е PayPalPaymentNotification екшъна (Action) от PaymentSystem контролера.

# 

При създаването на тази страница се използва SubscriptionsCreditsViewModel модела, който бива построен от контролера с информация за статуса на абонамента на потребителя от dbo.SubscriptionPlan таблицата, както и със списък от продукти от определн туп от dbo.Product таблицата.

public class SubscriptionsCreditsViewModel

{

public SubscriptionCreditsModel SubscriptionCreditsModel { get; set; }

public IEnumerable<ProductModel> Products { get; set; }

}

Вижда се, че SubscriptiоnsCreditsViewModel-a се състои от инстанция на SubscriptionCreditsModelа, както и от колекция от ProductModel-и.

public class SubscriptionCreditsModel

{

[Display(Name = "ActiveUntil", ResourceType = typeof(Resources.Resources))]

public DateTime? ActiveUntil { get; set; }

[Display(Name = "Credits", ResourceType = typeof(Resources.Resources))]

public int Credits { get; set; }

[Display(Name = "UnlimitedAccess", ResourceType = typeof(Resources.Resources))]

public bool UnlimitedAccess { get; set; }

}

public class ProductModel

{

public int Id { get; set; }

public int ProductTypeId { get; set; }

public string Name { get; set; }

public decimal Value { get; set; }

public string Currency { get; set; }

public string Display { get; set; }

}

.cshtml кодът, който генерира страницата, прави няколко проверки по време на своето изпълнение. Първо проверява, дали потребителят няма неограничен достъп до помещенията и ако това е така, не му предоставя възможност да извършва плащания. Също така <input> елементът от тип "hidden", с id - "hideFooter" служи, за да окаже на \_Layout изгледа да не изобразява във футъра на страницата навигационните линкове с информация за абонамента, тъй като тази информация я има на текущата страница и навигация до нея не е необходима.

<div class="row">

@if (!Model.SubscriptionCreditsModel.UnlimitedAccess)

{

string callback = "http://" + HttpContext.Current.Request.Url.Host + ((HttpContext.Current.Request.Url.Host == "localhost") ? (":" + HttpContext.Current.Request.Url.Port) : "/TUFitness") + "/payment/PayPalPaymentNotification";

PaymentController paymentController = new PaymentController();

<div class="col-md-6">

<h3 class="underline">

@Html.DisplayNameFor(model => model.SubscriptionCreditsModel.Credits)

@Html.DisplayFor(model => model.SubscriptionCreditsModel.Credits)

</h3>

@foreach (var product in Model.Products.Where(product => product.ProductTypeId == 1).ToList())

{

<div class="row very-small-margin">

<h4 class="pull-left">@product.Display @Resources.Credits </h4>

<span class="pull-right">

@product.Value.ToString("F") @Resources.Currency

<script async="async" src= "https://www.paypalobjects.com/

js/external/paypal-button.min.js?

merchant=admin@tu.com"

data-button="buynow"

data-name="@product.Name"

data-quantity="1"

data-amount= @paymentController.ConvertBGNToGBP(

product.Value.ToString().Replace(",","."))

data-currency="GBP"

data-callback=@callback

data-custom=@(product.Id + "\_" + HttpContext.Current.User.Identity.GetUserId())

data-env="sandbox">

</script>

</span>

</div>

}

</div>

<div class="col-md-6">

<h3 class="underline">

@Html.DisplayNameFor(model => model.SubscriptionCreditsModel.ActiveUntil)

@(Model.SubscriptionCreditsModel.ActiveUntil.HasValue ? Model.SubscriptionCreditsModel.ActiveUntil.Value.

ToString("dd/MM/yyyy") : "--/--/----")

</h3>

@foreach (var product in Model.Products.Where(product => product.ProductTypeId == 2).ToList())

{

<div class="row very-small-margin">

<h4 class="pull-left">

@product.Display @Resources.Month

</h4>

<span class="pull-right">

@product.Value.ToString("F") @Resources.Currency

<script async="async" src="https://www.paypalobjects.com/

js/external/paypal-button.min.js?

merchant=admin@tu.com"

data-button="buynow"

data-name="@product.Name"

data-quantity="1"

data-amount=@paymentController.

ConvertBGNToGBP(product.Value.ToString().

Replace(",","."))

data-currency="GBP"

data-callback=@callback

data-custom=@(product.Id + "\_" + HttpContext.Current.User.Identity.GetUserId())

data-env="sandbox">

</script>

</span>

</div>

}

</div>

}

else

{

<div class="col-md-12">

@Html.DisplayNameFor(model => model.SubscriptionCreditsModel.UnlimitedAccess)

@Html.DisplayFor(model => model.SubscriptionCreditsModel.UnlimitedAccess)

</div>

}

@Html.Hidden("HideFooter", new { }, new { id = "hideFooter" })

</div>

Callback функцията, която PayPalAPI извиква след като плащането завърши се изчислява програмно, в зависимост от това дали сайтът е хостнат или е стартиран на localhost на машината. За да може да се използва Yahoo! Finance Api, се инициализира PaymentController-а и се извиква ConvertBGNToGBP() метода, за да се получи актуалната цена на избрания продукт в Британски Паундове (GBP), тъй като Български Левове (BGN) не е валута, която се поддържа от системата на PayPal към този момент.

С цел лесното използване на PayPalAPI, бива включена и ползвана библиотеката, която PayPal предоставя (paypal-button.min.js) за лесно създаване на бутони за плащане, директно в HTML страницата, без да има необходимост от допълнителни настройки на сървъра за самото плащане.

За създаване на бутон, се извиква <script> елемент с определени атрибути, който в зависимост от настройките продуцира HTML в следния вид:

<form method="post"

action="https://www.sandbox.paypal.com/cgi-bin/webscr"

class="paypal-button"

target="\_top">

<div class="hide" id="errorBox"></div>

<input type="hidden" name="button" value="buynow">

<input type="hidden" name="item\_name" value="50 Credits">

<input type="hidden" name="quantity" value="1">

<input type="hidden" name="amount" value="3.99">

<button type="submit" class="paypal-button large">Buy Now</button>

<input type="hidden" name="currency\_code" value="GBP">

<input type="hidden" name="notify\_url" value="http://localhost:3761/payment/PayPalPaymentNotification">

<input type="hidden" name="custom" value="1\_cab764af-8d7f-4186-bed6-727292697f75">

<input type="hidden" name="env" value="www.sandbox">

<input type="hidden" name="cmd" value="\_xclick">

<input type="hidden" name="business" value="admin@tu.com">

<input type="hidden" name="bn" value="JavaScriptButton\_buynow">

</form>

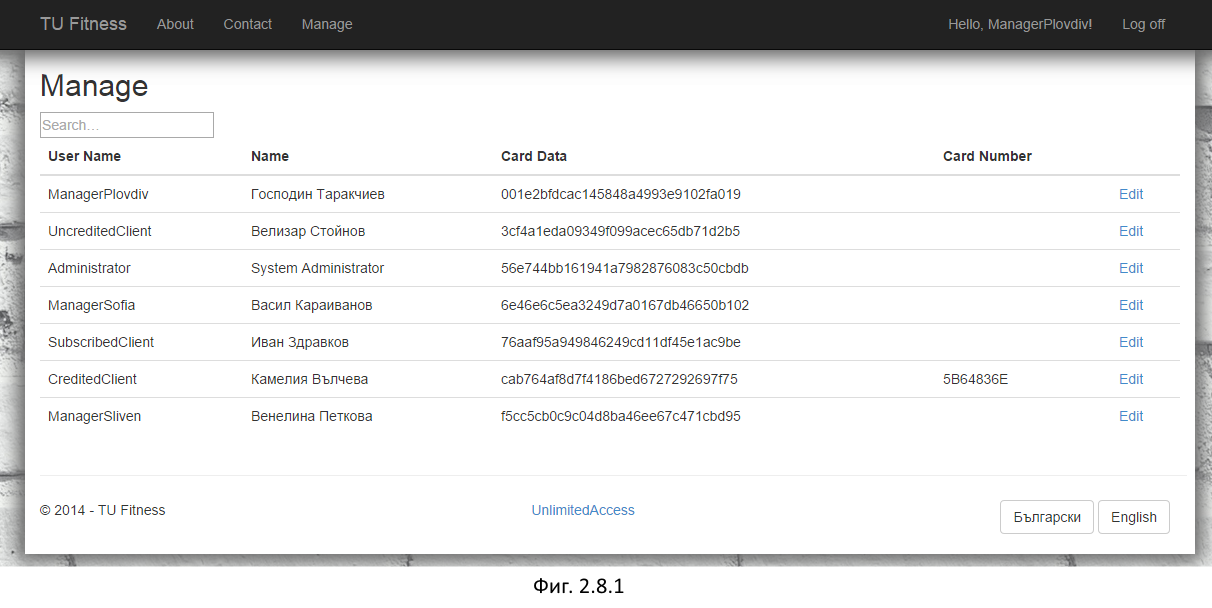
По-важните от тези <input> елементи са:

* **item\_name** - съдържа името на продукта
* **quantity -**  количеството от закупения продукт
* **amount -** цената на единица от продукта
* **currency\_code -** кодът на валутата
* **notify\_url** - callback URL, към който PayPalAPI се обръща след успешно плащане. Именно методът на този URL извършва действията в системата след плащането.
* **custom -** поле за специфична информация. В случая съдържа идентификатора на продукта и GUID идентификатора на потребителя
* **env -** Enviroment, средата в която се извършва плащането. Sandbox е тестовата среда.
* **business -** Потребителят (Бизнесът), до който се извършва плащането

## 2.8. Проектиране на Management страницата за управление на потребители

http://83.148.87.159/TuFitness/ Manage

Management страницата (фиг. 2.8.1) е страница, достъпна до потребители с ролите "Administrator", "FacilityManager" и "Receptionist". Основната и роля е да бъде използвана за начисляване на RFID карти на определен потребител.



При голям брой потребители в системата е наложително да се имплементира някакъв вид търсене. Класическият метод изисква всеки път при смяна на параметъра на търсене да се изпраща HTTP заявка и да се зареждат необходимите потребители наново, чрез Ajax или дори чрез напълно зареждане на страницата. Това е свързано освен с време за извършване на заявките и със сървърно време за извличане на данните от базата и подготвянето им в модели.

Поради тази причина се прибягва до използването на Knockout библиотеката. Самата библиотека е изградена на база на Model-View View-Model архитектурата. Това означава, че декларативно на HTML елементите се задава, с кои обекти от модела трябва да бъдат свързани, като при промяна на HTML елементите (например смяна стойността на <input> елемент или <select> елемент), то респективно се променя модела. Обратното също е вярно. Използвайко това свойство на Knockout моделите, се генерира таблица, и <input> елемент, играещ роля на Search. По този начин, чрез Аjax се зареждат всички потребители наведнъж, а филтрирането след това се осъществява програмно от Java Script Engine-a на браузъра, който се използва.

<h2>@Resources.ManagerPanel</h2>

<input placeholder="Search…" id="searchUsers">

<table class="table">

<thead>

<tr>

<th>@Resources.UserName</th>

<th>@Resources.Name</th>

<th>@Resources.CardData</th>

<th>@Resources.CardNumber</th>

<th></th>

</tr>

</thead>

<tbody data-bind="foreach: searchedUsers">

<tr>

<td data-bind="text: UserName"></td>

<td data-bind="text: Name"></td>

<td data-bind="text: CardData"></td>

<td data-bind="text: CardNumber"></td>

<td><a class="edit" href="#" data-bind="attr: { id: CardData }">Edit</a></td>

</tr>

</tbody>

</table>

Благодарение на тази архитектура и на атрибутите, които Knockout.js предоставя за използване от HTML елементите, съществува възможност да се пише изключително чист HTML.

data-bind="foreach: searchedUsers"

Този атрибут, използван в <tbody> елемента указва, че съдържанието на този елемент трябва да бъде повторено за всеки един от обектите в searchedUsers модела. Всеки от <td> елементите в неговото тяло, указва кое пропърти да бъде изобразено в тялото на съответния елемент.

При вписване на символи в <input> елемента, се задейства $( ).keyup( ) jQuery event (събитие), който филтрира данните в модела, според съвпадението на написаните символи след повдигането на клавиша с тези от колоните UserName и Name на таблицата.

Edit линкът на всеки един от елементите, окичен с атрибут id, равен на GUID идентификационния номер на съответния потребител дава възможност, чрез javascript prompt( ) прозорец да бъде въведена нова стойност за Card Number, която след затварянето на прозореца бива изпратена, чрез HTTP Post заявка до съответния екшън на сървъра, който създава нова карта за съответния потребител с указания номер на карта и деактивира старите карти на потребителя, ако той има такива. Моделът и описаната до тук логика се поместват в Scripts секцията на частичния изглед и в $(document).ready( ) jQuery функцията, които зареждат JavaScript кода, едва след като преди това са се заредили всички елементи на потребителския интерфейс.

@section scripts {

<script type="text/javascript">

$(document).ready(function () {

$.getJSON("/manage/getUsers").success(function (data) {

var users = [];

for (var i in data) {

var user = data[i];

users.push(user);

}

function UsersModel(search) {

var self = this;

self.users = ko.observableArray(users);

self.searchedUsers = ko.observableArray(users);

self.searchUsers = function (word) {

self.reset();

for (var i = self.searchedUsers().length - 1; i >= 0; i--) {

var user = self.searchedUsers()[i];

if ((user.Name.indexOf(word) == -1) && (user.UserName.indexOf(word) == -1)) {

self.searchedUsers.splice(i, 1);

}

}

}

self.reset = function () {

for (var i in data) {

var user = data[i];

var found = false;

for (var i = 0; i < self.searchedUsers().length; i++) {

var searchedUser = self.searchedUsers()[i];

if (searchedUser.UserName == user.UserName) {

found = true;

break;

}

}

if (!found) {

self.searchedUsers.push(user);

}

}

}

}

model = new UsersModel();

ko.applyBindings(model);

$('#searchUsers').keyup(function () {

var word = $('#searchUsers').val();

model.searchUsers(word);

});

$('.edit').click(function () {

var id = $(this).attr('id');

var cardId = prompt('@Resources.EnterNewCard', '@Resources.CardNumber');

if (cardId != null) {

var data = {

UserId: id,

CardId: cardId

};

$.post('manage/changeCard', data, function (response) {

if (response == true) {

location.reload(true);

}

else {

alert('@Resources.UnsuccessfulDataSave');

}

}, 'json');

}

});

})

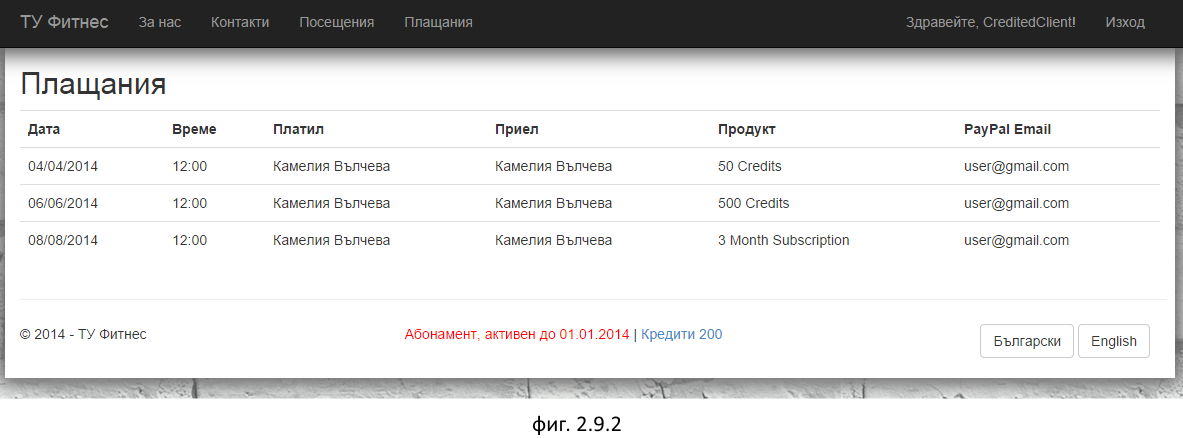
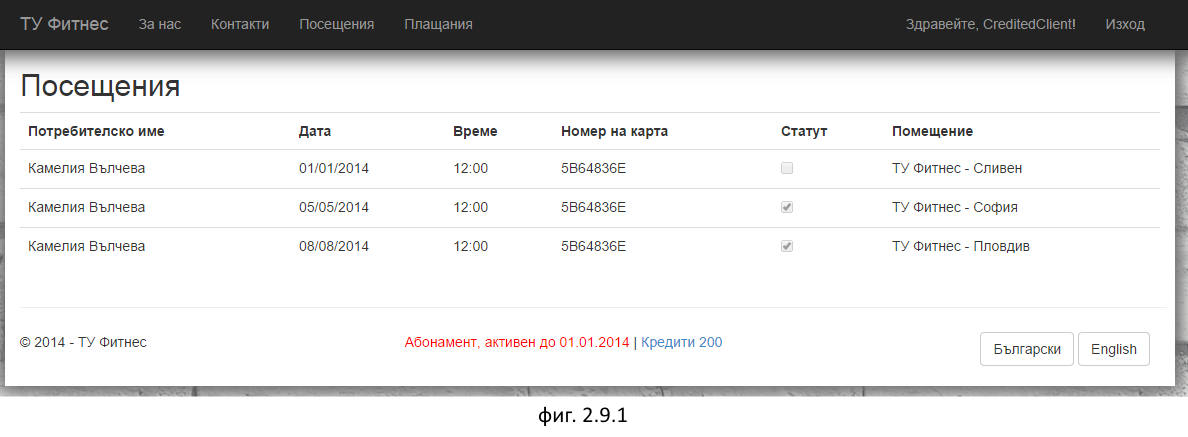
})

</script>

}

## 2.9. Проектиране на Visits и Payments страниците

Страниците за посещения и плащания са близки по структура страници, които са единствено информационни. Те се състоят от таблица с набор от колони, които съдържат пълна хронология на извършените от дадения потребител посещения (фиг. 2.9.1) или съответно плащания (фиг. 2.9.2) в системата.



# Глава 3. Проектиране на релационната база от данни

## 3.1. Релационен модел

В настоящата дипломна работа базата данни ще бъде организирана като релационна - информацията се съхранява в таблици, свързани помежду си чрез полета, съдържащи информация за свързване. Ще има първични и вторични индекси – съответно първичен (primary key) и вторичен ключ (foreign key), които служат за сортиране на базата данни, ускоряват търсенето в наборите от данни и служат за запазване цялостта на данните. Първичният ключ представлява уникален идентификатор на запис в таблицата, който еднозначно определя записа. Вторичният се явява указател към първичен от друга таблица.

Съществуват следните видове релации:

* **едно – към – едно (1:1) -** запис от дадена таблица е свързан с точно един запис от друга таблица.
* **едно – към – много** **(1:m)** - един запис от първата таблица може да бъде сочен от няколко записа от втората таблица. Това е най-често използваната релация при изграждане на база данни. При разработването на конкретната база данни ще бъде използван този тип релация за връзки между таблиците.
* **много – към – много** **(m:m)** - всеки един запис от първата таблица може да се свързва с всеки един елемент от другата таблица и обратно. Обикновено в случаите, в които е нужно да се осъществи такъв тип релация, се използва междинна таблица. По този начин релацията много-към-много се преобразува в две релации едно-към-много.

## 3.2. Identity таблици

Тъй като в по-нататъчния етап на разработка бъде използван използван ASP.NET Identity за управление на потребителите, ролите и твърденията им (Claims), се налага в етапа на разработка на базата от данни да бъдат предвидени и реализирани набор от таблици, от които ASP.NET Identity се нуждае за правилната си работа. Таблиците, които ASP.NET Identity изисква имат необходим минимален брой полета от точно определен тип и с точно определено наименование, за да бъде осъществена връзката (mapping) с Identity безпроблемно. Изискванията за таблиците са следните:

* AspnetRoles (Id: string, Name: string)
* AspnetUsers (Id: string, Username: string, PasswordHash: string, SecurityStamp: string, Discriminator: string)
* AspnetUserRole (RoleId: string, UserId: string)
* AspnetUserLogins (ProviderKey: string, UserId: string, LoginProvider: string)
* AspnetUserClaims (Id: string, ClaimType: string, ClaimValue: string, User\_Id: string)

Таблиците, техните полета, типове и описания, както и T-SQL скрипта за тяхното създаване са описани по-долу. Трябва да се обърне внимание, че има изискване за минимален брой на колоните и тяхния тип, като това не означава, че не може и на места не са добавени допълнителни колони, според нуждите.

Таблица **dbo.AspNetRoles -** Номенклатурна таблица, съдържаща имената на ролите, използвани в системата.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | nvarchar(128) | GUID идентификатор | PK |
| Name | nvarchar(50) | Име на ролята |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[AspNetRoles](

[Id] nvarchar(128) NOT NULL,

[Name] nvarchar(50) NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_AspNetRoles] PRIMARY KEY CLUSTERED ([Id] ASC)

)

Таблица **dbo.AspNetUsers -** Таблица, съдържаща информация за потребителското име, хешираната парола, проверката за сигурност, електронната поща, както и връзка 1:m с таблицата dbo.PersonalDetail

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | nvarchar(128) | GUID идентификатор | PK |
| UserName | nvarchar(max) | Потребителско име |  |
| PasswordHash | nvarchar(max) | Хеширана парола |  |
| SecurityStamp | nvarchar(max) | Проверка за сигурност |  |
| Discriminator | nvarchar(128) | Описание на типа потребител |  |
| Email | nvarchar(255) | Електронна поща |  |
| PersonalDetailsId | int | Персонални данни за потребителя | FK -> dbo.PersonalDetail |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[AspNetUsers](

[Id] nvarchar(128) NOT NULL,

[UserName] nvarchar(max) NULL,

[PasswordHash] nvarchar(max) NULL,

[SecurityStamp] nvarchar(max) NULL,

[Discriminator] nvarchar(128) NOT NULL,

[Email] nvarchar(255) NOT NULL,

[PersonalDetailsId] int NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_AspNetUsers] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_AspNetUsers\_PersonalDetails] FOREIGN KEY ([PersonalDetailsId]) REFERENCES [dbo].[PersonalDetails] (Id),

)

Таблица **dbo.AspNetUserRole** - междинна таблица, осигуряваща връзка **m:m** между таблиците **dbo.AspNetRoles** и **dbo.AspNetUsers.**

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| UserId | nvarchar(128) | GUID идентификатор | PK;  FK -> dbo.AspNetUsers |
| RoleId | nvarchar(128) | GUID идентификатор | PK;  FK -> dbo.AspNetRoles |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[AspNetUserRoles](

[UserId] nvarchar(128) NOT NULL,

[RoleId] nvarchar(128) NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_AspNetUserRoles] PRIMARY KEY CLUSTERED

([UserId] ASC, [RoleId] ASC),

CONSTRAINT [FK\_AspNetUserRoles\_AspNetRole] FOREIGN KEY

([RoleId]) REFERENCES [dbo].[AspNetRoles] ([Id]) ON DELETE CASCADE,

CONSTRAINT [FK\_AspNetUserRoles\_AspNetUser] FOREIGN KEY

([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE

)

Таблица **dbo.AspNetClaims** - таблица, съхраняваща твърденията (Claims), които могат да описват правата на потребителя да извършва определени действия.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| ClaimType | nvarchar(max) | Тип на твърдението |  |
| ClaimValue | nvarchar(max) | Стойност на твърдението |  |
| User\_Id | nvarchar(128) | Потребителски профил | FK -> dbo.AspNet.Users |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[AspNetUserClaims](

[Id] int IDENTITY (1, 1) NOT NULL,

[ClaimType] nvarchar(max) NULL,

[ClaimValue] nvarchar(max) NULL,

[User\_Id] nvarchar(128) NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_AspNetUserClaims] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_AspNetUserClaims\_AspNetUsers] FOREIGN KEY ([User\_Id]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE

)

Таблица **dbo.****AspNetUserLogins** - Таблица, съхраняваща логин информация от външни системи (Facebook, Twitter, Google Plus).

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| UserId | nvarchar(128) | GUID идентификатор | PK |
| LoginProvider | nvarchar(128) | GUID Id на издателя | PK |
| ProviderKey | nvarchar(128) | GUID Id на издадения ключ | PK |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[AspNetUserLogins] (

[UserId] nvarchar(128) NOT NULL,

[LoginProvider] nvarchar(128) NOT NULL,

[ProviderKey] nvarchar(128) NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_AspNetUserLogins] PRIMARY KEY CLUSTERED ([UserId] ASC, [LoginProvider] ASC, [ProviderKey] ASC),

CONSTRAINT [FK\_AspNetUserLogins\_AspNetUsers] FOREIGN KEY ([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id]) ON DELETE CASCADE

)

## 3.3. Таблици

Останалите таблици, които ще бъдат създадени, описват предметната област. Както може би е станало ясно от предходно описаните таблици на Identity, към всяка таблица в базата от данни, са добавени 4 служебни полета.

[CreatedBy] От кого е създаден записа

[CreatedAt] Кога е създаден записа

[LastModifiedBy] От кого последно е модифициран записа

[LastModifiedAt] Кога последно е модифициран записа

В една система за онлайн плащания е важно да може да бъдат проследени, както кой е създал даден запис, така и кой и кога последен е правил промяна по него. Тъй като в междинния слой за достъп до данните (DAL), се използва Entity Framework, в следващата глава е описано по какъв начин тези полета се попълват автоматично.

Таблица **dbo.Address -** тази таблица съдържа подробна информация за даден адрес. Тя може да бъде използвана, както от физически лица, служители и клиенти на системата (dbo.AspNetUsers), така и от помещенията (dbo.Facility), които са част от мрежата спортни зали на Технически Университет София. Колоните [Town] и [Country] не са нормализирани умишлено, за да не се натоварва базата с таблици и данни излишно, тъй като в момента в системата тази таблица бива използвана единствено от помещенията (dbo.Facility), като в нея има едва неколцина записа. На практика потребителите не попълват данни за своя адрес, тъй като те не са необходими за текущото работно задание. Ако обаче в бъдеще се породи нужда - базата може да бъде нормализирана, а на потребителите разрешено да попълват своите адреси.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| Country | nvarchar(50) | Държава |  |
| Town | nvarchar(50) | Град |  |
| PostCode | nvarchar(10) | Пощенски код |  |
| Address | nvarchar(255) | Адрес |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[Address](

[Id] int IDENTITY (1, 1) NOT NULL,

[Country] nvarchar(50) NULL,

[Town] nvarchar(50) NULL,

[PostCode] nvarchar(10) NULL,

[Address] nvarchar(255) NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_Address] PRIMARY KEY CLUSTERED ([Id] ASC),

)

Таблица **dbo.Contact -** тази таблица съдържа подробна информация за контактите на дадено физическо лице, служител, клиент (dbo.AspNetuUsers) или за определено помещение (dbo.Facility).

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| Phone | nvarchar(50) | Телефон |  |
| GSM | nvarchar(50) |  |  |
| FAX | nvarchar(50) | Факс |  |
| Email | nvarchar(255) | Електронна поща |  |
| Website | nvarchar(255) | Уеб сайт |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно модифицирано от |  |
| LastModifiedAt | datetime | Последно модифицирано на |  |

CREATE TABLE [dbo].[Contact](

[Id] int IDENTITY (1, 1) NOT NULL,

[Phone] nvarchar(50) NULL,

[GSM] nvarchar(50) NULL,

[Fax] nvarchar(50) NULL,

[Email] nvarchar(255) NULL,

[Website] nvarchar(255) NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_Contact] PRIMARY KEY CLUSTERED ([Id] ASC),

)

Таблица **dbo.Facility -** таблицата за помещенията съдържа информация за наименованието, адреса и контактите на дадено помещение, както и неговият отговорник (Manager)

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| Name | nvarchar(128) | Името на помещението |  |
| DeviceId | Int | Идентификационен номер на RFID четеца на помещението |  |
| ManagerId | int | Потребителския профил на ръководителя | FK -> dbo.AspNetUsers |
| AddressId | int | Адрес | FK -> dbo.Address |
| ContactId | int | Контакти | FK -> dbo.Contact |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[Facility](

[Id] int IDENTITY (1, 1) NOT NULL,

[Name] nvarchar(50) NOT NULL,

[DeviceId] int NULL,

[ManagerId] nvarchar(128) NOT NULL,

[AddressId] int NOT NULL,

[ContactId] int NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_Facility] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Facility\_AspNetUsers] FOREIGN KEY ([ManagerId]) REFERENCES [dbo].[AspNetUsers] (Id),

CONSTRAINT [FK\_Facility\_Address] FOREIGN KEY ([AddressId])

REFERENCES [dbo].[Address] (Id),

CONSTRAINT [FK\_Facility\_Contact] FOREIGN KEY ([ContactId])

REFERENCES [dbo].[Contact] (Id),

)

Таблица **dbo.Card -** таблица съхраняваща данните на RFID карта. Необходимата ни информация е уникалния идентификатор на самата карта, както и идентификатора на потребителя, за който тя се отнася, както и дали картата е активна.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| CardId | nvarchar(128) | Идентификатор на RFID картата |  |
| UserProfileId | nvarchar(128) | Идентификатора на потребителя, на който принадлежи картата | FK -> dbo.AspNetUsers |
| IsActive | bit | Активна ли е картата |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[Card]

(

[Id] int IDENTITY (1, 1) NOT NULL,

[CardId] nvarchar(128) NOT NULL,

[UserProfileId] nvarchar(128) NOT NULL,

[IsActive] bit NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_Card] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Card\_UserProfile] FOREIGN KEY ([UserProfileId]) REFERENCES [dbo].[AspNetUsers] (Id)

)

Таблица **dbo.EntranceHistory -** таблица, която пази информация за успешните или неуспешни посещения на потребителя, мястото и датата, където се е състоял входа.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| CardId | int | Идентификатор на картата, използвана за вход | FK -> dbo.Card |
| FacilityId | nvarchar(128) | Идентификатора на помещението, в което е осъществен входа | FK -> dbo.Facility |
| EntryStatus | bit | Успешен ли е бил входът |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[EntranceHistory]

(

[Id] int IDENTITY (1, 1) NOT NULL,

[CardId] int NOT NULL,

[FacilityId] int NOT NULL,

[EntryStatus] bit NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_EntranceHistory] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_EntranceHistory\_Card] FOREIGN KEY ([CardId]) REFERENCES [dbo].[Card] (Id),

CONSTRAINT [FK\_EntranceHistory\_Facility] FOREIGN KEY ([FacilityId]) REFERENCES [dbo].[Facility] (Id),

)

Таблица **dbo.PaymentHistory -** таблица, съхраняваща данни за извършените плащания, поръчителя им, бенифактора, както и PayPal акаунта, ако плащането е извършено през интернет.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| RecepientId | nvarchar(128) | Идентификатор на бенефактора | FK -> dbo.AspNetUsers |
| IssuerId | nvarchar(128) | Идентификатора на потребителя, извършил плащането | FK -> dbo.AspNetUsers |
| ProductId | bit | Идентификатор на продукта, който е закупен | FK -> dbo.Product |
| PayPalEmail | nvarchar(max) | PayPal елетронната поща, от която е извършено плащането |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[PaymentHistory]

(

[Id] int IDENTITY (1, 1) NOT NULL,

[RecepientId] nvarchar(128) NOT NULL,

[IssuerId] nvarchar(128) NOT NULL,

[ProductId] int NOT NULL,

[PayPalEmail] nvarchar(max) NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_PaymentHystory] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_PaymentHystory\_Recepient] FOREIGN KEY ([RecepientId]) REFERENCES [dbo].[AspNetUsers] (Id),

CONSTRAINT [FK\_PaymentHystory\_Issuer] FOREIGN KEY ([IssuerId]) REFERENCES [dbo].[AspNetUsers] (Id),

CONSTRAINT [FK\_PaymentHystory\_Product] FOREIGN KEY ([ProductId]) REFERENCES [dbo].[Product] (Id),

)

Таблица **dbo.PersonalDetails -** таблица, съдържаща допълнителна информация за потребителите, както и референция към абонаментния им план.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| FirstName | nvarchar(50) | Първо име |  |
| MiddleName | nvarchar(50) | Второ име |  |
| LastName | nvarchar(50) | Фамилия |  |
| AddressId | int | Идентификатор за адреса на потребителя (Не се изисква към момента) | FK -> dbo.Address |
| ContactId | int | Идентификатор за контактните данни на потребителя | FK -> dbo.Contact |
| SubscriptionPlanId | int | Идентификатор за абонаментния план на потребителя | FK -> dbo.SubscriptionPlan |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[PersonalDetails](

[Id] int IDENTITY (1, 1) NOT NULL,

[FirstName] nvarchar(50) NULL,

[MiddleName] nvarchar(50) NULL,

[LastName] nvarchar(50) NULL,

[AddressId] int NULL,

[ContactId] int NULL,

[SubscriptionPlanId] int NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_PersonalDetails] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_PersonalDetails\_Address] FOREIGN KEY ([AddressId]) REFERENCES [dbo].[Address] (Id),

CONSTRAINT [FK\_PersonalDetails\_Contact] FOREIGN KEY ([ContactId]) REFERENCES [dbo].[Contact] (Id),

CONSTRAINT [FK\_PersonalDetails\_SubscriptionPlan] FOREIGN KEY ([SubscriptionPlanId]) REFERENCES [dbo].[SubscriptionPlan] (Id),

)

Таблица **dbo.ProductType -** номенклатурна таблица, съдържаща видовете продукти, които съществуват в системата.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| Name | nvarchar(255) | Името на продукта |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[ProductType] (

[Id] int IDENTITY (1, 1) NOT NULL,

[Name] nvarchar(255) NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_ProductType] PRIMARY KEY CLUSTERED ([Id] ASC),

)

Таблица **dbo.Product -** таблица, съдържаща информация за имената, типовете и цените на продуктите в системата. Специално е полето IsActive, което отбелязва, дали даден продукт е активен. Ако например трябва да се промени цената на продукт, той бива маркиран като неактивен и същия продукт се създава с нова цена и поле IsActive = true. Това се прави с цел данните за старите плащания да бъдат консистентни, тъй като за всяко минало плащане пазим и продукта, който е закупен, включително цената.

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| ProductTypeId | int | Идентификатор за типа продукт | FK -> dbo.ProductType |
| IsActive | bit | Активен ли е продукта |  |
| Name | nvarchar(255) | Името на продукта |  |
| Display | nvarchar(max) | По какъв начин се изобразява продукта на екрана |  |
| Value | smallmoney | Стойността на продукта |  |
| Currency | nvarchar(3) | Валута |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[Product]

(

[Id] int IDENTITY (1, 1) NOT NULL,

[ProductTypeId] int NOT NULL,

[IsActive] bit NOT NULL,

[Name] nvarchar(255) NOT NULL,

[Display] nvarchar(max) NULL,

[Value] smallmoney NOT NULL,

[Currency] nvarchar(3) NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_Product] PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Product\_ProductType] FOREIGN KEY ([ProductTypeId]) REFERENCES [dbo].[ProductType] (Id),

)

Таблица **dbo.SubscriptionPlan -** таблица, описваща количеството кредити, които клиентът има, дали и до кога има платен достъп до помещенията и дали има неограничен достъп до помещенията (Само за служители и персонал)

|  |  |  |  |
| --- | --- | --- | --- |
| Име на колона | Тип | Описание | Връзки |
| Id | int | Целоцифрен идентификатор | PK |
| UnlimitedAccess | bit | Неограничен достъп |  |
| ActiveUnitl | Datetime | До кога е активен абонамента |  |
| Credits | Int | Брой кредити за вход, които потребителя има |  |
| CreatedBy | nvarchar(128) | Създадено от |  |
| CreatedAt | datetime | Създадено на |  |
| LastModifiedBy | nvarchar(128) | Последно  модифицирано от |  |
| LastModifiedAt | datetime | Последно  модифицирано на |  |

CREATE TABLE [dbo].[SubscriptionPlan] (

[Id] int IDENTITY (1, 1) NOT NULL,

[UnlimitedAccess] bit NOT NULL,

[ActiveUntil] datetime NULL,

[Credits] int NOT NULL,

[CreatedBy] nvarchar(128) NULL,

[CreatedAt] datetime NULL,

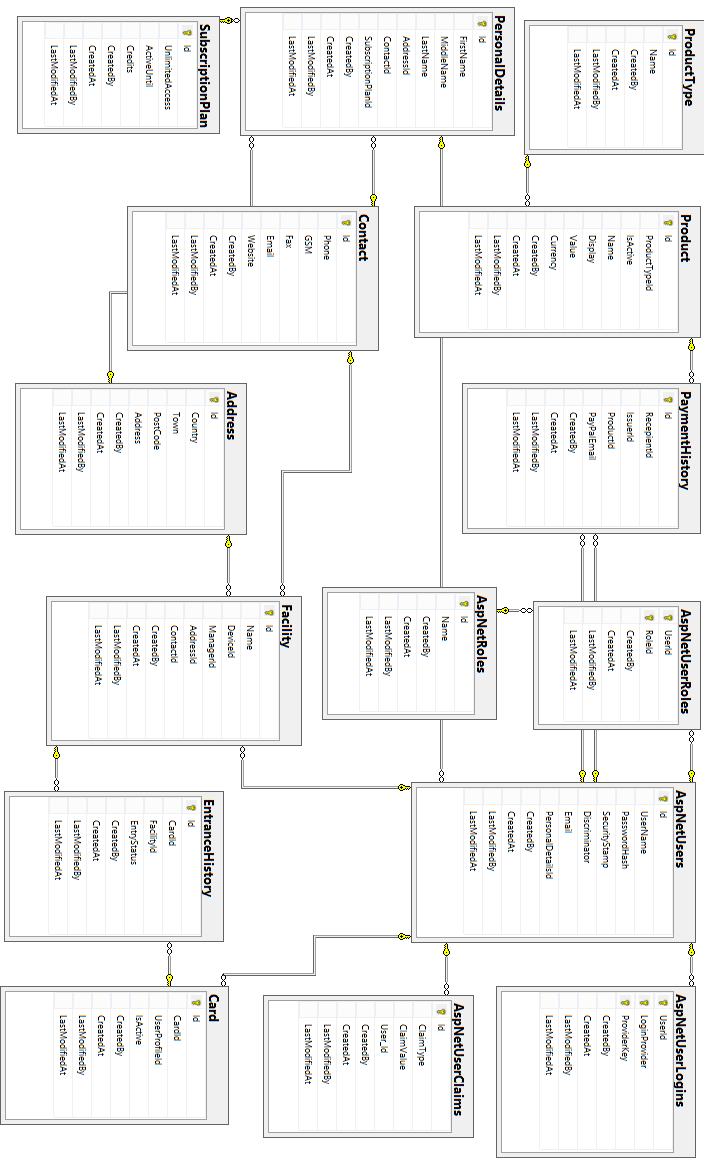
[LastModifiedBy] nvarchar(128) NULL,

[LastModifiedAt] datetime NULL,

CONSTRAINT [PK\_SubscriptionPlan] PRIMARY KEY CLUSTERED ([Id] ASC),

)

## 3.4. Entity-relationship диаграма



# Глава 4. Проектиране и реализация на Web услуги

## 4.1. BaseController и EntityHelpers класовете

Класовете, които предоставят функционалността за обработване на MVC и API завките са Controller и ApiController. За да може да бъде използван вече генерирания Entity Framework модел, тези 2 класа ще бъдат разширени и наследниците им ще бъдат използвани за всички контролери.

**BaseController**

public class BaseController : Controller

{

private EntityHelpers entityHelpers = new EntityHelpers();

public EntityHelpers EntityHelper

{

get { return entityHelpers; }

}

public ExtendedDBContext DB

{

get { return entityHelpers.DB; }

}

protected override IAsyncResult BeginExecuteCore(AsyncCallback callback, object state)

{

// Този метод вече беше разгледан в глава 2

}

}

**BaseApiController**

public class BaseApiController : ApiController

{

private EntityHelpers entityHelpers = new EntityHelpers();

public EntityHelpers EntityHelper

{

get { return entityHelpers; }

}

public ExtendedDBContext DB

{

get { return entityHelpers.DB; }

}

}

Вижда се, че двата класа имат сходна структира. Инстанцират пропъртита от тип EntityHelpers и от тип ExtendedDbContext. Целта на EnityHelpers класа е да служи като хранилище на често използвани заявки . Например, ако често в кода се налага да бъдат изписани 3-те имена на потребителя, със съкратено средно име, по принцип би се наложило да се използва следния код на всяко място, където това е необходимо:

AspNetUser user = DB.AspNetUsers.Find(userId).FirstOrDefault();

string fullName = String.Empty;

if (user != null)

{

fullName = user.FirstName + " " + user.MiddleName.Substring(0, 1) + " " + user.LastName;

}

С цел изнасяне на тази често използвана логика на едно място, в този клас бива създаден метод, който извършва тези действия, и при нужда бива извикан в кодът.

string fullName = EntityHelper.GetFullUserName(string userId);

Най-съществения клас в цялата имплементация е ExtendedDBContext, който наследява InternalDBEntities - класа, който съдържа модела и който се инстанцира, за да работи с обектите от базата.

**ExtendedDbContext**

public class ExtendedDBContext : InternalDBEntities

{

public override int SaveChanges()

{

ChangeTracker.DetectChanges();

string currentUserId = this.GetCurrentUserId();

ObjectContext ctx = ((IObjectContextAdapter)this).ObjectContext;

List<ObjectStateEntry> objectStateEntryList = ctx.ObjectStateManager.GetObjectStateEntries

(EntityState.Added | EntityState.Modified | EntityState.Deleted).ToList();

foreach (ObjectStateEntry entry in objectStateEntryList)

{

if (entry.State == EntityState.Added)

{

if (entry.Entity.GetType().GetProperties().Where(t => t.Name == "CreatedBy").Any())

{

Type type = entry.Entity.GetType();

type.GetProperties().Where(t => t.Name == "CreatedBy").First().SetValue(entry.Entity, currentUserId);

}

if (entry.Entity.GetType().GetProperties().Where(t => t.Name == "CreatedAt").Any())

{

Type type = entry.Entity.GetType();

type.GetProperties().Where(t => t.Name == "CreatedAt").First().SetValue(entry.Entity, DateTime.Now);

}

}

}

var trackables = ChangeTracker.Entries();

foreach (var entry in trackables.Where(t => t.State == EntityState.Modified))

{

if (entry.Entity.GetType().GetProperties().Where(t => t.Name == "LastModifiedBy").Any())

{

Type type = entry.Entity.GetType();

type.GetProperties().Where(t => t.Name == "LastModifiedBy").First().SetValue(entry.Entity, currentUserId);

}

if (entry.Entity.GetType().GetProperties().Where(t => t.Name == "LastModifiedAt").Any())

{

Type type = entry.Entity.GetType();

type.GetProperties().Where(t => t.Name == "LastModifiedAt").First().SetValue(entry.Entity, DateTime.Now);

}

}

try

{

return base.SaveChanges();

}

catch (DbEntityValidationException ex)

{

var errorMessages = ex.EntityValidationErrors.SelectMany(x => x.ValidationErrors).Select(x => x.ErrorMessage);

var fullErrorMessage = string.Join("; ", errorMessages);

var exceptionMessage = string.Concat(ex.Message, " The validation errors are: ", fullErrorMessage);

throw new DbEntityValidationException(exceptionMessage, ex.EntityValidationErrors);

}

}

private string GetCurrentUserId()

{

string currentUserId = null;

if (HttpContext.Current != null && HttpContext.Current.User != null && HttpContext.Current.User.Identity != null)

{

currentUserId = HttpContext.Current.User.Identity.GetUserId();

}

if (currentUserId == null)

{

currentUserId = GlobalConstants.SystemAdministratorId.ToString();

}

return currentUserId;

}

}

ExtendedDbContext класа се състои от 2 метода. Единият е private метода GetCurrentUserId, чиято роля е да върне GUID идентификатора на текущия потребител. Другия метод презаписва (overrides) метода SaveChanges, за който вече се спомена, че записва данните в базата, като при override-а на метода, изпълнява логика по попълване на [CreatedBy], [CreatedAt], [LastModifiedBy] и [LastModifiedAt] пропъртитата на всеки от моделите, които са били променени от Entity Framework след последния запис. След като метода попълни полетата, извиква метода SaveChanges на базовия клас и изпълнението продължава необезпокоявано. В известен смисъл този метод "подпъхва" програмна логика, изпълнявана преди всеки SaveChanges(). Целта на това е винаги, когато в базата се създава или модифицира запис, служебните полета с информацията за това кой и кога е създал/модифицирал полето да се попълват автоматично от рамката (famework), без да се натоварва излишно разработчика с тази задача.

Логичен тук би бил въпросът - как моделът разбира предварително за коя таблица се отнасят промените и какво се случва, ако има таблица, която няма нужните полета? Тъй като при построяването (Build) на проекта няма как да се знае, коя таблица ще бъде запазена в кой момента, се прибягва до помощта на така наречения Reflection.

Reflection е способността на дадена програма да изследва структурата и поведението на даден код по време на изпълнението си (runtime), като според особеностите на кода, може да взема дадени решения за това как да го изпълни. В конкретния случай, чрез Reflection се проверява, дали обектът, който трябва да се запази притежава свойствата, които искаме да зададем. Ако не ги притежава, кодът продължава своето изпълнение. Ако обаче ги притежава, стойностите им се попълват.

## 4.2. Payment Controller

Payment контролерът се грижи за обработването на плащането, след неговото извършване.

PayPalPaymentNotification методът е този, който е посочен като Callback метод при успешно плащане от Payment страницата. Като параметър получава стандартизирания от PayPal - PayPalCheckoutInfo. Този модел подлежи на проверка за истиност, преди да се пристъпи към обработване на информацията. С помощта на Verify метода на PayPalListenerModel се осъществява тази проверка и при успешното потвърждаване на плащането се преминава към същината на метода, а именно извличане на Id-то на продукта, който е купен и Guid идентификатора на потребителя. С тези данни се попълва запис в таблицата dbo.PaymentHistory и се обновява абонаментния план на потребителя, според извършеното плащане.

ConvertBGNToGBP методът осъществява връзката с Yahoo! Finance Api, за получаването на актуална цена на стоките в Британски Лири (GBP). Този метод се ползва при генерирането на PayPal бутоните.

**PaymentController**

[AllowAnonymous]

public class PaymentController : BaseController

{

public EmptyResult PayPalPaymentNotification(PayPalCheckoutInfo payPalCheckoutInfo)

{

PayPalListenerModel model = new PayPalListenerModel();

model.\_PayPalCheckoutInfo = payPalCheckoutInfo;

byte[] parameters = Request.BinaryRead(Request.ContentLength);

if (parameters != null)

{

model.GetStatus(parameters);

}

int productId = int.Parse(payPalCheckoutInfo.custom.Split('\_')[0]);

string userId = payPalCheckoutInfo.custom.Split('\_')[1];

Product product = DB.Products.Where(p => p.Id == productId).SingleOrDefault();

if (product != null)

{

DB.PaymentHistories.Add(new PaymentHistory

{

Product = product,

RecepientId = userId,

IssuerId = userId,

PayPalEmail = payPalCheckoutInfo.payer\_email

});

AspNetUser user = DB.AspNetUsers.Find(userId);

if (user != null)

{

SubscriptionPlan plan = user.PersonalDetail.SubscriptionPlan;

if (product.ProductTypeId == 1) // Credits

{

plan.Credits += int.Parse(product.Display);

}

else if (product.ProductTypeId == 2) // Subscription

{

if (plan.ActiveUntil.HasValue && plan.ActiveUntil.Value > DateTime.Today)

{

plan.ActiveUntil = plan.ActiveUntil.

Value.AddMonths(int.Parse(product.Display));

}

else

{

plan.ActiveUntil = DateTime.

Today.AddMonths(int.Parse(product.Display));

}

}

}

DB.SaveChanges();

}

return new EmptyResult();

}

[NonAction]

public string ConvertBGNToGBP(string amountString)

{

decimal amount = decimal.Parse(amountString, CultureInfo.InvariantCulture);

string fromCurrency = "BGN";

string toCurrency = "GBP";

WebClient web = new WebClient();

Uri uri = new Uri(string.Format(

"http://finance.yahoo.com/d/quotes.csv?s={0}{1}=X&f=l1",

fromCurrency, toCurrency));

string response = web.DownloadString(uri);

decimal rate = Convert.ToDecimal(response.Replace("/r/n", ""), CultureInfo.InvariantCulture);

return (amount \* rate).ToString("F").Replace(",", ".");

}

PayPalCheckoutInfo класът е взет от примерна имплементация, побликувана свободно на developer.paypal.com. Това е моделът, който приема данните от PayPalApi с разултатите на плащането. Кодът е оставен в оригинал, заедно с всички коментари.

**PayPalCheckoutInfo**

public class PayPalCheckoutInfo

{

#region "Transaction and Notification-Related Variables"

/// use this to verify its not spoofed, this is our info

public string receiver\_email { get; set; } //127

public string receiver\_id { get; set; } //13

/// Keep this ID to avoid processing the transaction twice

/// The merchant’s original transaction identification number for the payment from the

/// buyer, against which the case was registered.

public string txn\_id { get; set; }

/// The kind of transaction for which the IPN message was sent.

public string txn\_type { get; set; }

/// Encrypted string used to validate the authenticity of the transaction

public string verify\_sign { get; set; }

#endregion

#region "Buyer Information Variables"

public string address\_country { get; set; } //64

public string address\_city { get; set; } //40

public string address\_country\_code { get; set; } //2

public string address\_name { get; set; } //128 - prob don't need

public string address\_state { get; set; } //40

public string address\_status { get; set; } //confirmed/unconfirmed

public string address\_street { get; set; } //200

public string address\_zip { get; set; } //20

public string contact\_phone { get; set; } //20

public string first\_name { get; set; } //64

public string last\_name { get; set; } //64

public string payer\_email { get; set; } //127

public string payer\_id { get; set; } //13

public int? Zip

{

get

{

int temp;

if (int.TryParse(address\_zip, out temp))

{

return temp;

}

else

{

return null;

}

}

}

#endregion

#region "Payment Information Variables"

/\*

public string auth\_amount { get; set; }

public string auth\_exp { get; set; } //28

public string auth\_id { get; set; } //19

public string auth\_status { get; set; }

\*/

/// Token passed back from PayPal for cross ref

public string token { get; set; }

//public string checkoutstatus { get; set; }

/// Passthrough variable you can use to identify your Invoice Number for this

/// purchase. If omitted, no variable is passed back.

public string invoice { get; set; } //127

/// Item name as passed by you, the merchant. Or, if not passed by you, as

/// entered by your customer. If this is a shopping cart transaction, PayPal

/// will append the number of the item (e.g., item\_name1, item\_name2,

/// and so forth).

public string item\_name1 { get; set; } //127

public string item\_number1 { get; set; } //127

public string mc\_currency { get; set; } //currency of the payment.

public string mc\_fee { get; set; } //Transaction fee associated with the payment

public string mc\_gross { get; set; } //Full amount of the customer's payment

/// Whether the customer has a verified PayPal account.

/// verified – Customer has a verified PayPal account

/// unverified – Customer has an unverified PayPal account.

public string payer\_status { get; set; }

/// HH:MM:SS Mmm DD, YYYY PDT (28chars)

public string payment\_date { get; set; }

public DateTime TrxnDate

{

get

{

DateTime dt = DateTime.Now;

if (DateTime.TryParse(payment\_date, out dt))

{

return dt;

}

else

{

return DateTime.Now;

}

}

}

/// The status of the payment:

/// Canceled\_Reversal: A reversal has been canceled. For example, you

/// won a dispute with the customer, and the funds for the transaction that was

/// reversed have been returned to you.

/// Completed: The payment has been completed, and the funds have been added

/// successfully to your account balance

/// Created: A German ELV payment is made using Express Checkout. Denied: You denied

/// the payment. This happens only if the payment was

/// previously pending because of possible reasons described for the pending\_reason

/// variable or the Fraud\_Management\_Filters\_x variable.

/// Expired: This authorization has expired and cannot be captured.

///Failed: The payment has failed. This happens only if the payment was made from your

///customer’s bank account.

///Pending: The payment is pending. See pending\_reason for more information.

///Refunded: You refunded the payment.

///Reversed: A payment was reversed due to a chargeback or other type of reversal. The

///funds have been removed from your account balance and

///returned to the buyer. The reason for the reversal is specified in the ReasonCode

///element.

///Processed: A payment has been accepted.

///Voided: This authorization has been voided.

public string payment\_status { get; set; }

/// echeck: This payment was funded with an eCheck.

/// instant: This payment was funded with PayPal balance, credit card, or Instant

///Transfer.

public string payment\_type { get; set; }

/// This variable is set only if payment\_status = Pending. - too many reasons (look it

///up in pdf)

public string pending\_reason { get; set; }

public string protection\_eligibility { get; set; }

public string quantity { get; set; }

public string reason\_code { get; set; }

public string correlationID { get; set; }

public string ack { get; set; }

public string errmsg { get; set; }

public int? errcode { get; set; }

/// should hold the clientid passed in from setexpresscheckout

public string custom { get; set; }

public decimal Total

{

get

{

decimal amount = 0;

if (decimal.TryParse(mc\_gross, out amount))

{

return amount;

}

else

{

return 0;

}

}

}

public decimal Fee

{

get

{

decimal amount = 0;

if (decimal.TryParse(mc\_fee, out amount))

{

return amount;

}

else

{

return 0;

}

}

}

#endregion

}

PayPalListenerModel е класът, който осъществява проверката за истиност на получения модел от PayPalAPI. След получаването на модела, този клас има грижата да извлече съдържанието му, да го парсне в подходящ вид и да изпрати запитване до издаващия орган за валидност на плащането. Едва след потвърждаване на валидността може да се обработи останалата част от информацията. PayPalListenerModel класът е взет от примерна имплементация, побликувана свободно на developer.paypal.com. Кодът е оставен в оригинал, заедно с коментарите.

**PayPalListenerModel**

public class PayPalListenerModel

{

public PayPalCheckoutInfo \_PayPalCheckoutInfo { get; set; }

public void GetStatus(byte[] parameters)

{

string status = Verify(true, parameters);

if (status == "VERIFIED")

{

//check that the payment\_status is Completed

if (\_PayPalCheckoutInfo.payment\_status.ToLower() == "completed")

{

//check that txn\_id has not been previously processed to prevent //duplicates

//check that receiver\_email is your Primary PayPal email

//check that payment\_amount/payment\_currency are correct

//process payment/refund/etc

}

else if (status == "INVALID")

{

//log for manual investigation

}

else

{

//log response/ipn data for manual investigation

}

}

}

private string Verify(bool isSandbox, byte[] parameters)

{

string response = "";

try

{

string url = isSandbox ?

"https://www.sandbox.paypal.com/cgi-bin/webscr" : "https://www.paypal.com/cgi-bin/webscr";

HttpWebRequest webRequest = (HttpWebRequest)WebRequest.Create(url);

webRequest.Method = "POST";

webRequest.ContentType = "application/x-www-form-urlencoded";

//must keep the original intact and pass back to PayPal with a \_notify- //validate command

string data = Encoding.ASCII.GetString(parameters);

data += "&cmd=\_notify-validate";

webRequest.ContentLength = data.Length;

//Send the request to PayPal and get the response

using (StreamWriter streamOut = new StreamWriter(webRequest.GetRequestStream(), System.Text.Encoding.ASCII))

{

streamOut.Write(data);

streamOut.Close();

}

using (StreamReader streamIn = new StreamReader(webRequest.GetResponse().GetResponseStream()))

{

response = streamIn.ReadToEnd();

streamIn.Close();

}

}

catch { }

return response;

}

}

## 4.3. Subscription Controller

Subscription контролерът има 2 метода и се грижи за изобразяването на информацията за абонамента на потребителите.

**Index** методът извлича данни от базата и изготвя SubscriptionsCreditsViewModel, който се използва за изготвянето на PayPal бутоните, които бяха разгледани по-рано в Subscription страницата.

**GetSubscriptionCredits** методът извлича данни от базата и изготвя SubscriptionCreditsModel, който се използва при изготвянето на Footer елемента, по конкретно централната му част с информация за абонамента и линкове към Subscription страницата.

**SubscriptionController**

public class SubscriptionController : BaseController

{

[Authorize]

public ActionResult Index()

{

string userId = String.Empty;

userId = HttpContext.User.Identity.GetUserId();

AspNetUser user = DB.AspNetUsers.Find(userId);

PersonalDetail personalDetail = user.PersonalDetail;

SubscriptionCreditsModel subscriptionCreditsModel = new SubscriptionCreditsModel

{

ActiveUntil = personalDetail.SubscriptionPlan.ActiveUntil,

Credits = personalDetail.SubscriptionPlan.Credits,

UnlimitedAccess = personalDetail.SubscriptionPlan.UnlimitedAccess

};

List<ProductModel> products = DB.Products.Where(product => (product.IsActive && (product.ProductTypeId == 1 || product.ProductTypeId == 2))).Select(product => new ProductModel

{

Id = product.Id,

ProductTypeId = product.ProductTypeId,

Name = product.Name,

Display = product.Display,

Value = product.Value,

Currency = product.Currency

}).ToList();

SubscriptionsCreditsViewModel model = new SubscriptionsCreditsViewModel

{

SubscriptionCreditsModel = subscriptionCreditsModel,

Products = products

};

return View(model);

}

[NonAction]

public SubscriptionCreditsModel GetSubscriptionCredits(string userId)

{

AspNetUser user = DB.AspNetUsers.Find(userId);

if (user != null)

{

PersonalDetail personalDetail = user.PersonalDetail;

if (personalDetail == null)

{

Address address = DB.Addresses.Add(new Address() { });

Contact contact = DB.Contacts.Add(new Contact() { });

SubscriptionPlan subscriptionPlan = DB.SubscriptionPlans.Add(new SubscriptionPlan()

{

Credits = 0,

ActiveUntil = DateTime.Today.AddDays(-1),

UnlimitedAccess = false

});

personalDetail = DB.PersonalDetails.Add(new PersonalDetail()

{

FirstName = "",

LastName = "",

AddressId = address.Id,

ContactId = contact.Id,

SubscriptionPlanId = subscriptionPlan.Id,

});

user.PersonalDetailsId = personalDetail.Id;

DB.SaveChanges();

}

SubscriptionCreditsModel model = new SubscriptionCreditsModel()

{

ActiveUntil = personalDetail.SubscriptionPlan.ActiveUntil,

Credits = personalDetail.SubscriptionPlan.Credits,

UnlimitedAccess = personalDetail.SubscriptionPlan.UnlimitedAccess,

};

return model;

}

else

{

return null;

}

}

}

## 4.4. LogController

Log контролерът се грижи за изготвянето на отчети за клиентите, съответно за техните посещения и извършените от тях плащания. Създадените от 2-та метода на контролера модели се подават към съответните изгледи, които вече бяха разгледани.

**Log Controller**

[Authorize(Roles = "Client")]

public class LogController : BaseController

{

public ActionResult Visits()

{

string userID = this.User.Identity.GetUserId();

AspNetUser user = DB.AspNetUsers.Find(userID);

IEnumerable<EntranceHistoryItemModel> model = new List<EntranceHistoryItemModel>();

if (user != null)

{

List<EntranceHistory> entranceHistories = new List<EntranceHistory>();

foreach (Card card in user.Cards)

{

entranceHistories.AddRange(card.EntranceHistories);

}

model = entranceHistories.OrderBy(entranceHistory => entranceHistory.CreatedAt).

Select(entranceHistory => new EntranceHistoryItemModel

{

UserName = EntityHelper.GetFullName(user.Id),

Date = entranceHistory.CreatedAt.HasValue ? entranceHistory.CreatedAt.Value.ToString("dd/MM/yyyy", CultureInfo.InvariantCulture) : String.Empty,

Time = entranceHistory.CreatedAt.HasValue ? entranceHistory.CreatedAt.Value.ToString("hh:mm", CultureInfo.InvariantCulture) : String.Empty,

CardNumber = entranceHistory.Card.CardId,

EntryStatus = entranceHistory.EntryStatus,

FacilityName = entranceHistory.Facility.Name,

}).ToList();

}

return View(model);

}

public ActionResult Payments()

{

string userID = this.User.Identity.GetUserId();

AspNetUser user = DB.AspNetUsers.Find(userID);

IEnumerable<PaymentHistoryItemModel> model = new List<PaymentHistoryItemModel>();

if (user != null)

{

model = user.PaymentHistories.OrderBy(paymentHistory => paymentHistory.CreatedAt).Select(paymentHistory => new PaymentHistoryItemModel

{

Date = paymentHistory.CreatedAt.HasValue ? paymentHistory.CreatedAt.Value.ToString("dd/MM/yyyy", CultureInfo.InvariantCulture) : String.Empty,

Time = paymentHistory.CreatedAt.HasValue ? paymentHistory.CreatedAt.Value.ToString("hh:mm", CultureInfo.InvariantCulture) : String.Empty,

IssuerName = EntityHelper.GetFullName(paymentHistory.IssuerId),

RecepientName = EntityHelper.GetFullName(paymentHistory.RecepientId),

ProductName = paymentHistory.Product.Name,

PayPalEmail = paymentHistory.PayPalEmail

}).ToList();

}

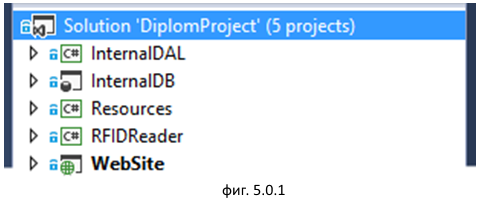
return View(model);

}

}

# Глава 5. Описание на програмното осигуряване

Проектът **DiplomProject,** който съдържа цялата дипломна работа е разделен условно на 5 по-малки проекта, чиято структура ще бъде разгледана (фиг. 5.0.1), както и приложението Mifare1k, предоставено от фирмата производител на RFID четеца, който е използван.

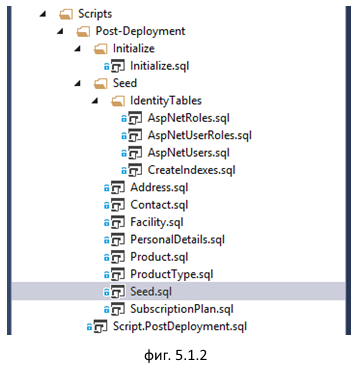


* InternalDAL проекта, грижещ се за EntityFramework ORM-а
* InternalDB проекта, грижещ се за създаването на базата и тестовите данни
* Resources проекта, грижещ се за .resx файловете за интернационализация
* RFIDReader проекта, грижещ се за четенето на RFID картите.
* WebSite проекта, грижещ се за MVC и WebApi частта и комуникацията
* Приложението Mifare1k

## 5.1. Проект InternalDB

Проектът InternalDB представлява SQL Project във VisualStudio. (фиг. 5.1.1) Неговата цел е да съдържа SQL файловете, които образуват базата от данни (Глава 3) Удобното при този тип проекти е възможността да образуваме Publish Preview, което създава комбиниран SQL скрипт, който може да бъде изпълнен, създавайки цялата база, както и тестовите данни за нея.

Тъй като вече беше разгледано подробно съдържанието на sql файловете в глава 3, ще бъдат описани тестовите данни, които INSERT-ваме във вече създадената база. Те се намират в директорията Scripts, която има следната структура. (фиг. 5.1.2)



Интересното тук са не толкова SQL файловете с INSERT заявки за отделните таблици, колкото Seed.sql и Script.PostDepoloyment.sql файловете. (фиг.5.1.2)

**Script.PostDeployment.sql**

-- Insert DB initialization data

:r .\Initialize\Initialize.sql

BEGIN

:r .\Seed\Seed.sql

END

Този файл е с точно определено име и по конвенция кодът в него се добавя към Publish файла, след като вече са създадени всички таблици. Това, което той прави е да инициализира празния файл Initialize.sql, за да има къде да се редят INSERT заявките от другите файлове и да извика Seed.sql файла.

**Seed.sql**

-- REQUIRES: none

:r .\IdentityTables\CreateIndexes.sql

:r .\IdentityTables\AspNetRoles.sql

:r .\SubscriptionPlan.sql

:r .\ProductType.sql

:r .\Contact.sql

:r .\Address.sql

----------------------------------------------

-- REQUIRES dbo.ProductType

:r .\Product.sql

-- REQUIRES dbo.SubscriptionPlan, dbo.Contact, dbo.Address

:r .\PersonalDetails.sql

----------------------------------------------

-- REQUIRES dbo.AspNetRoles, dbo.PersonalDetails

:r .\IdentityTables\AspNetUsers.sql

----------------------------------------------

-- REQUIRES dbo.AspNetRoles, dbo.AspNetUsers

:r .\IdentityTables\AspNetUserRoles.sql

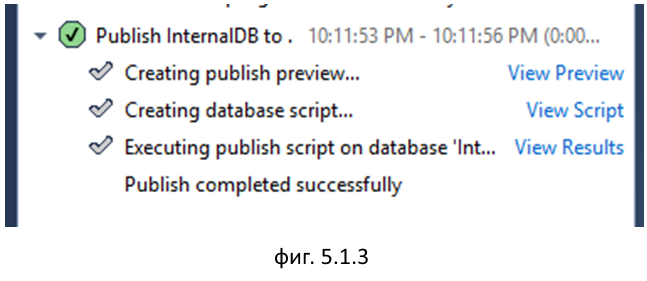
----------------------------------------------

-- REQUIRES dbo.Contact, dbo.Address, dbo.AspNetUsers

:r .\Facility.sql

----------------------------------------------

Работата на Seed.sql файла е да укаже реда, в който ще се зареждат .sql файловете, също така --REQUIRES директивата указва на билдъра, кои файлове биват изискани преди да може да се изпълни определен SQL. По този начин се решава проблема с ограниченията (Constrains), които са зададени като външни ключове и връзки между таблиците.



В крайна сметка се създава Publish скрипт, който публикува базата на съответната инстанция на MSSQL сървъра. Тъй като генерираният скрипт, заедно с тестовите данни е прекалено голям, ще бъде отбелязан само резултатът от действието (фиг. 5.1.3), а именно успешно създадена база, таблици и релационни връзки.

Creating InternalDB...

Creating [dbo].[Address]...

Creating [dbo].[AspNetRoles]...

Creating [dbo].[AspNetUserClaims]...

Creating [dbo].[AspNetUserLogins]...

Creating [dbo].[AspNetUserRoles]...

Creating [dbo].[AspNetUsers]...

Creating [dbo].[Card]...

Creating [dbo].[Contact]...

Creating [dbo].[EntranceHistory]...

Creating [dbo].[Facility]...

Creating [dbo].[PaymentHistory]...

Creating [dbo].[PersonalDetails]...

Creating [dbo].[Product]...

Creating [dbo].[ProductType]...

Creating [dbo].[SubscriptionPlan]...

Creating FK\_AspNetUserClaims\_AspNetUsers...

Creating FK\_AspNetUserLogins\_AspNetUsers...

Creating FK\_AspNetUserRoles\_AspNetRole...

Creating FK\_AspNetUserRoles\_AspNetUser...

Creating FK\_AspNetUsers\_PersonalDetails...

Creating FK\_Card\_UserProfile...

Creating FK\_EntranceHistory\_Card...

Creating FK\_EntranceHistory\_Facility...

Creating FK\_Facility\_AspNetUsers...

Creating FK\_Facility\_Address...

Creating FK\_Facility\_Contact...

Creating FK\_PaymentHystory\_Recepient...

Creating FK\_PaymentHystory\_Issuer...

Creating FK\_PaymentHystory\_Product...

Creating FK\_PersonalDetails\_Address...

Creating FK\_PersonalDetails\_Contact...

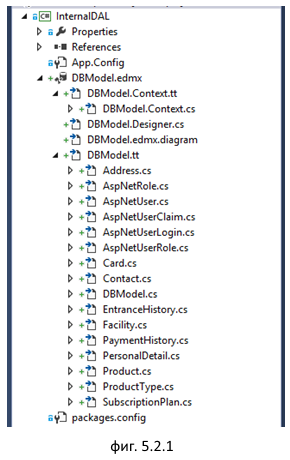
Creating FK\_PersonalDetails\_SubscriptionPlan...

Creating FK\_Product\_ProductType...

Update complete.

## 

## 5.2. Проект InternalDAL

 InternalDAL проектът е частта от системата, която се грижи за генерирането на ORM (Object-relational mapping) модела. (фиг. 5.2.1)

Следвайки помощника (Wizard) за създаване на модел, само чрез посочване на базата, която предстои да бъде мапната (описана), се генерира целия Entity Framework модел, заедно с неговите връзки, които могат да бъдат използвани от тук насетне. В DBModel.tt файла, който е частичен клас (PartialClass) на модела DBModel.edmx, всички таблици съществуват като .cs файлове. Всеки от тези файлове има аналогично наименуван клас, с пропъртита, същите като колоните на съответната таблица, и връзки към таблиците, според външните ключове, които съществуват в базата. Това може да се види много добре на диаграмата на модела (фиг. 5.2.2), която ясно показва пропъртитата на всеки клас, както и неговите връзки с другите класове. Интересно свойство, което тези връзки предоставят е възможността, да се работи с дадена колекция (записи в таблица), чрез връзките и с външните ключове (FK) на друга таблица. Това означава, че ако например се работи с конкретен запис от таблицата dbo.Card, представен в модела като единична инстанция на класа Card, то през неговите пропъртита Address.AspNetUser и Address.EntranceHistories, могат да бъдат достъпни за картата, списък от обекти от EntranceHistory класа, които имат като външен ключ CardId - тази карта (връзка 1:m), както и AspNetUser, но в този случай (връзка m:1), няма да бъде получен списък от AspNetUsers обекти, а един единствен AspNetUser обект, на когото принадлежи тази карта.

Естествено EntityFramework е достатъчно добре разработен, за да не праща излишни заявки до базата за всяко извикване на пропърти от обект. Има няколко различни типа поведение, които могат да бъдат настроени допънително, позволявайки на EntityFramework да кешира част от данните. Например, при извикване на дадена таблица, или дори преди извикването и, да се заредят всички свързани с нея записи с 1 заявка, за да намали броят на заявките до базата, което е за сметка на риск за консистентността на данните. Този вид поведение се нарича Eager Loading, когато рамката (framework) иска да вземе повече данни от необходимото, тъй като може да са необходими в непосредствено бъдеще.. Обратното също е възможно, всяко действие да бъде извършено до базата изрично за себе си, без зареждане на излишни данни. (Explicit Loading). Това поведение осигурява максимална консистентност на данните, но е за сметка на чести заявки до базата, което може да намали бързодействието. 3-я случай комбинира първите 2 (Lazy Loading). Той преценя кои полета да вземе, при първото извикване на дадено пропърти. По подразбиране, в текущата дипломна работа ще бъде използвано именно това поведение.

Интерес представлява още InternalDBEntities класът, който наследява DbContext класа. DbContext е класът, който се грижи за връзката с базата. Неговия SaveChanges() метод записва всички промени, правени до момента (които до сега били запазени само в локалното копие на модела в паметта) в базата. Тъй като DbContext работи с транзакции, ако някое ограничение (constraint) е нарушено, никои от промените не се запазват. Класът InternalDBEntities, разширява това поведение, добавяйки колекциите от класовете, генерирани от помощника. На практика се работи през този обект с цялата база. По-рано беше обяснено как се разширява обекта за лесно използване от контролерите и автоматично попълване на датата на създаване и модифициране, както и кой е извършил тези действия.

public partial class InternalDBEntities : DbContext

{

public InternalDBEntities()

: base("name=InternalDBEntities")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

throw new UnintentionalCodeFirstException();

}

public virtual DbSet<Address> Addresses { get; set; }

public virtual DbSet<AspNetRole> AspNetRoles { get; set; }

public virtual DbSet<AspNetUserClaim> AspNetUserClaims { get; set; }

public virtual DbSet<AspNetUserLogin> AspNetUserLogins { get; set; }

public virtual DbSet<AspNetUserRole> AspNetUserRoles { get; set; }

public virtual DbSet<AspNetUser> AspNetUsers { get; set; }

public virtual DbSet<Card> Cards { get; set; }

public virtual DbSet<Contact> Contacts { get; set; }

public virtual DbSet<EntranceHistory> EntranceHistories { get; set; }

public virtual DbSet<Facility> Facilities { get; set; }

public virtual DbSet<PaymentHistory> PaymentHistories { get; set; }

public virtual DbSet<PersonalDetail> PersonalDetails { get; set; }

public virtual DbSet<Product> Products { get; set; }

public virtual DbSet<ProductType> ProductTypes { get; set; }

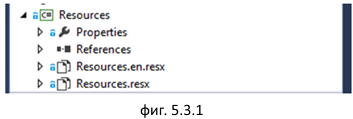
public virtual DbSet<SubscriptionPlan> SubscriptionPlans { get; set; }

}

Обектната диаграма (фиг. 5.2.2), която описва визуално връзките между създадените обекти силно наподобява ER диаграмата, която описва релационния модел на базата от данни.

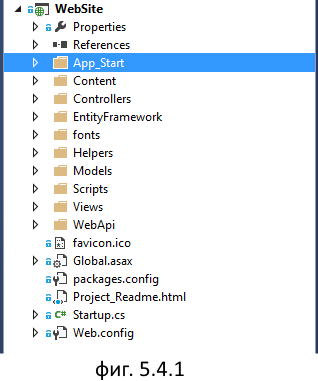


## 5.3. Проект Resources

 Проектът Resources (фиг. 5.3.1) e най-малкият от проектите. Той се състои единствено от 2 .resx файла, които имат грижата да осигурят двуезичността на проекта. Съдържанието на файловете и тяхното използване са описани подробно в Глава 2.

## 

## 5.4. Проект WebSite

 Проектът WebSite е ASP.NET приложение, с активирани MVC и WebApi модули. (фиг. 5.4.1) В себе си то съдържа както логиката, която инициализира цялото MVC5 приложение, така и всички необходими файлове за неговото правилно функциониране. Контролерите в директроията Controllers, изгледите в папката Views, моделите в директорията Models , както и сървърните контролери в директорията WebApi.

Тъй като огромна част от класовете на проекта вече са описани, ще бъдат разгледани единствено структурата му и основните моменти в архитектурата на приложението.

* **App\_Start** директорията съдържа класовете, които се инстанцират при самото стартиране на приложението. Такива например са BundleConfig и RouteConfig класовете, които съответно се грижат за CSS и JavaScript бъндълите и за раутинга (определянето на пътя до контролерите и методите).
* **Content** директорията съдържа всички CSS файлове. Вече беше разгледано как се използва BundleConfig за централизираното им ползване..
* **Controllers** директорията съдържа всички MVC контролери.
* **EntityFramework** директроята съдържа Helper класовете, които се използват за връзка с ORM модела, например ExtendedDBContext и BaseController, които вече бяха разгледани.
* **fonts** директорията съдържа Glyphicon файловете, които Bootstrap използва за изчертаването на SVG икони.
* **Helpers** директорията в момента съдържа единствено CultureHelper класа, който се грижи за двуезичието на системата.
* **Models** директорията съдържа всички използвани модели
* **Scripts** директрояита съдържа всички използвани JavaScript файлове
* **Views** директорията съдържа всички използвани изгледи
* **WebApi** директорията съдържа AccessControlController и PaymentSystemController WebApi класовете.
* **Global.asax** файлът е входната точка на приложението. По точно неговият метод Application\_Start()
* **packages.config** файлът съдържа описанието на използваните пакети, включени в приложението
* **Web.config** файлът съдържа използваната конфигурация на приложението

## Global.asax

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

GlobalConfiguration.Configure(WebApiConfig.Register);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

RegisterCors(MvcCorsConfiguration.Configuration);

}

private void RegisterCors(MvcCorsConfiguration corsConfig)

{

corsConfig

.ForResources("\*")

.ForOrigins("\*")

.AllowAll();

}

}

**packages.config**

<?xml version="1.0" encoding="utf-8"?>

<packages>

<package id="Antlr" version="3.4.1.9004" targetFramework="net45" />

<package id="bootstrap" version="3.0.0" targetFramework="net45" />

<package id="ECommerceHelper.Common" version="0.3.3" targetFramework="net45" />

<package id="ECommerceHelper.CurrencyConverter" version="0.3.3" targetFramework="net45" />

<package id="EntityFramework" version="6.0.0" targetFramework="net45" />

<package id="jQuery" version="1.10.2" targetFramework="net45" />

<package id="jQuery.Validation" version="1.11.1" targetFramework="net45" />

<package id="Microsoft.AspNet.Cors" version="5.2.2" targetFramework="net45" />

<package id="Microsoft.AspNet.Identity.Core" version="1.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.Identity.EntityFramework" version="1.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.Identity.Owin" version="1.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.Mvc" version="5.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.Razor" version="3.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.Web.Optimization" version="1.1.1" targetFramework="net45" />

<package id="Microsoft.AspNet.WebApi" version="5.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.WebApi.Client" version="5.2.2" targetFramework="net45" />

<package id="Microsoft.AspNet.WebApi.Core" version="5.2.2" targetFramework="net45" />

<package id="Microsoft.AspNet.WebApi.Cors" version="5.2.2" targetFramework="net45" />

<package id="Microsoft.AspNet.WebApi.WebHost" version="5.0.0" targetFramework="net45" />

<package id="Microsoft.AspNet.WebPages" version="3.0.0" targetFramework="net45" />

<package id="Microsoft.CompilerServices.AsyncTargetingPack" version="1.0.1" targetFramework="net45" />

<package id="Microsoft.jQuery.Unobtrusive.Validation" version="3.0.0" targetFramework="net45" />

<package id="Microsoft.Net.Http" version="2.0.20710.0" targetFramework="net45" />

<package id="Microsoft.Owin" version="2.1.0" targetFramework="net45" />

<package id="Microsoft.Owin.Host.SystemWeb" version="2.0.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security" version="2.1.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security.Cookies" version="2.0.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security.Facebook" version="2.1.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security.Google" version="2.1.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security.MicrosoftAccount" version="2.0.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security.OAuth" version="2.0.0" targetFramework="net45" />

<package id="Microsoft.Owin.Security.Twitter" version="2.0.0" targetFramework="net45" />

<package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net45" />

<package id="Modernizr" version="2.6.2" targetFramework="net45" />

<package id="Newtonsoft.Json" version="6.0.5" targetFramework="net45" />

<package id="Owin" version="1.0" targetFramework="net45" />

<package id="Respond" version="1.2.0" targetFramework="net45" />

<package id="System.IdentityModel.Tokens.Jwt" version="1.0.0" targetFramework="net45" />

<package id="Thinktecture.IdentityModel" version="3.6.1" targetFramework="net45" />

<package id="WebGrease" version="1.5.2" targetFramework="net45" />

</packages>

**Web.config**

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<configSections>

<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false"/>

</configSections>

<connectionStrings>

<add name="DBConnection" providerName="System.Data.SqlClient" connectionString="Data Source=.;

Initial Catalog=InternalDB;

Integrated Security=True;

MultipleActiveResultSets=True"/>

<add name="InternalDBEntities" connectionString="metadata=res://\*/DBModel.csdl|res://\*/DBModel.ssdl|res://\*/DBModel .msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=.;initial catalog=InternalDB;integrated security=True;multipleactiveresultsets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient"/>

</connectionStrings>

<appSettings>

<add key="webpages:Version" value="3.0.0.0"/>

<add key="webpages:Enabled" value="false"/>

<add key="ClientValidationEnabled" value="true"/>

<add key="UnobtrusiveJavaScriptEnabled" value="true"/>

</appSettings>

<system.web>

<globalization culture="auto" uiCulture="auto"/>

<authentication mode="None"/>

<compilation debug="true" targetFramework="4.5"/>

<httpRuntime targetFramework="4.5"/>

</system.web>

<system.webServer>

<modules>

<remove name="FormsAuthenticationModule"/>

</modules>

<handlers>

<remove name="ExtensionlessUrlHandler-Integrated-4.0"/>

<remove name="OPTIONSVerbHandler"/>

<remove name="TRACEVerbHandler"/>

<add name="ExtensionlessUrlHandler-Integrated-4.0" path="\*." verb="\*" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0"/>

</handlers>

</system.webServer>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="System.Net.Http.Formatting" publicKeyToken="31BF3856AD364E35" culture="neutral"/>

<bindingRedirect oldVersion="0.0.0.0-5.2.2.0" newVersion="5.2.2.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30AD4FE6B2A6AEED" culture="neutral"/>

<bindingRedirect oldVersion="0.0.0.0-6.0.0.0" newVersion="6.0.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Http" publicKeyToken="31BF3856AD364E35" culture="neutral"/>

<bindingRedirect oldVersion="0.0.0.0-5.2.2.0" newVersion="5.2.2.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="1.0.0.0-5.0.0.0" newVersion="5.0.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="WebGrease" publicKeyToken="31bf3856ad364e35"/>

<bindingRedirect oldVersion="0.0.0.0-1.5.2.14234" newVersion="1.5.2.14234"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin" publicKeyToken="31bf3856ad364e35" culture="neutral"/>

<bindingRedirect oldVersion="0.0.0.0-2.1.0.0" newVersion="2.1.0.0"/>

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin.Security" publicKeyToken="31bf3856ad364e35" culture="neutral"/>

<bindingRedirect oldVersion="0.0.0.0-2.1.0.0" newVersion="2.1.0.0"/>

</dependentAssembly>

</assemblyBinding>

</runtime>

<entityFramework>

<defaultConnectionFactory type="System.Data.Entity.Infrastructure.

LocalDbConnectionFactory, EntityFramework">

<parameters><parameter value="v11.0"/></parameters>

</defaultConnectionFactory>

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.

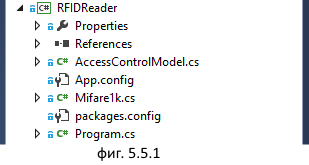
SqlProviderServices, EntityFramework.SqlServer"/>

</providers>

</entityFramework>

</configuration>

## 5.5. Проект RFID Reader

 RFDI Reader проектът (фиг. 5.5.1) е конзолно приложение, имащо за цел да използва RFID reader, като периодично прочита за поставена карта, изпраща заявка до AccessControlController-a на сървъра, ако такава карта е засечена, с данни за устройството - четец, номера на картата и персонални данни за потребителя, записани в паметта на картата. Тези данни биват достатъчни за идентифицирането на потребителя и определяне статуса на неговото запитване за допуск до помещенията.

Фирмата производител на RFID четеца предоставя библиотеката **MasterCom.dll,** притежаваща набор от функции за работа с устройството и специфични за работа с картите от туп Mifare1k. Пълен набор на функциите достъпни чрез библиотеката може да бъде намерен в потребителското ръководство на 13.56 MHz Reader / Writer SL500 (Номер 9 в приложенията).

**AccessControlModel.cs** е файла, съдържащ модела с данните, които се изпращат на сървъра.

public class AccessControlModel

{

public short Icdev { get; set; }

public string CardNumber { get; set; }

public string UserDetails { get; set; }

}

Както се вижда, това е целоцифреното число Icdev (Идентификационния номер на четеца), CardNumber - 4 битовият номер на картата във шестнадесетичен (Hex) формат, както и UserDetails - репрезентация на GUID на потребителя, използван за допълнително идентифициране на клиента.

**Myfare1k.cs** файлът съдържа класа Mifare1k, който работи с устройството. Този клас има за цел да извършва комуникацията с предоставената от фирмата-производител динамична библиотека (DLL файл), осигуряваща връзката с устройството, както и да изнесе някои публични методи за общо ползване, в частност методите:

public void ChangeColor(short icdev, short color)

Метод, позволяващ да сменим цвета на светодиода на устройството. Входни параметри са идентификационния номер на RFID четеца и номера на цвета от предварително определено множество.

public void Beep(short icdev, short duration)

Метод, позволяващ да активираме звуковата сигнализация на устройството за определено време. Входни параметри са идентификационния номер на RFID четеца и продължителността на сигнала в милисекунди (от 1 до 99).

public short InitializeRFIDReader(int port, int baud)

Метод, инициализиращ четеца. Осигурява връзка през определен COM порт на машината и дадена скорост на предаване. Входни параметри са COM порта, в който е включен RFID четецът и скоростта на предаване.

public string ReadCard(short icdev)

Метод, прочитащ номера на картата. Входен параметър е идентификационния номер на RFID четеца.

public string ReadBlock(short icdev, char type, string keyB, int sectorNumber, int blockNumber)

Метод, прочитащ даден блок, намиращ се в даден сектор от паметта на картата. Методът осигурява и вградена аутентикация на валидността на картата. При успешна аутентикация, се дава възможност за четене и запис по блоковете памет на картата. Необходимо условие за аутентикация е наличието на един от двата A или B ключа, които биват използвани за удостоверяване на правомерен достъп. Входни параметри са идентификационния номер на RFID четеца, модела на картата, която бива достъпвана, именуваният блок памет с ключа ('A' или 'B'), както и самият ключ (По подразбиране ключовете за достъп са 6 битови и са 000000000000 за А и FFFFFFFFFFFF за B). Ключовете, с които системата бива разработвана и тествана са тези по подразбиране. Естествено при експлоатацията и биха били подменени с други. Останалите 2 параметъра са номера на сектора и номера на блока, съдържанието на които да бъде прочетено.

**Myfare1k**

public class Mifare1K

{

[DllImport("kernel32.dll")]

static extern void Sleep(int dwMilliseconds);

[DllImport("MasterRD.dll")]

static extern int rf\_init\_com(int port, int baud);

[DllImport("MasterRD.dll")]

static extern int rf\_ClosePort();

[DllImport("MasterRD.dll")]

static extern int rf\_antenna\_sta(short icdev, byte mode);

[DllImport("MasterRD.dll")]

static extern int rf\_init\_type(short icdev, byte type);

[DllImport("MasterRD.dll")]

static extern int rf\_request(short icdev, byte mode, ref ushort pTagType);

[DllImport("MasterRD.dll")]

static extern int rf\_anticoll(short icdev, byte bcnt, IntPtr pSnr, ref byte pRLength);

[DllImport("MasterRD.dll")]

static extern int rf\_select(short icdev, IntPtr pSnr, byte srcLen, ref sbyte Size);

[DllImport("MasterRD.dll")]

static extern int rf\_M1\_authentication2(short icdev, byte mode, byte secnr, IntPtr key);

[DllImport("MasterRD.dll")]

static extern int rf\_M1\_read(short icdev, byte adr, IntPtr pData, ref byte pLen);

[DllImport("MasterRD.dll")]

static extern int rf\_get\_device\_number(out ushort pIcdev);

[DllImport("MasterRD.dll")]

static extern int rf\_beep(short icdev, short msec);

[DllImport("MasterRD.dll")]

static extern int rf\_light(short icdev, short color);

bool bConnectedDevice;

static char[] hexDigits = {

'0','1','2','3','4','5','6','7',

'8','9','A','B','C','D','E','F'};

// 0 - off; 1 - red; 2 - green; 3 - yellow

public void ChangeColor(short icdev, short color)

{

rf\_light(icdev, color);

}

public void Beep(short icdev, short duration)

{

rf\_beep(icdev, duration);

}

public short InitializeRFIDReader(int port, int baud)

{

ushort uicdev = 0;

short icdev = 0;

if(!bConnectedDevice)

{

int status;

status = rf\_init\_com(port, baud);

if (0 == status)

{

bConnectedDevice = true;

Console.WriteLine("Connect device success!");

int deviceStatus = rf\_get\_device\_number(out uicdev);

if (deviceStatus == 0)

{

icdev = short.Parse(uicdev.ToString());

rf\_beep(icdev, short.Parse("50"));

}

}

else

{

string strError;

bConnectedDevice = false;

Console.WriteLine("Connect device failed");

}

}

else{

rf\_ClosePort();

bConnectedDevice = false;

}

return icdev;

}

public string ReadCard(short icdev)

{

int status;

byte type = (byte)'A'; // Mifare Type A

byte mode = 0x52;

ushort TagType = 0;

byte bcnt = 0x04;

IntPtr pSnr;

byte len = 255;

sbyte size = 0;

pSnr = Marshal.AllocHGlobal(1024);

for (int i = 0; i < 2; i++)

{

status = rf\_antenna\_sta(icdev, 0);

if (status != 0)

continue;

Sleep(20);

status = rf\_init\_type(icdev, type);

if (status != 0)

continue;

Sleep(20);

status = rf\_antenna\_sta(icdev, 1);

if (status != 0)

continue;

Sleep(50);

status = rf\_request(icdev, mode, ref TagType);

if (status != 0)

continue;

status = rf\_anticoll(icdev, bcnt, pSnr, ref len);

if (status != 0)

continue;

status = rf\_select(icdev, pSnr, len, ref size);

if (status != 0)

continue;

byte[] szBytes = new byte[len];

for (int j = 0; j < len; j++)

{

szBytes[j] = Marshal.ReadByte(pSnr, j);

}

String m\_cardNo = String.Empty;

for (int q = 0; q < len; q++)

{

m\_cardNo += byteHEX(szBytes[q]);

}

Marshal.FreeHGlobal(pSnr);

return m\_cardNo;

}

Marshal.FreeHGlobal(pSnr);

throw new Exception("No card detected");

}

public string ReadBlock(short icdev, char type, string keyB, int sectorNumber, int blockNumber)

{

int status;

byte mode = (byte)0x60;

if (type == 'B')

mode = (byte)0x61;

byte secnr = Convert.ToByte(sectorNumber);

IntPtr keyBuffer = Marshal.AllocHGlobal(1024);

byte[] bytesKey = ToDigitsBytes(keyB);

for (int i = 0; i < bytesKey.Length; i++)

Marshal.WriteByte(keyBuffer, i, bytesKey[i]);

status = rf\_M1\_authentication2(icdev, mode, (byte)(secnr \* 4), keyBuffer);

Marshal.FreeHGlobal(keyBuffer);

if (status != 0)

{

throw new Exception("Authentication failed!");

}

IntPtr dataBuffer = Marshal.AllocHGlobal(1024);

for (int i = 0; i < 4; i++)

{

int j;

byte cLen = 0;

status = rf\_M1\_read(icdev, (byte)((secnr \* 4) + i), dataBuffer, ref cLen);

if (status != 0 || cLen != 16)

{

Marshal.FreeHGlobal(dataBuffer);

throw new Exception("rf\_M1\_read failed!");

}

byte[] bytesData = new byte[16];

for (j = 0; j < bytesData.Length; j++)

bytesData[j] = Marshal.ReadByte(dataBuffer, j);

if (i == 0)

return ToHexString(bytesData);

}

throw new Exception("Unsucessful Read!");

}

private static byte GetHexBitsValue(byte ch)

{

byte sz = 0;

if (ch <= '9' && ch >= '0')

sz = (byte)(ch - 0x30);

if (ch <= 'F' && ch >= 'A')

sz = (byte)(ch - 0x37);

if (ch <= 'f' && ch >= 'a')

sz = (byte)(ch - 0x57);

return sz;

}

#region byteHEX

private static String byteHEX(Byte ib)

{

String \_str = String.Empty;

try

{

char[] Digit = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A',

'B', 'C', 'D', 'E', 'F' };

char[] ob = new char[2];

ob[0] = Digit[(ib >> 4) & 0X0F];

ob[1] = Digit[ib & 0X0F];

\_str = new String(ob);

}

catch (Exception)

{

new Exception("Unsuccessful byte to hex convertion!");

}

return \_str;

}

#endregion

private static string ToHexString(byte[] bytes)

{

String hexString = String.Empty;

for (int i = 0; i < bytes.Length; i++)

hexString += byteHEX(bytes[i]);

return hexString;

}

private static byte[] ToDigitsBytes(string theHex)

{

byte[] bytes = new byte[theHex.Length / 2 + (((theHex.Length % 2) > 0) ? 1 : 0)];

for (int i = 0; i < bytes.Length; i++)

{

char lowbits = theHex[i \* 2];

char highbits;

if ((i \* 2 + 1) < theHex.Length)

highbits = theHex[i \* 2 + 1];

else

highbits = '0';

int a = (int)GetHexBitsValue((byte)lowbits);

int b = (int)GetHexBitsValue((byte)highbits);

bytes[i] = (byte)((a << 4) + b);

}

return bytes;

}

}

Класът RFIDReader във файлът Program.cs съдържа входната точка на конзолното приложение.

public static void Main(string*[] args)*

Методът Main очаква като параметри от входния ред масив от стрингови стойности. Тъй като в C# всеки масив пази и информация за своя размер в .Length пропъртито си, размерът на масива не се оказва изрично. В този случай, се очакват 3 подредени параметъра, а именно номерът на COM порта, в който е включено устройството, скоростта на предаване (Baud) и ключ B за ауторизация на картите.

Работата на Main метода е в безкраен цикъл да опитва да инициализира четеца, като при успех, отново в безкраен цикъл на определен интервал (200 ms) да опитва да прочете информацията от карта, ако такава е налична и при успех да изпрати прочетената информация до екшън на сървъра, който да я обработи и осигури достъпа до помещенията, като запише в базата запис за посещението.

Другият метод в Program класа е

public static bool PostData(short icdev, string cardNumber, string userDetails)

Този метод се грижи да създаде модела с данните и да ги изпрати до необходимия сървис. В зависимост от получения резултат (HTTP 200 OK или HTTP 400 Bad Request) със звуков и светлинен сигнал се определя резултатът от действоето непосредствено на устройството. Вторичното действие от това извършва екшън на сървъра, който отваря / не отваря вратата на помещението.

**RFIDReader**

public class RFIDReader

{

public static string baseUri = "http://83.148.87.159/TUFitness/";

public static string lastCardRead = String.Empty;

public static DateTime lastPersonEnteredAt = DateTime.Now;

public static void Main(string[] args)

{

while (true)

{

Mifare1K reader = new Mifare1K();

int portNumber = int.Parse(args[0]);

int baud = int.Parse(args[1]);

string keyB = args[2];

short icdev = 0;

try

{

icdev = reader.InitializeRFIDReader(portNumber, baud);

reader.ChangeColor(icdev, 2);

}

catch (Exception ex)

{

continue;

}

while (true)

{

try

{

string cardNumber = reader.ReadCard(icdev);

if (cardNumber == lastCardRead)

{

if (DateTime.Now - lastPersonEnteredAt > new TimeSpan(0, 2, 0)) // 2 minutes since last authenticate of same user.

{

continue;

}

}

string userDetails = reader.ReadBlock(icdev, 'B', keyB, 0, 0);

bool successful = PostData(icdev, cardNumber, userDetails);

if (successful)

{

reader.Beep(icdev, 10);

reader.ChangeColor(icdev, 1);

lastCardRead = cardNumber;

lastPersonEnteredAt = DateTime.Now;

Thread.Sleep(2000);

reader.ChangeColor(icdev, 2);

}

else

{

reader.ChangeColor(icdev, 1);

reader.Beep(icdev, 99);

Thread.Sleep(2000);

reader.ChangeColor(icdev, 2);

}

}

catch (Exception ex)

{

Thread.Sleep(200);

continue;

}

}

}

}

public static bool PostData(short icdev, string cardNumber, string userDetails)

{

try

{

using (var client = new HttpClient())

{

client.BaseAddress = new Uri(baseUri);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

AccessControlModel model = new AccessControlModel

{

Icdev = icdev,

CardNumber = cardNumber,

UserDetails = userDetails

};

HttpResponseMessage response = client.PostAsJsonAsync("api/AccessControl/CardDetected", model).Result;

if (response.IsSuccessStatusCode)

{

return true;

}

else

{

return false;

}

}

}

catch (Exception ex)

{

return false;

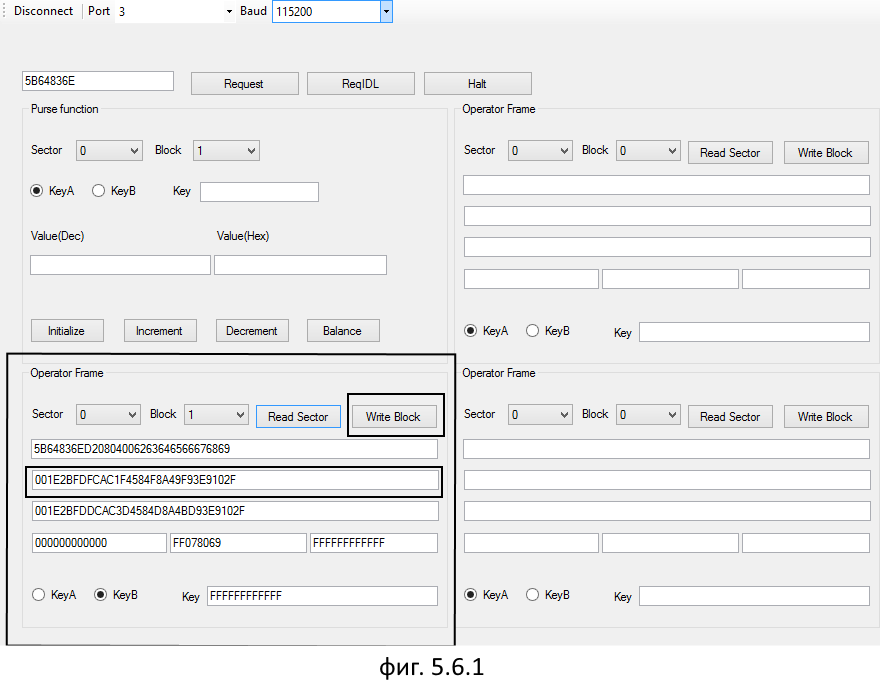
}

}

}

## 5.6. Приложение Mifare1k

Приложението Mifare1k е C# Windows Forms приложение (фиг. 5.6.1) е предоставено от фирмата производител (STRONGLINK) на използвания четец (SL500L), работещо с RFID картите Mifare1k. Основната роля, която то играе в интегрираната система е възможността, която предоставя за запис на потребителски данни в Sector 0, Block 1 на RFID картата, която трябва да бъде отъждествена с даден потребител. Попълването на този 16 битов блок със стойност, еднозначно генерирана от уникалния GUID идентификатор за всеки потребител, е всичко от което се нуждаем, за да бъде обвързана картата с потребителя. Самите карти Mifare1k имат уникален 4 байтов идентификатор, както и 2 6 байтови ключа за оторизация (по подразбиране KeyA - 000000000000 и KeyB - FFFFFFFFFFFF), комбинацията от които, както и данните, които запазваме в Сектор 0, Блок 1, еднозначно и сигурно обвързват дадена карта с даден потребител и правят фалшифицирането на неговата карта изключително трудно.

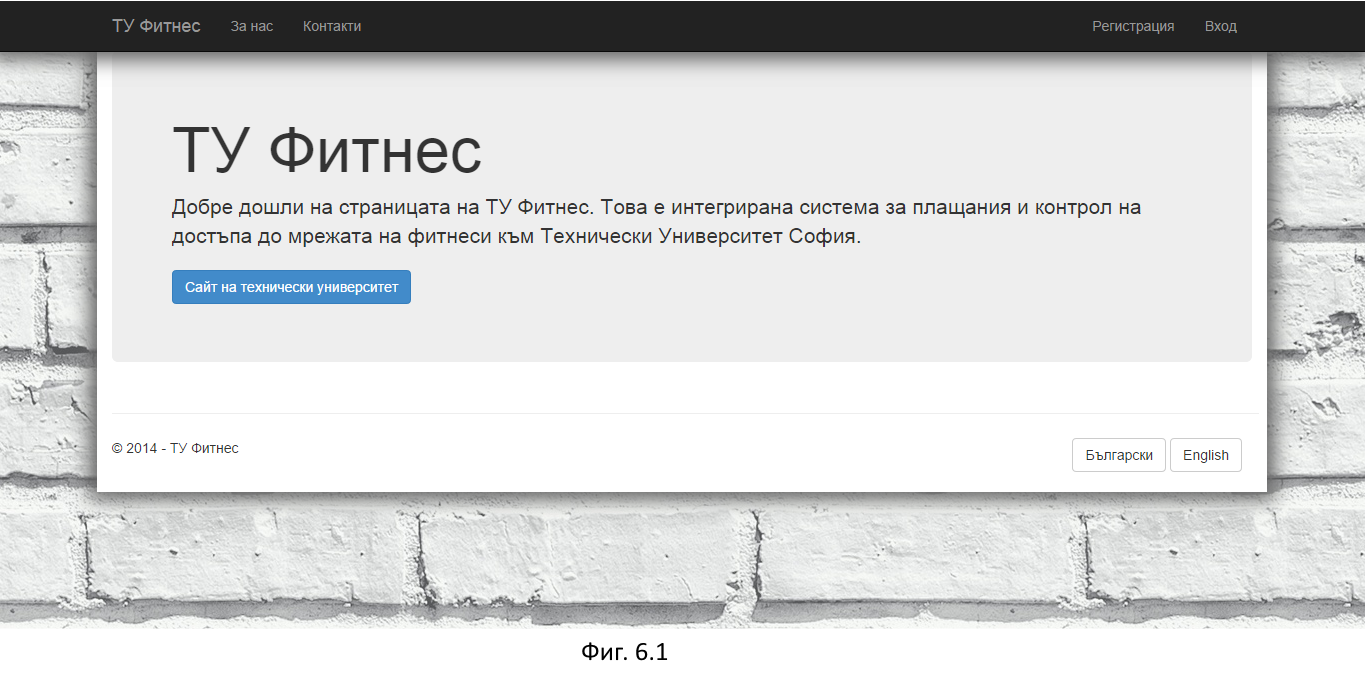


# Глава 6. Ръководство на потребителя

## 6.1. Клиент

Реализираната система е хостната на IIS7 сървър на личен компютър и може да бъде достъпена, чрез IP адреса на машината и релативния път до приложението.

http://83.148.87.159/TUFitness

****

Този URL Ви отвежда до началната страница на MVC приложението. (фиг. 6.1)

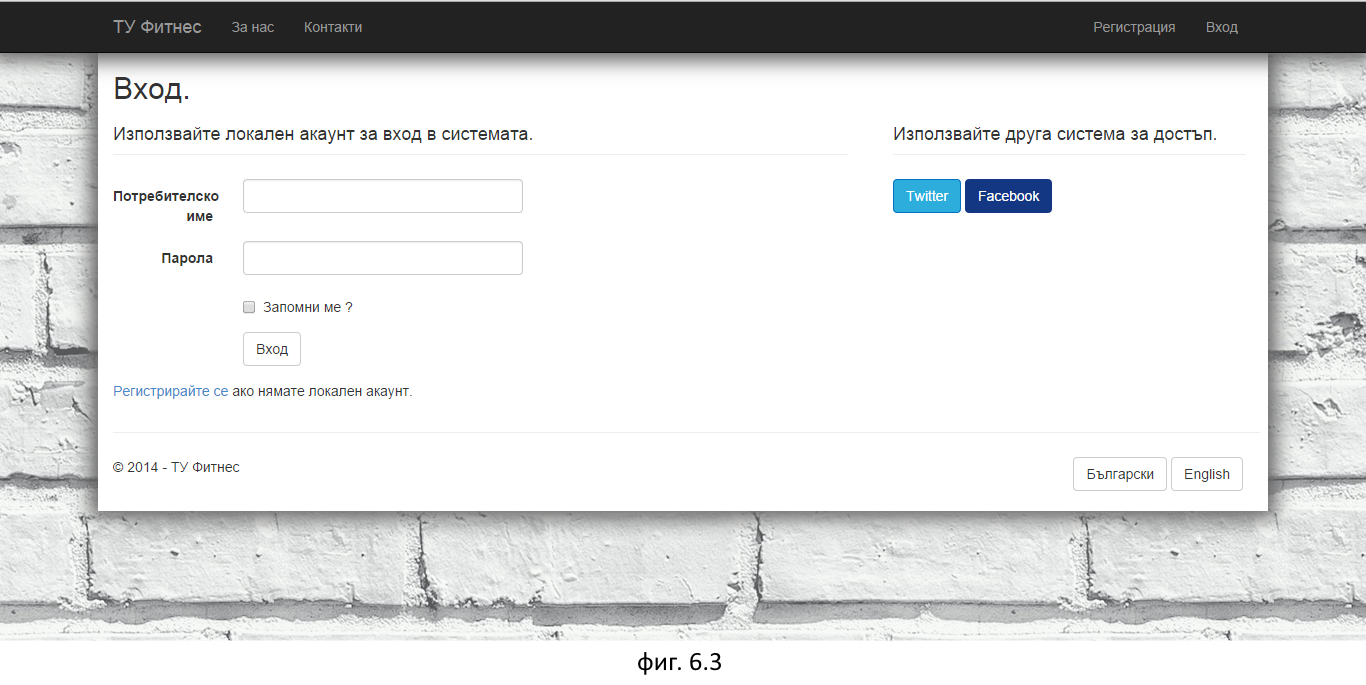
Двата бутона във Footer-a, в дъното на страницата "Български" и "English" са управляващите бутони за езика. В горната част на страницата, където е поместен Header-a се намират навигационните бутони. Първите 3 бутона са информационни и не представляват голям интерес, затова ще пристъпим към регистрация на потребител.

Натискайки, бутона "Регистрация", бивате отведени до:

http://83.148.87.159/TuFitness/Account/Register

Това е стандартна форма за регистрация с 4 полета за вход (Input elements) (фиг. 6.2). Тъй като процесът тук е линеен, при въвеждане на правилни и уникални данни за потребителско име и Email, както и достатъчно дълга парола и правилна проверка, бивате регистрирани в системата Вместо това, ще разгледаме процеса на ауторизация с 3-ти системи, натискайки бутона Вход.

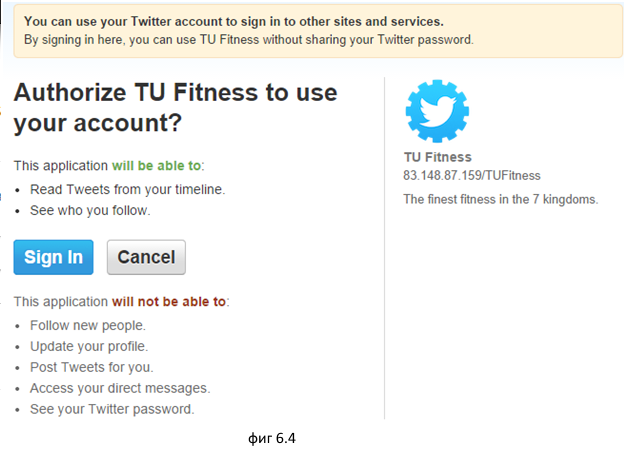


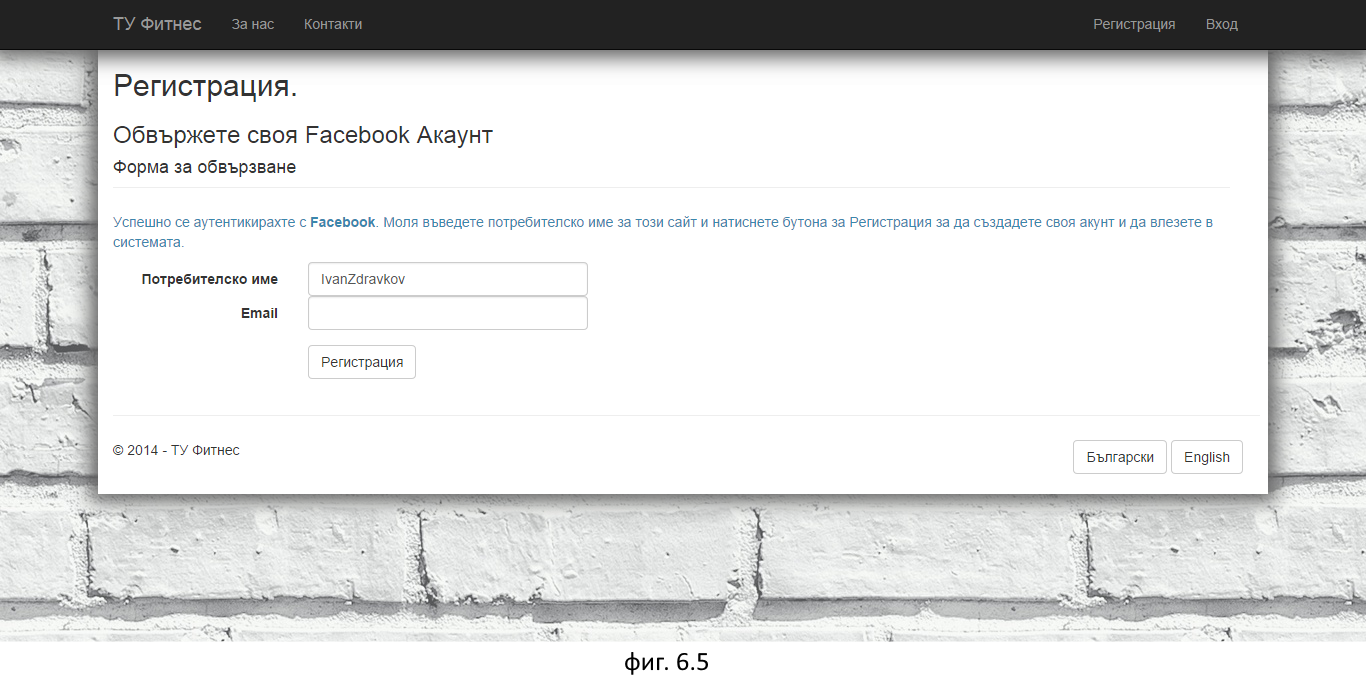
****

Натискането на бутона за вход Ви отвежда до страницата, поместваща входната форма (фиг. 6.3), намираща се на

http://83.148.87.159/TuFitness/Account/Login

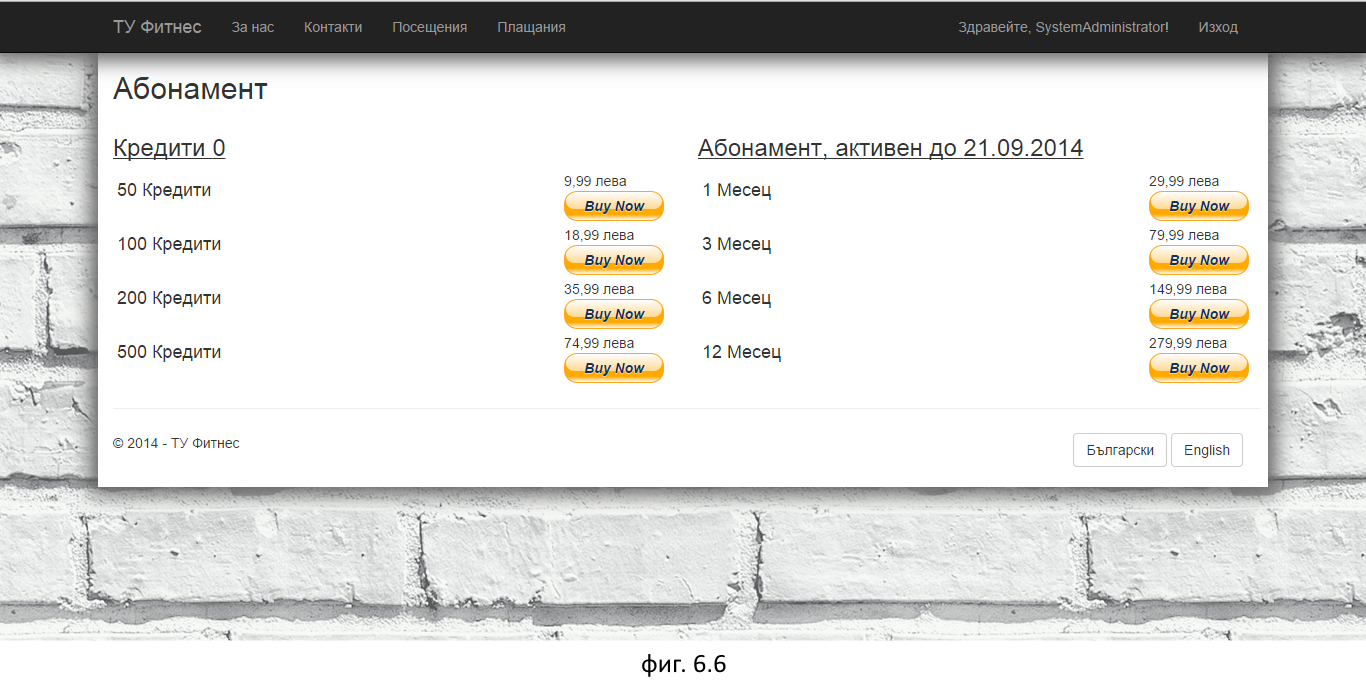
В лявата част на екрана можете да се логнете в системата, при положение че имате предварително създаден потребителски профил. В случая, ще използваме бутоните "Twitter" и "Facebook" в дясната част на страницата. Натискането на някой от тези бутони Ви отвежда до форма за потвърждение на идентичността в съответния сайт (фиг. 6.4). При успешно оторизиране на потребителя, бивате пренасочени към страницата за избор на потребителско име и парола (фиг. 6.5)

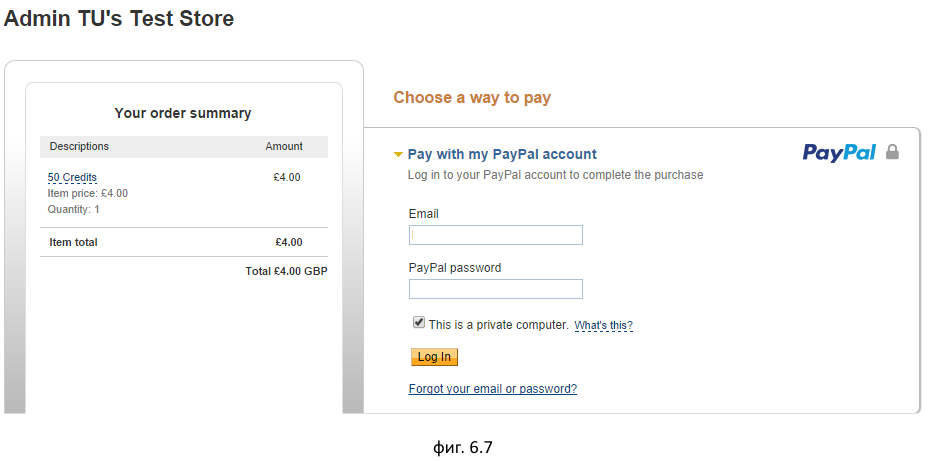


****

След като се регистрирате успешно, можете да посетите страницата за заплащане на абонаментен план (фиг. 6.6), линкът за която, както и допълнителна информация за оставащите кредити / статус на абонамента може да бъде намерен в средата на footer-a при успешно регистриран потребител.

http://83.148.87.159/TuFitness/Subscription

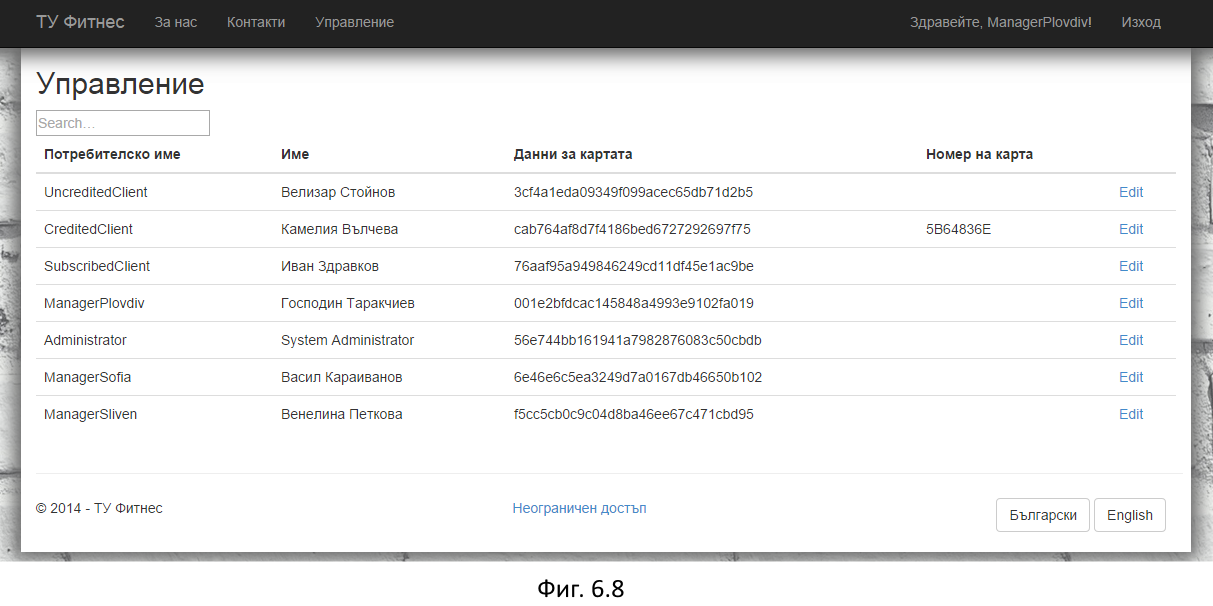
Избирайки някой от пакетите, бивате препратени към формата за Login на PayPal, където оторизирате плащането към системата. (фиг. 6.7) Цената автоматично се преобразува от BGN в GBP благодарение на актуалния курс получен чрез Yahoo! Finance API.



След като успешно извършите плащането трябва да се свържете с някой от служителите, които на място ще Ви издадат RFID карта за достъп до помещенията, ако вече нямаме такава. След като бъде издадена карта за достъп, можете свободно да я използваме във всяко от помещенията на "ТУ Фитнес" в цялата страна. При изтичане на абонаментния план или броя кредити, можете да заплатите допълнително услуги, без нужда от смяна на картата. Ако издадената карта бъде загубена, то трябва незабавно да се свържете със служител на TU Fitness, който да анулира старата ви карта и да издаде нова.

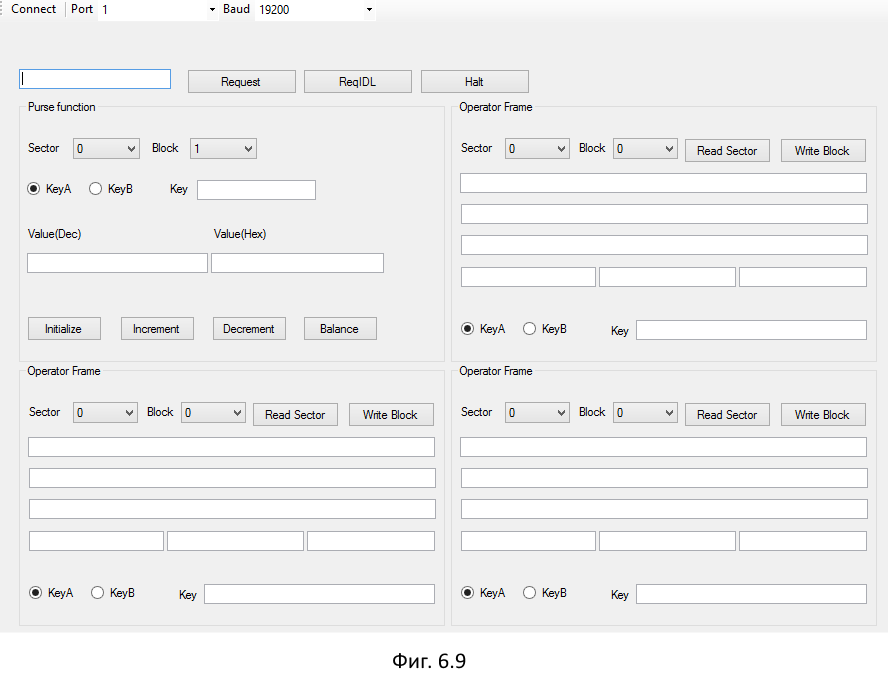
## 6.2. Персонал

След като потребителят успешно се регистира в системата и заплати своята такса е необходимо лице с роля "Administrator", "FacilityManager" или "Receptionst", което също така има достъп до специализирания софтуер за запис на карти (Mifare1k) и до RFID четеца да издаде RFID картата на потребителя.

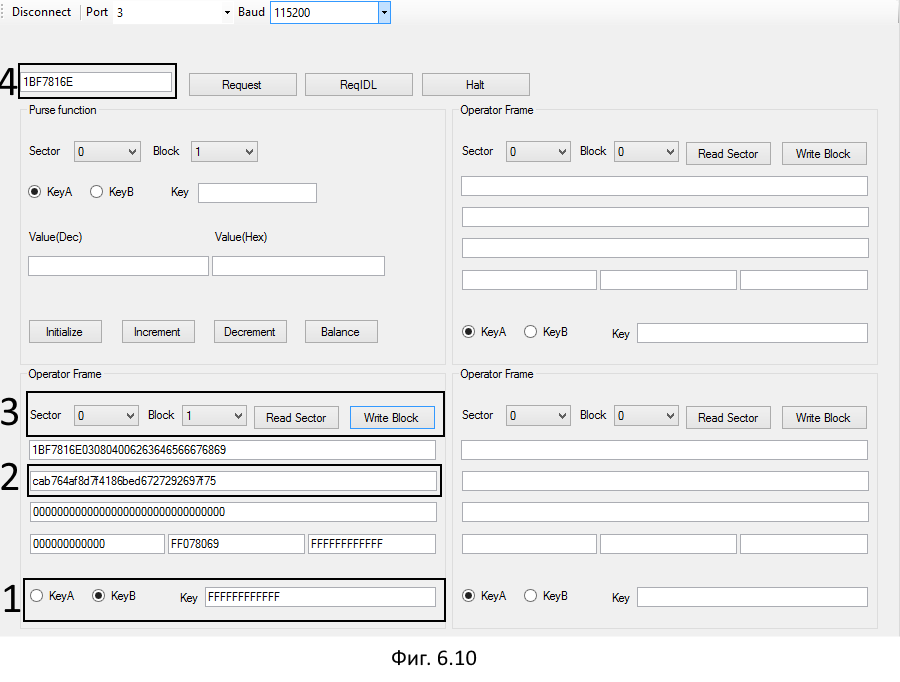


В случая се аутентикираме пред системата с потребител с роля "FacilityManager". И от навигационния хедър навигираме до страницата "Управление" (фиг. 6.8). Виждаме списък с всички потребители в системата. Благодарение на Knockout.js библиотеката имаме възможност да извършваме търсене в колекцията от потребители в модела, без да правим каквито и да е допълнителни заявки. Търсенето е по колоните "Потребителско име" и "Име". Вече виждаме, че има потребител (Камелия Вълчева), който има издадена карта. Процесът по преиздаване на вече съществуваща карта е аналогичен с този на издаване на нова карта, с единствената разлика, че при издаването на нова карта, старата се деактивира.

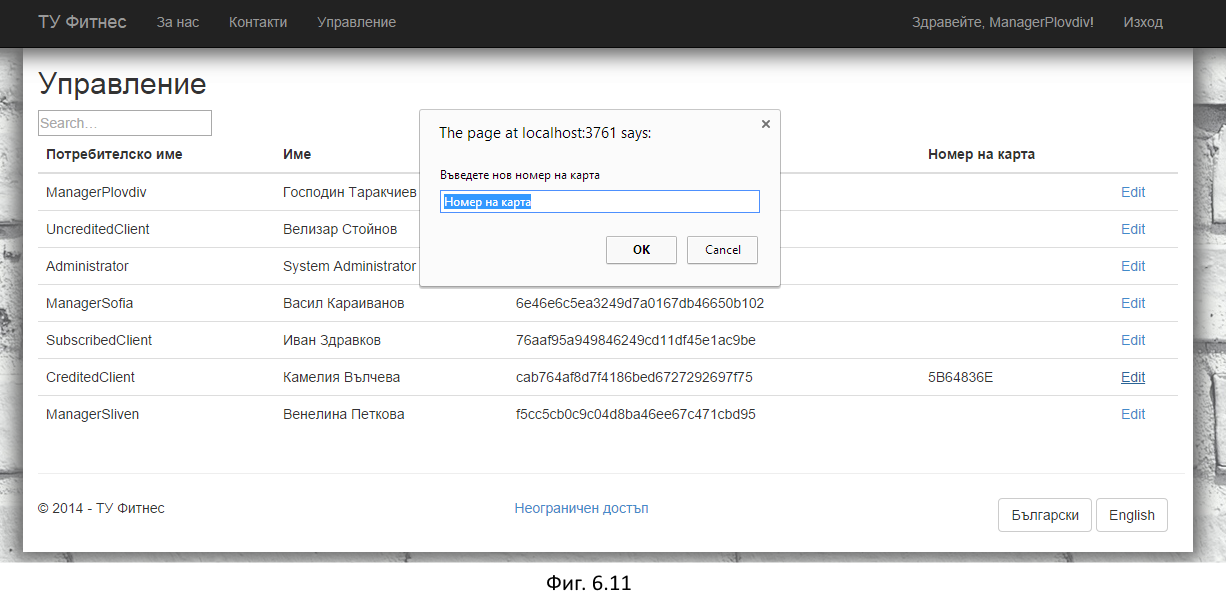
Ще пристъпим към преиздаването на картата на потребител (CreditedUser). За целта копираме данните от полето "Данни за картата". Кодът в това поле е GUID идентификационният номер на клиента, който ще запишем в определ блок на картата. След това стартираме приложението Mifare1k (фиг. 6.9).



Ако използваме същия RFID четец, като този, който се използва за четенето на карти, трябва да спрем конзолното приложение "RFIDReader", за да освободим COM порта. За да се свържем с RFID четеца избираме номера на порта, както и скоростта на предаване на данните и натискаме бутона Connect. След успешно свързване, можем да изискаме номера на поставената карта, натискайки бутона Request. След това можем да пристъпим към записване на потребителските данни на картата. (фиг. 6.10)



Първо избираме ключа, който ще използваме за ауторизация на картата (Key B) и въвеждаме стойността му (FFFFFFFFFFFF - стойност по подразбиране) в означената секция **(1).** Записваме GUID идентификационния номер на клиента, който копирахме от колоната "Данни за картата" в означеното поле **(2)**, уверяваме се, че избрания сектор е 0 а блокът, в които ще запишем стойността е 1 и натискаме бутона Write Block **(3)**. След като вече сме записали данните върху картата, копираме идентификационния номер от означеното поле **(4)** и се връщаме на страницата "Управление" на сайта, за да завършим процеса по регистрация на картата. (фиг. 6.11).



Натискайки "Edit" бутона за съответния потребител, въвеждаме новия номер на картата, който бива постнат на сървъра и запазен в базата данни, като останалите карти на потребителя се деактивират.

По този начин завършва издаването на карта на потребител. След като сме приключили с Mifare1k приложението, следва да го затворим, ако използваме един и същи RFID четец за запис и четене на карти и да стартираме RFIDReader конзолното приложение, което се грижи за четенето на картите на потребителите.

# Глава 7. Изводи и приложимост

С оглед на използваните технологии и времето за разработка, можем да заключим, че доброто познаване и използване на утвърдени рамки (frameworks) за разработка на приложения може да съкрати многократно процеса по имплементация, да повиши надеждността и сигурността, благодарение на интегрираните средства за валидация на входните данни и проверките за консистентност на данните, предстоящи за запис в базата от данни, както и на готовите класове, чиято функционалност цели да облекчи решаването на тривиалните задачи, оставяйки ни време да се фокусираме върху архитектурата и безнес логиката.

Използването на готови програмни API-та (като например това на PayPal или на Yahoo Finance), се оказват изключително надеждни и съкращават процеса на разработка драстично, предоставяйки ни готова функционалност, с достатъчно подробна и ясна документация, позволяваща ни да ги използваме, без задълбочени познания по темата и без да е необходимо да реализираме сложни алгоритми.

Ако отделим достатъчно време за обмисляне на архитектурата и подробно разучаване на развойните средства, можем да достигнем един бърз темп на разработка, като същевременно, ако в самия етап на имплементиране отделяме достатъчно време и енергия, за да спазваме конвенциите и се стремим преди всичко да продуцираме качествен код, то работата ни ще бъде продуктивна, а резултатите задоволителни.

Конкретни приложенията на системата, която е обект на разработка на текущата дипломна работа, биха могли да бъдат намерени в интегрирането и за малки и средни бизнес клиенти, нуждаещи се от модерна информационна система, предоставяща възможност на техните клиентите да извършват своите плащания по електронен път и да самоконтролират достъпа си до определени помещения, като същевременно предоставят свобода на служителите да управляват в известнва степен бизнес аспектите от системата, както и да имат достъп до пълен отчет за извършените плащания и достъпвания до помещенията, давайки на системата, необходимата гъвкавост и сигурност, които неминуемо биват изисквани за подобен вид дейност.

# Глава 8. Приложения

## 8.1. Програмен код

Текущата дипломна работа е изготвена с помощта на Atlassian Bitbucket - GIT Source Control System. Пълният проект със сорс кода може да бъде намерен на следния URL:

<https://bitbucket.org/ivanzdravkovdiplomproject>

## 

## 8.2. Използвана литература

1. <http://msdn.microsoft.com/en-us/library/ff926074.aspx>
2. <http://www.w3schools.com/aspnet/mvc_intro.asp>
3. <http://www.asp.net/mvc>
4. <http://www.asp.net/web-api>
5. <http://msdn.microsoft.com/en-us/library/bb510741.aspx>
6. <http://www.asp.net/entity-framework>
7. <http://www.beabigrockstar.com/guides/aspnet-oauth>
8. <https://developer.paypal.com/>
9. <http://www.stronglink-rfid.com/download/SL500-User-Manual.pdf>