ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ

****

Катедра „Компютърни науки и технологии“

**ДИПЛОМНА РАБОТА**

**Тема:**

Проектиране и реализиране на система за онлайн резервации – хотел за домашни любимци

**Изготвил:Джемиле Исмаил Ахмед**

**Специалност:**Софтуерни и интернет технологии

**Факултетен номер: 61462152**

ТУ Варна, 2018 г. Ръководител:

/доц. Г. Кунев/

Съдържание

[1. Увод 4](#_Toc523643294)

[2. Анализ на проблема и избор на технологии 5](#_Toc523643295)

[2.1. Анализ на проблема 5](#_Toc523643296)

[2.2. Избор на технологии 5](#_Toc523643297)

[2.3. Angular 5](#_Toc523643298)

[2.3.1. Историческо развитие на Angular 6](#_Toc523643299)

[2.3.2. Angular Архитектура 7](#_Toc523643300)

[2.3.3. HTTP клиент 10](#_Toc523643301)

[2.4. JWT 11](#_Toc523643302)

[2.5. Typescript 12](#_Toc523643303)

[2.5.1. Методив TypeScript 14](#_Toc523643304)

[2.5.2. Прихващане и обработка на изключения в Typescript 15](#_Toc523643305)

[2.5.3. Използване на класове и обекти в Typescript 15](#_Toc523643306)

[2.5.4. Модули в Typescript 16](#_Toc523643307)

[2.5.5. Цикли в Typescript 17](#_Toc523643308)

[2.5.6. Масиви в Typescript 17](#_Toc523643309)

[2.6. Java Spring 17](#_Toc523643310)

[2.7. Бази данни – MySQL 18](#_Toc523643311)

[3. Проектиране на системата 20](#_Toc523643312)

[3.1. Изисквания 20](#_Toc523643313)

[3.1.1. Спецификация на „Kingdom Pets” 20](#_Toc523643314)

[3.1.2. Функционален дизайн на „Kingdom Pets” 20](#_Toc523643315)

[3.2. Проектиране на приложението 33](#_Toc523643316)

[3.2.1. Концептуален модел 33](#_Toc523643317)

[3.2.2. UML диаграма 33](#_Toc523643318)

[3.2.3. Essential Use Case 34](#_Toc523643319)

[4. Реализиране на системата 35](#_Toc523643320)

[4.1. Разработка 35](#_Toc523643321)

[4.1.1. База данни 37](#_Toc523643322)

[4.1.2. Публичен модул 41](#_Toc523643323)

[4.1.3. Административен (управленски) модул 48](#_Toc523643324)

[4.2. Съпоставка с подобни приложения 58](#_Toc523643325)

[5. Тестове и резултати 60](#_Toc523643326)

[5.1. Тестове и грешки 60](#_Toc523643327)

[5.1.1. Клиентски част 60](#_Toc523643328)

[5.1.2. Сървърна част 63](#_Toc523643329)

[5.2. Резултати 64](#_Toc523643330)

[6. Заключение 65](#_Toc523643331)

[7. Източници на информация 66](#_Toc523643332)

[8. Приложение 66](#_Toc523643333)

# Увод

Онлайн резервациите са неизменна част от дейността на хотелиерите. Динамиката на ежедневието и присъствието на хората в интернет правят една такава система „животоспасяваща“ за бизнеса, което предполага нейното внимателно и максимално изчерпателно проектиране, респективно реализация. Текущата разработка обхваща няколко основни момента от дейността на хотел, като акцентира преди всичко върху резервациите, комуникацията и контрола от страна на администратора.   
 Създаденото приложение е предназначено да бъде използвано от хотел, като система за управление на резервациите. В случая хотелът е с по-специфичен характер, поради същността на предлаганите услуги и своите гости, които са домашни любимци. При постъпване на нов домашен любимец се съхранява информация за него, за собственика му, както и за отговорното за него лице. Записва се специфична информация за любимците, каквато е: информация за характерка на животното, специални хранителни режими, тренировъчни режими, списък с допълнителни важни дейности, които трябва да се следят по график, контактна информация, медицински картон и други. Чрез тази информация се гарантира комфорта и безопасността, както на животните, така и на хората отговорни за тях.

# Анализ на проблема и избор на технологии

## Анализ на проблема

При навлизането на пазари в условията на свръх конкуренция и предлагане на различни решения на близки по същност проблеми е трудно и е свързано, преди всичко с усещането, което даден продукт остава у потребителя. А това усещане, тази необходимост от употреба, която е обект на стремеж за създаване е в зависимост от т.нар. добавъчна стойност към основния продукт, който се предлага. В случая на текущата разработка, тази добавъчна стойност е комфорта за крайния потребител, осигурен от възможността за боравене с иновативна система достъпна на всички познати на човека устройства.   
Същевременно, по-голямата част от съществуващите решения на проблемите с осигуряването на комфорт и работа с подобни системи за контрол на резервациите не дават възможност за достъп на крайният потребител да задава редица параметри свързани с живота на любимецът си в реално време – в т.ч хранителен режим , тренировъчен режим, добавяне на нови екстри и прочие.

## Избор на технологии

Възприетите технологии са оптимизирани за работа с ограничени ресурси и в съставът им в текущата разработка се включват:

## Angular



AngularJS или Angular 1 е уеб базирана платформа с отворен код разработена от програмистите Миско Хевъри и Адам Абронс през 2009 година, като в последствие технологията е припозната от Google, който се грижи за нейното обслужване и развитие. Технологията внася един нов поглед върху уеб програмирането и бързо заема централно място и става любимо средство за изграждане на уеб-базирани системи. Изгражданите чрез платформата приложения имат архитектура „Приложение на една станица“[[1]](#footnote-2).

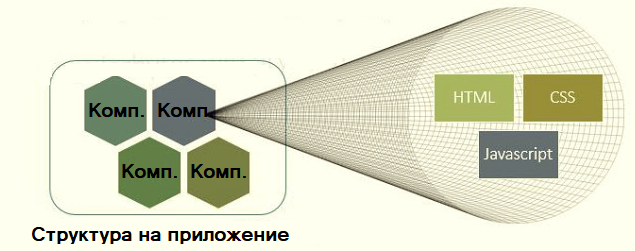
### Историческо развитие на Angular

В исторически план платформата Angular търпи голямо развитие, като минава през няколко основни версии. След „големия бум“ с въвеждането му през 2009-2010 г. Google поемат контрола и решават да пренапишат ядрото на AngularJS, като заместват Javascript, като език на който е писан с Typescript – диалект на Javascript, който дава възможност на програмистите да използват възможностите на строго типизираните езици, както и възможността за изграждане на структура от класове.   
Първата официална версия на Angular, след пренаписването е Angular 2, която влиза в употреба на 14 Септември 2016г. Очаквано и тя става хит и търпи особено голямо развитие. Тази нова версия на Angular платформата е коренно различна от AngularJS. Основните различия са:

* Angular вече не поддържа концепцията за „обхват“ или контролери , вместо това се използва йерархична структура от компоненти, които изграждат архитектурата на приложението;
* Модуларност – голяма част от логиката вече се изнася в т.нар модули. Модула в контекста на технологията е средство за обединяване на логически свързани компоненти, като идеята е всички услуги, които касаят тези компоненти да са включени само в този модул, без да товарят системата излишно;
* Angular има различен синтаксис. Използват се “[ ]” за свойствата, които обработват модели и “( )” за обработка на събития т.нар data binding[[2]](#footnote-3);
* Динамично зареждате – Angular дава възможност за динамично зареждане на ресурсите свързани с уеб приложението. За описване на този процес се използва и понятието “lazy loading” или „мързеливо зареждане“, при което ресурсите необходими за визуализиране на даден компонент се изтеглят само, когато са необходими, а не както до сега – цялото приложение;
* Въвеждане на Typescript – иновативен подход, чрез който се решават голяма част от проблемите свързани с използване на „чист“ Javascript, проблеми свързани преди всичко с организирането на данните и моделите в класове и методи;
* Въвеждане на т.нар Observable – това е изключително нова технология, която служи за събиране на информация от потребителския интерфейс и обработката на тази информация асинхронно. По своята същност Observable представлява поток на информация, който ни помага да обработваме информацията асинхронно и прави прехода от frontend към backend;

За да се различават тези концептуални различни версии на платформата AngularJS програмистите на Google преименуват новата версия в Angular. Тази първа основна версия на новото поколение на платформата е известна в интернет пространството, като Angular 2.   
 Следващата основна версия е Angular 4. Тя навлиза в практиката на 13 декември 2016г. и надгражда значително своя предшественик, като към основия набор от пакети се добавят и т.нар рутери. Рутерите дават за възможност за по-лесна навигация между страниците на приложението. В тази версия се въвежда и т.нар HttpClient, който представлява малка, лесна за употреба и мощна библиотека за правене на HTTP заявки.   
 Последната версия на Angular, която е и основа за приложението представено в тази разработка е Angular 5. Официалната дата на издаване е 1 ноември 2017 г. С тази версия разработчиците на Google внасят значителни подобрения над предходната версия, които се изразяват във внедряването на библиотека за интернационализация и глобализация, която е отговорна за осигуряването на мултиезичност на приложението. Акцента в тази версия пада върху възможността за създаване на т.нар Progressive Web Apps (PWAs) или уеб приложения, които са обикновени уеб страници или сайтове, но могат да осигурят на крайния потребител усещане за използване на традиционно мобилно приложение. PWA приложенията се описват, като комбинация от функциите и възможностите на съвременните браузъри, с предимствата на мобилните приложения.

### Angular Архитектура

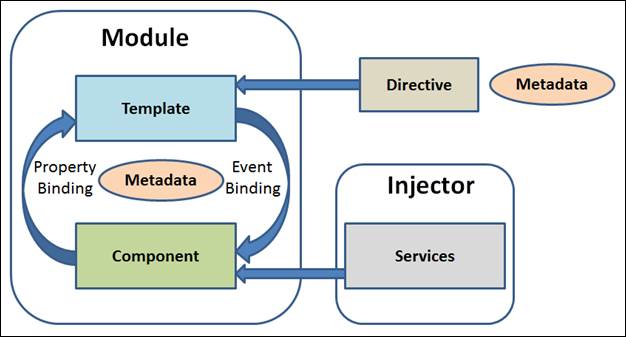
Angular е завършена работна рамка предназначена за изграждане на клиентски приложения, по-специално браузърно базирани приложения. Angular ни позволява да изграждаме приложенията си, като прилагаме т.нар Компонентно базиран дизайн (архитектура).   


Фиг. ‑ Компонентно базирана архитектута на уеб приложение

Браузър приложенията могат да бъдат представени, като съставени от HTML, CSS и Javascript. Проблемите възникват, когато приложението стане сложно (комплексно) и стане трудно да се поддържат и по-трудно става интегриране на промените.

Angular решава тези проблеми, като интегрира Компонентно базирана архитектура. Това е шаблон, който препозиционира различен брой компоненти в дадено приложение, като всеки компонент заема централното място в приложението. Шаблона ни позволява да преизползваме различни компоненти в различни места в приложението. Компонента може да бъде малка или голяма част от приложението. Тези части могат да бъдат и комплексни включващи в себе си и други компоненти.   
 Подходът за изграждане на приложението чрез компоненти е много полезен от гледна точка на потребителския интерфейс и усещане, защото компонентите са по-лесни за поддръжка и управление. Асоциация с реалния свят е например климатика, спрямо къщата. Климатика е комплексен компонент изграден от множество подкомпоненти – вентилатор, кондензатор, електроника и др. Той обаче е част от приложението ( къщата ), която е изградена от други компоненти. Проблем в един от компонентите в къщата не спира работата на останалите.

Структурата на цялостно приложение изградено с Angular е представено на фиг.2



Фиг. ‑Архитектура на приложение изградено в Angular 5

Компонентния подход разглежда приложението, като съвкупност от множество елементи. При тази архитектура е от изключително важно значение логическата групировка на тези елементи. Няколко компонента могат да бъдат логически групарини в модул. Всеки модул може да има подмодули, респективно компоненти. Angular дефинира директивата NgModule, която допълва JavaScript (ES2015) модула. В NgModule се декларира контекста, който подлежи на компилиране и който представлява набор от компоненти, качени на домейн, работен поток или тясно свързан набор от възможности. Чрез тази директива Angular свързва компонентите със свързаните с тях програмни кодове (например услуги), за да образуват функционални единици.   
 Всяко Angular приложение има един root модул, който конвенционално носи името AppModule, и чрез който се осигурява зареждането на компонентите в приложението.

Компонентите от своя страна пък представляват HTML темплейтни фрагменти, в които чрез интерполация може да се използват променливи дефинирани в контролера на съответния компонент.Дадено Angular приложение съдържа задължително поне един компонент. (root компонент). Чрез него се представят всички останали компоненти в структурата на страницата, а именно DOM дървото. Компонентите се репрезентират програмно чрез използване на декоратора @Component. Декораторите са функции, които модифицират JavaScript класове. В ядрото на Angular има дефинирани няколко ключови декоратора, които са от значение преди всичко за метаданните. Всеки компонент контролер е свързан с метаданни. Метаданните са свързани и описват темплейта, който изгражда изгледа.Както бе описано по-горе значително място в структурата заемат т.нар услуги ( сървиси ). Те се имплементират в цялостната структура на приложението с помощта на т.нар. декоратор Injector. Голяма част от тези услуги се грижат за достъпа на приложението до сървъра ( т.е правят HTTP заявки), използват се за data binding, за валидиране на полета за въвеждане и форми.   
 Routing е може би най-важната услуга, която се осигурява от ядрото на Angular. Посредством нея приложението може лесно да навигира между различните потребителски интерфейси изграждащи SPA приложението. Routing услугата е базирана на конвенциите за навигиране имплементирани във всички модерни браузъри, а именно:

* Въвеждане на URL в адрес бара и пренасочването, което прави браузъра към кореспондиращата страница;
* Кликания върху хипервръзки в страницата и пренасочването, което браузъра прави към нова страница;
* Интерфейсните браузърни операции за напред и назад намиращи се в менюто до адресната лента;

Рутера свързва URL като път към изглед, а не към страница. Когато потребителя инициира някакво действие, като например натискане на хипервръзка, това ще презареди текущата сесия, като покаже нова страница, рутера просто скрива и показва друг компонент, който изгражда дадения изглед без да презарежда страницата.

Настройката за функционирането на изложената до тук архитектура се прави в един специален конфигурационен файл наречен tscofig.json. Това е файл част от основната (root) директория на всеки проект. В този файл се посочват файловете в кореновата директория, някои настройки за четене и запис на допълнителни файлове (в т.ч и външни за дадения проект) в структурата, които могат да укажат влияние в процеса на компилиране на проекта. Самият процес на компилиране може да стане по един от следните начини:

* Чрез извикване на TSC безналичие на външни файлове, като в този случай компилатора търси конфигурационния файл, започвайки от текущата директория и продължавайки до края и след като го открие създава верига от извиквания и компилации на всички свързани с приложението файлове;
* Чрез извикване на TSC без наличие на външни файлове, но с команда –project ( или просто –p) в терминала, като опция, която дефинира пътя към директорията, в която се съдържа конфигурационния файл, или път към валиден JSON файл съдържащ конфигурации;
* Когато в процеса на компилиране с използването на команда TSC се зададе последователност от файлове за компилиране, настройките съдържащи се в tsconfig.json файла се игнорират;

### HTTP клиент

Повечето клиентски приложения в т.ч уеб и мобилни комуникират с някакви сървърни услуги чрез HTTP протокол. Модерните браузъри поддържат два различни типа API за правенето на HTTP заявки: XMLHttpRequest интерфейс и fetch() API. HttpClient в Angular осигурява опростен клиентски HTTP API, който е базиран на observable-и. Това означава, че всички заявки, които се правят от HTTP модула ще връщат потоци от данни, за които потоци следва да се направи абониране (subscribe) по един или друг начин. Във връзка с работата с тези потоци (observables) е необходимо да се изведат няколко особености:

* Ако не се абонираме за тези потоци от данни, означава че системата ще ги игнорира;
* Ако се абонираме повече от един път за даден поток от данни, това означава че толкова паралелни заявки ще бъдат отправени към сървъра;
* Потоците от данни дават възможност за обработка на грешки, като връщат грешката при провал в HTTP заявката;

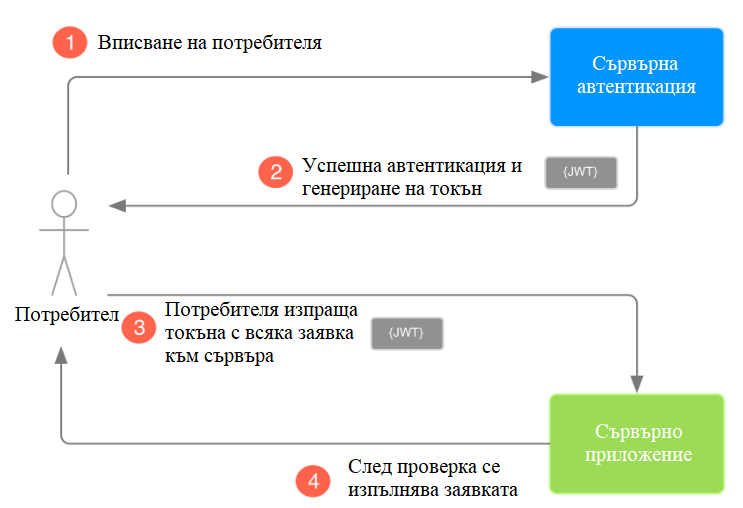
HTTP модула дава възможност за подаване на различни типове заявки към сървъра – PUT,POST,DELETE,GET и т.н В допълнение към това имаме възможност за дабавяне на параметри или някакви допълнителни хедъри, например Authorization, към който да се добави съответния token, с който сървъра да автентицира заявките от даден потребител, може да се добави тип на съдържанието Content-Type, с който да се зададе тип на връщания резултат и т.н (всичко това се прави с цел повишаване на сигурността).   
Важен момент при работата с потоци от данни е, че задължително следва да се абонираме за него, което означава че се ангажира някакъв ресурс – памет, процесорно време и прочие, който ресурс следва да бъде освободен след приключване на работата с дадения поток. Избора на подходящ момент за освобождаване на ресурса следва да бъде съобразен с фазите от жизнения цикъл на всеки компонент в структурата на Angular приложението. Обичайна практика е освобождаването на ресурсите да се осъществява на OnDestroy().

## JWT



JSON Web Token е JSON-базиран отворен стандарт (RFC 7519) за създаване на ключ за достъп, който съдържа в себе си последователност от искания за успешно автентициране на потребителя и правата му в системата.По своята същност представлява JSON обект, който се разглежда, като сигурен начин за предване на съвкупност от информация между страни в процеса на комуникация. Включването на токъна в процеса обмен на информация се изразява чрез включването му в заглавната част на всяка заявка към сървъра с помощта на етикета Authorization.

На фигура 2-3 е представено приложението на JWT в процеса на комуникация между потребителя и сървъра. След успешно въвеждане на входни данни и изпращането им към сървъра, в него се генерира т.нар токън, който се изпраща обратно към потребителя в случай, че данните за вписване са коректни.



Фиг. ‑ Приложение на JWT в процеса на комуникация между потребителя и сървъра

Структурата на жетона (токъна), включва няколко задължителни части:

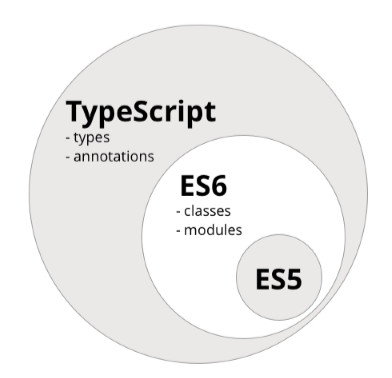
* Header
* Payload
* Signature

Обикновено хедъра включва в себе си типа на токъна (как payload може да бъде интерпретиран) и името на алгоритъма използван в signature. Цялата структура на жетона е криптирана чрез стрингова репрезентация base64.   
Payload представлява информацията съдържаща се в тялото на заявката представена във вид на JSON обект. В нея се съдържат т.нар claims (искания).

## Typescript



TypeScript представлява типизиран диалект на JavaScript, който се компилира до чист Javascript. Като диалект на друг език TypeScript не добавя нищо ново, а просто начин за писане, който е удобен, лесен за възприемане и изчистен. Всяка програма писана на TypeScript минава през режим на компилиране (транслиране) на JavaScript, като се поддържа от всички стари браузъри. Работи еднакво добре на всякакъв тип браузъри независимо от операционната система или платформа. TypeScript поддържа файловете за дефиниране, които могат да съдържат типова информация за съществуващи библиотеки на JavaScript, подобно на C++ заглавните файлове могат да опишат структурата на съществуващите обектни файлове. Това позволява на други програми да използват стойностите, дефинирани в файловете, ако те са зададени като статични в тях.   
 TypeScript произлиза от възприеманите недостатъци на JavaScript за разработването на мащабни приложения и като разработка на Microsoft е в основата на приложенията, както чисто вътрешните за Microsoft, така и за външните.



Фиг. ‑ Typescript, като разширение на съществуващите ECMAScript 5 и 6

TypeScript програмите подлежат на компилиране към JavaScript. Компилатора се нарича TSC и е написан на TypeScript. В резултат на това той може да бъде компилиран в обикновен JavaScript и след това да бъде изпълнен във всеки браузър engine. Пакета включващ компилатора се доставя заедно със сървърния пакет, който изпълнява кода.Текущата версия на компилатора поддържа и ECMAScript 5 по подразбиране. Има настройка, която позволява да се промени целевия стандарт, за писане на JavaScript код, към ECMAScript 2015 (ES 6), за да се направи използването на езика съобразено с последните въведени разширения и особености, като например генератори, класове, наследявания и други. Класовете въпреки, че официално са въведени с ES 6 са част и от двете версии на компилатора.

### Методи в TypeScript

Методите са изграждащи фундаментални блокове от всяко едно приложение на JavaScript. Наборът от методи определя и слоевете абстракция на всяко приложение, работата с класовете , обектите, модулите и са в основата на концепцията за обектно-ориентирано програмиране, а именно концепцията за скриване на информация и ограничаване на достъпа т.нар капсулация. В TypeScript, въпреки наличието на класове, именовани пространства и модули, функциите имат ключова роля в описването на това как се случват нещата в даденото приложение. TypeScript добавя и няколко нови възможности към стандартните на JavaScript, които правят тяхното разбиране и работата с тях доста по-лесна.   
 Подобно на JavaScript и тук методите могат да бъдат създадени по двата установени вече начина:

* Като именована функция – използване на ключова дума function;
* Като анонимна функция – () са индикатор за наличието на функция;

Съществено удобство в употребата на TypeScript за писане на методи е наличието на типизирани функции. Езика дава възможност за добавяне на тип, както на самата функция, така и на параметрите, които получава и обработва тя. В случаите, когато е налице функция, която връща резултат TypeScript дава възможност за указване на типа на връщания резултат още в момента на деклариране на метода. Наличието на компилатор и правила правят писането изключително лесно, защото повечето среди за програмиране т.нар IDE-та разполагат с т.нар intelliSense, който показва на момента наличието на синтактична грешка.   
 Тъй като TypeScript е диалект написан и поддържан от Microsoft стремежа на разработчиците е да доближат максимално разработването на софтуер до платформата .NET. Идентично и като смисъл, и като синтаксис е възможността за добавяне на незадължителни параметри (т.нар опционални) и параметри по подразбиране. Както и в другите езици за програмиране всеки параметър посочен в прототипа на функцията е задължително условие за нейното изпълнение, докато наличието на опционален параметър в прототипа не е задължително условие за изпълнението.

### Прихващане и обработка на изключения в Typescript

В TypeScript няма клас Exception вместо това се използва класа Error. Грешките, които възникват се извеждат в процеса на стартиране, а не на компилиране. Когато е налице някаква грешка тя е видима в конзолата на браузъра и не можем да я прихванем, а следва да пренапишем кода и да прекомпилираме приложението. Прихващането на изключения в TypeScript може да стане по един от следните начини:

* Клас Error – когато се стартира приложението и възникне грешка чрез ключовата дума new се създава инстанция на класа и в комбинация с throw извежда на екрана името на системната грешка. TypeScript дава възможност за извеждане на потребителско зададено име или съобщение;
* Създаване на модул за прихващане на грешките и изключения включват клас Error. Добрите практики включват създаването на логически групирана услуга за обработка на изключенията, която обхваща всички сървърни грешки, друга услуга за валидиране на входящата информация и извеждане на грешки чрез диалогови прозорци и т.н

Всяка грешка притежава две свойства – системно име и съобщение. Най-често извежданите грешки са:

* EvalError – представлява интерфейс базиран на класа Error. Tази грешка обикновено няма никакви самостоятелни свойства и се засича при извикаването на глобалната функция eval(). Това означава, че нейното прихващане трябва да стане с извикването на функцията, която я поражда;
* RangeError – подобно на EvalError няма самостоятелни свойства над тези дефинирани в главния клас за грешки. Грешката се хвърля, когато има зададена стойност, която не е разрешена за даден обхват или когато подаваме различен брой аргументи от задължителните за даден клас;
* ReferenceError – грешката се хвърля, когато е използвана променлива, която не е била инициализирана т.е референцията към дадения обект или променлива сочи към null;
* SyntaxError – грешката се извежда, когато сме използвали дума или съвкупност от ключовите думи по неправилен начин;

### Използване на класове и обекти в Typescript

JavaScript, като основа на TypeScript разглежда всеки обект, конструкция или клас, респективно компонент (в т.ч и функциите), като обекти. Традиционният JavaScript използва функции и прототипно наследяване за изграждане на компоненти за многократна употреба, но това е неудобно за програмистите, които са свикнали с обектно-ориентирания подход, при който класовете наследяват функционалност и обекти са изградени от тези класове именно това е причината за имплементиране на ООП в TypeScript.   
 Синтаксиса е базиран на C# . Инстанцирането на обект става с помощта на ключовата дума new. С инстанцирането на обект се вика неговия конструктор. В случаите, когато той не е написан от програмиста се създава фиктивен такъв, който не променя или задава никакви стойности на член променливите. За разлика обаче от другите езици за програмиране тук може да е налице един единствен, затова е необходимо програмиста да се съобрази с нуждите на програмата, респективно инициирането на класа. TypeScript дава възможност и за създаването на интерфейси и абстрактни класове. За наследяването на клас се използва ключовата дума extends, а за имплементиране на интерфейс – implements. Специфична особеност е, че член променливите могат да са инициирани с достъп readonly. Този тип променливи следва задължително да бъдат декларирани и инициирани в конструктора на дадения клас и тяхната стойност се разглежда, като константна в рамките на класа.

### Модули в Typescript

С приемането на ECMAScript 6 в JavaScript света се въвежда концепцията за модулите. Те представляват съвкупност от логически свързани конструкции в т.ч методи, константи, класове. Биват два вида:

* Вътрешни модули – наричат се именовани пространства;
* Външни модули – наричат се просто модули и се използват чрез импортиране в съответния клас или друг модул;

Модулите се изпълняват в техния собствен обхват, а не глобално за приложението. Това означава, че всички променливи, функции, класове и т.н декларирани в модула не са видими извън модула, освен ако изшлишно не са изнесени. Изнасяването на модули променливи и класове става чрез ключовата дума export пред съответната променлива. Модулите са декларативни, а връзките между тях се определят в зависимост от изнасянето, респективно внасянето (export/import) на ниво файл. Внасянето или т.нар импортиране става възможно чрез т.на модул loader. Поддържането на такъв loader изисква някаква външна JavaScript библиотека, в повечето случаи това зависи от типа зареждане на модула зададен в конфигурационния файл на всеки един проект. Има няколко варианта:

* CommonJS;
* Асинхронно дефиниране на модули (AMD);

Настройката по подразбиране е CommonJS, това е и начина на зареждане в браузъра (клиентската част), за зареждането в сървърната част отговаря NodeJS.

### Цикли в Typescript

TypeScript, като диалект на JavaScript дава възможност за реализиране на цикли чрез for, while и do-while. Циклите дават възможност да се повторят действие или група от действия различен брой интерации. Като всеки модерен език и тук има възможност за безкрайни цикли и условия има задължително условие за излизане от такива. Чрез ключовата дума continue се прескача на следващата итерация от цикъла.

### Масиви в Typescript

Използването на променливи за съхраняване на някакви стойности има следните ограничения:

* Променливите могат да съхраняват по една, единствена стойност към даден интервал от време. Чистият JavaScript дава възможността за предефиниране на дадена променлива, освен като стойност, но и като тип на променливата, но такъв тип предефиниране води до извеждане на синтактична грешка в TypeScript;
* Променливите в дадена програма заемат определен блок от паметта, който не е последователен;

За решаване на тези проблеми се използват масивите. Масива може да се дефинира, като хомогенна съвкупност от стойности. Създаването на масив и употребата му е както в повечето езици за програмиране. Задава се името на масива последвано от типа, в някои случаи размера. TypeScript подобно на другите езици дава възможност за иницииране в момента на декларирането на дадения масив.

## Java Spring



Spring Framework е технологична рамка (или „framework), която е с отворен код. Тя е предназначена за Java платформата, като предoставя множество различни функции, които улесняват разработката на различни приложения, базирани на Java. Spring е организиран по модулен начин. Въпреки че броят на пакетите и класовете е значителен трябва да се притеснявате само за онези, от които се нуждаете и да не обръщате внимание на останалите. Също така е добре базиран за МVC[[3]](#footnote-4)структура.Spring Framework е платформа на Java, която предоставя цялостна инфраструктурна поддръжка за разработване на Java приложения. Spring обработва инфраструктурата, за да можете да се съсредоточите върху приложението си.Терминът "Spring" означава различни неща в различни контексти. Oсновните функции, които Spring Framework предлагаса:

* Dependency injection (или просто „Инжектор“, но познат още и като „Контейнер на зависимостите“) – най-общо казано това е IoC контейнер (Inversion of Control), който предоставя логически средства за управление и настройка на Java обекти;
* Spring Batch – това е едно цялостно решение, което предоставя възможността за разработването на надеждни batch приложения, които се срещат в модерните бизнес системи. Чрез Spring Batch вие можете с лекота да извършвате обработка на големи обеми от данни, като логване/проследяване, статистика за обработени задачи, управление на ресурси и други;
* Аспектно-ориентирано програмиране (АОП) - това е фундаменталният стил на програмиране, начина по който отделните компоненти на програмата, комуникират помежду си. Има множество парадигми, но най-основните са функционално, обектно-ориентирано, императивно и декларативно. Почти всяка една парадигма има определен подход при решаването на дадени проблеми. Проблемите се групират и „капсулират“ в отделни, независими части и се използват различни абстракции (като методи, модули, класове и др.) чрез които се изграждат необходимите отношения. При изграждането си някои отношения „пресичат“ множество различни абстракции, поради което те се наричат „междусекторни отношения“

## Бази данни – MySQL

MySQL е най-популярната система за управление на SQL бази данни с отворен код. Тя се разпространява и поддържа от Oracle Corporation.

* **Управление на бази данни**

Бази данни са структурирано количество от данни. Бази данни наричаме всичко от най-обикновен списък за пазаруване, през изображения в галерия до огромните количества информация в корпоративните мрежи. За да добавите, получите достъп и да работите с данни, които са съхранени в базите данни на компютъра, вие се нуждаете от система за управление на базите данни, като MySQL Server. Тъй като компютрите вече могат да боравят с огромни количества от данни, системите за управление на базите данни играят ключова и централна роля, било то като отделни приложения или като част от такива.

* **Релационни бази данни**

Релационните бази данни съхраняват данни в отделни таблици, вместо да поставят всичките данни в една голяма директория. За по-бърза работа структурите от бази данни са организирани във физически файловe. Логическият модел, с обекти като бази данни, таблици, изгледи, редове и колони предлага гъвкава програмна среда. Вие настройвате правила, управлявате връзките между различните полета с данни, като „one-to-one“, „one-to-many“, „unique“, “required” или „optional“, както и различни показалци между различните таблици. Базите данни налагат тези правила, така че с добре-проектирана база данни вашите приложения никога няма да се сблъскат с несъвместими, дублиращи се, остарели или липсващи данни.

* **Разлика между SQL и MySQL**

SQL е абревиатура, която означава „Structured Query Language” и е програмен език. Всъщност това е най-често използвания програмен език, който се използва при достъп до бази данни.В зависимост от клиентската програмна среда може да се влезе в SQL директно и да се вградят SQL откъси в код изписан на друг език или да се използват специфични програмно-приложни интерфейси просто за да се скрие синтаксиса на SQL.

* **Отворен код - значение в MySQL**

.   
„Отворен код“ означава, че е възможно за всеки да модифицира софтуера. Всеки желаещ може да свали MySQL софтуера от Интернет и да го използва, без да заплаща нищо. Може да се изучи изходния код и да се промени така, че да отговаря на вашите нужди. Ако клиентът възнамерява да използва MySQL за комерсиални цели трябва да се сдобие и със съответния лиценз

# Проектиране на системата

## Изисквания

При първоначалното планиране е необходимо да са налице добре описани изисквания към приложението, което предстои да се създаде. В противен случай, крайната цел, към която се стреми даден проект би била трудно достижима и първоначалните ни виждания не биха се осъществили.

### Спецификация на„Kingdom Pets”

Приложението , обект на дипломната работа следва да отговаря на следните изисквания:

* Лесен и удобен интерфейс за потребителя;
* Оптимизиран за работа с ограничени ресурси;
* Прихващане на грешки в процеса на изпълнение на приложението;

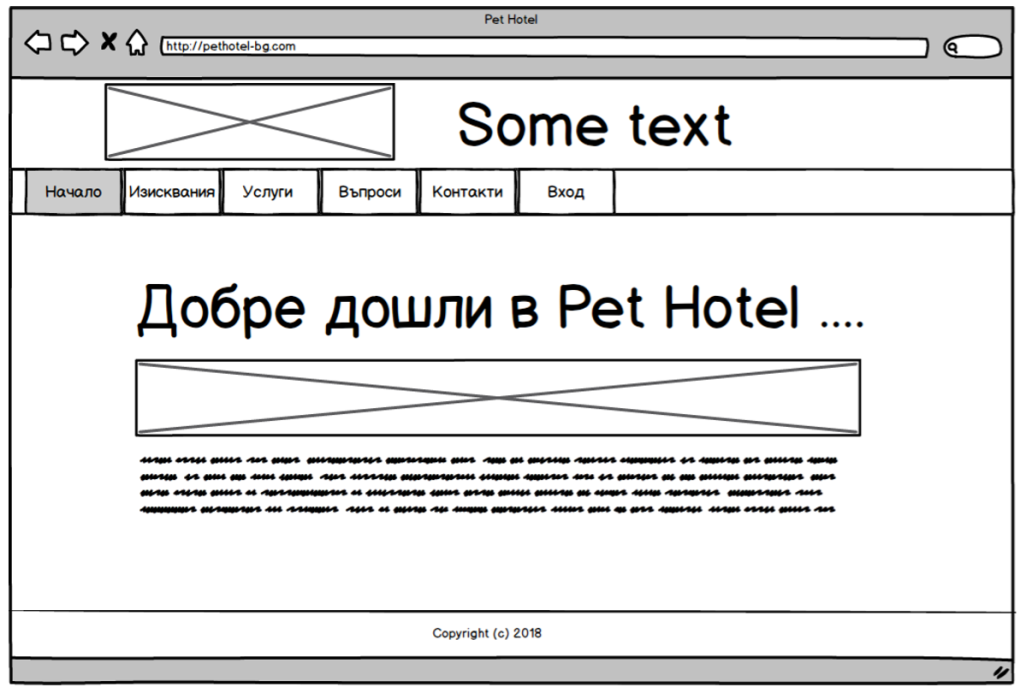
### Функционален дизайн на „Kingdom Pets”

Обект на текущата разработка е уеб-базираното приложение „Kingdom Pets”.В следващите няколко страници е представен функционален дизайн на съответните модули, които са в основата и на направената реализация.

* **Уеб-базирано приложение**

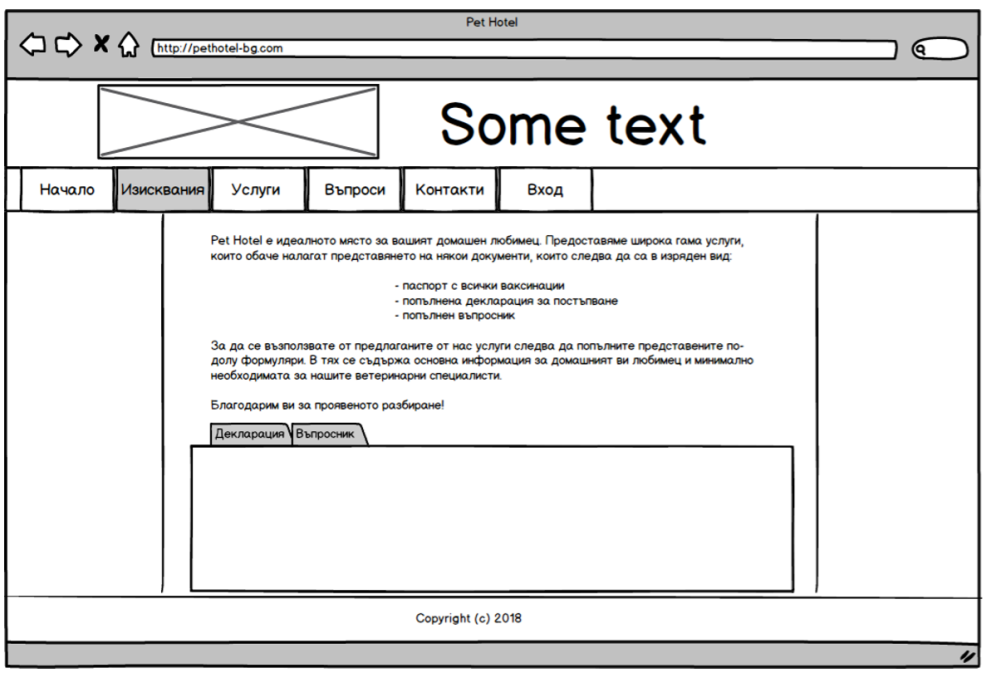
Уеб-базираното приложение Kingdom Pets разполага с няколко работни екрана, които ролево могат да бъдат групирани в три основни подмодула, а именно – публично достъпни, админ панел чрез който да се управлява съдържанието на публично достъпния модул и потребителски панел, в който съобразно делегираните правомощия , потребителя може да управлява резервацията си. Архитектурата на приложението следва да осигури възможност на крайния потребител да осъществява предвидените в спецификацията функционалности. Част от екраните в процеса на разработка претърпяват известни промени с цел подобряване на работата на приложението в програмен план – по-добро време за извеждане на екрана на съответните компоненти, и от гледна точка на клиента – по-добър дизайн, респективно по-лесна употреба. Предвидените функционалности са свързани преди всичко с, основни за системите за онлайн резервации, услуги в т.ч резервация, настройки на цени и услуги, обработка на имейл съобщения и други.

* Основен начален екран



Началният екран е входна точка на потребителя в Kingdom Pets. На вниманието му се представя галерия с представяните услуги и краткото им , в няколко думи, описание. Налице е навигационна лента, която включва бутони за достъп до предвидените функционалности.

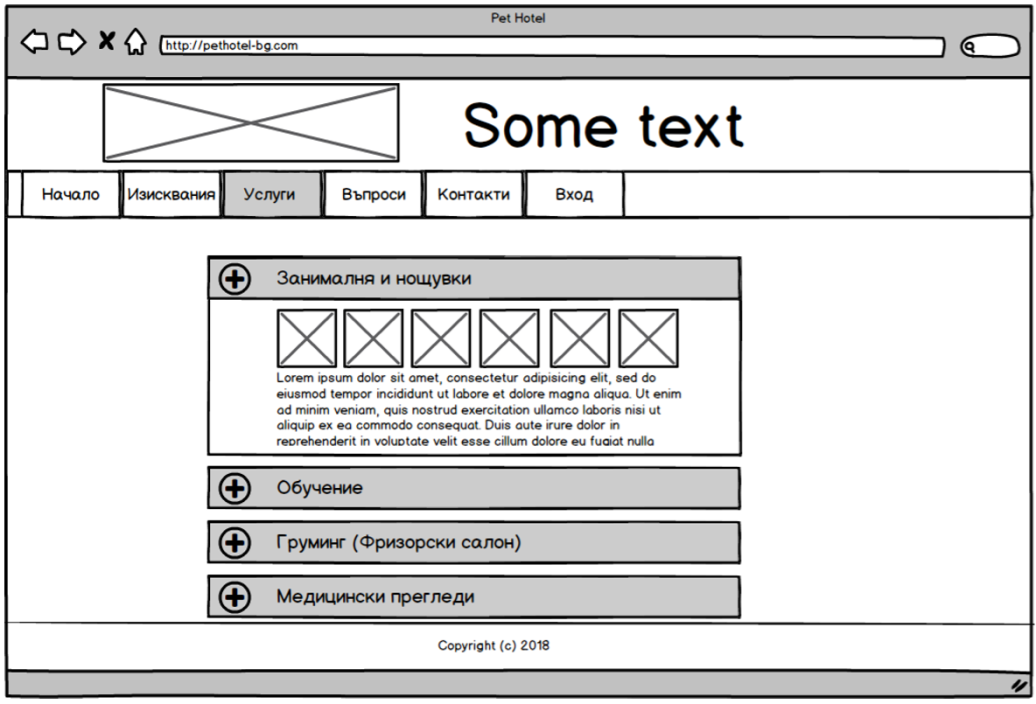
* Екран с изисквания



Тъй като тема на текущата разработка е онлайн система за резервации на стаи в хотел за домашни любимци, подобен тип дейност предполага наличието на документи, които да улеснят персонала, а и рецепцията в процеса на настаняване на животните, като въз основа на тези документи се прави подбор на меню за хранене, подходяща стая. На екрана чрез изглед с раздели са представени два документа, които следва да бъдат представени на физически носител при постъпване на любимците, а именно:

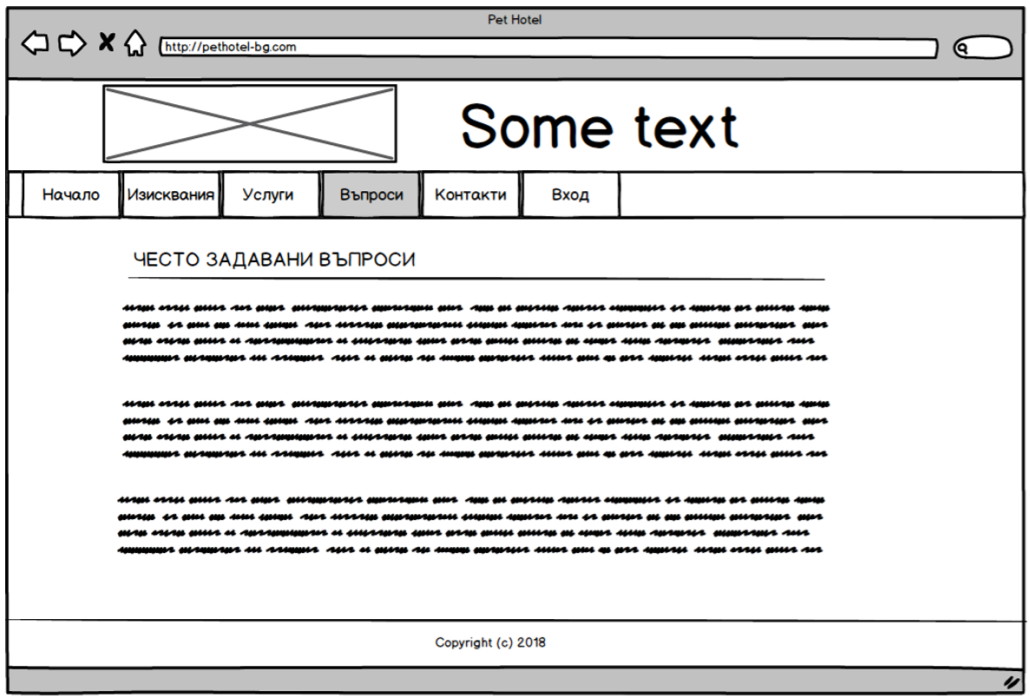
- Декларация – в която собственика декларира, че животното е регистрирано, ваксинирано и оставено на грижите на хотела с негово знание и съгласие;  
 - Въпросник – в който са поместени въпроси касаещи основни моменти свързани с престоя на животното, а именно – възраст на любимеца, име, номер на паспорт, алергии, някакви специални нужди и прочие;

* Предоставяни услуги – описание и цени



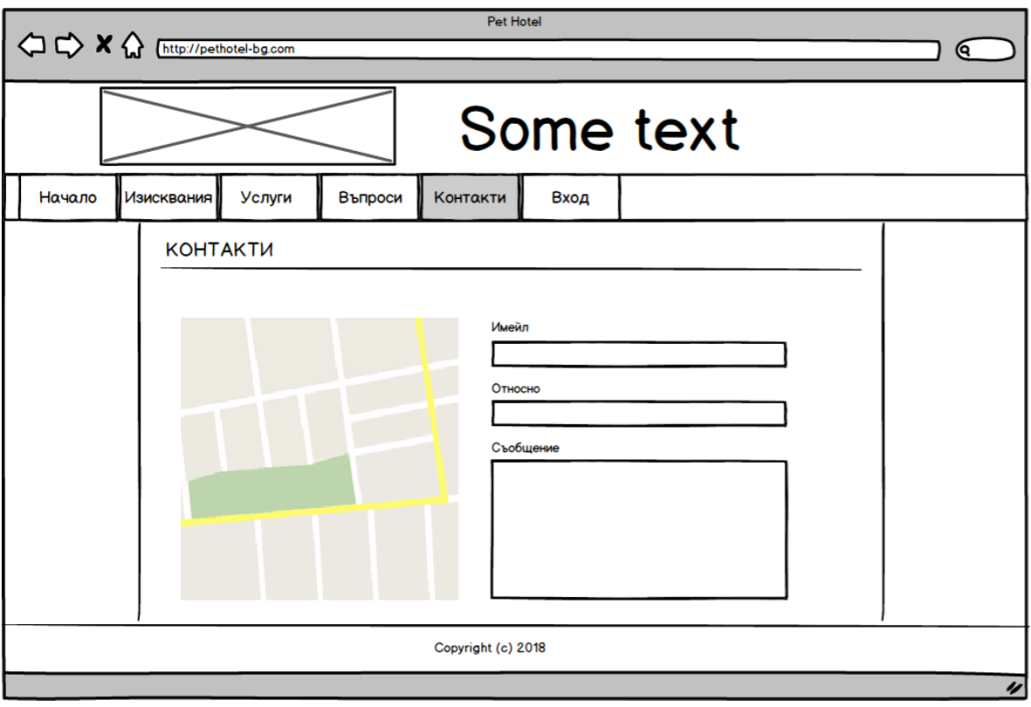
Следващия основен екран включва предоставяни услуги и цени. На него посредством заявка до сървъра се извличат услугите добавени от администратора на система – тяхно подробно описание и цени. Предвидената визуална реализация включва т.нар toggle бутони, които разкриват, респ. скриват панел с информация за конкретната услуга.

* Често задавани въпроси (ЧЗВ)



Екрана за ЧЗВ е изключително важен за всеки тип онлайн система, тъй като в него се поместват отговори на най-често възникващите проблеми. В контекста на разработката тук ще бъдат извеждани въпроси касаещи настаняването, престоя, обслужването, работното време на центровете за обслужване на любимците в т.ч медицински и тренировъчен и други.

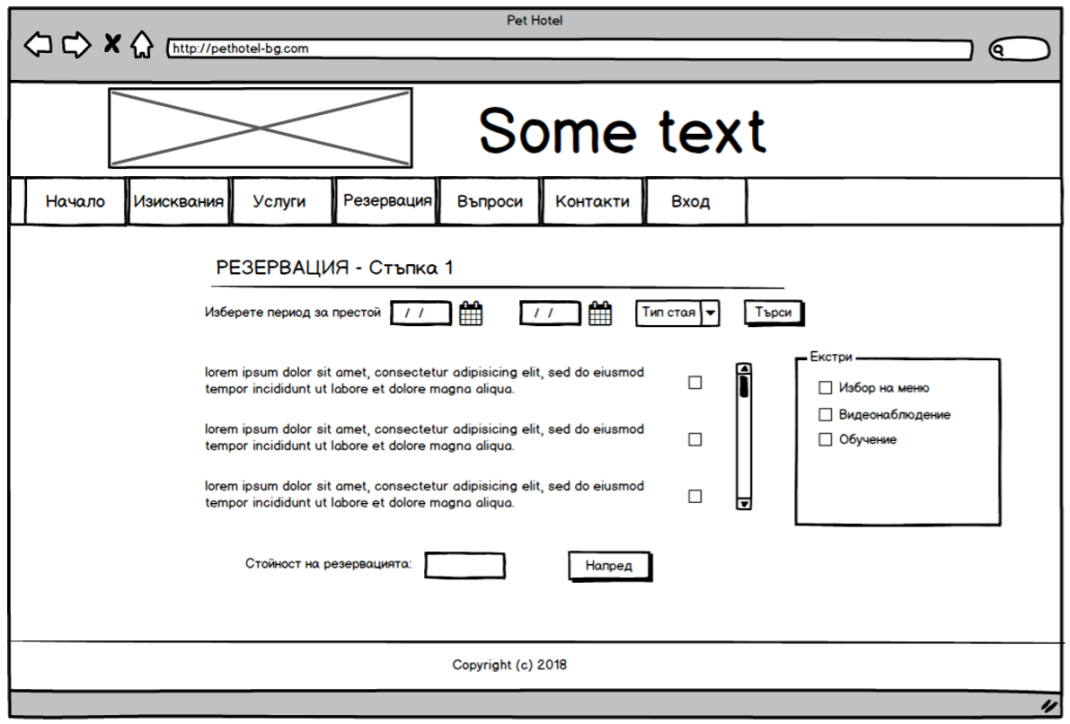
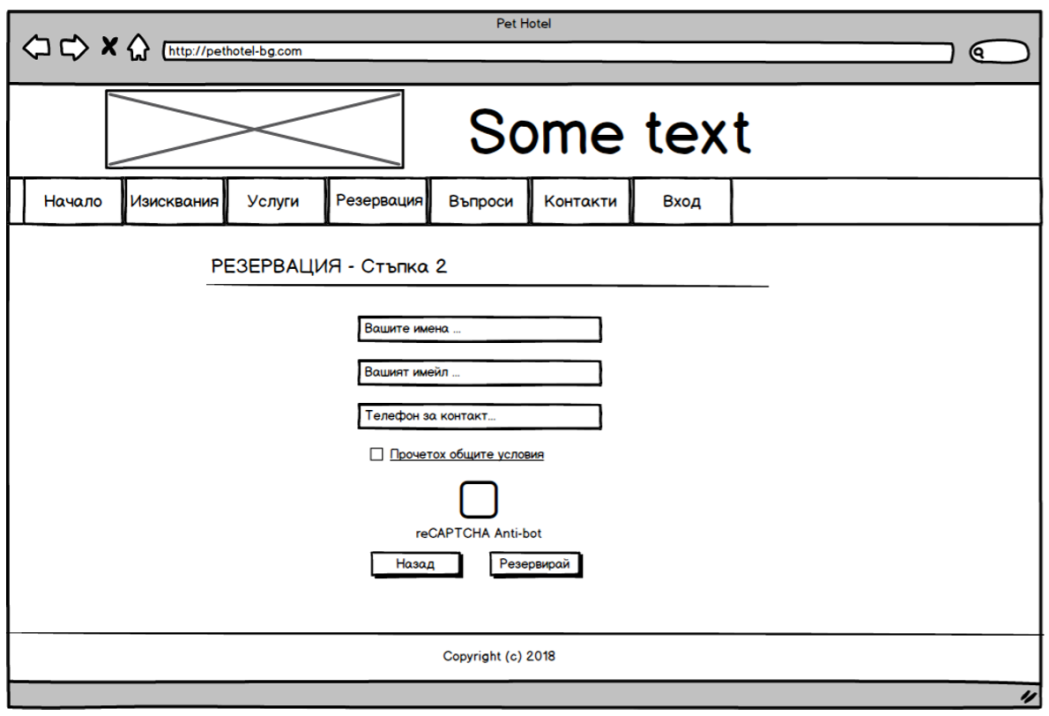
* Контакти



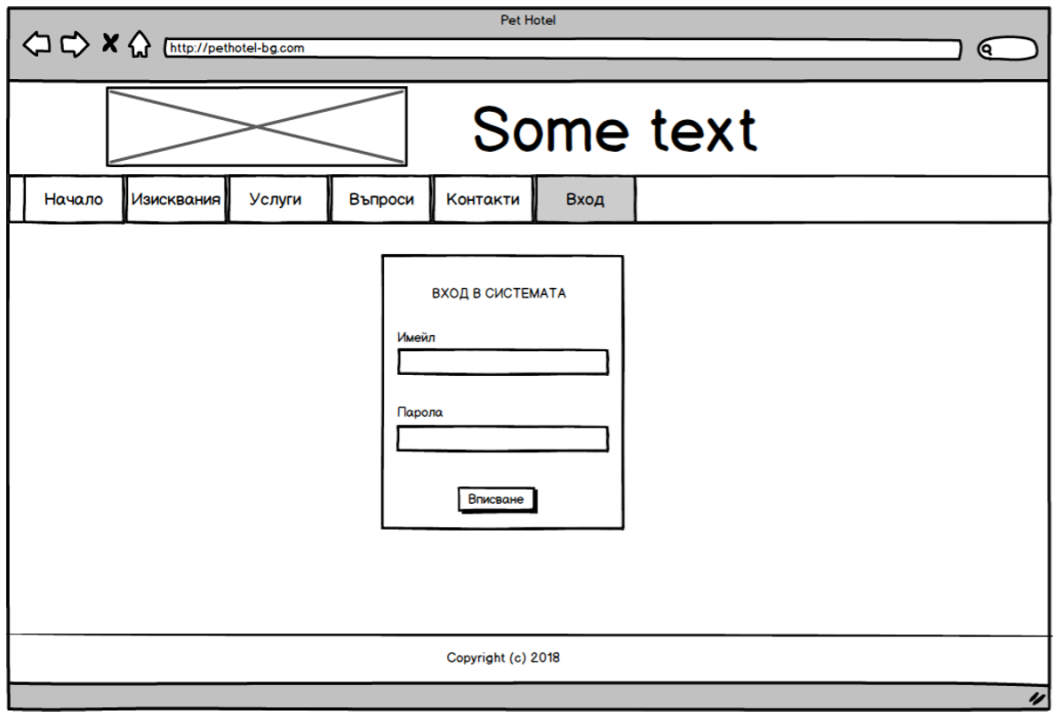
Екранът за контакти дава възможност за осъществяване на комуникация на анонимен потребител (потенциален клиент) с администратор на системата. Формата за контакти следва да валидира входните полета, така че да запазва данните в базата възможно най-чисти. На екрана е представена и карта с местоположението на „Kingdom Pets” с цел улеснение в откриването на обекта и въвеждането на адрес в навигационни системи.

* Резервация

Основен момент при изграждането на система за онлайн резеравции е формата за входни данни и филтрация. В текуща система формата за резервация условно е разделена на два екрана, които съдържат в себе си съответно:

- характеристики на типа резервация   
  
  
- лични данни на физическото лице в т.ч имена, имейл и телефон за контакт. Предвидено е интегриране на анти-бот система за преотвратяване на ботнет атаки към сървъра.   


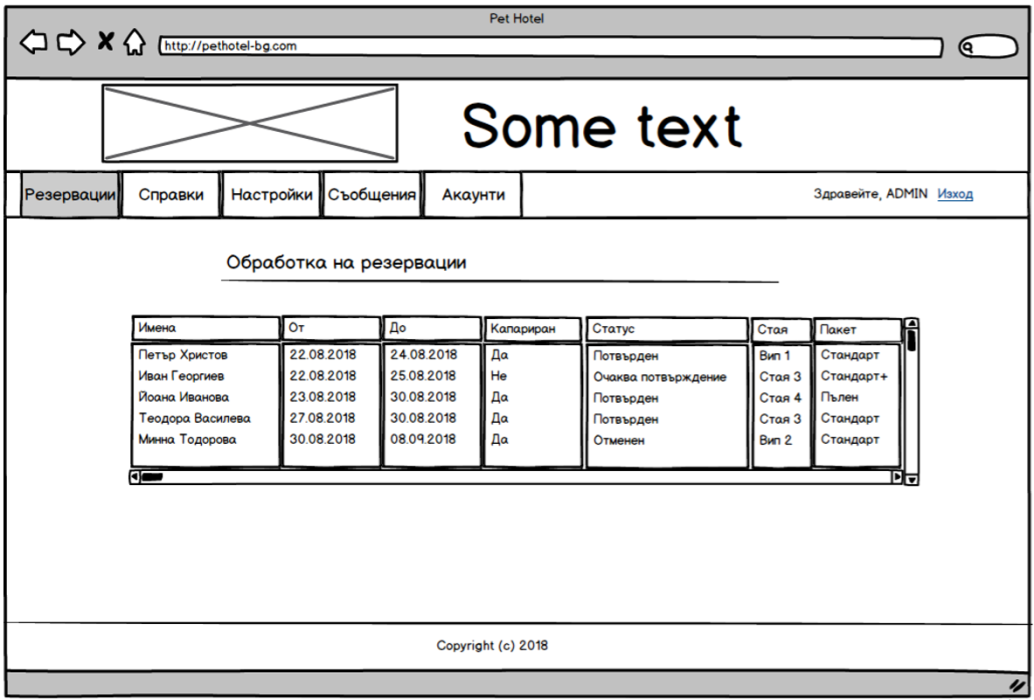
* Екран за вписване в системата



Вписването на потребител в системата включва въвеждането на две задължителни полета – за имейл и парола.

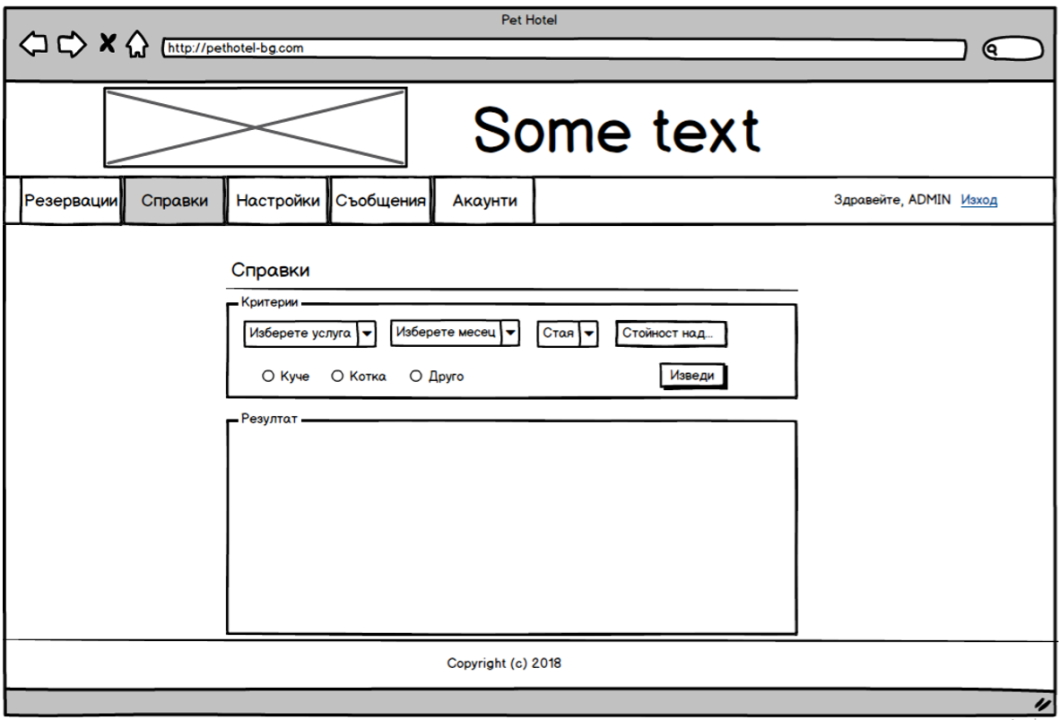
Административен панел

* Начален екран на Admin панел



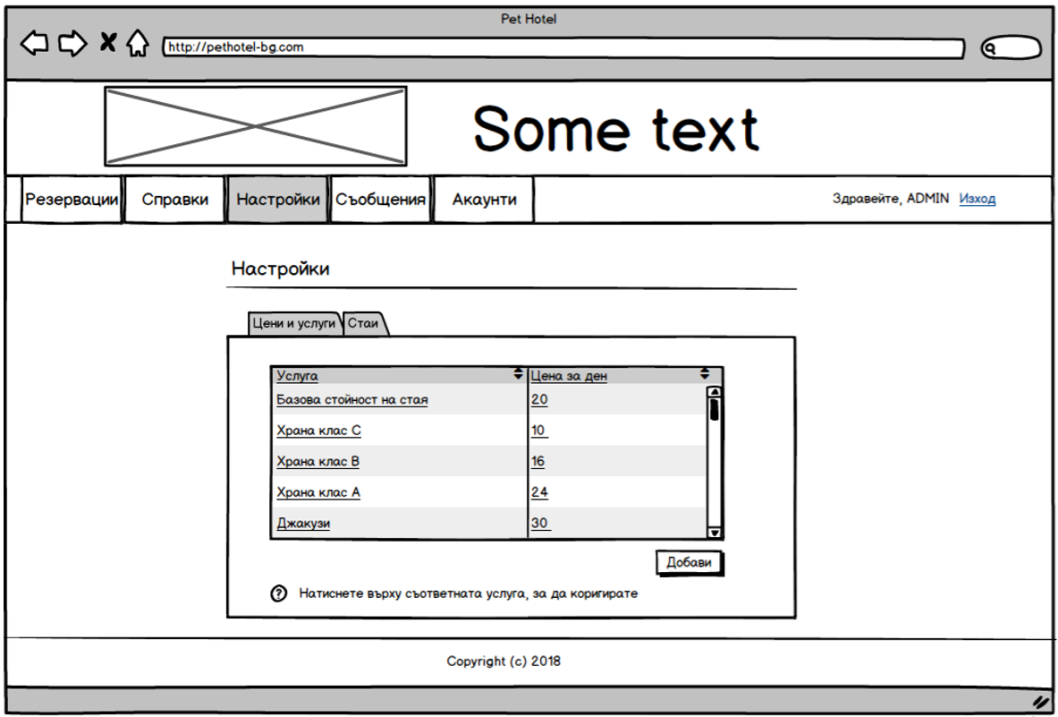
Началният екран в панела на администратора пренасочва към раздел Резервации, който дава възможност за обработка на резервациите – промяна на статуса, детайлен преглед .

* Справки



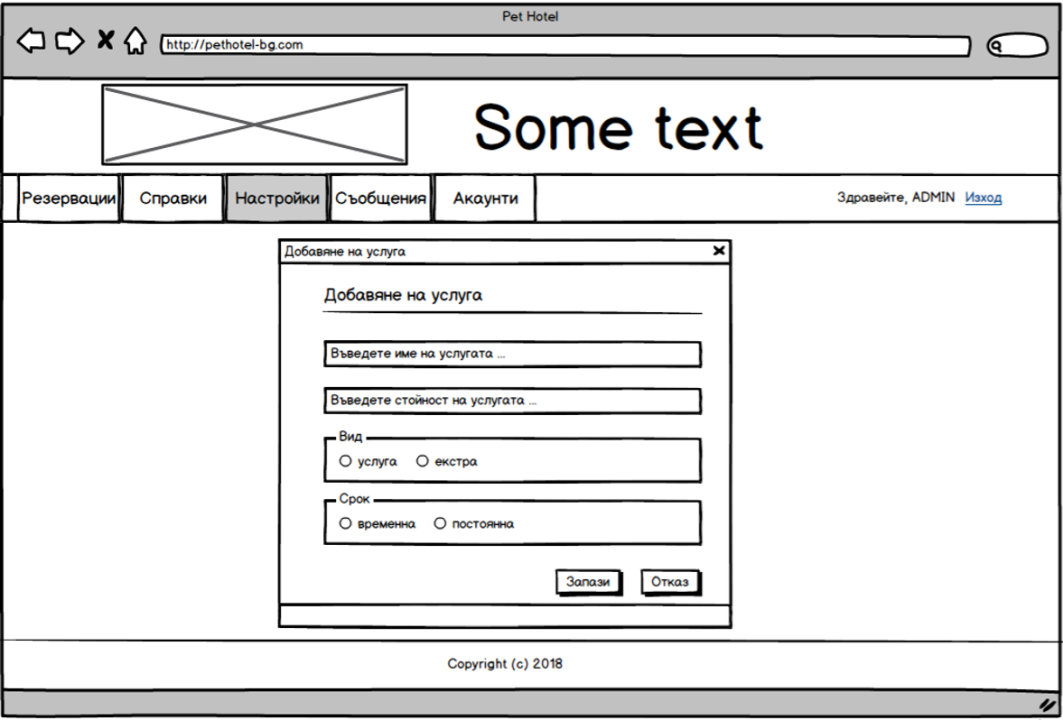
Екранът за справки предоставя възможност на администратора на системата да прави справки по вид услуга, по време, по домашен любимец.

* Настройки 🡪Цени и услуги



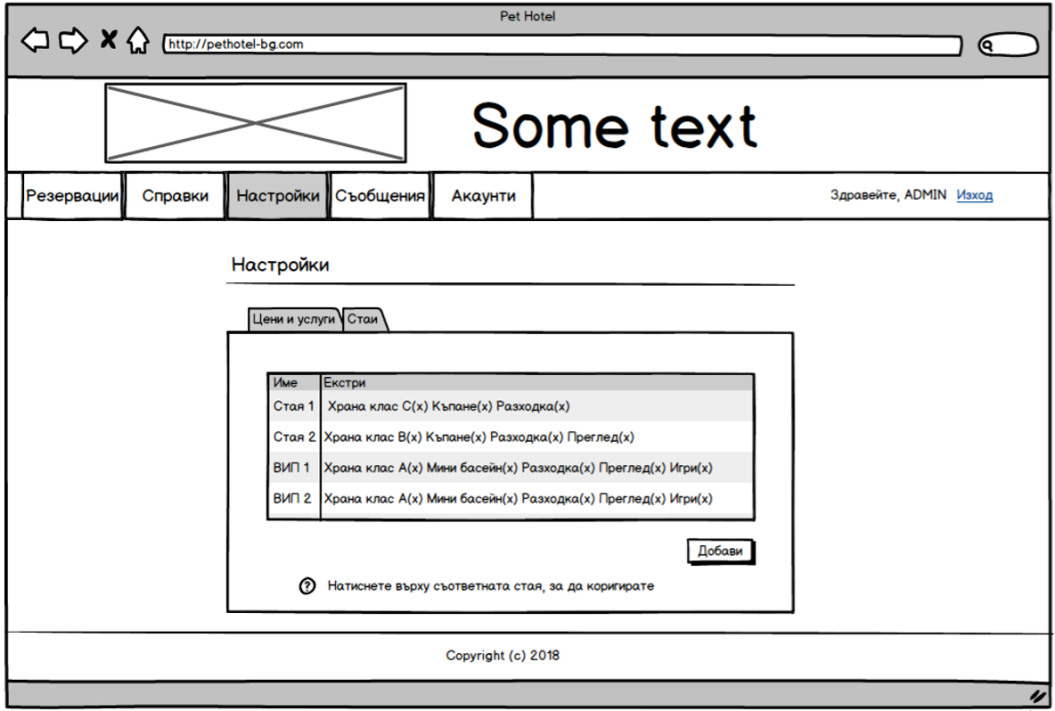
Менюто за настройки дава възможност на администратора на системата да добавя респективно коригира цените на услугите и на стаите. На екрана на потребителя се представят чрез изглед с раздели двете опции, като подканващо съобщение в долният край на всеки от екраните дава насока за неоходимото действие за извършване на корекция.

* Настройки 🡪 Цени и услуги 🡪 Добавяне/Редакция на услуга



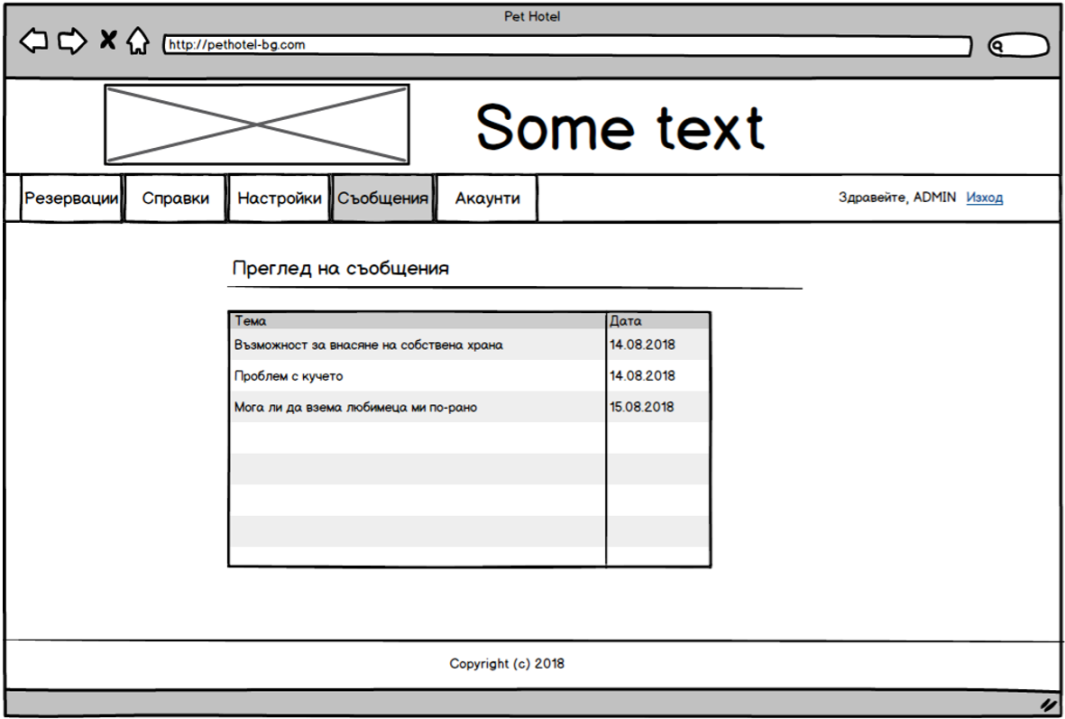
Добавяне/Редакция на услуги е екран с помощта, на който ще се въвеждат нови услуги или да се коригират съществуващите.

* Настройки🡪 Стаи



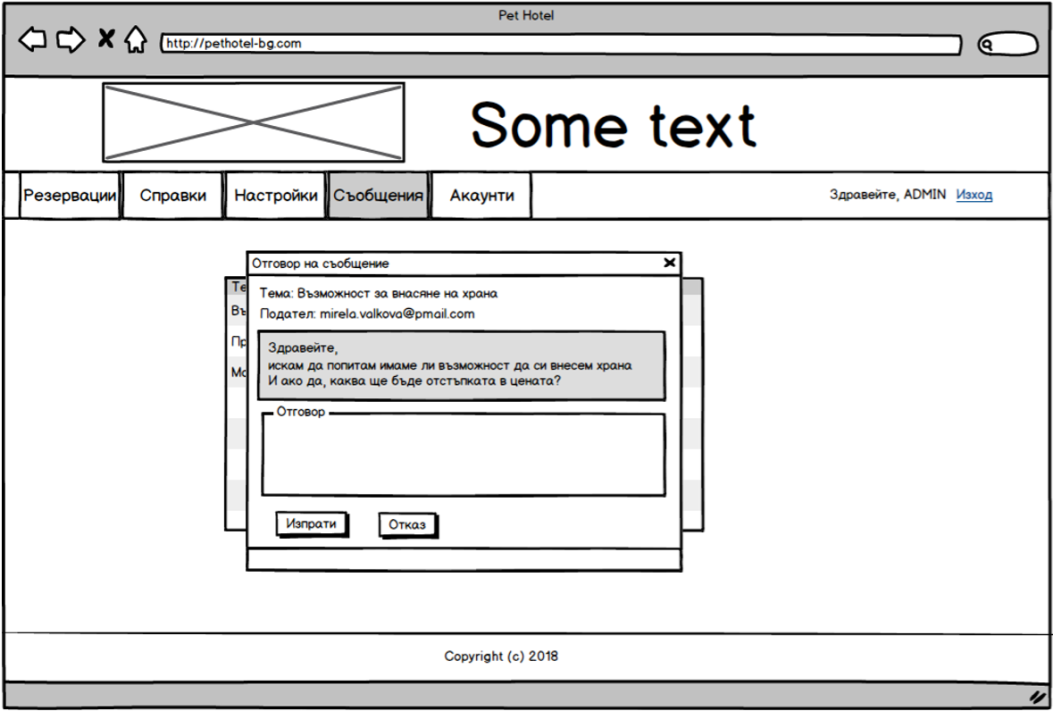
Подобно на екрана за преглед на цените и услугите и тук е представен чрез таблична форма списък с добавените към момента стаи с опции за добавяне на нова или корекция чрез натискане върху съществуващ запис от представения списък.

* Съобщения



Екранът за преглед на съобщения дава възможност на администратора да обработва постъпили запитвания, като детайлния преглед на съобщението изобразява подател, тема, дата на изпращане и съдържание.

* Съобщения 🡪 Отговор на съобщение



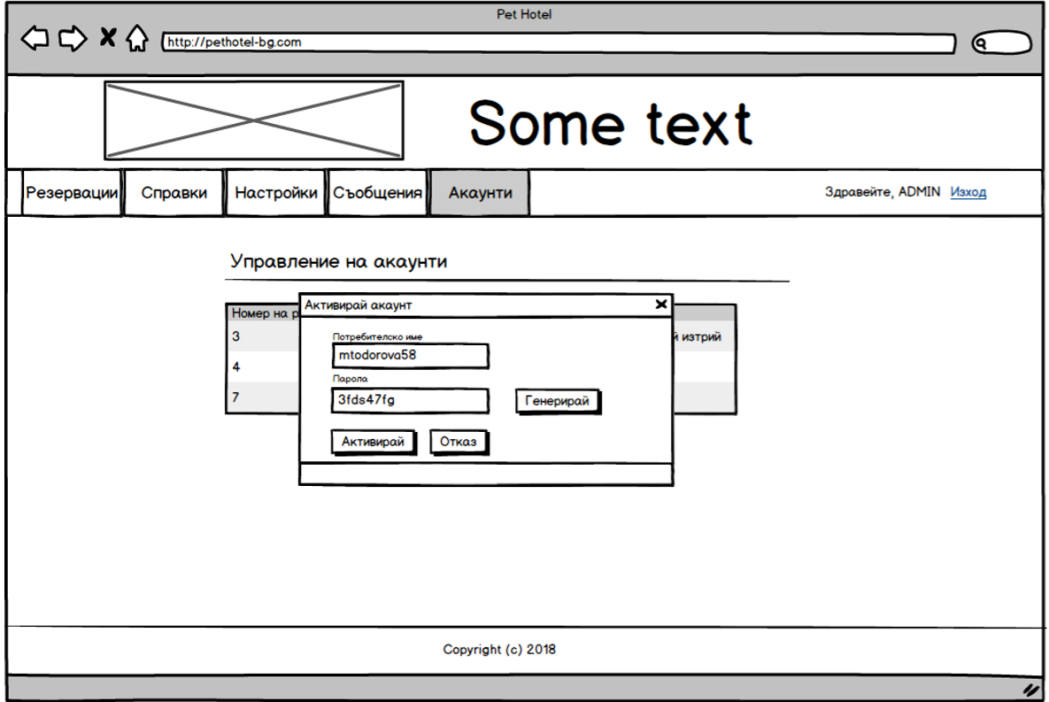
Отговор на съобщение дава възможност за отговор директно от панела в сайта без да е необходимо да се отварят нови раздели в браузъра или зареждане на нови сайтове. Това става възможност чрез интегриране на сървърна логика.

* Акаунти



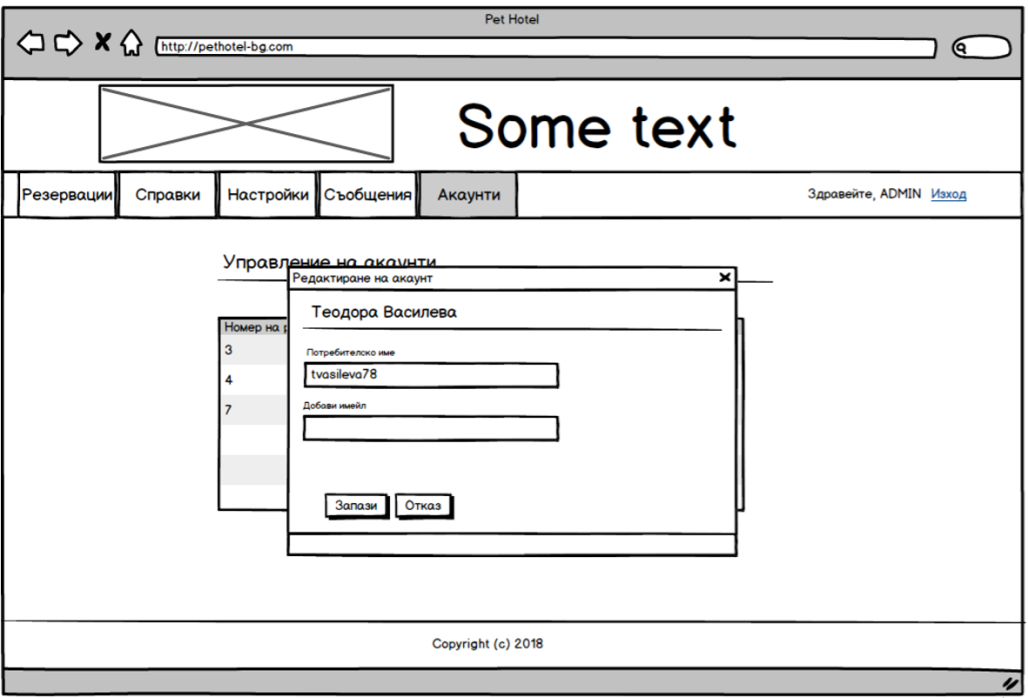
Екранът за управление на акаунти е изключително важен в текущата система, защото да ва възможност да се прегледат, активират, изтриват и редактират потребителски акаунти. Това е единствения начин за създаване на акунти в системата.

* Акаунти 🡪 Активиране на акаунт



Активиране на акаунт – чрез този екран админстратора дава възможност на потребителя направил дадена резервация да влиза в системата с генерирано от системата потребителско име и парола – паролата представлява комбинация от цифри и символи с дължина 8, а потребителското име се формира от първата буква от името и фамилията, последвани от две цифри, които представляват дължините на личните имена – пример: Иван Иванов става iivanov46. Ако има повтарящи се имена системата добавя цифри в края.

* Акаунти 🡪 Редакция на акаунт

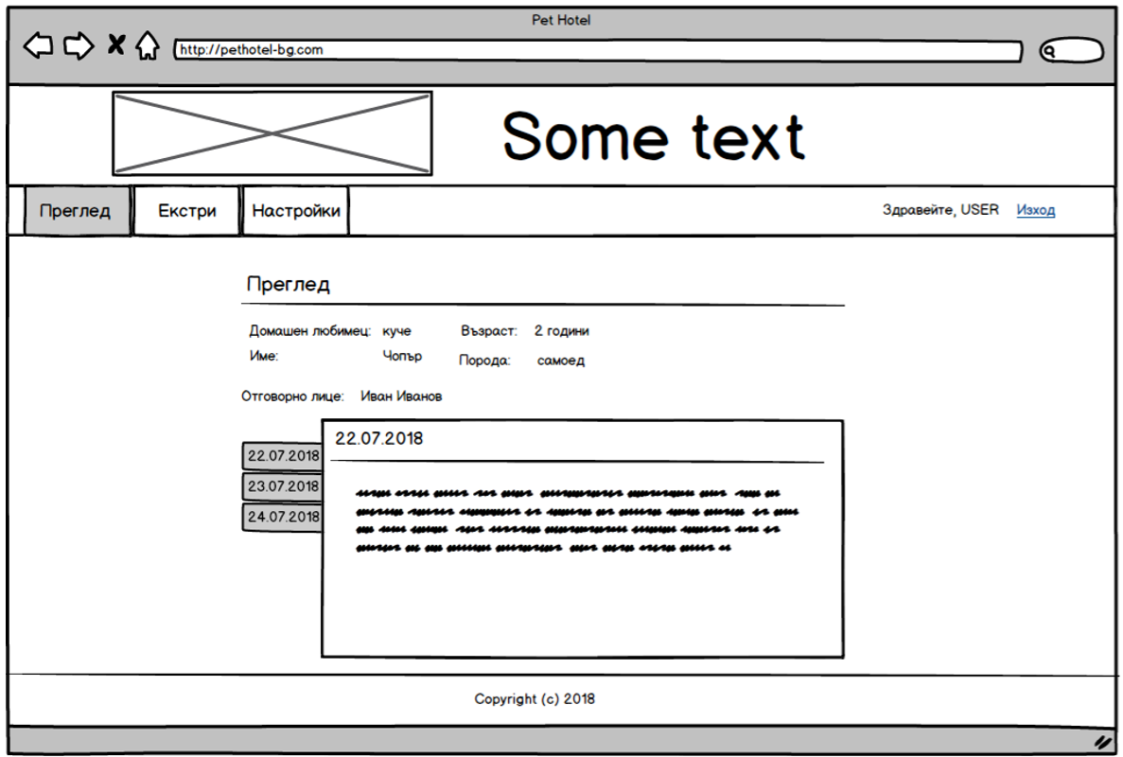


Екранът за редакция на акаунти дава възможност за коригиране на основна за потребителя информация, като имейл и други.

Потребителски панел

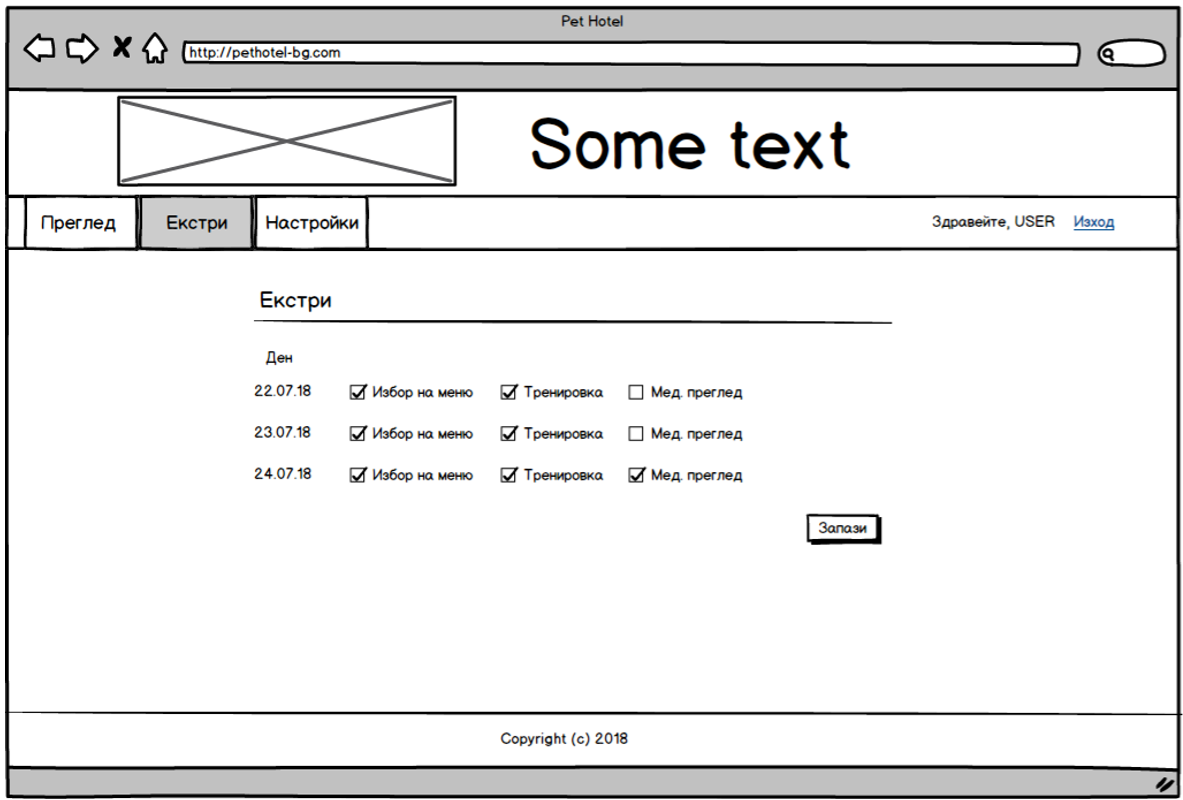
Този модул от системата е предназначен да обслужва нуждите на потребителя свързани със задаване на основни параметри във връзка с настаненият домашен любимец, като задаване на режим на трениране и хранене, преглед на резервацията, добавяне на нови екстри.

* Преглед



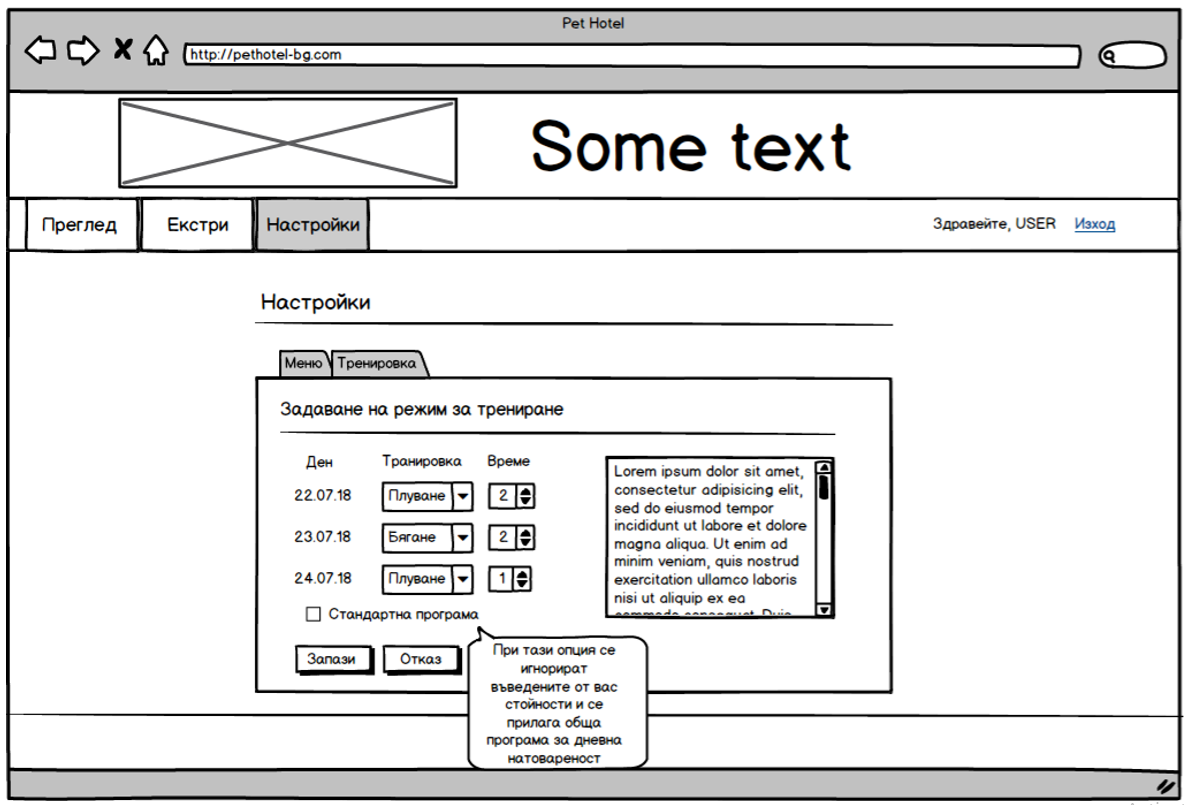
Екрана дава възможност за преглед на основна информация в т.ч име на любимеца отговорно лице, възраст , порода, както и детайли за дните на престой.

* Екстри



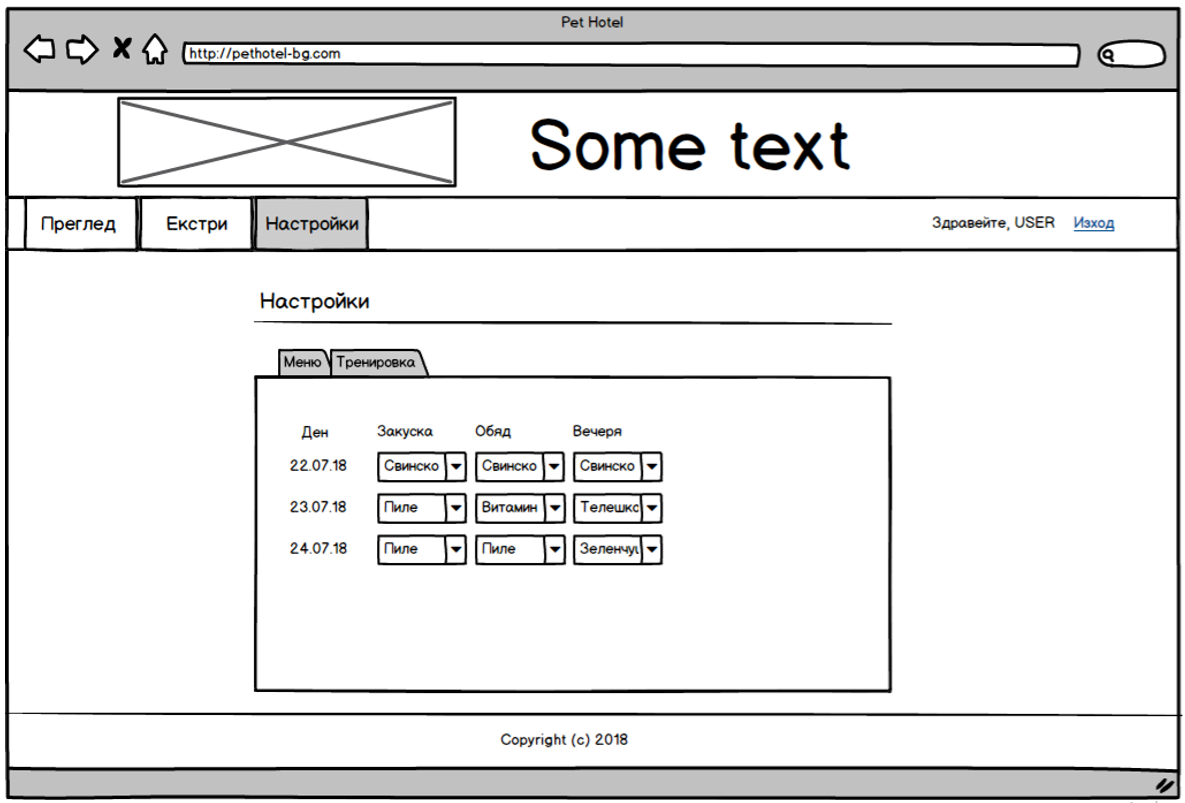
Представеният по-горе екран дава възможност за добавяне на допълнителни екстри към така формираната резервация.

* Настройки 🡪 Задаване на режим за трениране



Тъй като някои от домашните любимци имат специални режими на трениране системата дава възможност на собственика на животното да задава подобен режим с предоставяните от хотела тренировки. Максималното време за ден тренировка е 2 часа. Системата предоставя и възможност за избор на комплексна тренировка, която не акцентира върху някое специфично умение на домашния любимец.

* Настройки 🡪 Задаване на меню за престоя



*Забележка: В процеса на реализация част от екраните е много вероятно да претърпят промени с цел подобряване на представянето (performance) и усещането на потребителите при работата с приложенията(UX).*

## Проектиране на приложението

### Концептуален модел

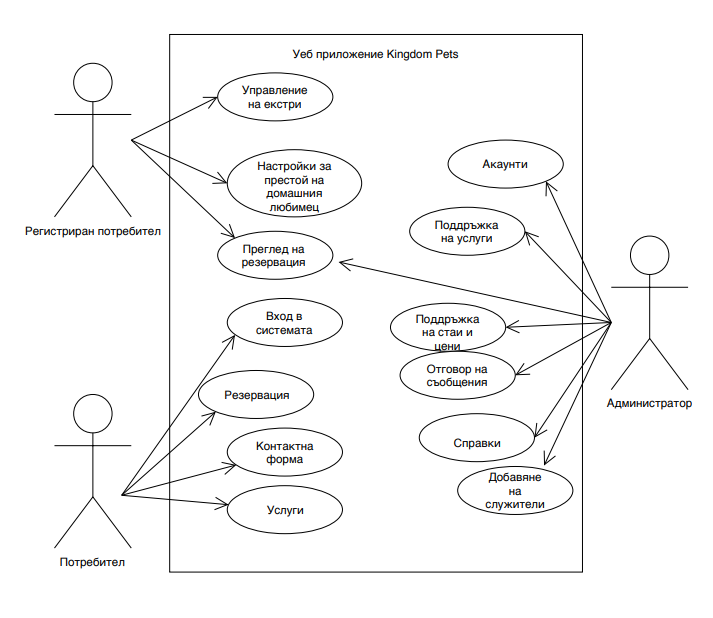


Фиг. ‑Концептуален модел на приложение "Kingdom Pets"

В основата на всяко уеб базирано приложение задължително присъства HTTP протокола и заявките, които клиентските приложения правят към сървъра. Приложението „Kingdom Pets” се придържа към тази концепция, като уеб модула комуникира с базата данни чрез WebAPI. По своята същност API-то представлява съвкупност от методи, които преди всичко извличат, респективно записват и обработват някакви данни от и към базата. За да се гарантира сигурността в процеса на извличане на информация е имплементиран т.нар JWT. Токън (жетон) чрез, който даден потребител се автентицира в системата и срещу, чието предоставяне той получава ресурси от базата от данни. Системата предоставя възможност за достъп до определени функционалности, които не зависят от съществуването на жетон т.нар. точки с публичен достъп. За всички други се изисква, като в структурата му се съдържа неговата роля в системата – USER, ADMIN и други важни за изграждането на интерфейса данни. За съхраняването на токъна се използва localStorage на браузъра, който дава възможност за съхранение на информация до 5 MB.

### UML диаграма

UML диаграмите служат за визуализиране, специфициране, конструиране и документиране на елементите на една софтуерна система. В текущо представеното приложение има няколко роли, като всяка от тях има възможност да достъпва определени функционалности чрез уеб приложението. На фиг.3.2 е представена UML диаграма на текущата разработка.



Фиг. ‑UML диаграма на уеб и мобилен модул

### Essential Use Case

Essential Use Case е специфичен вид UML диаграма, която често е наричана бизнес модел[[4]](#footnote-5) (business use case) и служи за таблично представяне на възможностите, които предоставя системата въз основа на желанията (намеренията) на участниците в процеса на работа с даденото приложение. В текущо представената разработка потребителската роля е една – приложението е проектирано за работа с един до няколко потребителя, като на този етап не е предвиден администраторски модул. За целите на по-добрата интерпретация в контекста на текущата Use Case диаграма вида приложение е възприет, като роля, която осигурява или не определените, в намеренията на крайния потребител, функционалности.

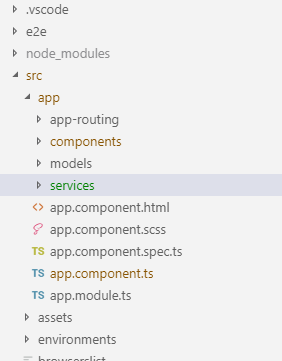
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Намерение |  | Роля | | Система |
| Потребител | Регистриран потребител | Администратор |
| Вписване в системата | Image result for yes | Image result for yes | Image result for yes | Предоставя подходящ интерфейс, чрез който потребителите да упражняват делигираните им права. |
| Достъп до услуги | Image result for yes | Image result for yes | Image result for yes |
| Добавяне/Редакция на стая | Image result for no | Image result for no | Image result for yes |
| Управление на акаунти | Image result for no | Image result for no | Image result for yes |
| Контактна форма | Image result for yes | Image result for yes | Image result for yes |
| Отговор на съобщения | Image result for no | Image result for no | Image result for yes |
| Справки | Image result for no | Image result for yes | Image result for yes |
| Потребителски настройки | Image result for no | Image result for yes | Image result for yes |

# Реализиране на системата

Въз основа на изработените функционален дизайн на Kingdom Pets беше реализирано уеб приложение с изчистен дизайн предоставящ всички предвидени възможности. В точки 4.1 и 4.2 от тази глава са изложени ключовите моменти в процеса на разработка на модулите съставляващи приложението и е направена съпоставка , с подобни на текущия, други продукти.

## Разработка

Kingdom Pets представлява уеб приложение, което образно може да бъде представено чрез 3 свързани модула – публичен (достъпен за всички), регистриран потребител и модул за управление на съдържанието (административен).  
Angular базираните приложения, каквото е и текущо разработеното имат специфична файлова структура.



Фиг. ‑ Файлова структура на Kingdom Pets

В директория app-routing е поместен файл, в който са описани всички пътища (адреси за достъп) до компонентите, от които е изградена системата. В папка services е изградена дървовидна структура от поддиректории, в които се намират файлове, в които са описани всички методи за достъп до сървърното приложение, респективно модификация на базата данни.

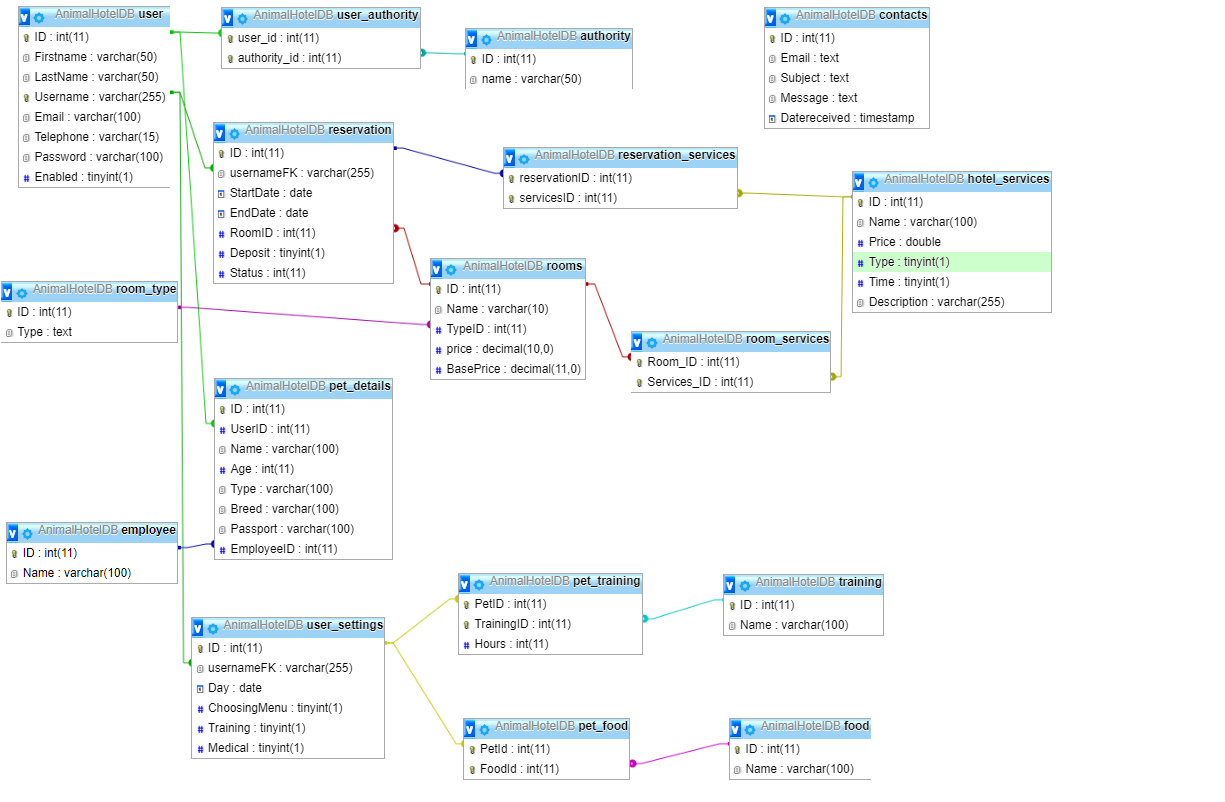
Приложенията със SPA[[5]](#footnote-6)архитектураса базирани на Model-View-Controller и като такива следва да се изгради система от модели, именно такива са позиционирани в директория models. Тъй като става дума за Angular базирана технология всички зависимости и настройки на проекта се правят във файла package.json. В текущата реализация се използват стандартен пакет от библиотеки и плъгини за тяхната употреба.В структурата на проекта са добавени и няколко външни библиотеки разработени от компанията Progress (по-конкретно Telerik). Kendo Angular Components са готови компоненти, които осигуряват възможност за изграждане на значително по-добър потребителски интерфейс, като идват с добре изградена структура от методи за конфигурация. Разработчиците са дали възможност за пълна промяна на облика на компонентите и интеграция във вече готов проект. Така изградената модулно ориентирана структура осигурява възможност за прилагането на стратегията lazy loading. Lazy loading е оптимизационна стратегия, която прави приложението много по-лесно за „зареждане“ на устройството на крайния потребител, като същността й се изразява в зареждане на отделни фрагменти от приложението, тогава, когато са необходими за извършване на определено действие на съответния екран. Това при равни други условия прави трафика по мрежата доста по-облегчен, а в условията на ограничения високоскоростен интернет на мобилните устройства е от изключително значение.



Фиг. ‑Списък с библиотеки (зависимости) на приложението Kingdom Pets

Въз основа на изложеното до тук в следващите няколко точки ще ви представя реализацията на трите, изграждащи приложението, модула, като ще бъде разгледана клиентска и сървърна част.

### База данни



Базата данни се състои от 17 таблици. Администратора има достъп до всички таблици може да добавя, изтрива и променя опциите в таблиците.

При нормалния потребител има достъп до настройките си, където задава храната, вида и времето на тренировка на домашния си любимец за всеки ден от дните, които е регистрирал.Потребителя разполага също с информацията за създадения акаунт, ако иска да има промяна то се съобщава до администратора.

Неригистрирани потребители имат достъп до три таблици които са контакти, където изпащат съобщение до администраторите, резервация където се очаква удобрениеот администратора, услуги където могат да видя какво предлага хотела за животни.

**Таблица *reservation -*** Таблица *reservation*  е проектирана да съдържа данните за резервациите. Нейните колони са:

* + *id* - първичен ключ, идентификационен номер на резервацията
  + *username* -външен ключ, идентификационен номер на потребителското име
  + *StartDate* - дата на постъпване
  + *EndDate* - дата на напускане
  + *RoomID* - идентификационен номер на стаята
  + *Deposit-*yes/no
  + *Status*

**Таблица *user\_authority-*** Таблица *user\_authority* е проектирана да свързва таблиците -*user* и *authority*. Нейните колони са:

* *user\_id-* първичен ключ, идентификационен номер на потребителя
* *authority\_id-* първичен ключ, идентификационен номер на автентификацията

**Таблица *authority-*** Таблица *authority* е проектирана да съдържа данните за автентификацията(права за достъп). Нейните колони са:

* *id-* първичен ключ, идентификационен номер на потребителя
* *name -* име на потребителя

**Таблица user*-*** Таблица *user*  е проектирана да съдържа данните за потребителите. Нейните колони са:

* *id-* първичен ключ, идентификационен номер на потребителя
* *Firstname-* име на потребителя
* *Lastname-*фамилия на потребителя
* *Username* - първичен ключ, потребителско име
* *email*
* *telephone*
* *enabled -* yes/no

**Таблица *reservation\_services-*** Таблица *reservation\_services* е проектирана да свързва таблиците - *reservation* и *room\_services*. Нейните колони са:

* *reservationID-* първичен ключ, идентификационен номер на резервацията
* *servicesID-* първичен ключ, идентификационен номер на услугата

**Таблица *room\_services-*** Таблица *room\_services* е проектирана да свързва таблиците - *reservation* и *room\_services*. Нейните колони са:

* *reservationID-* първичен ключ, идентификационен номер на стаята
* *servicesID-* първичен ключ, идентификационен номер на услугата

**Таблица *hotel\_services-*** Таблица *hotel\_services* е проектирана да съхранява информациия за услугите(екстри) , които предлага хотела. Нейните колони са:

* *ID-* първичен ключ, идентификационен номер на услугата
* *Name-* първичен ключ, идентификационен номер на услугата
* *Price-*цена на услугата
* *Type-*тип на ислугата
* *Time-*време на услугата
* *Description-*описание на услугата

**Таблица rooms*-*** Таблица *rooms*  е проектирана да съдържа данните за стаите. Нейните колони са:

* *ID-* първичен ключ, идентификационен номер на стаята
* *Name-* име на стаята
* *TypeID-FK,*тип на стаята- идентификационен номер
* *price*- цена на стаята
* *BasePrice*

**Таблица pet\_details*-*** Таблица *pet\_details*  е проектирана да съдържа данните за домашните любимци. Нейните колони са:

* *ID-* първичен ключ, идентификационен номер на стаята
* *UserID-* FK, идентификационен номер на потребителя
* *Name-*име на домашния любимец
* *Age*- възраст на домашния любимец
* *Type -* порода
* *Breed*
* *Passport -* лична карта на любимеца
* *EmployeeID-FK,* идентификационен номер на собственика

**Таблица room\_type*-*** Таблица *room\_type*  е проектирана да съдържа данните за типовете стаи. Нейните колони са:

* *ID-* първичен ключ, идентификационен номер на типа на стаята
* *Type -* тип на стаята(описание)

**Таблица employee*-*** Таблица *employee* е проектирана да съдържа данните за стопанина. Нейните колони са:

* *ID-* първичен ключ, идентификационен номер на типа на стопанина
* *Name -* име на собственика

**Таблица pet\_training*-*** Таблица *pet\_training* е проектирана да съдържа данните за тренировъчния режим. Нейните колони са:

* *petID-* първичен ключ, идентификационен номер на домашния любимец
* *TrainingID -* първичен ключ, идентификационен номер на тренировката
* *Hours-*продължителност на тренировката

**Таблица pet\_food*-*** свързващата таблица *pet\_food* е проектирана да съдържа данните за храната на домашния любимец. Нейните колони са:

* *PetID-* първичен ключ, идентификационен номер на домашния любимец
* *FoodID -* първичен ключ, идентификационен номер на тренировката

**Таблица training*-*** таблица *training* е проектирана да съдържа информация за тренировките. Нейните колони са:

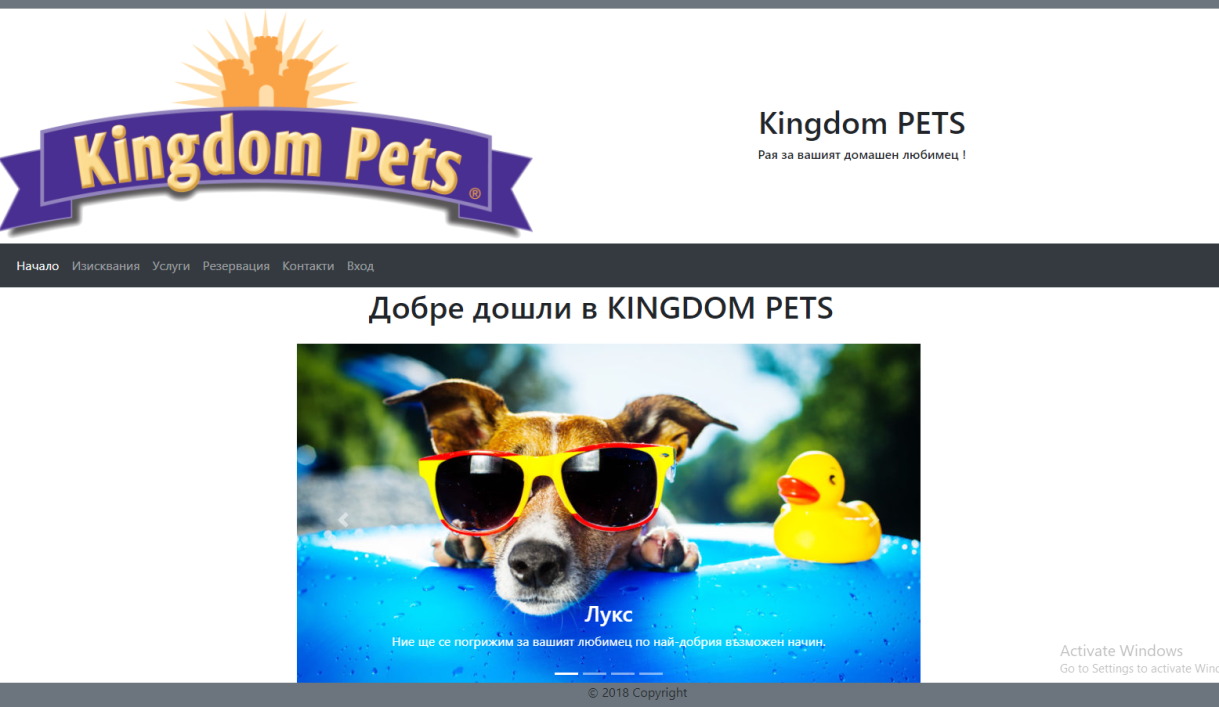
* *ID-* първичен ключ, идентификационен номер на тренировката
* *Name -* наименование на тренировката

**Таблица food*-*** таблица *food* е проектирана да съдържа информация за храната. Нейните колони са:

* *ID-* първичен ключ, идентификационен номер на храната
* *Name -* наименование на храната

### Публичен модул

Публичния модул на “Kingdom Pets” осигурява на потребителите възможност да достъпват преди всичко информационни страници от уеб приложението, както и комуникационен канал във връзка с резервации и други въпроси свързани с предлаганите услуги.

* Начален екран   
    
  В реализирания екран няма нищо по-различно от този описан във функционалния дизайн на приложението. По-особен е начинът, по който потребителят се пренасочва към страниците до, които е оторизиран. Чрез програмния фрагмент представен по-долу се прави проверка, като се декриптира жетона (ако съществува такъв) от локалното хранилище (localStorage).   
  private token: string;

constructor(private localStorageService: LocalstorageService) {

    this.isPublic = true;

    this.isAdmin = false;

    this.token = this.localStorageService.token;

    if (this.token !== undefined&&this.token !== ''&&this.token !== null) {

      this.isPublic = false;

      let tokenModel: TokenModel = this.localStorageService.parseToken();

      if (tokenModel.authorities === 'ADMIN') {

        this.isAdmin = true;

      }

      elseif (tokenModel.authorities === 'USER') {

        this.isAdmin = false;

      }

    }

  }

Използваме услугата, която се грижи за състоянието на токъна – LocalStorageService и чрез нейният метод parseToken() извличаме значимата за нас информация:   
  
parseToken(): TokenModel {

    const base64Url = this.\_token.split('.')[1];

    const base64 = base64Url.replace('-', '+').replace('\_', '/');

    const tokenModel = JSON.parse(window.atob(base64));

    let token = new TokenModel();

    token.authorities = tokenModel.authorities;

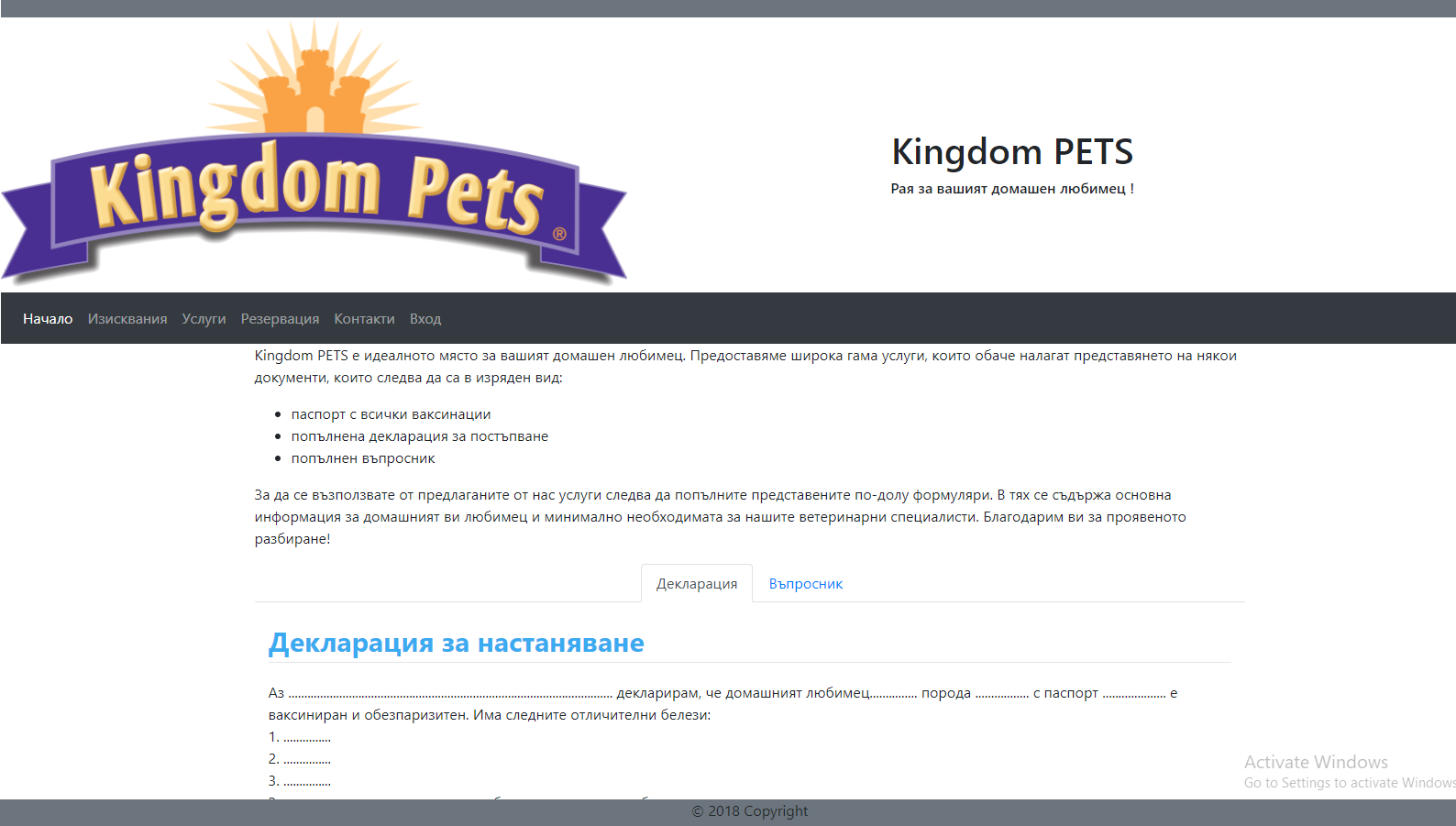
    token.fullName = tokenModel.fullName;

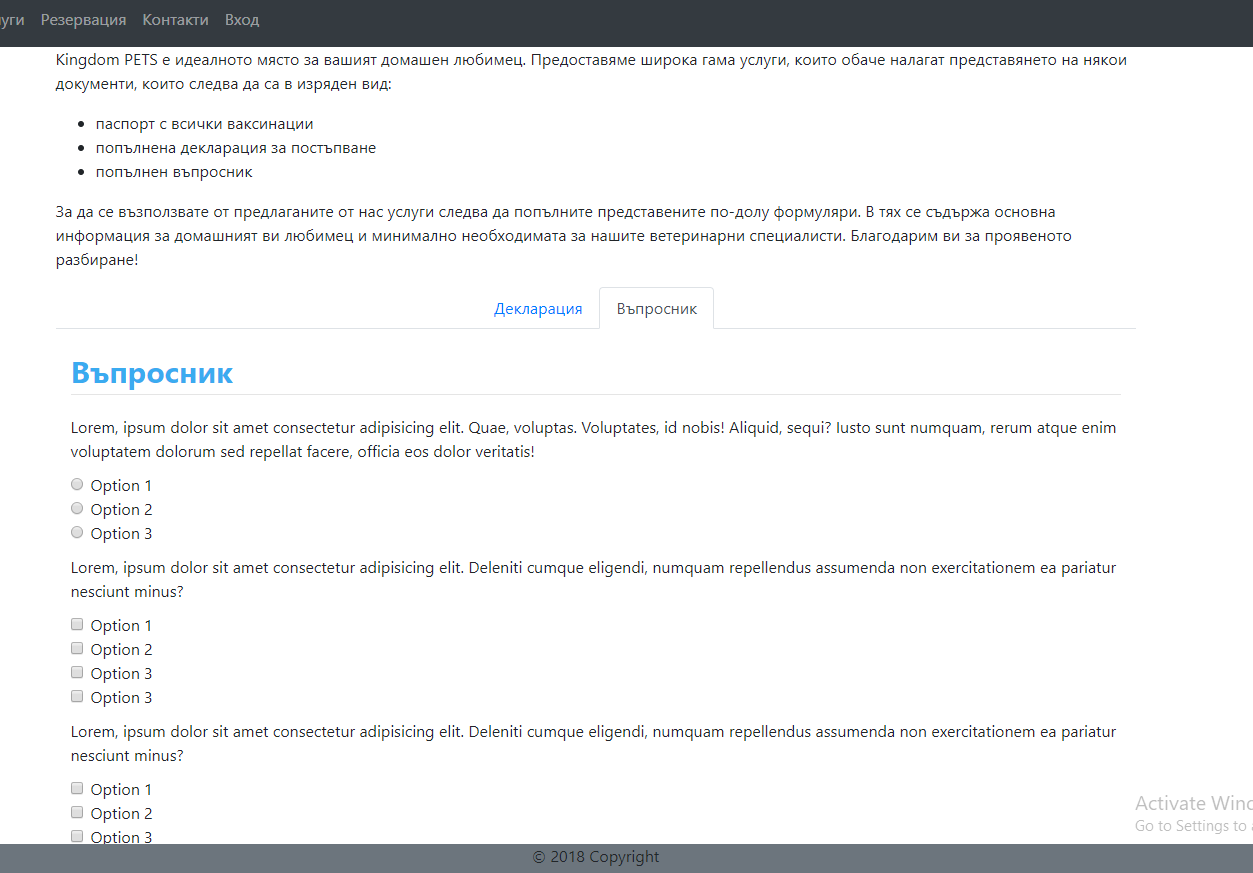
    token.email = tokenModel.email;

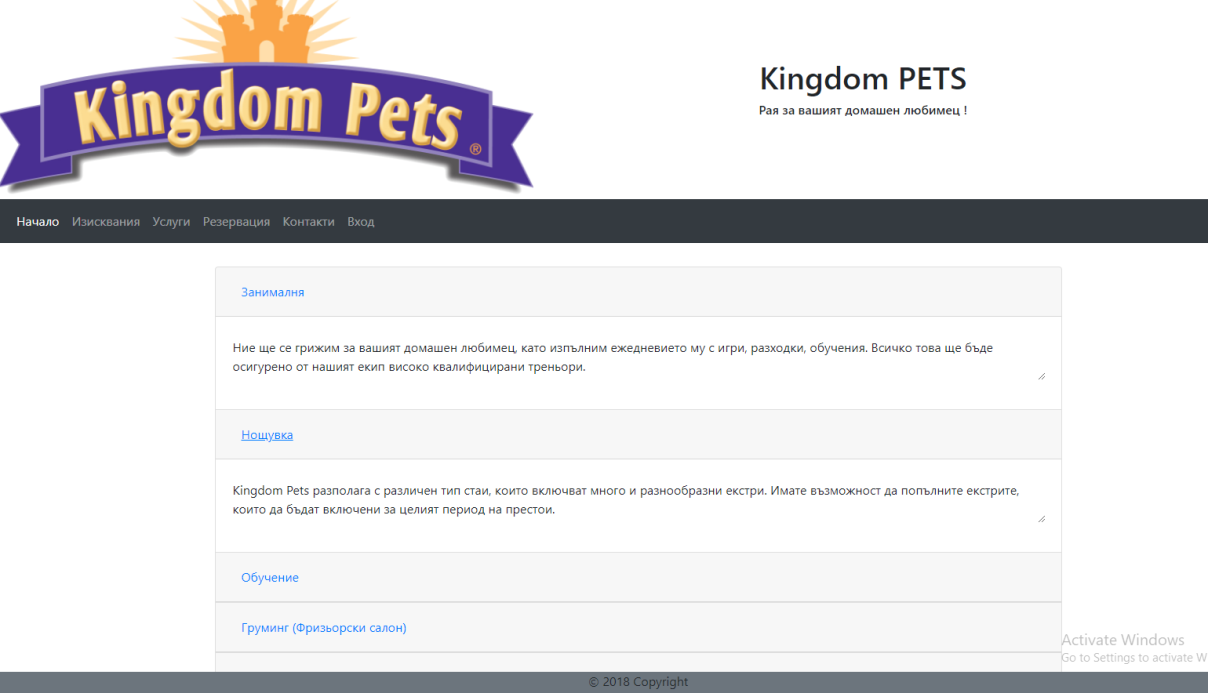
    token.user\_name = tokenModel.user\_name;

    return token;

  }

* Изисквания   
    
   Екраните за изисквания напълно отговарят на описаните в дизайна. На потребителя се дава възможност да изтегля документите, които са му необходими за настаняване на домашният любимец, както и допълнителна информация за хотела.   
    
  

Чрез въпросника се набява необходима информация, която е нужна за избор на отговорно лице, стая и други свързани с осигуряването на по-добър комфорт.   


Услуги   
  
Екранът за услуги не се различава съществено от планирания.  
  
  
  
 От гледна точка на клиентската част на приложението по-особеното е наличието на публичен метод за достъп до сървърното приложение, който връща списък с всички постоянни услуги, които предлага хотела. В контекста на темата услугите се делят условно на – постоянни и временни, като това разделение е в основата на формирането на т.нар екстри.

getHotelServicesOnly(): Observable<HotelProductViewModel[]> {

    returnthis.httpClient.get<HotelProductViewModel[]>(baseURL + 'api/hotel-services/services');

  }  
От сървърна гледна точка реализацията на тази функционалност става по следният начин:   
Създава се модел който се свързва с базата данни

@Entity

@Table(name="hotel\_services")

**publicclass** Extras {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Long id;

@Column(name="name")

**private** String name;

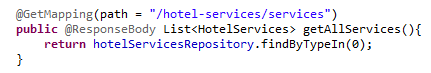
@Column(name="price")

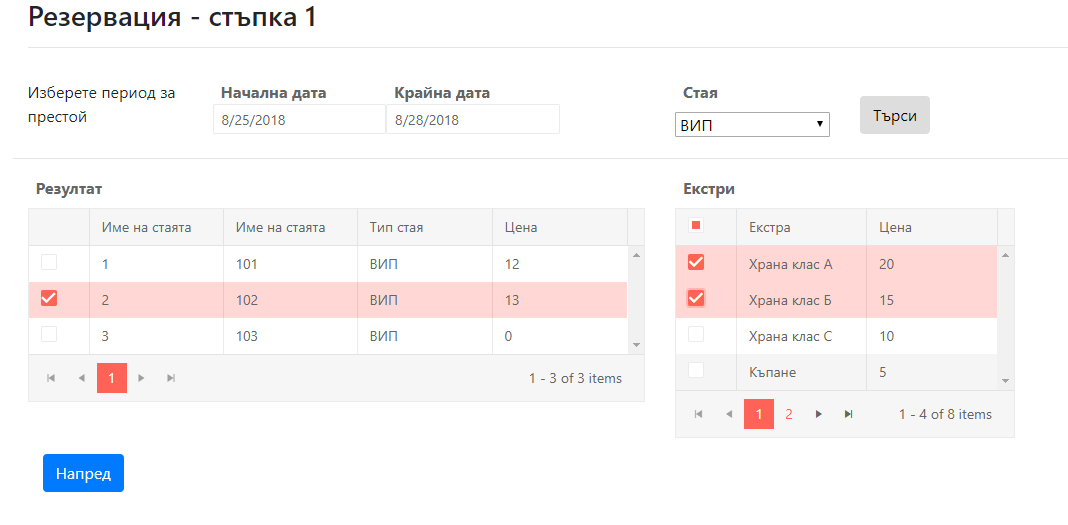
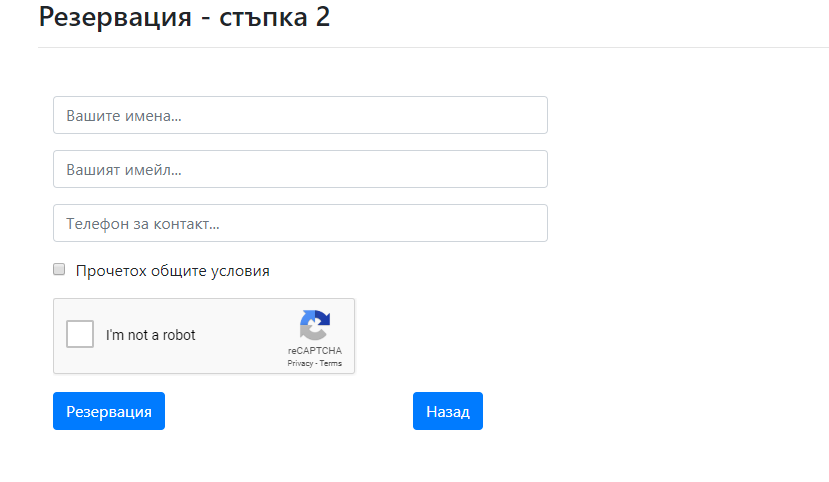
**private** Double price;

@Column(name="type")

**privateint**type;

Функцията която се достъпва е getAllServices, чрез findByTypeIn(0) се връщат всички услуги към потребителя



Резервация  
  
  
  
  
  
 Екраните за резервация са основни за приложението и като такива следваше да се реализират по начин, който да осигури максимално удобство за крайния потребител. Вмъкването на reCAPTCHA[[6]](#footnote-7) в комбинация с валидации на входящи полета, осигурява и чистотата на базата данни, както и защита от атаки към приложението.   
Интересен беше момента с внедряването на защитния механизъм reCAPTCHA.   
<divclass="form-group">

<re-captcha(resolved)="resolved($event)"siteKey="6LeDcmUUAAAAAEerpDa6E98kTEHKNvkyQ0mjQiaO"></re-captcha>

</div>  
 Чрез метода resolved() извличаме тайния ключ, който следва да се предостави, като част от тялото на заявката към сървъра, за осъществяване на резервацията. От там сървъра прави проверка и допуска или отхвърля постъпилата информация.

Прави се връзка към базата данни на таблица reservation, където се записва новата резервация

@Entity

@Table(name = "reservation")

**publicclass** Reservation **implements** Serializable {

**privatestaticfinallong*serialVersionUID*** = 1L;

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**private** Long id;

@Column(name = "usernamefk")

**private** String userName;

@Column(name = "startdate")

**private** Date startDate;

@Column(name = "enddate")

**private** Date endDate;

@Column(name = "roomid")

**privatelong**roomId;

@Column(name = "deposit")

**privateboolean**deposit;

@Column(name = "status")

**privateint**status;

Когато потребителя въведе необходимата информация тя се записва в три таблици, едната е при потребителите, втората при резервациите и третата в ифнормацията за животното

@RequestMapping(path = "reservations/create", method = RequestMethod.***POST***)

**public**@ResponseBody**void** sendUserReservation(@RequestBody Register register, @RequestParam(name="g-recaptcha-response") String recaptchaResponse, HttpServletRequest request){

String[] getTempName = register.getName().split(" ");

Reservation reservation = **new** Reservation();

User user = **new** User();

List<User>arrUser = **new** ArrayList<User>();

List<Authority>authorities = **new** ArrayList<Authority>();

Authority auth = **new** Authority();

auth.setId((**long**) 2);

auth.setName(AuthorityName.***ROLE\_USER***);

user.setFirstname(getTempName[0]);

user.setLastname(getTempName[1]);

user.setEmail(register.getEmail());

user.setTelephone(register.getPhoneNumber());

user.setAuthorities(authorities);

reservation.setStartDate(register.getStartDate());

reservation.setEndDate(register.getEndDate());

reservation.setRoomId((**long**) register.getRoomId());

CreatingUserUtil userCreation = **new** CreatingUserUtil();

String username = userCreation.CreateUsername(register.getName());

user.setUsername(username);

reservation.setUserName(username);

user.setPassword(username);

user.setEnabled(**false**);

arrUser.add(user);

auth.setUsers(arrUser);

authorities.add(auth);

userRepository.save(user);

reservationRepository.save(reservation);

PetDetails petDetails = **new** PetDetails();

petDetails.setUserId(user.getId());

petDetails.setName("");

petDetails.setAge(0);

petDetails.setType("");

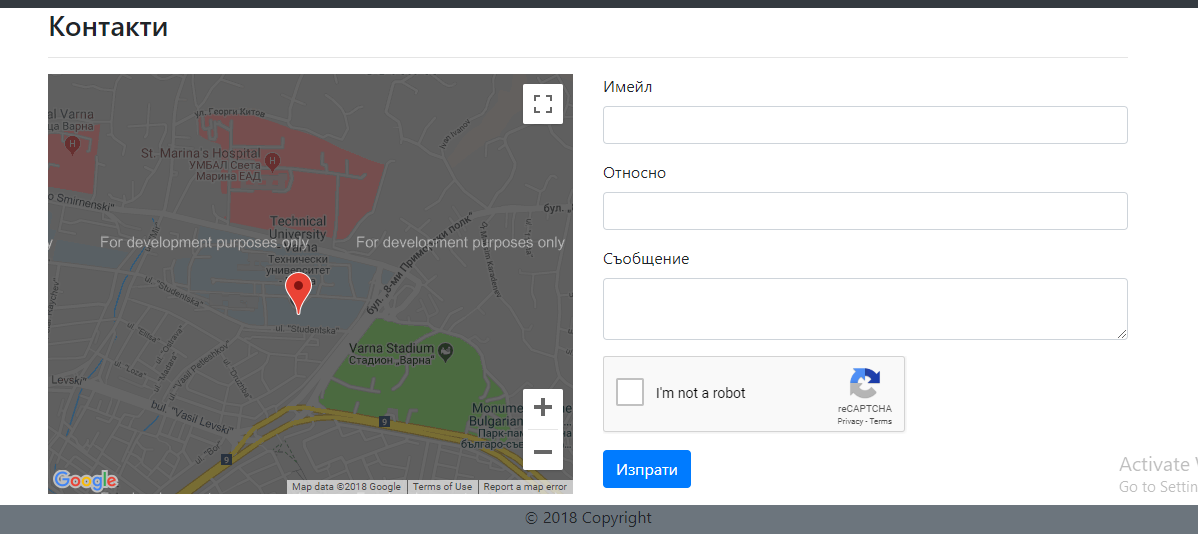
petDetails.setBreed("");

petDetails.setPassport("");

petDetails.setEmployeeId(1);

petDetailsRepository.save(petDetails);

}

Контактна форма   
  
  
  
 Както и във функционалния дизайн и в реализацията на приложението е добавена карта, защитен механизъм reCAPTCHA и валидация на всички полета, които са задължителни и съответно имат някакви ограничения в символите (имейла). Системата извежда съобщения за грешки в случаите, когато задължително поле е празно или въведената стойност е неправилна. Следващия фрагмент код демонстрира програмната реализация на валидациите на полетата в контактната форма.  
  
 this.contactForm = this.fb.group({

      email: ['',

        [

          Validators.required,

          Validators.pattern(/^[a-z0-9\_]+(\.[\_a-z0-9]+)\*@[a-z0-9-]+(\.[a-z0-9-]+)\*(\.[a-z]{2,15})$/ig)

        ]

      ],

      subject: ['',

        [

          Validators.required

        ]],

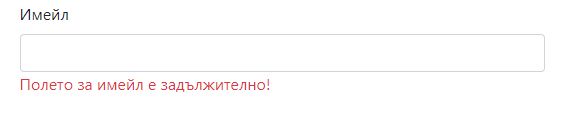
      message: ['',

        [

          Validators.required

        ]]

});



Фиг. ‑ Валидиране на полета за въвеждане на данни и извеждане на съобщения за грешка.

След попълване на данните в програмата се извиква метода onSubmit() , който проверява тайния ключ на reCAPTCHA и валидацията на формата и при преминаване на проверките прави заявка към сървъра.   
if (this.contactForm.valid) {

      if (this.captchaToken !== ''&&this.captchaToken !== undefined) {

        const contactModel: ContactBindingModel = this.contactForm.value;

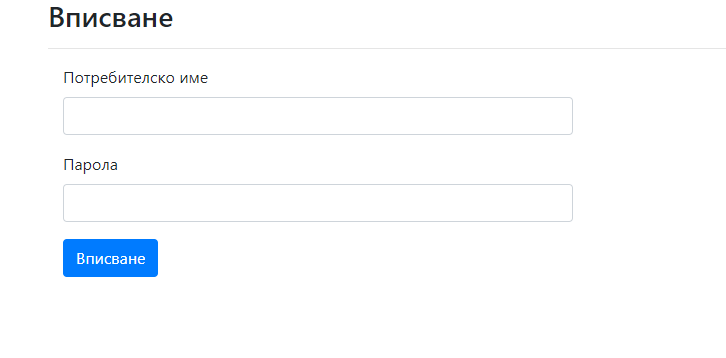
        this.contactSubscription.add(this.publicService.sendMessage(contactModel, this.captchaToken).subscribe());

      }

    } else {

      thrownew Error('Формата не е попълнена коректно!');

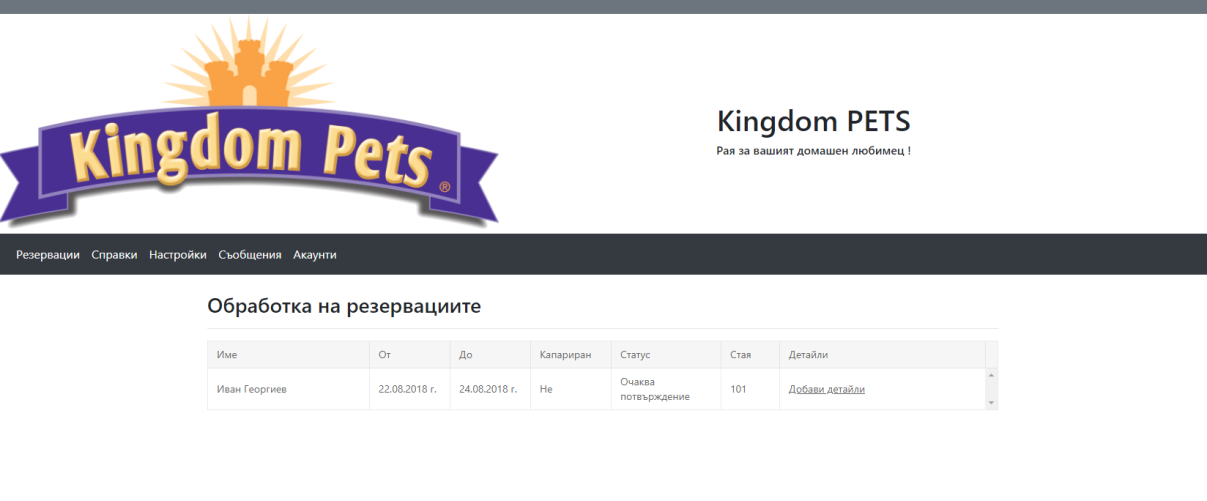
    }

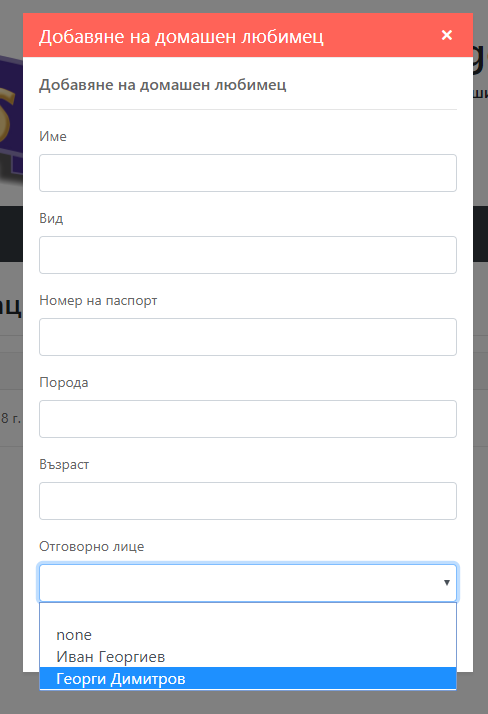
* Вход в системата   
    
    
   Екрана за вписване в системата не се различава от планирания, като и тук подобно на другите форми за въвеждане на данни са валидирани полетата за потребителско име и парола.

### Административен (управленски) модул

Важното при изграждането на система за онлайн резервации е наличието на модул за управление, който да бъде прост и лесен за възприемане от мениджърите. Реализирания управленски модул дава възможност на администратора на системата да управлява всеки аспект от дейността (респективно всяка част от приложението). По своята същност се доближава изключително до функционалния дизайн, но в процеса на реализация с цел оптимизиране и разширяване на възможностите модула претърпя промени.

* Обработка на резервации



Реализираният екран дава възможност за бърз преглед и редакция на резервации. Табличният изглед е реализиран чрез т.нар grid view, което е част от външния пакет kendo-angular-grid.   
  


Фиг. ‑Диалогов прозорец за добавяне на детайли към дадена резервация

Интересно е да се отбележи, че формата на датите е български. Това стана възможно чрез обработка на потока от информация получен при реализираната при първоначално зареждане на екрана GET заявка.   
  
parseToLocaleDate(date: Date): string {

     return date.toLocaleDateString('bg-BG');

   }

Тъй като отново е налице форма за попълване беше необходимо правилно да се създаде системата от валидации. Такава стана възможна с помощта на метода generatePetFormGroup(), като фрагмента код по-долу илюстрира същината му:  
let userIdFormControl = this.formBuilder.control(this.pet.userId);

    let nameFormControl = this.formBuilder.control('', [ Validators.required]);

    let typeFormControl = this.formBuilder.control('', [Validators.required]);

    let passportFormControl = this.formBuilder.control('', [Validators.required]);

    let breedFormControl = this.formBuilder.control('');

    let ageFormControl = this.formBuilder.control('');

    let employeeIdFormControl = this.formBuilder.control('', [Validators.required]);

От сървърна гледна точка

Модела е подобен като останалите  
  
@Entity

@Table(name = "pet\_details")

**publicclass** PetDetails {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "id")

**privatelong**id;

@Column(name = "userid")

**privatelong**userId;

@Column(name = "name")

**private** String name;

@Column(name = "age")

**privateint**age;

@Column(name = "type")

**private** String type;

@Column(name = "breed")

**private** String breed;

@Column(name = "passport")

**private** String passport;

@Column(name = "employeeid")

**privateint**employeeId;

Функцията приема JSON обект и го записва в базата данни

@RequestMapping(path = "/reservations/{reservationId}", method = RequestMethod.***POST***)

@PreAuthorize("hasRole('ADMIN')")

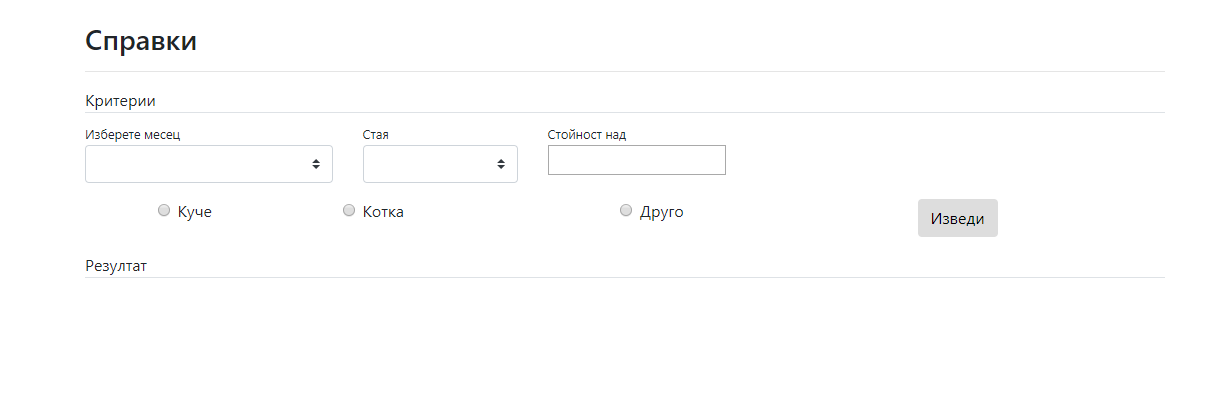
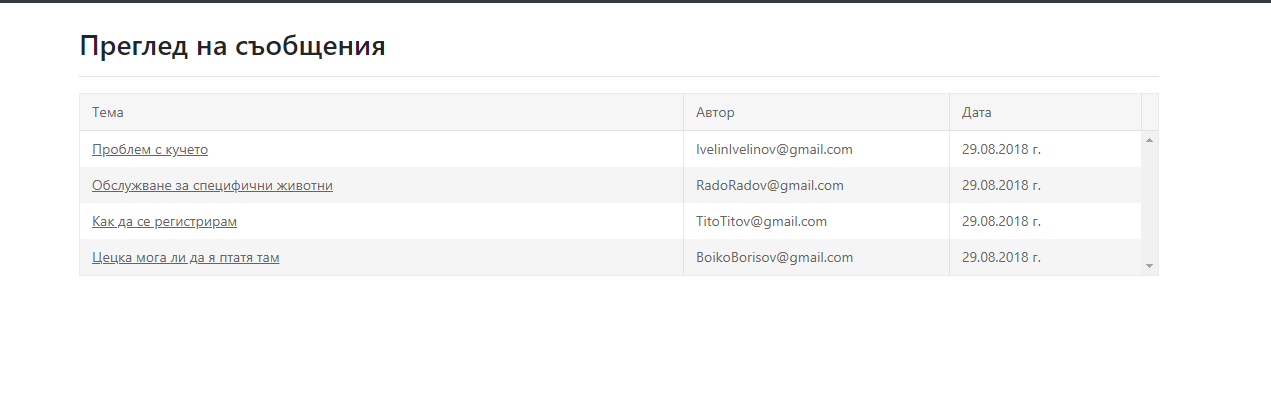
**publicvoid** getReservationDetail(@PathVariable Long reservationId, @RequestBody PetDetails petDetails) {

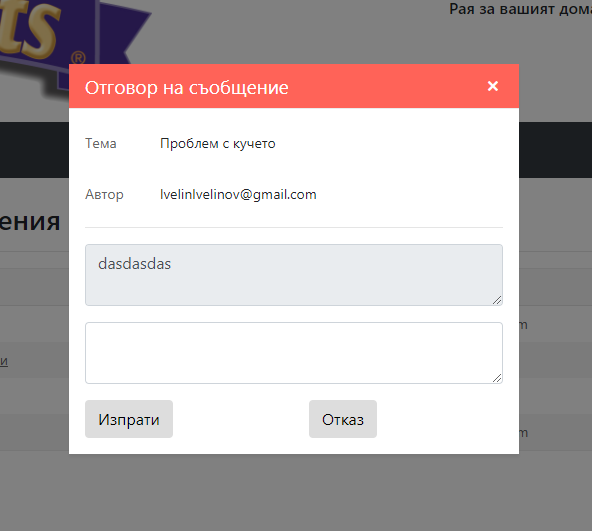
PetDetails petTemp = petDetailsRepository.findByUserIdIn(reservationId);

petDetails.setId(petTemp.getId());

petDetailsRepository.save(petDetails)

}

* Справки  
    
    
   За реализацията на екрана беше направен модел, чийто член променливи са свързани с полетата, които потребителят има възможност да модифицира, а именно – месец, стая, стойност и вид домашен любимец. Прави се заявка към сървъра и като резултат се получава същият модел – списък с всички отговарящи на критериите резервации.
* Настройки  
    
   Този екран е изключително важен за управленския модул, тъй като дава възможност да се коригират всички аспекти от приложението общи за трите модула. В табличен вид и с помощта на диалогови прозорци администратора има възможност да добавя/коригира и изтрива – цени и услуги, стаите в хотела, както и лицата отговорни за домашните любимци.   
    
  
* Съобщения  
    
  

Администратора има възможност да преглежда получените съобщения и да отговаря на тях чрез изградена система от диалогови прозорци и модели към тях.   
  
  
 if (this.answerMessage.message !== '') {

      this.subscription = this.adminService.sendAnswerMessage(this.answerMessage).subscribe();

      this.answerMessage = new ContactBindingModel();

      this.isDialogOpened = false;

    } else {

      alert ('Не може да изпратите празно съобщение!');

    }

От страна на сървъра се изпраща съобщение до потребителя чрез пощата която е въвел, с командата PreAuthorize се проверява дали потребителя е администратор ако е се изпълнява функцията.

@RequestMapping(path = "/admin/messages/answer", method = RequestMethod.***POST***)

@PreAuthorize("hasRole('ADMIN')")

**publicvoid** sendEmailToUser(@RequestBody SendMessage sendMessage) {

Contacts contact = contactRepository.findByIdIn(sendMessage.getMessageId());

Properties props = System.*getProperties*();

String host = "smtp.gmail.com";

props.put("mail.smtp.starttls.enable", "true");

props.put("mail.smtp.host", host);

props.put("mail.smtp.user", "platishell@gmail.com");

props.put("mail.smtp.password", "2365050aasddsa");

props.put("mail.smtp.port", "587");

props.put("mail.smtp.auth", "true");

Session session = Session.*getDefaultInstance*(props);

MimeMessage message = **new** MimeMessage(session);

**try** {

message.setFrom(**new** InternetAddress("platishell@gmail.com"));

InternetAddress toAddress;

toAddress = **new** InternetAddress(contact.getEmail());

message.addRecipient(Message.RecipientType.***TO***, toAddress);

message.setSubject(contact.getSubject());

message.setText(sendMessage.getMessage());

Transport transport = session.getTransport("smtp");

transport.connect(host, "platishell@gmail.com", "2365050aasddsa");

transport.sendMessage(message, message.getAllRecipients());

transport.close();

}

**catch** (AddressException ae) {

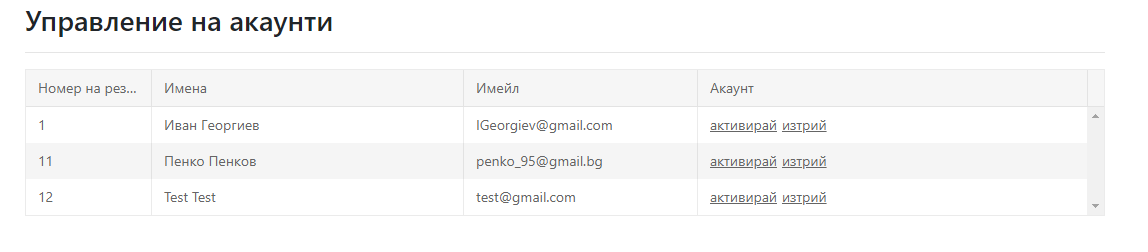
ae.printStackTrace();

}

**catch** (MessagingException me) {

me.printStackTrace();

}

* Акаунти  
  

Екранът за управление на акаунти е изключително важен, тъй като до голяма степен функционалността на реалния клиент (този потвърдил резервацията), зависи от успешното активиране на акаунта. От този панел администратора на системата има възможност да активира, изтрива и редактира акаунти. Акаунти в системата се активират само от тук. По дизайн системата няма отворени резервации идеята, на която се базира това е преди всичко защита и съответно по-лесно управление на уеб приложението.   
Интересен бе момента с активирането на акаунтите, тъй като там следваше да се направи метод, който да генерира парола с дължина 8 символа от случайни символи в т.ч цифри и букви. Фрагмента код по-долу демонстрира именно този метод.   
  
generatePassword(): string {

    let password = '';

    let possible = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

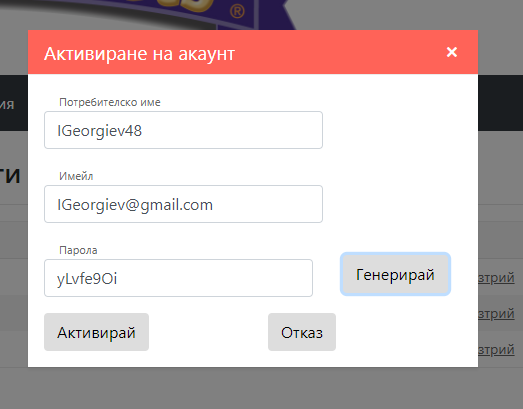
    for (let i = 0; i <8; i++) {

      password += possible.charAt(Math.floor(Math.random() \* possible.length));

    }

    return password;

  }



В следващия фрагмент е представен метода, в който уеб приложението прави заявка към сървъра.

onActivationSubmitted(): void {

    this.subscription.add(this.adminService.activateAccount(this.selectedAccount.id, this.selectedAccount)

      .subscribe(() => {

        this.selectedAccount.isActivated = true;

        this.selectedAccount.isEnabled = true;

        alert('Акаунта беше активиран');

      }, (erroResponse: HttpErrorResponse) => {

        console.log(erroResponse.message);

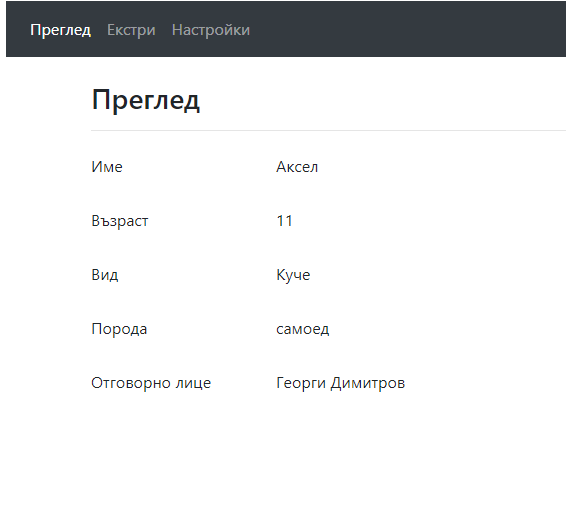
      }));

    this.isActivateDialogOpened = false;

  }

В диалоговия прозорец за активиране на акаунт се извършва и редакция на вече активирани акаунти.

### Клиентски модул

* Преглед  
    
  

Екранът за преглед на общата информация претърпя известна промяна спрямо изготвения във фунцкионалния дизайн. Реализацията беше свързана с изпращането на GET заявка към сървъра, която програмно беше реализирана по следният начин:

getReservationDetails(userName: string): Observable<OverviewViewModel> {

    let params = new HttpParams().set('username', userName);

    returnthis.httpClient.get<OverviewViewModel>(baseURL + 'api/customer/getReservationDetails', { headers: this.httpAuthorized , params: params });

  }

Тъй като потребителското име не е конфденциална информация няма пречка да бъде част от пътя, който достъпваме от сървъра, поради тази причина е и добавено, като параметър към заявката. В резултат в клиентското приложение постъпва модел от тип OverviewViewModel, който съдържа следните данни:

exportclass OverviewViewModel {

  petName: string;

  petAge: number;

  petType: string;

  petBreed: string;

  employeeName: string;

}

В сървърното приложение екранът за преглед е реализиран, както следва:

**publicclass** UserReview {

**private** String petName;

**privateint**petAge;

**private** String petType;

**private** String petBreed;

**private** String employeeName;

модел със данни на домашния любимец и име на отговорното лице.

@RequestMapping(path = "/customer/getReservationDetails", method = RequestMethod.***GET***)

@PreAuthorize("hasRole('USER')")

**public**@ResponseBody UserReview getUser(@RequestParam String username) **throws** ClassNotFoundException, SQLException {

UserReview userReview = **new** UserReview();

String temp = "SELECT `pet\_details`.`Name`, `pet\_details`.`Age`, `pet\_details`.`Type`, `pet\_details`.`Breed`, `employee`.`Name` as 'EmployeeName' FROM `pet\_details` JOIN `employee` ON `employee`.`ID` = `pet\_details`.`EmployeeID` JOIN `user` ON `user`.`ID` = `pet\_details`.`UserID` WHERE `user`.`Username` LIKE '"+ username + "'";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con=(Connection) DriverManager.*getConnection*("jdbc:mysql://192.168.10.220:3306/AnimalHotelDB", "diplom", "mantrakuka");

Statement statement = (Statement) con.prepareStatement(temp);

ResultSet resultSet = statement.executeQuery(temp);

**while**(resultSet.next()) {

userReview.setPetName(resultSet.getString("Name"));

userReview.setPetAge(resultSet.getInt("Age"));

userReview.setPetType(resultSet.getString("Type"));

userReview.setPetBreed(resultSet.getString("Breed"));

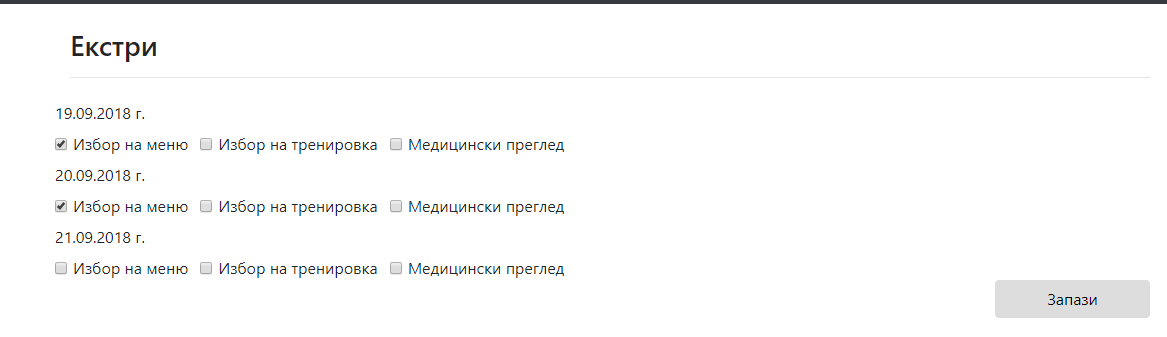
userRe view.setEmployeeName(resultSet.getString("EmployeeName"));

}

**return**userReview;

}

Функцията изпраща заявка до базата данни и извлича необходимата информация след което се изпраща до съответния потребител.

* Екстри  
  

Чрез екрана Екстри потребителя има възможност да си добавя нови екстри към вече създадената и потвърдена резервация. Екрана беше реализиран чрез извличане на състоянието на текущата резервация:

  this.subscription.add(this.customerService.getReservationDatesDetails(userName).subscribe((data: ReservationDateViewModel[]) => {

        data.forEach((date) => {

          let temp = new Date(date.date);

          date.date = this.parseToLocaleDate(temp).toString();

        });

        this.dateDetails = data;

      }, (httpErrorResponse: HttpErrorResponse) => {

        alert('Възникна проблем! Проверете конзолата за повече детайли и се свържете с администратор');

        console.log(httpErrorResponse.message);

      }));

От сървърна гледна точка реализацията на този екран изисква да се вземе модела и да се изпрати в базата данни.

@RequestMapping(path = "/customer/getReservationDetails", method = RequestMethod.***GET***)

@PreAuthorize("hasRole('USER')")

**public**@ResponseBody UserReview getUser(@RequestParam String username) **throws** ClassNotFoundException, SQLException {

UserReview userReview = **new** UserReview();

String temp = "SELECT `pet\_details`.`Name`, `pet\_details`.`Age`, `pet\_details`.`Type`, `pet\_details`.`Breed`, `employee`.`Name` as 'EmployeeName' FROM `pet\_details` JOIN `employee` ON `employee`.`ID` = `pet\_details`.`EmployeeID` JOIN `user` ON `user`.`ID` = `pet\_details`.`UserID` WHERE `user`.`Username` LIKE '"+ username + "'";

Class.*forName*("com.mysql.jdbc.Driver");

Connection con=(Connection) DriverManager.*getConnection*("jdbc:mysql://192.168.10.220:3306/AnimalHotelDB", "diplom", "mantrakuka");

Statement statement = (Statement) con.prepareStatement(temp);

ResultSet resultSet = statement.executeQuery(temp);

**while**(resultSet.next()) {

userReview.setPetName(resultSet.getString("Name"));

userReview.setPetAge(resultSet.getInt("Age"));

userReview.setPetType(resultSet.getString("Type"));

userReview.setPetBreed(resultSet.getString("Breed"));

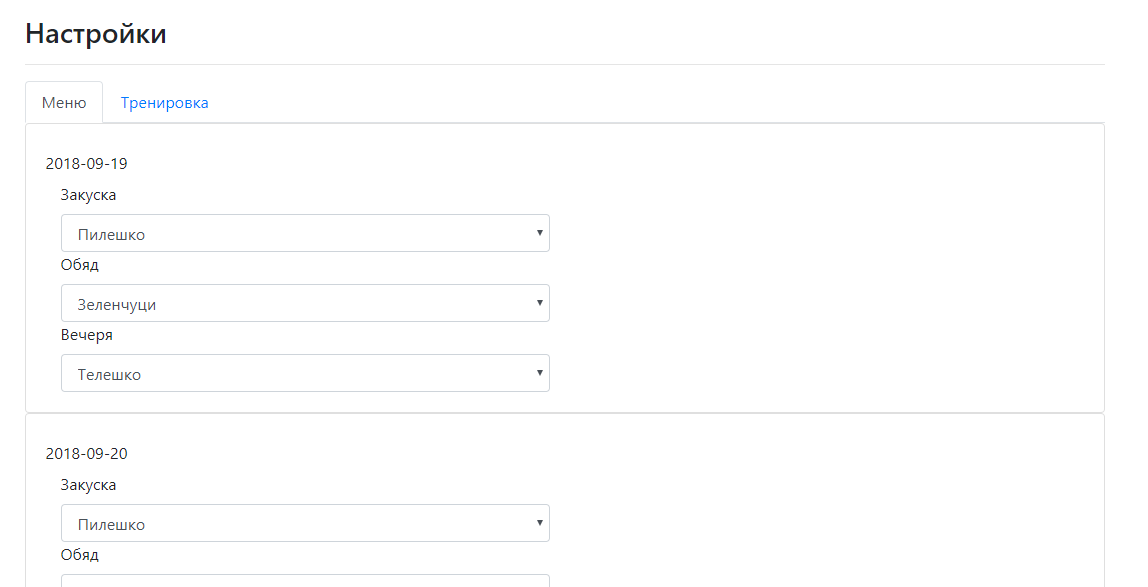
userReview.setEmployeeName(resultSet.getString("EmployeeName"));

}

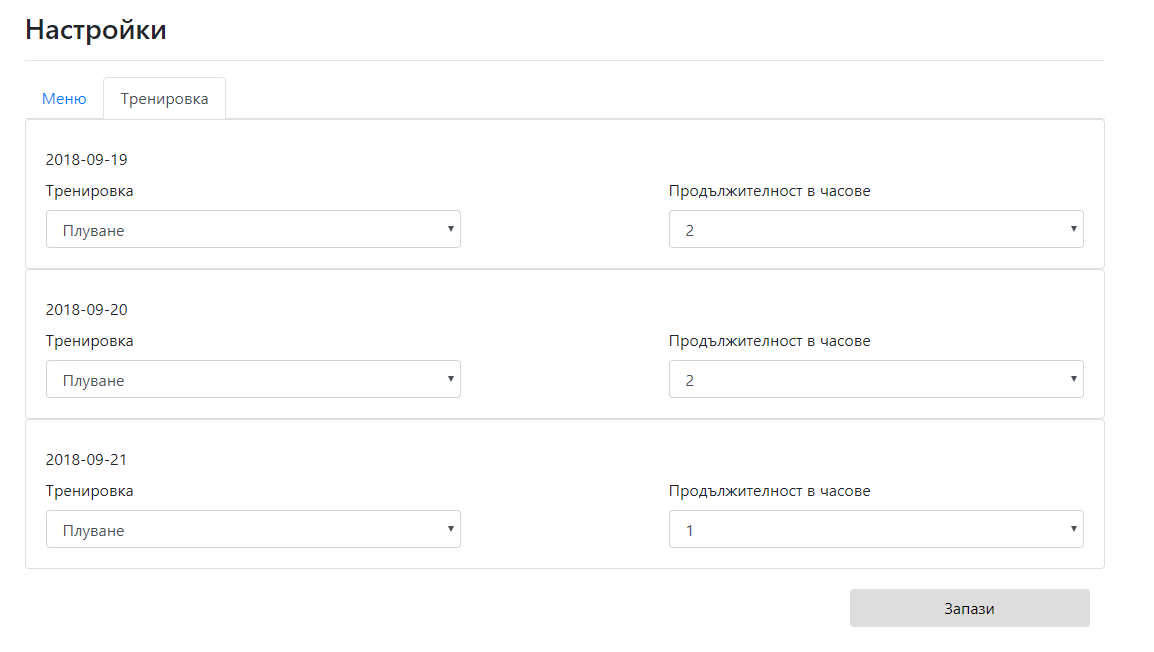
**return**userReview;

}

* Настройки



Екранът за настройки включва задаване на дневно меню за всеки от дните, през които трае резервацията и настройки за вид и продължителност на тренировка. От сървърната страна е подобно като екстрите.



## Съпоставка с подобни приложения

Системите за онлайн резервации са неизменна част от хотелиерския бизнес в частност и сегмента от него свързан с домашните любимци. Тъй като този бизнес не е развит на високо ниво в България, предвид факта, че стандарта на живот на средно статистическия българин не му позволява да задели средства за комфорта на своя домашен любимец, за целите на разработката ще направя съпоставка с хотел за домашни любимци от САЩ – името му е Posh Pet Hotel (PPH). В таблица 4.1 е представено кратко сравнение на двете системи по редица ключови критерии.

|  |  |  |
| --- | --- | --- |
| Критерий | Kingdom Pets | PPH |
| Контактна форма | Двете системи имат почти идентични контактни форми, като във връзка с промените в закона за защита на личните данни в Kingdom Pets се изисква минимално необходимата информация без излишни детайли. | |
| Вписване в системата | Функционалността е възможна след потвърждаване на направена резервация, като данните за вписване се генерират от администратор | Няма модул за регистриране, респективно вписване в системата |
| Онлайн резервация | Удобно разположена в структурата на сайта | Не разполага с такава, резервациите се правят по телефон или имейл |
| Разнообразие в услугите | Kingdom Pets предоставя не малък брой услуги и екстри на свойте потребители в т.ч избор груминг, медицински грижи, обучения и др. | Системата е изключително развита в тази насока. Предоставят на клиентите си много богата гама от всевъзможни услуги и грижи. |
| Блог към сайта | На този етап системата не разполага с блог, но като точка за бъдещо развитие може да се създаде такъв | Много добре изграден блог с интересно съдържание и аудитория |
| Избор на отговорно лице и персонални конфигурации | Kingdom Pets дава възможност на потребителите си да задават конфигурации за всеки аспект от пребиваването на домашните любимци от задаването на храненията, през тренировките, до избор на треньори и т.н | Системата не предоставя възможност за персонализиране на резервацията и престоя. |

Таб. ‑ Съпоставка на Kingdom Pets с конкурентна система PPH по ключови критерии

Kingdom Pets дава възможност за персонализация, такава каквато е необходима за по-взискателните потребители на предоставяните услуги, а също така и предоставя възможност за упражняването на контрол върху цялото съдържание на уеб приложението чрез добре изграден административен панел.

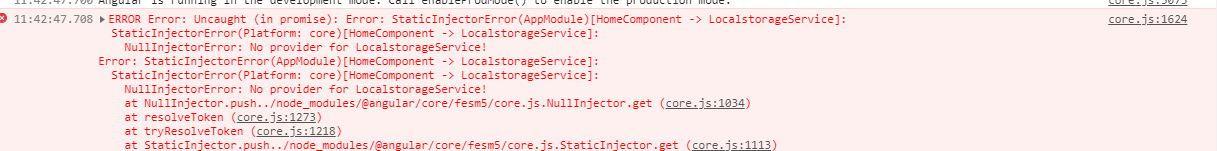
# Тестове и резултати

Процеса на разработване на софтуер е свързан с възникването на множество грешки и неточности, от чието успешно отстраняване зависи завършването на даденото приложение. В точка 5.1 от тази глава са посочени част от грешките, които възникнаха в процеса на разработка на уеб приложението. В точка 5.2 са представени резултатите от тестовете.

## Тестове и грешки

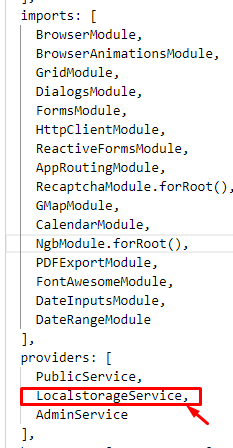
### Клиентски част

На фиг.5-1 е представена една от най-често срещаните грешки, която е породена в случаите, когато е използвана услуга от service файловете, без този файл да е бил добавен в модула на компонента.

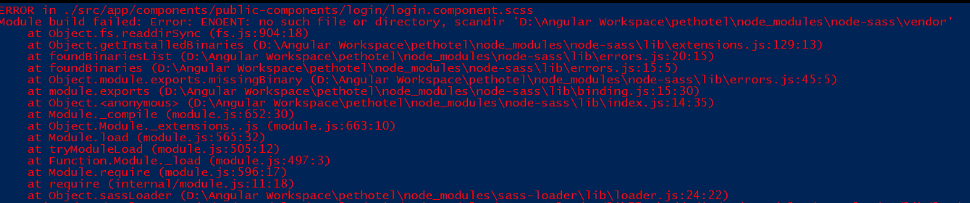


Фиг. ‑Грешка, която възниква в случаите, когато е използван компонент без да е добавен в съответния модул

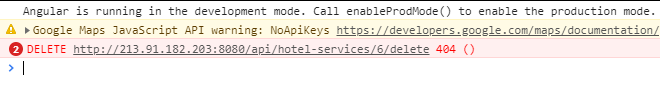
Решението на проблема е свързано с добавянето на съответния service в модула. В конкретния пример представен по-горе решението е:



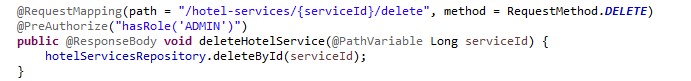
Друг често срещан проблем е терминална грешка, която е в следствие от липсата на директория node\_modules, в която се намират всички файлове, които изграждат ядрото на Angular.

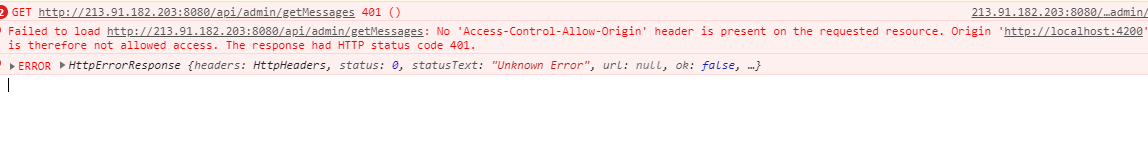


Фиг. ‑Грешка в следствие на липсата на директория node\_modules.

Решение на проблема представен на фиг. 5-2 е конзолна команда npm install , с която се изтеглят всички библиотеки описани във файла със зависимости на проекта – project.json.   
  
Липса на ресурс (грешка 404). Възниква в случаите, когато на дадения адрес сървъра не връща нищо.   


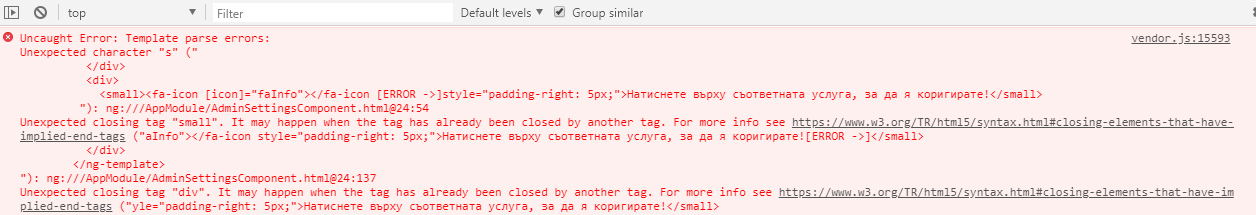
Фиг. ‑ Липса на регистриран адрес в сървърното приложение

Решението на проблема беше чрез добавяне на адрес в сървърното приложение.   
  
  
 Чест проблем в приложенията с оторизация на достъпа е опит за достъпване на ресурси, които изискват автентикация. В следствие на тези опити в конзолата на браузъра се появя грешката представена на фиг. 5-4 – неоторизиран достъп 401.



Фиг. ‑ Грешка 401 Неоторизиран достъп

На фиг. 5-5 е представена грешка в следствие на сбъркан синтаксис в HTML.



Фиг. ‑Грешка в следствие на неправилен HTML.

Решението на проблема е корекция в сгрешения файл.



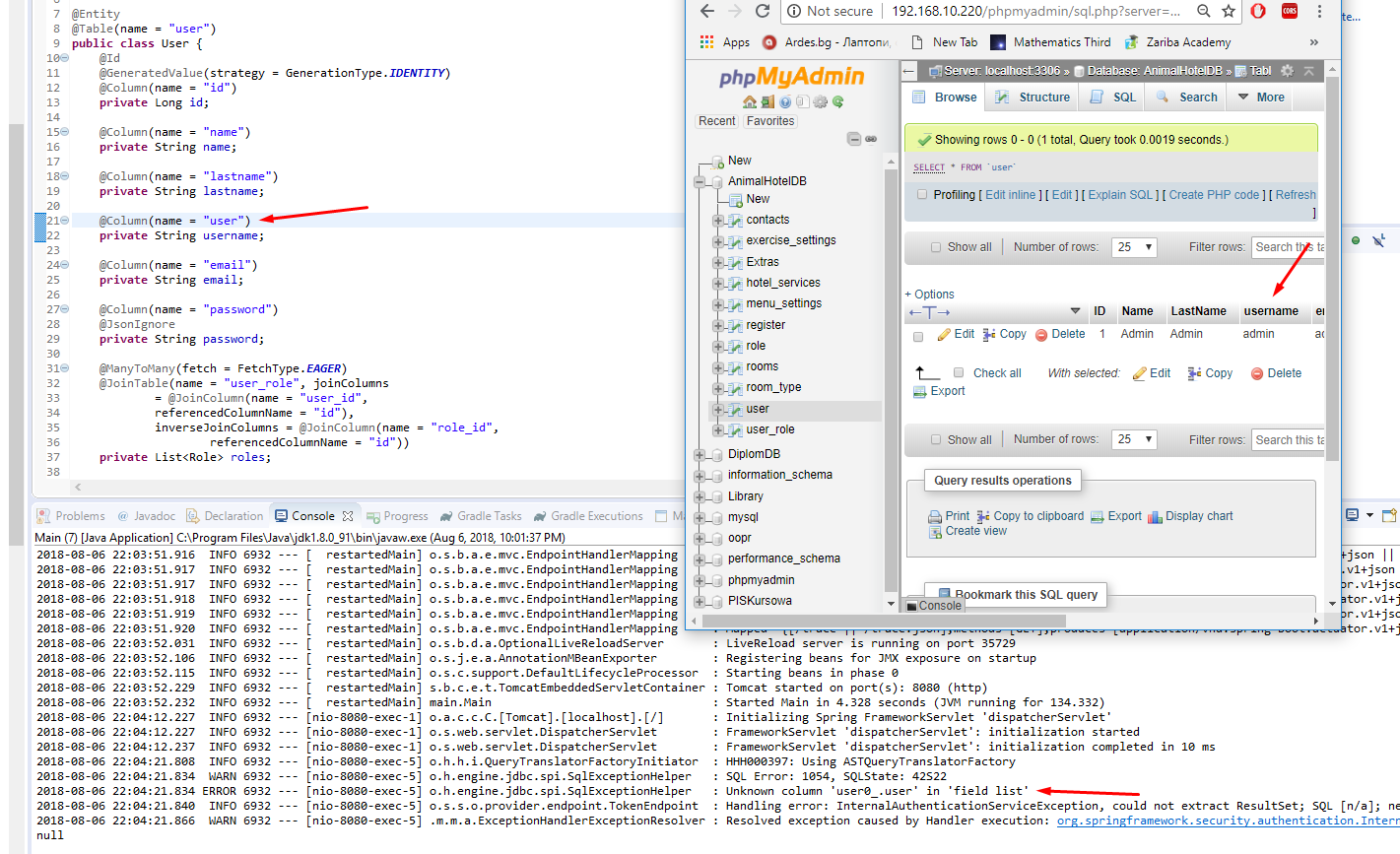
Фиг. ‑ Грешка при липса на инициализация на променлива



Фиг. ‑ Грешка поради липса на път във файла описваш сървърните адреси (за достъп до фунцкионалности)

### Сървърна част

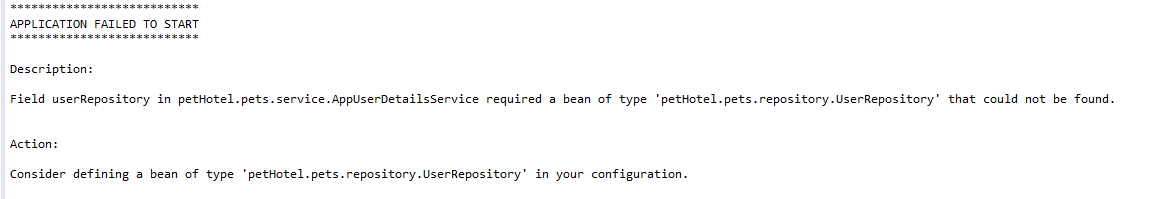
Дадената грешка във фиг. 5-1 се оправя като се смени или името в програмата или името в колоната



Фиг. 5‑8 Грешка когато връзката с базата данни не съответства с модела

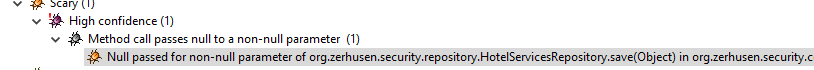
Грешката във Фиг. 5-2 се появява когато програмата не може да намери пътя към класа.

Оправя се, когато се сложи на правилното място или в главния клас се зададе пътя към файла



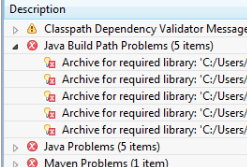
Фиг. 5‑9 Когато програмата не може да намери файла

Грешка където се подава празни стойности към функцията. Оправя се като се сложат валидации в модела или default стойности в обекта. Такъв вид грешки пълнят базата данни с нулеви стойности които трябва да се изчистят.



Фиг. ‑10Нулеви стойности

Понякога когато се теглят допълнителни библиотеки се получава дадения проблем, оправя се когато се изчисти проекта, ако продължава да задава проблема се влиза в директорията на проекта, изтриват се всички библиотеки и се стартира приложението отново, за да ги изтегли.



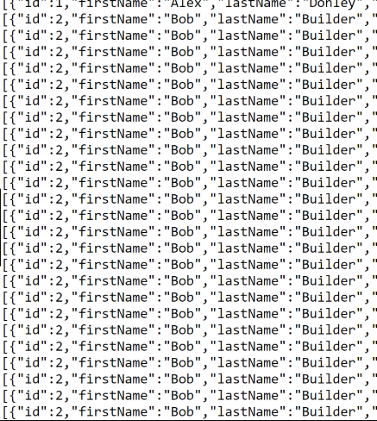
Фиг. 5‑11 Грешка когато се теглят библиотеки

Когато не се уточни че е delete метод изписва дадения проблем и не може да изтрий даден запис в базата данни



Фиг. ‑12 Грешка когато не се направи уточни дали е Delete метод

Дадения проблем във фиг.5-13 се появява когато не са свързани правилно моделите. Често се случва когато се правят опити много със много връзки, създавайки безкраен цикъл, който чупи сървъра.



Фиг. 5‑13 Безкраен Цикъл

## Резултати

В резултат на направените тестове бяха отстранени голяма част от потенциалните грешки в следствие на осъществяване на връзка със сървъра,като 400 Bad Request, 404 Resource Not Found, 500 Server is not available, съответно прихванати и обработени изключения. Това доведе до подобряване на потребителските усещания (UX).

# Заключение

Неизменна част от развитието на всеки един бизнес е изграждането на система, която да го представя в интернет и чрез която масовия потребител има възможност да се възползва от предлаганите услуги. В частност при хотелиерския бизнес е задължително наличието на система за онлайн резервации, като успеха зависи от добре изградените модули за управление и най-вече влиянието на приложението върху потребителските усещания и сигурността, която вдъхва системата. В Kingdom Pets е приложение изградено именно върху това. То разполага с няколко модула, които са достъпвaни в зависимост от правомощията, като администратора има пълен контрол върхy всеки аспект от сайта. Една такава система следва да бъде съпроводена от блог/форум, в който да се получават отзиви за качеството на услугите в текущата версия такъв липсва, но е част от възможностите за бъдещо развитие към които може да се добавят:

* Видеонаблюдение на домашния любимец – една такава опция би дала на клиента сигурност и възможност (особено за по-привързаните) да види домашният си любимец;
* По-широка гама от продукти – в момента системата предлага стандартeн набор от услуги включващ всички основни нужди на животното;

Софтуерния продукт “Kingdom Pets” описан в текущата документация отговаря напълно на поставените в спецификацията на проекта характеристики, като осигурява на крайният потребител удобство и сигурност в боравенето с изградения потребителски интерфейс.

# Източници на информация

1. [*http://stackoverflow.com/*](http://stackoverflow.com/) *- форум за въпроси свързани с програмиране;*
2. [*https://*](https://docs.unity3d.com/Manual/index.html%20)*angular.io - официална документация за Angular;*
3. [*https://spring.io/*](https://spring.io/)*- Официялен сайт за уроци и документация на Java Spring*

# Приложение

Contact.component.html

<divclass="container">

<divclass="form-group">

<h3>Контакти</h3>

<hr>

</div>

<divclass="row">

<divclass="col-12 col-sm-6">

<p-gmap[options]="options"[overlays]="overlays"[style]="{'width':'100%','height':'420px'}"></p-gmap>

</div>

<divclass="col-12 col-sm-6">

<form[formGroup]="contactForm"(ngSubmit)="onSubmit()"novalidate>

<divclass="form-group">

<labelfor="email">Имейл</label>

<inputformControlName="email"id="email"type="email"class="form-control"required>

<divclass="text-danger"\*ngIf="contactForm.get('email').hasError('required') && contactForm.get('email').touched">

Полето за имейл е задължително!

</div>

<divclass="text-danger"\*ngIf="contactForm.get('email').hasError('pattern') && contactForm.get('email').touched">

Невалиден имейл!

</div>

</div>

<divclass="form-group">

<labelfor="about">Относно</label>

<inputformControlName="subject"id="about"type="text"class="form-control"required>

<divclass="text-danger"\*ngIf="contactForm.get('subject').hasError('required') && contactForm.get('subject').touched">

Полето за тема е задължително!

</div>

</div>

<divclass="form-group">

<labelfor="msg">Съобщение</label>

<textareaformControlName="message"id="msg"class="form-control"required></textarea>

<divclass="text-danger"\*ngIf="contactForm.get('message').hasError('required') && contactForm.get('message').touched">

Съобщението е задължително!

</div>

</div>

<divclass="form-group">

<re-captcha(resolved)="resolved($event)"siteKey="6LeDcmUUAAAAAEerpDa6E98kTEHKNvkyQ0mjQiaO"></re-captcha>

</div>

<divclass="form-group mb-7">

<buttontype="submit"class="btn btn-primary">Изпрати</button>

</div>

</form>

</div>

</div>

</div>

Contact.component.ts

/// <referencetypes="googlemaps"/>

import { Component, OnInit, OnDestroy } from'@angular/core';

import { FormBuilder, FormGroup } from'@angular/forms';

import { Validators } from'@angular/forms';

import { PublicService } from'../../../services/public.service';

import { Subscription } from'rxjs';

import { ContactBindingModel } from'../../../models/shared-models/binding-models/contact-binding-model';

@Component({

  selector: 'app-contacts',

  templateUrl: './contacts.component.html',

  styleUrls: ['./contacts.component.scss']

})

exportclass ContactsComponent implements OnInit, OnDestroy {

  options: any;

  overlays: any[];

  contactForm: FormGroup;

  private captchaToken: string;

  private contactSubscription: Subscription;

  constructor(private publicService: PublicService, private fb: FormBuilder) {

    this.contactSubscription = new Subscription();

    this.contactForm = this.fb.group({

      email: ['',

        [

          Validators.required,

          Validators.pattern(/^[a-z0-9\_]+(\.[\_a-z0-9]+)\*@[a-z0-9-]+(\.[a-z0-9-]+)\*(\.[a-z]{2,15})$/ig)

        ]

      ],

      subject: ['',

        [

          Validators.required

        ]],

      message: ['',

        [

          Validators.required

        ]]

    });

  }

  ngOnInit() {

    this.options = {

      center: { lat: 43.2229, lng: 27.9368 },

      zoom: 12

    };

    this.overlays = [

      new google.maps.Marker({ position: { lat: 43.2229, lng: 27.9368 }, title: 'Kingdom Pets' }),

    ];

  }

  resolved(captchaResponse: string) {

    this.captchaToken = captchaResponse;

  }

  onSubmit(): void {

    if (this.contactForm.valid) {

      if (this.captchaToken !== ''&&this.captchaToken !== undefined) {

        const contactModel: ContactBindingModel = this.contactForm.value;

        this.contactSubscription.add(this.publicService.sendMessage(contactModel, this.captchaToken).subscribe());

      }

    } else {

      thrownew Error('Формата не е попълнена коректно!');

    }

  }

  ngOnDestroy(): void {

    // prevent memory leaks

    this.contactSubscription.unsubscribe();

  }

}

Requirements.component.html

<divclass="container">

<divclass="form-group">

Kingdom PETS е идеалното място за вашият домашен любимец. Предоставяме широка гама услуги, които обаче налагат представянето

на някои документи, които следва да са в изряден вид:

</div>

<divclass="form-group">

<ul>

<li>паспорт с всички ваксинации</li>

<li>попълнена декларация за постъпване</li>

<li>попълнен въпросник</li>

</ul>

</div>

<divclass="form-group">

За да се възползвате от предлаганите от нас услуги следва да попълните представените по-долу формуляри. В тях се съдържа

основна информация за домашният ви любимец и минимално необходимата за нашите ветеринарни специалисти. Благодарим ви

за проявеното разбиране!

</div>

<divclass="form-group">

<ngb-tabsetjustify="center">

<ngb-tabtitle="Декларация">

<ng-templatengbTabContent>

<divclass="container"style="margin-top: 20px;">

<kendo-pdf-export#pdfpaperSize="A4"margin="2cm">

<app-declaration></app-declaration>

</kendo-pdf-export>

<divclass="form-group"style="margin-top: 25px; margin-bottom: 90px;">

<buttonclass="form-control col-12 col-sm-3"kendo-button(click)="pdf.saveAs('declaration.pdf')">

Запази, като PDF...

</button>

</div>

</div>

</ng-template>

</ngb-tab>

<ngb-tabtitle="Въпросник">

<ng-templatengbTabContent>

<divclass="container"style="margin-top: 20px;">

<kendo-pdf-export#pdfpaperSize="A4"margin="2cm">

<app-questions></app-questions>

</kendo-pdf-export>

<divclass="form-group"style="margin-top: 25px; margin-bottom: 90px;">

<buttonclass="form-control col-12 col-sm-3"kendo-button(click)="pdf.saveAs('declaration.pdf')">

Запази, като PDF...

</button>

</div>

</div>

</ng-template>

</ngb-tab>

</ngb-tabset>

</div>

</div>

Requirements.component.ts

import { Component, OnInit } from'@angular/core';

import { defineFont } from'@progress/kendo-drawing/pdf';

defineFont({

  'DejaVu Sans' : '//kendo.cdn.telerik.com/2014.3.1314/styles/fonts/DejaVu/DejaVuSans.ttf',

  'DejaVu Sans|Bold' : '//kendo.cdn.telerik.com/2014.3.1314/styles/fonts/DejaVu/DejaVuSans-Bold.ttf',

  'DejaVu Sans|Bold|Italic' : '//kendo.cdn.telerik.com/2014.3.1314/styles/fonts/DejaVu/DejaVuSans-Oblique.ttf',

  'DejaVu Sans|Italic' : '//kendo.cdn.telerik.com/2014.3.1314/styles/fonts/DejaVu/DejaVuSans-Oblique.ttf'

});

@Component({

  selector: 'app-requirements',

  templateUrl: './requirements.component.html',

  styleUrls: ['./requirements.component.scss']

})

exportclass RequirementsComponent implements OnInit {

  currentJustify = 'start';

  constructor() { }

  ngOnInit() {

  }

}

Reservation.component.html

<divclass="container"style="margin-bottom: 80px !important;">

<divclass="row">

<divclass="form-group col-12">

<h3\*ngIf="!isSecondResStep">Резервация - стъпка 1</h3>

<h3\*ngIf="isSecondResStep">Резервация - стъпка 2</h3>

<hr>

</div>

</div>

<divclass="form-group">

<div\*ngIf="!isSecondResStep"class="row">

<divclass="col-12 col-sm-2">

<span>Изберете период за престой</span>

</div>

<divclass="col-12 col-sm-5">

<kendo-daterange>

<label>

<spanclass="label">Начална дата</span>

<kendo-dateinputkendoDateRangeStartInput[(value)]="range.start"></kendo-dateinput>

</label>

<label>

<spanclass="label">Крайна дата</span>

<kendo-dateinputkendoDateRangeEndInput[(value)]="range.end"></kendo-dateinput>

</label>

</kendo-daterange>

</div>

<divclass="col-12 col-sm-2">

<labelfor="roomType"class="label">Стая</label>

<selectstyle="width: 100%"name=""id="roomType"[(ngModel)]="filter.roomType">

<option></option>

<optionvalue="1">ВИП</option>

<optionvalue="2">Стандарт</option>

</select>

</div>

<divclass="col-12 col-sm-2 mt-3">

<button(click)="onFindButtonClicked()"class="btn">Търси</button>

</div>

</div>

<div\*ngIf="!isSecondResStep"class="row">

<hrclass="col-12">

<divclass="col-12 col-sm-7">

<labelfor=""class="label">Резултат</label>

<kendo-grid

[kendoGridBinding]="filteredReservations | async"

[pageSize]="4"

[pageable]="true"

[selectable]="selectableSettings"

[kendoGridSelectBy]="'roomId'"

[selectedKeys]="roomsIds">

<kendo-grid-checkbox-column[width]="60"></kendo-grid-checkbox-column>

<kendo-grid-columnfield="roomId"title="Име на стаята"></kendo-grid-column>

<kendo-grid-columnfield="roomName"title="Име на стаята"></kendo-grid-column>

<kendo-grid-columnfield="roomType"title="Тип стая">

<ng-templatekendoGridCellTemplatelet-room>

<span\*ngIf="room.roomType === 1">ВИП</span>

<span\*ngIf="room.roomType !== 1">Стандарт</span>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnfield="roomPrice"title="Цена"></kendo-grid-column>

</kendo-grid>

</div>

<divclass="col-12 col-sm-4">

<labelfor=""class="label">Екстри</label>

<kendo-grid[kendoGridBinding]="extras | async"[pageSize]="4"[pageable]="true"[selectable]="true"[kendoGridSelectBy]="'id'"

[selectedKeys]="extrasIds">

<kendo-grid-checkbox-columnshowSelectAll="true"[width]="60"></kendo-grid-checkbox-column>

<kendo-grid-columnfield="name"title="Екстра"></kendo-grid-column>

<kendo-grid-columnfield="price"title="Цена"></kendo-grid-column>

</kendo-grid>

</div>

</div>

</div>

<divclass="row">

<div\*ngIf="isSecondResStep"class="col-12 col-sm-6">

<divclass="container">

<divclass="form-group">

<inputtype="text"class="form-control"placeholder="Вашите имена..."[(ngModel)]="reservationBindingModel.name">

</div>

<divclass="form-group">

<inputtype="text"class="form-control"placeholder="Вашият имейл..."[(ngModel)]="reservationBindingModel.email">

</div>

<divclass="form-group">

<inputtype="text"class="form-control"placeholder="Телефон за контакт..."[(ngModel)]="reservationBindingModel.phoneNumber">

</div>

<divclass="form-group">

<divclass="form-check form-check-inline">

<labelclass="form-check-label">

<inputclass="form-check-input"type="checkbox"id="inlineCheckbox1"value="option1"> Прочетох общите условия

</label>

</div>

</div>

<divclass="form-group">

<re-captcha(resolved)="resolved($event)"siteKey="6LeDcmUUAAAAAEerpDa6E98kTEHKNvkyQ0mjQiaO"></re-captcha>

</div>

</div>

</div>

</div>

<div\*ngIf="!isSecondResStep"class="form-group">

<divclass="col-12 col-sm-3">

<buttonclass="btn btn-primary"(click)="isSecondResStep=!isSecondResStep"[disabled]="roomsIds.length === 0">Напред</button>

</div>

</div>

<divclass="container">

<div\*ngIf="isSecondResStep"class="form-group row">

<divclass="col-12 col-sm-3">

<button(click)="onSubmit()"class="btn btn-primary">Резервация</button>

</div>

<divclass="col-12 offset-sm-4 col-sm-3 offset-md-1">

<buttonclass="btn btn-primary"(click)="isSecondResStep=!isSecondResStep">Назад</button>

</div>

</div>

</div>

</div>

Reservation.component.ts

import { HttpErrorResponse } from'@angular/common/http';

import { ReservationFilterBindingModel } from'../../../models/public-models/binding-models/reservation-filter-binding-model';

import { FilteredReservationViewModel } from'../../../models/public-models/view-models/filtered-reservation-view-model';

import { Component, OnInit, OnDestroy } from'@angular/core';

import { SelectableSettings } from'@progress/kendo-angular-grid';

import { PublicService } from'../../../services/public.service';

import { Observable, Subscription } from'rxjs';

import { HotelProductSimpleViewModel } from'../../../models/public-models/view-models/hotel-product-simple-view-model';

import { ReservationBindingModel } from'../../../models/public-models/binding-models/reservation-binding-model';

import { Router } from'@angular/router';

@Component({

  selector: 'app-reservation',

  templateUrl: './reservation.component.html',

  styleUrls: ['./reservation.component.scss']

})

exportclass ReservationComponent implements OnInit, OnDestroy {

  value: Date;

  isSecondResStep: boolean;

  extrasIds: number[] = [];

  roomsIds: number[] = [];

  filter: ReservationFilterBindingModel;

  reservationBindingModel: ReservationBindingModel;

  filteredReservations: Observable<FilteredReservationViewModel[]>;

  extras: Observable<HotelProductSimpleViewModel[]>;

  public selectableSettings: SelectableSettings;

  public range = { start: null, end: null };

  private captchaToken: string;

  private subscription: Subscription;

  constructor(private publicService: PublicService, private router: Router) {

    this.isSecondResStep = false;

    this.filter = new ReservationFilterBindingModel();

    this.setSelectableSettings();

    this.reservationBindingModel = new ReservationBindingModel();

  }

  ngOnInit() {

    this.getExtras();

  }

  onSubmit(): void {

    this.reservationBindingModel.startDate = this.range.start;

    this.reservationBindingModel.endDate = this.range.end;

    this.reservationBindingModel.roomId = this.roomsIds[0];

    this.reservationBindingModel.extras = this.extrasIds;

    this.subscription = this.publicService.doReservation(this.reservationBindingModel, this.captchaToken).subscribe(() => {

        this.router.navigate(['']);

    }, (errorResponse: HttpErrorResponse) => {

      if (errorResponse.status >500) {

        alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

      } else {

        alert(errorResponse.message);

      }

    });

  }

  public setSelectableSettings(): void {

    this.selectableSettings = {

      checkboxOnly: false,

      mode: 'single'

    };

  }

  ngOnDestroy(): void {

    if (this.subscription !== null&&this.subscription !== undefined) {

      this.subscription.unsubscribe();

    }

  }

  onFindButtonClicked(): void {

    console.log(this.range.start);

    console.log(this.range.end);

    this.filter.startDate = this.range.start;

    this.filter.endDate = this.range.end;

    this.filteredReservations = this.publicService.getFilteredReservations(this.filter).pipe();

  }

  resolved(captchaResponse: string) {

    console.log(`Resolved captcha with response ${captchaResponse}:`);

    this.captchaToken = captchaResponse;

  }

  private getExtras(): void {

    this.extras = this.publicService.getHotelExtras().pipe();

  }

}

Extras.component.html – Customer module

<divclass="container">

<divclass="form-group mt-4">

<h3>Екстри</h3>

<hr>

</div>

<divclass="form-group">

<div\*ngFor="let date of dateDetails; let i = index"class="row">

<inputtype="text"readonlyclass="form-control-plaintext"[(ngModel)]="dateDetails[i].date"value="{{date.date}}">

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="checkbox"id="menu"[(ngModel)]="dateDetails[i].menu"value="{{date.menu}}">

<labelclass="form-check-label"for="menu">Избор на меню</label>

</div>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="checkbox"id="training"[(ngModel)]="dateDetails[i].training"value="{{date.training}}">

<labelclass="form-check-label"for="training">Избор на тренировка</label>

</div>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="checkbox"id="medAssist"[(ngModel)]="dateDetails[i].medAssist"value="{{date.medAssist}}">

<labelclass="form-check-label"for="medAssist">Медицински преглед</label>

</div>

</div>

<divclass="row">

<divclass="col-12 offset-sm-9 col-sm-3 offset-md-10 col-md-2">

<buttonclass="btn btn-block"(click)="onSaveButtonClicked()">Запази</button>

</div>

</div>

</div>

</div>

Extras.component.ts

import { Component, OnInit } from'@angular/core';

import { Observable, Subscription } from'rxjs';

import { CustomerService } from'../../../services/customer-services/customer.service';

import { LocalstorageService } from'../../../services/localstorage.service';

import { ReservationDateViewModel } from'../../../models/customer-models/view-models/reservation-date-view-model';

import { HttpErrorResponse } from'@angular/common/http';

import { ReservationDateBindingModel } from'../../../models/customer-models/binding-models/reservation-date-binding-model';

@Component({

  selector: 'app-extras',

  templateUrl: './extras.component.html',

  styleUrls: ['./extras.component.scss']

})

exportclass ExtrasComponent implements OnInit {

  dateDetails: ReservationDateViewModel[];

  private editDateDetails: ReservationDateBindingModel[];

  private subscription: Subscription;

  private userName: string;

  constructor(private customerService: CustomerService, private localStorageService: LocalstorageService) {

    this.dateDetails = [];

    this.editDateDetails = [];

    this.subscription = new Subscription();

  }

  ngOnInit() {

    this.userName = this.localStorageService.getUserName();

    if (this.userName !== ''&&this.userName !== null&&this.userName !== undefined) {

      this.subscription.add(this.customerService.getReservationDatesDetails(this.userName).subscribe((data: ReservationDateViewModel[]) => {

        data.forEach((date) => {

          date.date = new Date(date.date);

        });

        this.dateDetails = data;

      }, (httpErrorResponse: HttpErrorResponse) => {

        alert('Възникна проблем! Проверете конзолата за повече детайли и се свържете с администратор');

        console.log(httpErrorResponse.message);

      }));

    }

    else {

      alert('problem');

    }

  }

  onSaveButtonClicked(): void {

    this.editDateDetails = this.dateDetails;

    this.subscription.add(this.customerService.editReservationDates(this.editDateDetails, this.userName).subscribe(() => {},

    (httpErrorResponse: HttpErrorResponse) => {

      alert('Възникна проблем! Проверете конзолата за повече детайли и се свържете с администратор');

      console.log(httpErrorResponse.message);

    }));

  }

  parseToLocaleDate(date: Date): string {

    return date.toLocaleDateString('bg-BG');

  }

}

Overview.component.html

<divclass="container">

<divclass="form-group mt-4">

<h3>Преглед</h3>

<hr>

</div>

<ng-container\*ngIf="overview | async as overview">

<divclass="form-group row">

<labelclass="col-sm-2 col-form-label">Име</label>

<divclass="col-sm-10">

<inputtype="text"readonlyclass="form-control-plaintext"[value]="overview.petName">

</div>

</div>

<divclass="form-group row">

<labelclass="col-sm-2 col-form-label">Възраст</label>

<divclass="col-sm-10">

<inputtype="text"readonlyclass="form-control-plaintext"[value]="overview.petAge">

</div>

</div>

<divclass="form-group row">

<labelclass="col-sm-2 col-form-label">Вид</label>

<divclass="col-sm-10">

<inputtype="text"readonlyclass="form-control-plaintext"[value]="overview.petType">

</div>

</div>

<divclass="form-group row">

<labelclass="col-sm-2 col-form-label">Порода</label>

<divclass="col-sm-10">

<inputtype="text"readonlyclass="form-control-plaintext"[value]="overview.petBreed">

</div>

</div>

<divclass="form-group row">

<labelclass="col-sm-2 col-form-label">Отговорно лице</label>

<divclass="col-sm-10">

<inputtype="text"readonlyclass="form-control-plaintext"[value]="overview.employeeName">

</div>

</div>

</ng-container>

</div>

Overview.component.ts

import { Component, OnInit } from'@angular/core';

import { Observable } from'rxjs';

import { OverviewViewModel } from'../../../models/customer-models/view-models/overview-view-model';

import { CustomerService } from'../../../services/customer-services/customer.service';

import { LocalstorageService } from'../../../services/localstorage.service';

@Component({

  selector: 'app-overview',

  templateUrl: './overview.component.html',

  styleUrls: ['./overview.component.scss']

})

exportclass OverviewComponent implements OnInit {

  overview: Observable<OverviewViewModel>;

  constructor(private customerService: CustomerService, private localStorageService: LocalstorageService) { }

  ngOnInit() {

    let userName = this.localStorageService.getUserName();

    if (userName !== ''&& userName !== null&& userName !== undefined) {

      this.overview = this.customerService.getReservationDetails(userName).pipe();

    }

    else {

      alert('problem');

    }

  }

}

Message-box.component.html

<divclass="container">

<divclass="form-group mt-4">

<h3>Преглед на съобщения</h3>

<hr>

</div>

<divclass="form-group">

<kendo-grid[data]="messages | async"height="auto">

<kendo-grid-columnfield="subject"title="Тема"width="250">

<ng-templatekendoGridCellTemplatelet-message>

<span(click)="onMessageClicked(message)"style="text-decoration: underline; cursor: pointer;">{{message.subject}}</span>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnfield="email"title="Автор"width="110"></kendo-grid-column>

<kendo-grid-columnfield="dateReceived"title="Дата"width="80">

<ng-templatekendoGridCellTemplatelet-message>

<span>{{parseToLocaleDate(message.dateReceived)}}</span>

</ng-template>

</kendo-grid-column>

</kendo-grid>

</div>

<kendo-dialogtitle="Отговор на съобщение"\*ngIf="isDialogOpened"(close)="onDialogClosed()"[minWidth]="250"[width]="450">

<divclass="form-group row">

<labelfor="subject"class="col-2 col-form-label">Тема</label>

<divclass="col-10">

<inputtype="text"readonlyclass="form-control-plaintext"id="subject"value="{{receivedMessage.subject}}">

</div>

</div>

<divclass="form-group row">

<labelfor="author"class="col-2 col-form-label">Автор</label>

<divclass="col-10">

<inputtype="text"readonlyclass="form-control-plaintext"id="author"value="{{receivedMessage.email}}">

</div>

</div>

<hr>

<divclass="form-group row">

<divclass="container">

<textareaclass="col-12 form-control"value="{{receivedMessage.message}}"[disabled]="true"></textarea>

</div>

</div>

<divclass="form-group row">

<divclass="container">

<textareaclass="col-12 form-control"[(ngModel)]="answerMessage.message"></textarea>

</div>

</div>

<divclass="row">

<divclass="col-6">

<buttonclass="btn"(click)="onSendButtonClicked()">Изпрати</button>

</div>

<divclass="col-6">

<buttonclass="btn"(click)="onDialogClosed()">Отказ</button>

</div>

</div>

</kendo-dialog>

</div>

Message-box.component.ts

import { Component, OnInit, OnDestroy } from'@angular/core';

import { Observable, Subscription } from'rxjs';

import { tap } from'rxjs/operators';

import { parseDate } from'@progress/kendo-angular-intl';

import { AdminService } from'../../../services/admin-services/admin.service';

import { ContactViewModel } from'../../../models/shared-models/view-models/contact-view-model';

import { ContactBindingModel } from'../../../models/admin-models/binding-models/contact-binding-model';

@Component({

  selector: 'app-messege-box',

  templateUrl: './messege-box.component.html',

  styleUrls: ['./messege-box.component.scss']

})

exportclass MessegeBoxComponent implements OnInit, OnDestroy {

  isDialogOpened: boolean;

  receivedMessage: ContactViewModel;

  messages: Observable<ContactViewModel[]>;

  answerMessage: ContactBindingModel;

  private subscription: Subscription;

  constructor(private adminService: AdminService) {

    this.isDialogOpened = false;

    this.answerMessage = new ContactBindingModel();

    this.receivedMessage = new ContactViewModel();

  }

  ngOnInit(): void {

    this.messages = this.adminService.getReceivedMesseges()

    .pipe(tap( (data: ContactViewModel[]) => {

      data.forEach((msg) => {

        msg.dateReceived = parseDate(msg.dateReceived);

      });

    }));

  }

  onMessageClicked(message: ContactViewModel): void {

    this.receivedMessage = message;

    this.answerMessage.messageId = message.id;

    this.isDialogOpened = true;

  }

  onSendButtonClicked(): void {

    if (this.answerMessage.message !== '') {

      this.subscription = this.adminService.sendAnswerMessage(this.answerMessage).subscribe();

      this.answerMessage = new ContactBindingModel();

      this.isDialogOpened = false;

    } else {

      alert ('Не може да изпратите празно съобщение!');

    }

  }

  onDialogClosed(): void {

    this.isDialogOpened = false;

  }

  parseToLocaleDate(date: Date): string {

    return date.toLocaleDateString('bg-BG');

  }

  ngOnDestroy(): void {

    if (this.subscription !== null&&this.subscription !== undefined ) {

      this.subscription.unsubscribe();

    }

  }

}

Admin-settings.component.html

<divclass="container">

<divclass="form-group mt-4">

<h3>Настройки</h3>

<hr>

</div>

<divclass="form-group">

<ngb-tabset#t="ngbTabset">

<ngb-tabid="tab-selectbyid1"title="Цени и услуги">

<ng-templatengbTabContent>

<div>

<kendo-grid[kendoGridBinding]="hotelServices | async"[pageSize]="4"[pageable]="true"[sortable]="true"height="auto">

<kendo-grid-columnfield="name"title="Услуга"width="200">

<ng-templatekendoGridCellTemplatelet-service>

<span(click)="onAddServiceButtonClicked(service)"style="text-decoration: underline; cursor: pointer;">{{service.name}}</span>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnfield="type"title="Тип"width="80">

<ng-templatekendoGridCellTemplatelet-service>

<p\*ngIf="service.type === 0">Услуга</p>

<p\*ngIf="service.type === 1">Екстра</p>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnfield="price"title="Цена за ден"width="80">

</kendo-grid-column>

<kendo-grid-columnfield="time"title="Период"width="80">

<ng-templatekendoGridCellTemplatelet-service>

<p\*ngIf="service.time === 0">Временна</p>

<p\*ngIf="service.time === 1">Постоянна</p>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnwidth="80">

<ng-templatekendoGridCellTemplatelet-service>

<span(click)="onDeleteServiceButtonClicked(service)"style="text-decoration: underline; cursor: pointer;">Изтрий</span>

</ng-template>

</kendo-grid-column>

</kendo-grid>

</div>

<divclass="float-right mt-2">

<buttonclass="btn"(click)="onAddServiceButtonClicked(null)">Добави</button>

</div>

<kendo-dialogtitle="Изтриване на услуга/продукт"\*ngIf="isDeleteDialogOpened"(close)="onDialogClosed()"[minWidth]="250"

[width]="450">

<pstyle="margin: 30px; text-align: center;">Искате ли да изтриете {{selectedService.name}}?</p>

<divclass="form-group">

<buttonclass="btn offset-2 col-3"(click)="onDeleteSubmitted()">Да</button>

<buttonclass="btn offset-2 col-3"(click)="onDialogClosed()">Не</button>

</div>

</kendo-dialog>

<kendo-dialogtitle="Добавяне на услуга/продукт"\*ngIf="isAddProductDialogOpened"(close)="onDialogClosed()"[minWidth]="250"

[width]="450">

<divclass="container">

<divclass="form-group">

<h6\*ngIf="!isEditProduct">Добавяне на услуга</h6>

<h6\*ngIf="isEditProduct">Редактиране на услуга</h6>

<hr>

</div>

<divclass="form-group">

<inputtype="text"class="form-control mb-1"placeholder="Въведете име на услугата"[(ngModel)]="newHotelService.name">

<inputtype="text"class="form-control mb-1"placeholder="Въведете кратко описание"[(ngModel)]="newHotelService.description">

<inputtype="number"class="form-control mb-1"placeholder="Въведете цена за ден"[(ngModel)]="newHotelService.price">

</div>

<divclass="form-group">

<labelstyle="display: block;">Вид</label>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions"id="inlineRadio1"[(ngModel)]="newHotelService.type"

value="0">

<labelclass="form-check-label"for="inlineRadio1">услуга</label>

</div>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions"id="inlineRadio2"[(ngModel)]="newHotelService.type"

value="1">

<labelclass="form-check-label"for="inlineRadio2">екстра</label>

</div>

</div>

<divclass="form-group">

<labelstyle="display: block;">Срок</label>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions1"id="inlineRadio4"[(ngModel)]="newHotelService.time"

value="0">

<labelclass="form-check-label"for="inlineRadio4">временна</label>

</div>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions1"id="inlineRadio5"[(ngModel)]="newHotelService.time"

value="1">

<labelclass="form-check-label"for="inlineRadio5">постоянна</label>

</div>

</div>

</div>

<divclass="form-group">

<buttonclass="btn offset-2 col-3"(click)="onAddDialogSubmitted()">Запази</button>

<buttonclass="btn offset-2 col-3"(click)="onDialogClosed()">Отказ</button>

</div>

</kendo-dialog>

<div>

<small>

<fa-icon[icon]="faInfo"></fa-icon> Натиснете върху съответната услуга, за да я коригирате!</small>

</div>

</ng-template>

</ngb-tab>

<ngb-tabid="tab-selectbyid2">

<ng-templatengbTabTitle>Стаи</ng-template>

<ng-templatengbTabContent>

<div>

<kendo-grid[kendoGridBinding]="rooms | async"[pageSize]="4"[pageable]="true"[sortable]="true"height="auto">

<kendo-grid-columnfield="name"title="Име"width="60">

<ng-templatekendoGridCellTemplatelet-room>

<span(click)="onAddRoomButtonClicked(room)">{{room.name}}</span>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnfield="extras"title="Екстри"width="180">

</kendo-grid-column>

<kendo-grid-columnfield="price"title="Цена за ден"width="80">

</kendo-grid-column>

<kendo-grid-columnwidth="80">

<ng-templatekendoGridCellTemplatelet-room>

<span(click)="onDeleteRoomButtonClicked(room)"style="text-decoration: underline; cursor: pointer;">Изтрий</span>

</ng-template>

</kendo-grid-column>

</kendo-grid>

</div>

<divclass="float-right mt-2">

<buttonclass="btn"(click)="onAddRoomButtonClicked()">Добави</button>

</div>

<kendo-dialogtitle="Изтриване на стая"\*ngIf="isDeleteRoomDialogOpened"(close)="onRoomDialogClosed()"[minWidth]="250"

[width]="450">

<pstyle="margin: 30px; text-align: center;">Искате ли да изтриете {{selectedRoom.name}}?</p>

<divclass="form-group">

<buttonclass="btn offset-2 col-3"(click)="onDeleteRoomSubmitted()">Да</button>

<buttonclass="btn offset-2 col-3"(click)="onRoomDialogClosed()">Не</button>

</div>

</kendo-dialog>

<kendo-dialogtitle="Добавяне/Редакция на стая"\*ngIf="isAddRoomDialogOpened"(close)="onRoomDialogClosed()"[minWidth]="250"

[width]="450">

<divclass="container">

<divclass="form-group">

<h6>Добавяне на услуга</h6>

<hr>

</div>

<divclass="form-group">

<inputtype="text"class="form-control mb-1"placeholder="Въведете име на услугата"[(ngModel)]="newHotelService.name">

<inputtype="text"class="form-control mb-1"placeholder="Въведете кратко описание"[(ngModel)]="newHotelService.description">

<inputtype="number"class="form-control mb-1"placeholder="Въведете цена за ден"[(ngModel)]="newHotelService.price">

</div>

<divclass="form-group">

<labelstyle="display: block;">Вид</label>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions"id="inlineRadio1"[(ngModel)]="newHotelService.type"

value="0">

<labelclass="form-check-label"for="inlineRadio1">услуга</label>

</div>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions"id="inlineRadio2"[(ngModel)]="newHotelService.type"

value="1">

<labelclass="form-check-label"for="inlineRadio2">екстра</label>

</div>

</div>

<divclass="form-group">

<labelstyle="display: block;">Срок</label>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions1"id="inlineRadio4"[(ngModel)]="newHotelService.time"

value="0">

<labelclass="form-check-label"for="inlineRadio4">временна</label>

</div>

<divclass="form-check form-check-inline">

<inputclass="form-check-input"type="radio"name="inlineRadioOptions1"id="inlineRadio5"[(ngModel)]="newHotelService.time"

value="1">

<labelclass="form-check-label"for="inlineRadio5">постоянна</label>

</div>

</div>

</div>

<divclass="form-group">

<buttonclass="btn offset-2 col-3"(click)="onAddRoomDialogSubmitted()">Запази</button>

<buttonclass="btn offset-2 col-3"(click)="onRoomDialogClosed()">Отказ</button>

</div>

</kendo-dialog>

<div>

<small>

<fa-icon[icon]="faInfo"></fa-icon> Натиснете върху името на стаята, за да я коригирате!</small>

</div>

</ng-template>

</ngb-tab>

<ngb-tab>

<ng-templatengbTabTitle>Отговорни лица</ng-template>

<ng-templatengbTabContent>

<div>

<kendo-grid[kendoGridBinding]="employees | async"[pageSize]="4"[pageable]="true"[sortable]="true"height="auto">

<kendo-grid-columnfield="name"title="Име"width="110">

<ng-templatekendoGridCellTemplatelet-employee>

<span\*ngIf="employee.id !== 1">{{employee.name}}</span>

</ng-template>

</kendo-grid-column>

<kendo-grid-columnwidth="80">

<ng-templatekendoGridCellTemplatelet-employee>

<span(click)="onDeleteEmployeeButtonClicked(employee)"style="text-decoration: underline; cursor: pointer;">Изтрий</span>

</ng-template>

</kendo-grid-column>

</kendo-grid>

<divclass="float-right mt-2">

<buttonclass="btn"(click)="onAddEmployeeButtonClicked()">Добави</button>

</div>

<kendo-dialogtitle="Изтриване на служител"\*ngIf="isDeleteEmployeeDialogOpened"(close)="onEmployeeDialogClosed()"[minWidth]="250"

[width]="450">

<pstyle="margin: 30px; text-align: center;">Искате ли да изтриете {{selectedRoom.name}}?</p>

<divclass="form-group">

<buttonclass="btn offset-2 col-3"(click)="onDeleteEmployeeSubmitted()">Да</button>

<buttonclass="btn offset-2 col-3"(click)="onEmployeeDialogClosed()">Не</button>

</div>

</kendo-dialog>

<kendo-dialogtitle="Добавяне на служител"\*ngIf="isAddEmployeeDialogOpened"(close)="onEmployeeDialogClosed()"[minWidth]="250"

[width]="450">

<divclass = "form-group">

<inputtype="text"class="form-control mb-1"placeholder="Въведете име на служителя"[(ngModel)]="newEmployee.name">

</div>

<divclass="form-group">

<buttonclass="btn offset-2 col-3"(click)="onAddEmployeeSubmitted()">Запази</button>

<buttonclass="btn offset-2 col-3"(click)="onEmployeeDialogClosed()">Отказ</button>

</div>

</kendo-dialog>

</div>

</ng-template>

</ngb-tab>

</ngb-tabset>

</div>

</div>

Admin-settings.component.html

import { Component, OnInit, OnDestroy } from'@angular/core';

import { Subscription, Observable } from'rxjs';

import { faInfoCircle } from'@fortawesome/free-solid-svg-icons';

import { HotelProductViewModel } from'../../../models/public-models/view-models/hotel-product-view-model';

import { AdminService } from'../../../services/admin-services/admin.service';

import { PublicService } from'../../../services/public.service';

import { HttpErrorResponse } from'@angular/common/http';

import { HotelProductBindingModel } from'../../../models/admin-models/binding-models/hotel-product-binding-model';

import { RoomBindingModel } from'../../../models/admin-models/binding-models/room-binding-model';

import { RoomExtendedViewModel } from'../../../models/admin-models/view-models/room-extended-view-model';

import { EmployeeBindingModel } from'../../../models/admin-models/binding-models/employee-binding-model';

import { EmployeeViewModel } from'../../../models/admin-models/view-models/employee-view-model';

@Component({

  selector: 'app-admin-settings',

  templateUrl: './admin-settings.component.html',

  styleUrls: ['./admin-settings.component.scss']

})

exportclass AdminSettingsComponent implements OnInit, OnDestroy {

  faInfo = faInfoCircle;

  // Hotel services variables

  isDeleteDialogOpened: boolean;

  isAddProductDialogOpened: boolean;

  isEditProduct: boolean;

  newHotelService: HotelProductBindingModel;

  selectedService: HotelProductViewModel;

  hotelServices: Observable<HotelProductViewModel[]>;

  // Rooms variables

  isAddRoomDialogOpened: boolean;

  isDeleteRoomDialogOpened: boolean;

  isEditRoom: boolean;

  newRoom: RoomBindingModel;

  selectedRoom: RoomExtendedViewModel;

  rooms: Observable<RoomExtendedViewModel[]>;

  // Employees variables

  isAddEmployeeDialogOpened: boolean;

  isDeleteEmployeeDialogOpened: boolean;

  newEmployee: EmployeeBindingModel;

  selectedEmployee: EmployeeViewModel;

  employees: Observable<EmployeeViewModel[]>;

  private subscription: Subscription;

  constructor(private adminService: AdminService, private publicService: PublicService) {

    // Hotel services initialize

    this.isDeleteDialogOpened = false;

    this.isAddProductDialogOpened = false;

    this.isEditProduct = false;

    this.newHotelService = new HotelProductBindingModel();

    this.selectedService = new HotelProductViewModel();

    // Room initialize

    this.isAddRoomDialogOpened = false;

    this.isDeleteRoomDialogOpened = false;

    this.isEditRoom = false;

    this.newRoom = new RoomBindingModel();

    this.selectedRoom = new RoomExtendedViewModel();

    // Employee initialize

    this.isAddEmployeeDialogOpened = false;

    this.isDeleteEmployeeDialogOpened = false;

    this.newEmployee = new EmployeeBindingModel();

    this.selectedEmployee = new EmployeeViewModel();

    this.subscription = new Subscription();

  }

  ngOnInit(): void {

    this.getAllHotelServices();

    this.getAllRooms();

    this.getAllEmployees();

  }

  // Service Methods

  onAddServiceButtonClicked(service: HotelProductViewModel | null): void {

    if (service !== null) {

      this.isEditProduct = true;

      this.selectedService = service;

      this.newHotelService.name = service.name;

      this.newHotelService.price = service.price;

      this.newHotelService.description = service.description;

      this.newHotelService.time = service.time;

      this.newHotelService.type = service.type;

    } else {

      this.selectedService = new HotelProductViewModel();

    }

    this.isAddProductDialogOpened = true;

  }

  onAddDialogSubmitted(): void {

    if (!this.isEditProduct) {

      this.subscription.add(this.adminService.addHotelService(this.newHotelService).subscribe(() => {

        this.getAllHotelServices();

        alert('Записът беше успешно добавен!');

      }, (errorResponse: HttpErrorResponse) => {

        if (errorResponse.status >500) {

          alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

        }

      }));

    } else {

      this.subscription.add(this.adminService.editHotelService(this.selectedService.id, this.newHotelService).subscribe(() => {

        this.getAllHotelServices();

        alert('Записът беше редактиран добавен!');

      }, (errorResponse: HttpErrorResponse) => {

        if (errorResponse.status >500) {

          alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

        }

      }));

    }

    this.isAddProductDialogOpened = false;

    this.newHotelService = new HotelProductBindingModel();

    this.isEditProduct = false;

  }

  onDeleteServiceButtonClicked(service: HotelProductViewModel): void {

    this.isDeleteDialogOpened = true;

    this.selectedService = service;

  }

  onDialogClosed(): void {

    this.isDeleteDialogOpened = false;

    this.isAddProductDialogOpened = false;

    this.isEditProduct = false;

    this.newHotelService = new HotelProductBindingModel();

  }

  onDeleteSubmitted(): void {

    this.subscription.add(this.adminService.deleteHotelService(this.selectedService.id).subscribe(() => {

      this.getAllHotelServices();

      alert('Операцията беше успешна');

    }, (errorResponse: HttpErrorResponse) => {

      if (errorResponse.status >500) {

        alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

      }

    }));

    this.isDeleteDialogOpened = false;

  }

  // Room Methods

  onRoomDialogClosed(): void {

    this.isDeleteRoomDialogOpened = false;

    this.isAddRoomDialogOpened = false;

    this.isEditRoom = false;

    this.newRoom = new RoomBindingModel();

  }

  onAddRoomButtonClicked(room: RoomExtendedViewModel | null): void {

    // TODO: Have to fix model binding

    if (room !== null) {

      this.isEditProduct = true;

      this.selectedRoom = room;

      this.newRoom.name = room.name;

      this.newRoom.price = room.price;

    } else {

      this.selectedRoom = new RoomExtendedViewModel();

    }

    this.isAddProductDialogOpened = true;

  }

  onAddRoomDialogSubmitted(): void {

    if (!this.isEditRoom) {

      this.subscription.add(this.adminService.addRoom(this.newRoom).subscribe(() => {

        this.getAllRooms();

        alert('Записът беше успешно добавен!');

      }, (errorResponse: HttpErrorResponse) => {

        if (errorResponse.status >500) {

          alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

        }

      }));

    } else {

      this.subscription.add(this.adminService.editRoom(this.selectedRoom.id, this.newRoom).subscribe(() => {

        this.getAllRooms();

        alert('Записът беше редактиран добавен!');

      }, (errorResponse: HttpErrorResponse) => {

        if (errorResponse.status >500) {

          alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

        } else {

          alert(errorResponse.message);

        }

      }));

    }

    this.isAddRoomDialogOpened = false;

    this.newRoom = new RoomBindingModel();

    this.isEditRoom = false;

  }

  onDeleteRoomButtonClicked(room: RoomExtendedViewModel): void {

    this.isDeleteRoomDialogOpened = true;

    this.selectedRoom = room;

  }

  onDeleteRoomSubmitted(): void {

    this.subscription.add(this.adminService.deleteRoom(this.selectedRoom.id).subscribe(() => {

      this.getAllRooms();

      alert('Операцията беше успешна!');

    }, (errorResponse: HttpErrorResponse) => {

      if (errorResponse.status >500) {

        alert('Възникна проблем със сървъра. Моля свържете се с администратор!');

      }

    }));

    this.isDeleteRoomDialogOpened = false;

  }

  // Employees Methods

  onAddEmployeeButtonClicked(): void {

    this.isAddEmployeeDialogOpened = true;

  }

  onAddEmployeeSubmitted(): void {

    this.subscription.add(this.adminService.addEmployee(this.newEmployee).subscribe(() => {

      this.getAllEmployees();

      alert('Записът беше успешно добавен!');

    }, (errorResponse: HttpErrorResponse) => {

      if (errorResponse.status >500) {

        alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

      }

    }));

    this.newEmployee = new EmployeeBindingModel();

    this.isAddEmployeeDialogOpened = false;

  }

  onDeleteEmployeeButtonClicked(employee: EmployeeViewModel): void {

    this.selectedEmployee = employee;

    this.isDeleteEmployeeDialogOpened = true;

  }

  onEmployeeDialogClosed(): void {

    this.isDeleteEmployeeDialogOpened = false;

    this.isAddEmployeeDialogOpened = false;

    this.newEmployee = new EmployeeBindingModel();

  }

  onDeleteEmployeeSubmitted(): void {

    this.subscription.add(this.adminService.deleteEmployee(this.selectedEmployee.id).subscribe(() => {

      this.getAllEmployees();

      alert('Операцията беше успешна');

    }, (errorResponse: HttpErrorResponse) => {

      if (errorResponse.status >500) {

        alert('Възникна проблем със сървъра.Моля свържете се с администратор!');

      }

    }));

    this.isDeleteEmployeeDialogOpened = false;

  }

  ngOnDestroy(): void {

    this.subscription.unsubscribe();

  }

  private getAllHotelServices(): void {

    this.hotelServices = this.publicService.getHotelServices().pipe();

  }

  private getAllRooms(): void {

    this.rooms = this.adminService.getRoomsDetailed().pipe();

  }

  private getAllEmployees(): void {

    this.employees = this.adminService.getEmployees().pipe();

  }

}

**package** org.zerhusen.model;

**import** java.util.Date;

**public** **class** UserMenu {

**private** Date date;

**private** **int** breakfast;

**private** **int** lunch;

**private** **int** dinner;

**public** Date getDate() {

**return** date;

}

**public** **void** setDate(Date date) {

**this**.date = date;

}

**public** **int** getBreakfast() {

**return** breakfast;

}

**public** **void** setBreakfast(**int** breakfast) {

**this**.breakfast = breakfast;

}

**public** **int** getLunch() {

**return** lunch;

}

**public** **void** setLunch(**int** lunch) {

**this**.lunch = lunch;

}

**public** **int** getDinner() {

**return** dinner;

}

**public** **void** setDinner(**int** dinner) {

**this**.dinner = dinner;

}

}

**package** org.zerhusen.security.service;

**public** **class** CreatingUserUtil {

**public** String CreateUsername(String register) {

String[] getTempName = register.split(" ");

String username, createdUsername = "";

username = getTempName[0];

username = ConvertToEnglish(username);

createdUsername += username.charAt(0);

username = getTempName[1];

username = ConvertToEnglish(username);

createdUsername += username + getTempName[0].length() + getTempName[1].length();

**return** createdUsername;

}

**private** String ConvertToEnglish(String Text) {

String temp = "";

**for**(**int** i = 0; i < Text.length(); i++) {

**if**(Text.charAt(i) == 'А' || Text.charAt(i) == 'а') {

temp += "a";

}**else** **if**(Text.charAt(i) == 'Б' || Text.charAt(i) == 'б') {

temp += "b";

}**else** **if**(Text.charAt(i) == 'В' || Text.charAt(i) == 'в') {

temp += "v";

}**else** **if**(Text.charAt(i) == 'Г' || Text.charAt(i) == 'г') {

temp += "g";

}**else** **if**(Text.charAt(i) == 'Д' || Text.charAt(i) == 'д') {

temp += "d";

}**else** **if**(Text.charAt(i) == 'Е' || Text.charAt(i) == 'е') {

temp += "e";

}**else** **if**(Text.charAt(i) == 'Ж' || Text.charAt(i) == 'ж') {

temp += "j";

}**else** **if**(Text.charAt(i) == 'З' || Text.charAt(i) == 'з') {

temp += "z";

}**else** **if**(Text.charAt(i) == 'И' || Text.charAt(i) == 'и') {

temp += "i";

}**else** **if**(Text.charAt(i) == 'Й' || Text.charAt(i) == 'й') {

temp += "i";

}**else** **if**(Text.charAt(i) == 'К' || Text.charAt(i) == 'к') {

temp += "k";

}**else** **if**(Text.charAt(i) == 'Л' || Text.charAt(i) == 'л') {

temp += "l";

}**else** **if**(Text.charAt(i) == 'М' || Text.charAt(i) == 'м') {

temp += "m";

}**else** **if**(Text.charAt(i) == 'Н' || Text.charAt(i) == 'н') {

temp += "n";

}**else** **if**(Text.charAt(i) == 'О' || Text.charAt(i) == 'о') {

temp += "o";

}**else** **if**(Text.charAt(i) == 'П' || Text.charAt(i) == 'п') {

temp += "p";

}**else** **if**(Text.charAt(i) == 'Р' || Text.charAt(i) == 'р') {

temp += "r";

}**else** **if**(Text.charAt(i) == 'С' || Text.charAt(i) == 'с') {

temp += "s";

}**else** **if**(Text.charAt(i) == 'Т' || Text.charAt(i) == 'т') {

temp += "t";

}**else** **if**(Text.charAt(i) == 'У' || Text.charAt(i) == 'у') {

temp += "u";

}**else** **if**(Text.charAt(i) == 'Ф' || Text.charAt(i) == 'ф') {

temp += "f";

}**else** **if**(Text.charAt(i) == 'Х' || Text.charAt(i) == 'х') {

temp += "h";

}**else** **if**(Text.charAt(i) == 'Ц' || Text.charAt(i) == 'ц') {

temp += "tc";

}**else** **if**(Text.charAt(i) == 'Ч' || Text.charAt(i) == 'ч') {

temp += "ch";

}**else** **if**(Text.charAt(i) == 'Ш' || Text.charAt(i) == 'ш') {

temp += "sh";

}**else** **if**(Text.charAt(i) == 'Щ' || Text.charAt(i) == 'щ') {

temp += "sht";

}**else** **if**(Text.charAt(i) == 'Ъ' || Text.charAt(i) == 'ъ') {

temp += "u";

}**else** **if**(Text.charAt(i) == 'ь') {

temp += "i";

}**else** **if**(Text.charAt(i) == 'Ю' || Text.charAt(i) == 'ю') {

temp += "iu";

}**else** **if**(Text.charAt(i) == 'Я' || Text.charAt(i) == 'я') {

temp += "q";

}

}

**if**(temp.length() == 0) {

**return** Text;

}

**return** temp;

}

}

package org.zerhusen.security.service;

import java.util.ArrayList;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.zerhusen.model.FilterRooms;

import org.zerhusen.model.Reservation;

import org.zerhusen.model.RoomInfo;

import org.zerhusen.model.Rooms;

import org.zerhusen.security.repository.ReservationRepository;

import org.zerhusen.security.repository.RoomsRepository;

@Service

public class ReservationService {

@Autowired

private RoomsRepository roomsRepository;

@Autowired

private ReservationRepository reservationRepository;

public List<RoomInfo> findFreeRoom(FilterRooms filterRooms) {

boolean IsRoomFree = true;

List<RoomInfo> availableRooms = new ArrayList<RoomInfo>();

if(filterRooms.getRoomID()==0) {

for(Rooms rooms: roomsRepository.findAll()) {

if(filterRooms.getRoomType() == rooms.getType() || filterRooms.getRoomType() == 0) {

filterRooms.setRoomID(rooms.getID());

for(Reservation reservated: reservationRepository.findAll()) {

if(filterRooms.getRoomID() == reservated.getRoomId()) {

if(filterRooms.getStartDate().after(reservated.getEndDate()) || filterRooms.getEndDate().before(reservated.getStartDate())) {

IsRoomFree = true;

} else {

IsRoomFree = false;

break;

}

} else {

IsRoomFree = true;

}

}

if(IsRoomFree) {

System.out.println(rooms.getID());

availableRooms.add(new RoomInfo(rooms.getID(), rooms.getName(), rooms.getType(), rooms.getRoomPrice()));

}

}

}

}

return availableRooms;

}

}

package org.zerhusen.security.service;

import java.util.ArrayList;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.zerhusen.model.PetDetails;

import org.zerhusen.model.Reservation;

import org.zerhusen.model.ReservationDetails;

import org.zerhusen.model.Rooms;

import org.zerhusen.model.security.User;

import org.zerhusen.security.repository.PetDetailsRepository;

import org.zerhusen.security.repository.ReservationRepository;

import org.zerhusen.security.repository.RoomsRepository;

import org.zerhusen.security.repository.UserRepository;

@Service

public class ReservationDetailsService {

@Autowired

private UserRepository userRepository;

@Autowired

private ReservationRepository reservationRepository;

@Autowired

private RoomsRepository roomsRepository;

@Autowired

private PetDetailsRepository petDetailsRepository;

public List<ReservationDetails> getAllReservations(){

List<ReservationDetails> reservations = new ArrayList<ReservationDetails>();

for(Reservation reservation: reservationRepository.findAll()) {

ReservationDetails reservationDetails = new ReservationDetails();

reservationDetails.setId(reservation.getID());

reservationDetails.setStartDate(reservation.getStartDate());

reservationDetails.setEndDate(reservation.getEndDate());

reservationDetails.setDeposit(reservation.getDeposit());

reservationDetails.setStatus(reservation.getStatus());

for(Rooms rooms: roomsRepository.findAll()) {

if(reservation.getRoomId() == rooms.getID()) {

reservationDetails.setRoomName(rooms.getName());

break;

}

}

for(User user: userRepository.findAll()) {

for(PetDetails petDetails: petDetailsRepository.findAll()) {

if(user.getId() == petDetails.getUserId()) {

reservationDetails.setUserId(user.getId());

if(petDetails.getEmployeeId() == 1 || petDetails.getAge() == 0 || petDetails.getBreed().isEmpty() || petDetails.getType().isEmpty() || petDetails.getPassport().isEmpty()) {

reservationDetails.setIsComplete(false);

}

else {

reservationDetails.setIsComplete(true);

}

}

}

if(reservation.getUserName().equals(user.getUsername())) {

reservationDetails.setName(user.getFirstname() + " " + user.getLastname());

break;

}

}

reservations.add(reservationDetails);

}

return reservations;

}

public void saveReservationById(Long reservationId, ReservationDetails reservationDetails) {

Reservation reservation = new Reservation();

for(Rooms rooms: roomsRepository.findAll()) {

if(rooms.getName().equals(reservationDetails.getRoomName())) {

reservation.setRoomId(rooms.getID());

break;

}

}

reservation.setID(reservationId);

reservation.setStartDate(reservationDetails.getStartDate());

reservation.setEndDate(reservationDetails.getEndDate());

reservation.setDeposit(reservationDetails.getDeposit());

reservation.setStatus(reservationDetails.getStatus());

1. Приложение на една страница или Single Page Application е архитектура, която дава възможност за динамично зареждане на частите, от които е изградено приложението; [↑](#footnote-ref-2)
2. Data binding –е основна техника, която свързва източниците на данни от дадена услуга с консуматора на тези данни, като синхронизира тяхната работа. [↑](#footnote-ref-3)
3. MVC – Model View Controller: модела е дата която се използва от програмата; изгледа означава показване на обектите в приложението; контролера обновява модела и изгледа едновременно. [↑](#footnote-ref-4)
4. <http://agilemodeling.com/artifacts/essentialUseCase.htm> [↑](#footnote-ref-5)
5. Single Page Application [↑](#footnote-ref-6)
6. reCAPTCHA е модул на Google, който осигурява защита от BOTNET атаки чрез използване на таен ключ изискван при изпращането на дадена заявка. [↑](#footnote-ref-7)