|  |
| --- |
| **ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**  **Факултет – „Компютърни системи и управление“** |

**КУРСОВ ПРОЕКТ**

**Дисциплина – „Програмни среди“**

***Студент: Иван Николаев Колев, фак. № 121213114, гр. 43***

***Специалност: „Компютърно и софтуерно инженерство“***

**Дата: 31.05.2016 г. Заверил:**

**/ гл. ас. д-р А. Ташева /**

**София  
2016**

Съдържание

[1. Въведение 4](#_Toc452539979)

[1. Възникване на идеята: 4](#_Toc452539980)

[2. Описание на проблема 4](#_Toc452539981)

[3. Описание на решението: 4](#_Toc452539982)

[4. Описание на предимствата: 4](#_Toc452539983)

[2. Анализ на съществуващи разработки 4](#_Toc452539984)

[2.1. Microsoft Outlook 5](#_Toc452539985)

[2.1.1 Предимства: 5](#_Toc452539986)

[2.1.2 Недостатъци: 6](#_Toc452539987)

[2.2 Yarooms 6](#_Toc452539988)

[2.2.1 Предимства: 7](#_Toc452539989)

[2.2.2 Недостатъци: 7](#_Toc452539990)

[2.3 EMS Software 7](#_Toc452539991)

[2.3.1 Предимства 9](#_Toc452539992)

[2.3.2 Недостатъци 9](#_Toc452539993)

[3. Проектиране 9](#_Toc452539994)

[3.1 Таргет група (Кой ще използва продукта?) 9](#_Toc452539995)

[3.2 Функционални изисквания 9](#_Toc452539996)

[3.2.1 Логин форма, за автентикация и оторизация на потребителите 9](#_Toc452539997)

[3.2.2 Потребителски интерфейс за намиране на зала 9](#_Toc452539998)

[3.2.3 Потребителски интерфейс за визуализиране на графика на избраната зала 9](#_Toc452539999)

[3.2.4 Възможност за създаване на събитие за дадена зала (Администратори и модератори) 9](#_Toc452540000)

[3.2.5 Възможност за промяна на данните на вече създадено събитие (Администратори и модератори) 10](#_Toc452540001)

[3.2.6 Възможност за изтриване на съществуващо събитие (Администратори и модератори) 10](#_Toc452540002)

[3.2.7 Възможност за презентационен режим 10](#_Toc452540003)

[3.3 Нефункционални изисквания (сървър, data storage, клиент) 10](#_Toc452540004)

[3.3.1 Изисквания към сървърната част 10](#_Toc452540005)

[3.3.2 Изисквания към частта за съхранение на данни 10](#_Toc452540006)

[3.3.3 Изисквания към клиентската част 10](#_Toc452540007)

[3.4 Архитектура на системата 11](#_Toc452540008)

[3.4.1 Диаграма на архитектурата 11](#_Toc452540009)

[3.4.2 Описание на архитектурата 11](#_Toc452540010)

[3.5 Физическа схема на базата от данни 12](#_Toc452540011)

[3.6 Примерен потребителски интерфейс 13](#_Toc452540012)

[4. Реализация 15](#_Toc452540013)

[4.1 Създаване на базата от данни по подхода Code-first 15](#_Toc452540014)

[4.2 Захранване на базата с примерни данни за тестови цели 18](#_Toc452540015)

[4.3 Autocomplete за Desktop клиента 18](#_Toc452540016)

[4.4 Сигурност на данните и криптиране 19](#_Toc452540017)

[4.5 Алгоритъм за откриване на застъпвания при създаване и промяна на събития 20](#_Toc452540018)

[5. Потребителско ръководство 23](#_Toc452540019)

[5.1 Вход в системата 23](#_Toc452540020)

[5.2 Главен панел 24](#_Toc452540021)

[5.3 Форма за съдаване на събитие 24](#_Toc452540022)

[5.4 Презентационен режим 24](#_Toc452540023)

[6. Заключение 24](#_Toc452540024)

[7. Литература 24](#_Toc452540025)

[7.1 FeserWard Intellibox 24](#_Toc452540026)

[7.2 Microsoft Developers Network 24](#_Toc452540027)

[7.3 DrawIO 24](#_Toc452540028)

[8. Приложение 24](#_Toc452540029)

[8.1 Halls controller 25](#_Toc452540030)

[8.2 Events controller 26](#_Toc452540031)

[8.3 Identity controller 28](#_Toc452540032)

[8.4 Base controller 29](#_Toc452540033)

[9. Source code 29](#_Toc452540034)

# 1. Въведение

Текущият проблем който разглеждаме е свързан със запазването на зали за провеждане на учебни занятия (лекции, упражнения, лабораторни) и изготвянето на дигитален график удобен за ползване от студенти и преподаватели.

## 1. Възникване на идеята:

Идеята е плод на ситуации, при които преподавателите не се уведомени, че графика на залата в която трябва да водят лекции или упражнения е променен и там се обучава друга група, или когато преподавателя лично е сменил залата, но не всички студенти са уведомени – тогава настава хаос, неразбирателство и закъснения.

## 2. Описание на проблема

Сегашният процес по промяна на разписанието на залите, в случая в който преподавател променя залата - разчита на предаване на информацията от уста на уста (преподавател информира един или няколко студенти, да разпространят на другите си колеги, че залата и часът се променя) или в най-добрия случай – изпращане на email до някоя от групите засегнати от промяната. А в случая, когато друг преподавател изиска зала и промени графика – първият, който е трябвало да ползва залата, трябва да отиде до кабинета на охраната, да прегледа хартиените списъци с предадените ключове и да прецени коя зала е свободна в момента, за да я заеме, но това отключва възможността за възпрепиятстване на занятията на друг преподавател по веригата, понеже няма как всеки да знае часовете за групи на всички останали и това е много сериозен организационен проблем.

## 3. Описание на решението:

Решението което предлагам е интегриране на централизирана уеб услуга, която да бъде консумирана от мобилни, настолни или уеб приложения (Mobile, Desktop, Web clients). Идеята е всеки да може да достъпва графика през смартфона си, през уеб приложение или през настолен компютър, за да следи най-актуалните промени по графиците на всяка от залите, а преподавателите (които ще бъдат администратори в тази система) да ползват лесен интерфейс за промяна или добавяне на нови събития към графиците на залите.

## 4. Описание на предимствата:

По този начин, за по-малко от 20 секунди един преподавател може да актуализира графика на залата за конкретен ден, като успоредно с това има достъп до вече запазените часове, като всичката информация е събрана на едно място и се избягва нуждата от търсене във всичките N – на брой pdf файла, които в момента се ползват за седмичен разпис на часовете на отделните групи от всички факултети. А студентите, могат лесно да проверяват актуализирания график директно от мобилните си устройства или от персоналните си компютри. Така, вината за това че някой не е в час с новата програма, пада изцяло върху него.

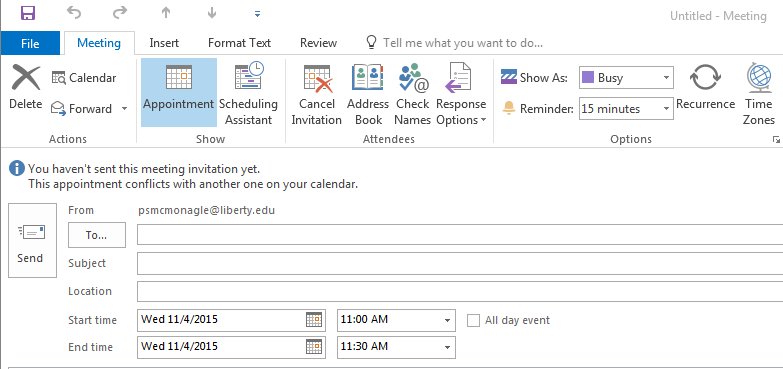
Тази система позволява по-удобно запазване на часове и извън седмичния разпис, предоставен от факултетната администрация. Например при нужда за отработване на дадено упражнение, могат да се запазят зали за някой от почивните дни, без да се случват застъпвания с други групи, инцидентно отработващи занятия. Или просто запазването на зали за събития, които не са в пряка връзка с обучителния процес – външни лектори, вътрешни събития само за академичния състав, събрания, форуми (Career fair).

# 2. Анализ на съществуващи разработки

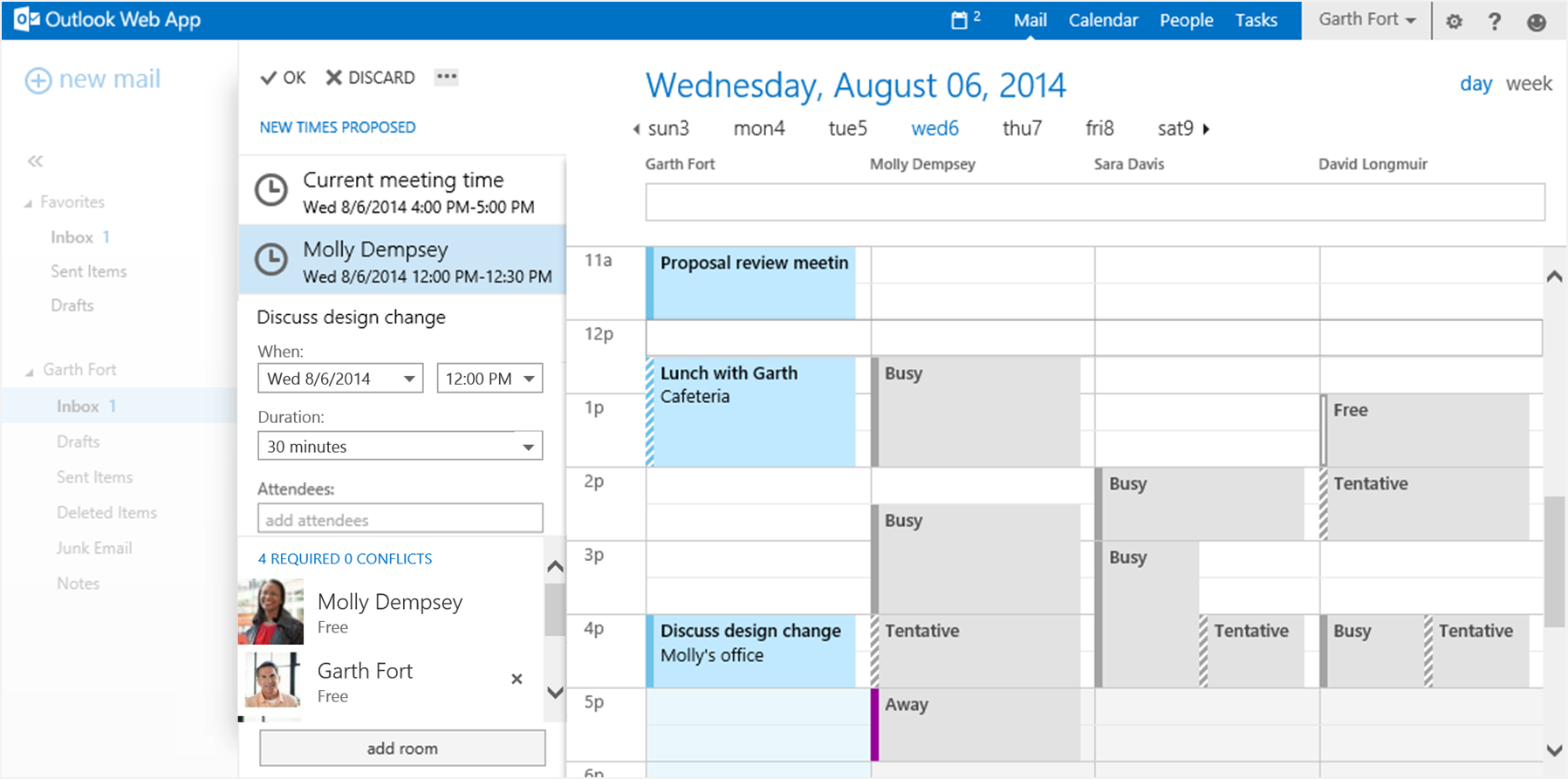
## 2.1. Microsoft Outlook

Enterprise решение на Майкрософт с възможност за интеграция с други MS продукти, което включва огромен пакет функционалности, една от които е **Scheduler** и **Meetings assistant**, които дават възможности за запазване на зали, които предварително се вкарват ръчно в базата от данни на приложението, и график, през който може да се разглежда заетостта на залите. Като при опит за записване на събитие, което се припокрива с друго такова – MS Outlook връща съобщение, че заявката е отказана и събитието не може да бъде включено в графика, както и при успешно записване – това се визуализира на клиента.

*Пример:*



**Фигура 1. Записване на ново събитие в зала.**



**Фигура 2. Преглеждане график на зали.**

### 2.1.1 Предимства:

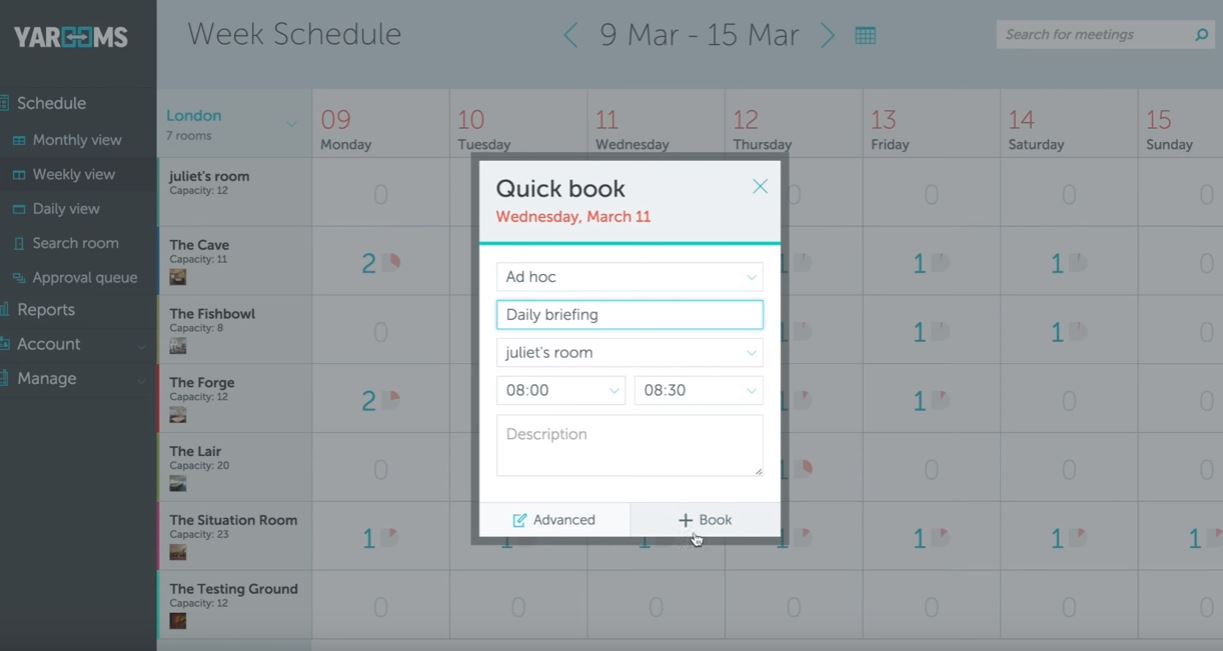
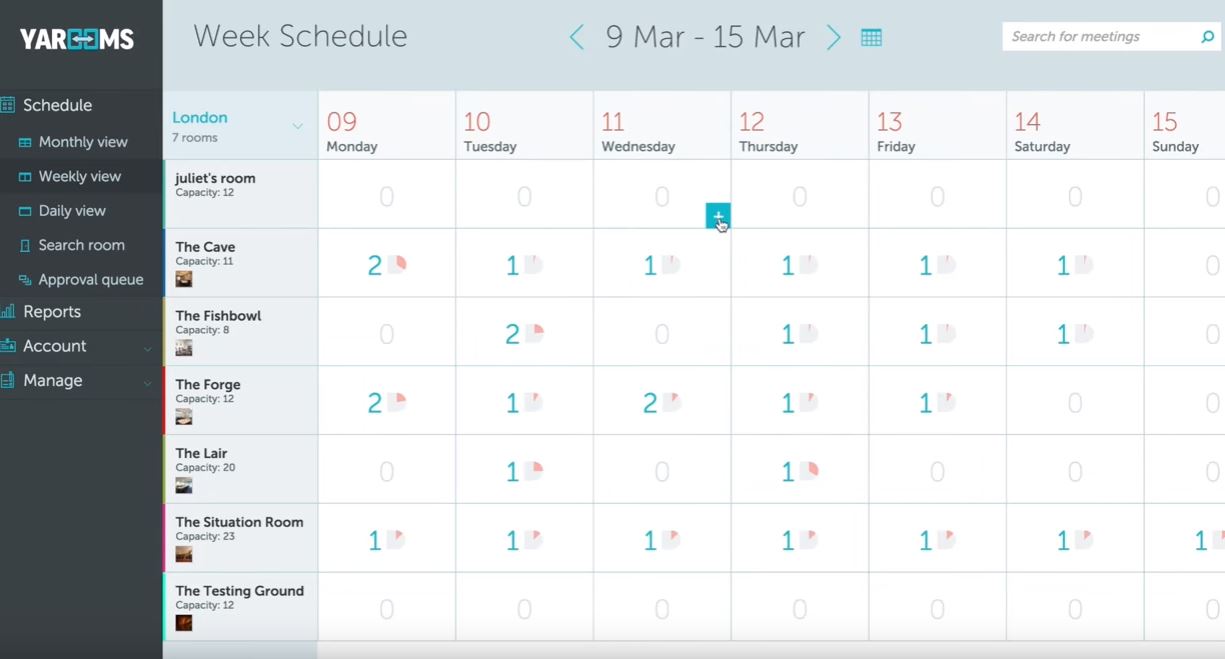
Поддържа пълен пакет функционалности нужни за запазване и преглеждане на зали с допълнителни възможности за интеграция с други продукти на Майкрософт.

### 2.1.2 Недостатъци:

Платен (ценовите пакети започват от $109 щатски долара – нагоре), носи със себе си огромен пакет от излишни за решаването на конкретния проблем функционалности, потребителският интерфейс става претрупан и труден за ползване, при наличието на множество записи.

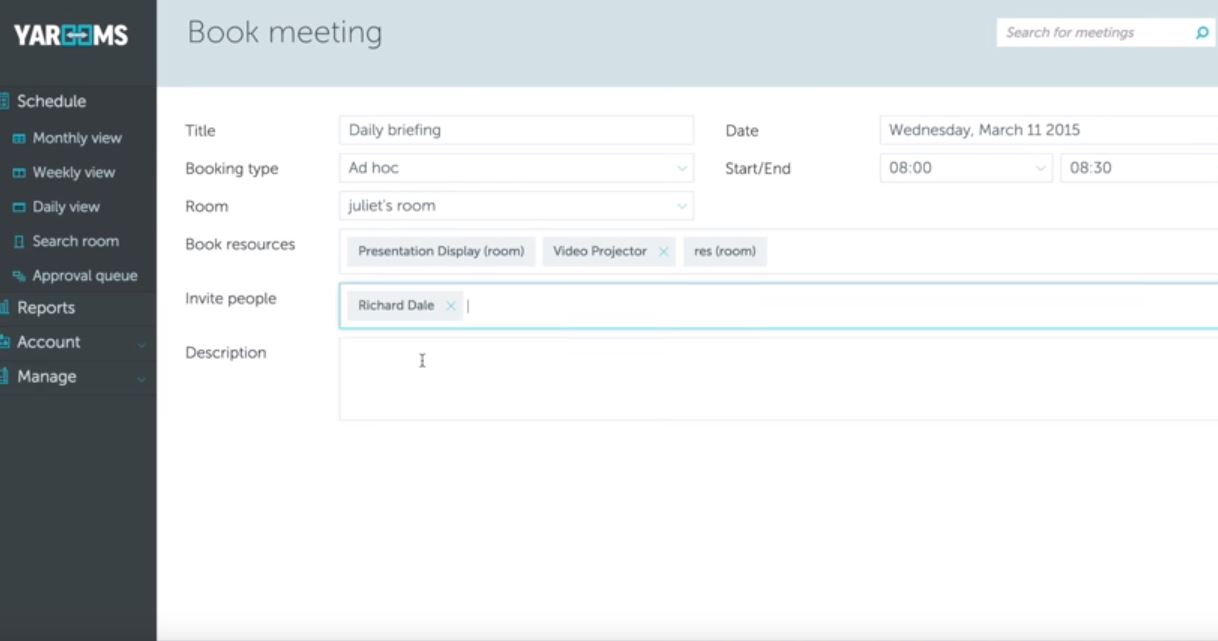
## 2.2 Yarooms

Medium scale – Enterprise решение, което е специално изработено за запазване на зали за срещи и събития.

*Пример:*

Фигура 3. Изглед на графика на стаите.

Фигура 4. Изглед на функционалността за запазване на стая. (Бързо запазване)



Фигура 5. Изглед на функционалността за запазване на стая (Детайлен изглед)

### 2.2.1 Предимства:

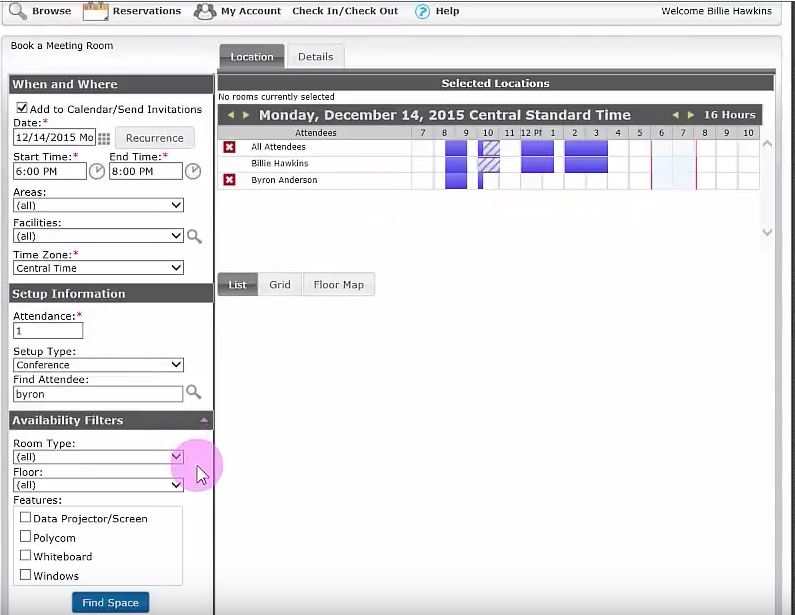
Изключително удобен потребителски интерфейс, с много полезни функционалности като: покана на хора, за участие в събитието и добавяне на изискване за ресурси, необходими за провеждане на събитието. Има уеб и мобилен клиент, което улеснява достъпа на хората от всякакви устройства с включен интернет. Предоставя интеграция с MS Outlook.

### 2.2.2 Недостатъци:

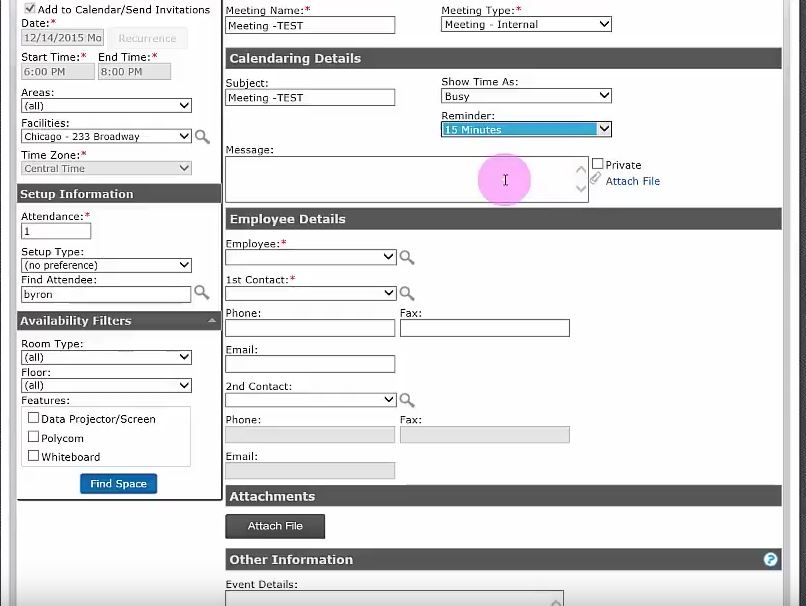
Решението отново е платено, като пакетите започват от $0.3 щатски долара на заявка, като минимума е $5 щатски долара на месец, за потребител. Което означава че ако студентите в университета са близо 5000 + близо 300 преподавателя, това прави $265 000 щатски долара на учебна година, за приложение за запазване на зали.

## 2.3 EMS Software

Enterprise насочено решение, чиито фокус е повече върху големите корпорации. Носи със себе си голям пакет от услуги, като повечето са свързани със заемането на зали и преглеждане на график за събитията.

Пример:

Фигура 6. Потребителски изглед за преглеждане на график на зала.



Фигура 7. Потребителски изглед за записване на събитие.

### 2.3.1 Предимства

Начина за записване на събития е доста абстрактен, което позволява приложение за различни сфери и различни конфигурации на зали. Всяко събитие има възможност за детайлно описание и допълнителни функционалности като покана на хора, които да присъстват и т.н.

### 2.3.2 Недостатъци

Решението е платено, като ценовите пакети започват от $59 щатски долара на година за потребител. Потребителският интерфейс изглежда архаичен, неподреден и сложен за употреба. Решението е повече насочено към корпоративни организации и съответно фокуса на функционалностите е изместен в тази посока.

# 3. Проектиране

Описание на всички аспекти от крайния продукт

## 3.1 Таргет група (Кой ще използва продукта?)

Фокусът на продукта е насочен към управлението на залите за Технически университет – София, съответно е предназначен за студентите и хора от академичния състав, които ползват материалната база на университета.

## 3.2 Функционални изисквания

### 3.2.1 Логин форма, за автентикация и оторизация на потребителите

Тази функционалност е необходима, за да има начин за контролиране на достъпа до приложението и други негови функционалности, като например – само модератори и администратори могат да запазват и променят графика на учебните зали. Потребителските акаунти са необходими също, за да може даден студент да маркира съответно събитие като любимо, и да получава съобщения, ако има промяна на часа, темата или лектора.

### 3.2.2 Потребителски интерфейс за намиране на зала

Тази функционалност трябва да бъде имплементирана с текстово поле за автоматично показване на подсказки на потребителя (autocomplete), което да улеснява въвеждането на зали във формата за търсене.

### 3.2.3 Потребителски интерфейс за визуализиране на графика на избраната зала

Тук потребителят трябва да вижда Tab menu със списък с всички дни от седмицата, като при избор на ден, да се визуализира списък със събитията записани за избраната зала за конкретния посочен ден, като събитията са подредени по начален час и за всяко събитие е показан начален-краен час, тема и лектор.

### 3.2.4 Възможност за създаване на събитие за дадена зала (Администратори и модератори)

След като потребителят е избрал зала, трябва да вижда бутон (Schedule event), като при натискането му трябва да излиза прозорец със данните необходими за запазване на събитие (Лектор, Тема, Начален и краен час, Ден от седмицата). При успешно добавяне, трябва да се покаже съобщение на потребителя, че заявката е отразена, а при неуспешно добавяне, трябва да се покаже причината поради която не е успешна заявката (Застъпване с друго събитие, грешка на сървъра и т.н). Обикновените потребители (студенти) нямат достъп до тази функционалност.

### 3.2.5 Възможност за промяна на данните на вече създадено събитие (Администратори и модератори)

При двоен клик на събитие от списъка със събития за дадена зала, трябва да излиза меню с данните, които могат да се коригират (Лектор, Тема, Начален и краен час, Ден от седмицата). Като отново при успешна промяна, трябва да се визуализира подходящо съобщение на потребителя, както и при грешка (Застъпване с друго събитие или грешка на сървъра). Преподавателите, трябва да могат да променят само собствените си събития. Обикновените потребители (студенти) нямат достъп до тази функционалност.

### 3.2.6 Възможност за изтриване на съществуващо събитие (Администратори и модератори)

Преподавателите трябва да могат да изтриват само собствените си събития, а администраторите трябва да могат да премахват всякакви събития. Обикновените потребители (студенти) нямат достъп до тази функционалност.

### 3.2.7 Възможност за презентационен режим

Тази функционалност трябва да предоставя режим на работа на приложението, който да включва елегантен background, информация за текущото събитие, ако има такова (час, тема и лектор) както и информация за събитието което следва (час, тема и лектор). Допълнително изискване – background-а на прозореца да се сменя през определен интервал от време.

## 3.3 Нефункционални изисквания (сървър, data storage, клиент)

### 3.3.1 Изисквания към сървърната част

Платформа за работа на системата - .NET Framework v4.0+. Обектно-ориентиран език за програмиране – C#.

Сървърната част трябва да бъде реализирана с технологията ASP.NET Web API 2.0. Тази технология предоставя възможност за силна модулация на елементите, които конструират функционалностите. Позволява лесна манипулация и обработка на различни формати за трансфер на данни (XML, JSON), а това е ключов елемент при създаването на RESTful услуги.

Сървърът трябва да може да обслужва повече от 1000 клиента едновременно, без осезаемо забавяне. (Ако се ползва новата версия на ASP.NET Core 1.0 в комбинация с новият им сървър Kestrel, може да се достигне обработка на брой заявки между 500 000 - 1 300 000 в секунда (в зависимост от тежестта на заявките).

### 3.3.2 Изисквания към частта за съхранение на данни

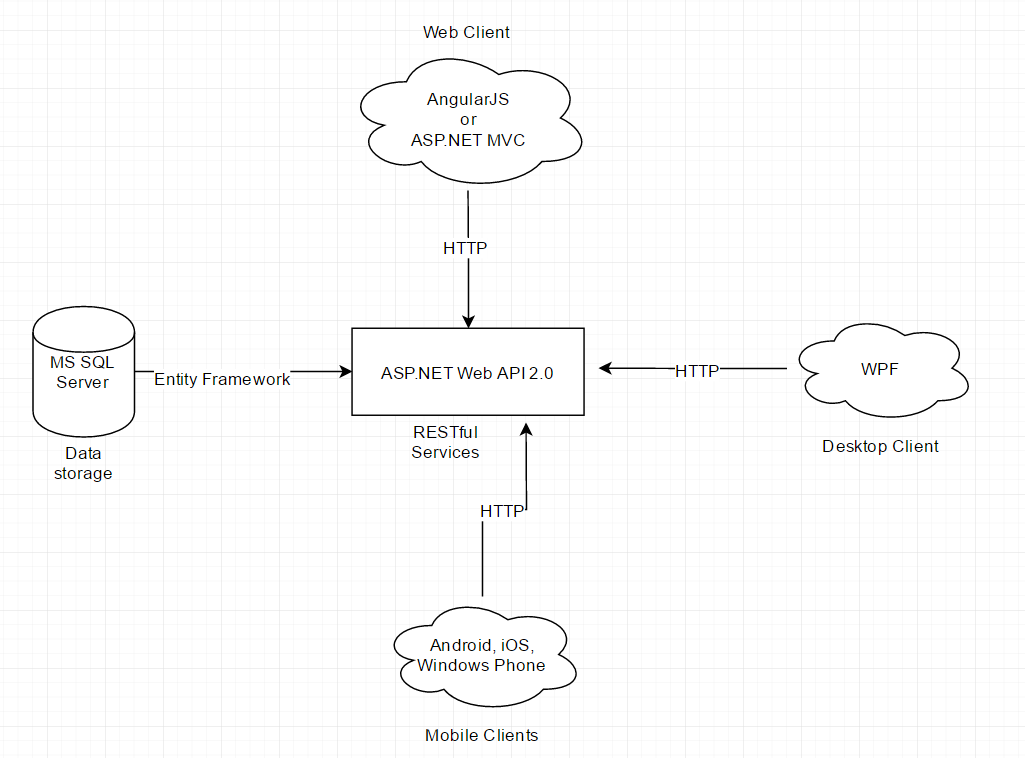
За съхранение на всички данни, които се ползват от системата, трябва да се ползва релационна база от данни - MS SQL Server 2012+.

### 3.3.3 Изисквания към клиентската част

Клиентски приложения трябва да има реализирани за Desktop, Mobile & Web, за да е максимално удобно за всеки ползването на системата. Изискване за Desktop приложението е да бъде реализирано на технологията WPF, изискване за мобилните приложения е да бъдат реализирани за трите най-модерни операционни системи – Android, iOS, Windows phone, а изискване за Уеб приложението е да бъде имплементирано на ASP.NET MVC или като Single Page Application чрез AngularJS.

## 3.4 Архитектура на системата

### 3.4.1 Диаграма на архитектурата



### 3.4.2 Описание на архитектурата

В ядрото на системата се намира **RESTful** уеб услуга, реализирана на **ASP.NET Web API**, която представлява бизнес логиката на приложението. Тя е отправната точка към която се пращат заявки, които биват обработени и се връща резултат на потребителя под формата на данни в JSON формат, които трябва да бъдат обработени на клиента, а резултатът - визуализиран на потребителя. Клиентите биват Web, Desktop & Mobile базирани, като те консумират тази уеб услуга, чрез изпращане на HTTP заявки, към съответните адреси.

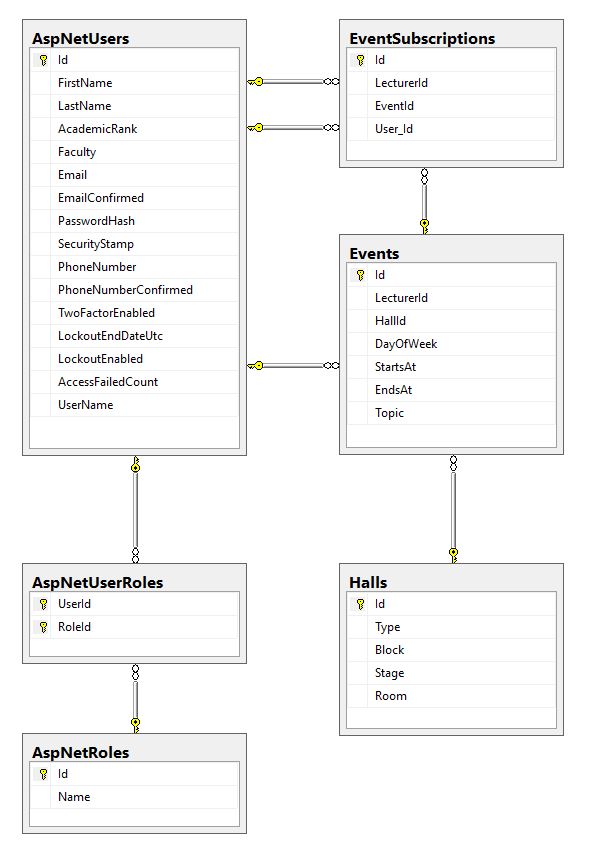
За съхранение на данни системата използва релационна база от данни Microsoft SQL Server 2014, като връзката между уеб сървъра (Web API) и базата от данни (MS SQL) се реализира чрез Entity Framework (ORM – Object-relational-mapping), който представлява технология, която се грижи за връзката между таблици на базата от данни и класовете в обектно-ориентирания език за програмиране C#. Entity framework позволява лесно и удобно манипулиране на данни директно през C# кода, като разработчиците могат да се възползват от LINQ-TO-SQL за декларативен стил на програмиране.

Desktop клиентът трябва да бъде имплементиран чрез технологията WPF и да има всички потребителски интерфейси и функционалности описани по-горе във функционалните изисквания. Като той консумира уеб услугите предоставени от Web API.

Mobile клиентите трябва да бъдат имплементирани за Android, iOS & Windows Phone, за да бъде приложението максимално достъпно за хора ползващи различни устройства.

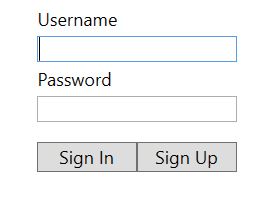
Web клиентът може да бъде реализиран като Single Page Application, чрез AngularJS или като стандартно уеб приложение с ASP.NET MVC, като и двата варианта имат предимства и недостатъци и са еднакво приемливи.

## 3.5 Физическа схема на базата от данни

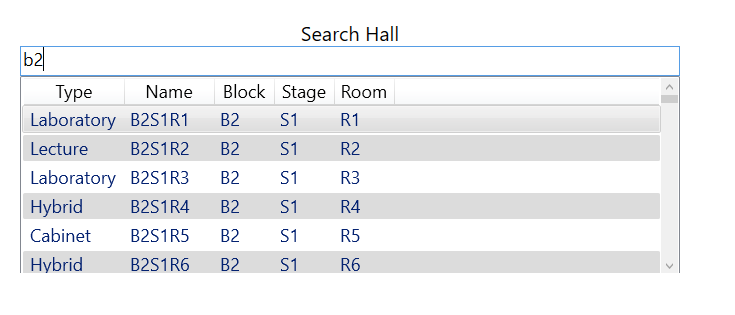


Фигура 8. Таблици необходими за реализиране на желаните функционалности

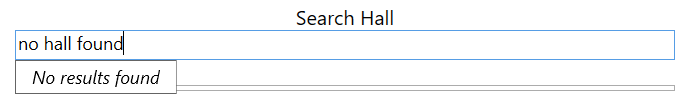
## 3.6 Примерен потребителски интерфейс



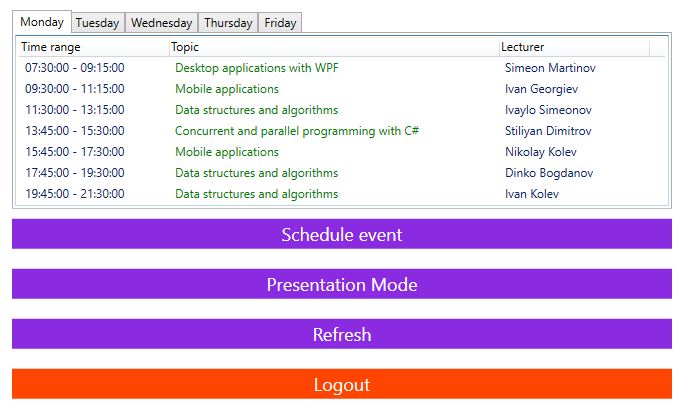
Фигура 9. Форма за вход в системата (паролите се пазят в структура от данни тип SecureString, което представлява криптиран масив от байтове, за предпазване от атаки насочени към рам паметта)



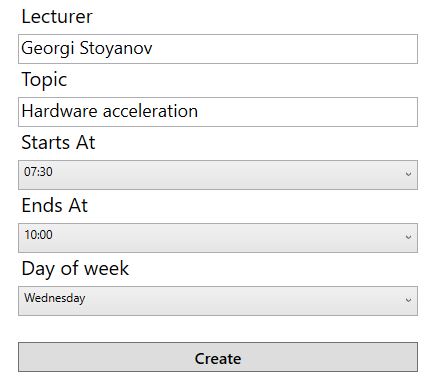
Фигура 10. Текстово поле за търсене на зали 1.



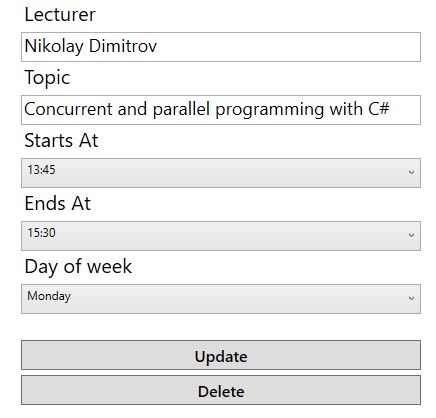
Фигура 11. Текстово поле за търсене на зали 2.



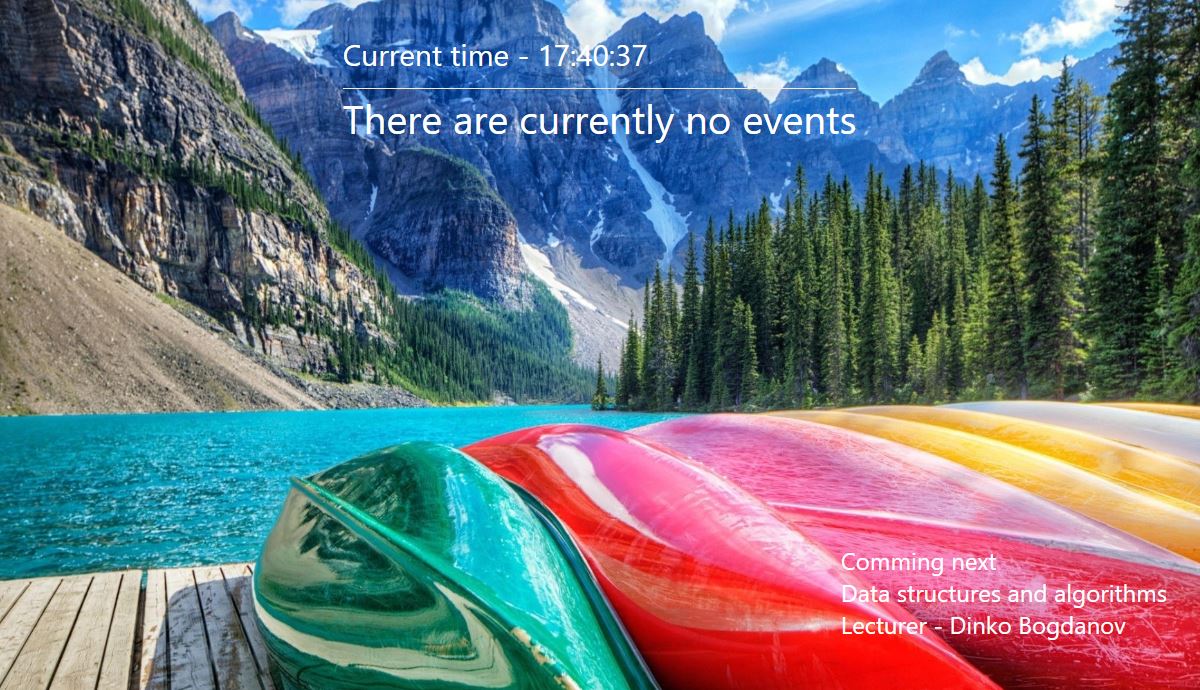
Фигура 11. График на избраната зала по дни.



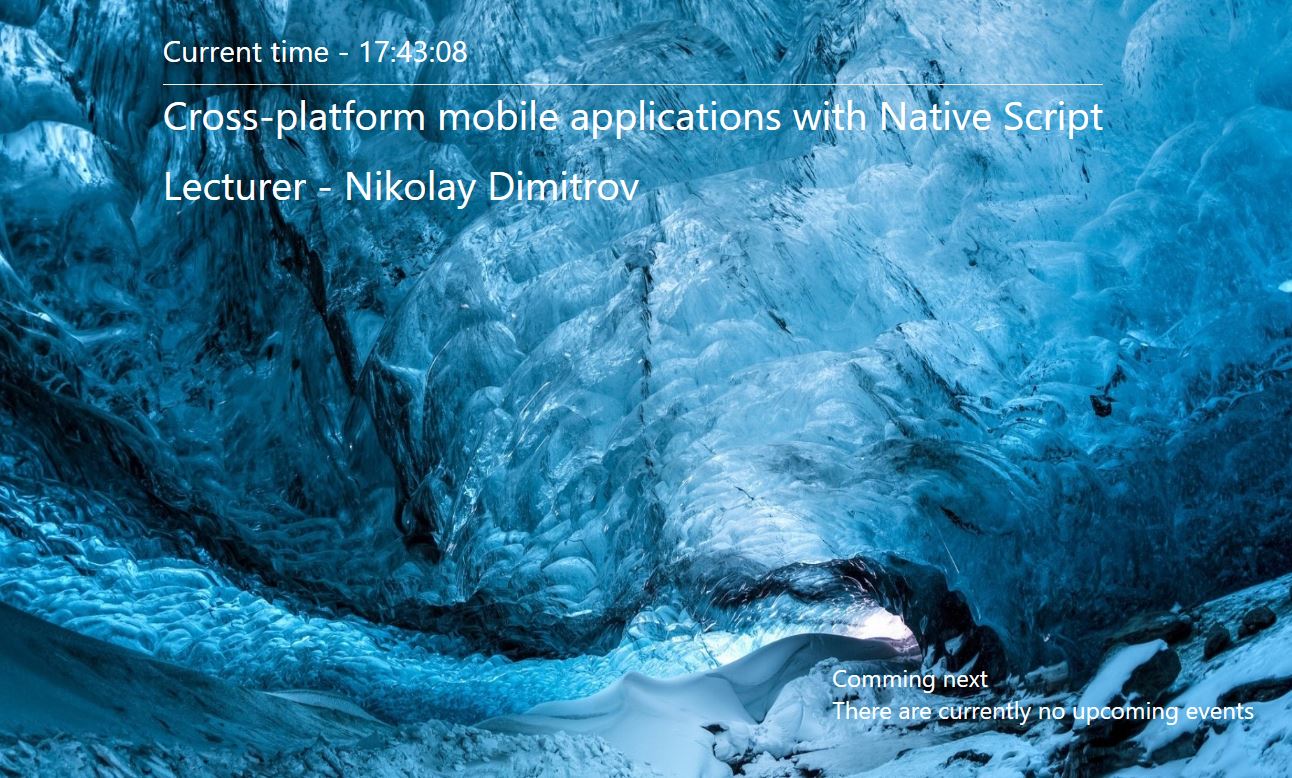
Фигура 12. Форма за създаване на събитие



Фигура 13. Форма за промяна или изтриване на вече съществуващо събитие.



Фигура 14. Презентационен режим на зала в която в момента не се провеждат събития.



Фигура 15. Презентационен режим на зала в която в момента тече събитие.

# 4. Реализация

## 4.1 Създаване на базата от данни по подхода Code-first

Използвам code-first, подхода за създаване на database схемата, понеже ми позволява да работя почти изцяло само в C# кода, таблиците в базата реализирани през C# обекти изглеждат по следния начин:

namespace HallScheduler.Data.Models

{

using System;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using Common.Constants;

using Common.Contracts;

public class Event : IEvent

{

[Key]

public int Id { get; set; }

[ForeignKey(NavigationPropertiesConstants.LecturerId)]

public virtual User Lecturer { get; set; }

[Required]

public string LecturerId { get; set; }

[ForeignKey(NavigationPropertiesConstants.HallId)]

public virtual Hall Hall { get; set; }

[Required]

public int HallId { get; set; }

[Required]

[EnumDataType(

typeof(DayOfWeek),

ErrorMessage = ValidationConstants.EventDayOfWeekErrorMessage)]

public DayOfWeek DayOfWeek { get; set; }

[Required]

public TimeSpan StartsAt { get; set; }

[Required]

public TimeSpan EndsAt { get; set; }

[Required]

[MinLength(ValidationConstants.EventTopicMinLength, ErrorMessage = ValidationConstants.EventTopicMinLengthErrorMessage)]

[MaxLength(ValidationConstants.EventTopicMaxLength, ErrorMessage = ValidationConstants.EventTopicMaxLengthErrorMessage)]

public string Topic { get; set; }

}

}

namespace HallScheduler.Data.Models

{

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Common.Constants;

using Common.Enumerations;

public class Hall

{

[Key]

public int Id { get; set; }

[Required]

[EnumDataType(

typeof(HallType),

ErrorMessage = ValidationConstants.HallTypeErrorMessage)]

public HallType Type { get; set; }

[Required]

[EnumDataType(

typeof(BlockType),

ErrorMessage = ValidationConstants.HallBlockTypeErrorMessage)]

public BlockType Block { get; set; }

[Required]

[EnumDataType(

typeof(StageType),

ErrorMessage = ValidationConstants.HallStageTypeErrorMessage)]

public StageType Stage { get; set; }

[Required]

[EnumDataType(

typeof(RoomType),

ErrorMessage = ValidationConstants.HallRoomValueErrorMessage)]

public RoomType Room { get; set; }

public virtual ICollection<Event> Schedule { get; set; }

}

}

Атрибутите над всяко пропърти, се използват за налагане на ограничения за валидация на данните, които постъпват в съответните колони в базата, те определят минимални и максимални стойности, които биват допускани за запис, дали са задължителни или не и т.н. Използват се също и навигационни атрибути, които помагат на Entity Framework да реализира връзките между таблиците – например ForeignKey(“PropertyName”).

През model builder-а на Entity Framework, можем през кода да зададем допълнителни ограничения за нашите таблици в базата от данни. Например можем да кажем при изтриване на ред от дадена таблица, дали SQL Server-ът да изпълни каскадно изтриване на всички реферирани редове или не:

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<EventSubscription>()

.HasRequired(x => x.Lecturer)

.WithMany()

.WillCascadeOnDelete(false);

modelBuilder.Entity<EventSubscription>()

.HasRequired(x => x.Event)

.WithMany()

.WillCascadeOnDelete(false);

modelBuilder.Entity<Hall>()

.HasRequired(x => x.Schedule)

.WithMany()

.WillCascadeOnDelete(true);

modelBuilder.Entity<Event>()

.HasRequired(x => x.Hall)

.WithMany()

.WillCascadeOnDelete(false);

}

## 4.2 Захранване на базата с примерни данни за тестови цели

За пълнене на базата с тестови данни, ползвам Configuration класа който идва при инсталиране на пакет за миграции за Entity Framework. Той предоставя Seed метод, който се извиква всеки път при стартиране на приложението. Използвам собствен авторски шаблон за сийдване на базата, като за целта ползвам Reflection и автоматични генератори на данни:

namespace HallScheduler.Data.Migrations.Seed

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using HallScheduler.Common.Constants;

using Seeders;

public class SeedFactory

{

public SeedFactory()

{

this.LoadSeeders();

}

public ICollection<ISeeder> Seeders { get; set; } = new List<ISeeder>();

private void LoadSeeders()

{

var assembly = Assembly.Load(Assemblies.Data);

var types = assembly.GetExportedTypes();

var seeders = types.Where(

x => x.GetInterfaces().Contains(typeof(ISeeder)) &&

!x.IsInterface &&

!x.IsAbstract);

foreach(var seederType in seeders)

{

this.Seeders.Add((ISeeder)Activator.CreateInstance(seederType));

}

this.Seeders = this.Seeders.OrderBy(x => x.Priority).ToList();

}

}

}

## 4.3 Autocomplete за Desktop клиента

Имплементацията на текстово поле с възможност за Autocomplete на Desktop клиента е реализирана с помощта на готова UI контрола, предоставена като Open-Source проект от организацията FeserWard. Контролата се казва Intellibox и след като добавим всички референции на dll-ки, описани в документацията, можем да я ползваме в XAML по следния начин:

<autocomplete:Intellibox

x:Name="autocompleteTextBox" SelectAllOnFocus="True"

DataProvider="{Binding QueryProvider}"

SelectedItem="{Binding SelectedItem}"

DisplayedValueBinding="{Binding Name}"

SelectedValue="{Binding SelectedValue}"

Height="30"

FontSize="18"

ResultsHeight="200" ResultsWidth="660"

ExplicitlyIncludeColumns="True"

Width="660">

<autocomplete:Intellibox.Columns>

<autocomplete:IntelliboxColumn DisplayMemberBinding="{Binding Type}" Header="Type" Width="100"/>

<autocomplete:IntelliboxColumn DisplayMemberBinding="{Binding Name}" Header="Name" Width="90"/>

<autocomplete:IntelliboxColumn DisplayMemberBinding="{Binding Block}" Header="Block" Width="60"/>

<autocomplete:IntelliboxColumn DisplayMemberBinding="{Binding Stage}" Header="Stage" Width="60"/>

<autocomplete:IntelliboxColumn DisplayMemberBinding="{Binding Room}" Header="Room" Width="60"/>

</autocomplete:Intellibox.Columns>

</autocomplete:Intellibox>

## 4.4 Сигурност на данните и криптиране

За подсигуряване на потребителските данни в Desktop клиента, ползвам вградения в .NET тип данни – Secure String, който съхранява символни низове като криптиран масив от байтове, от който с няколко допълнителни метода можем да извличаме нужната информация, без да я излагаме на достъп от хора, които практикуват атаки фокусирани към RAM паметта, и могат да прочетат всички данни записани в нея. Пример:

namespace HallScheduler.Desktop.Infrastructure.ExtensionMethods

{

using System;

using System.Runtime.InteropServices;

using System.Security;

public static class StringExtensionMethods

{

public static string ConvertToUnsecureString(this SecureString securePassword)

{

if (securePassword == null)

{

throw new ArgumentNullException("\"securePassword\" argument, in ConvertToUnsecureString() method, must not be NULL");

}

IntPtr unmanagedString = IntPtr.Zero;

try

{

unmanagedString = Marshal.SecureStringToGlobalAllocUnicode(securePassword);

return Marshal.PtrToStringUni(unmanagedString);

}

finally

{

Marshal.ZeroFreeGlobalAllocUnicode(unmanagedString);

}

}

public static SecureString ConvertToSecureString(this string password)

{

if (password == null)

{

throw new ArgumentNullException("password");

}

var securePassword = new SecureString();

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

{

securePassword.AppendChar(password[i]);

}

securePassword.MakeReadOnly();

return securePassword;

}

}

}

## 4.5 Алгоритъм за откриване на застъпвания при създаване и промяна на събития

Реализирал съм собствен алгоритъм, използвайки структурата двоично дърво със следните методи:

namespace HallScheduler.Data.Common

{

using System;

using System.Collections.Generic;

using Contracts;

public class ScheduleTree<T>

where T : IEvent

{

public ScheduleNode<T> Root { get; set; }

public List<ScheduleNode<T>> TreeAsList { get; set; }

public bool Insert(ScheduleNode<T> nodeToInsert)

{

return this.Insert(this.Root, nodeToInsert);

}

public bool Insert(ScheduleNode<T> currentNode, ScheduleNode<T> nodeToInsert)

{

if (currentNode == null)

{

this.Root = nodeToInsert;

return true;

}

if (nodeToInsert == null)

{

return false;

}

if (nodeToInsert.Data.EndsAt <= currentNode.Data.StartsAt)

{

if (currentNode.LeftChild == null)

{

// Directly insert as left child

currentNode.LeftChild = nodeToInsert;

return true;

}

else

{

// Go left

return this.Insert(currentNode.LeftChild, nodeToInsert);

}

}

else if (currentNode.Data.EndsAt <= nodeToInsert.Data.StartsAt)

{

if (currentNode.RightChild == null)

{

// Directly insert as right child

currentNode.RightChild = nodeToInsert;

return true;

}

else

{

// Go right

return this.Insert(currentNode.RightChild, nodeToInsert);

}

}

else

{

return false;

}

}

public bool CanInsert(ScheduleNode<T> nodeToInsert)

{

return this.CanInsert(this.Root, nodeToInsert);

}

public bool CanInsert(ScheduleNode<T> currentNode, ScheduleNode<T> nodeToInsert)

{

if (currentNode == null)

{

return true;

}

if (nodeToInsert == null)

{

return false;

}

if (nodeToInsert.Data.EndsAt <= currentNode.Data.StartsAt)

{

if (currentNode.LeftChild == null)

{

return true;

}

else

{

// Try left

return this.Insert(currentNode.LeftChild, nodeToInsert);

}

}

else if (currentNode.Data.EndsAt <= nodeToInsert.Data.StartsAt)

{

if (currentNode.RightChild == null)

{

return true;

}

else

{

// Try right

return this.Insert(currentNode.RightChild, nodeToInsert);

}

}

else

{

return false;

}

}

public void Print(ScheduleNode<T> startNode, int level = 1)

{

if (startNode != null)

{

Console.WriteLine($"Level: {level} - {startNode.Data.StartsAt} - {startNode.Data.EndsAt}");

}

else

{

Console.WriteLine("No elements in the tree");

}

if (startNode.LeftChild != null)

{

this.Print(startNode.LeftChild, level + 1);

}

if (startNode.RightChild != null)

{

this.Print(startNode.RightChild, level + 1);

}

}

public void BuildListFromTree()

{

this.TreeAsList = new List<ScheduleNode<T>>();

this.BuildListFromTree(this.Root);

}

public void BuildListFromTree(ScheduleNode<T> startNode)

{

if (startNode != null)

{

this.TreeAsList.Add(startNode);

}

else

{

return;

}

if (startNode.LeftChild != null)

{

this.BuildListFromTree(startNode.LeftChild);

}

if (startNode.RightChild != null)

{

this.BuildListFromTree(startNode.RightChild);

}

}

public void BuildTreeFromList(List<ScheduleNode<T>> schedule)

{

if(schedule != null)

{

for (int i = 0; i < schedule.Count; i++)

{

this.Insert(schedule[i]);

}

}

}

}

}

Пример за употреба на дървото:

public int Add(Event model)

{

// Prepare data

var modelAsScheduleNode = new ScheduleNode<Event>(model);

var schedule = this.Halls.GetById(model.HallId).Schedule

.Where(x => x.DayOfWeek == model.DayOfWeek)

.Select(x => new ScheduleNode<Event>(x))

.ToList();

// Build tree

var scheduleTree = new ScheduleTree<Event>();

scheduleTree.BuildTreeFromList(schedule);

// Try insert

var canInsertEvent = scheduleTree.CanInsert(modelAsScheduleNode);

if (canInsertEvent)

{

if (model.StartsAt.TotalMilliseconds < model.EndsAt.TotalMilliseconds)

{

try

{

this.Events.Add(model);

this.Events.SaveChanges();

return model.Id;

}

catch (Exception)

{

return API.Conflict;

}

}

else

{

return API.InvalidModel;

}

}

else

{

return API.Overlap;

}

}

# 5. Потребителско ръководство

## 5.1 Вход в системата

При стартиране на десктоп клиента, потребителят вижда формата за вход, която има 2 полета – email и парола. След попълване на 2-те полета и кликайки върху бутона „Sign in” потребителят изчаква сървърът да провери акаунта му и да реши дали да го вкара в системата. При неуспешен вход, заради грешни данни – във формата се появява съобщение в червен текст, което уведомява потребителя за възникналата грешка. При успешна валидация на потребителските данни – потребителят бива пренасочен към нова страница, която представлява главния панел с функционалности в системата.

## 5.2 Главен панел

След успешен вход, потребителят вижда главния панел с функционалности, където може да потърси зала по нейното име, блок, етаж или стая. Може да създаде ново събитие за конкретна зала, чрез бутон „Schedule event”. Може да стартира презентационен режим на залата, чрез бутон „Presentation mode”. Може да промени вече съществуващо събитие, чрез двоен клик върху някое от събитията визуализирани в графика на залата. Може и да излезе от системата, чрез натискане на бутона „Logout”, който автоматично затваря текущия прозорец и го връща към формата за вход.

## 5.3 Форма за съдаване на събитие

Тя се състои от всички полета необходими за създаване на събитие, като част от тях са имплементирани с Autocomplete текстово поле или Dropdown менюта, за по-голямо удобство на потребителя. В тази форма има само един бутон „Create”, който изпраща заявка към сървъра за създаване на събитие, а резултатът от заявката (съобщение) се визуализира в текстово поле със зелени или червени букви, за да уведоми потребителя какво се случва.

## 5.4 Презентационен режим

При клик върху бутона “Presentation mode” от главния панел с функционалности, потребителят бива пренасочен към нов прозорец, който показва текущото събитие, което се провежда в залата, ако има такова и следващото, което предстои да бъде проведено, ако има такова.

Тук показвате как вашият потребител трябва да може да взаимодейства с вашия продукт. – Описание на преходи от различни страници, резултати от работата със системата и др.

# 6. Заключение

Десктоп версията на продукта е готова за употреба. Всички функционалности описани във функционалните изисквания са имплементирани (освен добавянето на любими събития). С малки конфигурации, продуктът може да се пусне Open-Source, free for use, за да може да бъде доразработен и евентуално ползван и от други институции – училища, библиотеки, университети и т.н.

# 7. Литература

## 7.1 FeserWard Intellibox

<https://intellibox.codeplex.com/>

## 7.2 Microsoft Developers Network

<https://msdn.microsoft.com/bg-bg/>

## 7.3 DrawIO

<https://www.draw.io/>

# 8. Приложение

## 8.1 Halls controller

namespace HallScheduler.Server.API.Controllers

{

using System.Data.Entity;

using System.Threading.Tasks;

using System.Web.Http;

using Common.Constants;

using DataTransferObjects.Halls;

using Infrastructure;

using Infrastructure.Mapping;

using Services.Data.Contracts;

[RoutePrefix(API.Halls)]

public class HallsController : BaseController

{

public HallsController(IHallsService halls)

{

this.HallsService = halls;

}

public IHallsService HallsService { get; set; }

[HttpGet]

[Route(API.All)]

public async Task<IHttpActionResult> GetAll()

{

var result = await this.HallsService.All().To<HallBriefDTO>().ToListAsync();

return this.Ok(

new ResponseResultObject(

API.Success,

API.ReturnedItems(result.Count),

result));

}

[HttpGet]

[Route(API.Schedule)]

public IHttpActionResult GetSchedule(int hallId)

{

var hall = this.HallsService.GetById(hallId);

var result = this.Mapper.Map<HallScheduleDTO>(hall);

return this.Ok(

new ResponseResultObject(

API.Success,

API.ReturnedItems(API.Single),

result));

}

[HttpGet]

[Route("HallDetailsWithSchedule")]

public IHttpActionResult GetHallDetailsWithSchedule(int hallId)

{

var hall = this.HallsService.GetById(hallId);

var result = this.Mapper.Map<HallDetailedDTO>(hall);

return this.Ok(

new ResponseResultObject(

API.Success,

API.ReturnedItems(API.Single),

result));

}

[HttpGet]

[Route(API.FullSchedule)]

public async Task<IHttpActionResult> GetFullSchedule()

{

var result = await this.HallsService.All().To<HallDetailedDTO>().ToListAsync();

return this.Ok(

new ResponseResultObject(

API.Success,

API.ReturnedItems(result.Count),

result));

}

}

}

## 8.2 Events controller

namespace HallScheduler.Server.API.Controllers

{

using System.Web.Http;

using Common.Constants;

using Data.Models;

using DataTransferObjects.Events;

using Infrastructure;

using Services.Data.Contracts;

[RoutePrefix(API.Events)]

public class EventsController : BaseController

{

public EventsController(IEventsService eventsService)

{

this.EventsService = eventsService;

}

public IEventsService EventsService { get; set; }

[HttpPost]

[Route(API.Update)]

public IHttpActionResult UpdateEvent(EventDTO model)

{

if (!this.ModelState.IsValid)

{

return this.BadRequest(this.ModelState);

}

// Try update

var eventDatabaseModel = this.Mapper.Map<Event>(model);

var updateResult = this.EventsService.Update(eventDatabaseModel);

// Prepare response

var message = string.Empty;

var isSuccessful = false;

if (updateResult > 0)

{

message = "Event successfully updated.";

isSuccessful = true;

}

else if (updateResult == API.Overlap)

{

message = "Event overlap occured. Update failed.";

}

else if (updateResult == API.Conflict)

{

message = "Database validation error.";

}

else if (updateResult == API.InvalidModel)

{

message = "Event start time cannot be after event end time";

}

return this.Ok(

new ResponseResultObject(

isSuccessful,

message,

eventDatabaseModel));

}

[HttpPost]

[Route(API.Create)]

public IHttpActionResult CreateEvent(EventDTO model)

{

// Validate model

if (!this.ModelState.IsValid)

{

return this.BadRequest(this.ModelState);

}

// Try create

var eventDatabaseModel = this.Mapper.Map<Event>(model);

var creationResult = this.EventsService.Add(eventDatabaseModel);

// Prepare response

var message = string.Empty;

var isSuccessful = false;

if (creationResult == API.Overlap)

{

message = "Event overlap occured. Creation failed.";

}

else if (creationResult == API.Conflict)

{

message = "Database validation error.";

}

else if (creationResult == API.InvalidModel)

{

message = "Event start time cannot be after event end time";

}

else if (creationResult > 0)

{

message = "Event successfully added.";

isSuccessful = true;

}

return this.Ok(

new ResponseResultObject(

isSuccessful,

message,

this.Mapper.Map<EventDTO>(eventDatabaseModel)));

}

[HttpGet]

[Route(API.Delete)]

public IHttpActionResult DeleteEvent(int eventToDeleteId)

{

var eventToDelete = this.EventsService.GetById(eventToDeleteId);

this.EventsService.Delete(eventToDelete);

return this.Ok(

new ResponseResultObject(

true,

"Event successfully deleted",

this.Mapper.Map<EventDTO>(eventToDelete)));

}

}

}

## 8.3 Identity controller

namespace HallScheduler.Server.API.Controllers

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

using Common.Constants;

using Data.Common.Enumerations;

using DataTransferObjects.Account;

using Infrastructure;

using Microsoft.AspNet.Identity;

using Services.Data.Contracts;

[RoutePrefix(API.Identity)]

public class IdentityController : BaseController

{

public IdentityController(IUsersService users, IRolesService roles)

{

this.Users = users;

this.Roles = roles;

}

private IUsersService Users { get; set; }

private IRolesService Roles { get; set; }

[HttpGet]

[Authorize]

public IHttpActionResult Get()

{

var userId = this.User.Identity.GetUserId();

var user = this.Users.GetById(userId);

var userRoles = user.Roles.ToList();

var identityRoles = new List<IdentityRoleType>(userRoles.Count);

foreach(var role in userRoles)

{

var roleName = this.Roles.GetNameById(role.RoleId);

var identityRole = (IdentityRoleType)Enum.Parse(typeof(IdentityRoleType), roleName);

identityRoles.Add(identityRole);

}

var result = new UserIdentity()

{

Id = user.Id,

UserName = user.UserName,

Email = user.Email,

AcademicRank = user.AcademicRank,

Faculty = user.Faculty,

FullName = $"{user.FirstName} {user.LastName}",

Roles = identityRoles

};

return this.Ok(

new ResponseResultObject(

API.Success,

API.ReturnedItems(API.Single),

result));

}

}

}

## 8.4 Base controller

namespace HallScheduler.Server.API.Controllers

{

using AutoMapper;

using Infrastructure.Mapping;

using System.Web.Http;

public class BaseController : ApiController

{

protected IMapper Mapper

{

get

{

return AutoMapperConfig.Configuration.CreateMapper();

}

}

}

}

# 9. Source code

<https://github.com/Hall-Scheduler>