|  |  |  |
| --- | --- | --- |
|  | **Софийски университет „Св. Кл. Охридски”**  Факултет по математика и информатика  *Катедра „Софтуерни технологии”* |  |

**ДИПЛОМНА РАБОТА**

на тема

„Разработване на уеб базирана информационна система за управление на отсъствия на служители в корпоративна среда”

Дипломант: **Стоян Едрев Стоянов**

Специалност: **Софтуерни технологии**

Факултетен номер: **M24369**

Научен ръководител:

**доц. д-р Десислава Петрова-Антонова**

София, 2016 г.

**Съдържание** (препоръчителен обем без приложенията: от 60 до 80 стр.)

**Глава 1. Увод** (3-5стр.)

1.1. Актуалност на проблема и мотивация (0,5-1стр.)

1.2. Цел и задачи на дипломната работа (1-2стр.)

1.3. Очаквани ползи от реализацията (1-2стр.)

1.4. Структура на дипломната работа (0,5-1стр.)

**Глава 2. Преглед на предметната област (да се замени с конкретно заглавие според заданието)** (10-15стр.)

2.1. Основни дефиниции

2.2. Подходи, методи (евентуално модели и стандарти) за решаване на проблемите

2.3. Съществуващи решения (практически реализации)

2.4. Избор на критерии за сравнение и сравнителен анализ на решения/методи/стандарти/...

2.5. Изводи

**Глава 3. Използвани технологии, платформи и/или методологии (за практическото решаване на проблема)** (10-15стр.)

3.1. Изисквания към средствата (технологии, платформи и методологии)

3.2. Видове средства (технологии, платформи и методологии) и начин и място за използването им – сравненителен анализ

3.3. Избор на средствата (технологии, платформи и методологии)

3.4. Изводи

**Глава 4. Анализ** (10-15стр.)

4.1. Концептуален модел

4.2. Потребителски (функционални) изисквания (права, роли, статуси, диаграми, ...)

4.3. Качествени (нефункционални) изисквания (като напр. преносимост, използваемост, скалируемост, поддръжка, ...)

4.4. Работни (бизнес) процеси

4.5. Изводи

**Глава 5. Проектиране** (10-15стр.)

5.1. Обща архитектура – напр. слоеве, модули, блокове, компоненти...

5.2. Модел на данните (напр. база данни, файлова структура, ...)

5.3. Диаграми (на структура и поведение - по слоеве и модули, с извадки от кода)

5.4. Потребителски интерфейс (опционално)

5.5. Ресурсни и спомагателни модули (опционално)

**Глава 6. Реализация, тестване/експерименти и (евентуално) внедряване** (10-15стр.)

6.1. Реализация на модулите

6.2. Системна интеграция (опционално)

6.3. Планиране на тестването - тестови сценарии, процедури, ...

6.4. Модулно и системно тестване

6.5. Анализ на резултатите от тестването и начин на отразяването им

6.6. Експериментално внедряване (технологични изисквания, инсталиране, условия, използване, ...)

**Глава 7. Заключение** (1-2стр.)

7.1. Обобщение на изпълнението на началните цели

7.2. Насоки за бъдещо развитие и усъвършенстване

**Използвана литература** (min 10 литературни източника – статии, книги, с, форматирани съгласно MLA Style - <http://www.library.mun.ca/guides/howto/mla.php>)

**Приложения (опционално)**

**Приложение 1: Терминологичен речник (опционално)** (1-2стр.)

**Приложение 2, 3, ....:** напр. код или извадки от кода, наръчник на потребителя, екрани от потребителския интерфейс, спецификации, диаграми, и др.

**Изисквания за оформяне на дипломната работа:**

1. Това е препоръчителен шаблон, в зависимост от конкретното задание.
2. Йерархията на структуриране на съдържанието да не бъде повече от 3 нива, номерирани с арабски цифри – напр. 1.2.3.
3. Чуждестранните термини да бъдат преведени, а където това не е възможно – цитирани в *курсив* и нечленувани.
4. Страниците да бъдат номерирани с арабски цифри, в долния десен ъгъл.
5. Използваният шрифт за основния текст на описанието да бъде Times 12 или Arial 10, и Courier 9 за кода, с междуредие 16pt.
6. Да се избягват пренасянията на нова страница на заглавия на секции, фигури и таблици.
7. Да се избягват празни участъци на страници вследствие пренасянето на фигури на нова страница.
8. Всички фигури и таблици да бъдат номерирани и именовани (непосредствено след фигурата или таблицата).
9. Всички фигури и таблици да бъдат цитирани в текста.
10. Използваните фигури от други източници да бъдат цитирани.
11. Всички цитати да бъдат отразени в списъка на използваната литература.
12. Всички източници от списъка на използваната литература да бъдат цитирани в текста.
13. Използваната литература да се цитира съгласно MLA Style - <http://www.library.mun.ca/guides/howto/mla.php>

**Глава 1. Увод**

* 1. **Актуалност на проблема и мотивация (0,5-1стр.)**

Понятието *корпоративен* произлиза от латинското *corpus* (тяло, тяло от хора). По времето на император Юстениан (527 – 565 г.), в римското право съществуват няколко единици, които попадат в категорията корпоративен. Това например са самата държава, общините, политическите групи и търговците.

В наши дни, думата е понятие обозначаващо юридически лица в системата на общото право. Корпораците са сдружения със стопанска или нестопанска цел създадени със закон и регистрирани според определена процедура, и имат права и задължения, независими от тези на участниците в тях. В разговорната реч, обикновенно с думата се наричат големи търговски дружества с голям брой съдружници, служители и свободна промяна на собствеността на дяловете.

Под отсъствие на служител, ще разбираме всяко негово неяваване в работно време, поради използването на платен или неплатен отпуск, заболяване или други причини.

В исторически план, борбата за платена почивка започва в края на 19-ти век, със разпространението на индустриалната революция и работата в завод. По това време единствените почивни дни са неделя и някои религиозни празници. През 30-те години на двадесети век, международната работническа организация, одобрява конвенция номер 52 за платената ваканция, която призовава за поне шест работни дни почивка на година. Идеята за платена почивка отново се разглежда през 1948 г. от ООН чрез Универвална Деклатация на правата на човека, която гласи „*Всеки има право на почивка, включително разумно ограничаване на работните часове и периодични платени отсъствия*“. [8]

В наши дни, в повечето страни по света има приети закоnи, които налагат на работодателите да осигурят определено количество работни дни за отпуск. Например в Европейският съюз минималния брой дни е 20, в Канада е 10. В някои страни, например в САЩ няма изискване за платен годишен отпуск, въпреки че някои щати налагат на работодателите да осигурят поне 10 работни дни платен отпуск.

Един от най-малко организираните аспекти в съвременните компании е управлението на отсъствията на служителите. В преобладаващата част от фирмите, масово се използват изпращане на вътрешни имейли, електронни таблици, лепящи бележки и др., за да се поиска, одобри и проследи времето, в което служителят няма да е на работа.

Използването на софтуерна система за управление на този вид дейност, означава, че всички отсъствия, заявки и одобрения за служителя ще бъдат централизирани и съхранени поне на едно място. В резултат на това, служителите ще могат да следят техния текущ баланс и да направят избор, кога да използват оставащите им дни, ръководителите ще могат да проследят всички отсъствия на своите подчинени, а административните разходи ще се намалят и ще се повиши производителността на служителите.

* 1. **Цел и задачи на дипломната работа (1-2стр.)**

Дипломната работа си поставя за цел, да се разработи уеб базирана информационна система, съобразена с нуждите на съвременна компания за по-лесното и бързо управление на времето, в което служителите й отсъстват. Уеб системата трябва да притежава модерен и изчистен потребителски интерфейс и да може да бъде достъпвана посредством, мобилни устройства, както и от настолни компютри независимо от операционните им системи. Системата трябва да притежава потребителски модул, чрез който да бъдат изпълнявани основни функционалности, като например заявяване на отпуск, справка за текущи и минали заявки, информиране за болест или друг тип отсъствия. Освен това системата трябва да има административна част, чрез която да се управляват различните потребители, потребителски роли, както и основни елементи от нея, като например одобряване на потребителски заявки, въвеждане на задължителни почивни дни във фирмата, официални празници и др.

Първоначално, системата ще бъде доставена в облачна среда, което допълнително ще улесни достъпа и използваемостта на системата. Рабира се това не е ограничение, и при желание на клиента, системата лесно може да достави на предоставена от него среда.

За успешното реализиране на така поставената цел, е нужно да се решат следните основни задачи:

* Изследване и анализ на нуждите за разработка на система за управление на отсъствия на служителите. За целта ще бъде изследвана дейността на компаниите, свързани с управлението на този аспект от работата им.
* Проучване и сравнителен анализ на съществуващите решения в предметната област. Запознаване с техните предимства и недостатъци.
* Анализ на изискванията към информационната система.
* Проектиране на информационната система. За целта ще бъдат направени модел, архитектура на високо ниво, обектно-ориентиран анализ.
* Разработване на информационната система.
  + Избор на платформа за разработка. Два са основните и най-важни фактори при избора за платформа за разработка. Първия от тях е до колко дадена платформа е масова, с което ще се улесни поддръжката и надграждането. Другият е до колко дадена платформа е позната на разработчика, което ще се отрази върху бързината и качеството на изработка.
  + Избор къде ще се инсталира. Най-общо има два варианта за инсталация на системата: при клиента или отдалечено. Както се спомента по-рано, първоначално системата ще се достави в облачна среда, като при желание на клиента, тя може да се достави на среда предоставена от него.
  + Защита.
  + Потребителски интерфейс.
  + Реализация.
* Тестване на информационната система.
  + Тестване на ниво едица.
  + Тестване на ниво компонент.
  + Тестване от край до край
* Внедряване на информационната система.
  + Внедряване на системата в облачна среда.

**1.3 Очаквани резултати от реализацията**

С успешната реализация на тази дипломна работа предоставяме документ, анализиращ проблема с управлението на отсъствията на служители.

Освен този документ ще е налична и уеб базирана информационна система, илюстрираща на практика реализацията на една такава система. Когато обобщаваме ползите, следва да имаме предвид, че една система

никога не може да се хареса на абсолютно всички, както и да бъде удобна на

абсолютно всички. Всяка огранизация, може да използва изграденият обобщаващ документ и приложение като отправни точки при реализацията на подобна система за нуждите й.

* 1. **Структура на дипломната работа (0,5-1стр.)**

**В Глава 2** – Предметна област, са включени основни дефиниции, най-често прилаганите методи и подходи, направен е обзор на съществуващи решения, както и най-добрите практики за решаване на такъв тип проблеми. Присъства и сравнителен анализ на представените решения. В краят на главата са представени съответните изводи.

**В Глава 3** – са разгледани различните видове технологии, чрез които може да се

разработи настоящата система. Част от технологиите са представени и визуално, чрез графики, с цел по-лесното разбиране на идеите, които седят зад тях. Направен е сравнителен анализ на различните технологии, като на база този анализ са избрани конкретни технологии, които да се използват при разработката на системата.

**В Глава 4** – се съдържа задълбочен анализ включващ концептуален модел, фунцкионални изискивания, нефункционално изискивания, както и работните бизнес процеси. Концептуалния модел описва от високо ниво, различните секции в системата, фунцкионалните изискивания описват в детайлност конкретните изискивания към системата, а частта с работни процеси съдържат в себе си, доста изображения и диаграми описващи бизнес процесите, които реализира системата.

**В Глава 5** – е показана архитектурата на самата система. Разгледани са различните слоеве, както и модулите на системата (тяхната роля и в кой слой се намират). В главата са представени също така модел на данните, потребителският интерфейс на системата, спомагателни модули, както и някои от основните процеси протичащи при взаимодействие със системата. Освен това, са показани и разгледани извадки от код-а, от основни модули и слоеве в уеб системата.

**В Глава 6** - е описан процесът при разработката на отделните модули, системната интеграция, предвидените тестови сценарии са описани в детайност. Също така е показана и описана примерно внедряване с помощта на Microsoft Azure.

**В Глава 7** – е направен кратък анализ и обобщение на ползите от разработката на текущата система. Също така са формирани и възможните бъдещи подобрения на системата.

**Глава 2. Преглед на предметната област** (10-15стр.)

**2.1. Основни дефиниции**

*Работник/служител -* е физическо лице, което отдава под наем работната си сила, като я предоставя за временно ползване от друг субект на правото. За работник се приема лице, което полага предимно физически труд. А за служител се приема лице, което престира предимно интелектуален, умствен труд. [1]

*Работодател* - физическо лице, юридическо лице или негово поделение, както и всяко друго организационно и икономически обособено образувание (предприятие, учреждениеи и др, което самостоятелно наема работници или служители по трудово правоотношение. [5]

*Кодекс на труда* - урежда правото на работника или служителя на различни видове отпуск - основен и допълнителен платен годишен отпуск, неплатен, за обучение, за бременност, раждане и майчинство, за временна неработоспособност и т.н.

*Отпуск* - oсвобождаването на работник/служител да предоставя временно работната си сила за ползване от работодателя, при това трудовите правоотношения продължават да съществуват между неговите страни. [2]

*Платен отпуск -* период от време, в рамките на който работникът/служителят не е задължен да изпълнява трудовите си задължения, но има право на парично обезщетение, което е съразмерно с неговото трудово възнаграждение (чл.155 от Кодекса на труда) [2]

*Неплатен отпуск -* период от време, през което работникът или служителят не е на работното си място, не изпълнява трудовите си функции, не полага труд и не получава трудово възнаграждение, но трудовото му правоотношение не се прекъсва, а се запазва (чл.160 от Кодекса на труда). [3]

*Отпуск за изпълнение на граждански и обществени задължения:*

* *встъпване в брак* – на служителя/работника се предоставят 2 платени работни дни (чл. 157, ал. 1 от Кодекса на труда)
* *кръводаряване* – служителя/работника се освобождава да изпълнява трудовите си задължения в рамките на 2 дни (платени) за деня на прегледа и кръводаряването, както и 1 ден след него (чл. 157, ал. 2 от Кодекса на труда)
* *смърт на близки* – служителят/работника се освобождава в рамките на 2 работни дни при смърт на родител, дете, съпруг, брат, сестра и родител на другия съпруг или други роднини по права линия (чл. 157, ал. 3 от Кодекса на труда)

*Отпуск за временна неработоспособност -* Работникът или служителят има право на отпуск при временна неработоспособност поради общо заболяване или професионална болест, трудова злополука, за санаторно-курортно лечение и при належащ медицински преглед или изследване, карантина, отстраняване от работа по предписание на здравните органи, гледане на болен или на карантиниран член от семейството, належащо придружаване на болен член от семейството за медицински преглед, изследване или лечение, както и за гледане на здраво дете, върнато от детско заведение поради карантина в заведението или на детето. (чл. 162, ал. 1 от Кодекса на труда)

*Отпуските за майчинство -* ползването на такъв вид отпуск е субективно право на майката и се зачита за трудов стаж. Жената-майка има право на отпуск в размер на 410 дни за всяко свое дете, като първите 45 дни се ползват преди самото раждане [4]

*Отпуските за обучение* - Правото на ползване на отпуски за обучение е регламентиран в чл. 169-171а от Кодекса на труда. Съгласно чл.169, ал.1 от Кодекса на труда, право на платен отпуск за обучение в размер на 25 работни дни за всяка учебна година имат работници или служители, обучаващи се без откъсване от производството със съгласието на работодателя. Отпуските за обучение се отнасят само за работници и служители, които се обучават без откъсване от производството /задочно учащи се, вечерно обучаващи се, дистанционна форма на обучение/.

*Отдел човешки ресурси –* отдел, чиято работа се свързва най-често с набор на нови служители, подбор на служители, заплащане, оценката на изпълнението, обучиние и развитие, комуникация и администриране.

Мениджър - член на организация, който обединява и координира работата на други нейни членове.

**2.2. Подходи, методи за решаване на проблемите**

Идеята за разработване на дипломната работа се роди, след забелязаните несъвършенства и неефективности в съществуващите методи за управлението на отсъсвия на служителите в компаниите, в които авторът е работил. За изясняването и дефинирането на проблемите се осъществиха няколко разговора със служители от отдел човешки ресурси и от служители от финансовия отдел. По време на разговорите стана явно, че настоящите методи за администриране на този тип дейност, са твърде неефективни, времеотнемащи и често предразполагащи към грешки. Например, в настоящият момент, процесът за искане на платен отпуск е следният: служителят изпраща имейл до мениджъра си и клиента, с който работи (ако има такъв), че желае да използва част от годишният си платен отпуск. След получване на одобрение, на изпратеният имейл, служителят го изпраща на специална имейл група [vacation@company.com](mailto:vacation@company.com). По този начин, служителите от отдел човешки ресурси са уведомени за отсъствието на служителя и пристъпват към попълване (обновяване) на електронна таблица, чрез програмата Excel фиг. 1.



Фигура 1. Текущ процес за заявяване на отпуск

Следва, разпечатване на молбата за отпуск на хартиен носител от служителя и разписването й. Готовата молба се доставя на служителите от отдел „човешки ресурси“.

След разговорите, сформирахме следните проблеми на текущия процес:

* Липса на валидация на отставащи дни от годишния платен отпуск.
* Липса на валидация на брой работни дни при избор начална-крайна дата.
* Трудности при синхронизацията на електронните таблици, тъй като всеки служител от отдел „човешки ресурси“ има локално копие.
* Трудности при управлението на дни компенсация и прехвърляне на отпуск от предходна година.
* Липса на т.нар. календар на отсъствия, в който се отразява всеки служител, който в момента отсъства.
* Отговаряне на едни и същи въпроси на служители свързани с оставащ брой дни отпуск да края на годината.

Ясно беше заявено, че хартиените документи не могат да бъдат премахнати, тъй като чрез тях, компанията се отчита пред съответните държавни структури.

В крайна сметка, се стигна до извода, че внедряването на система решаваща тези проблеми е наложителна и силно препоръчителна. Тя би улеснила, както служителите така и администрацията на този тип дейност.

След провеждането на множество разговори и попълването на въпрешна за фирмата анкета, се стигна до следният списък с изисквания и функционалности:

* Информационната система винаги да бъде достъпна онлайн за устройства с различен размер на екрана.
* Валидация на оставащи дни след като заявка за отсъстрие е направена.
* Улеснено и бързо попълване на форма за заявка за отсъствие.
* Добавяне на бележки.
* Справка оставащи дни за годишен платен отпуск.
* Справка за история на подадените заявления за отсъствие.
* Възможност за лесно и бързо одобреване/отказ за мениджъри.
* Възможност за подръжка на повече от един език в системата, тъй като в компанията работят служители от различни и в различни държави.
* Указания и често задавани въпроси.

**2.3.** **Съществуващи решения**

**2.3.1 TrackStar Time Off Management System**

TrackStar Time Off Management System [6] e уеб базирана информационна система, решаваща проблема с управлението на отсъсвия на служители. Подходяща е както за малки така и за големи голени компании. Системата позволява интеграция със съществуващи системи. Модификациите спрямо специфичните нужди на фирмите са ограничени.

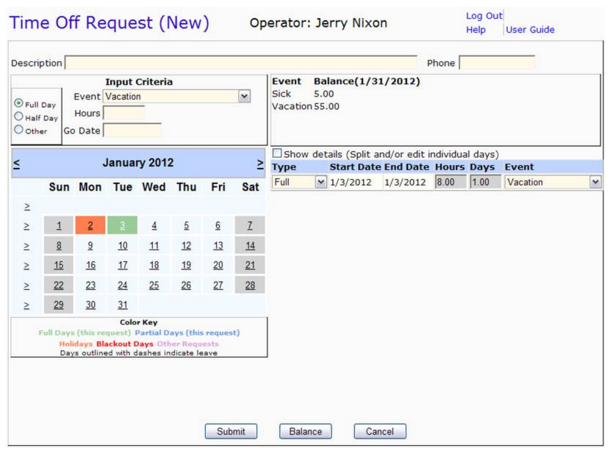
В главният модул на системата за управление на отсъстия присъстват стандартни функционалности като:

* Заявка за отсъствие.
* Изчисляване на баланс.
* Известяване чрез имейл.
* Възмжност за интеграция с календар.

Като минуси на системата, могат да се подчертаят:

* Липсата на валидация на оставащи дни платен годишен отпуск.
* Липса на функционалност за въвеждане на дни компенсация.
* Автоматични принтиране на одобрените заявки.

Изглед към системата може да се види на фигура 2.



Фигура 2. Изглед към TrackStart Time Off системата

**2.3.2 CWS TimeOut**

CWS TimeOut [7] е уеб базирана систем за управление на отсъствия на служители. Предвидена е за малки и средни предприятия. Съществуват три вида потребители на системата – обикновен потребител, мениджър и администратор. Процесът на заяваване на отсъствие е следния: потребителя попълва нужната форма, а мениджъра одобрява или отказва.

Администраторите могат да редактират почти всичко в системата, от промяната на роли на потребители до промяна на заяка за отсъствие.

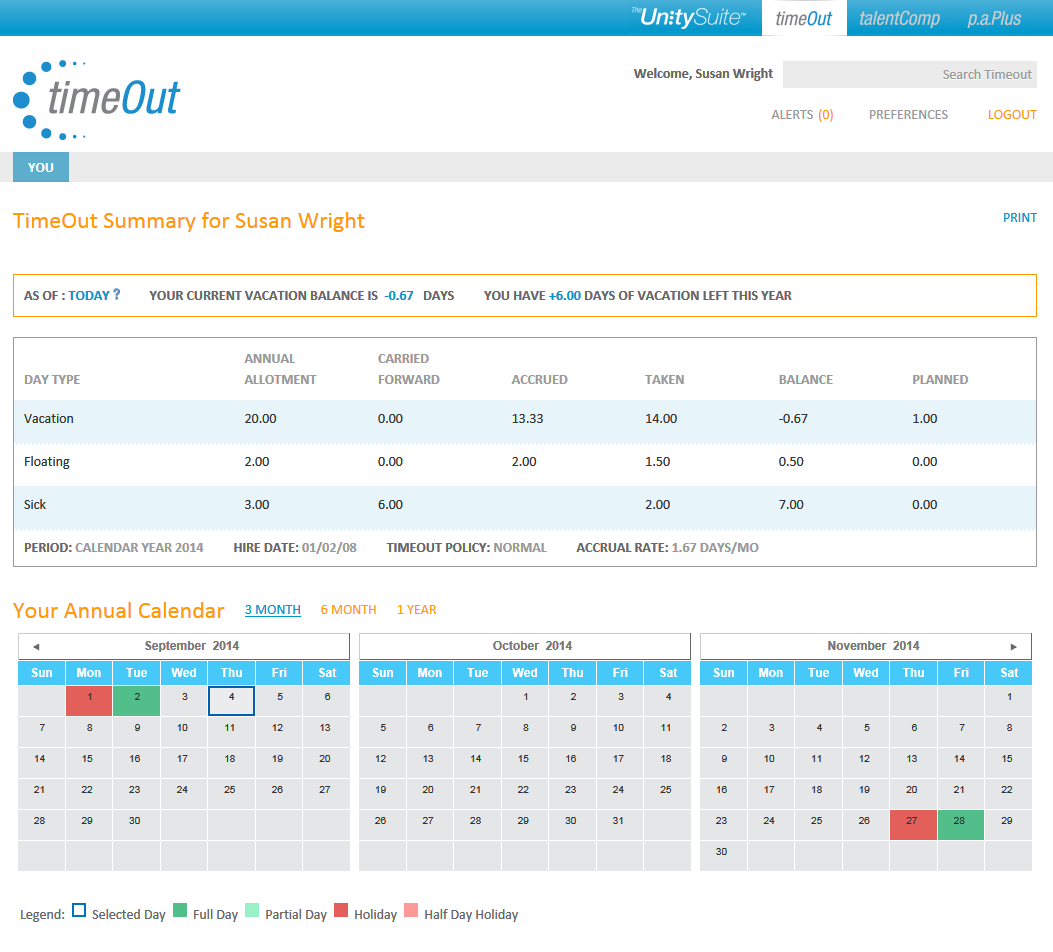
Основни функионалности на системата са:

* Въвеждане на брой платени дни на годишна база.
* Възможност за справка за направени заявки.
* Въжможност за валидация на оставащи/използвани дни.
* Поддръжка на много езици.

Като минуси към системата могат да се разгледат:

* Липса на функционалност за въвеждане на дни за компенсация.
* Липса на фунцтионалност за автоматично принтиране на одобрени заявки.
* Ограничени възможности за инеграция със съществуващи системи.
* Трудности на визуализация на устройства с по-малки екрани.

Изглед към системата може да се види на фигура 3.



Фигура 3. Изглед към системата CWS TimeOut

**2.3.3 DataBasics Time Off**

DataBasics Time Off е онлайн уеб базирана система, която позвалява на компании (от малки до големи) да управляват ефективни отсъствията на служителите си. Системата предоставя възможност за проследяване на всички видове отсъствия от платени/неплатени до използването на болнички. Също така могат да бъдат въвеждани и празници.

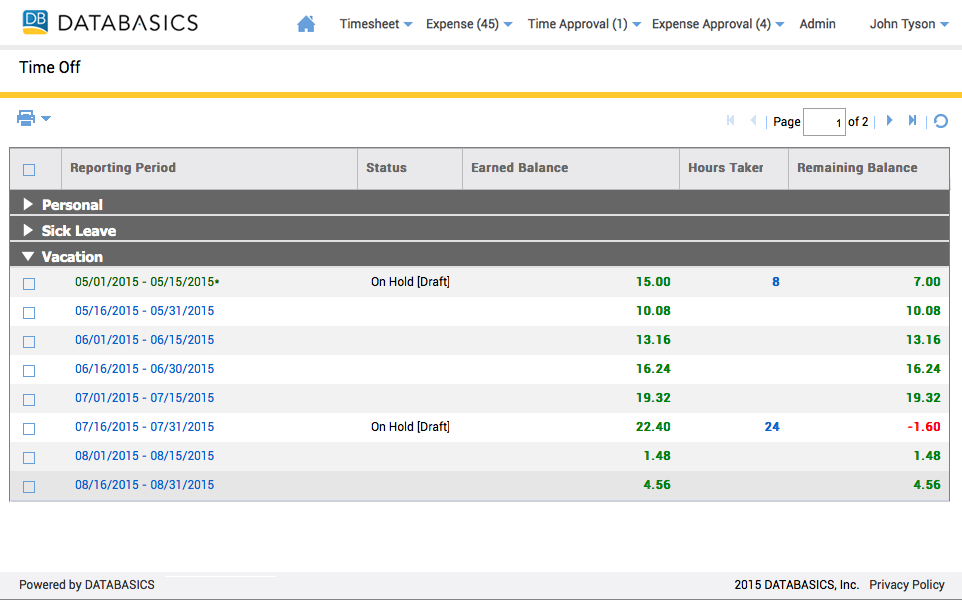
Основни функционалности на системата са:

* Въможност въвеждане на политики за платен отпускт на ниво компания.
* Възможност за детайлни справки за използвани дни.
* Възможност за интеграция със съществуващи продукти.

Като минуси към системата могат да се отбележат:

* Липса на функционалност за принтиране на одобрени заявки по шаблон.
* Високата цена на системата.

На фигура 4 може да се види и снимка от системата.



Фигура 4. Изглед към DataBasics Time Off

* 1. **Избор на критерии за сравнение и сравнителен анализ**

Някои от основните критерии, по които ще бъдат сравнени съществуващите системи и новопроектирана и разработана система в тази дипломна работа са:

* Потребителски интерфейс.
* Достъпност от устройства с по-малък размер на екрана.
* Справка за оставащи дни платен отпуск, включително компенсации и прехвърлени дни от предишна година.
* Възможност за принтиране на молби за отпуск по шаблон.
* Възможност за задаване на официални празници и работни дни които да се обявят като неработни за цялата огранизация.
* Модул с упътвания и често задавани въпроси.

Следващата таблица съдържа обобщени данни за разгледаните по-горе съществуващи системи. Важно е да се отбележи, че в колоната „Потребителски интерфейс“ оценките са от 1 до 5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Потребителски интерфейс | Достъпност от устройства с по-малък размер на екрана | Справка за оставащи дни платен отпуск, вкл. компенсации и прехвърлени дни от предишна година | Принтиране на молби за отпуск по шаблон | Задаване на официални празници и работни дни които да се обявят като неработни за цялата огранизация | | Модул с упътвания и често задавани въпроси | | Възможност за внедряване на системата в облачна среда |
| TrackStar Time Off | 2 | не | да | не | не | не | | не | |
| CWS TimeOut | 4 | не | да | не | не | да | | не | |
| DataBasics Time Off | 4 | да | да | не | не | не | | не | |
|  |  |  |  |  |  |  | |  | |

**2.5.** **Изводи**

След изготването на сравнителният анализ, се вижда ясно, че всички съществуващи системи, притежават основни функционалности, които позволяват управлението на този тип дейност. Всички разгледани системи могат да бъдат достъпвани онлайн, но не при всички е проекритано показването на по-малки екрани. Също така при всички системи липсва принтирането на заявка по шаблон, което допълнително улеснява администрирането пред държавните институции. Както видяхме, никоя от разгледаните системи, не притежава възможности за отпределяне официални празници, както и за задаване на неработни дни за цялата компания, последното е полезно когато се правят големи блокове от почивка, например около коледните и новогидишни празници. Освен това, при всички разгледани системи, липсва възможност за лесното им внедряване в облачна среда. Също така при системата CWS TimeOff липсва и лесна интеграция със съществуващи системи, което допълнително затруднява внедряването й.

Основната задача, на тази дипломна работа е да проектира и реализира система, която решава част от тези проблеми и пропуски, и да улесни възможно най-много работния процес свързан с тази дейност.

**Глава 3. Използвани технологии, платформи и/или методологии (за практическото решаване на проблема)** (10-15стр.)

При започването на проектирането и реализацията на дипломната работа имаме пълната свобода да изберем платформата, езика за програмиране, методите и шаблоните за решаването на дефинираните и поставени проблеми.

* 1. **Изисквания към средствата (технологии, платформи и методологии)**

За успешното изграждане на уеб базираната информационна система, най-общо ще имаме нужда от: езици за програмиране, уеб сървър (локален и отдалечен), интегрирана среда за разработка (IDE), система за управление на бази от данни (СУБД), хранилище и система за управление на изходният код. За успешното реализиране на проекта, трябва да се определят и изберат съответните платформа и инструменти.

* 1. **Видове средства и начин и място за използването им**

**3.2.1 Езици за програмине**

***C#*** *-*  обектно-ориентиран език за програмиране, който е разработен от Microsoft, като част от софтуерната платформа .NET. Програмите на C# представляват един или няколко файла с разширение .cs, в които се съдържат дефиниции на класове и други типове. Важно е да се отбележи, че код написан на C# не се компилира до машинен код за определен процесор, а до специфичен код, наречен междинен код, поради това за изпълнението на програма написана на езика е нужна виртуална машина наречена обща машина за изпълнение (common language runtime – CLR). C# е силно типизиран език, като позвалява част от кода да съдържа императивни, декларативни и функционални парадигми. Създаден през 2000г. днес езикък се поддържа и обновява редновно, като в моменра на писането на този текст актуалната му версия е 6.0. C# може да се използва за създаването на клиентски приложения, уеб услуги, разпределени системи, клиент-сървър приложения и др. Също така, езикък е стандартизиран от две огранизации: ECMA-334 и ISO-23270:2006. [9][10]

***Java***- е обектно-ориентиран език за програмиране. Създаден е през 1995г. от Sun Microsystems, и закупен от Oracle през 2009г. Кодът, написан на Java не се компилира до машинен код за определен процесор, а до специфичен за езика код, наречен байт код. Поради това за изпълнението на програма, написана на Java е необходима т. нар. Виртуална машина (на английски: Java Virtual Machine). Това решава редица проблеми, но и създава нови. Основните предимства на тази реализация са: лесната преносимост на кода между различните платформи, автоматичното управление на паметта, висока степен на согурност и др. Главен недостатък е необходимостта от допълнителни ресурси (под формата на процесорно време и памет) за изпълнението на самата виртуална машина. [11]

***Python*** - е интерпретируем, интерактивен, обектно-ориентиран език за

програмиране, създаден от Гуидо ван Росум в началото на 90-те години.

Философията на езика, набляга на четимостта на кода. Синтаксиса му позволява, програмистите да се изразяват в по-малко редове код.

Той притежава вградени сложни типове данни като гъвкави масиви и речници.

Python позволява разделянето на една програма на модули, които могат

да се използват отново в други програми. Също така притежава голям набор от

стандартни модули, които да се използват като основа на програмите.

Съществуват и вградени модули, които обезпечават такива неща като файлов

вход/изход,различни системни функции, сокети, програмни интерфейси към

GUI-библиотеки и други. Тъй като Python е език, който се интерпретира, се

спестява значително време за разработка, тъй като не са необходими

компилиране и свързване за тестването на дадено приложение. В наши дни, езикът активно се развива. Има две основни версии, като в момента на писане на текста те са 2.7 и 3.5. Двете версии не са напълно съвместими. [12]

***Ruby***- е интерпретируем, динамичен, интерактивен, обектно-ориентиран език за

Програмиране, създаден от Якохиро Матсумото през 1995г. Според създателя си, Ruby е вдъхновен от Perl, Smalltalk и Lisp. Най-новата му версия, в момента на писане на този текст е 2.3. Езикът е с отворен код. Най-известната рамка за разработване към този език се казва Ruby on Rails, или само Rails. Тя също е с отворен код и се използва за създаването на уеб базирани системи. [13]

***JavaScript***- е интерпретируем език за програмиране, разпространяван с повечето Уеб браузъри. Поддържа обектно-ориентиран и функционален стил на програмиране. Създаден е от Брендан Айк в Netscape през 1995-та. Най-често се прилага към HTML описанието на дадена Интернет страница, с цел добавяне на функционалност и зареждане на данни. Може да се ползва също за писане на сървърни скриптови приложения с помощта на платформата NodeJS. Javascript е стандартизиран под името EcmaScript. В момента, най-новата версия е EcmaScript-2015, позната още като EcmaScript-6. [14]

***HTML*** - e съкращение от HyperText Markup Language /език за маркиране

на хипертекст/. Той е стандарт в Интернет, като правилата се определят от

Международната организация по стандартизация W3C . Описанието на

документа става чрез специални елементи, наречени HTML тагаве. Елементите

са основната градивна единица на уеб страниците, тъй като чрез тях се оформят

отделните части от текста на една уеб страница, като заглавия, цитати, раздели,

хипертекстови препратки и т.н. В повечето случаи HTML кодът е написан в

текстови файлове и се хоства на сървъри, свързани към Интернет.

Създаването на HTML-базирана уеб страница може да се извърши с

помощта на обикновен текстов редактор. Този начин изисква познаване на

HTML тагове, така че те да бъдат интегрирани в текста, който ще се показва на

страницата. Към настоящият момент, най-новата версия е 5.0, която въвежда много нови свойства и семантични елементи, като например video, audio, canvas, article, header, nav елементи. [15]

***CSS*** *-* е създаден през 1996г., първоначално като средство за разделяне на

съдържанието от представянето му, днес той се използва основно за визуално

оформление на HTML страници, но също така може да се приложи върху

произволен XML документ. Cascading Style Sheets е удобен начин за

форматиране на текста, шрифтовете, изображенията и всичко останало на

страницата. Разполагането на различните елементи на страницата е с точност до

последния пиксел. Важна особеност е че ако се промени един стил, зададен в

началото на HTML страницата, промените се отразяват върху целия документ.

CSS предлага големи удобства при изграждането на даден HTML документ.

Най-голямото от тях е свързано с контрола на голям набор HTML документи,

като контролирането на външния вид на страниците става чрез промяна на един

единствен файл – CSS файла, без да е нужно да се променя HTML кода във всяка една от HTML страниците. Позиционирането на CSS в отделен файл позволява на уеб дизайнера лесно да разграничи съдържанието /HTML/ и дизайна /CSS/ на страницата. В настоящият момент най-актуалната версия на стандарта е 3.0.

**3.2.2 Бази от данни**

***MySQL*** – една от най-популярните СУБД с отворен код. MySQL e популярен избор за много уеб информационни системи. Също така системата заема главна част от т.нар. LAMP (Linux, Apache, MySQL, Perl/PHP/Python) стек за разработка на уеб системи с отворен код. MySQL е използвана в много известни и широкомащабни уеб системи. Пример за това са Google, Facebook, Twitter, Flickr и YouTube.

Предимствата на MySQL включват:

* Висока производителност;
* Ниска цена;
* Лесно конфигуриране и обучение;
* Преносимост на различни операционни системи;
* Достъпен сорс код.

След закупуването си от Oracle, няколко нови версии бяха създадени, които предлагат по-широк спектър от възможности и функционалности. Важно е да се отбележи, че след придобиването от Oracle, няколко ключови разработчика от MySQL започнаха развието на системата в различна посока и продукт, който отново е изцяло с отворен код. Неговото име е MariaDB. [16]

***Oracle******Database***– една от най-популярните платени обектно-релационна системи за управление на бази от данни, създадена от Oracle Corporation през 1977г. Системата е изградена около релационна рамка, в която обектните данни могат директно да бъдат достъпени от потребителя посредствум комнади на езика SQL. Системата е напълно мащабируема и често е използвана в големи частни организации, които съхраняват и обработват големи количества данни. [17]

***PostgreSQL***– обектно-релационна система за управление на бази от данни с отворен код, акцентираща върху разширяемостта и съвместимостта със стандартите.[18] Тя е многоплатформена система, поддържаща почти всички операционни системи, като за всички от тях съществуват инструменти за работа както от конзолата така и през графичен потребителски интерфейс.

Предимствата на PostgreSQL включват:

* Корпоративна сигурност
* Отлична производителност
* Съвместимост с функционалности на Oracle СУБД

***Microsoft SQL Server*** *­*– система за управление на бази данни разработена от Майкрософт.[19] В момента има над 10 различни варианта на системата: от безплатна версия до корпоративна такава. В най-новите си версии, системата разполага с отлични възможности за производителност, и в-паметта технология за бързи заявки на всякакъв тип данни.

Предимства на Microsoft SQL Server:

* Разширени възможности за анализ на данните
* Разширени услуги за интеграция
* Възможности за репликация
* Фунцкионалности за разширени справки и уведомяване в реално време
* Налична версия в облачна среда на Windows Azure

**3.2.3 Сървъри**

***Apache***- уеб сървър с отворен код, който има ключова роля за първоначалното разрастване на WWW и интернет. Чрез него работят над 70% от сайтовете в световен мащаб. Счита се от много специалисти за платформа, според която се разработват и оценяват другите уеб сървъри.

Приложението е налично на много операционни системи, включително Unix, GNU, FreeBSD, Linux, Solaris, Mac OS X, Microsoft Windows, OS/2, Novell NetWare и други платформи.

Apache се разработва от отворено общество от разработчици – Apache Software Foundation. Обществото поддържа и разработва множество други проекти най-важните от които са Apache Ant, Apache SpamAssassin, Apache Tomcat и огромен брой проекти основани на XML езика, но Apache е най-известния и широко разпространен продукт. [20]

***Nginx***- е високопроизводителен уеб сървър с отворен код под BSD лиценз.

Подобно на други приложения от този вид, архитектурата на nginx е модулна - при компилиране на софтуера се определя кои модули да бъдат вградени в него. Съществуват и над 20 потребителски модула.

Уеб сървърът nginx се използва от множество големи уеб сайтове. Сред тях са Fastmail.fm, WordPress.com, Hulu, Github, Ohloh и много други.

Пример за сайт-титан в България, който използва nginx е Vbox7.com [21]

***IIS***-Internet Information Services е пакет от сървърни услуги, включващ уеб сървър, пощенски сървър и сървър за споделяне на файлове. Създаден от Microsoft, той е конкурента на Apache и sendmail при Unix системите. Най важния от горепосочените компненти е уеб сървърът. Той поддържа всички стандарти за уеб като HTTP 1.0/1.1. За разлика от Apache той може да работи само под Windows.

**3.2.4 Облачни среди**

***AppHarbor*** - представлява .NET платформа-като-услуга (PaaS). Създадена е през септември, 2010 от Рюн Сьоренсън, Тролс Томсън и Майкъм Фрис.

Разработчиците качват кода си на AppHarbor, използвайки Git или Mercurial. След това АppHarbor компилира кода, като се опитва да изпълни всички тестове на ниво единица (ако има такива). След като всички тези операции минат успешно, кодът вече е внедрен в облачната платформа на AppHarbor.

Едно от основните предимства на AppHarbor e, че позволява на разработчиците да съсредоточат своите усилия върху създаването на нови идеи и намирането на начини за тяхната реализация, вместо да генерират безброй версии на продуктите си, да се притесняват за внедряването им и да настройват различни конфигурационни файлове.

***Amazon Web Services (AWS)*** *-* колекция от отдалечени компютърни услуги, наричани още уеб услуги, които съставят една облачна платформа, предлагана от Amazon.com. Основните услуги са Amazon EC2 и Amazon S3. Тези продукти се предлагат на пазара като услуга за предоставяне на изчислителни мощности по-бързо и по-евтино отколкото ако клиентската фирма трябва сама да създаде физическа сървърна група. AWS се намира в 11 географски региона: Северна Вирджиния, където са базирани мнозинството сървъри, Северна Калифорния, Орегон, Бразилия – Сао Паоло, Европа – Ирландия и Германия, Югоизточна Азия – Сингапур, Източна Азия – Токио и Пекин и Австралия – Сидни. Има и „GovCloud“, базирана в Северозападната част на САЩ, предвидена за американски правителствени клиенти. Съществуват държавни агенции, които вече използват услугата в източния регион на САЩ. Всяка област се съдържа изцяло в рамките на една държава и всички нейни данни и услуги остават в рамките на определения район.

Всеки регион има множество достъпни зони, които са обособени центрове за данни, предоставящи AWS услуги. Достъпните зони са изолирани една от друга, за да се предотврати прекъсване на тяхното разпространение между зоните. Няколко услуги действат отвъд достъпните зони, докато други могат да бъдат конфигурирани да възпроизведат всички зони за разпространение на търсенето и да се избегне прекъсване от неуспехи. Към декември 2014 г., Amazon Web Services използват 1,4 милиона сървъри в 28 достъпни зони. [22]

***Microsoft Azure -*** платформа за облачни и инфраструктурни изчисления създадена от Майкрософт през 2010, позволяваща за създаването, доставянено и управлението на приложения и услуги чрез световна мрежа, управлявана в големи изчислителни центрове на Майкрософт.

Майкрософт Azure предоставя всички модерни услуги на пазара, като PaaS (платформа като услуга), IaaS (инфраструкура като услуга), SaaS (софтуер като услуга) и др. Освен това се поддържат различни езици и платформи за разработка, бъдейки както специфични за екосистемата за Майкрософт така и от трети страни.

В момента някои от най-използваните услуги са следните – мобилни услуги, услуги за съхранение, медийни услуги, мрежи за доставяне на съдържание и др.

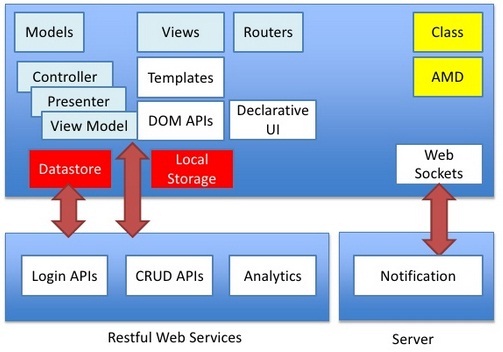
**3.3. Избор на средствата (технологии, платформи и методологии)**

Стратегията, използвана за реализиране на решението, описано в настоящата теза е подход, който се налага все повече при уеб-базираните информационни системи – SPA (Single Page Application) от страна на клиента и RESTFull услуги от страна на сървъра.

**3.3.1 SPA -** с развитието на високите технологии и разнообразието от устройства на пазара, с подобряването на свързаността и достъпа до интернет в глобален аспект се налага тенденция за използване все повече на уеб-базирани приложения. Изискванията и очакванията на потребителите също все повече се завишават, именно поради голямата конкуренция и разнообразие от решения достъпни в интернет пространството. Интерактивност, бърза реакция, разнообразие от предоставяни функционалности и то по възможно по-бърз и елегантен начин, максимална достъпност и платформена независимост са някои от очакванията, които налагат радикална промяна при подхода за предоставяне на решения в уеб-пространството. Така се заражда и се популяризира стратегията за SPA и по-конкретно реализирането ѝ базирано на JavaScript, HTML5 и CSS3.

В най-общ смисъл, SPA е уеб приложение, което се побира в една уеб страница, предоставяйки по-гладко и улеснено потребителско изживяване, което много наподобява, класическите декстоп системи. Говорейки за SPA, се има в предвид, че или всичкият нужен код - HTML, JavaScript и CSS се зарежда наведнъж, или нужните ресурси се зареждат динамично и се добавят към страницата, обикновенно при дадено взаимодействие на потребителя със системата. Страницата не презарежда изчяло в който и да било момент в процеса на използване, нито пренасочва към друга такава. Взаимодействието с SPA, често включва динамична комуникация с уеб сървър.

На фигура 5. представяме архитектура на дадене SPA приложение от високо ниво.



Фигура 5. Архитектура на SPA.

Както виждаме на фигура 5. една от основните идеи на дадено SPA приложение е ясното разделение на отговорностите. Това разделение се постига чрез, обособяването на съществени за всяка такава система обекти. Например имаме ясно разграничене на моделите, изгледите и контролерите, както и на начина по който потребителят ще се навигира в системата.

**3.3.2** **Платформа за разработка на клиентски приложения AngularJS** - е платформа с отворен код за уеб базирани SPA приложения. Разработена през 2009, в момента се поддържа от Google и общност от програмисти и корпорации целящи справянето с различни трудности при разработването на приложения от една страница. Целта е да се опрости както разработването, така и тестването на такъв вид приложения, чрез предоставянето на платформа по модела на архитектурата - Модел-Изглед-Контролер, заедно с всички често използвани компоненти за интернет приложения.

Платформата първо прочита HTML страницата, в която има вложени допълнителни специални тагове и атрибути. Тези атрибути се интерпретират от AngularJS като директиви сочещи дали да бъдат визуализрани входящи или изходящи части на страницата от модела представен от страндартни JavaScript обекти. Стойностите на тези JavaScript обекти могат да бъдат зададени в кода или извлечени от услуга на даден уеб сървър.

Основната концепция, въз основа на която е изграден AngularJS е model-view-controller (MVC) архитектурният модел, чиято идея е при разработването на големи приложения информацията да се раздели на логически дялове. MVC моделът разделя уеб приложението на три отделни части.

Модел (Model) – основно включва информация относно данните в приложението, която обикновено се взема от сървъра.

Изглед (View) – представлява информацията, която се визуализира на екрана.

Контролер (Controller) представлява софтуерният код, който осъществява връзката между данните, които се пазят на сървъра (Модел) и това кои от тях да се визуализират на екрана (Изглед).

AngularJS е разработен, като се следва идеята, че декларативното програмиране трябва да се използва за разработката на потребителски софтуер и свързвайки софтуерните компоненти, докато императивното програмиране е по-добре да се използва при дефинирането на бизнес логиката. Платформата адаптира и увеличава възможностите на традиционния HTML да представя динамично съдържание чрез двупосочно събиране на данни (data-binding), която позволява автоматичната синхронизация на модел и изглед (MV). В резултат, AngularJS не набляга толкова на DOM манипулацията, а цели подобряването на възможността за тестване на приложението и неговата производителност.

На фигура 6. е представена архитектурата на AngularJS.



Фигура 6. Архитектура на AngularJS.

Друга основна характеристика на платформата, е че благодарение на вгдаденото инжектиране на зависимости (Dependency injection) тестването на ниво единица и от край до край е услеснено максимално.

**3.3.3 Други инструменти за разработка на клиетско приложение** –

**3.3.3.1 RequireJS** - е библиотека, която предоставя зареждане само на необходимите ресурси нужни на дадено уеб приложение и това да се случва асинхронно с цел подобряване на производителността. RequireJS отговаря за дефинирането на връзки между отделни обособени единици, техните зависимости и за тяхното зареждане на клиента само в случай, че те са реферирани. Този подход се нарича AMD (Asynchronous Module Definition).

**3.3.3.2 jQuery** - библиотека, която опростява достъпа до всеки елемент на дадена уеб-страница, като по този начин позволява лесно изграждане на динамична функционалност в страниците. jQuery е безплатен и софтуер с отворен код, лицензиран под MIT лиценз.

**3.3.3.3 Bootstrap** - безплатна и с отворен код рамка за клиентски уеб проложения. Bootstrap съдържа HTML и CSS, дизайнерски шаблони за форми, бутоно, навигации и други компоненти от потребителския интерфейс. Рамката улеснява изключително много изработването на така наречените уеб проложения с responsive дизайн.

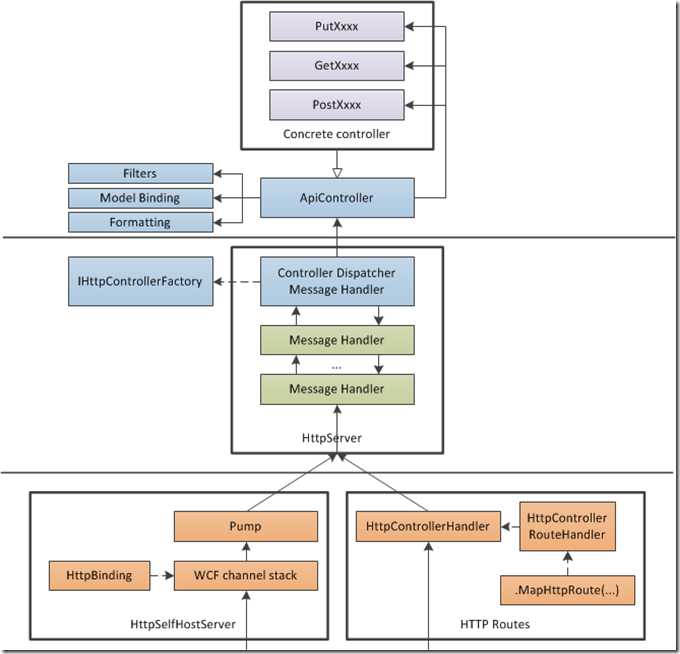
**3.3.4 Платформа за разработка на сървърни приложения – ASP.NET Web Api 2 -** е уеб базирана рамка за разработка, която е изградена на базата на Майкрософт .NET, версия 4.0 и по-висока, имплементираща HTTP (HyperText Transfer Protocol) спецификацията, и можеща да се използва за изграждането или консумирането на HTTP услуги, за разлика от SOAP (Simple Object Application Protocol) или WCF (Windows Communication Foundation) услуги.

Глен Блок, един от ключовите ръководители на проекта казва, че ASP.NET Web API обгръща HTTP като "първокласен играч" и позволява изключително лесното изграждане на RESTful услуги.

Рамката ASP.NET Web API имплементира RFC 2616 (HTTP 1.1 спецификация) оставяша и не налагаща мнение за REST. Това е ключово решение, което запазва платформата далеч от постоянните противоречия около REST, позволявайки да остане практична и фокусирана върху доставянето на стойностни услуги използвайки HTTP протокола.

Web API може да бъде използвано в лесното преобразуване на традиционнен ASP.NET уеб проект в мощен HTTP API, способен в използването на всички предимства на HTTP, включително разбирателство за съдържанието (content negotiation), кеширане, валидиране на конкурентност и др. Клиентската част от рамката, може да бъде използвана както от уеб, десктоп или мобилни приложения, така и от сървърни системи за консумирането на който и да е HTTP API създаден чрез която и да е рамка, стига тя да остане съвместима с HTTP спецификацията. На фигура 7. е представена и архитектурата на

продукта.

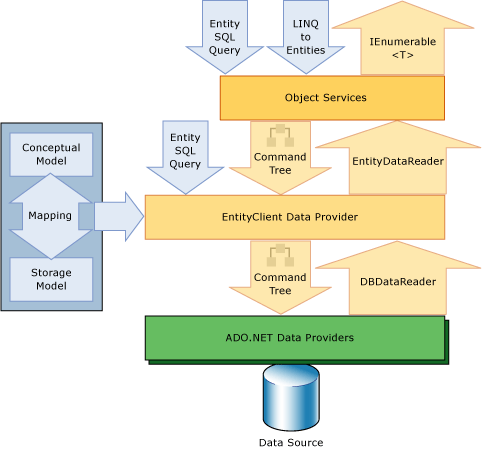


Фигура 7. Архитектура на ASP.NET Web API

Web API е революционен в много аспекти. Това е един от първите големи проекти на Майкрософт, които са разработени напълно като проекти с отворен код и получавайки съдействие и помощ от общността. Рамката е вдъхновена от концепциите и дизайна на ASP.NET MVC и е изградена с подобни абстракции.

**3.3.5 EntityFramework 6 -** Създадена през 2008г., Entity Framework е множество от технологии в ADO.NET, които поддържат и подпомагат разработването на ориентирани към данни системи. Платфоремата позволява на разработчиците да работят с данни по формата на бизнес обекти и свойства, като например Клиенти и адрес на клиент, без нужда да се притесняват от това, каква е базата данни в която се съхраняват тези обекти. С Entity Framework, програмистите могат да работят на по-високо ниво на абстракция, когато е нужен достъп до данните, което от своя страна позволява създаването и поддържането на ориентирани към данни приложения с по-малко код. Тъй като платформата е част от .NET, тя може да работи на всеки комкпютър на който има инстралирана версия на .NET.

Към настоящият момент, Entity Framework е публично достъпна с отворен код. В последните си версии, тя поддържа два основни механизма за разработване на слой за данни на дадено приложение. Първият от тях е така нареченият Code First. При него програмистът създава бизнес обектите, които ще се използват от системата, и базирано на тях Entity Framework създава базата от данни. Втория механизъм е така наречения Database First. При него, платформата прочита вече съществуваща база от данни и на базата на нея създава бизнес обекти, които разработчикът да използва. На фигура 8. е предтавена архитектурата на Entity Framework.



Фигура 8. Архитектура на Entity framework

Въпреки, че платформата е разработена от Майкрософт, тя далеч не се ограничава в подрръжката си на различни системи за управление на бази от данни. Съществуват доставчици за всички най-популярни системи за управление на бази от данни, като например Orable, DB2, MySQL и др.

**3.3.6 ASP.NET Identity 2** - е система за управление на членството, която позволява разработчикът да добави лесно възможности за вход и регистрация в дадена система. Потребителите могат да си създават профили с потребителско име и парола или могат да използват външни доставчици за вход като Facebook, Google, Microsoft, Twitter и други.

Платформата може да се конфигурира да използва, която е да е система за управление на бази от данни, където да запазва потребителските имена, пароли и лични данни.

ASP.NET Identity поддържа всички най-нови и модерни функционалности, които могат да се очакват от подобна платформа. В нея са включени, потребителско потвръждение, възстановяване на пароли, дву-факторно заверяване (two-factor authentication) чрез СМС И други.

**3.3.7 Unity -** е лек и разширяем контейнер за инжектиране на зависимости, разработен от Майкрософт, като проект с отворен код. Платформата значително улеснява изграждането на слабо свързани приложения и предоставя на разработчиците следните предимства:

* Опростено създаване на обекти, особено обекти с йерархична структура и зависимости.
* Абстракция на изискванията; това позволява на разработчиците да уточняват зависимостите по време на изпълнение на приложението и улеснява управлението на дублирането на код и силните зависимостите на различните модули в системата.
* Повишена гъвкавост благодарение на богатите възможности за конфигурация предоставени от контейнера.
* Услуги за местоположение; това позволява клиентите да запазват или кешират контейнера.
* Регистрация по конвенция.

**3.3.8 AutoMapper** - Програмен код, който преобразува даден тип обект към друг тип обект е податлив към грешки. Дейността по тестването на този код е дори още повече склонна към грешки. Преобразуването на типове и обекти може да се появи на много места в дадено приложение. Най-честите прояви са границите между отделни слоеве, като например слоя на потребителския интерфейс със бизнес слоя. Преобразуването на обект към обект води до изолирани модели, където всеки слой знае за типовете в него.

AutoMapper е преобразувател за обект към обект (object mapper). Преобразуването на обекти работи чрез трансформирането на входящ обект от даден тип към изходящ обект от друг тип. Това, което прави AutoMapper интересен, е че той предоставя конвенции решаващи проблeма с намирането на начин за преобразуването на тип А до тип Б изключително елегантно. Докато типовета А и Б спазват конвенции дефинирани в AutoMapper, интрумента работи почти автоматично без никаква допълнителна настройка.

**3.3.9 NLog** - безплатна и с отворен код платформа за регистриране и записване на всякакви събития в .NET. Разполога с богати възможсти за управление и настройване, което спомага за изключително лесното създаване и управляване на високо качествени доклади за дадено .NET приложение, независимо от неговата големина и сложност. Платформата се интегрира и работи с всеки .NET език. Възможностите за съхранение на доклади са изключително разнообразни, като част от тях са съхранение в бази от данни, файлове на локалната файлова система и други.

**3.3.10 Node.**js - мулти-платформена среда за изпълнение на сървърни и мрежови приложения с отворен код. Приложенията се пишат на JavaScript и могат да се изпълняват в Node.js среда под OS X, Microsoft Windows, Linux и IBM.

Node.js предоставя задвижвана от събития архитектура и неблокираща входно-изходна система за програмиране на приложенията, която оптимизира производителността и мащaбируемостта на дадена програма. Тези технологии са често използвани за мрежови приложения, работещи в реално време.

Node.js използва JavaScript двигателя V8 на Google, за да изпълнява код и голяма част от основните модули са написани на JavaScript.

**3.3.11 Gulp.js** - JavaScript базирана поточен инструмент за изграждане (build toolkit) на клиентски код. Някои от предимствата на Gulp включват: автоматизация на чести задачи при разработка, опростяването на повтарящи се задачи, и намаляването на времето за разработка. Най-честите употреби на Gulp са автоматизация на обединяването и минифицирането на JavaScript файлове, на сканирането и прилагането на строги правила за напинания код (т.нар линтери) и др.

**3.3.12 Visual Studio 2015** - Microsoft Visual Studio е мощна интегрирана среда за разработка (integrated development environment, IDE ) на софтуерни приложения за платформата .NET Framework. Използва се за разработка на конзолни и графични потребителски интерфейс приложения, както и Windows Forms или WPF приложения, уеб сайтове, уеб приложения и уеб услуги на всички поддържани платформи от Microsoft.

Visual Studio предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване (както за високо така и за машинно ниво), тестване на приложения, дизайн на потребителски интерфейс (форми, диалози, уеб страници, визуални контроли и други), моделиране на данни, моделиране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции. Могат да се добавят и разширения, които повишават функционалността на почти всяко ниво – включително добавянето на поддръжка за source-control системи, добавяне на нови инструменти като редактори и визуални дизайнери.

Visual Studio поддържа различни езици за програмиране, например C#, JavaScript, CSS и други и различни технологии за разработка на софтуер (ASP.NET, ADO.NET Entity Framework).

**3.3.13 Git** - децентрализирана система за контрол на версиите на файлове. Създадена е от Линус Торвалдс за управление на разработката на Linux. Поради нуждата да се контролира огромната база от код на Linux ядрото, основна цел при разработката на Git е била бързината. В момента Git се разработва от Джунио Хамано. Всяка локална Git директория е хранилище с пълна история и възможности за следене на версиите. Това прави Git независим от мрежови връзки към централен сървър. Git е свободен софтуер и се разпространява под GPL лиценз версия 2.

**3.4. Изводи**

За разработката на системата ще бъде нужно Visual Studio 2015, което е налично безплатно по програмата на Майкрософт за студенти (във версията си Enterprise), или в безплатната си, но по-бедна на възможности Community версия. Всички останали изтрументи и рамки за разработка за напълно безплатни с лиценз на отворен код.

Системата ще представлява SPA приложение на клиентската част, реализирано чрез AngularJS, и множество RESTFull уеб услуги реализирани чрез ASP .NET Web API.

**Глава 4. Анализ** (10-15стр.)

* 1. **Концептуален модел**

Уеб информационната систем е разделена на три основни части – публична, част за обиковенни потребители в системата и част за администратори (административен панел).

* + 1. **Публична част**

Публичната част от уеб системата представлява потребителски интерфейс, който позволява даден потребител да въведе своите потребителско име и парола, след което да бъде допуснат до потребителската част от системата.

**4.1.2 Потребителска част**

Потребителската част от системата се състои от пет под модула, а именно: Начална страница, Заявка/уведомление за отсъствие, моите заявки, заявки за одобрение, често задавани въпроси.

* **Начална страница –** съдържа обща информация за текущия потребител. На преден план е позициониран интерактивен календар, даващ възможност за разглеждане на потенциални дати за отсъствие. Освен това в календара са заредени, дни в които потребителят е отсъствал за текущата година, официални празници както и дни обявени за неработни за цялата компания. В дясно от календара, се намира панела мениджър за одобрение, който информира потребителя, кой е ръководителя, който трябва да одобри заявките му. Под панела с мениджър-а за одобрение, е поставен панел „Ваканционен статус“. Този панел дава информация за състоянието на използвани планени/не платени дни през текущата година, колко дни е отсъствал поради заболяване, дни трансферирани от предишна година, компенсации, както и текущ баланс.
* **Завка за отпуск/отсъствие –** съсдържа панели и функционалност за заяване на отсъствие. На преден план е позиционирана форма с четири полета: дата на започване, дата на край, тип на отсъствие, коментар и бутон за въвеждане. В дясно от полетата за дата на начало и край са разположение и малки бутони за показване на календар. Календарите показани от тези бутони са свързани с полетата. Например при избор за начало на отпус от 15 Май, за крайна дата не може да се избира по ранна дата от 15 Май същата година. В полето „Тип на отсъствие“, могат да се избират следните стойности: платен, неплатен, болест, брак, даряване на кръв, смърт на близък или друг. В дясно от този панел е позициониран интерактивен календар.
* **Моите заявки –** съдържа информация за вече направени завки. На преден план е позициониран панел с под елементи описващи направени заявки. Ако заявката е подадена но нито е одобрени нито отхвърлена, елемента даващ информация за нея е оцветен в синьо, а самата заявка е в статус „Подадена“. Ако е одобрена, то елементът съдържащ информация за нея е оцеветен в зелено, и тя е в статус „Одобрена“. При отхвърлена завка, цветът на елемента е червен, а статута е „Отхвърлена.“
* **Заявки за одобрение** – панел, който е активен само за определена група от потребители, а именно тези които имат права да одобряват заявки на други потребители. В тази част от системата, е добавена функционалност, чрез която потребителят да прегледа всички заявки към него и съответно да одобри или не одобре съответната заявка.
* **Често задавани въпроси** – секция съдържаща често задавани въпроси. Администратор на системата, може да добавя или редактира съществуващи въпроси.

**4.1.3 Административна част**

Административната част от системата съдържа в себе си три подсекции, позволяващи извършването на основни дейности управленски свързани със системата. Тук са разположени: „*Управление на почивни дни*“, „*Управление на потребителси*“ и „*Управление на заявки“*. Важно е да се отбележи, че този модул от уеб информационната система е единствено достъпен от потребителите с нужни права.

* **Управление на почивни дни** – този панел от системата позволява на администратор да добави официален почивен ден (празник или почивен за цялата компания). На преден план се визуализира интерактивен грид, който позволява добавянето на нов почивен ден или управлението на вече добавени такива. Тези дейности се извършват посредствум диалози, в които могат да се попълнят нужните данни.
* **Управление на потребители** – тази секция от уеб системата, има за цел да подпомогне цялостната дейност по управлението на потребители. Тук администратор може да регистрира, редактира или изтрива потребители. При дейностите регистриране и редактиране, интерактивен диалог се показва с възможност за промяна на данните на потребителя, включително и какви роли изпълнява в системата.
* **Управление на заявки –** този модул, съдържа в себе си функционалност за цялостните дейсности по управлението на заявки. Администратор на системата може да премахва или редактира заявки, като това отново става чрез интерактивен диалог.

**4.2. Потребителски (функционални) изисквания**

Функционалните изисквания следва да представят набор от услуги предоставяни от системата. В настоящата дипломна работа те са описани като потребителски изисквания.

**4.2.1 Права**

Новоизградената уеб информационна система ще съдържа множество потребители, всеки от които да притежа една или повече от предварително дефинираните роли. Входът към системата се осъществява, чрез уеб браузър на предварително известен адрес. Потребителите получават достъп до системата, след като въведат своите потребителско име и парола. Базата данни на системата, не съдържа никаква чувствителна информация за потребителите като ЕГН, личен телефон или адрес. Съществуват потребители с роли, които имат контрол над други потребители в системата.

**4.2.2 Роли**

По предварително изготвени изисквания, системата ще поддържа три типа потребители – обиковен потребител, потребител с права за одобрение на заявки и администратори. Тези потребители изпълняват така наречените роли. Важна характерискика на системата, е че независимо с какви свойства е създаден даден потребител, например каква е ролята която заема, в по-късен етап, това може да се редактира от администратор. Следва и по-подробното описание на ролите в системата:

* **Потребител** – това е роля, с единствена възможност за подаване на заявка за отсъствие.
* **Потребител одобрител** – това е потребител, който освен, че може да заявява, може и да одобрява заявките на други потребители.
* **Администратор** – това е ролята с най-големи права в системата. Освен да заявява отсъствие, администраторите ще могат да управляват други потребители, официални почивни дни както и всички заявки. Последното е полезно, в случай на заявка, която трябва да бъде одобрена или неодобрена, но одобряващия потребител отсъства. В този случай, администратор може служебно да одобри или неодобри дадена заявка.

**4.2.3 Състоявния**

* **Потребителски състояния –** най-общо потребителите ще могат да бъдат в две състояния: намиращ се в системата и намиращ се извън системата.
  + **Намиращ се в системата –** в това състояние, потребителят ще попада след като въведе своите потребителко име и парола. След влизането в системата, потребителят може да извършва действия, които са позволени от ролята от която изпълнява.
  + **Намиращ се извън системата –** в това състояние, потребителят попада по три причини. Първата, е когато навигира до страницата за вход в системата, втората, е когато изрично напусне системата след като е бил в нея (посредстум потребителския интерфейс за изход) и третия начин е неактивност в системата от 20 минути. Както ще разгледаме по-късно, системата ще има вграден механизъм за отчитане на потребителска дейсност. Така, ако даден потребител, независимо от неговата роля, не е активен в продължение на 20 минути, системата автоматично изкарва потребителя и го пренасочва към страницата за вход.
* **Състояния за заявка –** заявките в системата ще имат следните състояния – подадена, одобрена и отхвърлена:
  + **Подадена заявка –** това е състоянието, в което попада всяка заявка в момента, в който тя бъде попълнена и подадена от потребителя. След подаването й, тя попада в списъка на подадени заявки на потребителя, който я е създал. Освен това, тя попада и в списъка на завки за одобрение на всички потребители с роля на одобрител, които са одобрители на подателя.
  + **Одобрена заявка –** след като дадена заявка бъде одобрена, от всички потребители одобрители, тя попада в техните списъци на одобрени заявки, както и в списъка на потребителя подал заявката. За прегледност и за по-ясен потребителски интерфей, заявките които са одобрени биват оцветени със зелен цвят. Важно е да се отбележи, че има типове заявки, които се одобряват автоматично от системата. Такива например са: болест, брак, смърт на роднина и майчинство.
  + **Отхрълена заявка –** всяка отхвърлена заявка, попада в списъците на вече направени заявки. За яснота и прегледност, от гледна точка на потребителският интерфейс, отхвърлените заявки са оцветени в червено.

**4.2.4 Изисквания към заявки**

* **Валидирането на заявка –** съществена част от частта със създаването на заявка ще е нейното валидиране. Както на клиентската така и на сървърната част ще са изградени възможности за моменталното валидиране на новосъздаващата се заявка и навременното уведомяване на потребителя.
* **Изпечатване на заявка –** една от най-важните функционалности за дадена заявка, след като бъде одобрена, е тя да може принтирана по шаблон от системата, тъй като в редица страни законодателството изисква, архивиране и поддържане на цялата тази документация. По този начин се улеснява работата, както на потребителя, така и на служителите от отдел Човешки ресурси. Важна характеристика на отпечатването на заявки, е че те могат да бъдат отпечатване не само от потребителя направил дадена заявко, но и от администратор.

**4.2.5 Допълнителни изисквания към системата**

* **Автоматично извеждане на потребителя след неактивност –** една от важните функционалности свързана със сигурността на потребителите, е тяхното автоматично извеждане от системата, след определен перион на неактивност. След извеждането им, тяхната сесия става невалидна и затова, те трябва отново да въведат своите потребителско име и парола, за да влязат в системата.
* **При възникване на грешка в системата показването на прозорец за грешки –** в уеб базиратана система ще е изграден механизъм, за показване на специален екран на потребителя при възникване на грешка. Независимо, дали грешката е на клиента или на сървъра, тя се записва в специален файл за грешки, който в последтвие може да се разгледа от администратор, който да съдейства на разработчиците за поправянето й.

**4.3. Качествени (нефункционални) изисквания**

**4.3.1 Доспътност**

Основно изискване на уеб базираната информационна система, ще е тя да бъде достъпна през всички браузъри – от настолни машини до мобилни телефони. Това разбира се, включва специално пренареждане на модулите в зависимост от вида на браузъра и размера на екрана на потребителя. Структурата и дизайна на уеб системата трябва да имат следните характеристики: добре разположени информационни панели, консистентност измежду различни браузъри, изчистен от ненужни банери и реклами.

**4.3.2 Архитектура**

Системната архитектура ще е изградена на две основни и взаимозаменяеми части: клиентска и сървърна. На клиентската част ще се използва така наречения MV\* архитектурен шаблон. От страна на сървъра ще са имплементиани множество RESTFull услуги, които ще могат да бъдат извикани от много на брой и различни клиенти. По този начин, клиента и сървъра ще са напълно отделени един от друг.

Идеята за разделеност ще е развита допълнително в клиентската и сървърната част. Двата компонента вътрешно ще трябва са разделени на допълнителни слоеве и посредство инжектиране на зависимости, значително ще се улеснява подмяната или редактирането на които и да е слой.

**4.3.3 Преносимост**

Системата ще изградена по начин, по който тя лесно да бъде използвана на различни компютри. При клиентската част, проблема ще е решен доста елегантно. Компютъра на клиена трябва да разполага единствено с уеб браузър, който да е произведен след 2008 година, като операционната система няма никакво значение каква е. При сървъра, преносимостта също не би създавала много проблеми. Като изискване е поставен Windows 7 / Windows Server 2008 и по-високи версии. В момента на писането на дипломната теза, Майкрософт подготват и портавивна версия на .NET, ASP.NET Web API и Entity Framework, известно още с наименованията си .NET Core, ASP.NET Core и Entity Framework Core. Целта е, тези технологии да бъдат налични и на други операционни системи, като например Линукс и MacOS. След завършването на тези портативвни версии, миграцията към други операционни системи на разработената уеб система ще бъде значително улеснено.

**4.3.4 Скалируемост**

Скалируемостта е друга характеристика, която ще налична в системата. Тя ще трябва да е е разработена по такъв начин, че добавянето на допълнителни уеб сървъри и балансъри да е възможно и напрактика тривиално. По този начин значително ще се подобри работната й сила при големи натоварености.

**4.3.5 Сигурност на системата**

Системата ще бъде подсигурена със задължително вписване на всички потребители. Преди даден потребител да може да използва системата, той ще трябва да въведе своите име и парола. Ще трябва да се извърши Auth2 автентикация, след която потребителя ще получи токен, който ще е валиден за 2 часа. След изтичането на този токен, потребителя отново трябва да се впише в системата. Паролите на потребителите ще се хешират чрез подобрен SHA-1 алгоритъм.

**4.3.6 Производителност**

Разработената уеб информационна системата, ще трябва без пролемно да работи 24 часа в денонощието. Изискванията към уеб сървъра не са големи. Очакваният брой на потребителите също не е голям, тъй като системата ще се използва само в корпоративна среда.

**4.3.7 Поддръжка**

Кодът изграждащ системата, трябва да е добре структуриран и лесно четим. Вътрешните модули на клиентската и сървърната част трябва да са добре обособени в собствени слоеве с добре дефинирани интерфейси. Архитектурата ясно разграничава презентацията, логиката и данните. Всички тези свойства, ще доведът до лесната поддръжка на системата.

**4.3.8 Тестваемост**

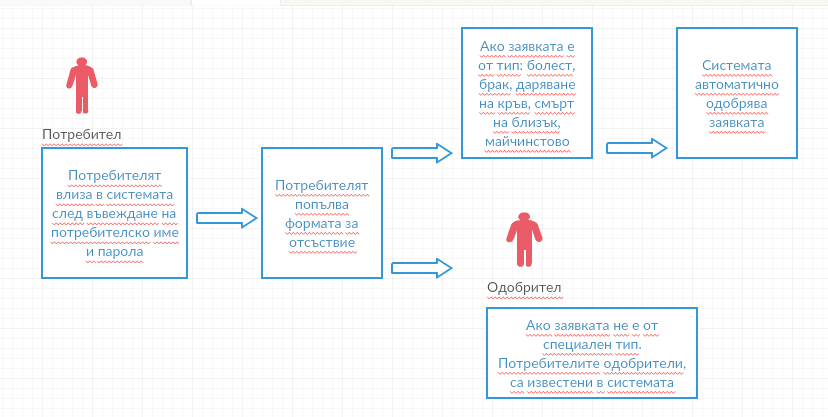
Системата трябва да бъде реализирана така, че да услеснява както тестването на ниво единица, така и автоматизираното тестване от край до край. Това важи както за клиентската, така и за сървърната част от системата.

**4.4. Работни (бизнес) процеси**

**4.4.1 Подаване на заявка за отсъствие**

Първоначално, потребителят е нужно да влезе в системата, използвайки, своите потребителско име и парола. Следва попълването на формата за заявка за отсъствие. В зависимост от типа на заявката, може има най-общо два варианта за одобрение. Ако заявката е от специален тип, т.е. е за болест, брак, смърт на близък, майчинство то заявката се одобрява автоматично системата и не е нужно по-висш служител да се намеси. В останалите случаи, заявката се приема от системата и съответния служител одобрител се уведомява.

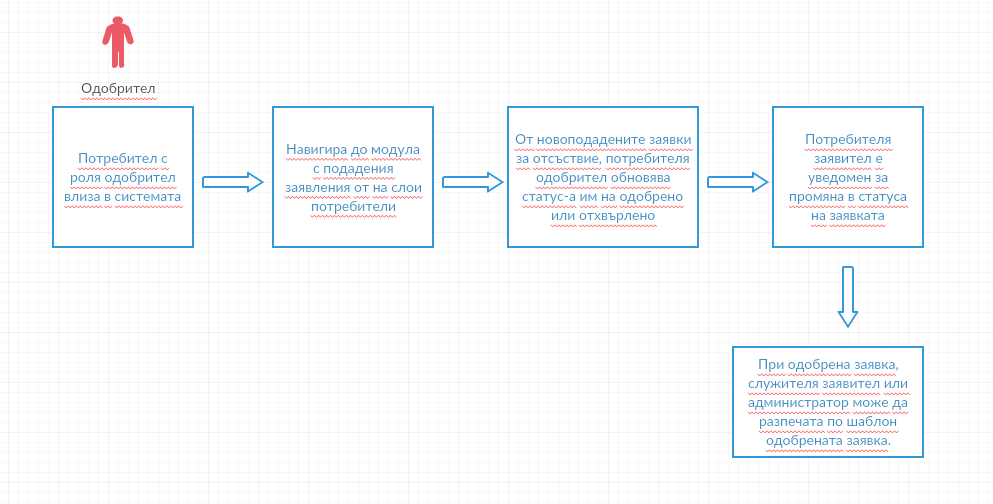
На фигура 9, са показани стъпките, през които трябва да се премине за попълването и подаването на заявка.



**Фигура 9**. Подаване на молба за отсъсвие.

**4.4.2 Одобряване на заявка**

При бизнес процеса за одобряването на заявка за отсъствие, е необходимо, потребителя одобрител, да взлезе в системата. Следва да се избере секцията заявки за одобрение. Последната стъпка е да промени състиянията на заявките направени от неговите служители. Независимо какво е новото състояние, потребителите инициатори на заявките са уведомени. На фигура 10. е показан бизнес процеса графично.



**Фигура 10**. Бизнес процес при одобряване на заявка.

TODO – ако трябва пълнеж могат да се добавят процеси при администриране сайта на диаграмите е <https://creately.com>.

**4.5. Изводи**

Отизвършеният задълбочен анализ, става ясно, че има ясно разграничаване на потребителите, чрез техните роли. Освен това потребителите имат достъп до системата единствено след като представят своите потребителско име и парола. В зависимост от ролята си, служителите могат да заявяват отсъствие, одобряват заявки или администрират цялостната система.

Разглеждайки и анализирайки нефункционалните изисквания, става ясно, че системата работи надеждно и сигурно. Освен това, тя е лесто преносима, лесно променяема, скалируемта и тестваема.

**Глава 5. Проектиране**

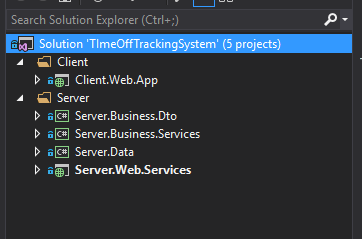
**5.1. Обща архитектура – напр. слоеве, модули, блокове, компоненти...**

Както вече заключихме в изложението, архитектурата на системата се състои от два основни компонента – клиент и сървър. Клиентската част следва модела MV\*, който е реализиран с помощта на AngularJS. Този архитектурен шаблон, позволява ясно разграничаване на отделните модули, като презентационен, бизнес логика и данни.

При сървъра, архитектурата доста наподобява клиентската, с едно изключение, а именно презентационния слой. Сървърът е реализиран като множество RESTFull услуги, изградени с помощта на ASP.NET Web API 2. По този начин, се постига, максимално разграничаване на клиента и сървъра. Те си комуникатар посредством данни в JSON формат. По този начин, клиента не се интересува от конкретната имплементация на сървъра стига, формата на JSON данните да не се променя. Още повече, сървърът, няма ограничения, кой клиент изпълнява услугите му. Това могат да бъдат, традиционни уеб проложения, настолни или мобилни приложения.

За пълнота, ще отбележим, че в сървърната част на система, е изградено още едно уеб проложение. То е помощно за разработчиците и показва помощна страница с всички публични RESTFull услуги. Архитектурата е стандарт MVC и е изградена с помощта на ASP.NET MVC 5.

На фигура 11, е показана структурата на системата:



**Фигура 11**. *Структура на системата.*

Както виждаме, системата е изградена от два основни проекта „*Client.Web.App*“ и „*Server.Web.Services*“ и три помощни, а именно „*Server.Business.Dto*“, „*Server.Business.Services*“ и „*Server.Data*“.

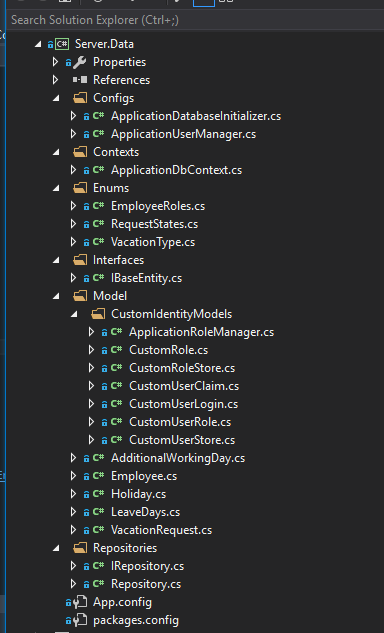
Трите помощни проекта, съдържат общи функционалности, които могат да бъдат преизползвани в бъдеще при развитието на проекта.

**5.2. Модел на данните**

**5.2.1 Моделни класове**

Проектът „*Server.Data*“ съдържа в себе си моделът на данните с които системата ще работи. В изградената система се използва така наречената „*Entity Framework* *Code First*“ стратегия. При нея е характерно, че форматът и структурата на данните се моделират посредствум създаването на обикновенни класове. В последтвие Entity Framework трансформира новосъздадените класове в таблици в базата от данни.

На фигура 12, е показано съдържанието на проекта:



**Фигура 12.** *Структура на Server.Data.*

В папаката „*Configs*“ са поставени два конфигурационни файла: *ApplicationDatabaseInitializer.*cs и *ApplicationUserManager.cs*. Първият файл, има за цел да настрои базата от данни при първо стартиране на системата, а вторият да конфигурира членството на потребители.

В папката „*Context*“ е поставен единсвен файл: *ApplicationDbContext.cs*. Той има за цел да настрои Entity Framework. В него се задават например, адрес на сървър с бази от данни, как Entity Framework ще представи класовете като таблици и др.

Папката „*Enums*“, съдържа изброени типове, които са използвани при дефинирането на свойствата на някои от класовете. EmployeeRoles съдържа възможните роли в системата, RequestStates състоянията на заявките, а VacationTypes какви типове заявка се поддържат.

Папката „*Interfaces*“ съдържа единсвен файл *IBaseEntity.cs*, който представлява C# интерфейс, който по конвенция, трябва да бъде имплементиран от всички моделни класове в системата.

В папката „Models“ се намират и основните моделни класове в системата. Основен клас за системата се явява класът *Employee* дефиниран в *Employee.cs.* Той има следната дефиниция:

public class Employee : IdentityUser<int, CustomUserLogin, CustomUserRole, CustomUserClaim>, IBaseEntity

{

#region Constructors

public Employee()

{

LeaveDays = new List<LeaveDays>();

LeaveDays.Add(new LeaveDays() { AllowedNonPaidDays = 60, EmployeeID = this.Id, AllowedPaidDays = 20, ForYear = DateTime.Now.Year });

}

#endregion

#region Public methods

public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<Employee, int> manager, string authenticationType)

{

// Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType

var userIdentity = await manager.CreateIdentityAsync(this, authenticationType);

// Add custom user claims here

return userIdentity;

}

#endregion

public virtual IList<VacationRequest> Requests { get; set; }

public virtual IList<LeaveDays> LeaveDays { get; set; }

public int? ManagerID { get; set; }

public virtual Employee Manager { get; set; }

}

Един от основните моменти в този клас е, че той описва таблица, която има релация, сочеща към самата нея. По този начин е осъществена връзката между служител и служителен ръководител.

Друг основен момент тук, е че класът *Employee.cs* разширява системния клас *IdentityUser.* Освен това *IdentityUser* е настроен да работи със специално дефинирани за приложението *CustomUserLogin*, *CustomUserRole*, *CustomUserClaim* класове, които точно отразаяват изискванията към системата за вход, роли и др. Други основни файлове в тази папка са: папката „*CustomIdentityModels“,* която съдържа всички специално създадени класове за работа с потребителски роли и др, класовете *Holiday*, *LeaveDays*, *VacationRequest*, *AddionalWorkingDays*. Всички те, описват бизнес изисквания, свързани с работата на системата.

Папката „*Repositories*“ съдържава в себе си, множество класове имплементиращи шаблона за дизайн *Repository.* Този шаблон се използва за разграничаването на логиката по извличането на данни и трансформирането й в моделни класове. Бизнес логиката не трябва да разбира от типа на данните, които са в основата на слоя за данни. Например, слоя за данни, може да бъде система за управления на релационни бази от данни или уеб услуга. Ето и дефиницията на *IRepository.cs* интерфейса, който по конвенция, трябва да бъде имплементирант от всички *Repository* класове:

public interface IRepository<TEntity> : IDisposable

{

/// <summary>

/// Creates a new empty entity.

/// </summary>

TEntity Create();

/// <summary>

/// Creates the existing entity.

/// </summary>

TEntity Create(TEntity entity);

/// <summary>

/// Updates the existing entity.

/// </summary>

TEntity Update(TEntity entity);

/// <summary>

/// Delete an entity using its primary key.

/// </summary>

void Delete(long id);

/// <summary>

/// Delete the given entity.

/// </summary>

void Delete(TEntity entity);

/// <summary>

/// Deletes the existing entity.

/// </summary>

void Delete(Expression<Func<TEntity, bool>> where);

/// <summary>

/// Finds one entity based on provided criteria.

/// </summary>

TEntity FindOne(Expression<Func<TEntity, bool>> where = null);

/// <summary>

/// Finds one entity based on its Identifier.

/// </summary>

TEntity FindById(long id);

/// <summary>

/// Finds entities based on provided criteria.

/// </summary>

IQueryable<TEntity> FindAll(Expression<Func<TEntity, bool>> where = null);

/// <summary>

/// Save any changes to the TContext

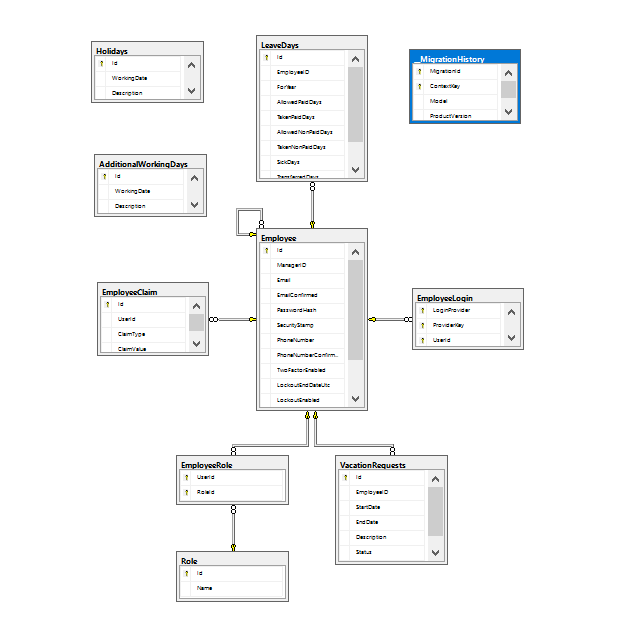
/// </summary>

bool SaveChanges();

}

**5.2.2 База от данни**

На фигура 13 е преставена структурата на новосъздадената база от данни.

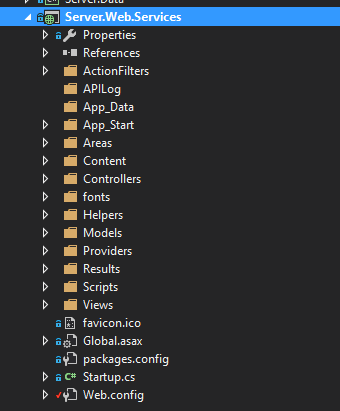


**Фигура 13.** *Структура на базата от данни.*

Базата от данни се създава при първоначалното стартиране на системата. Платформата Entity Framework, е конфигурарана по начин по който, ако не съществува база от данни да създаде. Ако има несъотверствие между намераната съществуваща бази от данни с текуще модел, то Entity Framework ще направи опит да обнови базата от данни.

**5.3. Проект съдържащ уеб услуги – извадки от кода**

Един от основните проекти в разработената система сървърния проект „*Server.Web.Services*“. Неговата структура е показана на фигура 14.



**Фигура 14**. *Структура на „Server.Web.Services“*

Основната част от фунцкионалностите в този проект се намират в директорията „*Controllers*“. В нея се намират следните контролери: *AccountController*, *EmployeeController*, *HolidayController*, *LeaveDaysController* и *VacationRequestController*.

Ще разгледаме съдържанието на един от основните контролери в приложението, а именно *EmployeeController*:

[Authorize]

[RoutePrefix("Employees")]

public class EmployeeController : ApiController

{

IEmployeeService employeeService;

public EmployeeController(IEmployeeService empService)

{

employeeService = empService;

}

[HttpGet]

[Route("")]

[Authorize(Roles = "admin")]

public IHttpActionResult GetAll()

{

var entities = employeeService.GetAll();

return Ok(entities);

}

[NonAction]

private EmployeeDto GetEmployeeById(int id)

{

EmployeeDto entity = employeeService.GetById(id);

return entity;

}

[HttpGet]

[Route("current")]

public IHttpActionResult GetCurrentUserInfo ()

{

int currentEmployeeId = this.User.Identity.GetUserId<int>();

EmployeeDto currentEmployee = this.GetEmployeeById(currentEmployeeId);

if (currentEmployee != null)

{

return Ok(currentEmployee);

}

else

{

return NotFound();

}

}

[HttpGet]

[Route("{id}")]

public IHttpActionResult GetById(int entityId)

{

EmployeeDto entity = employeeService.GetById(entityId);

if (entity != null)

{

return Ok(entity);

}

else

{

return NotFound();

}

}

[HttpPost]

[Route("")]

[Authorize(Roles = "admin")]

public IHttpActionResult CreateEntity([FromBody] EmployeeDto newEntity)

{

var createResult = employeeService.CreateEntity(newEntity);

return Ok(createResult);

}

[HttpPut]

[Route("")]

[Authorize(Roles = "admin")]

public IHttpActionResult UpdateEntity([FromBody] EmployeeDto newEntity)

{

bool updateResult = employeeService.UpdateEntity(newEntity);

if (updateResult)

{

return Ok();

}

return InternalServerError();

}

[HttpDelete]

[Route("{id}")]

[Authorize(Roles = "admin")]

public IHttpActionResult DeleteEntity(int id)

{

bool deleteResult = employeeService.DeleteEntityById(id);

if (deleteResult)

{

return Ok();

}

else

{

return InternalServerError();

}

}

}

От атрибута [Authorize] преди името на класа, става ясно, че тази уеб услуга е подсигурена само за потребители, които са представили името и паролата си.

Освен това класа *EmployeeController* разширява, системемния клас *ApiController*. Това е стандартна техника, за преизползване на код и получаване голямо количество вече съществуваща фунцкионалност на готово.

На първият ред от класа виждаме декларацията *IEmployeeService employeeService*. На следващия ред следва публичен конструктор, който приема като параметър обект имплементиращ *IЕmployeeServicе*. Това е стандартна техника за инжектиране на зависимости (на английки Dependency Injection). По този начин се постига много добро разделение на отговорности измежду модулите на системата. Това води до много по-лесното поддържане и тестване на системата. По-късно в текста, ще разгледаме как е реализирано инжектирането на зависимости с платформата Unity DI Container.

Следват няколко публични метода за извършване на т.нар. CRUD (Create, Read, Update, Delete) операции. Системата напълно спазва, добрите практири при реализирането на RESTFull услуги. Поддържат се GET метод, който е само за четене, не променя състоянието на системата. POST метод, който се използва, когато се създава нов ресурс. PUT метод за обновяване на ресурт и разбира се DELETE метод за изтриване на ресурс.

Както споменахме по-рано, всички контролери по конвенция съдържат поле от тип *IBaseBusinesService.* Което е в последствие инжектирано от платформата Unity DI Container. Следва представянето на интерфейса и една конкретна имплементация на него:

public interface IBaseBusinessService<TEntity>

{

IEnumerable<TEntity> GetAll();

TEntity GetById(int id);

int CreateEntity(TEntity newEntity);

bool UpdateEntity(TEntity newEntity);

bool DeleteEntityById(int id);

}

public class BaseBusinessService<TEntity, TEntityDto> : IDisposable, IBaseBusinessService<TEntityDto> where TEntity : class, IBaseEntity

{

protected IRepository<TEntity> entityRepository;

public BaseBusinessService()

{

entityRepository = UnityDIResolver.DefaultContainer.Resolve<IRepository<TEntity>>();

}

public BaseBusinessService(IRepository<TEntity> repository)

{

entityRepository = repository;

}

public virtual int CreateEntity(TEntityDto newEntity)

{

TEntity entityToCreate = Mapper.Map<TEntity>(newEntity);

TEntity createdEntity = entityRepository.Create(entityToCreate);

entityRepository.SaveChanges();

return createdEntity.Id;

}

public virtual bool DeleteEntityById(int id)

{

entityRepository.Delete(id);

bool deleteResult = entityRepository.SaveChanges();

return deleteResult;

}

public virtual IEnumerable<TEntityDto> GetAll()

{

var entities = entityRepository

.FindAll()

.ToList();

var resultEntities = Mapper.Map<List<TEntityDto>>(entities);

return resultEntities;

}

public virtual TEntityDto GetById(int id)

{

var entity = entityRepository.FindById(id);

var resultEntity = Mapper.Map<TEntityDto>(entity);

return resultEntity;

}

public virtual bool UpdateEntity(TEntityDto newEntity)

{

TEntity entityToUpdate = Mapper.Map<TEntity>(newEntity);

entityRepository.Update(entityToUpdate);

bool updateResult = entityRepository.SaveChanges();

return updateResult;

}

public void Dispose()

{

entityRepository.Dispose();

}

}

Както виждаме от общата имплементация, интерфейса се състои от основни CRUD операции, които са имплементирани от *BaseBusinessService.* Имплементацията е опционална, тъй като наследнитеците на *BaseBusinessService* могат да реализират свои имплеметации.

Интересно е да се разгледа и как се осъществява, инжектирането на зависимости в системата. Конфигурация се намира в проекта „*Server.Business.Services*“ в папката „*Configs*“ във файла *UnityDIResolver.cs*:

public class UnityDIResolver

{

private static IUnityContainer defaultContainer = null;

public static IUnityContainer DefaultContainer

{

get

{

if (defaultContainer == null)

{

defaultContainer = CreateUnityContainer();

}

return defaultContainer;

}

set

{

defaultContainer = value;

}

}

private static IUnityContainer CreateUnityContainer()

{

IUnityContainer container = new UnityContainer();

container.RegisterType<IEmployeeService, EmployeeService>(new HierarchicalLifetimeManager());

container.RegisterType<IHolidayService, HolidayService>(new HierarchicalLifetimeManager());

container.RegisterType<ILeaveDaysService, LeaveDaysService>(new HierarchicalLifetimeManager());

container.RegisterType<IVacationRequestService, VacationRequestService>(new HierarchicalLifetimeManager());

container.RegisterType<IAdditionalWorkingDaysService, AdditionalWorkingDaysService>(new HierarchicalLifetimeManager());

container.RegisterType<DbContext, ApplicationDbContext>();

container.RegisterType(typeof(IRepository<>), typeof(Repository<>));

container.RegisterType<UserManager<Employee, int>, ApplicationUserManager>();

container.RegisterType<IUserStore<Employee, int>, CustomUserStore>();

return container;

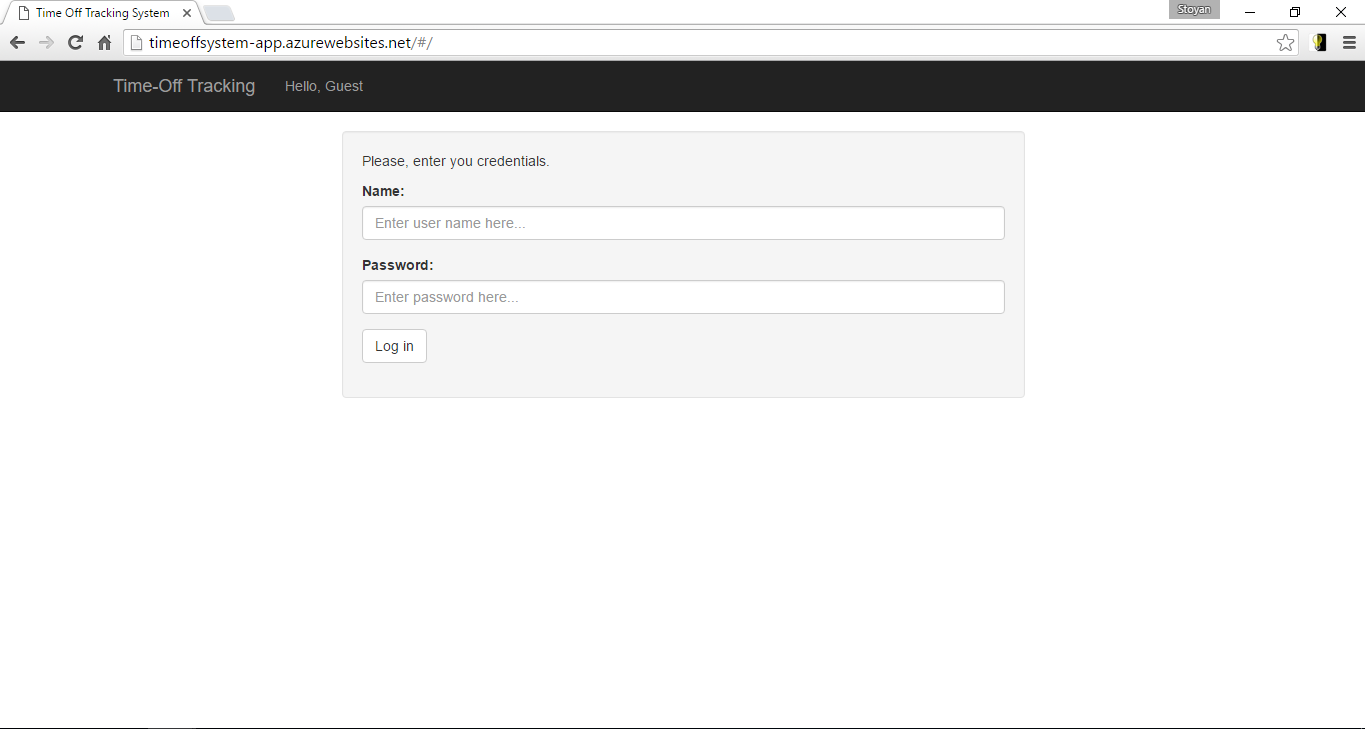
}

}

От показания код, виждаме, че инжектирането на зависимости е изключително лесно конфигуриуемо – единствено с код на C#. Това ни позволява да инжектираме фалшиви типове, когато не се нуждаем от реална работеща имплементация, например когато създаваме тестове на ниво единица.

**5.4. Потребителски интерфейс**

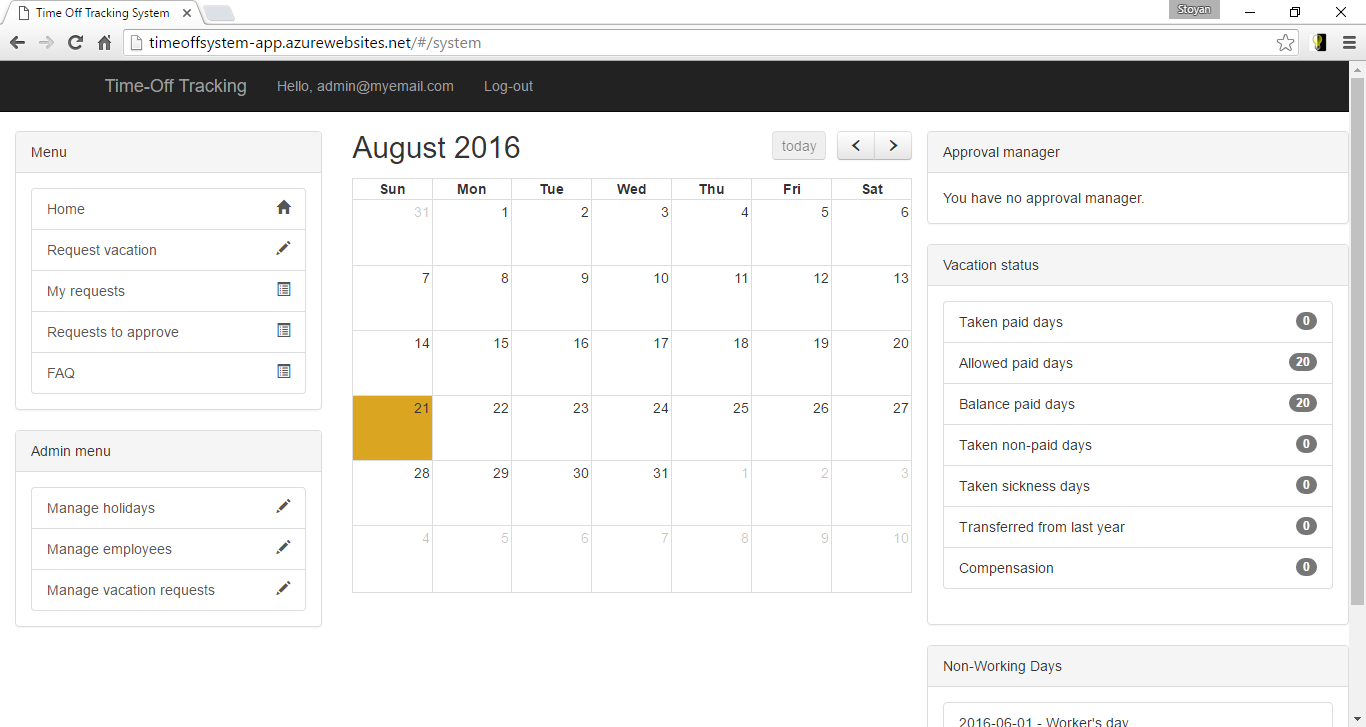
След като потребителя, стартира системата, се показва страница за въвеждане на потребителско име и парола. На фигура 15, е позан началният екран. Важно е да отбележим, че поради високите изисквания за сигурност, системата не поддържа потребителска сесия, от това следва, че потребителя трябва да въведе наново своето име и парола, при всяко стартиране на приложението.

****

**Фигура 15.** *Начален екран на системата.*

Както виждаме от началната страница, в системата няма публични части. Ако поребителя въведе грешно своите име и парола, ще получи съобщение за грешка. При последователни 5 грешни опита, потребителя ще бъде временно спрян. Това е превантивна мярка, срещу опити за получаване на неправомерен достъп до системата.

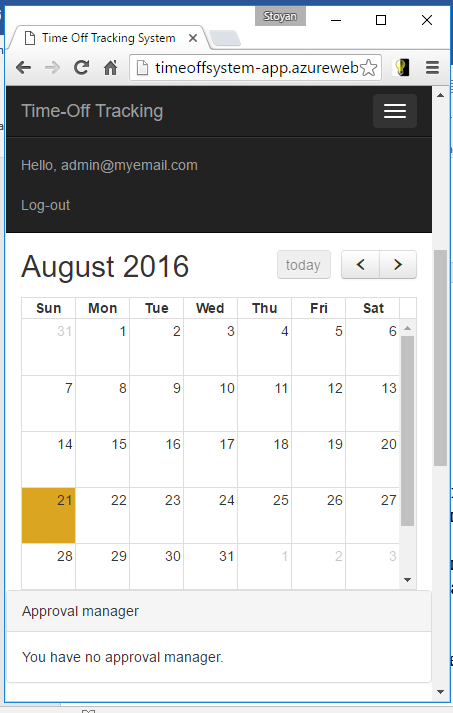
След успешно влизане в системата, потребителя е пренасочен към началната страница за потребители на системата. Изглед от нея може да се види на фигура 16.



**Фигура 16.** *Изглед към начална страница за регистрирани потребители.*

Началната страница за вписани потребители, преставлява съвкупност от информация за текущия потребител. В лявата част от са позиционирани потребителските панели с полезни връзки към функционалности отговарящи на съответните им роли. В централната част на страницата е изграден интерактивен календар, който има допълнителни възможности за визуализиране на официални неработни дни, отсъствия на текущия потребител и др. В дясната част на уеб страницата, служителя получава информация, кой е неговия одобрител на заявки, какъв е статуса за текущата година за отсъствия, както и официални почивни дни за годината.

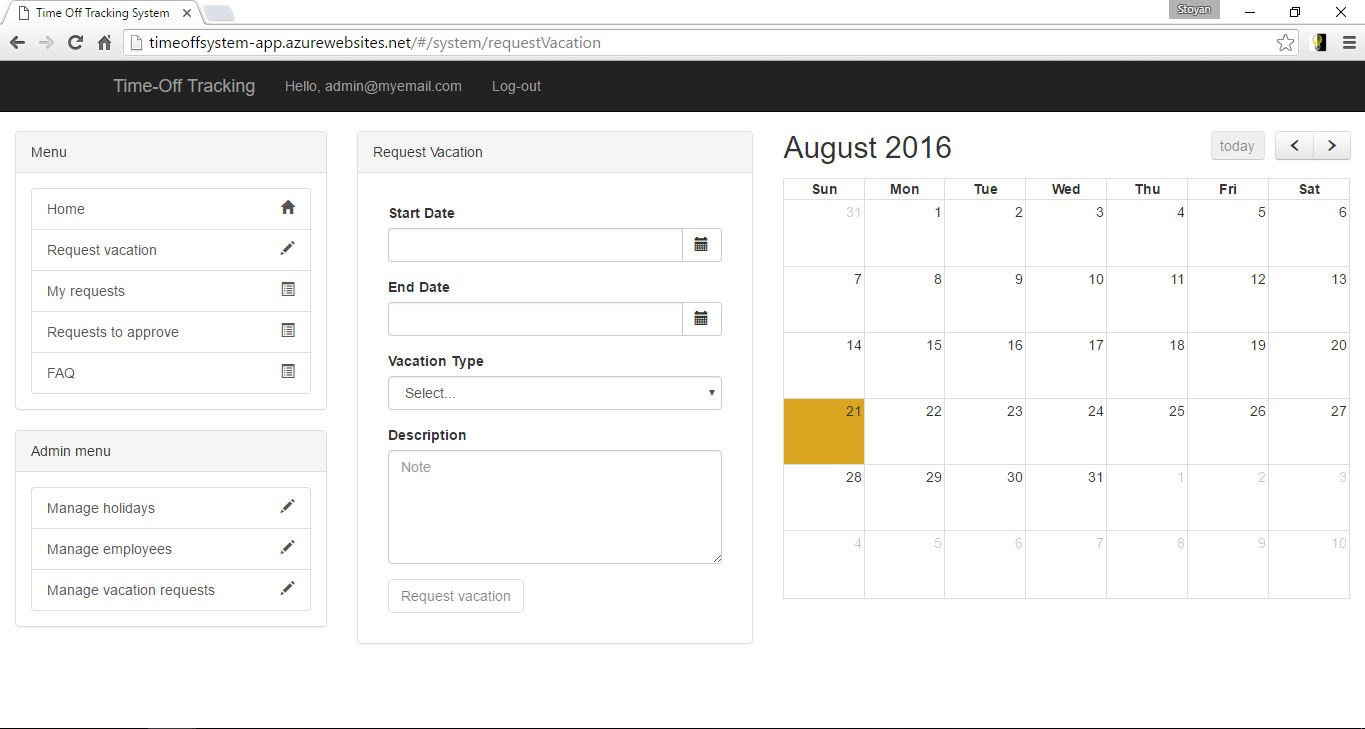
Както обсъдихме по-рано, системата трябва да поддържа всякакви размери на екрана на потребителя. Фигура 17, показва пример за пренареждане на елементите от приложението при по-малък размер на екрана.



**Фигура 17.** *Изглед към системата от екран с малък размер.*

Важно е да се отбележи, че когато служител, натисне логото на системата, то той е пренасочен към тази начална страница.

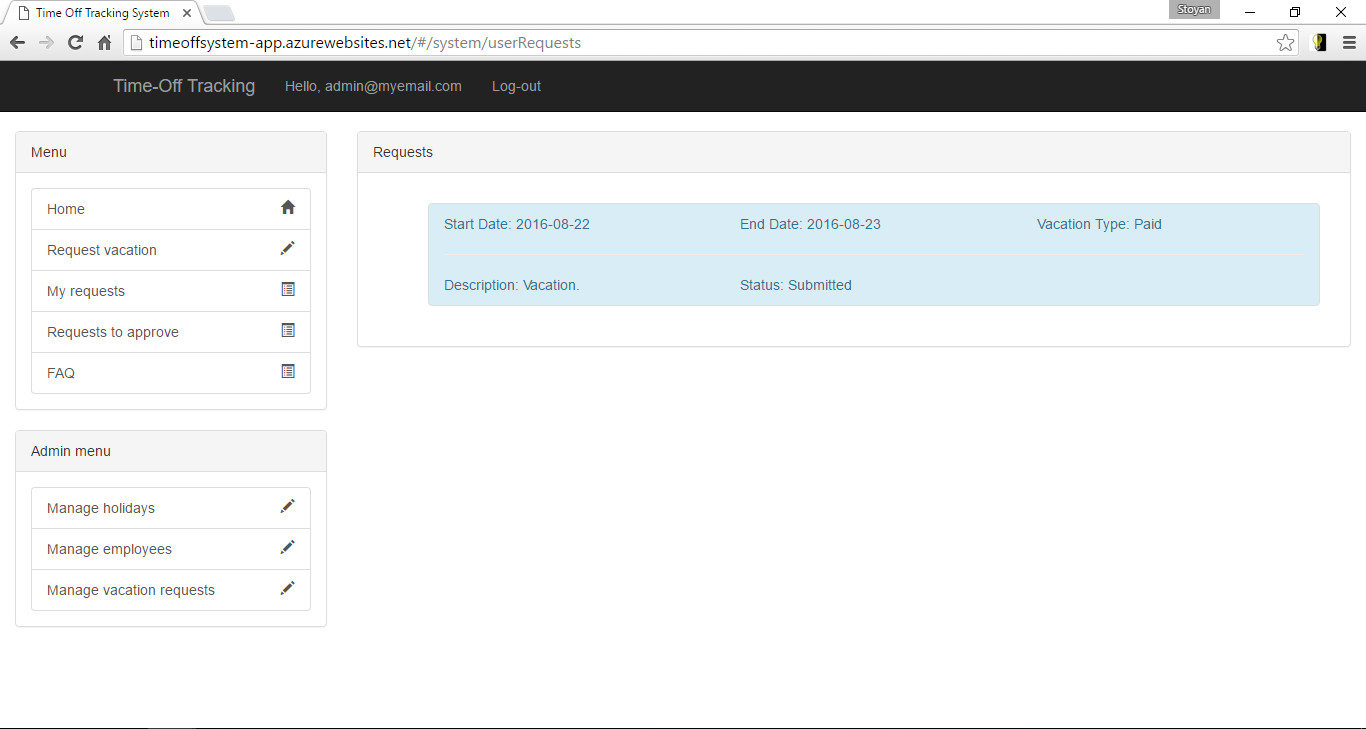
На фигура 18 е показан един от най-важните екрани от системата, а именно този за подаване на заявка за отсъствие.



**Фигура 18.** *Екран за заявка.*

Формата за попълване е стадартна и не изисква много данни. Както отбелязахме не веднъж, базирано на типа на заявката, тя може да бъде одобрена автоматично (например при болест, брак, даряване на кръв, брак) или да бъде пренасочена към потребител одобрител.

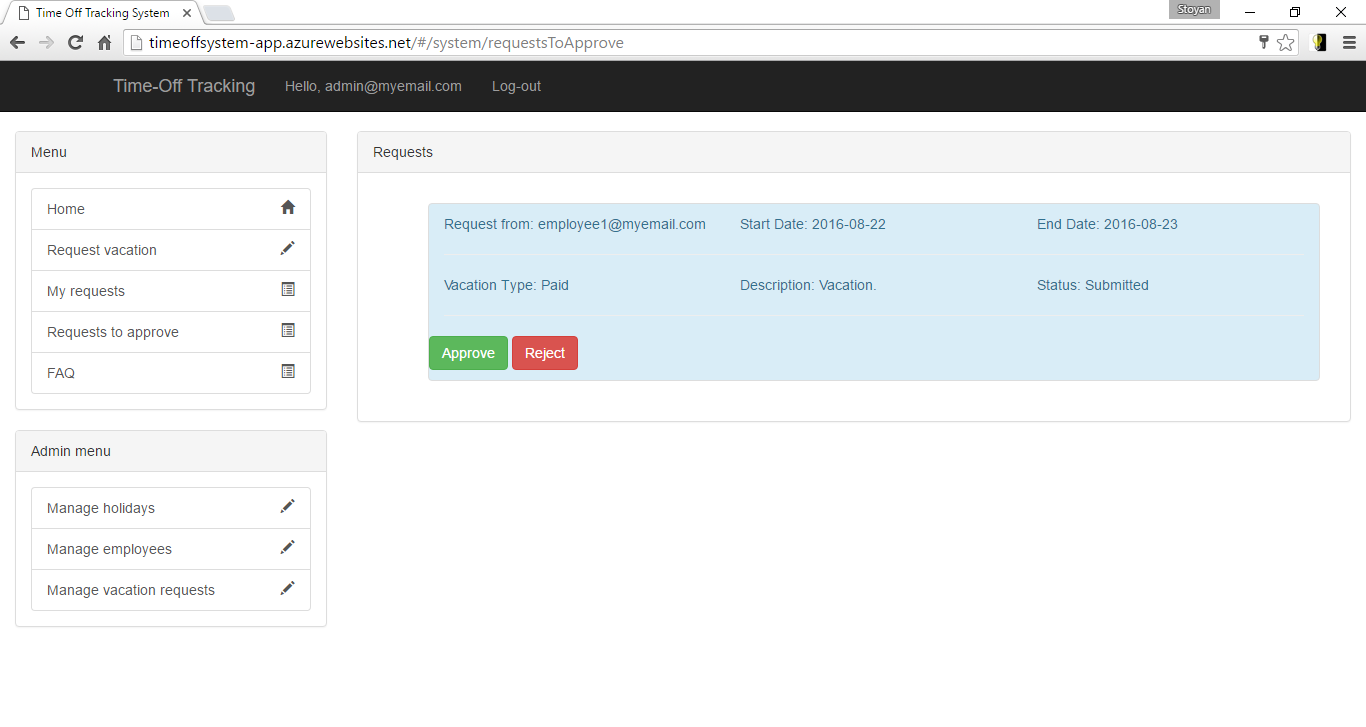
На фигура 19, е показан изглед към секцията „Моите заявки“.



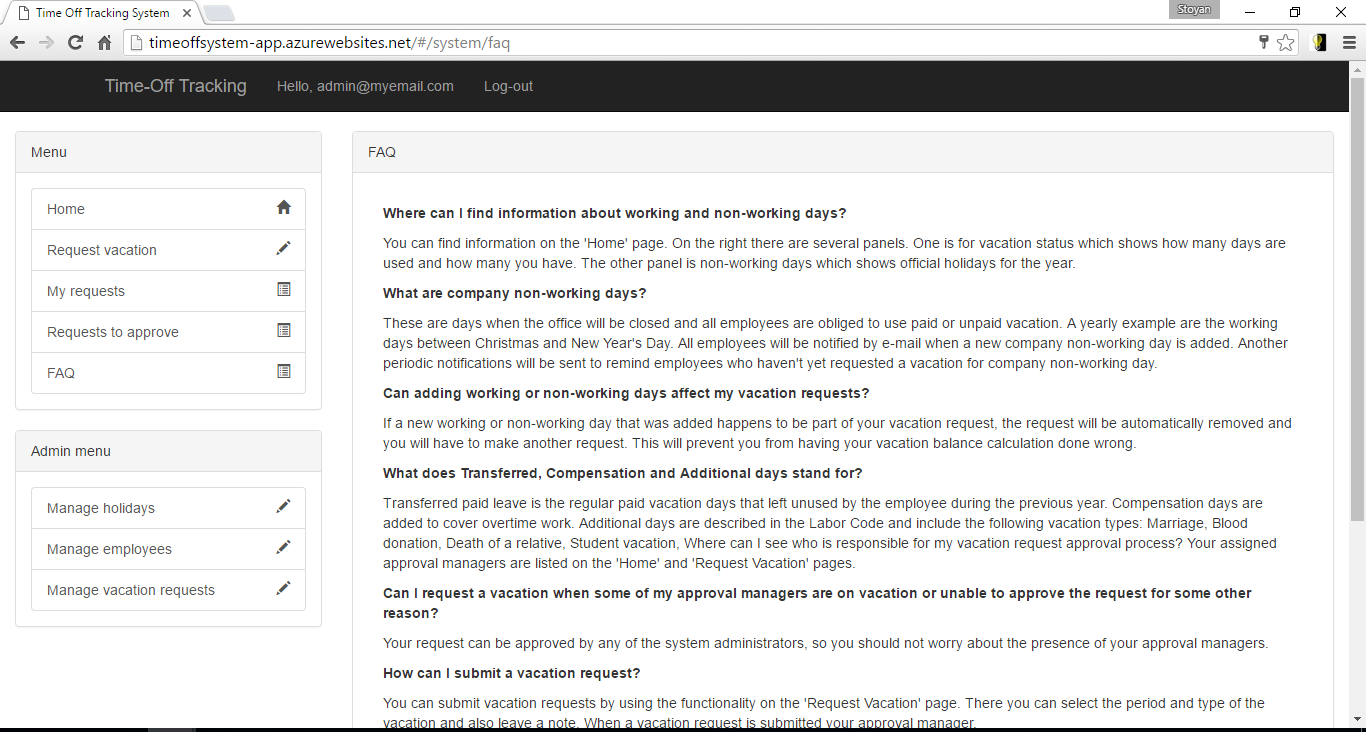
**Фигура 19.** *Секция „Моите заявки“.*

Базирано на това, какъв е статуса по одобрение, с цел по-ясен потребителски интерфейс, различните типове статуси се оцветяват в различен цвят.

На фигура 20 е показана страницата „Заявки за одобрение“. Както се вижда, потребителският интерфейс, позволява съответното одобрение или отхвърляне на дадена заявка.

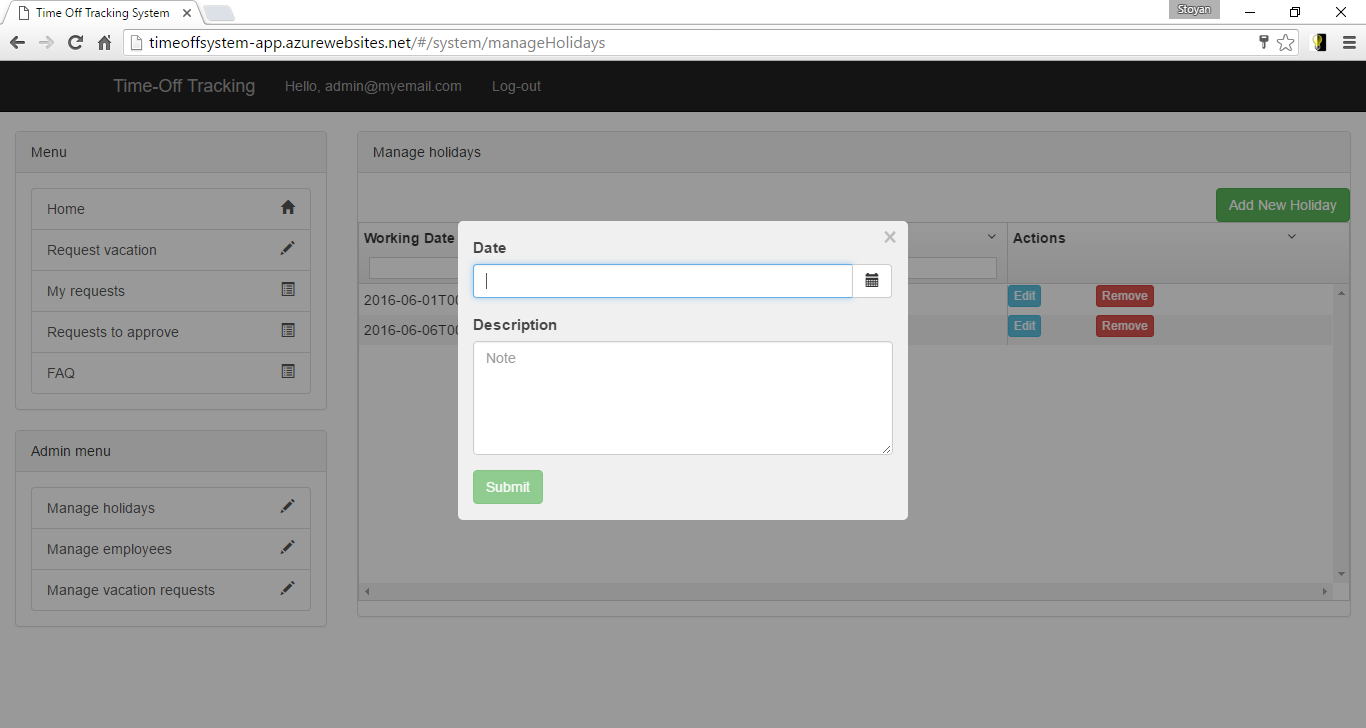
**Фигура 20.** *Заявки за одобрение.*

Последната страница, която е видима за обикновенните потребители е страницата за често задавани въпроси. На фигура 21 можем да видим и изглед от нея.



Фигура 21. Изглед към страницата „Често задавани въпроси“

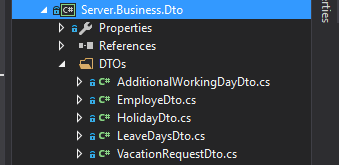
Потребителите с роли „Администратор“ имат достъп до административния панел. Там системата позволява извършването на дейности свързани с управлението на почивни дни, потребители, заявки за отсъствия. Фигура 22, показва изглед към секцията „Управление на почивни дни“.



**Фигура 22.** *Изглед към страница „Управление на почивни дни“.*

**5.5. Ресурсни и спомагателни модули**

Един от важнитете спомагателни модули в системата е проекта „Server.Business.Dto“. На фигура 23 е показан изглед от неговата структура.



**Фигура 23.** *Структура на „Server.Business.Dto“.*

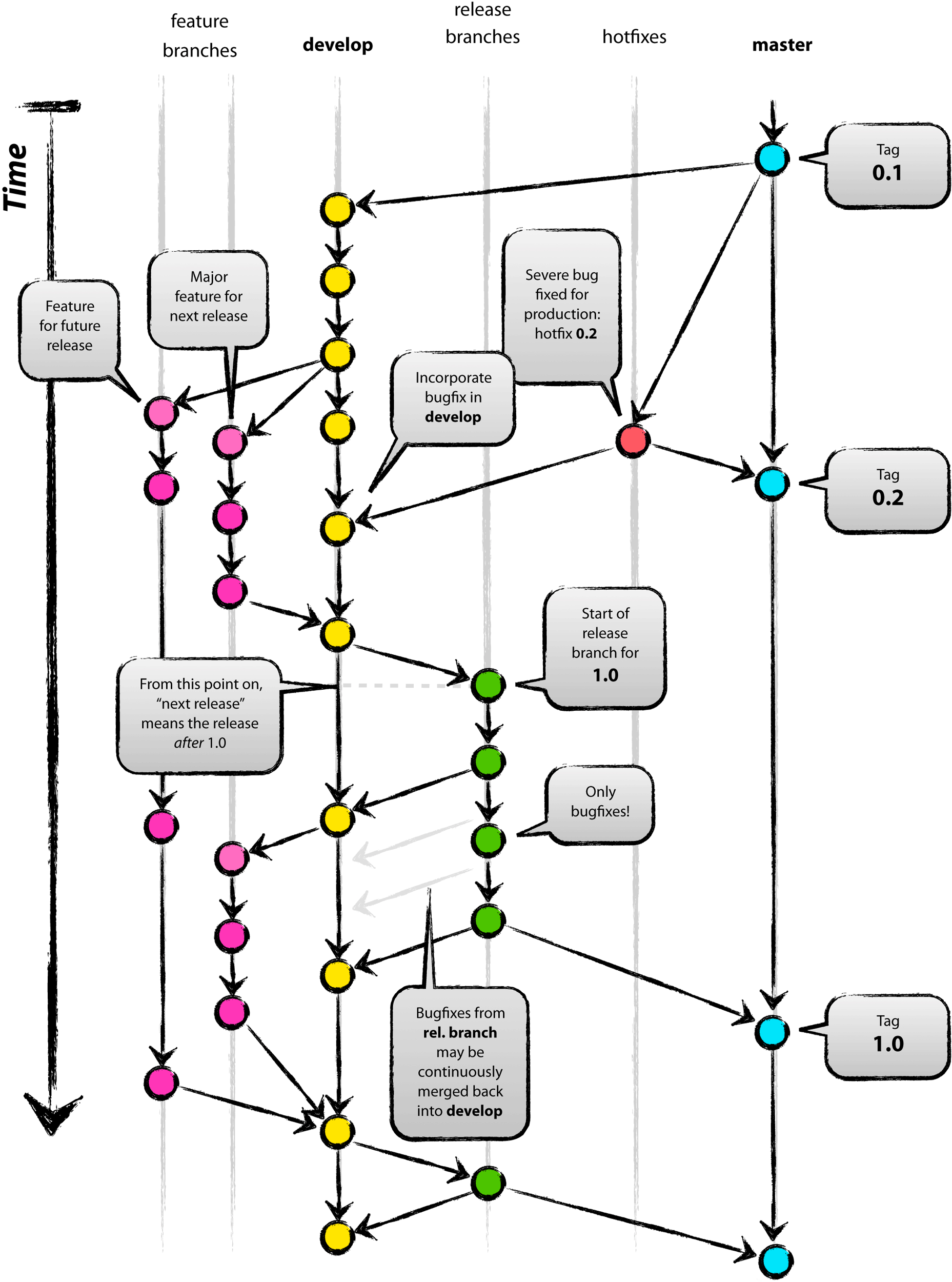
В този спомагателен проект, се имплементира шаблона DTO (Data transfer object), който се явява изключително полезен при комуникацията между клиент и сървър. Едино от основните предимства на този шаблон, е че комуникацията между клиента и сървъра се оптимизира максимално, като те разменят съобщения само с важните за тях данни.

**Глава 6. Реализация, тестване/експерименти и (евентуално) внедряване** (10-15стр.)

* 1. **Реализация на модулите**

При реализацията на модулите максимално сме се придържали към предварително дефинираните изисквания, както за публичната част, така и за административния модул. За изграждането на системата, беше използвана интегрираната среда за разработка *Visual Studio 2015*. По време на реализацията на системата, беше използвана системата за управление на изходен код – *Git*, а самият изходен код се съхранява в публично хранилище в платформата *GitHub*.

С цел по-добра организация на хранилището и по-голяма ефективност при създаването на изходният код, беше използван модела *GitFlow*[23]. На фигура 24, може да се види и визуално, какво представлява модела.



**Фигура 24.** *Моделът GitFlow.*

Едно от основните предимства на модела *GitFlow,* е че при него хранилището се разделя на няколко основни части. Най-общо съществуват два клона – *master* и *develop*. Master клона, съдържа изходен код, който е готов за доставяне, докато develop, съдържа код, който е току що завършен.

Съществуват още три клона, а именно *feature*, *release*, *hotfix*. Всяка нова функционалност се разработва в нов клон feature/име-на-функционалност. Важно е да се отбележи, че началото на всеки нов feature клон е от клона develop. След завършването на дадена функционалност, съответния клон се слива с develop. По този начин, работата по проекта се модулизира и всяка нова функционално се отделя в нов клон. Така, много лесно може да се проследи от кога, къде и кой е работил по дадена функционалност. За по-лесното управление доставянето на нова версия на системата (релийз) се използва realease/номер-на-релийз клона.

Този клон, задължително трябва да започне от master клона. Ако се намери дефект, в някой от release клоновете, се създава hotfix/име-на-дефект клон. Той задължително започва от release клон. След отстраняването на дефекта, hotfix клон-а се слива със съответния release клон, който от своя страна се слива с master клона.

Клиента и сървъра са отделени максимално един от друг. Поради тази причина, клиента трябва сам, да управлява голяма част от дейностите свързани с използването на системата. Например, когато даден потребител влезе в системата, сървърът изпраща информация, каква е неговата роля. Клиента има отговорността да осигури достъп до съответните секции. Следващата извадка от изходния код, демонстрира как е реализирано това:

app.config(['$stateProvider', '$urlRouterProvider', 'USER\_ROLES',

function ($stateProvider, $urlRouterProvider, USER\_ROLES) {

// For any unmatched url, redirect to /

$urlRouterProvider.otherwise("/");

// Now set up the states

$stateProvider

.state('login', {

url: '/',

templateUrl: 'app/views/login.html',

controller: 'LoginController',

data: {

authorizedRoles: [USER\_ROLES.guest]

}

})

.state('system', {

url: '/system',

templateUrl: 'app/views/system.html',

controller: 'SystemController',

data: {

authorizedRoles: [USER\_ROLES.admin, USER\_ROLES.regular]

}

})

.state('admin', {

url: '/admin',

templateUrl: 'app/views/admin.html',

data: {

authorizedRoles: [USER\_ROLES.admin]

}

})

.state('requestVacation', {

parent: 'system',

url: '/requestVacation',

templateUrl: 'app/views/requestVacation.html',

controller: 'RequestVacationController',

data: {

authorizedRoles: [USER\_ROLES.admin, USER\_ROLES.regular]

}

})

.state('userRequests', {

parent: 'system',

url: '/userRequests',

templateUrl: 'app/views/userRequests.html',

controller: 'UserRequestsController',

data: {

authorizedRoles: [USER\_ROLES.admin, USER\_ROLES.regular]

}

})

.state('requestsToApprove', {

parent: 'system',

url: '/requestsToApprove',

templateUrl: 'app/views/requestsToApprove.html',

controller: 'RequestApprovingController',

data: {

authorizedRoles: [USER\_ROLES.admin, USER\_ROLES.regular]

}

})

.state('manageHolidays', {

parent: 'system',

url: '/manageHolidays',

templateUrl: 'app/views/manageHolidays.html',

controller: 'HolidayController',

data: {

authorizedRoles: [USER\_ROLES.admin]

}

})

.state('error', {

parent: 'system',

url: '/error',

templateUrl: 'app/views/error.html',

data: {

authorizedRoles: [USER\_ROLES.admin, USER\_ROLES.regular]

}

})

}]);

От извадката, става ясно, че полето *authorizedRoles* е ключово за достъп до секция в системата.

Същинската част от системата, е реализирана с помощта на библиотеките jQuery, Bootstrap, Angular UI Bootstrap. Чрез тях, са изградени календарните компоненти, полета и други елементи, както и цялата валидация свързана с тях.

* 1. **Системна интеграция**

За използването на системата, в най-общ план са нужни следните изисквания: *Visual Studio 2015*, *Microsoft SQL Server 2008* или по-нов, *Internet Information Services 7.5* (IIS)или по-нов.

За интегрирането на системата с Microsoft SQL Server, е необходимo да се изполва валиден стринг за връзка (connection string) в конфигурационния файл (web.config) към работеща инстанция на Microsoft SQL Server. Важно е да се отбележи, че не е нужно да им локална инсталация на MS SQL Server.

Интегрирането на системата с IIS, може да стане по два начина: ръчно или чрез Visual Studio. Накратко с натискането на бутона “Publish” се осъществява връзка

между Visual Studio и уеб сървъра IIS, задължително криптирана, и информацията се предава на сървъра.

* 1. **Планиране на тестването**

За осигуряването на правилната работа на уеб системата е необходимо да бъдат изпълнени разгледаните по-долу тест сценарии, обхващащи основните функционалности, предоставени от системата. Тестовите сценарии имат за цел да потвърдят функционалните изисквания. За осъществяването на всички тестове трябва да се отиде на адрес [*http://timeoffsystem-app.azurewebsites.net*](http://timeoffsystem-app.azurewebsites.net).

Тестовите сценарий, ще бъдат структурирани в табличен вид, съдържащ следната информация: номер и име на сценарий, номер на стъпка, действие и очакван резултат.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 1 | Име на Сценарий: Вход в системата | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Отваряне на URL на системата. | Показва се начална страница за вход. |
| 2 | Попълване на валидни потребителско име и парола и натискане на бутона “Вход” | Показва се начална страниза за потребители с достъп. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 2 | Име на Сценарий: Изход от системата | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Уверете се, че сте влезли в системата. | Потребителят е вписан в системата. |
| 2 | Натиснете бутона „Изход“. | Показва се начална страница за вход. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 3 | Име на Сценарий: Заявка за платено отсъствие | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Заявка за отсъсвие“. | Показване на формата за заявка за отсъствие. |
| 2 | Попълнете начална и крайна дата, както и полето тип на отсъствие „Платено“. | Бутона за заявка променя състоянието си от неактивен към активен. |
| 3 | Натиснете бутона „Заявка“. | Показва се начална страница за влезни потребители. В секцията „Моите заявки“ е отразена ново направената заявка. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 4 | Име на Сценарий: Заявка за отсъсвие поради встъпване в брак. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Заявка за отсъсвие“. | Показване на формата за заявка за отсъствие. |
| 2 | Попълнете начална и крайна дата, както и полето тип на отсъствие „Встъпване в брак“. Тук трябва да се добави и коментар. | Бутона за заявка променя състоянието си от неактивен към активен. |
| 3 | Натиснете бутона „Заявка“. | Показва се начална страница за влезни потребители. В секцията „Моите заявки“ е отразена ново направената заявка. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 5 | Име на Сценарий: Принтиране на заявка за отпуск. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Моите заявки“. | Показва се списък с всички направени заявки до момента (независимо дали са одобрени, отхръвлени или в изчакване). |
| 2 | Натиснете бутона „Принтирай“ за съответната одобрена заявка. | Показва се диалогов прозорец за изпечатване на заявката. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 6 | Име на Сценарий: Одобряване/отхвърляне на заявка. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Заявки за одобрение“. | Показване на списък с заявките за одобрение. |
| 2 | За съответната заявка, изберете бутона одобри или отхвърли. | Показва се начална страница за вписани потребители и съобщение, че заявката е одобрена. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 7 | Име на Сценарий: Добаване на почивен ден. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Управление на почивни дни“. | Показва се таблица с вече добавени почивни дни. |
| 2 | Натиснете бутона „Добави почивен ден“. | Показва се диалог с полета описващи почивен ден. |
| 3 | Попълнете полетата в диалога и натиснете бутона „Добави“. | Диалогът за добавяне на почивни дни се скрива. В таблицата с почивни дни е добавен нововъведения почивен ден. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 8 | Име на Сценарий: Премахване на почивен ден. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Управление на почивни дни“. | Показва се таблица с вече добавени почивни дни. |
| 2 | Натиснете бутона „Премахни“ за съответния почивен ден. | Почивният ден е премахнат и вече не се показва в таблицата. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 9 | Име на Сценарий: Редакция на почивен ден. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Управление на почивни дни“. | Показва се таблица с вече добавени почивни дни. |
| 2 | Натиснете бутона „Редактирай“ за съответния почивен ден. | Показва се диалог с данните за съответния почивен ден. |
| 3 | Променете полетата за почивния ден и натиснете бутона „Запази“. | Диалогът за добавяне на почивни дни се скрива. В таблицата с почивни дни е променен съответния почивен ден. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 10 | Име на Сценарий: Добаване на потребител. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Управление на потребители“. | Показва се таблица с вече добавени потребители. |
| 2 | Натиснете бутона „Добави потребител“. | Показва се диалог с полета описващи даден потребител. |
| 3 | Попълнете полетата в диалога и натиснете бутона „Добави“. | Диалогът за добавяне на потребители се скрива. В таблицата с потребители е добавен нововъведения потребител. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 11 | Име на Сценарий: Редакция на потребител. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Управление на потребители“. | Показва се таблица с вече добавени потребители. |
| 2 | Натиснете бутона „Редакция“ за съответния потребител. | Показва се диалог с попълнени полета описващи даден потребител. |
| 3 | Редактирайте полета свързани с потребител и натиснете бутона „Запази“. | Диалогът за редакция на потребители се скрива. В таблицата с потребители е отразена промяната за съответният потребител. |

Таблица #.

|  |  |  |
| --- | --- | --- |
| Сценарий №: 12 | Име на Сценарий: Премахване на потребител. | |
| Стъпка №: | Действие | Очакван резултат |
| 1 | Влезте в системата и навигирайте до секцията „Управление на потребители“. | Показва се таблица с вече добавени потребители. |
| 2 | Натиснете бутона „Премахване“ за съответния потребител. | Потребителят е премахнат и вече не се показва в таблицата. |

Таблица #.

Изброените тестови сценарии могат да се изпълняват автоматично чрез платформата *Protractor*, което позволява много кратно изпълнение на едни и същи тестове с цел регресионно тестване. Всеки от тестовите сценарии може да бъде включен в тестова сюита заедно с други сценарии с цел тестване на специфична функционалност. Друго предимство на платформата, е че позволява едни и същи тестове да се изпълняват под различни браузъри и операционни системи.

* 1. **Модулно и системно тестване**

Освен описаните тестови сценарии е предивено и тестване от отделните функционалности при разработването и преди сливенето на съотвения feature клон към develop клона. За отделен модул се тества функционалността, която той предоставя, потокът на данни и потребителския интерфейс свързан с този модул.

Освен това, след завършването на тестването на отделните функционалности от разработчика и тестовите сценарий се извършват и тестове за натовареност на системата. Инструмента, койтo може да се използва е jMeter [24].

* 1. **Експериментално внедряване**

Експерименталното внедряване е осъществено, чрез облачната платформа Microsoft Azure.

**6.5.1 Microsoft Azure**



**Фигура 25.** *Лого на Microsoft Azure.*

Както се спомена по-рано в този текст, Microsoft Azure е платформа за облачни и инфраструктурни изчисления създадена от Майкрософт през 2010, позволяваща за създаването, доставянено и управлението на приложения и услуги чрез световна мрежа, управлявана в големи изчислителни центрове на Майкрософт.

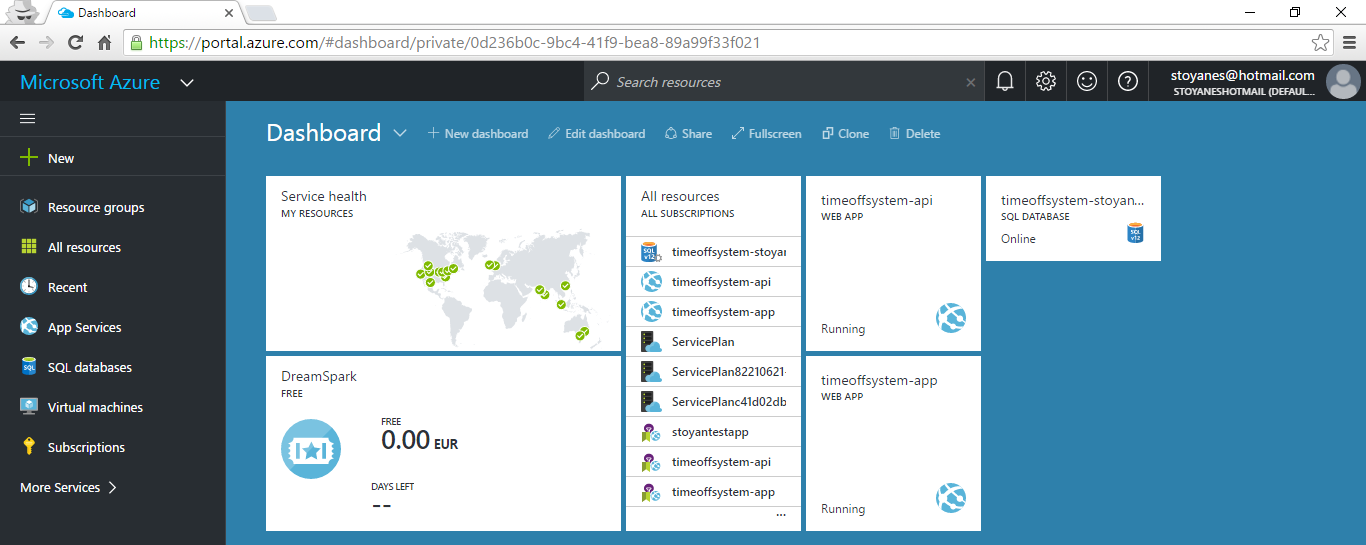
Предоставят всички модерни услуги на пазара, като PaaS (платформа като услуга), IaaS (инфраструкура като услуга), SaaS (софтуер като услуга) и др.

Едно от най-големите предимства на облачната платформа, е че доставянето на готовият код става изключително лесно. Не са нужни сложни процеси и системи за да се осъществи това, въпреки че при нужда те лесно могат да се интегрират с платформата.

Microsoft Azure е напълно безплатен за използване от студенти за учебни цели. Единственото изискване, е студента да направи регистрация в системата DreamSpark, след което се получава достъп до платформата. Въпреки, че не се предоставят всички възможни услуги в безплатната версия, налични са до пет уеб сайта, уеб услуги и бази от данни, които са напълно достатъчни за учебни цели.

**6.3.2 Процес по внедряване**

На фигура 26, е показан порталът на платформата Microsoft Azure.

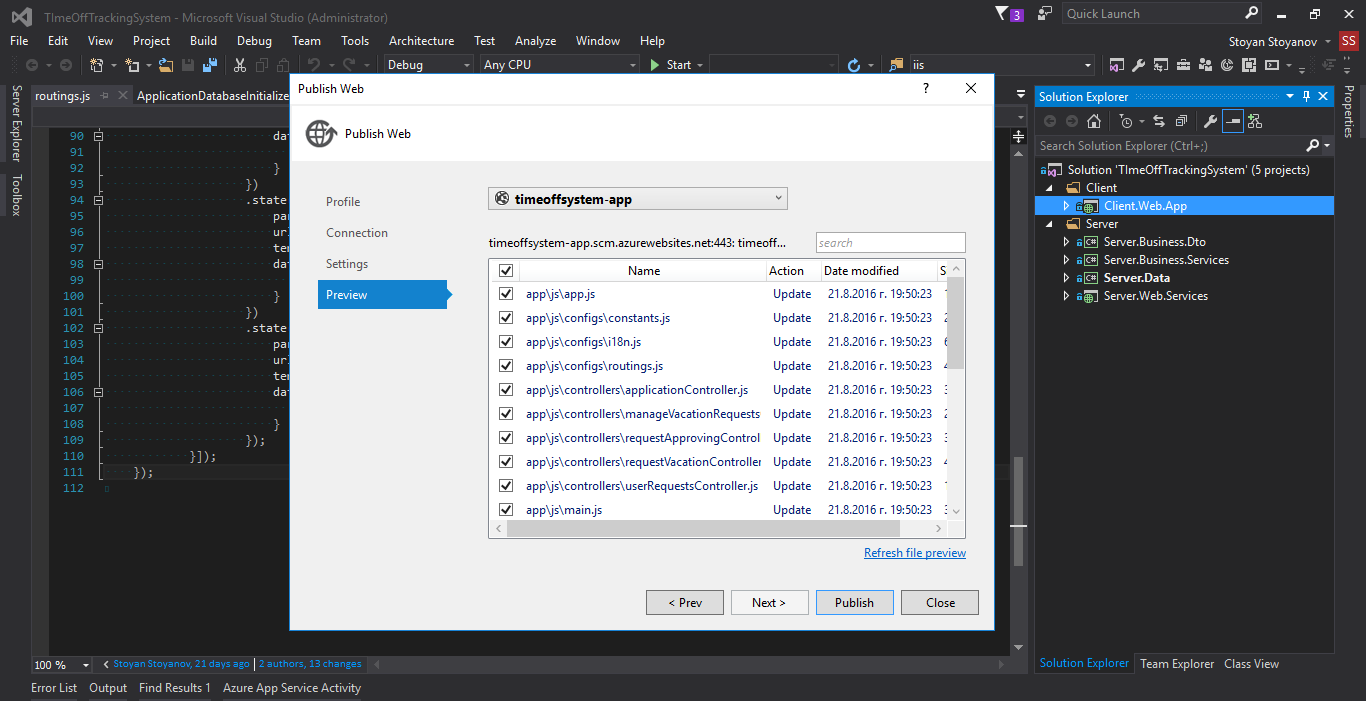


**Фигура 26.** *Портал на Microsoft Azure.*

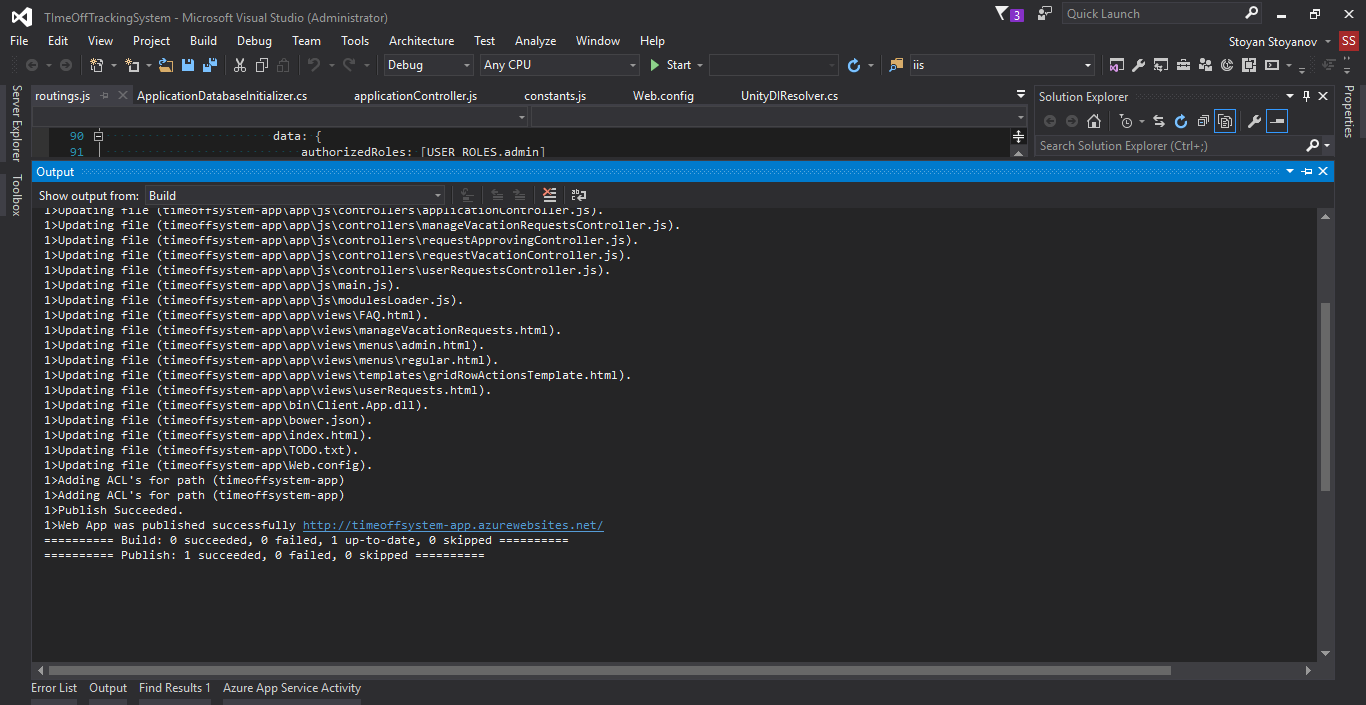
Процесът по внедряване е разделен на две основни части: внедряване на уеб базираната информационна система и внедряване на RESTFull уеб услугите.

Уеб системата се внедрява на адрес: <http://timeoffsystem-app.azurewebsites.net>, докато RESTFull уеб услугите се доставят на адрес: http:// <http://timeoffsystem-api.azurewebsites.net>. Това разделение, допълнително улеснява, както внедряването, така и скалируемостта и преносимостта на системата.

Внедряването се извършва чрез Visual Studio. Важно е да обележим, че доставянето винаги се извършва от master клона от Git хранилището. На фигури 27 и 28 е показан изглед от процеса на вденряване, чрез Visual Studio.



**Фигура 27.** *Първа стъпка от внедряване на уеб системата.*



**Фигура 28.** *Резултат от внедряване на уеб системата.*

Базата от данни, се създава и управлява, посредствум RESTFull уеб услугите. Поради факта, че се използва механизма Code First, уеб услугите, ще създадът или обновят структурата на базите от данни автоматично.

Важно е да се отбележи, че облачната платформа Microsoft Azure, сама определя как ресурсите ще се използват от дадено уеб приложените. Така например, ние нямаме директед достъп до IIS и съответния Application Pool към който е зачислена уеб системата.

**Глава 7. Заключение** (1-2стр.)

**7.1. Обобщение на изпълнението на началните цели**

В настоящата дипломна работа беше представен проблемът за управлението на отсъсвията на служители в корпоративна среда, както и примерно решение, реализирано чрез използването на съвременни технологии и базирано на най-добрите практики и разработки съществуващи до момента в тази област.

Акцентира се върху основните недостатъци на съществуващите решения, като на база сравнителен анализ бяха избрани качествените и функционални

изисквания към системата. Изборът на технологии за разработка на системата се базира на сравнителен анализ между различните възможности и това доколко те удовлетворяват изискванията към решението. Бяха избрани, въможно най-съвременни платформи и иделогии за разработка на подобен тип софтуер. Беше създадено SPA приложение с AngularJS, както и RESTFull уеб услуги с ASP.NET Web API. Всичко това, беше внедрено в облачната среда Microsoft Azure.

Качеството на разработения продукт се потвърждава от различните тестови сценарии изпълнявани по време на самата разработка.

Архитектурата на системата и разделянето ѝ на компоненти, предполагат бъдещо развитие и правят системата адаптивна към променящата се среда както в технологично, така и в бизнес отношение. Така например, дори в бъдеще да се реши реализацията на системата да бъде променена на мобилно или декстоп приложение, само клиентската част би имала нужда от трансформация.

**7.2. Насоки за бъдещо развитие и усъвършенстване**

Възможностите за бъдещо развитие на уеб системата са много и от най-различно естество. Към системата, може да се добави допълнителен календар, наречен „Календар на колеги“, който интерактивно да показва, кога даден потредител отсъства и поради каква причина.

Друга възможност за подобряване е интегрирането на системата със съществуваща база от данним например с корпоративната активна директория. По-този начин, може да се подобри процеса по създаване на потребители и по-лесната интеграция със съществуващи системи.

Също така, към системата може да се добави и функционалност за препоръка за почивни дни, като по този начин максимално да се улеснят служители при избор на дни за тяхната почивка.

Не на последно място към системата може да се добави автоматизирано тестване на потребителския интерфейс. Както се спомена по-рано, тази работа би се свършлия прекрасно от платформата на Protractor.

Източници

[1] - <http://www.lawsbg.com/lectures/67-trudovopr/266-rabotnik.html>

[2] - <http://www.bg-pravo.com/2011/04/28_30.html>

[3] - <http://lex.bg/laws/ldoc/1594373121>

[4] - <http://pravatami.bg/584>

[5] - <http://balans.bg/5205-vyznikvane-na-trudovo-pravootnoshenie/>

[6] - <http://www.paidtimeoffsoftware.com/PaidTimeOffSoftware.htm>

[7] - <http://www.cws-software.com/timeout.htm>

[8] - <http://www.un.org/en/universal-declaration-human-rights/index.html>

[9] - <https://en.wikipedia.org/wiki/C_Sharp_(programming_language)>

[10] - <https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx>

[11] - <https://en.wikipedia.org/wiki/Java_(programming_language)>

[12] - <https://en.wikipedia.org/wiki/Python_(programming_language)>

[13] - <https://en.wikipedia.org/wiki/Ruby_(programming_language)>

[14] - <https://en.wikipedia.org/wiki/JavaScript>

[15] - <https://en.wikipedia.org/wiki/HTML>

[16] - <https://en.wikipedia.org/wiki/MySQL>

[17] - <https://en.wikipedia.org/wiki/Oracle_Database>

[18] - <https://en.wikipedia.org/wiki/PostgreSQL>

[19] - <https://www.microsoft.com/en-us/cloud-platform/sql-server>

[20] - <https://en.wikipedia.org/wiki/Apache_HTTP_Server>

[21] - <https://en.wikipedia.org/wiki/Nginx>

[22] - <https://en.wikipedia.org/wiki/Amazon_Web_Services>

[23] - <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

[24] - <http://jmeter.apache.org/index.html>

[25]