ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ

****

Катедра „Компютърни науки и технологии“

**ДИПЛОМНА РАБОТА**

**Тема:**

Система за обслужване нуждите

на бежански център

**Изготвил:** Илиян Станчев Станчев

**Специалност:** Софтуерни и Интернет технологии

**Факултетен номер:** 18621750

ТУ Варна, 2022 г. Ръководител:

10.06.2022г. /х. ас. Д. Димитров/

**Съдържание**

[1. Увод 3](#_Toc105855537)

[2. Анализ на проблема и избор на технологии 5](#_Toc105855538)

[2. 1. Анализ на проблема 5](#_Toc105855539)

[2. 2. Анализ на конкуренцията 6](#_Toc105855540)

[3. Избор на технологии 9](#_Toc105855541)

[3.1 PostgreSQL 10](#_Toc105855542)

[3.2 Java 13](#_Toc105855543)

[3.3 Spring Boot 15](#_Toc105855544)

[3.4 Hibernate 18](#_Toc105855545)

[3.5 React 19](#_Toc105855546)

[3.6 PayPal JavaScript SDK 22](#_Toc105855547)

[3.7 Google reCAPTCHA 23](#_Toc105855548)

[3.8 JavaMail library 23](#_Toc105855549)

[3.9 Abstract API 24](#_Toc105855550)

[3.10 Google Maps API 25](#_Toc105855551)

[3.11 Twilio 26](#_Toc105855552)

[4. Проектиране на системата 27](#_Toc105855553)

[7. Речник 27](#_Toc105855554)

[8. Източници на информация 28](#_Toc105855555)

# 1. Увод

Изборът на тема за дипломна работа не е лесна задача. Трябва да се отговори на много въпроси като: „Дали темата е интересна?“, „Дали има какво да се разработва по нея?“, „Дали е достатъчно комплексна?“, „Дали би била полезна в днешно време?“. В днешно време е трудно човек да създаде софтуер, който досега не е виждан и представлява нещо ново за потенциалните му потребители. Все пак живеем в 21-ви век и ИТ секторът е сериозно развит.

Моята тема не представлява нищо ново под слънцето. По – скоро е нещо актуално в днешно време. Все пак, макар и като на филм, наблюдаваме развитието на една война и повечето от нас се опитват да помогнат на хората, които са принудени да изоставят домовете си, за да се спасят от дебнещата ги опасност. Това бяха мотивите ми да се заема със система за обслужване нуждите на бежански център.

В момента има бежанци по целия свят. Не казвам, че е заради войната, защото и преди нея отново се е налагало на държавите да се справят и да помагат на бежанци, които напускат страната си по една или друга причина. Управлението на всяка държава се заема с най – голяма отговорност, що се отнася до бежанци, но освен държавата има редица частни организации, които подпомагат на хора, изпаднали в такова положение.

Тези организации, наричани често просто „Бежански центрове“, се поставят на мястото на хората от другата страна и се опитват да им осигурят всичко, което са загубили за толкова кратък период от време. Определено е голям стрес за един човек, който бива „прокуден“ от дома и страната си и започва да скита и да си търси нов подслон. Тук говорим за първостепенни нужди като покрив над главата и храна, които в даден момент започват да липсват на тези хора. Няма как да не спомена, че търсенето на такива нужди в една непозната държава, говореща на непознат език никак не е лесно. Една от ползите на бежанските центрове, е че играе ролята на посредник между бежанците и услугите, които може да се използват в държавата.

Друг основен проблем е, че дори и да има такива центрове, как е възможно един току що пристигнал човек без никакви връзки и без да познава местния език да намери такова място? В днешно време всичко е много по – лесно, тъй като съществува Интернет. Но освен Интернет е необходимо някой да е създал платформа или да е предоставил информация, че съществува бежански център, който да посрещне бежанците.

Условията на живот за такива хора няма как да се нарекат „розови“. Но за сметка на това текущите условия на живот изискват създаването на по – удобни решения на проблеми, които засягат голяма част от обществото. Пример за такъв проблем е именно обслужването на бежански център и лесната ориентация за хората в нужда. Такива проблеми изискват технологични решения. А понякога важни проблеми се решават от не чак толкова сложни технологични решения.

# 

# 2. Анализ на проблема и избор на технологии

## 2. 1. Анализ на проблема

Услугите на бежански център се използват основно за обслужване на бежанци, но трябва да доставят достатъчно административни функционалности, за да може да се постигне максимална автоматизация на процесите, като по този начин се намалят до минимум възможностите за допускане на грешки от човешкия фактор. В бъдеще най – вероятно ще бъде възможно такива системи да бъдат управлявани изцяло от технологичните решения на изкуствения интелект, но дотогава трябва да се осигури удобство на потребителите, които си взаимодействат със системата.

Приложения от този вид, специализирани в обслужването на хора, трябва да отговарят на множество фактори. На първо място е защитата на данните, особено ако системата съхранява чувствителни данни. След това е самото взаимодействие на потребителя със системния интерфейс. Той трябва да е достатъчно прост, за да може да се осигури лесната ориентация на потребителите в него. Освен това трябва менютата и подменютата да бъдат разположени на правилните места и да са логически групирани, за да може човек без да познава системата лесно да намери това, което търси. И не на последно място трябва да се осигури конкурентен на пазара продукт, който е взел впредвид наличните решения и се е постарал да създаде нещо ново или продукт с повече плюсове, отколкото минуси.

## 2. 2. Анализ на конкуренцията

Анализът на конкуренцията е основна стъпка при създаването на софтуер. Много често идеите, които искаме да реализираме като продукт, са вече реализирани. Трудно се измисля нещо ново и невиждано досега в 21 век. Чрез анализ на конкуренцията се сравняват функционалностите на други подобни продукти и може да се даде реалистична преценка дали нашият продукт ще е по – добър от конкурентните продукти.

Конкретно за Варна няма изградена онлайн платформа, която да предоставя възможност на потребители да помогнат онлайн, например чрез дарения. В официалния сайт на варна има информация за самия бежански център, неговото местоположение и контакти.

Системи за обслужване на бежанци по света обаче има доста. Платформите, които съм подбрал за анализ на конкуренцията са The Refugee Center и Refugee Dream Center. Двете приложения са първото нещо, което излиза в Google и определено задоволяват нуждите на съвременна информационна и обслужваща система за бежански център.





|  |  |  |
| --- | --- | --- |
| Анализ на конкуренцията | The Refugee Center | Refugee Dream Center |
| Съвпадение на нуждите на реалният свят с предоставените от приложение | Системата предоставя достатъчно информация както за своята цел, така и допълнителна информация как да станеш доброволец или да започнеш работа при тях. Регистрирането и работата с мигранти става на място в техните офиси. Освен основните нужди на един център за обслужване на бежанци, предлагат също и социални, посреднически услуги, както и помощ при ученето на нов език, както и попълването на бланки на чужд език. | Системата предоставя информация за идеите на приложението. История и информация за отминали и бъдещи мероприятия. Отново има възможност да станеш доброволец, като дариш средства или приютиш имигрант при теб. Има достъпно обобщение за постигнатите цели през изминалата година. В приложението има информация за бежанци къде могат да намерят клиники, транспорт, храна и други услуги. |
| Потребителски контрол и свобода на действията | Като потребител имаш възможността да станеш доброволец, чрез попълване на форма за доброволец, която трябва да бъде одобрена от оторизиран за това потребител. Правенето на дарения е възможно и от нерегистрирани потребители. | Като потребител можеш единствено да дариш средства. Всичко останало, като приютяване на имигрант, ставане на доброволец и взимане на участие в мероприятия става чрез изпращане на указан имейл. |
| Консистентно представяне на информация и стандарти | Прост дизайн, който позволява лесното намиране на информацията, от която се нуждаеш. Опциите не са много, така че е почти невъзможно да се обърка потребителят. | Лесно се намира нужната информация. Има  лента за навигация, за да можеш лесно да се върнеш назад или да отвориш нова страница с информация. |
| Предотвратяване на грешки и тяхното прехващане | Като обикновен потребител системата позволява единствено въвеждането на форма за доброволец, която ограничава въвеждането на данни. Налични са валидации за непопълнени полета, имейл адрес и други. | Форма за въвеждане на данни от потребител е опцията за дарение, където има валидации за въведените данни.  Има и форма за изпращане на съобщение, като има валидации за имейл адрес, за да може да получиш отговор. |
| Осигуряване на интерфейс пред действие по памет | Интерфейса е доста интерактивен, можеш лесно да се ориентираш и да достъпиш нужната информация от различни места. | Можеш лесно да се ориентираш и да намериш каквото ти трябва. Приложението има препратки към други сайтове, ако потребителят търси нещо по – конкретно. |
| Гъвкавост и ефективност | Ефективен за информационните нужди на потребителите, както и за правене на дарения. | Ефективен за информационните нужди на потребителите, както и за правене на дарения. |
| Обратна връзка | Има страница с контакти: адреси на работещи информационни центрове, имейл, телефони за връзка. | Страница за контакти, от която можеш да изпратиш съобщение. Имейл, адрес на офиса, работно време на офиса, телефон за връзка. |

# 3. Избор на технологии

Изборът на технологии за реализация на всеки продукт зависи много от целите, които трябва да се реализират и крайния срок. Друг важен фактор е, че технологиите, които се използват, е добре да са познати за екипа, който ще разработва софтуера, тъй като непознати технологии изискват време за запознаване, което може да доведе до пресрочване на крайния срок. От друга страна изборът на непознати технологии е възможно да мотивира амбицираните разработчици да научат нещо ново, да се развиват и да стават все по – комплексни в областта си.

Видът на приложението също е много важен и трябва да се определи в зависимост от потребителските нужди и изискванията към софтуера. Специално за бежански център най- добрият вариант е Уеб приложение, тъй като Интернет е първото нещо, което би отворил всеки краен потребител, ако се интересува от нещо ново. Освен това Уеб приложенията могат да се достъпват от всякакви операционни системи и е лесно да всеки да сподели линк към даден сайт на приятел или в социална мрежа.

За мен изборът на технологии беше много лесен. На първо място исках да използвам технологии, които намирам за интересни, независимо колко съм запознат с тях. Идеята беше да науча нещо ново, да опитам нови неща и да се постарая да ги имплементирам правилно.

## 3.1 PostgreSQL



*Фиг. 1 – PostgreSQL лого*

PostgreSQL е най – напредналата и най – модерна open source релационна база. Мощна релационна база с повече от 30 години активна разработка, което и печели силна репутация, надеждност, нови функционалности и производителност.

PostgreSQL е обектно-релационна система за менажиране на база данни, базирана на POSTGRES, разработен от Отдел за компютърна наука Бъркли, Калифорнийски университет. Базата поддържа голяма част от стандартните SQL функционалности, както и по- модерни такива, като комплексни заявки, тригери, динамични View-та, конкурентен контрол на версиите. Осигурява възможност за дефиниране на потребителски типове данни, функции, оператори и други.

Голям плюс е либералният лиценз, който позволява PostgreSQL да бъде използвана, променяна и разпределяна напълно безплатно от всеки, било то за лично, академично или комерсиално ползване.

Самият проект POSTGRES, върху който е базирана релационната база, е спонсориран през 1986 г. с цел да се използва за системи за анализи, мониторинг и следене. Предшественик на POSTGRES е INGRES, който представлява система за управление на релационни бази, също създаден от Калифорнийския университет. Самото име идва от това, че проектът е продължение на INGRES ( POST INGRES ).

През 1994 година към POSTGRES се добавя SQL интерпретатор и името на системата става Postgres95. Кодът на Postgres95 е преобразуван напълно до ANSI C и размерът му е смъкнат с 25 процента. По – малко код водят дори и до 50 % по – голяма производителност на системата спрямо предшественикът и POSTGRES.

Мнозина казват, че логото на PostgreSQL е слон, защото слоновете имат много голяма памет и на практика казват, че не забравят нищо през живота си. Друга, може би по – достоверна история е, че PostreSQL използва система за репликиране на бази данни, наречена Slony-I. Името Slony идва от руски ( слоны), което означава именно слонове.

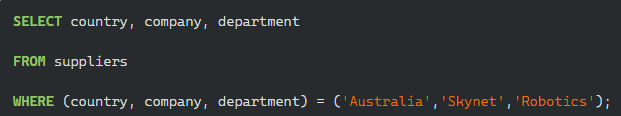
Някой от интересните функционалности на PostgreSQL в днешно време са:

* Връщане на данните в таблицата в единична колона:



Взима всички записи в дадена таблица и връща данните в текстов формат, разделени със запетая.

* Where клауза с множество колони и стойности:



По този начин може да се избегнат поредица от AND условия. Същата множествена клауза може да се използва и с IN условие, като за всяка колона се изброяват множество допустими стойности.

* Таблиците са типове данни





В PostgreSQL всяка създадена таблица може да се използва като тип при създаване на друга таблица.

При добавяне на данни в таблица, която е данна в друга таблица трябва да се разделят данните със скоби:



## 3.2 Java

*Фиг. 2 – Java лого*

Java е базиран на класове, обектно – ориентиран език за програмиране от високо ниво, който е създаден да позволява възможно най – оптимално преизползване на програмен код, разпределен в отделни модули. Идеята на Java е да помогне на програмистите да „пишат веднъж, да използват навсякъде“ ( от англ. Write once, run anywhere „WORA“ ). Това означава, че компилиран Java код може да се иползва на всякакви платформи, които поддържат Java без нужда от рекомпилиране.

Синтаксисът на Java е подобен на този на C++, но с по – ограничен достъп до функционалностите на ниско ниво. Създаването и започва през 1991г., писана е на C++.

През 2019 г. Java е един от най – разпространените програмни езици с над 9 милиона разработчици, според GitHub. Използва се най – вече за клиент – сървър приложения.

Създаването на Java е имало пет основни цели:

* Да бъде прост, обектно – ориентиран и познат. ( Познат, защото прилича на C++, а по това време той е бил най – известният програмен език )
* Да бъде мощен и сигурен.
* Да бъде неутрален към архитектури и лесно трансферим.
* Да изпълнява команди с голяма производителност.
* Да бъде интерпретируем, многонишков и динамичен.

Тези цели са все още важат при създаването на всеки един проект и всеки

разработчик трябва да се старае да ги спазва.

Може би най – големият плюс на Java като наследник на C++ е именно автоматичния garbage collector. Заделянето и освобождаването на памет по време на жизнения цикъл на всеки обект се менажира от програмния език без необходимост от намеса на разработчик. Нужно е единствено да укажем кога ще бъде създаден един обект, а Java е отговорна за връщането на паметта обратно след като обектът не се използва вече.

Колекторът работи на следния принцип: Ако памет, която вече не се реферира от обект остане незачистена, колекторът освобождава тази недостижима памет. Това обаче не означава, че не може да се случи memory leak, ако програмен код държи референция към обект, който вече не е нужен. Това е често срещан проблем, когато обекти, които се използват веднъж, се съдържат в контейнери, които все още се използват.

Както споменах по – рано синтаксисът на Java е силно взаимстван от този на C++, като има малки промени. Всичкият код е разписан в класове. За разлика от C++ където имаме структури от данни, в Java има единствено класове, които могат да послужат както за съхранение на данни, така и за комплексни функционалности. За разлика от C++, в Java нямаме пренаписване на оператори, множествено наследяване, макар че множественото наследяване може да се постигне чрез интерфейси.

Програмният език не спира да се развива. Тази година излезе версия 18, а използването му расте все повече и все повече. Голям принос към използването на Java в днешни дни имат мобилните приложения на Android, които се пишат главно на Java. Освен за мобилни приложения, Java намира широко приложение в създаването на Desktop, Уеб приложения, Уеб сървъри и софтуери за връзка с бази данни.

## 3.3 Spring Boot



*Фиг. 3 – Spring Boot лого*

Java Spring Framework е популярен open source framework, който е предназначен за създаване на самостоятелни приложения, които вървят на Java виртуална машина ( JVM ). Spring Framework е базиран на архитектура MVC ( Model, View, Controller ), като това улеснява до голяма степен разделението на различните модули в приложението и води до по - лесна и същевременно възможност за по – комплексна разработка.

Spring Boot от друга страна е улеснен инструмент, създаден от Spring Framework, който помага на разработчиците да създават микросервизи и приложения по – бързо и по – лесно чрез тези три основни функционалности:

* Автоматична конфигурация
* Догматичен подход към конфигурацията
* Способността да създава самостоятелни приложения

Spring Boot предлага т.нар “dependency injection”, което позволява на обектите да дефинират използването на други обекти в себе си, които ще бъдат използвани в по – късен етап. Това позволява автоматично менажиране на обекти в даден контейнер.

Големият плюс на Spring Boot е именно неговата автоматична конфигурация, която улеснява възможно най – много разработчика. През официалния сайт може да се изтегли стартов проект, който съдържа всичко необходимо да стартира сървър без допълнителни настройки. Това е възможно, тъй като Spring не разчита на външен Уеб сървър. По време на инициализация на проекта се настройва вграден Уеб сървър като Tomcat или Netty.

Както споменах по – рано Spring Boot е конфигуриран инструмент, създаден от Spring Framework. Всичко, което може да се постигне със Spring Boot е постижимо със Spring Framework, но чрез самостоятелната конфигурация на Spring Boot нещата стават много по – лесно. От друга страна тази конфигурация ограничава да се дефинират потребителски настройки. Spring Framework позволява свободата да настройш всичко сам.

Голямо удобство за Spring са анотациите, които може да се използват при създаване на приложение. Някой от тях са:

* **@Bean**

Тази анотация сигнализира на ниво метод и индикира че метод ще върне Bean обект, който ще се менажира от контейнера. Самият Bean обект обикновено е инициализиран обект с някаква настройка. Тази анотация ни позволява да използваме инициализирания обект с нужните настройки през от контекста на друг клас.

* **@Configuration**

Анотация на ниво клас, която съдържа методи, които връщат @Bean обекти с предефинирани настройки.

* **@Service**

С тази анотация указваме, че класът се използва за бизнес логика и менажира действията върху данновия слой.

* **@Repository**

Това са класове, които достъпват данните в базата данни директно. Така наречените Data Access Object ( DAO ) класове. Те се грижат за Create/Read/Update/Delete ( CRUD ) операции на данните в базата.

* **@Controller**

Тази анотация индикира, че класът ще служи за обработка на Уеб заявки. Съдържа в себе си обработчици за различни заявки, които се инициират от клиентското приложение и спрямо входните данни връща резултат на крайния потребител.

* **@Autowired**

Управлява инициализацията на member в клас, като по този начин ограничава възможността да се достъпва неинициализиран обект, което води до терминиране на приложението.

* **@Component**

Указва, че класът ще се превърне в Bean по време на стартиране на приложението. Класове с тази анотация най – често имат в себе си методи, които служат за автоматични процеси. Такива са методите с анотация @Scheduled.

* **@Scheduled**

Анотация на ниво метод, която указва изпълнението на дадения метод през определен период от време. Тялото на метода не се изпълнява на основния процес, а на нишки в background режим. Броят нишки за изпълнение на даден процес може да бъде дефинирано от разработчик. Важно е да се осигури конкурентен достъп до ресурсите при многонишкови процеси, тъй като е възможно един елемент да бъде обработен от няколко нишки едновременно.

## 3.4 Hibernate



Hibernate е стартиран през 2001 г. от Гавин Кинг с колеги от Cirrus Technologies като алтернатива на използването на Entity Bean в стил EJB2. Първоначалната цел беше да се предлагат по- добри възможности за постоянство от тези, предлагани от EJB2; чрез опростяване на сложността и допълване на някои липсващи функции.

Hibernate ORM е инструмент за обектно-релационно картографиране за езика за програмиране Java. Той предоставя рамка за съпоставяне на обектно- ориентиран модел на домейн към релационна база данни. Hibernate се справя с проблемите на несъответствието на обектно-релационния импеданс, като заменя директния, постоянен достъп до база данни с функции за обработка на обекти от високо ниво.

Основната функция на Hibernate е картографиране от Java класове към таблици на база данни и картографиране от типове данни на Java към типове данни на SQL. Hibernate също така предоставя средства за запитване и извличане на данни. Той генерира SQL повиквания и освобождава разработчика от ръчната обработка и преобразуването на обекти на набора от резултати.

Съпоставянето на Java класове в таблици на база данни се осъществява чрез конфигурация на XML файл или чрез използване на Java анотации. Когато използва XML файл, Hibernate може да генерира скелетен изходен код за класовете на постоянство. Това е спомагателно, когато се използват анотации. Hibernate може да използва XML файла или анотациите на Java, за да поддържа схемата на базата данни.

## 3.5 React



*Фиг. 4 –React лого*

React е JavaScript библиотека за построяване на потребителски интерфейси. Хубаво е от самото начало да се каже, че React не е Framework, а е библиотека. Това означава, че използването на React и неговото конфигуриране се определя изцяло от самият разработчик и няма ограничения в при създаването на потребителския интерфейс. При използване на frameworks, като Angular например, разработчикът е длъжен да спазва архитектурата и ограниченията на framework-а. React предлага възможността да се използва доколкото е нужен. Плюсове на React:

* Декларативен

React прави лесно създаването на интерактивни потребителски интерфейси. Създавайте прости страници за всяка състояние във вашето приложение. React ефективно ще актуализира и ще визуализира правилните компоненти, когато данните ви се променят.

Декларативните интерфейси правят кода по-предвидим и по-лесен за поддръжка

* Компонентно ориентиран

Изгражда добре капсулирани компоненти, които управляват собствено вътрешно състояние. Дава възможност за комбиниране на тези компоненти и създаването на сложни потребителски интерфейси.

Тъй като логиката на компонента е написана в JavaScript вместо в шаблони, лесно можем да предавате данни през приложението си и да се дистанцираме от DOM манипулации.

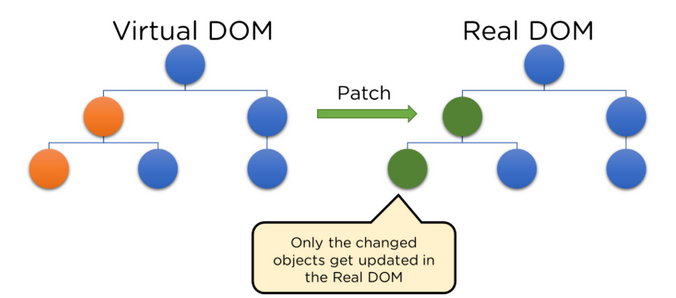
React е създаден от Facebook и open-source общност от разработчици. Въпреки че React е просто библиотека, той често се използва като основен градивен елемент при създаването на Уеб приложения. Появява се за първи път през 2013 г. и днес е една от най – широко използваните библиотеки за front-end разработване.

Освен за създаване на потребителски интерфейс, React предлага и възможност за поддръжка на архитектура за изцяло самостоятелно приложение посредством Flex или React Native.

React е особено популярен в днешно време защото има следните предимства:

* Позволява лесното създаване на динамични приложения. За разлика от JavaScript, React предлага повече функционалности с по – малко код.
* По – голяма производителност, тъй като React използва Virtual DOM.

Virtual DOM е система, която сравнява старото и новото състояние на компонентите и променя само тези, които наистина са се променили. Това помага да се избегне рефрешването на всички компоненти, ако само един е бил променен.



*Фиг. 5 – Virtual DOM начин на работа*

* Възможност за преизползване на компоненти. Компонентите са градивните частици на потребителския интерфейс. Обикновено едно React приложение се състои от множество компоненти, като всеки един от тях има свои собствени контроли, логика за извикване на заявки и други. Веднъж създадени, тези компоненти могат да се преизползват в обхвата на приложението, което намаля до голяма степен времето за разработка.
* Има голяма общност от разработчици и решения на основни проблеми се намират много лесно в Интернет. Освен това са създадени инструменти за дебъгване в самият браузър ( напр. Chrome ), което улеснява разработката и намирането на бъгове.

React разполага с т.нар настройки ( props, от англ. „properties” ), които позволяват да се предават данни между различните компоненти. Това спомага на компонентите да бъдат по динамични и да визуализират определен набор от данни спрямо входните настройки ( props ).

Друга интересна структура в архитектурата на React е състоянието ( state ). Състояние се задава на променливи, които може да бъдат променени по време на използване на приложението. Състояние може да бъде променено от потребител или от мрежова промяна. При промяна на състоянието на дадена променлива, React автоматично презарежда компонента, в чиито обхват се намира променливата.

Добре е да се прави разлика между настройките ( props ) и състоянието ( state ). Настройките не могат да бъдат променяни и се подават от родителски компонент към вложен компонент. Състоянието е декларирано в даден компонент и може да бъде променяно, което води до презареждане на компонента. Освен данни, към настройките може да бъдат подавани и функции. Тези функции най – често се използват да прихванат действие от вложения компонент и да актуализират нещо в родителския компонент.

useEffect е друга удобна функционалност, която се извиква при всяко обновяване на компонента. Може да съдържа в себе си различна логика спрямо нуждите на приложението. Най – често се използва за извличане на ресурси от сървъра за визуализация при направена промяна. useEffect е удобен и за валидиране на достъпа до определен компонент. Ако потребител няма оторизиран достъп до даден компонент може да се разпише логика за прехвърляне към друга страница, съдържаща друг компонент.

## 3.6 PayPal JavaScript SDK



PayPal SDK е полезен инструмент, който позволява на разработчиците да интегрират PayPal система за плащане в техните приложения. Чрез този инструмент може да се добавят бутони за плащане, да се правят валидации на възможните методи за плащане и за статуса на плащането.

Конфигурацията е лесна, а използването предоставя стандартен User Experience за плащане с PayPal. За целите на проекта е използван безплатен тестови профил, който изпраща API заявка към PayPal за оторизиране на плащането и винаги връща успех.

## 3.7 Google reCAPTCHA



reCAPTCHA е безплатен сервиз от Google, който защитава Уеб сайтовете от спам атаки. CAPTCHA е тест за определяне, дали потребителския интерфейс се използва от човек, а не от робот. Тестът представлява избиране на снимки, които са лесни за нормален човек, но трудни за робот. Обикновено се слага навсякъде, където може да има възможност неоторизиран потребител да въвежда данни и да изпраща заявки към сървъра, например логин страницата. Чрез валидация, която се случва на клиентската част, се ограничават излишни заявки към сървъра. Това прави спам атаките невъзможни. За по – лесно интегриране с React има създаден компонент Reaptcha, като това позволява по – лесното му менажиране. Налични са функционалности за ресетване на reCAPTCHA, управляване на нейното изтичане и други.

## 3.8 JavaMail library

Библиотека за Spring Boot приложения, която осигурява възможност за изпращане на мейли. Изисква да се създаде инициализираща настройка, която включва настройки за SMTP( Simple Mail Transfer Protocol ), име и парола за оторизиране в избрания хост сървър за изпращане на имейли. В моето приложение се използва Gmail за хост сървър. Библиотеката предлага лесно изпращане на имейли, подаване на получател, тема и съобщение. Също така поддържа прикачване на файлове и визуализация на съобщението в два възможни формата: обикновен текст или html вариант.

## 3.9 Abstract API

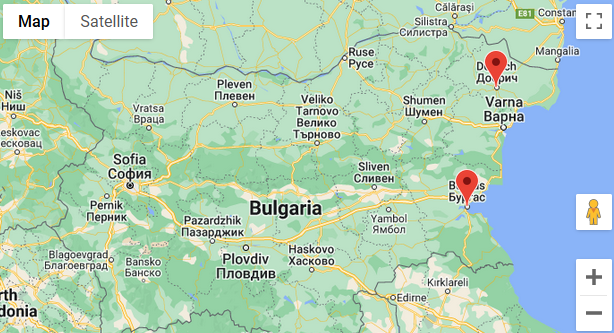


Abstract предлага множество API-та за постигане на различни цели, подобряване на UX-а и автоматизиране на работния процес. Предоставя следните функционалности:

* Следене на локация
* Празнични дни за различните геополитически зони
* Извличане на валутни курсове
* Валидация на имейли
* Валидация на телефони
* Валидация на IBAN

За целите на приложението, Abstract API се използва за валидиране на имейли. Чрез изпращане на заявка с данни за имейл, API-то връща дали този имейл съществува и дали може да се изпращат мейли към него.

## 3.10 Google Maps API



*Фиг. 6 – Визуализация на Google Maps API*

Google Maps API осигурява възможност за визуализация на карта в реално време. Всеки потребител може да използва картата както пожелае, има опции за смяна на вид карта ( карта, сателит, терен ) увеличаване и намаляване на размера. Google Maps API предоставя множество функционалности, но за целите на проекта се използва за закачане на т.нар. маркери, които служат за акцентиране върху локацията на даден обект.

## 3.11 Twilio



Twilio е американска компания, базирана в Сан Франциско, която предоставя програмируеми комуникационни инструменти за извършване и получаване на телефонни обаждания, изпращане и получаване на текстови съобщения и извършване на други комуникационни функции, използвайки своите API за Уеб услуги.

Услугите на Twilio са платени, но има безплатен пробен перидо, който позволява изпращането на телефонни съобщения до телефонни номера по целия свят. Нужно е да се създаде Twilio профил и да се генерира безплатен Twilio телефонен номер, който да инициира текстовите съобщения.

Twilio е конкретно специализиран в комуникацията, като предоставя възможности за изпращане на гласови съобщения, провеждане на разговори с програмируем робот, изпращане на имейли и много други.

# 4. Проектиране на системата

Проектирането е важен етап от създаването на всеки продукт. По време на проектирането трябва да се определят крайните цели на продукта. Трябва да се използват

# 5. Речник

* API (Application Programming Interface) - приложен програмен интерфейс, представляващ набор от правила, която позволява на програмите да разговарят помежду си.
* REST API (Representational state transfer API) - REST е набор от архитектурни правила, а не протокол или стандарт. Крайната точка на REST API е уникален URL адрес, който представлява обект или група обекти от данни.
* FE (Front-End) – Уеб интерфейса на приложението, който е видим за потребителя.
* BE (Back-End) – Сървърната страна на приложението, която предоставя данни при поискване от FE или от някой API инструмент.
* JWT (Json Web Token) - е отворен стандарт който дефинира компактен и самостоятелен начин за сигурно предаване на информация между страните като JSON обект. Тази информация може да бъде проверена и надеждна, защото е с цифров подпис. JWT могат да бъдат подписани с помощта на тайна (с алгоритъма HMAC) или двойка публичен / частен ключ, използвайки RSA или ECDSA. JWT се състой от три части разделени със точка. Тези части са Header, Payload и Signature.
* SPA (Single Page Application) – Уеб приложение на една страница. То динамично динамично пренаписва текущата уеб страница с нови данни от уеб сървъра.
* MPA (Multi Page Application) – Уеб приложение на много страници. То презарежда цялата страница и показват новата, когато потребителят взаимодейства с уеб приложението.
* XSS ( Cross-Site Scripting) – вид инжекция, при която злонамерени скриптове се инжектират в иначе доброкачествени и надеждни уебсайтове.
* XSSM ( Cross-Site Scripting Middleware) – механизъм за проверка и филтриране на HTTP заявки, съдържащи XSS.

# 6. Източници на информация

1. <https://www.therefugeecenter-cu.org/> *- официална страница на The Refugee Center*
2. [*https://www.postgresql.org/docs/*](https://www.postgresql.org/docs/) *- официална документация на PostgreSQL*
3. [*https://en.wikipedia.org/wiki/Java\_(programming\_language)*](https://en.wikipedia.org/wiki/Java_(programming_language)) *– Уикипедия за Java*
4. [*https://www.ibm.com/cloud/learn/java-spring-boot*](https://www.ibm.com/cloud/learn/java-spring-boot) *- официална страница на IBM с информация за Spring Boot*
5. [*https://bg.reactjs.org/*](https://bg.reactjs.org/) *- страница с информация за React*
6. [*https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs*](https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs) *- статия, Какво е React?*
7. [*https://developer.paypal.com/sdk/js/reference/*](https://developer.paypal.com/sdk/js/reference/) *- официална страница на PayPal SDK*
8. [*https://www.google.com/recaptcha/about/*](https://www.google.com/recaptcha/about/) *- официална страница на Google reCAPTCHA*
9. [*https://www.developer.com/design/working-with-javamail-and-the-spring-mail-apis/*](https://www.developer.com/design/working-with-javamail-and-the-spring-mail-apis/) *- статия за работа с JavaMail библиотека*
10. [*https://www.abstractapi.com/*](https://www.abstractapi.com/) *- официална страница на Abstract API*
11. [*https://developers.google.com/maps*](https://developers.google.com/maps) *- официална страница на Google Maps*
12. [*https://www.twilio.com/*](https://www.twilio.com/) *- официална страница на Twilio*