ТЕХНИЧЕСКИ УНИВЕРСИТЕТ-СОФИЯ

ФИЛИАЛ ПЛОВДИВ

ФАКУЛЕТЕТ „ 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 4](#_Toc398589171)

[1.1. Постановка на задачата 4](#_Toc398589172)

[1.2. Архитектура 4](#_Toc398589173)

[1.3. Използвани сървърни технологии (Back End) 5](#_Toc398589174)

[1.4. Използвани клиентски технологии (Front End) 6](#_Toc398589175)

[1.5. Инструменти за разработка 9](#_Toc398589176)

[Глава 2. Проектиране на елементите на визуалния потребителски интерфейс 10](#_Toc398589177)

[2.1. Многоезичен интерфейс, чрез .resx файлове 10](#_Toc398589178)

[2.2. Проектиране на \_Layout страница 13](#_Toc398589179)

[2.3. Проектиране на индивидуални страници 13](#_Toc398589180)

[Глава 3. Проектиране на релационната база от данни 13](#_Toc398589181)

[3.1. Релационен модел 13](#_Toc398589182)

[3.2. Identity таблици 13](#_Toc398589183)

[3.3. Таблици 13](#_Toc398589184)

[3.4. Entity-relationship диаграма 13](#_Toc398589185)

[Глава 4. Проектиране и реализация на Web услуги 13](#_Toc398589186)

[4.1 Entity Framework ORM модел 13](#_Toc398589187)

[4.2 BaseController и EntityHelpers класовете 13](#_Toc398589188)

[Глава 5. Описание на програмното осигуряване 13](#_Toc398589189)

[Глава 6. Ръководство на потребителя 14](#_Toc398589190)

[Глава 7. Изводи и приложимост 14](#_Toc398589191)

[Глава 8. Приложения - програмен код и използвана литература 14](#_Toc398589192)

**Сраница на дипломния ръководител**

# Глава 1. Увод

## 1.1. Постановка на задачата

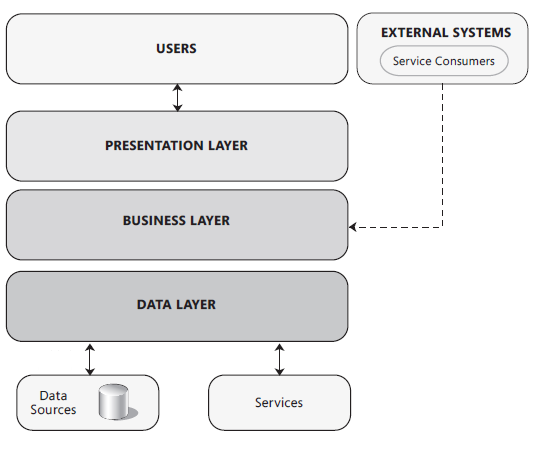
Настоящата дипломна работа има за цел да опише създаването на Web-базирана система за организиране процеса на разплащане и контролиране на достъпа на клиентите на мрежа от спортни зали. Всеки клиент, желаещ да използва системата, след създаване на собствен акаунт и издаване на RFID карта, може да извършва 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 неограниени 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. 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 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 ни 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 избраната ни многослойна архитектура.
* **ASP.NET Web Api 2.0** - дава ни 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). 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 ни осигурява още, автоматично обновяване на данните, работещо в двете посоки, както и възможност за автоматично 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 и Bootstrap.

* **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. Можем да използваме вложен CSS (inline) в самия таг на елемента, посредством атрибута **stylе.**

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

Можем също така да използваме елементът **<style>**, в който да поместим стиловете, които са ни необходими. Това е полезно, ако например искаме да опишем стилове за повече от 1 елемент или за елементи, изпълняващи някакво условие.

Нека вземем за пример следния частичен HTML

<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 желаем да смени цвета на всички <p> елементи, които се намират в <div> елемент на нашата страница, всичко което е необходимо да направим е да добавим следния <style> елемент:

<style>

div p {

color: yellow;

}

</style>

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

<style>

#paragraph-1 {

color: yellow;

}

</style>

Ако желаем да изберем всички елементи, споделящи даден клас, използваме селектора точка.

<style>

.red {

color: red;

}

</style>

Най-добрата практика при използването на CSS е отделянето на стиловете в отделен файл и зреждането 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 нашия сайт.

<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 напрактика 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 езика да се ползва обектно-ориентирано, съществуват и множество библиотеки и рамки (frameworks), които разширяват възможностите му още повече, позволявайки MVC разработки, едно-странични 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 незаменим 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 клиенткса форм валидация. Бибилиотеката 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 ни осигуряват валидация на сървъра.
* **Modernizr** - Modernizr е JavaScript библиотека, която при зареждането на страницата, проверява дали браузъра на клиента поддържа определенит 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 нашето приложение стилистично издържан външен вид, който скалира с намаляне размера на екрана, правейки сайта подходящ и за ползване от мобилни устройства.

## 

## 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 Engin, който използваме за генериране на HTML страници от даден Model, можем много лесно да имплементираме двуезичен или многоезичен интерфейс на нашето приложение. За целта създаваме проект **Resources** в нашето решение (Solution). Проекта ще ни 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 нашия **Resources** проект сме създали 2 .resx файла: Resources.resx и Resources.en.resx, съответно за Български (език по подразбиране) и Английски (език по избор). 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 (виж 4.2.), наследяваме класа Controller и заместваме (override) функционалността на метода BeginExecutionCore със следната функционалност:

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.

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 използваме за да зададем желана стойност на културата, която да използваме. Това позволява на клиента да избере езика (виж 2.2), който желае, като неговия избор ще бъде запазен, докато бисквитката, издадена от сайта ни е достъпна за ползване от клиентския браузър.

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);

}

## 2.2. Проектиране на \_Layout страница

## 2.3. Проектиране на индивидуални страници

# Глава 3. Проектиране на релационната база от данни

## 3.1. Релационен модел

## 3.2. Identity таблици

## 3.3. Таблици

## 3.4. Entity-relationship диаграма

# Глава 4. Проектиране и реализация на Web услуги

## 4.1 Entity Framework ORM модел

## 4.2 BaseController и EntityHelpers класовете

# Глава 5. Описание на програмното осигуряване

# Глава 6. Ръководство на потребителя

# Глава 7. Изводи и приложимост

# Глава 8. Приложения - програмен код и използвана литература