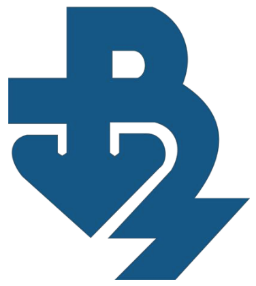
ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА И АВТОМАТИЗАЦИЯ

****

Катедра „Софтуерни и интернет технологии“

**ДИПЛОМНА РАБОТА**

**Тема:**

Проектиране и разработка на система за обслужване 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 4](#_Toc106469852)

[2. Анализ на проблема 6](#_Toc106469853)

[2. 1. Анализ на проблема 6](#_Toc106469854)

[2. 2. Анализ на конкуренцията 6](#_Toc106469855)

[3. Избор на технологии 9](#_Toc106469856)

[3.1 PostgreSQL 10](#_Toc106469857)

[3.2 Java 13](#_Toc106469858)

[3.3 Spring Boot 15](#_Toc106469859)

[3.4 Hibernate 18](#_Toc106469860)

[3.5 React 19](#_Toc106469861)

[3.6 PayPal JavaScript SDK 22](#_Toc106469862)

[3.7 Google reCAPTCHA 22](#_Toc106469863)

[3.8 JavaMail library 23](#_Toc106469864)

[3.9 Abstract API 23](#_Toc106469865)

[3.10 Google Maps API 24](#_Toc106469866)

[3.11 Twilio 24](#_Toc106469867)

[4. Проектиране на системата 25](#_Toc106469868)

[4.1 Спецификация на приложението 26](#_Toc106469869)

[4.2 Концептуален модел 32](#_Toc106469870)

[4.3 UML 32](#_Toc106469871)

[4.3.1 Use Case диаграма 33](#_Toc106469872)

[4.3.2 Activity диаграма 34](#_Toc106469873)

[4.3.3 Sequence диаграма 35](#_Toc106469874)

[4.4 Sitemap 36](#_Toc106469875)

[5. Реализиране на системата 37](#_Toc106469876)

[5.1 Структура на База данни 39](#_Toc106469877)

[5.2 Кратко описание на таблиците в базата 41](#_Toc106469878)

[5.3 Структура на Back - End 42](#_Toc106469879)

[5.4 Структура на Front – End 45](#_Toc106469880)

[5.5 Home interface модул 45](#_Toc106469881)

[5.5.1 Задаване на въпрос 47](#_Toc106469882)

[5.5.2 Извършване на парични дарения 48](#_Toc106469883)

[5.5.3 Влизане в системата 49](#_Toc106469884)

[5.6 Администраторски модул 51](#_Toc106469885)

[5.6.1 Регистрация на потребител 51](#_Toc106469886)

[5.6.2 Потвърждаване на регистрации 57](#_Toc106469887)

[5.6.3 Потребителски групи 60](#_Toc106469888)

[5.6.4 Съобщения 61](#_Toc106469889)

[5.6.5 Съоръжения 64](#_Toc106469890)

[5.6.6 Дарения 67](#_Toc106469891)

[5.6.7 Потребителски профил 68](#_Toc106469892)

[5.7 Модул за бежанци 70](#_Toc106469893)

[5.7.1 Заявки 71](#_Toc106469894)

[5.7.2 Преглед на изпратени заявки 73](#_Toc106469895)

[5.7.3 Потребителски профил 74](#_Toc106469896)

[6. Тестове и резултати 75](#_Toc106469897)

[6.1 Тестове 75](#_Toc106469898)

[6.2 Резултати 78](#_Toc106469899)

[7. Проблеми и решения 79](#_Toc106469900)

[7.1 Защита от спам атаки и настройка на reCAPTCHA: 79](#_Toc106469901)

[7.2 Изпращане на мейли през Gmail: 79](#_Toc106469902)

[7.3 Кодиране на данни 80](#_Toc106469903)

[7.4 Достъп до потребителски страници от неоторизиран потребител: 80](#_Toc106469904)

[7.5 Конкурентен достъп до ресурсите: 81](#_Toc106469905)

[8. Заключение 81](#_Toc106469906)

[8.1 Потенциално развитие на системата 81](#_Toc106469907)

[8.2 Заключителни думи 82](#_Toc106469908)

[9. Речник 83](#_Toc106469909)

[10. Източници на информация 84](#_Toc106469910)

[11. Приложение 85](#_Toc106469911)

# 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

## 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.

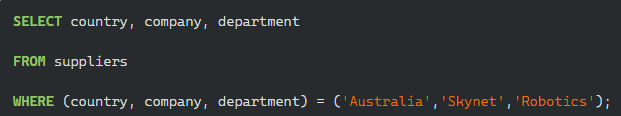
Мнозина казват, че логото на PostgreSQL е слон, защото слоновете имат много голяма памет и на практика казват, че не забравят нищо през живота си. Друга, може би по – достоверна история е, че PostreSQL използва система за репликиране на бази данни, наречена Slony-I. Името Slony идва от руски ( слоны),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

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. 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:

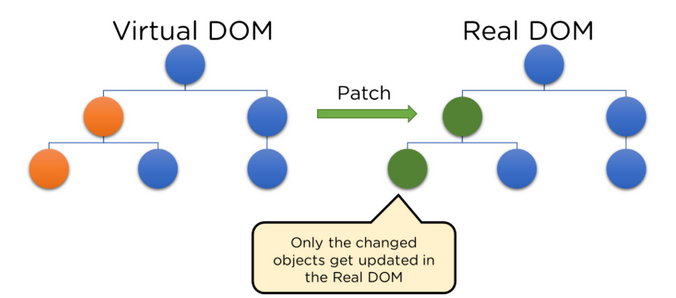
Изгражда добре капсулирани компоненти, които управляват собствено вътрешно състояние. Дава възможност за комбиниране на тези компоненти и създаването на сложни потребителски интерфейси. Тъй като логиката на компонента е написана в JavaScript вместо в шаблони, лесно можем да предавате данни през приложението 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

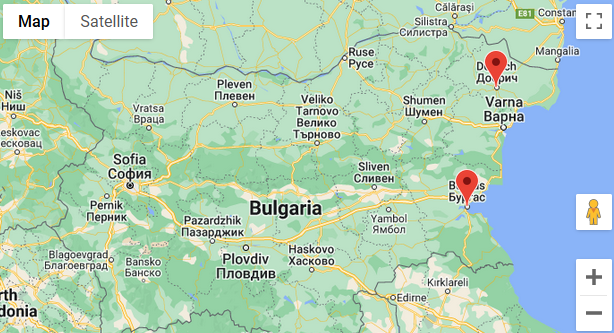


Abstract предлага множество API-та за постигане на различни цели, подобряване на UX-а и автоматизиране на работния процес. Предоставя следните функционалности:

* 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. Проектирането трябва да определи изискванията към приложението, крайните срокове за изпълнение на задачите и времето, за което ще бъде изпълнена всяка една задача поотделно.

## 4.1 Спецификация на приложението

Приложение “Safe Shelter”, обект на дипломната работа, трябва да отговаря на следните изисквания:

* Удобен потребителски интерфейс;
* Лесно взаимодействие със системата;
* Интегриране на разплащателна система;
* Визуализация на физически центрове и контакти;
* Управление на данните в системата;
* Автоматизиране на процеси;
* Изпращане на масови съобщения;
* Прихващане на грешки и записване на ситуацията;

Уеб приложението “Safe Shelter” има няколко обособени потребителски интерфейса спрямо правата за достъп на всеки потребител. Системата поддържа три роли: Администратор, Модератор и Бежанец. Това води до разделение на приложението до три основни потребителски интерфейса:

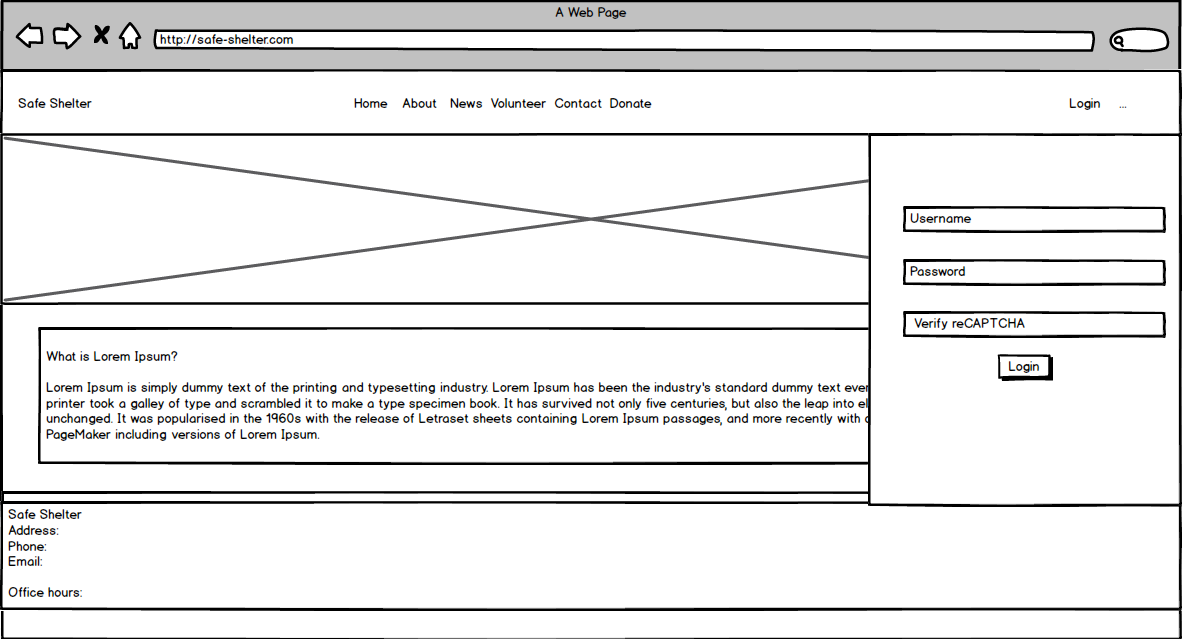
* **Модул за администратор** – основните администраторски функционалности са:
  + **Регистрация на потребител** ( модератор или бежанец ), като регистрацията за модератор изисква минимални данни. Регистрацията на бежанец изисква допълнителна лична информация;
  + **Потвърждаване на регистрации на бежанци.** Ако бежанецът е регистриран от модератор, регистрацията му изисква потвърждение от администратор;
  + **Менажиране на потребителски групи.** Създаване и редакция на потребителски групи от бежанци, служители или общи групи. Възможност за добавяне на потребители в различните потребителски групи;
  + **Изпращане на съобщения** до потребители или потребителски групи. Преглед на съобщения;
  + **Менажиране на бежански лагери.** Създаване и редакция на бежански лагери, добавяне на потребители в лагер, управление на усвояването на даренията за всеки лагер;
  + **Регистриране на дарения.** Добавяне на нови видове дарения. Преглед и редакция на постъпили дарения;
  + **Преглед на профила и промяна на парола**;
* **Модул за модератор** – този модул разполага с функционалностите на администраторския модул, като има ограничения. Модератор не може да регистрира друг модератор в системата. Регистрация на бежанец трябва да се потвърди от администратор;
* **Модул за бежанец** – основните функционалности са:
  + **Заявка за материални запаси.** Потребител може да опише от какво се нуждае и ще бъде изпратена заявка за преглед от служител;
  + **Заявка за промяна на местоположение**. Потребител може да избере нов бежански център за престой;
  + **Заявка за медицинска помощ.** Заявките за медицинска помощ изискват описание на ситуацията. При такава заявка се изпраща текстово съобщение към управителя на бежанския център, че има проблем;
  + **Преглед на съобщения.** Преглед и изпращане на съобщения до потребители/ потребителски групи;
  + **Преглед на изпратени заявки.** Преглед и следене статуса на всяка изпратена заявка от текущия потребител;
  + **Преглед на профила.** Смяна на парола и редакция на лични данни;

Наличен е допълнителен модул с интерфейс за потребители, които нямат регистрация или не са се логнали в системата. Основните функционалности са:

* **Home**. Начална страница с информация за организацията;
* **About.** Страница с информация за целине и идеалите на организацията;
* **Новини.**  Актуална информация за случващото се с организацията;
* **Доброволец.** Информация как да станеш доброволец;
* **Контакти.** Информация за контакти и локация на основните бежански центрове;
* **Дарения.** Функционалност за даряване на парични средства за бежанския център;
* **Вход в системата;**
* **Забравена парола;**

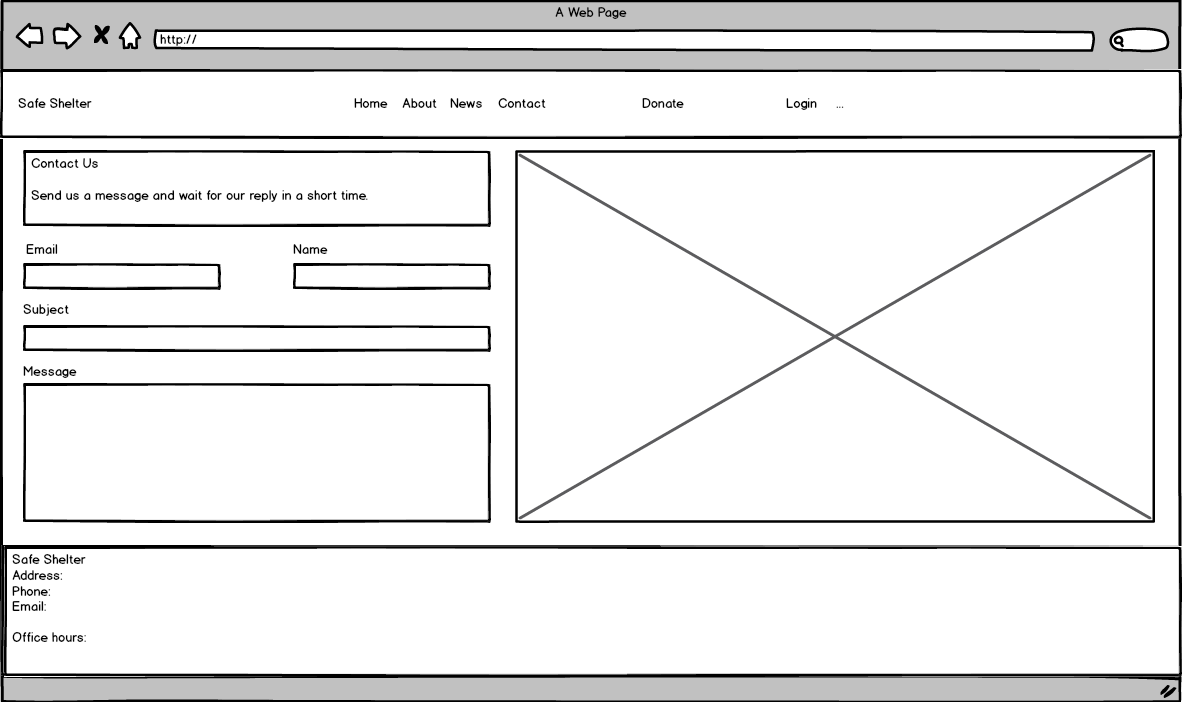
Проектирането на потребителския интерфейс премина през създаване на примерен прототип с основните страници в приложението. Този прототип може да даде идея каква би била визията на интерфейса, дали менютата и тяхното разпределение би било лесно за използване от нормален потребител.

Създаването на подобен прототип става много лесно с приложение Balsamiq Mockups. Приложението предлага създаване на страници и добавяне на контроли посредством drag - and – drop.



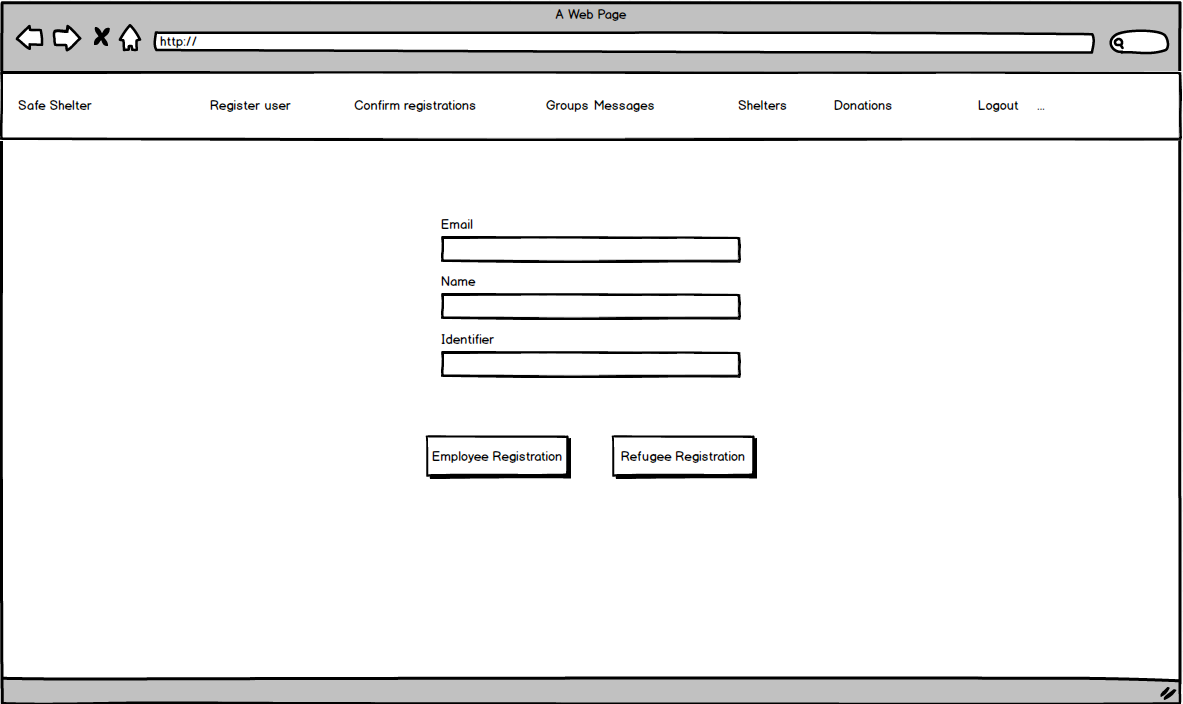
*Фиг. 7.1 – Примерен дизайн на началната страница*

Страницата за контакти трябва да предоставя възможност на потребителите да задават въпроси. Освен това трябва да съдържа информация за бежански центрове и контакти. С напредването на разработката ми дойде идея за интеграция на Google Maps в тази страница, за да може адресите на бежанските центрове да се показват като маркери на картата и ориентацията на обикновения потребител да е много по – лесна.



*Фиг. 7.2 – Примерен дизайн на страницата за контакти*

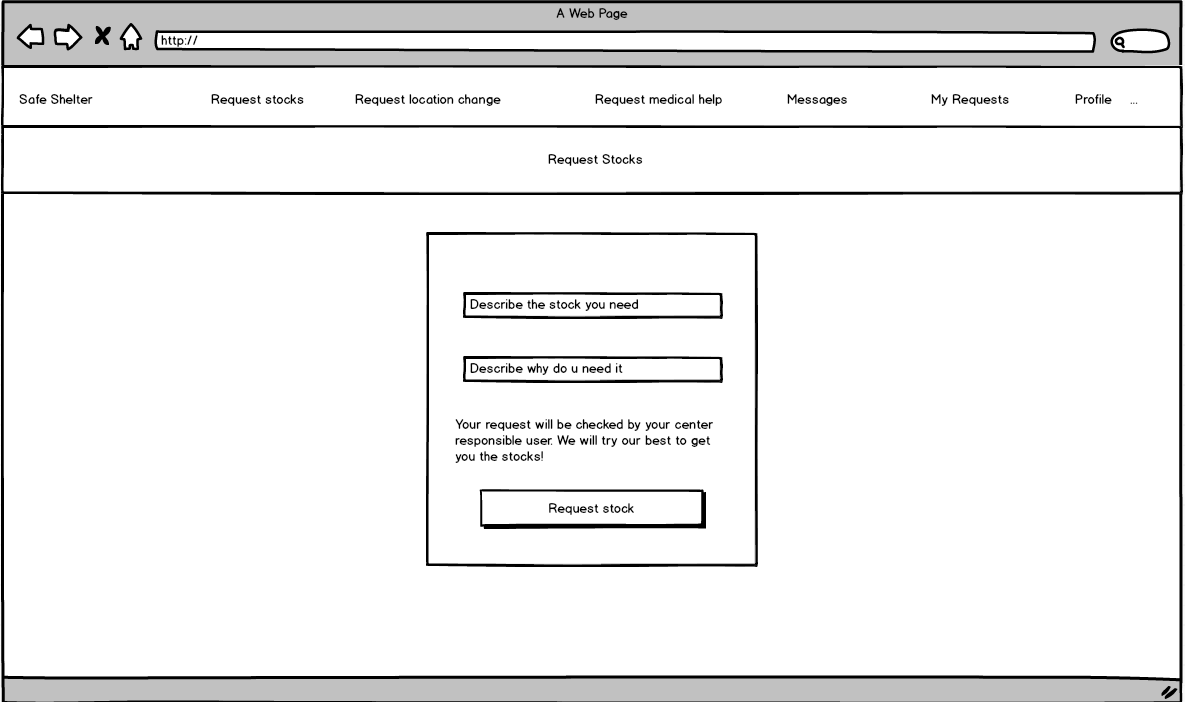
При вход в администраторския модул се визуализира администраторския навигационен панел с избрана опция за регистрация на потребител по подразбиране.



*Фиг. 7.3 – Примерен дизайн на администраторски интерфейс*

Както споменах по – рано модераторският профил е същият като администраторския, като изключим опцията за потвърждаване на бежански регистрации и регистрацията на служители.

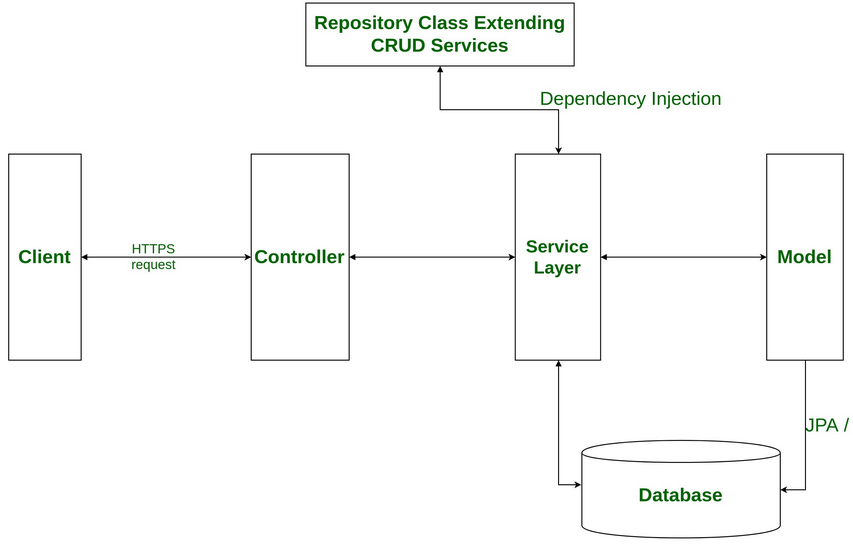
Бежанският интерфейс предлага друг набор от менюта в навигационния панел и функционалностите, които един бежанец може да предприеме са свързани с него. Той не може да манипулира други данни освен своите. По малкото функционалности улесняват взаимодействието с интерфейса, което е плюс за приложението, тъй като основната цел при изграждане на потребителски интерфейс за обикновенни потребители е да се опрости възможно най – много самият продукт.



*Фиг. 7.4 – Примерен дизайн на бежански интерфейс*

## 4.2 Концептуален модел

Концептуалните модели са абстрактна представа как трябва да функционира една система. Те са неизменна част от проектирането на един продукт. На база на концептуален модел по – късно се създават модули и интерфейси, които си взаимодействат за изграждането на функциониращ софтуер.



*Фиг. 8 – Концептуален модел на приложението “Safe Shelter”*

Както се вижда от Фиг. 8, концептуалният модел е реализиран на базата на MVC ( Model – View – Controller ) архитектурата. MVC е архитектурен модел, който разделя приложението на три основни слоя:

* **Model –** даннов слой, който си взаимства с базата данни;
* **View –** клиентски слой, съдържащ в себе си визуализация на данните;
* **Controller –** междинен слой, който управлява извикванията към Model и към View;

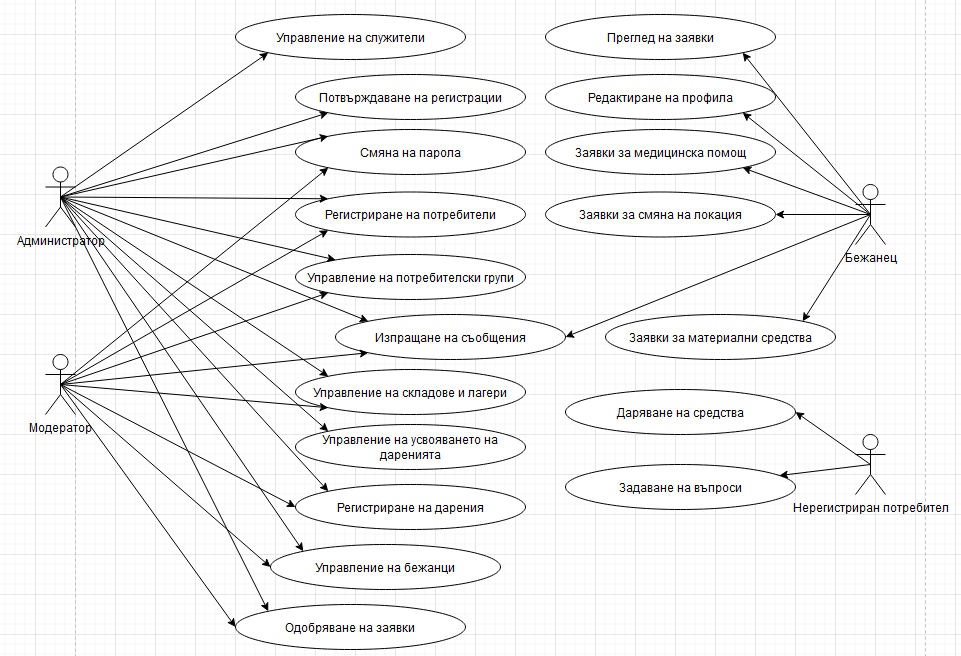
Важното тук е, че не трябва Model и View да се достъпват директно. Всички операции трябва да минават през Controller.

## 4.3 UML

Унифицираният език за моделиране ( UML ) е модерен подход към конвенционалния процес на моделиране и документиране на софтуер. Този подход представлява схематично представяне на компонентите и процесите, участващи в софтуера. Това е като да се обясни целия софтуер чрез визуални представи, така че да е лесно да се разберат и премахнат възможни недостатъци и грешки в системата.

### 4.3.1 Use Case диаграма

Use Case е може би най – известната диаграма от UML. Чрез нея се описват функционалностите на приложението и кои потребителски роли ( актьори ) имат достъп до тях.

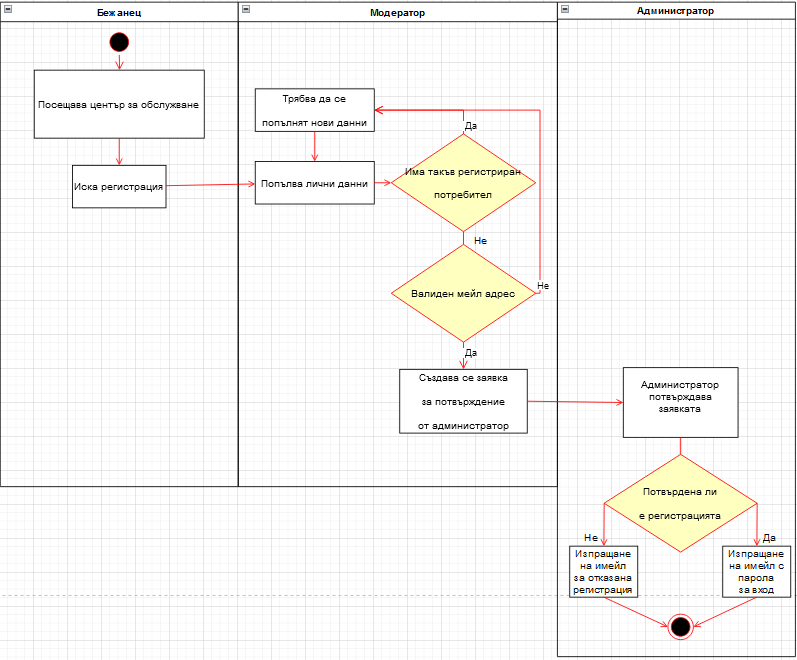


*Фиг. 9 – Use Case диаграма на „Safe shelter”*

### 4.3.2 Activity диаграма

Activity диаграмите са най – важните диаграми при дефинирането на един бизнес процес. При разработването на софтуер, такива диаграми са много полезни, защото описват протичането на процесите в системата. На такива диаграми се представят извършваните действия от различните потребители на системата и какво се случва след определено действие.

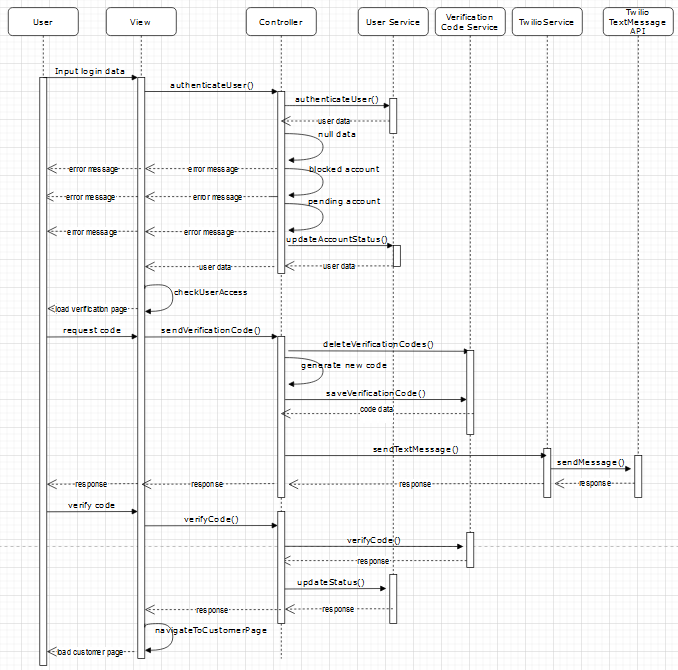
Процесът не се фокусира толкова върху това, което се създава, а по скоро от последователността от действия, които водят до финалното състояние.



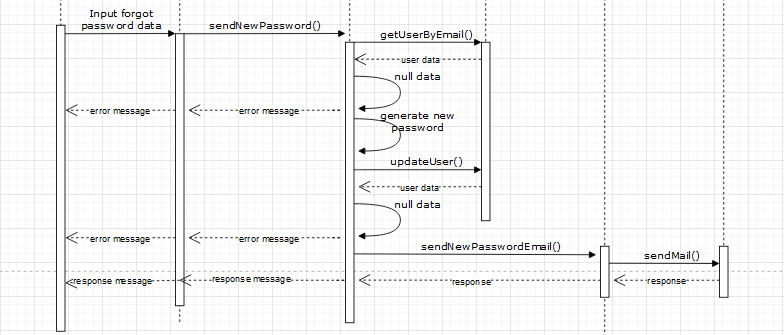
*Фиг. 10 – Activity диаграма за регистрация на бежанец*

### 4.3.3 Sequence диаграма

Sequence диаграмите се използват за по – детайлно проследяване на даден процес. Този вид диаграми проследяват взаимодействието между отделните потребирели или системи. Диаграмата показва кога даден потребител или система е активна и извършва действия. Тук при комуникация между различните обекти се описват и какви функционални извиквания се правят към системата.



*Фиг. 11 – Sequence диаграма за login на бежанец след потвърждение от администратор*



*Фиг. 12 – Sequence диаграма за забравена парола*

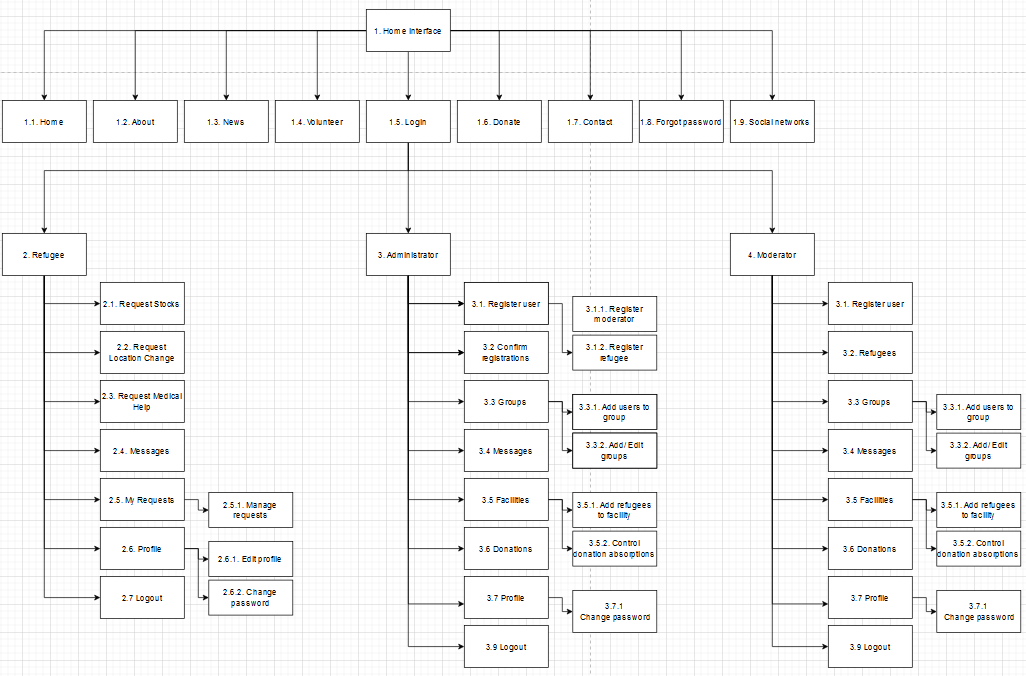
Sequence диаграмите са особено полезни, когато трябва да се определи жизнения цикъл на всеки обект. Както може да се види от Фиг. 11 и Фиг 12. потребителят и потребителският интерфейс остават активни през цялото време, докато контролерът и отделните сервизи се активират само когато са необходими. Един сервиз се активира, когато към него се изпраща съобщение и жизненият му цикъл приключва, когато неговата памет вече не се реферира от друг обект – контейнер. Потребителският интерфейс остава активен през цялото време, защото той инициира заявка и изчаква нейното обработване.

Потребителският интерфейс може да инициира заявки към контролера, като изчаква техния резултат асинхронно, но конкретно за оторизиращи заявки като login това не е добър вариант, защото искаме да ограничим възможността от действия на потребителя, докато не върнем резултат от заявката.

## 4.4 Sitemap

Казано най-общо, това е упътване, което съдържа списък с линкове към всички страници на приложението. Теоретично погледнато, такава страница е създадена с идеята да даде на потребителите бърз и улеснен достъп до всички части на продукта, за да се подобри тяхната ориентация при използването на системата.

Картата на приложението също има за цел да елиминира нуждата от поставяне на линк от началната страница към всички останали страници от сайта. Използва се за групиране на информационно зависими компоненти в софтуерния продукт. Тези карти са една от важните стъпки при процеса на създаване на дизайн, ориентиран към потребителя, като гарантират, че съдържанието, което потребителите желаят да прегледат, ще бъде на най-очакваното за тях място.



*Фиг. 13 - Sitemap на приложение “Safe Shelter”*

# 5. Реализиране на системата

Реализирането на системата може да се раздели на три основни части: разработка на базата данни, разработка на сървърната част на приложението и разработка на потребителския интерфейс. Тези три части са ключови за финализирането на проект и той не може да бъде завършен при липсата на която и да е от тях. Отделните задачи по време на разработка бяха свързани със създаване на цялостна функционалност, напр „Създаване на функционалност за регистрация на потребители“, за което беше необходимо да надграждам базата данни, сървърната част и потребителския интерфейс едновременно.

Това ми помогна до голяма степен за да систематизирам процеса на разработка и да оптимизирам времето за разработка на следващите функционалности. Макар с различни данни и с допълнителна логика, всички функционалности се състоят от няколко общи неща:

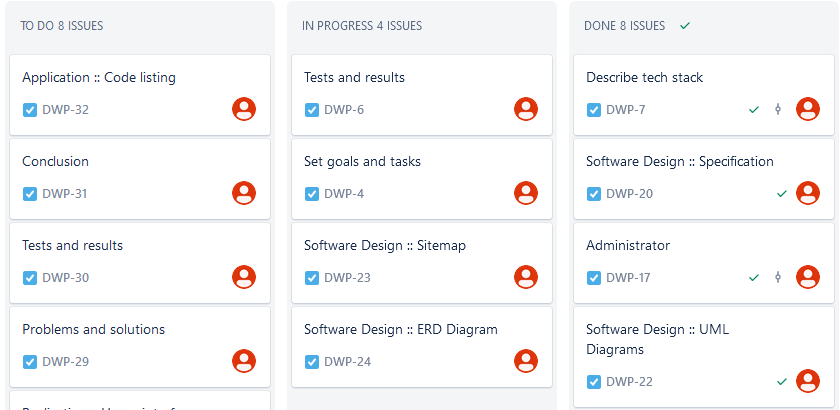
* Изграждане на потребителски интерфейс за въвеждане на данни;
* Контрол на входните данни на ниво клиент;
* Извикване на заявка към сървъра;
* Бизнес валидации на ниво сървър;
* Обработка на данните на сървъра;
* Връщане на отговор към клиента;

След първото завършване на една функционалност, повтарях същите стъпки отново и отново докато проектът не беше напълно завършен. С всяка следваща функционалност ставах все по – добър, идваха ми нови идеи и се стремях да изпълня всичките цели, които си бях поставил.

За да се систематизира един такъв процес и да се следи прогреса на проекта обикновено се използват различни софтуери, които са създадени за тази цел.

С помощта на моя ръководител, успях да създам профил в Jira и да се възползвам от предимствата на този продукт. Чрез Jira поставянето на цели и разпределението на задачите във времето ставаше много лесно. Интерфейсът предлага създаване на проект и прикачване на код към него посредством няколко системи за контрол на версиите ( GitHub, Bitbucket ).

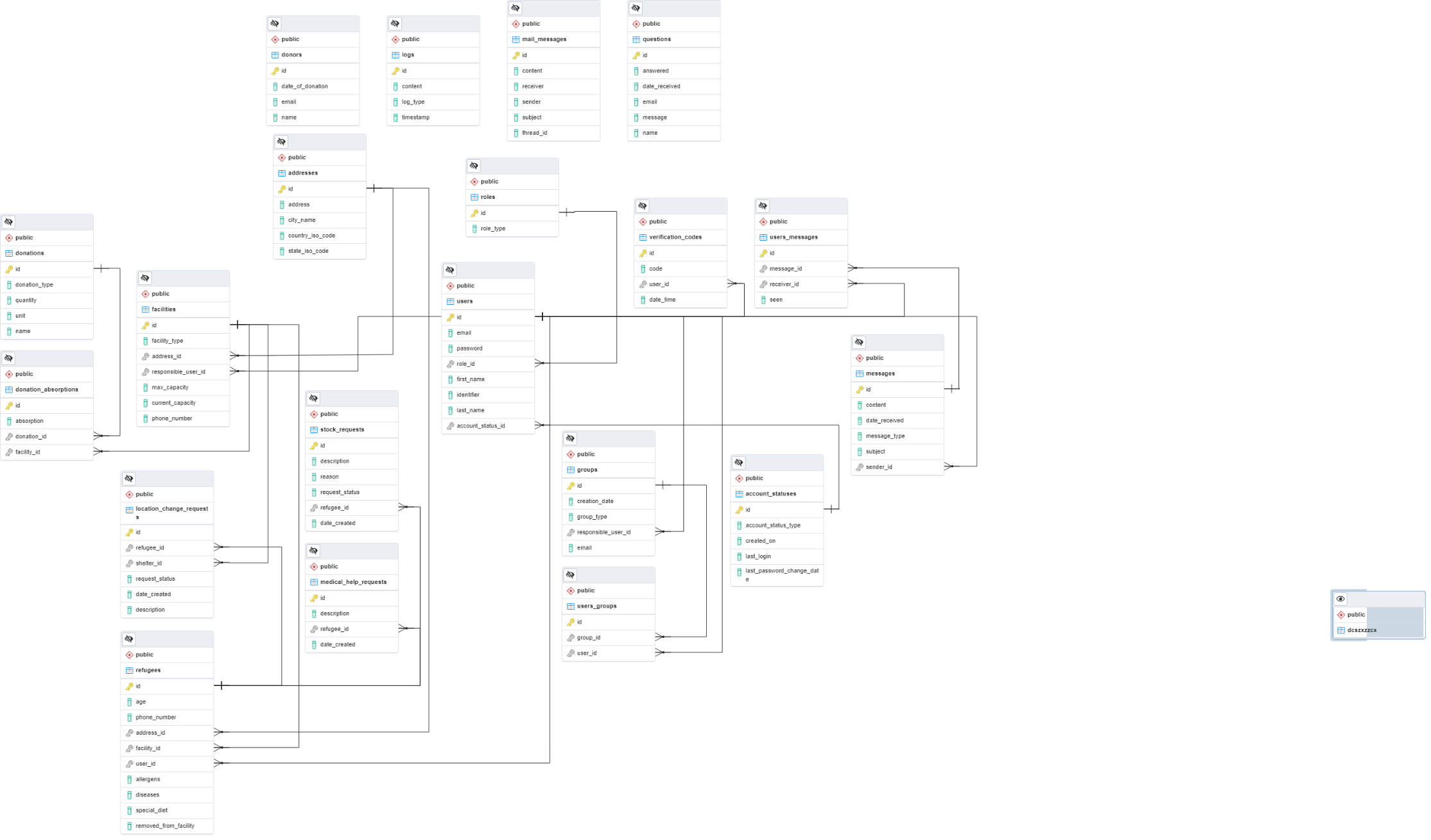
За система за контрол на версиите се спрях на Bitbucket, защото това е системата по подразбиране при работа с Jira, а освен това не бях работил с Bitbucket досега. Bitbucket хостинг система за хранилище на изходен код, базирана на Git. Тази система се използва предимно за проекти, които се разработват от екип. Добрата интеграция с Jira позволява да се създават задачи с описание на изискванията, срок, кой ще прави задачата и кой ще я провери след това. Всяка задача има свой уникален код и посредством този код може да се прикачва код към задачата. Тъй като Bitbucket е система за контрол на версиите, всеки Commit към основното хранилище може да бъде разглеждан, като се сравнява старото и новото състояние на проекта.



*Фиг. 14 - Примерна Jira дъска със задачи*

## 5.1 Структура на База данни

Основна стъпка в разработката на софтуер е изграждането на база данни, която да може да съхрани нужната информация. Добра практика е цялата информация да се раздели на отделни таблици според връзките между различните обекти, за да може да се предотврати повтарянето на една и съща информация на няколко места



*Фиг. 14 - ERD диаграма на приложение “Safe Shelter”*

На фиг. 14. е показана ERD диаграма на базата данни, съдържаща всички таблици с данни, които са необходими за функционирането на системата.

Доброто проектиране на базата данни е от голямо значение за по – лесната последваща разработка. Трябва да се предвидят уникални индекси и връзки между таблиците, за да се ограничи възможността от повторяемост на данните в базата.

## 5.2 Кратко описание на таблиците в базата

Именоването на таблиците в базата следва общ стандарт. Имената на таблиците са в множествено число, като думите са разделени с долни черти ( т.нар. snake\_case ).

**ACCOUNT\_STATUSES:** Съхранява данни за статус на потребителски акаунт. Дата на създаване, дата на последно влизане в системата и статус на акаунт, като всеки акаунт може да бъде блокиран, изчакващ одобрение, отхвърлен, одобрен и потвърден.

**ADDRESSES:** Съдържа данни за адрес.

**DONATION\_ABSORPTIONS:** Съдържа информация за усвояване на даренията. Полагаемо количество от дарение, което се усвоява в рамките на 24ч. за определен бежански център.

**DONATIONS:** Данни за дарение, тип дарение, количество.

**DONORS:** Данни за дарители. Допълват се данни за дарители при регистриране на дарение в бежански център или при даряване на парични средства през страницата на приложението.

**FACILITIES:** Данни за складове или бежански центрове.

**GROUPS:** Данни за потребителски групи ( мейл на групата, отговорно лице ).

**LOCATION\_CHANGE\_REQUESTS:** Заявки за промяна на бежански център.

**LOGS:** Таблица с логове от грешки, случили се при работа на системата.

**MAIL\_MESSAGES:** Опашкова таблица със съобщения, които предстои да бъдат изпратени на потребителите по мейл.

**MEDICAL\_HELP\_REQUESTS:** Заявки за медицинска помощ

**MESSAGES:** Таблица със съобщенията в системата.

**QUESTIONS:** Въпроси от нерегистрирани потребители.

**REFUGEES:** Таблица с основни данни за бежанци.

**ROLES:** Данни за поддържани роли в системата.

**STOCK\_REQUESTS:** Заявки за материални запаси.

**USERS:** Потребителски данни за всеки потребител на системата.

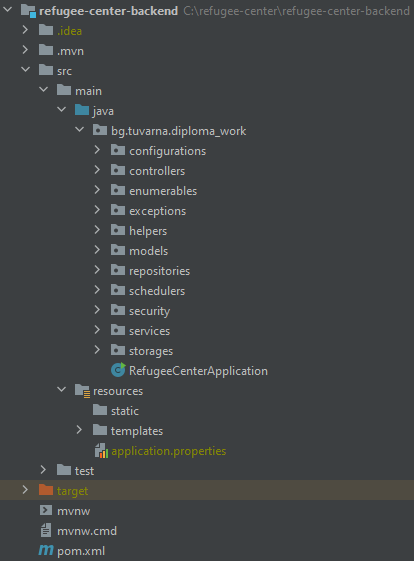
**USERS\_GROUPS:** Таблица за реализиране на връзка Many To Many. Един потребител може да е в няколко групи и една група може да има много потребители.

**USERS\_MESSAGES:** Таблица за реализиране на връзка Many To Many. Едно съобщение може да има много получатели. Един потребител може да има много съобщения.

**VERIFICATION\_CODES:** Данни за код за потвърждение. Съдържа в себе си криптиран код и период на валидност.

## 5.3 Структура на Back - End

Файловата структура на back- end на “Safe Shelter” излгежда по следния начин:



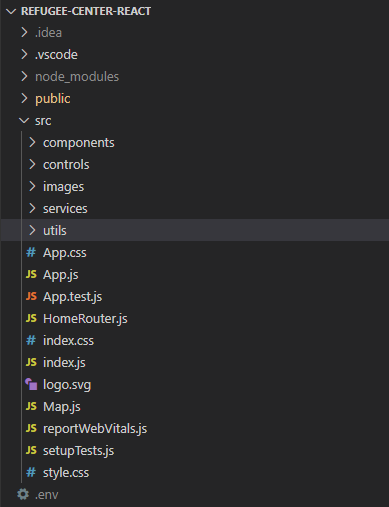
*Фиг. 15 - Файлова структура на back- end*

Основните модули на приложението са разделени в няколко поддиректории, като всеки модул има определена цел и функционалности:

* **Configurations –** съдържа предефинирани настройки за работата на някой от функционалностите. Тук са настроени **JavaMailSender** класа за изпращане на мейли и **SchedulingConfigurer,** за настройка на нишките за обработка на процеси в background режим;
* **Controllers –** всички контролери, които „слушат“ за извиквания към сървъра. Контролерите са разделени спрямо това, какви заявки обработват;
* **Enumerables –** всички класове Enum в системата, които служат за добавяне на тип на други обекти. ( напр. стойностите за всяка роля са Enum );
* **Exceptions –** тук са разположени наследници на стандартните Exception класове при изпращане на заявки. Целта е създаването на response обект за връщане на съобщения за грешка към клиента в случай, че валидация на сървъра не е преминала;
* **Helpers –** помощни класове;
* **Models –** модели на базата данни. Това са класове за съхраняване на данни, които отговарят за таблиците в базата;
* **Repositories –** тук са всички класове, които отговарят за манипулация на данните в базата ( DTO класове );
* **Schedulers –** класове, чиито функционалности се изпълняват автоматично през определен период от време;
* **Security –** съдържа клас за криптиране на данни ( пароли, ключове за верификация );
* **Services –** всички сървърни класове, които се грижат за обработката на данните, спрямо бизнес логиката на приложението;
* **Storages –** транспортни класове с данни;

В директория **resources** има папка **templates,** където са разположени различните темплейтни мейли.

## 5.4 Структура на Front – End



*Фиг. 16 - Файлова структура на back- end*

* **Components –** съдържа основните компоненти на потребителския интерфейс;
* **Controls –** custom контроли и теми на приложението;
* **Images –** снимки за визуализация ;
* **Services –** класове, които изпращат заявки към сървъра;
* **Utils –** помощни класове ;

## 5.5 Home interface модул

Това е потребителският интерфейс, който се визуализира при отваряне на Уеб приложението. Той е изцяло обособен да информира потребителите и да им позволи достъп до потребителските модули.

*Фиг. 17 - Начална страница на приложение „Safe Shelter”*

Навигационният панел има няколко опции, като повечето съдържат информация за платформата. Footer-ът е с данни за контакти, а в дясно има линкове към социалните мрежи на сайта. Горе в дясно има бутон за отваряне на логин панел. Няма регистрация на потребирел, тъй като регистрацията се извършва единствено в центрове за обсужване от служител на бежанския център.

Home interface, макар в по – голямата си част просто информативен, дава възможност за извършване на няколко действия:

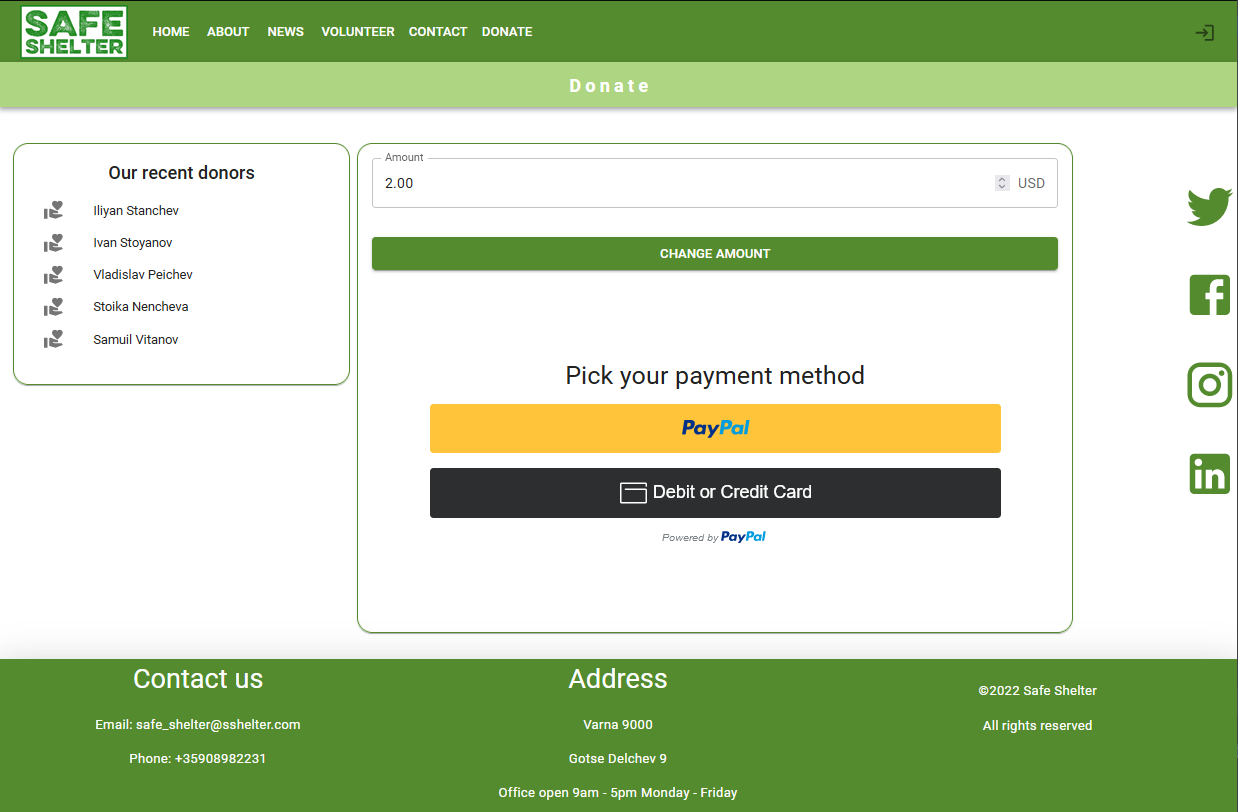
### 5.5.1 Задаване на въпрос

*Фиг. 18. Контакти*

На тази страница се визуализират регистрираните бежански центрове върху Google Maps карта. Потребителят има възможност да изпрати запитване към администраторите на сайта, като предостави имейл и име за обратна връзка.

За да се ограничи възможността от spam атаки, всяка точка за изпращане на заявки към сървъра от неоторизиран потребител има допълнителен контрол. Аз съм избрал reCAPTCHA за тази цел, тъй като е удобно и широко използвано решение на такива проблеми.

### 5.5.2 Извършване на парични дарения

*****Фиг. 19. Дарения*

Страницата позволява извършването на дарения чрез интеграция на платежна система PayPal. При опит за дарения, се отваря popup диалог, осигурен от PayPal за извършване на плащането.

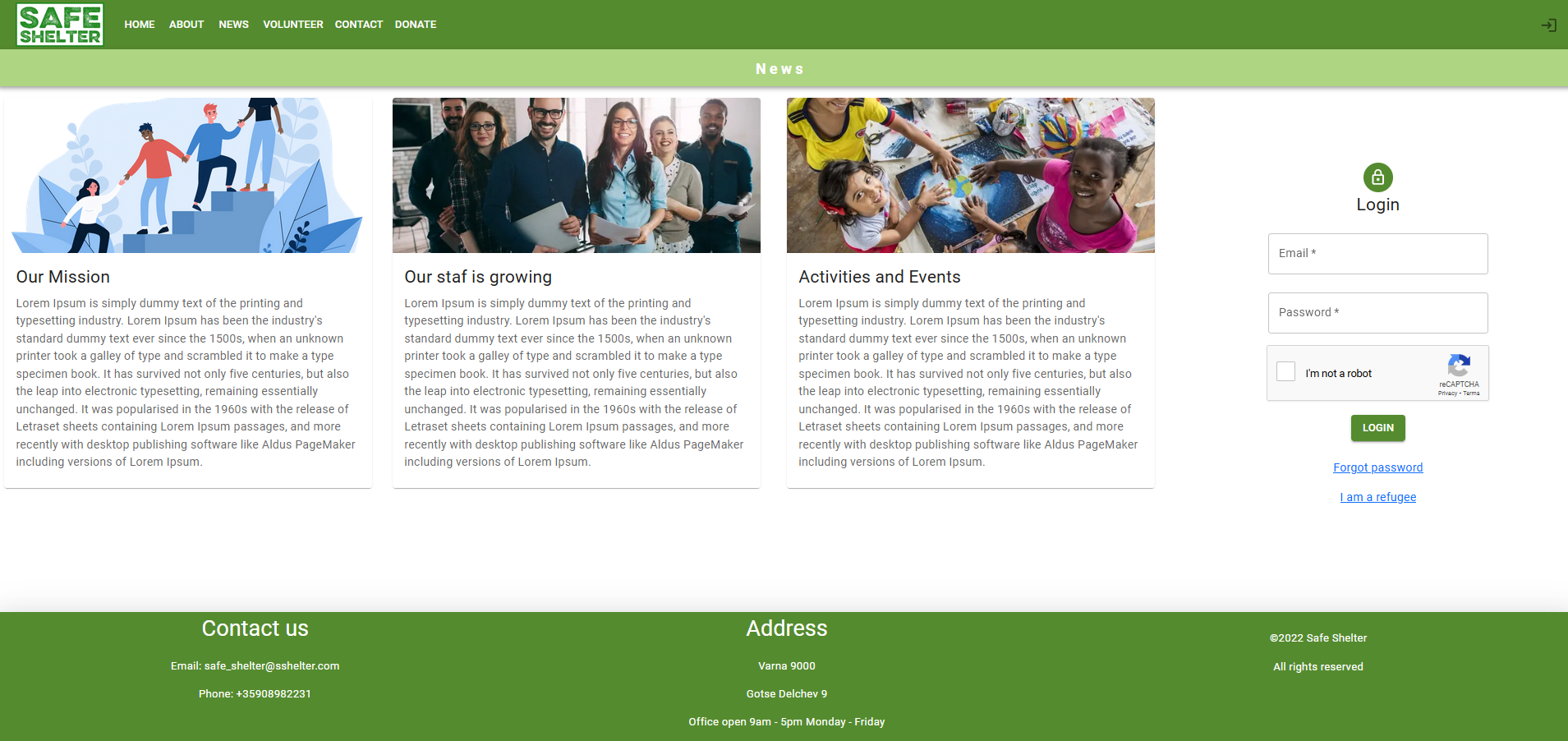
За да се зареди, PayPal изисква добавяне на компонент:



Където, client-id е генерираният идентификатор от PayPal, който всеки може да получи при регистрация в сайта. Оторизирането на плащанията за тестови профил като моя се случва с автоматично потвърждаване, че плащането е преминало успешно.

В ляво се зареждат автоматично имената на последните дарители. Това става чрез GET заявка към сървъра, която се извиква при обновяване на текущата страница.

### 5.5.3 Влизане в системата

*Фиг. 20. Визуализация на login панел*

Влизането в системата може да се покаже от всяка потребителска страница, като измества компонентите вляво. Login панелът съдържа две полета: имейл и парола. Данните в полетата не се валидират. Единствената валидация е да се потвърди reCAPTCHA. При опит за вход се изпраща заявка към сървъра за автентикация на потребителя. Повече детайли как работи Login автентикацията може да се видят на Фиг. 11.

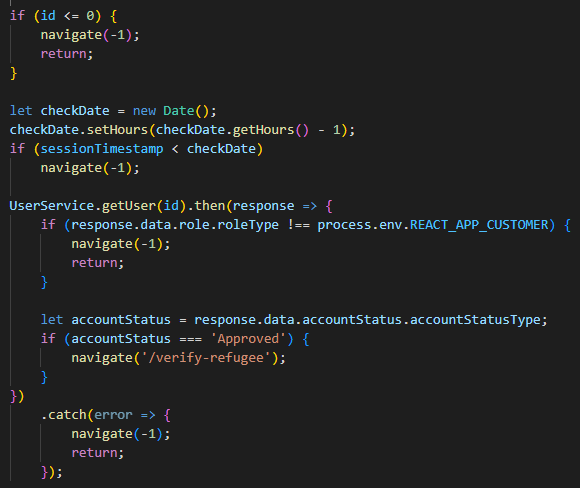
При успешен вход се създава потребителска сесия, която се съхранява в session storage на браузъра. Предимствата на session storage са, един съхранението на данните се пази на ниво отворен таб, а не за целият браузър, което означава, че може да се отворят няколко инстанции на приложението на един и същ браузър. Плюс това при обновяване на страницата, информацията не се зачиства.

Идеята на потребителската сесия е да не се допускат неоторизирани лица до компоненти, които изискват първоначално логване в системата. В потребителската сесия се пази уникалния идентификатор на потребителя и времето на създаване на потребителската сесия.

За по – голямо улеснение, съм използвал вграден компонент ReactSession, който ми осигурява достъп до session storage на браузъра по лесен начин.



При всеки опит за достъп до компонент на логнат потребител се проверява съдържанието на клиентската сесия:



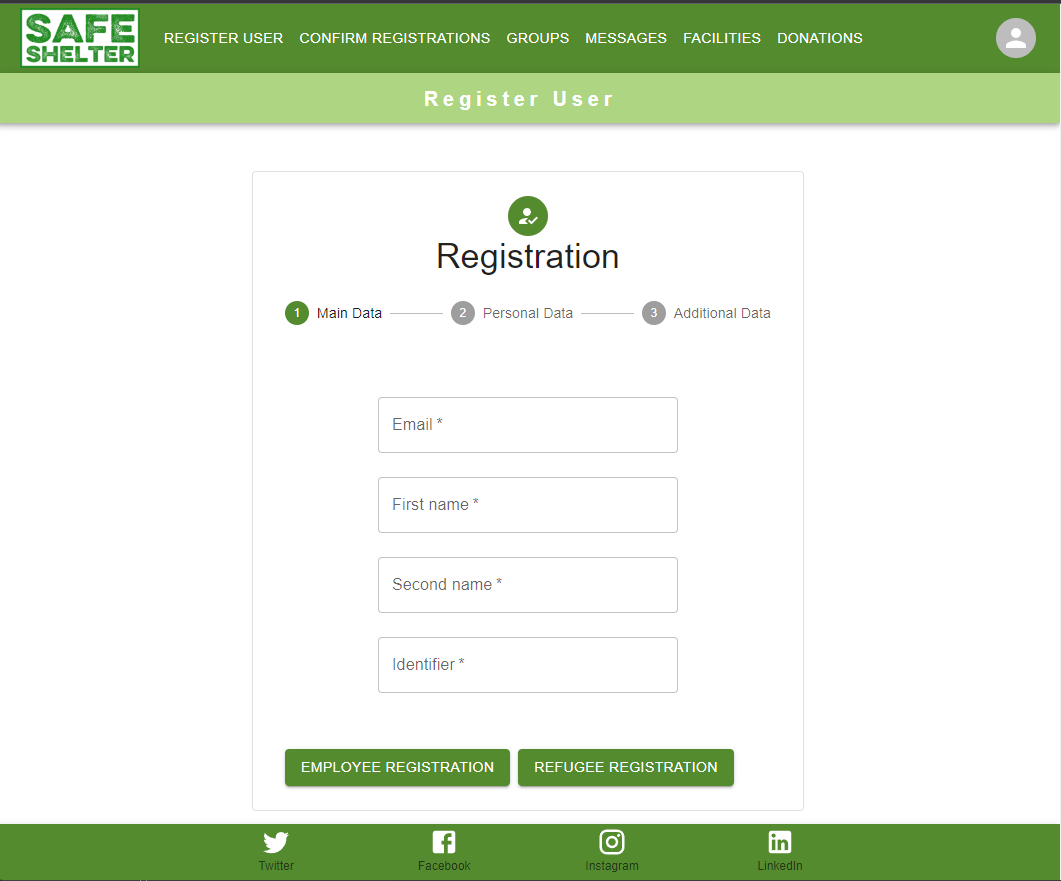
Извършват се няколко валидации на съдържанието на клиентската сесия:

* Проверка за валиден клиентски идентификатор;
* Проверка дали клиентската сесия не е изтекла. Всяка клиентска сесия губи валидност след един час и се налага ново влизане на потребител;
* Дали текущата роля на потребителя има достъп до текущия ресурс;

## 5.6 Администраторски модул

Администраторският модул съдържа най – много функционалности, като действията на администратора няма нужда да бъдат потвърждавани от друг потребител. След логин се визуализира следния потребителски интерфейс:

### 5.6.1 Регистрация на потребител

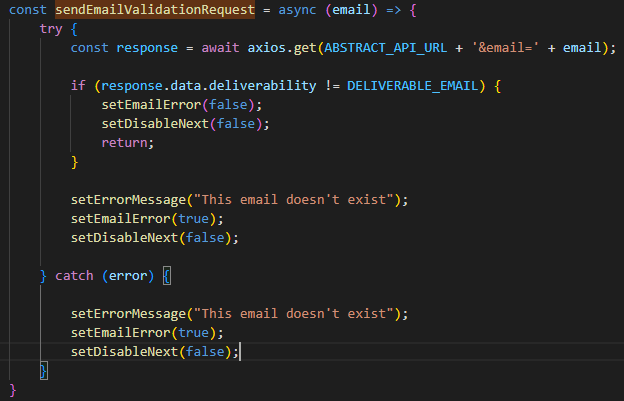
*Фиг.21. Първа стъпка при регистрация на потребител*

По дефолт е отворена регистрация на потребител. Самата регистрация е реализирана като постъпков Wizard, като всяка следваща стъпка е достъпна след валидация на предходната. Позволено е връщане назад, като входните данни се запазват и могат да се редактират, но за продължаване напред отново се извършва валидация.

На първата стъпка от Wizard-а за регистрация има две опции ( регистрация на служител и регистрация на бежанец ). Регистрацията на бежанец изисква въвеждане на допълнителни данни, които са на следващите стъпки.

В първа стъпка се случват следните валидации:

* Контрол на формата на данните на ниво клиент
* Изпращане на заявка към Abstract API за проверка на валиден имейл адрес



Чрез GET заявка към Abstract API се взимат данни за имейла, като за валидност се проверява дали имейл адресът може да получава съобщения. Това означава, че съществува.

* Валидация на входните данни на сървъра.

На сървъра се правят валидации за уникални полета ( мейл и телефон ). При възникнала грешка се визуализира съобщение на клиента с информация за проблема.

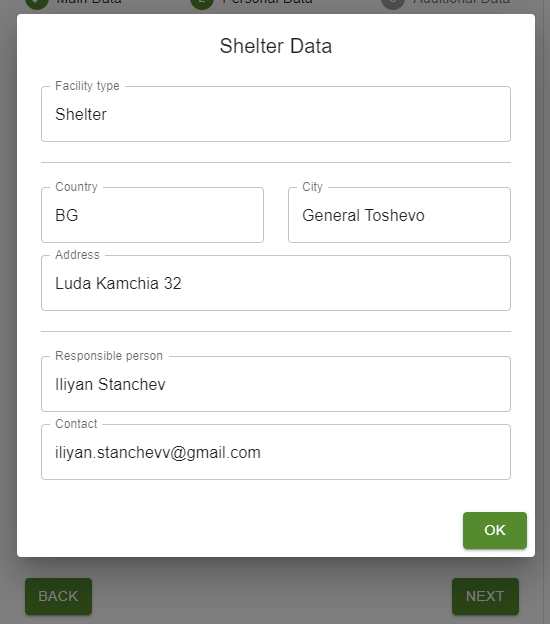
Ако всички валидации са минали успешно бутонът за регистрация на служител добавя нов валидиран запис в базата данни.

При регистрация на бежанец, Wizard - ът продължава към следваща стъпка:



*Фиг.22. Втора стъпка при регистрация на потребител*

Втората стъпка изисква въвеждане на бежански център за престой. Бежанските центрове се зареждат от сървъра. Има възможност да се визуализират допълнителни данни за бежанския център, чрез натискане на popup бутон, разположен в комбо бокса



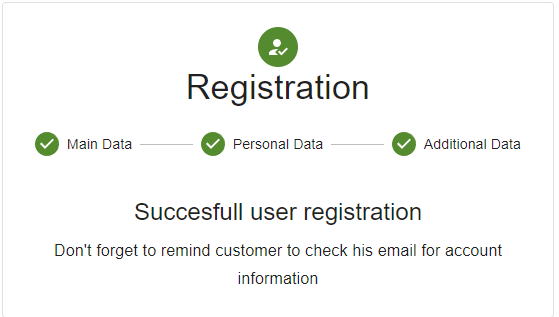
*Фиг.23. Визуализация на данни за бежански център от контекста на регистрация*



*Фиг.24. Трета стъпка при регистрация на потребител*

На последната стъпка се попълват допълнителни данни, ако има такива. Те не се валидират, защото не са задължителни.

При успешна регистрация се появява следния екран:

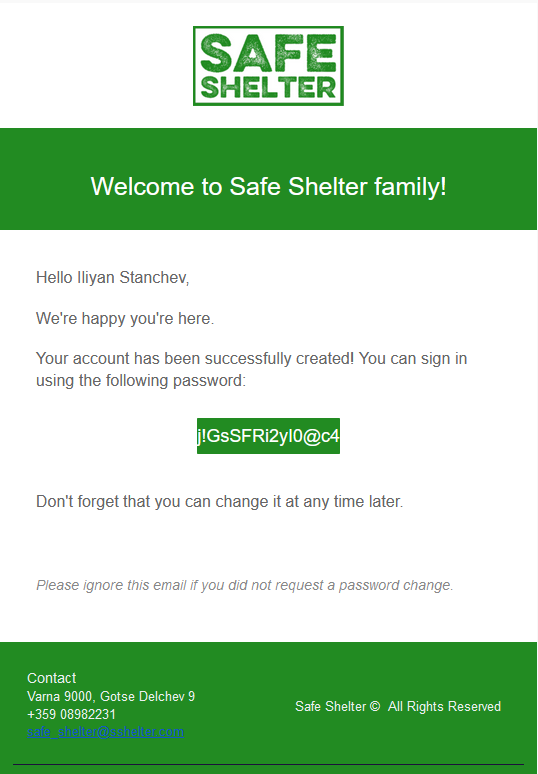


*Фиг.25. Финализиране на регистрация*

При регистрация на потребител се генерира случайна потребителска парола, която се изпраща по мейл. Всеки потребител може да смени паролата си в по – късен етап. За изпращането на мейли използвах JavaMailSender. За автентикация са нужни потребителски профил в Gmail, както и генерирана парола за влизане през приложение.



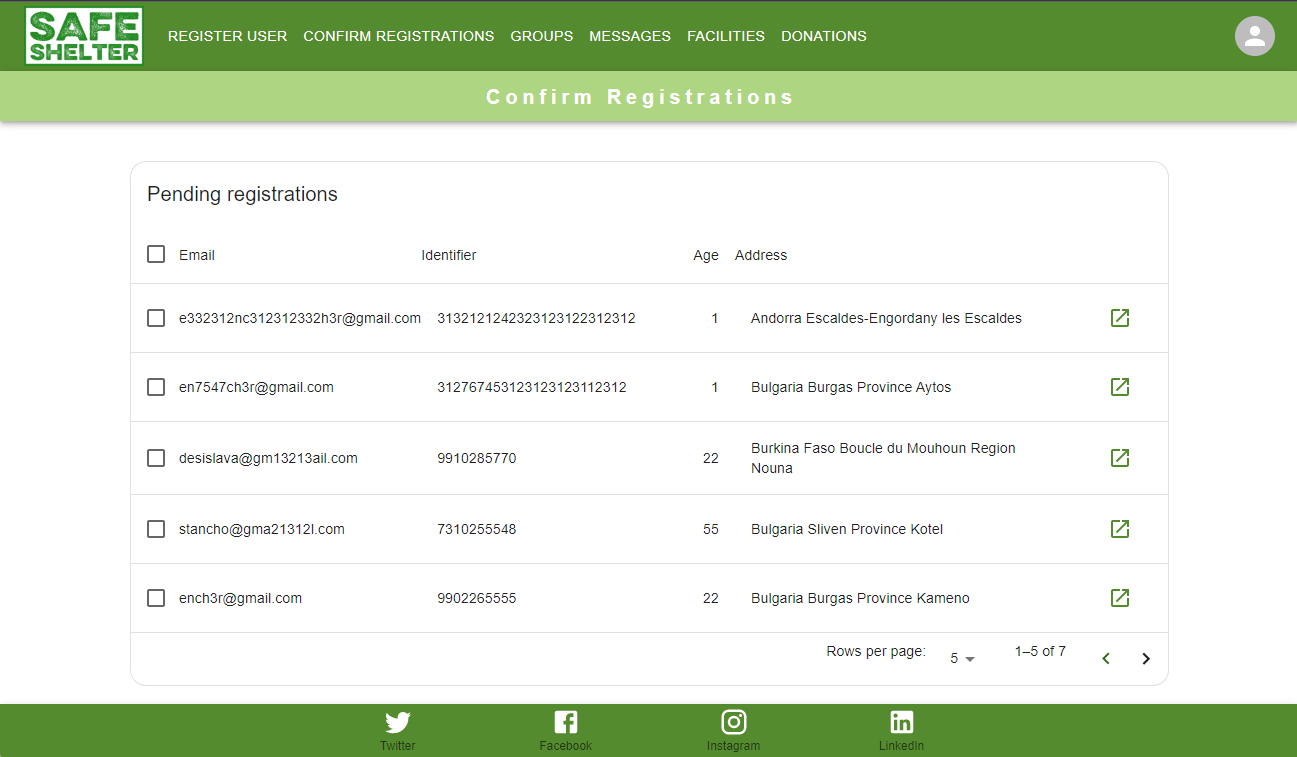
JavaMailSender предлага опция за съдържание на съобщението в html формат, което позволява форматиране:



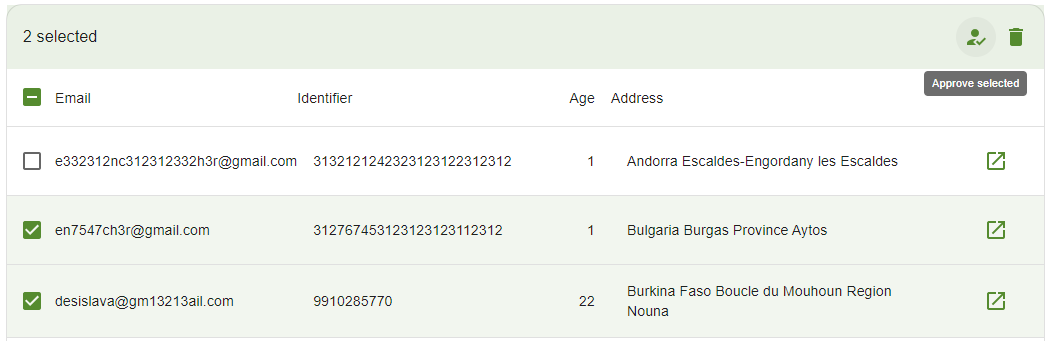
*Фиг.26. Темплейт на мейл за нов потребителски профил*

### 5.6.2 Потвърждаване на регистрации

Създадени регистрации на бежанци от модератори следва да бъдат потвърдени от администратор. За тази цел се налага създаване на панел с регистрации, чакащи потвърждение:

*Фиг.27. Регистрации, изчакващи потвърждение*

Тук се визуализира таблица с всички регистрации на бежанци, които не са потвърдени от администратор. Има възможност за избор на няколко елемента, маркиране на всички регистрации. При избор на един или много елементи, администратор може да избере дали да потвърди избраните регистрации или да ги откаже.



След потвърждение на регистрация се изпраща мейл на за новорегистриран потребител, съдържащ паролата на профила. Ако потребителска регистрация е била отказана от администратор, потребителят получава информативен мейл. На практика и в двата случая се изпраща мейл към потребителя, но в единия случай ако нямаме изпратем мейл за нова регистрация, потребителят няма да получи парола, а в другия случай потребителят просто няма да бъде уведомен.

Поради тази причина, при изпращане на мейл за потвърдена регистрация, сървърът проверява дали мейлът е изпратен успешно. Потвърждението не може да продължи без изпратен мейл и води до край на транзакцията и връщане на промените.

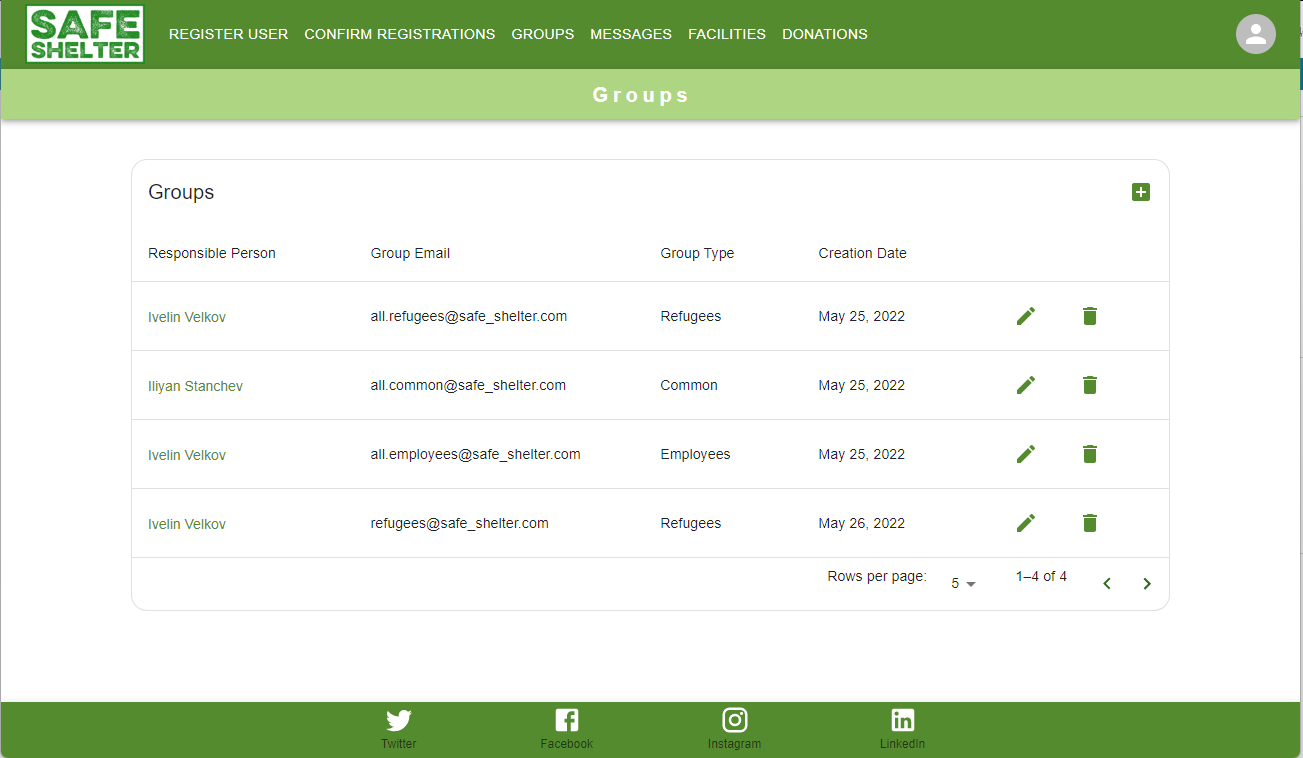


От друга страна, мейлите за отказани регистрации не е нужно да забавят основния процес, който се грижи да отговора на заявката. Такива мейли се записват в опашкова таблица в базата и се изпращат от background процес през определен период от време.



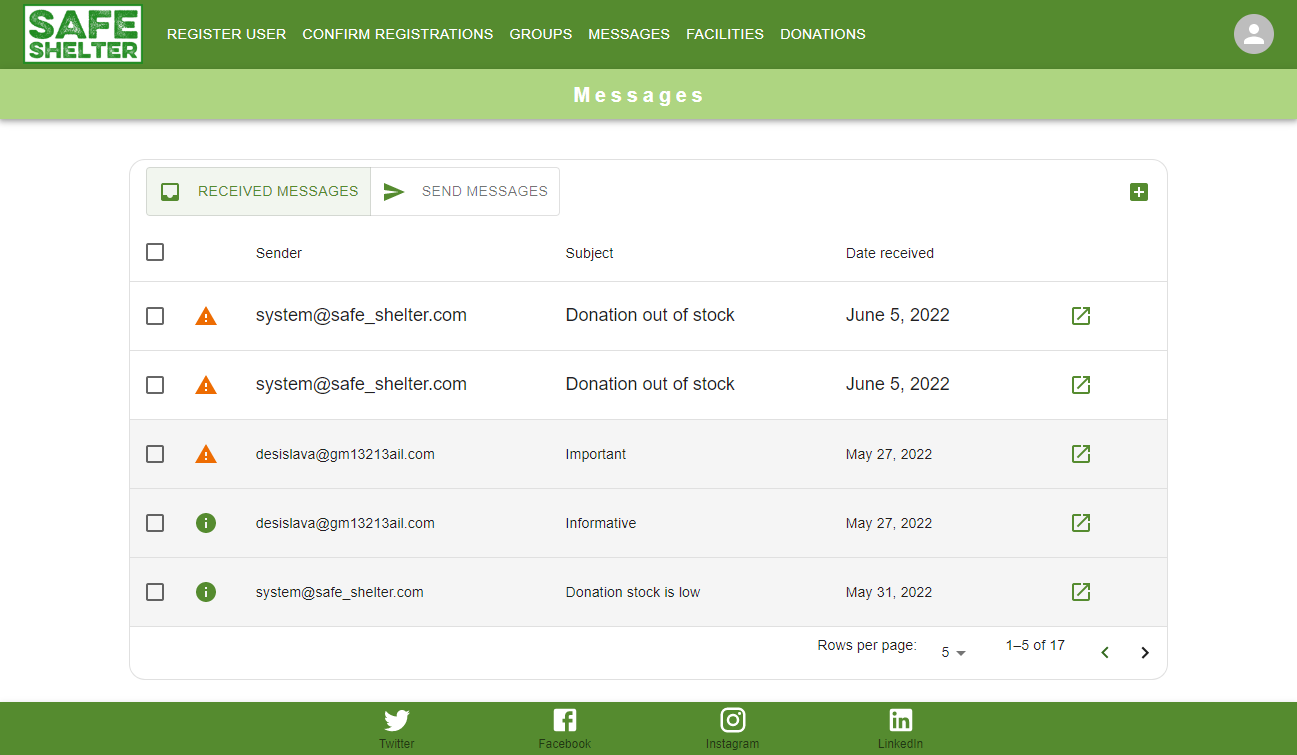
Показаният метод се грижи за изпращането на мейли от опашковата таблица. През 30 секунди се сабужда нишков процес, който извлича чакащите мейли и ги изпраща.

### 5.6.3 Потребителски групи

*Фиг.28. Потребителски групи*

Потребителските групи имат отговорно лице, мейл тип на групата. Могат да се създават различни потребителски групи ( общи, бежански, групи на служители ). Системата разполага с общи групи, които съдържат всички потребители. При добавяне на потребител в системата, той автоматично става член на тези групи. Не се допуска премахване на потребители от тези групи, както и тяхното изтриване. Връзкате между потребители и групи е много към много. Това позволява един потребител да участва в няколко групи.

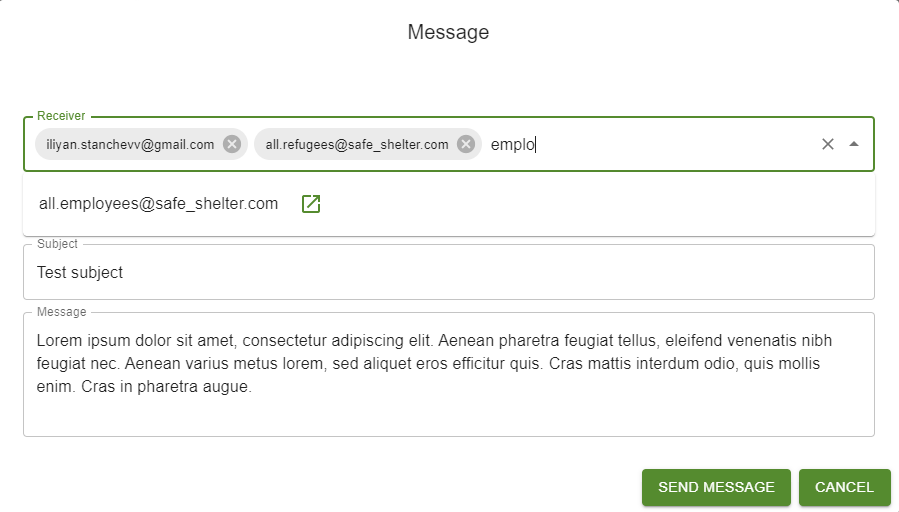
### 5.6.4 Съобщения



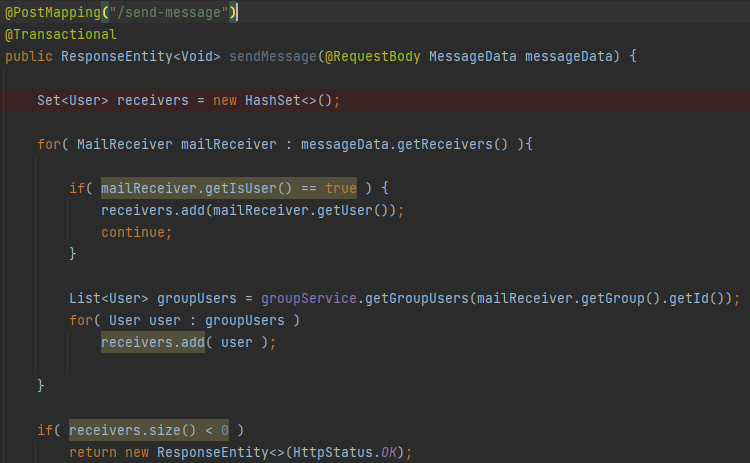
*Фиг.29. Съобщения*

Системата позволява изпращане на съобщения до други потребители. Модулът за изпращане на съобщения е реализиран по подобие на Gmail. Има два таба, получени и изпратени съобщения. Получените съобщения се зареждат по дата, като непрочетените са с приоритет. Прочетените съобщения са засивени. За всеки ред се визуализира видът на съобщението като икона. При отваряне на съобщение, то се маркира като прочетено. Има възможност за селектиране на няколко съобщения и изтриване/ маркиране като прочетени на избраните.

Диалогът за изпращане на ново съобщение изглежда по следния начин:

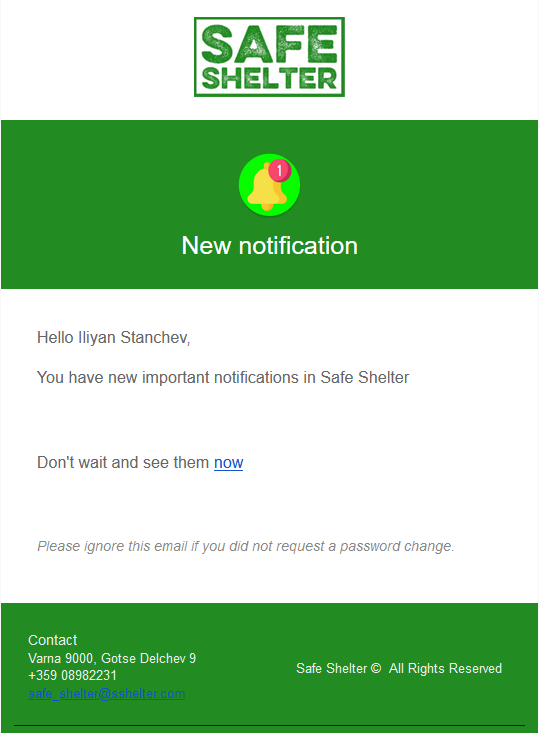


*Фиг.27. Ново съобщение*

Получателите могат да са няколко, като може да се избират мейлите на всички потребители в системата, както и на всички потребителски групи. Тъй като е възможно да един потребител да е член на много групи, на сървъра се извличат всички потребителски имейли, на които трябва да се изпрати съобщението:



След това се записва съобщение за всеки получател. Ако съобщението е маркирано като ‚Важно‘, се изпраща мейл на получателите, с нотификация, че имат важни съобщения в приложението.



*Фиг.28. Темплейт за ново известие в Safe Shelter*

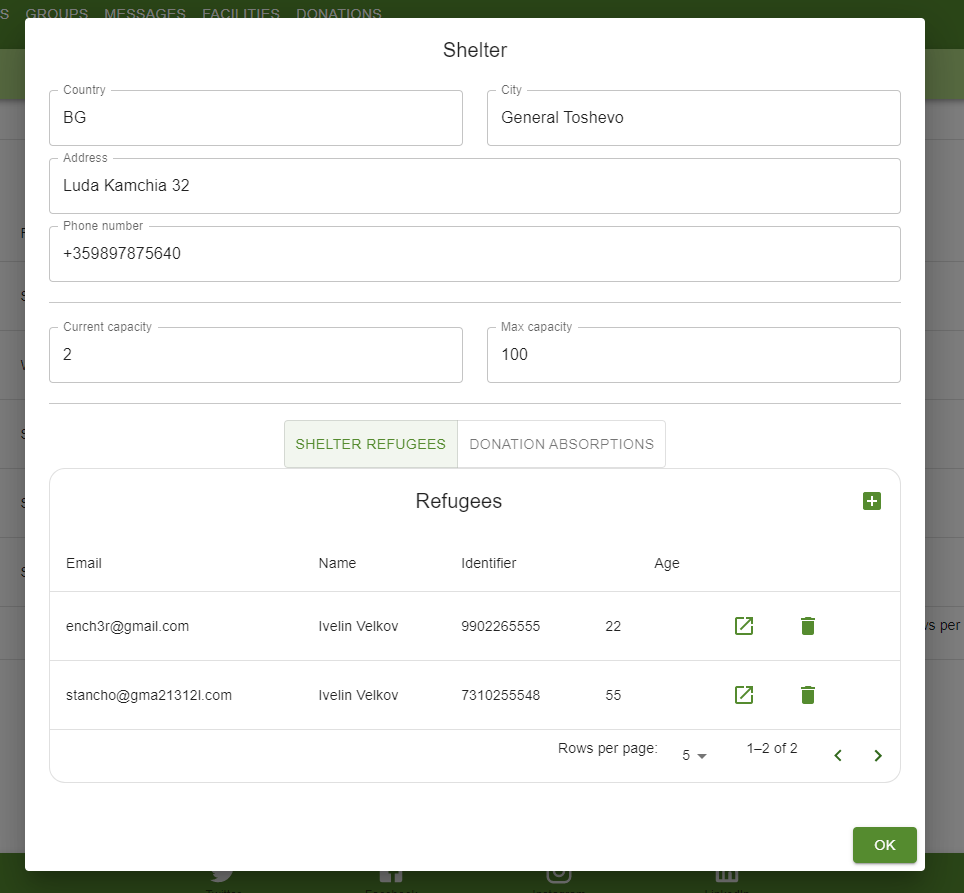
### 5.6.5 Съоръжения



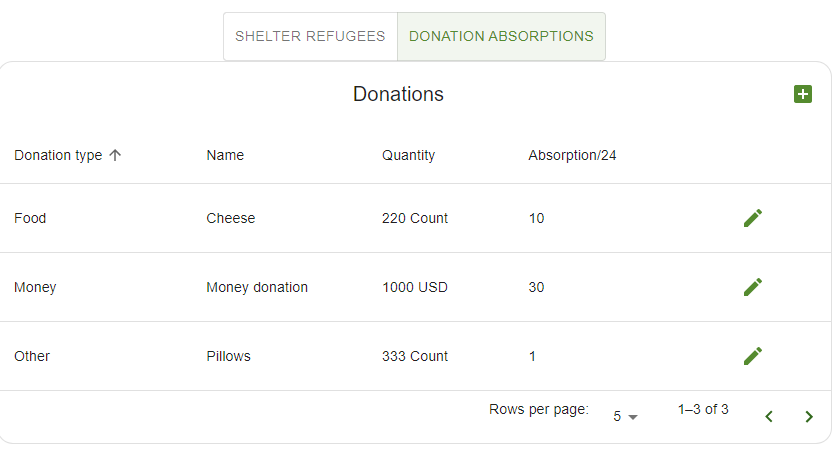
*Фиг.29. Съоръжения*

Съоръженията в системата са регистрирани складове за съхранение на запаси и бежански лагери. Всяко съоръжение си има отговорно лице, като при клик на името му се визуализира допълнителна информация. Има възможност за добавяне на нови бежански лагери и складове. При отваряне на бежански лагер се визуализира деталйна информация. Има опции за добавяне на бежанци в лагера, както и за управление на усвояването на даренията за конкретния лагер.

Тук един потребител може да е само в един бежански лагер, усвояването на даренията може да е различно за всеки лагер. Всеки лагер може да има нужда от различни дарения. Потребител може да добавя нови усвоявания или да редактира текущи.

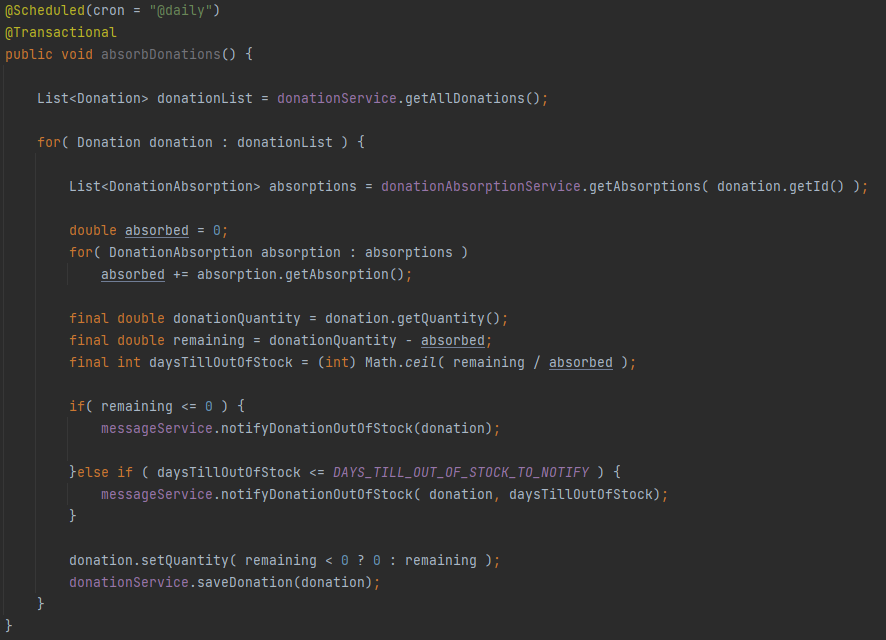


*Фиг.30. Редакция на съоръжения*

**

*Фиг.31. Промяна на усвояванията*

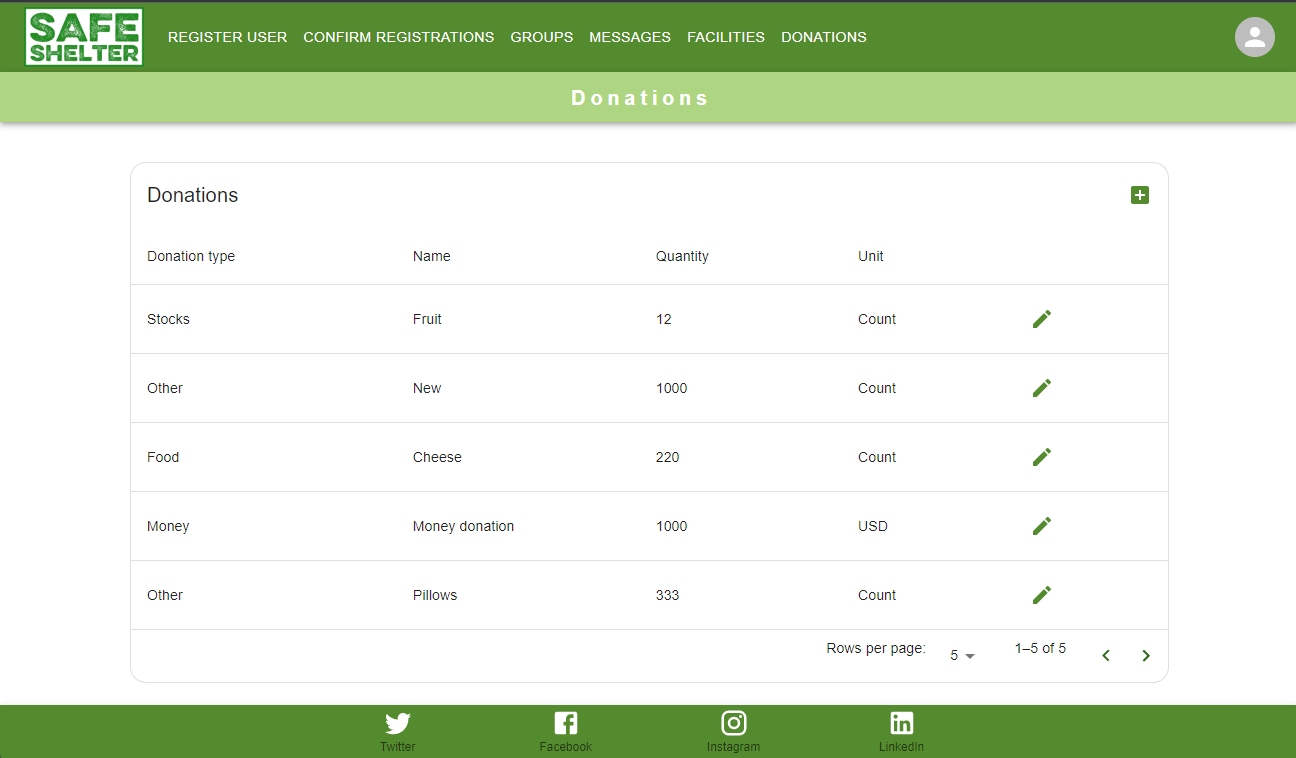
Усвояването на даренията се случва чрез систематизиран процес, който се изпълнява веднъж на ден.



Всяко регистрирано дарение има подчинена таблица с усвоявания, където се пази информация за усвояването на всеки бежански център спрямо настройката. При изпълнение на процедурата за усвояване се събира стойността на усвояването от всички центове и се изважда от общото количество на дарението.

Осигурени са известия по мейл, които се изпращат на всички служители, ако текущия запас на даренията ще свърши за по- малко от седем дни или ако запасът е свършил. Такива системни съобщения се изпращат от системен потребител.

### 5.6.6 Дарения



*Фиг.32. Дарения*

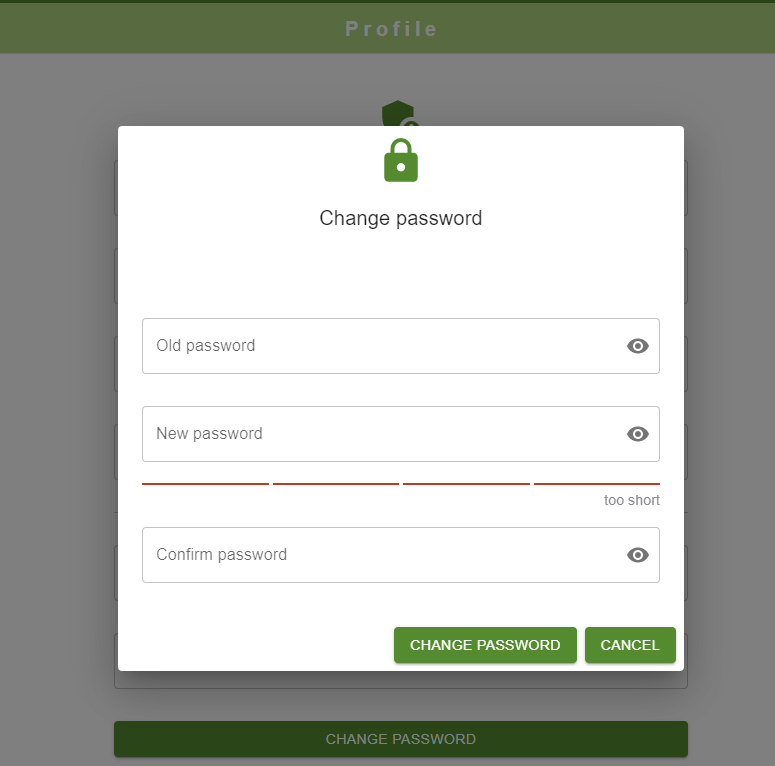
Регистрирането на дарения в системата става чрез добавяне на нови записи или актуализиране на съществуващи. Всяко дарение има име, тип, количество и мерна единица. При регистриране на ново дарение се търси съвпадение по име на дарение. Ако има вече такова дарение, новите количества се добавят към текущия запас.Добавянето на нови дарения се случва като се запише несъществуваща стойност за име на дарение.

Освен регистриране на нови дарения, потребителят има възможност да редактира съществуващи. Позволените полета за редакция са тип на дарението, количество и мерна единица.

При регистриране на ново дарение, донорът може да пожелае да остави лична информация ( име, мейл ). Тези данни се съхраняват в таблица с донори и се визуализират в страницата за контакти на Home Interface.

### 5.6.7 Потребителски профил

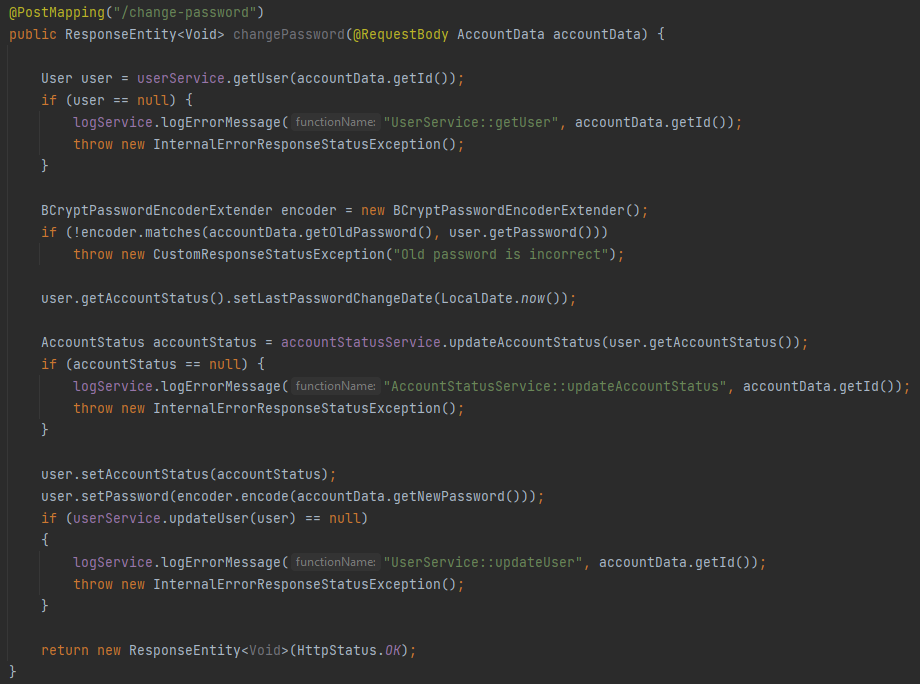
Потребителският профил за администратори и модератори съдържа системни данни за потребителя, тъй като други не са приложими. При отваряне на потребителския профил има опция за смяна на парола:



*Фиг.33. Смяна на парола*

Задължително за попълване е предишната парола на потребителя. Новата парола следва да отговаря на изискванията за силна парола. В React има много компоненти за проверка на силна парола. Следва задължително потвърждаване на новата парола.

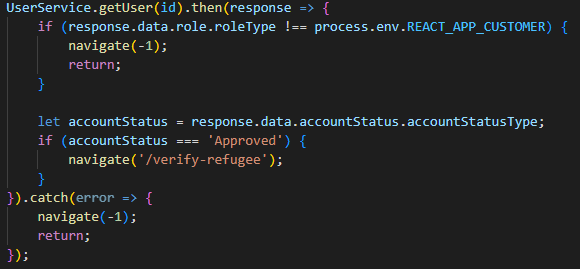
При успешна валидация на клиента се изпраща заявка към сървъра, която сравнява старата парола с тази от базата. При успех криптира новата парола, записва я в базата и променя датата на последна промяна на паролата.



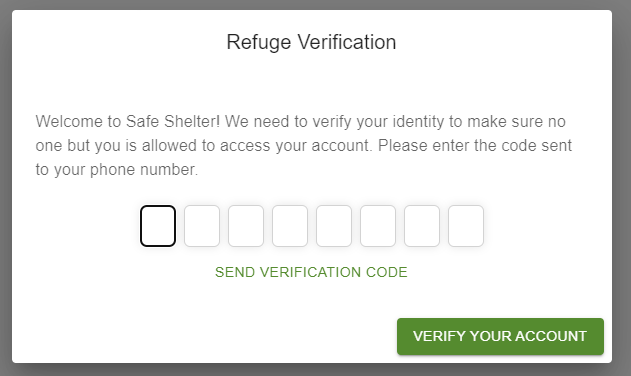
## 5.7 Модул за бежанци

Модулът за бежанци разполага с по – прост потребителски интерфейс и предоставя възможност за изпълнение само на необходими заявки. Бежанците също могат да изпращат и да получават съобщения. Това е единствената функционалност от модулите на служителите, която присъства и тук.

Обслужването на бежанци изисква по – стриктен контрол и верификация на потребителския профил. След одобряване на бежанец, профилът му става в статус „одобрен“, но този статус не е достатъчен.

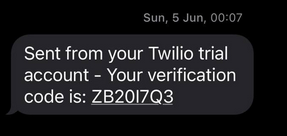


Одобрени потребители се препращат към екран за верификация:



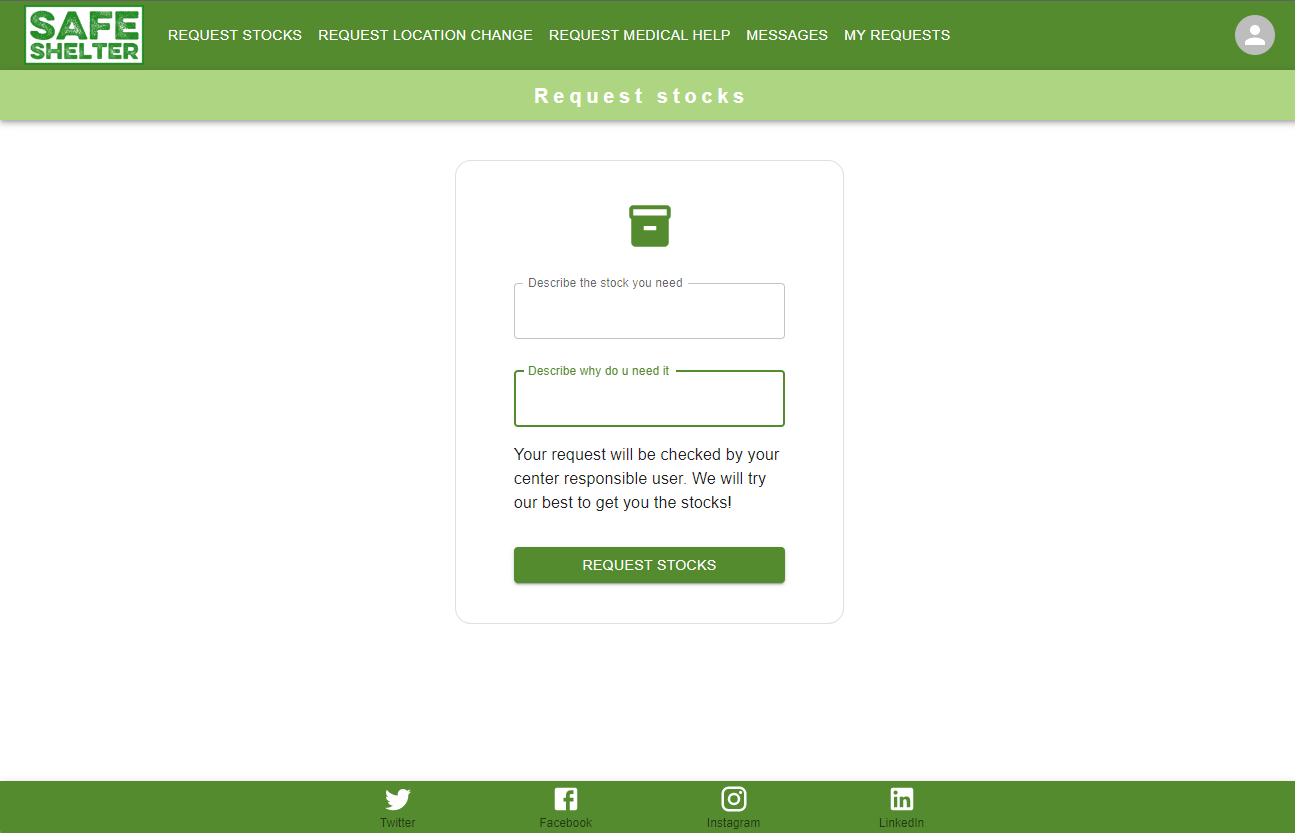
*Фиг.35. Верификация на одобрен потребител*

Верификацията позволява на потребител да изпрати текстово съобщение до телефона, който е дал като телефон за връзка при регистрацията си в бежанския център. При изпращане на код се изпраща текстово съобщение, съдържащо генериран потребителски код, който е валиден десет минути:



Код за потвърждение може да се изпраща през една минута. При въведен правилен код потребителският профил се верифицира и той вече има достъп до системата.

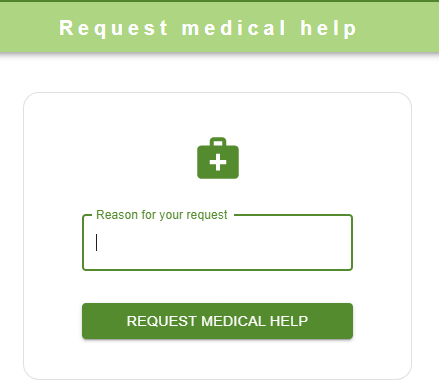
### 5.7.1 Заявки



*Фиг.34. Модул за бежанци, заявка за материални запаси*

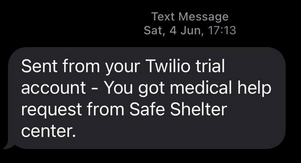
Бежанският модул поддържа изпращане на заявки за материални запаси. Нужно е описание на стоката, от която се нуждае потребителя и причина за нейната необходимост. При нова заявка се добавя запис в таблица със потребителски заявки в статус „чакаща одобрение“. Всички заявки могат да бъдат одобрявани от отговорното лице за бежанския център.

Системата предлага също така и заявки за смяна на бежански център. При този тип заявка потребител може да избере нов бежански център, в който желае да бъде приет. Отново е необходимо да бъде написана причина за заявката.

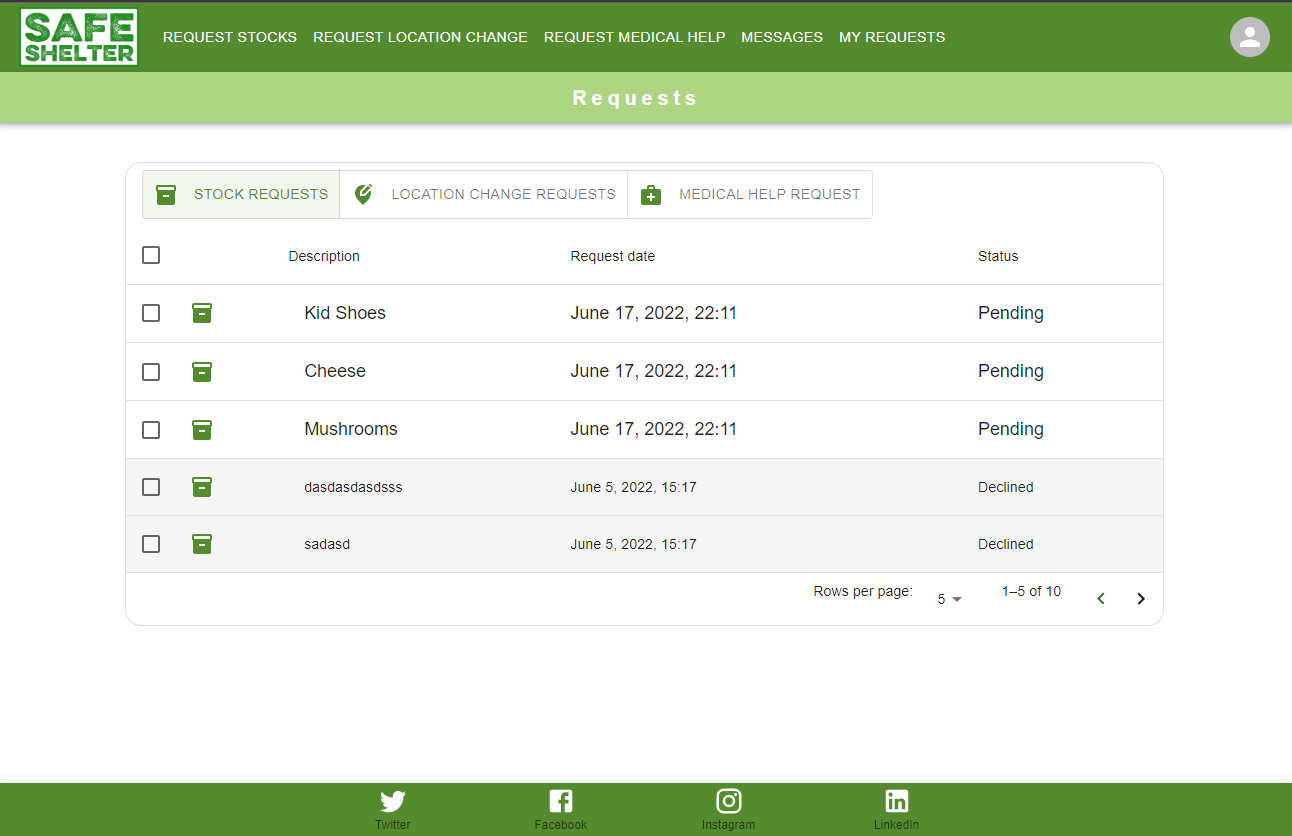


*Фиг.35. Изпращане на заявка за медицинска помощ*

Заявките за медицинска помощ са единствените, които могат да бъдат изпратени без допълнителна информация. При заявка за медицинска помощ се изпраща текстово съобщение на телефона на съответния бежански център.



### 5.7.2 Преглед на изпратени заявки



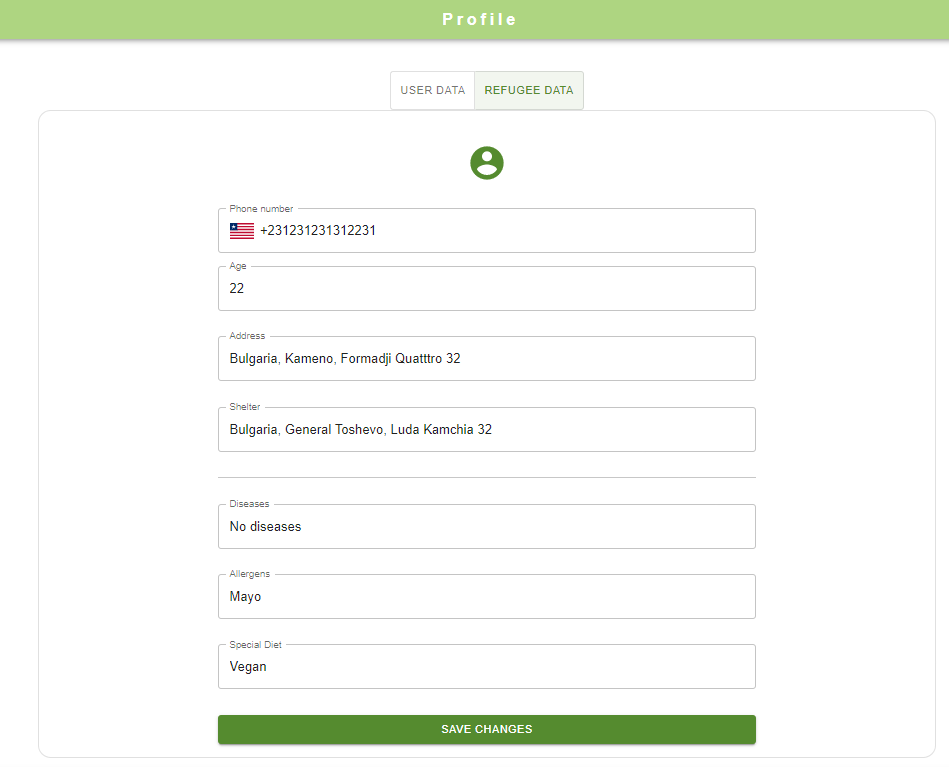
*Фиг.37. Преглед на изпратени заявки*

Потребител може да преглежда изпратените от него заявки и да проследява техния статус. Заявките остават в статус „чакаща“, докато служител не ги одобри или откаже. Потребител има възможност да отказва сам своите заявки, в случай, че е объркал нещо. Не е възможно потребител да има повече от една чакаща заявка за промяна на бежански център.

### 5.7.3 Потребителски профил

Потребителският профил е разделен на две страници:

* Данни за потребителски профил: Тук се визуализират системните данни за профила, които се визуализират и при изпор на профил на служител;
* Данни за бежанец: Тук се визуализират специфични данни за бежанците, като част от тях могат да бъдат редактирани от текущия потребител:



*Фиг.38. Редактиране на потребителски профил*

Възможните полета за редакция са телефонен номер, възраст, заболявания, алергени и специална диета.

# 6. Тестове и резултати

Тестването на системата е метод за определяне дали нашият продукт отговаря на очакваните изисквания и специфики. Чрез тестове можем да си гарантираме, че базови функционалности, които са важни за работата на приложението, няма да бъдат неволно счупени от нови функционалности. При добавяне на нов код често се случва да се редактират стари механизми, което може да доведе до дефекти в системата.

## 6.1 Тестове

Системните тестове тестват основни заявки към сървъра. Проверява се дали отговорът на заявката спрямо входните данни е очакван или не:

* Get заявка за потребител:



* Get заявка за потребител с грешни данни:



* Post заявка за вход в системата:



* Post заявка за проверка на код за верифициране:



* Post заявка за създаване на служител с грешка за съществуващ мейл



* Post заявка за създаване на служител



## 6.2 Резултати

Резултатите от тестването показват нормално поведение на софтуера при изпращането на заявки с разнотипни данни. Както се вижда на снимката всички заявки се обработват сравнително бързо с изключение на заявката за създаване на служител.

Успешното създаване на служител изисква повече време, тъй като се изпраща мейл на новорегистрирания потребител.



# 7. Проблеми и решения

По време на разработката имах за цел да се постарая да правя всеки модул и всяка функционалност както трябва, а не просто да работи. Исках етапите на различните процеси да преминават последователно и да бъдат достатъчно разбираеми за потребителите. За да се случват нещата както трябва, обърнах малко повече внимание на процесите в различни утвърдили се вече софтуерни продукти. Взаимствах възможно най – много от тях, тъй като самият аз съм доволен от начина им на работа и потребителското преживяване.

## 7.1 Защита от спам атаки и настройка на reCAPTCHA:

Спам атаките от неоторизирани потребители са често явление. Затова е създадена reCAPTCHA. Продуктът на Google не е пригоден за динамичните обновявания на React, което водеше до трудната му инициализация в компонента. За щастие React има осигурена библиотека с компонент, капсулиращ работата на reCAPTCHA, който върши работа.

## 7.2 Изпращане на мейли през Gmail:

Gmail е най – известната платформа за изпращане на мейли, не само заради лесното използване, но и заради добрата защита на потребителските данни. Това означава, че не е толкова лесно да изпращаш мейли през неоторизирано third – party приложение. Доскоро Gmail позволяваше настройка на потребителския профил, където се позволяваше достъп до „ не чак толкова сигурни приложения „.

Тази настройка позволяваше да се изпращат мейли чрез изчитане на данните за логин от настройка на сървъра. От май месец тази настройка не съществува, което блокира напълно възможността за изпращане на мейли по този начин. Текущата опция е генериране на парола за използване от приложение. За всяко приложение трябва да се използва различна парола, която ти предоставя Gmail.

## 7.3 Кодиране на данни

Пароли и кодове за верификация на потребители никога не трябва да се съхраняват в базата в чист формат. За тази цел се прилага криптиране на данните. Използваните алгоритми за кодиране или хеширане на данни затрудняват разобличаването на информацията, в случай че злонамарен потребител е откраднал чувствителни данни.

За целта в приложението се използва BCrypt – алгоритъм за хеширане на пароли. Той позволява енкодирането, но не може да декодира парола, което го прави още по – сигурно решение.

Името BCrypt идва от:

* B ( Blowfish ): кодиращ алгоритъм с променлива дължина. Един от най- бързо шифриращите се блокове алгоритми;
* Crypt: функция за хеширане на пароли в UNIX;

## 7.4 Достъп до потребителски страници от неоторизиран потребител:

Потребителския интерфейс на неоторизиран потребител не позволява достъп до вътрешни страници преди логин, но това не ограничава потребителите да достъпят страница чрез нейния URL.

За целта всеки логнат потребител създава потребителска сесия, която се съхранява в session storage на браузъра. При опит за достъп до оторизирана страница се проверява дали текущата инстанция на приложението има създадена потребителска сесия и дали ролята на потребителя има достъп.

Всяка потребителска сесия изтича след един час и е нужно да се влезе в системата наново.

## 7.5 Конкурентен достъп до ресурсите:

Настроено е до 100 нишки да могат да изпращат мейли паралелно. Това води до конкурентна манупулация на данните. Възможно е за части от секундата един изчакващ мейл да бъде обработен от няколко нишки, което ще доведе до изпращането на мейла няколко пъти.

Проблемът се решава чрез резервиране на мейли за обработка. Всяка нишка си има уникален идентификатор, който се раздава от операционната система по време на изпълнение. Този идентификатор е достъпен от приложението и може да се използва. Първа стъпка при обработването на мейли е всяка нишка да резервира до сто мейла за себе си, като редактира чакащите записи в базата със своето ID.

Следва изчитане на резервираните данни и обработка на мейлите един по един. При възникнала грешка се освобождава записът от текущата нишка, за да може да бъде обработен от следваща след известно време. Ако мейл не се обработи успешно десет пъти, то той бива изтрит.

# 8. Заключение

## 8.1 Потенциално развитие на системата

Реализираният продукт „Safe Shelter – система за обслужване на бежански център“ удовлетворява базовите нужди на един бежански център според моите виждания. Нямам опит в тази сфера, което наложи да си представя как биха протичали процесите в реална среда. Макар да реализирах всички функционалности, за които се сетих, винаги има опции за потенциално развитие:

* Абониране на доброволци да получават нотификации при събития на бежанските центрове;
* Добавяне на нови роли в системата ( напр. Доброволец );
* Добавяне на курсове за научаване на език;
* Създаване на потребителска карта за всеки регистриран бежанец с QR код за верификация в центровете;
* Създаване на pdf файл с нужните средства от дарения за всеки бежански център;
* Експорт на логовете в системата в Excel за по – лесно намиране на проблеми;
* Разширяване на потребителските профили ( добавяне на снимки, допълнителна информация );
* Добавяне на страница за feedback, където всеки потребител да може да оцени обслужването;
* Създаване на мобилно приложение;

## 8.2 Заключителни думи

Разработката на “Safe Shelter” ми помогна да се поставя на мястото на един бежанец и на един служител в бежански център. Опитах се да пресъздам мислено целия този процес и да го реализирам като работещ продукт. Основната ми цел беше да науча нови неща, да се опитам да направя нещо, което не съм правил досега и да премина зоната си на комфорт.

Системата може да бъде подобрена. Интерфейсът, макар и взаимстван от множество утвърдили се продукти на пазара, не е ползван от друг потребител, освен от мен. Всяка една система трябва да се развива в насока удовлетворяване на потребителските нужди. Обратната връзка на потребителите и тяхното мнение относно системата трябва да се взимат под внимание и да се реализират корекции в най – кратки срокове.

Реализираният продукт, описан в текущата документация, отговаря на поставените в спецификацията на проекта характеристики. Системата осигурява удобен потребителски интерфейс и систематизира голяма част от процесите, като по този начин улеснява работата на служителите. „Safe Shelter” е сигурно място, което може да се справи с проблемите на един бежански център.

# 9. Речник

* API (Application Programming Interface) - приложен програмен интерфейс, представляващ набор от правила, която позволява на програмите да разговарят помежду си.
* Framework – инструмент, който доставя готови архитектурни решения и компоненти, които са настроени за да улеснят процеса на разработка.
* 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.
* Front-End – Уеб интерфейса на приложението, който е видим за потребителя.
* Back-End – Сървърната страна на приложението, която предоставя данни при поискване от FE или от някой API инструмент.
* DTO ( Data Access Object ) – класове, които отговарят за манипулирането на данните в базата
* MVC ( Model – View – Controller ) – архитектурен модел, който разделя приложението на три основни логически компонента, които са изградени да обработват отделни аспекти от системата.

# 10. 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.*
13. [*https://tallyfy.com/uml-diagram/*](https://tallyfy.com/uml-diagram/) *- статия за UML.*

# 11. Приложение