



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

## ДИПЛОМНА РАБОТА

по професия код 481020 „Системен програмист“  
специалност код 4810201 „Системно програмиране“

Тема: Приложение за действие при здравословни проблеми

Дипломант:

*Ивона Вълчева*

Научен ръководител:

*инж. Николай Николов*

СОФИЯ

2023



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Дата на заданието: 22.11.2022 г.  
Дата на предаване: 22.02.2023 г.

Утвърждавам: .....  
/проф. д-р инж. П. Якимов/

## ЗАДАНИЕ за дипломна работа

ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ  
по професия код 481020 „Системен програмист“  
специалност код 4810201 „Системно програмиране“

на ученика Ивона Светославова Вълчева от 12 в клас

1. Тема: Приложение за действие при здравословни проблеми
2. Изисквания:
  - Предоставяне на възможност за първоначална настройка.
  - Потребителят избира от предоставените му опции типа на събитието. Ако след определено време от натискане на бутона за спешност това не се случи, местоположение му се изпраща до номера по подразбиране.
  - Връзка с предварително определени номера.
  - Предоставяне на списък на здравните заведения, отговарящи на избраното събитие.
  - Възможност за автоматично задаване на навигация до необходимия обект и позвъняване.
  - Предоставяне на указания за действие при извънредни ситуации в зависимост от избора на потребителя.
3. Съдържание    3.1 Теоретична част  
                       3.2 Практическа част  
                       3.3 Приложение

Дипломант: .....  
/ Ивона Вълчева /

Ръководител: .....  
/ Николай Николов /

Директор: .....  
/ доц. д-ринж. Ст. Стефанова /

## **ОТЗИВ НА ДИПЛОМЕН РЪКОВОДИТЕЛ**

Настоящата дипломна работа има за цел да научи дипломанта Ивона Вълчева да решава основни задачи при разработката на софтуерни приложения като:

- изготвяне на план за работа
- проучване достойнствата и недостатъците на съществуващи по темата приложения
- бизнес анализ
- избор на подходяща съвременна технология
- реализация на приложението

Всички тези задачи са решени в дипломната работа. Трябва да се отбележи самостоятелността в работата на дипломанта, нейната прецизност и прилежност. С изготвянето на работещо приложение може да се счита, че работата има необходимата завършеност.

На основание гореизложеното предлагам дипломантът да бъде допуснат до защита и за рецензент - Невяна Алдева-Стоянова – н-к отдел в Дирекция комуникационни и информационни системи към МВР.

Научен ръководител: \_\_\_\_\_  
/ Николай Николов /

## **УВОД**

Дефиницията на думата “здраве” според Световната здравна организация (СЗО) се формулира като симбиоза от физическо, умствено и социално благополучие. Значителният прогрес, който човечеството е постигнало в борбата си с различните заболявания, допринася за подобряването на живота на хората и повишаване на продължителността му.

За съжаление, ненадейно човек може да бъде изправен пред здравословен проблем, който изиска вземането на своевременни мерки и дори намесата на медицински екип.

В стресови ситуации хората често се поддават на паниката и се затрудняват при вземането на решения. Това довежда до загубата на ценни минути, които могат да бъдат пагубни за човешкия живот. Нерядко се случва хората да не са запознати с необходимите процедури и да не знаят как да помогнат на пострадалия и дори да му навредят с действията си. Точното определяне на състоянието на пациента и неговото бързо лечение могат да спасят живота му или да му спестят трудностите, свързани с продължително възстановяване. Това подчертава важността на разработването на системи, които предоставят бърза и ефективна помощ в случаи на извънредни здравни събития.

Ако самите ние попаднем в подобна ситуация, не винаги близо до нас ще има човек, готов да се притече на помощ. В днешни времена процентово голяма част от населението дори и възрастните хора притежават мобилен телефон, разполагащ със сензорен еcran. Това позволява внедряването на различни софтуерни разработки, подобряващи ежедневието.

Целта на дипломния проект е създаването на мобилно приложение, до което при неочекван здравословен проблем, може да се допитаме и да получим насока как да преодолеем настъпилото положение. Ако установим, че ситуацията не е по силите ни, се предоставя възможност да уведомим съответните служби. В зависимост от здравословния проблем потребителят получава списък от здравни заведение, до които може да бъде навигиран или до които можем да се свърже чрез позвъняване. Мобилното приложение ще ни позволи да изпратим точните си координати и текущо време до избрани от нас мобилни номера, които да уведомим, че сме в състояние на неразположение и се нуждаем от тяхната помощ.

## **ПЪРВА ГЛАВА**

### **ПРЕГЛЕД НА СЪЩЕСТВУВАЩИТЕ ПРОГРАМНИ РЕШЕНИЯ И РАЗВОЙНИ СРЕДИ**

#### **1.1 Обзор на съществуващите на пазара приложения**

Нуждата от подобен тип програмен продукт е принудила пазара да създаде множество приложения за спешни случаи. Повечето от тях притежават подобна функционалност, като изпращане на GPS координати на определени контакти, предоставяне на инструкции за първа помощ, изпращане на SOS сигнали и др.

Голям процент от приложенията са разработени от неправителствени организации като Българския Червен кръст (БЧК), които притежават по-голям авторитет и достоверност, поради което могат да бъдат предпочитани от потребителите. Докато други са създадени от частни разработчици.

Можем да заключим, че пазарът на подобен тип приложения е конкурентен, като изборът на конкретно приложение зависи от личните предпочтения на потребителите.

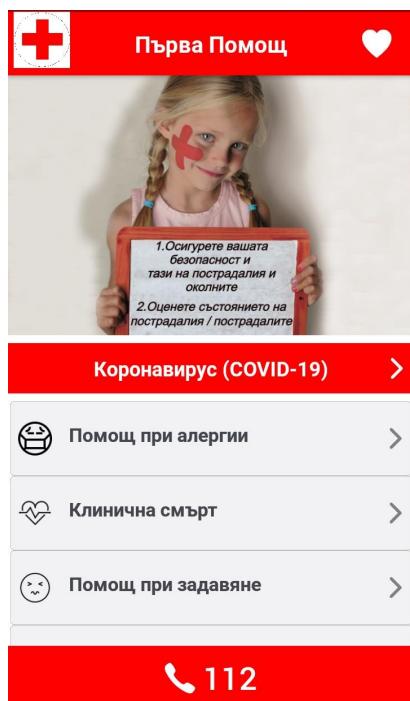
##### **1.1.1 Първа помощ БЧК**

Българският Червен кръст предостави на пазара бесплатно приложение за оказване на първа долекарска помощ в критични ситуации. То е достъпно в Google Play магазин и може да се използва от потребители на мобилни телефони с операционна система Android [1]. Разработено и дарено безвъзмездно от доброволеца и ИТ специалист Георги Тенев.

В приложението са заложени 15 критични ситуации. Няколко от тях са изобразени на *Фигура 1.1.*, като към всяка една е наличен кратък видеоклип с насоки, достъпен за гледане, още след стартирането на приложението, дори без наличието на интернет връзка. В него са взели участие известни български артисти. Разположението на списъка на стъпките за определена спешна здравословна ситуация на *Фигура 1.2.*.

Предоставя се възможност за директна връзка с телефонен номер 112 - "Европейски хармонизиран номер за достъп до спешни повиквания".

При желание от страна на потребителското лице, приложението притежава секция, в която той може да дари средства, подпомагайки дейността на Българския Червен кръст. Потребителския интерфейс, изобразяващ тази функционалност, е показан на *Фигура 1.3.*



*Фигура 1.1.*



*Фигура 1.2.*

### ХУМАННОСТ

Червеният кръст, роден от желанието да оказва помощ без дискриминация. Неговата цел е да закрия живота и здравето на човека, както и да изисква уважение към човешката личност. Той способства за установяването на взаимно разбирателство, дружба, сътрудничество и траен мир между всички народи.

---

### БЕЗПРИСТАСТНОСТ

Червеният кръст не проявява предпочтение по отношение на националност, раса, религия, социално положение или политически убеждения. Неговият стремеж е единствено да подпомага хората в зависимост от степента на страданието им и да осигури предимство на онези, които се намират в

ДАРЕТЕ СЕГА

*Фигура 1.3.*

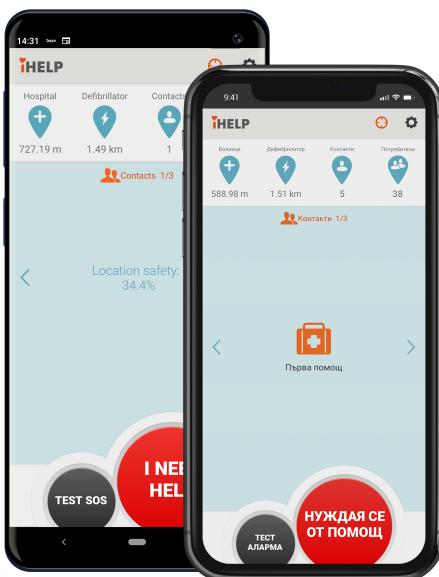
### 1.1.2 iHelp

Статистиката показва, че линейките в България пристигат след двадесетата минута от позвъняването по спешност, а човек на разстояние до 500 метра, разполагащ с познанията и нужните средства да окаже необходимата медицинска помощ, може да пристигне при пострадалия до 5 минути. Средната продължителност на времето, в което може да се спаси човешки живот при сърдечен проблем е от 5 до 7 минути. Имайки предвид късото време за реакция, често линейките не съумяват да помогнат на пострадалия да съхрани живота си.

Идеята за създаване на приложение, което да се заеме с решаването на този належащ проблем, е на предприемач от Словения, но лицето говорител, отговарящо за България, е Христо Христов от Габрово. Самата община също допринася за развитието и популяризирането [2].

Основната цел на проекта iHELP (превод на български - аз помагам) е активизирането на мрежа за взаимопомощ, обединяваща семейства, приятели, доброволци и професионални (медицински) спасители в случай, че дадено лице се нуждае от оказване на спешна медицинска помощ [3]. Чрез SOS бутон автоматично се алармират посочените лица и всички останали потребители в радиус от 300 метра от мястото на пострадалия, изпращайки се информация за местоположението, спешното събитие и други, свързани със случая, медицински данни. Паралелно самият той получава обратна информация за броя на хората, получили алармирането, колко от тях ще се отзоват и до какво време ще пристигнат при него. На всеки потребител се дава възможност да състави списък от три лица, които непременно да бъдат уведомени.

Апликацията има и опция връзка със спешен телефон 112. Идеята ѝ не е да изключи използването на бърза помощ, а точно обратното. Всичко е въпрос на избор, никой не е задължен, регистрирайки се, винаги да се отзовава при нужда, но колкото повече доброволци има, толкова повече се наброяват спасените животи. Потребителския интерфейс на приложението е изобразен на *Фигура 1.4.*.



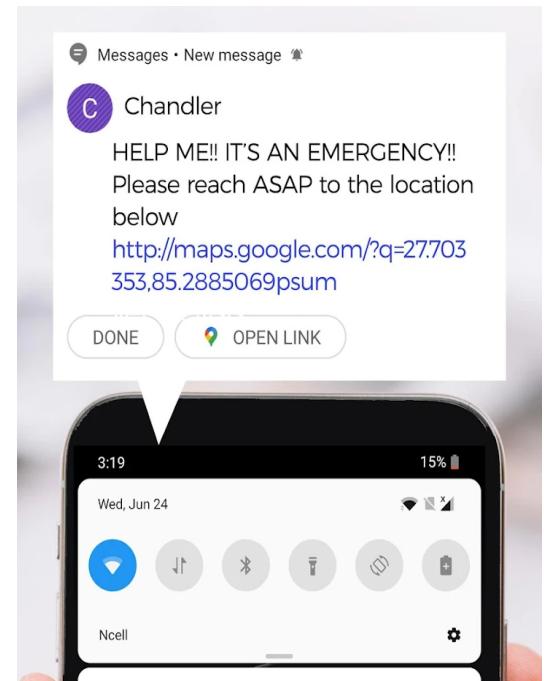
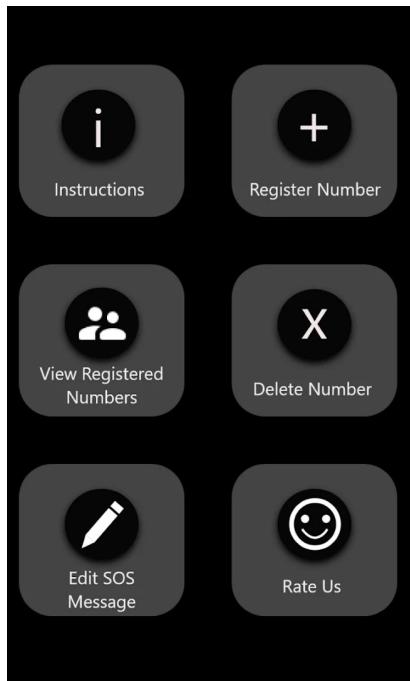
*Фигура 1.4.*

Друга функционалност, която се предоставя на потребителя, е списък на всички болнични заведения в страната. Спрямо местоположението на пострадалия се посочва най-близкото до него, отговарящо на настъпилия ненадейно здравословен проблем.

Мобилното приложение iHELP има възможността да предостави съвети за основните действия, които всеки да предприеме, докато очаква пристигането на помощта. Успоредно с това търси местоположението на най-близкия дефибрилатор, така че ако е необходимо да се стигне до него, то да е по възможно най-бързия начин, прибавяйки и инструкции как той да бъде използван.

### 1.1.3 SOS Alert | Emergency & Safety

Това е мобилно приложение, което предоставя възможност на потребителя да зададе телефонни номера по подразбиране, на които при спешен случай ще бъде изпратено неговото текущо местоположение, придружено със съобщение, което може да бъде променено от ползвателя на приложението. То представлява бутон за спешност, който при натискането му отброява 3 секунди, в продължение на които може да бъде прекратено. Изминели успешно указаното време, уведомителното съобщение за спешност е изпратено успешно [4]. Потребителския интерфейс на приложението е изобразен на *Фигура 1.5.*, *Фигура 1.6.* и на *Фигура 1.7.*.



*Фигура 1.5.*

*Фигура 1.6.*

*Фигура 1.7.*

## **1.2 Основни принципи, технологии за реализиране на мобилно приложение на операционната система Android и развойни среди**

Всяко едно мобилното приложение представлява софтуерна програма, която се използва на мобилни устройства като смартфони и таблети, предназначена да се изпълнява на определена операционна система, независимо дали това ще бъде Android, iOS или друго.

Android е операционна система с отворен код, създадена от Google, и се използва от много различни производители, включително Samsung, HTC, LG и други.

Приложенията за Android обикновено се пишат на Java или Kotlin, като се използва пакет за разработка на софтуер (SDK - Software development kit), който предоставя набор от инструменти и библиотеки за изграждане на приложения като: интегрирана среда за разработка (IDE - integrated development environment), поддръжка на език за програмиране, библиотеки за достъп до функциите на устройството, примерни кодове и шаблони, документация и уроци, инструменти за отстраняване на грешки и профилиране, емулатори или симулатори за тестване на приложението на различни устройства. Доста предпочитани за разработка на Android приложения са и технологии като React Native, Flutter и други.

Могат да се разпространяват чрез Google Play Store, който е основният пазар за приложения за Android, както и чрез други магазини за приложения и директно изтегляне.

Приложенията за Android могат да бъдат проектирани за широк спектър от цели, от прости игри и помощни програми до сложни корпоративни разработки. Те имат достъп до широк набор от функции на устройството, включително камера, GPS, сензори и съхранение, и могат също така да се интегрират с други услуги като: социални медии, платежни системи и облачни услуги.

Разработването на мобилно приложение за Android включва набор от задачи, включително: проектиране на потребителския интерфейс, програмиране на логиката на приложението, тестване и внедряването му на целевите устройства. Android Studio е основният инструмент, използван от разработчиците за създаване на приложения за операционната система Android, и предоставя набор от функции и инструменти в подкрепа на процеса на разработка.

### **1.2.1 Android Studio**

Android Studio е интегрирана среда за разработка (IDE) за създаване на мобилни приложения за Android, реализирана от Google [5]. Изградена е върху платформата IntelliJ IDEA и допринася с набор от функции в подкрепа на разработването на софтуер. Предоставя се редактор на продукта с функции като: довършване на код, подчертаване на синтаксис и форматиране, както и редактор на визуалното оформление за проектиране и разработка на потребителски интерфейси.

В допълнение към тези основни функции, Android Studio включва и инструменти за отстраняване на грешки, тестване и профилиране на Android приложения. Допринася също с вградена поддръжка на системи за контрол на версии като Git и Subversion и позволява на разработчиците лесно да създават и управляват виртуални устройства за тестване на своите приложения.

Използването му е бесплатно, достъпен е за изтегляне за операционни системи Windows, macOS и Linux. Той поддържа разработка на различни езици за програмиране, включително: Java, Kotlin и C++. С Android Studio разработчиците могат да създават широк набор от приложения за Android, от прости приложения, предназначени за един экран, до сложни такива.

Емуляторът на Android Studio е инструмент, който позволява на разработчиците да тестват своите Android приложения, независимо от езика, на който те са написани. Той е вграден в самата среда и предоставя широк набор от виртуални устройства с различни конфигурации, на които те да визуализират своите приложения.

Захранва се от Мениджъра за виртуални устройства с Android (Android Virtual Device (AVD) Manager), който предоставя набор от опции за конфигуриране на визуализацията включително: размер на екрана, разделителна способност, функции на устройството и версията на Android [6]. Разработчиците могат да тестват своите приложения за съвместимост с тези конфигурации, както и за производителност, използваемост и други фактори. Емуляторът включва функции като: симулация на сензорен еcran, поддръжка за множество типове въвеждане и възможност за заснемане на екранни снимки и видео.

### 1.2.2 WebStorm

WebStorm е интегрирана среда (IDE), специално създадена за уеб разработка. Тя е продукт на чешката компания JetBrains. Включва в себе си всички инструменти, необходими за създаване, редактиране и отстраняване на грешки в код на JavaScript, HTML, CSS, XML и други езици, свързани с мрежата. Предназначена е да направи процеса на разработка по-лесен и по-ефективен [7]. Съвместим е с операционни системи Windows, macOS и Linux.

Притежава вграден уеб сървър, който позволява на разработчиците да преглеждат и тестват своите уеб приложения, без да напускат средата. Поддържа интеграция с популярни инструменти за изграждане като Webpack, Grunt и Gulp, както и с популярни рамки за тестване като Jest, Mocha и Jasmine.

WebStorm също поддържа няколко различни рамки и библиотеки, включително: Angular, React, React Native, Vue.js и други.

Инструментът също така съдържа: интелигентен мениджър на изходния код, автоматичното му допълване и форматиране, както и навигация в него. Притежава също: визуален дебъгер, интегриране на контрол на версии и други. Тези функции подпомагат разработчиците в написването на по-качествен код.

Това е зряла платформа, известна със своите изключителни характеристики за разработка през последните 15 години. Един от плюсовете на средата е в интеграцията ѝ с Git и GitHub, а друг е поддръжката на голям брой плъгини. Недостатъците са, че самата платформа не е с отворен код и по отношение на производителността, някои други разработки надделяват.



Фигура 1.8. - Android Studio лого



Фигура 1.9. - WebStorm лого

## **ВТОРА ГЛАВА**

# **ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА МОБИЛНО ПРИЛОЖЕНИЕ ЗА ДЕЙСТВИЕ ПРИ ЗДРАВОСЛОВНИ ПРОБЛЕМИ**

### **2.1 Функционални изисквания към програмния продукт**

Идеята на дипломния проект е създаването на мобилно приложение, което да включва в себе си обединение на всички функционални изисквания, които да дадат възможност на потребителя при настъпването на здравословен проблем, най-бързо да успее да преодолее ситуацията. Самият той представлява симбиоза от съществуващите на пазара приложения, предоставящи най-основните и значими функционалности. Изискванията към програмния продукти са:

- При стартиране на приложението потребителят придобива възможността за първоначална настройка. Той може да добави определен брой телефонни номера по подразбиране. Те могат да бъдат от един до три на брой, в зависимост от предпочтитанието на потребителя. Задължително е да са валидни, ако не са, няма да бъдат приети. Необходимо е да се потвърди разрешението за достъп до текущо местоположение по време на използване на приложението. След тези настройки на потребителя не му се налага да повтори процедурата отново.
- Имплементация на бутон за спешност, който да бъде използван при нужда от медицинска помощ. След натискането му се предоставя списък от събития, от които потребителя има възможност да избере. Ако не го направи, му се позволява да изпрати SMS съобщение до номерата по подразбиране, които е въвел в първоначалните настройки.
- Опция за връзка с предварително определени номера за спешност 112 и 150.
- На потребителя след избор на спешно събитие му се предлага филтриран списък на здравни заведения, отговарящи на посочения случай.

- След като бъде избрано здравното заведение, се дава възможността за задаване на автоматична навигация до необходимия обект и телефонен номер за позвъняване.
- Предоставяне на списък от указания за действие при извънредни здравословни ситуации в зависимост от избора на събитие.

## 2.2 Съображения за избор на програмен език

### 2.2.1 JavaScript срещу TypeScript

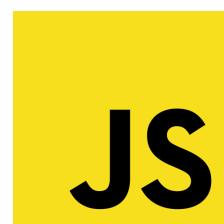
В света на програмирането съществуват статично и динамично типизирани езици. Разликата е в това, че при статичните езици е необходимо да декларираме типа на данните на променливите, в момента когато ги дефинираме. Те са безопасни за работа, тъй като при подаване на различен вид от необходимия, ще получим грешка. Динамичните езици от своя страна оптимизират скоростта и лекотата на използване. Те ни позволяват да предаваме всякакъв тип данни към променлива, това описва защо безопасността им е значително по-малка.

Това е основната разлика между JavaScript и TypeScript [8]. Докато JavaScript е динамично типизиран, TypeScript е статично надмножество на JavaScript. Това означава, че предлага стриктно статично въвеждане като опция, но ще позволи и динамично такова. Той добавя функции само към езика, не го ограничава или променя. В резултат на това кодът на TypeScript е по-безопасен, но малко по-сложен за писане, което води до повече грешки в разработката.

TYPESCRIPT	JAVASCRIPT
По-малко проблеми и грешки в производството	По-малко съобщения за грешки в разработката
По-висока крива на обучение	По-ниска крива на обучение
Компилируем език	Интерпретирам език
Може да се компилира във всеки тип JS, включително ES5 и ES6	Не може да се компилира във всеки тип и е написан само в една версия

Таблица 2.1. - Различия между JavaScript и TypeScript

Изборът за програмен език в дипломния проект е JavaScript. Той е по-добрият избор при разработката на по-малък по обем проект, не се налага използването на функции и рамки, изискващи строга типизация. Сравнен с TypeScript той предлага по-бързо изграждане на проект. По-широко разпространен е, което прави използването му много по-лесно, както и информацията, свързана с него, по-голяма.



Фигура 2.2. - TypeScript лого

Фигура 2.3. - JavaScript лого

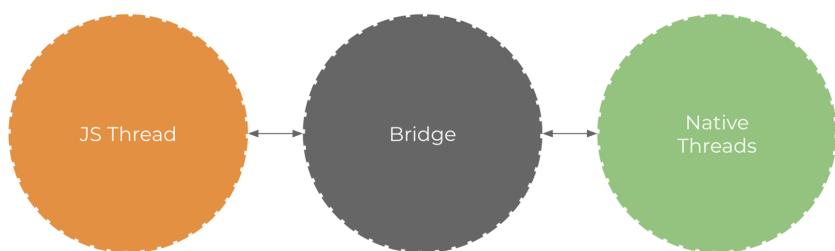
## 2.2.2 Какво представлява React Native

React Native е популярна рамка, фреймуърк за мобилни приложения, смес от JavaScript и специален език за маркиране JSX, който като синтаксис се различава от маркиращите езици - HTML и CSS. Позволява на разработчика да създава свободно мобилни приложения за iOS и Android, използвайки една и съща кодова база, намалявайки времето и усилията за разработка. React Native поддържа възможността, промените, които се правят в кода, да се наблюдават в реално време, без да се налага да се презарежда приложението. Това значително ускорява процеса. React Native използва собствени компоненти, специфични за всяка платформа, което му позволява да осигури по-естествен вид и усещане [9].

За първи път е пуснат на пазара от Facebook като проект с отворен код през 2015 година. За къс период от време успява да натрупа славата като едно от най-добрите решения, използвани за мобилно развитие. Програмният език взима участие в едни от водещите мобилни приложения в света, включващи Instagram, Facebook и Skype.

React (известен още като ReactJS) е JavaScript библиотека, използвана за изграждане на интерфейса на уеб сайтове, той също е изработен от инженерния екип на Facebook и е проектиран да работи с HTML и CSS. React Native не е по-нова версия на React, а може да се възприеме като негово разширение.

Друга тяхна разлика е начина на изобразяване. React използва виртуалния DOM, за да актуализира ефективно потребителския интерфейс, когато се правят промени, докато React Native прилага концепция наречена “мост” [10]. Той е написан на Java / C++ и осигурява комуникация между нишката на основното приложение и нишката на JavaScript частта, използвайки персонализиран протокол за предаване на съобщения. Тази връзка е представена схематично на *Фигура 2.4*. По този начин се осигурява по-бърз и по-отзовчив потребителски интерфейс за мобилни приложения.



*Фигура 2.4.*

Технологиите се различават и по техните библиотеки с компоненти. Като заключение можем да отсъдим, че и двете технологии споделят някои прилики, но все пак те са различни инструменти с различаващи се цели.

### 2.2.3 React Native срещу Flutter

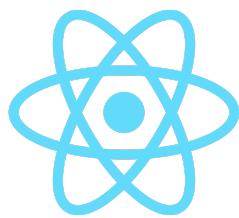
За разработката на тази дипломна работа е предпочтен React Native. В тази точка ще бъдат разгледани както неговите позитиви, така и негативи.

Flutter е мобилна рамка с отворен код, създадена от Ерик Сейдел - софтуерен инженер в Google, която използва различен подход за изграждане на мобилни приложения. Използва единична кодова база, написана на Dart, и включва собствен набор от персонализирани уиджети (widgets), които осигуряват изключително отзивчив потребителски интерфейс. Първата стабилна версия стартира през декември 2018 г. От пускането си на пазара Flutter придобива популярност сред разработчиците на мобилни приложения поради: бързия си цикъл на разработка, гъвкав потребителски интерфейс и висока производителност. Позволява създаването на висококачествени собствени интерфейси за iOS и Android. Flutter захранва не само много услуги на Google (като Google Ads), но и азиатския технологичен гигант Alibaba.

Една от основните причини изборът на технология за реализация на дипломната работа да е React Native, се основава на опита за разработка. JavaScript е популярен и широко използван език за програмиране, с който в някакъв етап всеки програмист е имал възможността да се срещне. Flutter използва Dart, който е съвсем нов език, създаден през 2011 г. от Google. От друга страна това, че е свеж продукт на пазара допринася за по-добре структурираната информация, която предоставя при разработка. React Native притежава известен брой изоставени библиотеки и пакети, които затормозяват процеса. Въпреки това превъзхожда, когато става въпрос за екосистемата. Притежава пет пъти повече пакети от Flutter, който наброя бройка над 1450 на свое разположение [11].

Друга причина за избор на React Native е по-широката общност от разработчици, поради по-ранното му дебютиране в сравнение с Flutter. Това допринася за по-голям обем от ресурси и ръководства за учене, той е имал повече време да бъде изпробван и усъвършенстван, което го направи по-стабилен и надежден. Поради голямата популярност могат по-бързо да се намерят отговори на проблеми, както голям брой готови решения.

Недостатък на Flutter е големият размер на неговите приложения. Наличието на малък размер на файла осигурява подобрено време за изпълнение и производителност. Освен това мобилните потребители се нуждаят от достатъчно място в паметта за съхранение.



Фигура 2.5. - React / React Native лого



Фигура 2.6. - Flutter лого

### 2.3. Локален запис на данни на мобилното устройство

Почти всяко приложение, което използваме или разработваме, трябва да съхранява някаква информация под една или друга форма за изпълнението дадена цел. При по-голям обем данни е необходимо да се изгради база. Тя представлява логическо свързани записи в конкретна област, структурирани по определен начин, които могат да бъдат извлечени по зададен критерий.

Информацията, която ни се налага да съхраним, е с малък обем, разработвайки мобилно приложение, ние можем и избираме да го направим в локалното хранилище. Всеки потребител при стартиране на приложението, въвежда своите данни, те се записват на самото мобилно устройство и независимо дали то бива затваряно или рестартирано, те не се губят до момента, когато самото приложението не бъде изтрито.

Съществуват множество решения за локален запис на данни. AsyncStorage е пристапа некриптирана асинхронна постоянна система за съхранение и организация на записи, тип ключ-стойност, която е достъпна в React Native. Тя позволява на разработчиците да запазят малки количества данни в хранилище, където те могат да бъдат записвани и извлечани въз основа на ключове. Максималния размер на хранилището за Android приложения е 6 MB. Данните могат да представляват токени за сесия, настройки и други подобни типове, които могат да се използват за подобряване на потребителското изживяване. Те не се споделят между устройства или потребители. Важно е обаче да се отбележи, че това не е заместител на пълнофункционална база данни или решение за съхранение от страна на сървъра. Ако трябва да съхраняваме данни, които се актуализират често, може да е необходимо по-стабилно решение.

Основен недостатък на AsyncStorage е, че представлява некриптирана система. Поради това чувствителна информация не е препоръчително да бъде съхранявана. Възможно е да се използват библиотеки за криптиране и декриптиране на данни. Една от тях е ‘react-native-crypto-js’. Тя поддържа различни алгоритми като ES, DES, Triple DES, Rabbit и други. Използва се ключ, благодарение на който се извършва криптирането. Библиотеката е с отворен код и е лесна за инсталација и използване.

В JavaScript Promise е обект, който представлява евентуалното завършване (или неуспех) на асинхронна операция и произтичащата от нея стойност. Promise е по същество заместител за стойност, която може да не е налична все още, но ще бъде в някакъв момент в бъдеще. Promise може да бъде в едно от трите състояния: в очакване, изпълнено или отхвърлено. В React Native, AsyncStorage е проектиран да се използва с Promises. Това означава, че всички методи на обекта AsyncStorage го връщат, което може да се използва за обработка на резултата от асинхронната операция.

# ТРЕТА ГЛАВА

## ПРОГРАМНА РЕАЛИЗАЦИЯ НА МОБИЛНО ПРИЛОЖЕНИЕ ЗА ДЕЙСТВИЕ ПРИ ЗДРАВОСЛОВНИ ПРОБЛЕМИ

### 3.1. Основни концепции в едно React Native приложение

React Native куките (hooks) ни позволяват да използваме състоянието и други функции на React в компоненти, без необходимост от създаването на клас или методи на жизнения цикъл. Те са ново допълнение във версия 16.8 и са се превърнали в популярен начин за писане на React компоненти. Наименуват се с префикс "use" (да използвам), последван от глагол, който описва тяхната цел, като useState, useEffect, useContext, useReducer, useRef, useCallback и много други.

#### 3.1.1 useState

Често по време на разработка на React Native приложение се налага актуализация на стойността, промяна на състоянието на дадена променлива. За целта се използва куката useState, която приема като аргумент първоначална стойност и връща масив с два елемента: текущата стойност на състоянието и функция за актуализиране на стойността. Чрез псевдокод е представена функционалността във *Фрагмент 3.1.*

```
import {useState} from 'react';

const [canContinue, setCanContinue] = useState(false);

if (status === 'granted') {
    setCanContinue(true);
}
```

*Фрагмент 3.1.*

#### 3.1.2 useEffect

Куката useEffect ни позволява да изпълняваме странични ефекти в нашите компоненти, като за пример - извлечане на данни от API. Приема функция за обратно извикване като първи аргумент и незадължителен масив от зависимости като втори. Ако му се подаде празен масив ( [] ), ще се изпълни само веднъж.

Функцията, която подаваме като първи аргумент, ще бъде извикана, след като компонентът бъде изобразен и всеки път, когато някоя от зависимостите се промени. Чрез псевдокод е представена функционалността във *Фрагмент 3.2.* .

```
import {useEffect} from 'react';

useEffect(() => {
  const getTime = () => {
    setTime(new Date().toLocaleTimeString());
  };

  getTime();
}, []);
```

*Фрагмент 3.2.*

### 3.1.3 useNavigation

Куката useNavigation, предоставена от React Navigation, ни позволява достъп до обекта за навигация, с който да се придвижим между еcranите в приложението. Чрез псевдокод е представена функционалността му във *Фрагмент 3.3.* . Като втори аргумент могат да се предават параметри, които “навигираме” към посочения от нас еcran.

```
import {useNavigation} from '@react-navigation/native';

const navigation = useNavigation();

const handleButtonPress = () => {
  navigation.navigate('PhoneNumberEntryScreen', {pickerNumber: pickerNumber} );
}
```

*Фрагмент 3.3.*

Във *Фрагмент 3.4.* отново с псевдокод е показано как се получават навигираниите от нас данни към определен еcran на приложението.

```
const PhoneNumberEntryScreen = ({route}) =>{
  const pickerNumber = route.params.pickerNumber;
}
```

*Фрагмент 3.4*

## 3.2. Представяне на първоначална настройка

### 3.2.1 Променлив брой полета за въвеждане на телефонни номера според избора на потребителя

Потребителят избира какъв брой телефонни номера иска да въведе. До тях ще бъдат изпращани СМС съобщения при спешност с точните координати и текущо време на потребителя. Във *Фрагмент 3.5.* е показана част от реализацията на алгоритъма.

```
import {Picker} from '@react-native-picker/picker';

const [pickerNumber, setPickerNumber] = useState(1);

<View style={styles.pickerContainer}>
  <Picker
    selectedValue={pickerNumber}
    style={styles.picker}
    onValueChange={handlePickerChange}
  >

    <Picker.Item label='1' value={1} />
    <Picker.Item label='2' value={2} />
    <Picker.Item label='3' value={3} />
  </Picker>
</View>
```

*Фрагмент 3.5.*

Променливата pickerNumber се предава като параметър на следващия екран. В зависимост от нейната стойност - цифра от 1 до 3, се определя броя на полетата за въвеждане на телефонни номера по подразбиране, които ще се визуализират. За изобразяването им се използва библиотеката “react-native-intl-phone-input”. Тя предоставя адаптивен и лесен за използване компонент за въвеждане на телефонен номер. Той включва инструмент за избор на държава, поле за въвеждане и помощна програма за форматиране според избраната държава. В разработваното приложение по подразбиране могат да се запишат само български номера. Това е задено стриктно като параметър. Информацията, която получаваме от полето за въвеждане е обект, който подлежи на форматиране, за да се извлече единствено нужната информация.

В комплект с нея е използвана библиотеката 'google-libphonenumber'. Тя предоставя валидация на въведената от потребителя информация спрямо избраната държава, в случая България.

Добавена е и допълнителна валидация на входните данни. Проверява се дали при запазване на информацията, полетата са празни и дали са въведени едни и същи телефонни на поне 2 от полетата. Псевдокод на двете библиотеки е представен на *Фрагмент 3.6.*

```
import IntlPhoneInput from 'react-native-intl-phone-input';
import PhoneNumber from 'google-libphonenumber'

const PhoneNumberEntryScreen = ({route}) => {

    const pickerNumber = route.params.pickerNumber;
    const [inputNumber, setInputNumber] = useState(Array(pickerNumber).fill(""));

    const phoneUtil = PhoneNumber.PhoneNumberUtil.getInstance();

    const handlePhoneInputChange = (index, value) => {
        const updatedNumbers = [...inputNumber];
        const formattedPhoneNumber = `${value.dialCode}${value.unmaskedPhoneNumber}`;
        updatedNumbers[index] = formattedPhoneNumber;
        setInputNumber(updatedNumbers);
    }

    const saveNumbers = async() => {
        let isValid = true;

        for(let i = 0; i < pickerNumber; i++) {
            if (inputNumber[i].trim() === "") {
                isValid = false;
                Alert.alert('Няма въведена информация за "Телефонен номер "' + (i + 1));
                break;
            }
        }

        const parsedPhoneNumber = phoneUtil.parseAndKeepRawInput(inputNumber[i], 'BG');

        if (!phoneUtil.isValidNumber(parsedPhoneNumber)) {
            isValid = false;
            Alert.alert('Въведенитеят телефонен номер ' + (i + 1) + ' е неправилен');
        }
    }
}
```

```

...
return(
    ...
    <View style={styles.inputContainer}>
        <IntlPhoneInput
            defaultValue = {number}
            defaultCountry="BG"
            flagStyle={{display: 'none'}}
            disableCountryChange={true}
            placeholder="Въведете номер тук"
            onChangeText ={{value => handlePhoneInputChange(index, value)}}
        />
    </View>
    ...
);

```

*Фрагмент 3.6*

### 3.2.2 Запис и извличане на въведената информация в локалното хранилище

След като потребителя успешно е въвел цялата нужна информация, тя трябва да бъде съхранена. За целта се използва библиотеката “@react-native-async-storage/async-storage”. Телефонните номера биват запазени в локалното хранилище на устройството. Важна стъпка е предоставянето на сигурност на тази информация. За да се осъществи това, се използва библиотеката “react-native-crypto-js”. Създава се секретен ключ, с помощта на който се криптират телефонните номера. Част от кода е изложена във *Фрагмент 3.7.* .

```

import AsyncStorage from '@react-native-async-storage/async-storage';
import CryptoJS from 'react-native-crypto-js';

const PhoneNumberEntryScreen = ({route})=>{

    const pickerNumber = route.params.pickerNumber;
    const [inputNumber, setInputNumber] = useState(Array(pickerNumber).fill(""));
    const [secretKey, setSecretKey] = useState("");

    useEffect(() => {
        const newSecretKey = CryptoJS.lib.WordArray.random(16).toString();
        setSecretKey(newSecretKey);
        dispatch({ type: 'SET_SECRET_KEY', secretKey: newSecretKey });
    }, [dispatch]);
}

```

```

const encryptedNumbers = inputNumber.map((number) => {
    const encryptedNumbers = CryptoJS.AES.encrypt(number, secretKey).toString();
    return encryptedNumbers;
});

const saveNumbers = async() => {
    ...
    try {
        await AsyncStorage.setItem('@number', JSON.stringify(encryptedNumbers))
    } catch (e) {
        console.log(e);
    }
    ...
};

}

```

*Фрагмент 3.7.*

Следователно, ако искаме да прочетем номерата от паметта на устройството с помощта на AsyncStorage, е необходимо първо да ги дешифрираме. За целта е нужно да се извика метода CryptoJS.AES.decrypt с шифрования номер и специалния секретен ключ. Това е кратко описание на *Фрагмент 3.8.* .

```

const encryptedNumbers = await AsyncStorage.getItem('@number');

if (encryptedNumbers !== null) {
    const decryptedNumbers = JSON.parse(encryptedNumbers).map((encryptedNumber) => {
        const decryptedNumber = CryptoJS.AES.decrypt(encryptedNumber, secretKey)
            .toString(CryptoJS.enc.Utf8);
        return decryptedNumber;
    });
    ...
}

} catch (error) {
    console.error(error);
}

```

*Фрагмент 3.8.*

### 3.2.3 Разрешение за използване на текущо местоположение

Задължава се потребителят да даде своето съгласие за използване на местоположението му по време на действие на приложението. В противен случай той не може да продължи към същинската част.

За целта се използва асинхронната функция `requestForegroundPermissionsAsync` от библиотеката “expo-location”. Отваря се прозорец с опции, от които потребителят трябва да избере. Връща се обект, който съдържа състоянието на разрешението. Ако то е “granted” (предоставено), потребителят може да продължи напред, в противен случай се показва предупреждение със съобщение, което го информира, че не е дал разрешение на приложението да има достъп до неговото местоположение. Реализацията е представена на *Фрагмент 3.9.*

```
import * as Location from 'expo-location';

const requestLocationPermission = async () => {
  let {status} = await Location.requestForegroundPermissionsAsync();

  if (status === 'granted') {
    setCanContinue(true);
  } else {
    Alert.alert('Не разрешихте използването на местоположението Ви')
  }
};
```

*Фрагмент 3.9.*

### 3.3. Библиотеката Redux

Възможността за първоначална настройка се предоставя само веднъж през целия живот на приложението. За да се постигне това, е необходимо да се контролира глобалното състояние. За целта се използва библиотеката Redux. Състоянието, което е под нейния контрол обикновено е по-сложно и може да бъде достъпно от множество компоненти в приложението, което улеснява потока на данни и действия. Изграждането на подобна структура може да се раздели на няколко точки:

- Actions (Действия) - чрез тяхна помощ се изпращат данни от компонент на приложението към библиотеката Redux. Те са единственият източник на информация. Това означава, че ако е необходима промяна на състоянието, ще бъде изпратена чрез тях.

- Reducers (Редуктори) - когато се изпрати действие за промяна на състоянието, редукторите имат задължението да направят необходимите промени и да го обновят с новата му версия.
- Store (Хранилище) - с помощта на редукторите може да се създаде хранилище, което съдържа цялото състояние на приложението.
- Components (Компоненти) - това е мястото, където се съхранява потребителският интерфейс на приложението и откъдето може да се променя или извлича глобалното състояние на приложението.

При запазването на телефонните номера, за да се осигури някаква доза сигурност, се налага те да бъдат криптирана. За целта се използва специален секретен ключ. За да можем да прочетем разкриптирана информацията от локалното хранилище, той отново се изиска. Това обуславя нуждата от запазване на стойността му. Redux библиотеката се превръща в решението на поставения проблем. Функцията, представена във *Фрагмент 3.10.* дефинира редуктор, чието първоначално състояние е нула. Той следи за типа на действие SET\_SECRET\_KEY. Когато го получи връща стойността на secretKey от обекта за действие. Ако редукторът получи друг тип действие, той връща текущото състояние такова, каквото е, без никакви модификации. С други думи, този редуктор е отговорен за управлението на състоянието secretKey - ключът, управляващ криптирането.

```
// Define a reducer to manage the state
const secretKeyReducer = (state = null, action) => {
  switch (action.type) {
    case 'SET_SECRET_KEY':
      return action.secretKey;
    default:
      return state;
  }
};

export default secretKeyReducer;
```

*Фрагмент 3.10.*

Във *Фрагмент 3.11.* функцията `combineReducers` приема обект като вход, където всеки ключ съответства на различен отрязък от състоянието, а стойността за всеки един от тях е съответната редуцираща функция, която управлява този отрязък. `SecretKeyReducer` се предава на `combineReducers` като стойност за ключа `secretKey` във входния обект.

```
import { combineReducers } from '@reduxjs/toolkit';
import secretKeyReducer from './SecretKeyReducer';

// Combine the reducers
const rootReducer = combineReducers({
  secretKey: secretKeyReducer,
});

export default rootReducer;
```

*Фрагмент 3.11.*

*Фрагмент 3.12.* експортира конфигурирано хранилище на Redux, което използва основния редуктор `RootReducer`. Първо, той импортира функцията `configureStore` от пакета `@reduxjs/toolkit` и редуктора от `RootReducer`, като извиква `configureStore` с аргумент на обект, който съдържа свойството `rootReducer`, което е присвоено на полето за редуциране.

```
import { configureStore } from '@reduxjs/toolkit';
import rootReducer from './RootReducer';

// Create a Redux store with the root reducer
const store = configureStore({
  reducer: rootReducer,
});

export default store;
```

*Фрагмент 3.12.*

*Фрагмент 3.13.* е функционален компонент на React, който използва куката `useDispatch` от `react-redux`, за да изпрати действие до хранилището на Redux. Компонентът инициализира променлива на състоянието `secretKey` с помощта на куката `useState` и след това използва куката `useEffect`, за да генерира нов 16-байтов таен ключ с помощта на библиотеката `CryptoJS` и изпраща действие Redux с генеририания ключ.

```
import {useDispatch} from 'react-redux';
const dispatch = useDispatch();

const [secretKey, setSecretKey] = useState("");

useEffect(() => {
  const newSecretKey = CryptoJS.lib.WordArray.random(16).toString();
  setSecretKey(newSecretKey);
  dispatch({ type: 'SET_SECRET_KEY', secretKey: newSecretKey });
}, [dispatch]);
```

*Фрагмент 3.13.*

*Фрагмент 3.14.* импортира куката useSelector от библиотеката react-redux, която предоставя достъп до състоянието на Redux. Тя приема функция като аргумент, която извлича данни. В този случай тя получава стойността на целия обект на състояние като параметър и я връща на свойството secretKey, управлявано от хранилището на Redux.

```
import {useSelector} from 'react-redux';

const secretKey = useSelector(state => state.secretKey);
```

*Фрагмент 3.14.*

### **3.4. Липса на действие от страна на потребителя след натискане на бутона за спешност**

Ако след определен интервал от време не бъде избрана опция от менюто за спешни случаи, се позволява на потребителя да изпрати СМС съобщение с информация за текущото си положение и време до номерата по подразбиране, които е въвел в първоначалните настройки.

Първата стъпка от реализацията на тази функционалност е установяването на географските координати на мобилното устройство и записване на часа, в който е натиснат бутона за спешност.

#### **3.4.1 Установяване на текущо време на натискане на бутона за спешност**

Компонентът CurrentTime, изображен на *Фрагмент 3.15.*, ще ни позволи да вземем времето, в което е бил натиснат бутона за спешност.

Функцията `getTime()` ще зададе текущото време на променливата `time`, използвайки `new Date().toLocaleTimeString()`. Това ще създаде нов дата обект, а методът `toLocaleTimeString()` ще форматира часа в зависимост от локалната настройка на браузъра. В края на компонента, текущото време се връща като резултат.

```
const CurrentTime = () => {
  const [time, setTime] = useState(null);

  useEffect(() => {
    const getTime = () => {
      setTime(new Date().toLocaleTimeString());
    };

    getTime();
  }, []);

  return time;
};
```

*Фрагмент 3.15.*

### 3.4.2 Установяване на текущо местонахождение на потребителя

*Фрагмент 3.16.* дефинира функционален компонент, наречен `CurrentLocation`, който връща обект с текущото местоположение на потребителя. Използват се две променливи - географска ширина и дължина. Първоначалните им стойности са зададени на `null` с помощта на `useState`. Куката `useEffect` се използва за асинхронно извличане на тези данни с помощта на модула за местоположение `Location` на Expo. Той включва в себе си метода `getCurrentPositionAsync()`, който приема различни опции. В случая те биват:

- **accuracy (точност)** - това определя желаната точност на данните за местоположението. В този случай той е настроен на `Location.Accuracy.Highest`, което означава, че върнатите данни за трябва да бъдат възможно най-точни;
- **maximumAge (максимална възраст)** - това определя колко стари могат да бъдат данните за местоположението, преди да се считат за остарели. В този случай е зададено на 10 000 милисекунди (10 секунди), което означава, че ако няма налични свежи данни, методът ще върне последните известни такива, които не са по-стари от 10 секунди;

- **timeout (изчакване)** - това определя максималното време (в милисекунди), което е разрешено на метода да върне данните за местоположението. В този случай е зададено на 20 000 милисекунди (20 секунди).

Ключовата дума await се използва, за да се изчака отговор от асинхронния метод getCurrentPositionAsync, преди да се присвои резултата на променливата резултат. Ако операцията е успешна, взетите данни за местоположение се връщат в резултатната променлива. Ако възникне грешка, като това, че местоположението е недостъпно, блокът catch ще бъде изпълнен с обекта за грешка като аргумент. Стойностите за географската широта и дължина се задават с помощта на функциите setLatitude() и setLongitude().

```
import * as Location from 'expo-location';

const CurrentLocation = () => {
  const [latitude, setLatitude] = useState(null);
  const [longitude, setLongitude] = useState(null);

  useEffect(() => {
    async function getLocation (){
      try {
        let result = await Location.getCurrentPositionAsync({
          accuracy: Location.Accuracy.Highest,
          maximumAge: 10000,
          timeout: 20000
        });

        setLatitude(result.coords.latitude);
        setLongitude(result.coords.longitude);
      } catch (error) {
        console.log('Error getting location:', error);
      }
    }

    getLocation();
  }, []);
}

return {latitude, longitude};
}
```

*Фрагмент 3.16.*

### 3.4.3 Отчитане на отминалото време по зададен интервал

Втората стъпка е определянето на интервала, след който ще последват останалите действия и механизът, благодарение на който ще бъде отчитано изминалото време.

```
import {useEffect, useState} from 'react';

const Stopwatch = () => {
    const [end, setEnd] = useState(false);
    const [seconds, setSeconds] = useState(0);

    useEffect(() => {
        const interval = setInterval(() => {
            setSeconds(seconds => seconds + 1);
        }, 1000);

        if (seconds == 20) {
            setEnd(true);
        } else if (seconds > 20) {
            clearInterval(interval);
        }
    });

    return () => clearInterval(interval);
}, [seconds]);

return end;
}
```

*Фрагмент 3.17.*

*Фрагмент 3.17.* реализира поставените условия. Той дефинира компонент, наречен Stopwatch, изпълняващ ролята на хронометър. Той използва куката useEffect, за да актуализира стойността на променливата на състоянието seconds всяка секунда с помощта на setInterval(). Това е вградена функция в JavaScript, която позволява на разработчиците да изпълняват код периодично на определено време. Тя връща уникален идентификатор, който може да се използва за изчистване на интервала по-късно. На setInterval() трябва да се подаде функция, която да се изпълнява периодично и времеви интервал (милисекунди), който да се изчака преди изпълнението ѝ да се повтори отново. В представения код това е setSeconds, която задава стойност на променливата seconds в интервал от 1000 милисекунди (или 1 секунда).

Когато стойността на seconds достигне 20 секунди, се задава стойност на end променливата. Ако тя приеме стойност true и потребителят не е изbral нито една опция, можем директно да преминем към изпращането на СМС съобщение. Ако се случи противното и е избран спешен случай, нейната стойност е без значение.

Когато стойността на seconds надхвърли 20, интервалът се прекратява чрез clearInterval, извикваща се с идентификатора на интервала. useEffect гарантира, че интервалът ще се прекрати.

#### 3.4.4 Оформяне и изпращане на СМС съобщения

*Фрагмент 3.17.* изобразява компонента SendSMS, който се използва за изпращане на СМС съобщения с помощта на библиотеката expo-sms. Необходимо е да изброим телефонните номера, до които искаме то да бъде изпратено. За целта създаваме екземпляр на класа GetDefaultNumbers, който извлича от локалното хранилище нужните ни данни. Той връща списък с телефонните номера, които сме въвели при първоначалните настройки на приложението.

Следващата стъпка е да оформим съдържанието на съобщението, което искаме да изпратим. Функцията createSMSMessage го създава с помощта на подадените ѝ променливи - установените географска ширина и дължина, както и време. Когато съобщението е създадено и записано в message, то може да бъде изпратено до определените получатели. Използва се методът sendSms, който приема телефонен номер като аргумент и методът SMS.sendSMSAsync, за да изпрати СМС съобщение до него. Когато всичко приключи се използва куката useNavigation за достъп до навигационния обект на родителския компонент. Извиква се метода returnToHomeScreen, навигирайки обратно към началния экран.

useEffect се използва за извикване на метода createSMSMessage всеки път, когато променливата на състоянието phoneNumbers се промени. Това гарантира, че СМС съобщенията се изпращат всеки път, когато списъкът с телефонни номера се измени.

```
const SendSMS = (props) => {
  const navigation = useNavigation();
  const [message, setMessage] = useState(null);
  const phoneNumbers = new GetDefaultNumbers();
  const returnToHomeScreen = () => {
    navigation.navigate('BottomTabNavigator');
  };
}
```

```

const sendSms = async (number) => {
    await SMS.sendSMSAsync([number], message);
    returnToHomeScreen();
};

const createSMSMessage = async () =>{
    const lat = props.latitude;
    const long = props.longitude;
    const url = `https://www.google.com/maps/search/?api=1&query=${lat},${long}`;

    const result = `Time: ${props.time}\nLocation:\n${url}`;
    setMessage(result);

    for (const number of phoneNumbers) {
        if (number) {
            await sendSms(number);
        }
    }
}

useEffect(() => {
    createSMSMessage();
}, [phoneNumbers]);
}

```

*Фрагмент 3.17.*

### 3.5 Реализация на телефонно обаждане

На потребителя се предоставя възможност да извърши телефонно обаждане. Но то често не се осъществява директно. Реализацията е представена на *Фрагмент 3.18.*. Това е модул, който експортира функция, наречена makeCall, която приема параметър phoneNumber. Когато се извика, makeCall създава аргументи на обект с две свойства: номер, който е зададен на параметъра phoneNumber, и prompt (подканата), който е зададен на false. Въпреки, че изрично е изискано да не се търси потвърждение за извършване на телефонно обаждане, някои устройства имат софтуерна защита, която е необходимо самият потребител да откаже в настройките на приложението. След това PhoneCall(args) се извиква с обекта args като параметър. Това използва библиотеката react-native-phone-call за иницииране на телефонно обаждане до посочения phoneNumber.

```
import PhoneCall from 'react-native-phone-call';

const makeCall = (phoneNumber) => {
  const args = {
    number: phoneNumber,
    prompt: false,
  };

  PhoneCall(args).catch(console.error);

  return makeCall;
};

export default makeCall;
```

*Фрагмент 3.18.*

### 3.6 Навигиране до избрано болнично заведение

При нужда потребителят може да избере от списък болнично заведение, до което да бъде навигиран - *Фрагмент 3.19.*

```
const navigateToEndPoint = (latitude, longitude) => {
  const url = `https://www.google.com/maps/dir/?api=1&destination=${latitude},${longitude}`;

  Linking.openURL(url);
};
```

*Фрагмент 3.19.*

Кодът представлява JavaScript функция, която приема два параметъра: `latitude` и `longitude`. Тези параметри определят координатите на дестинацията, към която потребителят ще бъде препратен.

Функцията създава URL адрес към уеб страница на Google Maps, като използва `latitude` и `longitude` като координати на дестинацията. Тази уеб страница ще съдържа маршрут за достигане до дестинацията от текущото местоположение на потребителя.

След това, чрез `Linking.openURL(url)` адресът към Google Maps се отваря в браузъра по подразбиране на мобилното устройство. Така потребителят може да види маршрута, да го следва или да използва навигацията на Google Maps, за да стигне до дестинацията.

### 3.7 Прочитане на информация от локален файл с данни

```
import {firstAid} from '../data/FirstAid';

const instructions = firstAid.filter(item => item.case === caseNum);

return (
    ...
    <View style={styles.instructionsContainer}>
        <FlatList
            data={instructions}
            keyExtractor={item => item.id}
            renderItem={({item}) => {
                return <>
                    <View style={styles.secondaryTextContainer}>
                        <Text style={styles.secondaryText}>
                            Стъпки при {item.description}:
                        </Text>
                    </View>

                    <View style={{alignItems:'center'}}>
                        {item.steps.map((step, index) => (
                            <View key={step.text.length}>
                                <TextField
                                    number={index}
                                    text={step.text}
                                />
                            </View>
                        )));
                    </View>
                </>
            }>
        </View>
    ...
);
```

Фрагмент 3.20.

Информацията за видовете здравни заведения, както и инструкциите за първа долекарска помощ, се съхранява в локални файлове. Причината е, че тя не е много голяма и не е динамично променяща се, а е статична. По този начин тя бива достъпвана изключително бързо. Всеки запис има свой уникален идентификатор. В зависимост от избраната опция за спешност се филтрира информацията така, че да останат само записите, отговарящи на събитието.

Във *Фрагмент 3.20.* е изложен начина на филтране, както и този на намаляване на елементите спрямо определен критерий. Използва се същата логика при болничните заведения.

### 3.8 Видове навигация между различните екрани

React Navigation е мощна библиотека, която предоставя удобни инструменти за навигация в приложението, като създаване на различни екрани, превключване между тях и контрол на поведението при взаимодействие на потребителя.

- **Stack Navigator** - това е най-често използваният начин за навигация. Той предоставя стек с екрани, позволяйки на потребителя да се връща назад към предишния от тях. Може да включва и заглавна лента за всеки экран и да изпраща параметри между тях.
- **Tab Navigator** - този вид навигация предоставя табове (tabs) на дъното на екрана, чрез които се извършва превключването между различни функционалности в приложението. Те могат да бъдат персонализирани с икони и текстове, предоставящи бърз достъп до най-използваните части на приложението.
- **Drawer Navigator** - този вид навигация предоставя "чекмедже" от страничната част на екрана, който съдържа навигационни връзки към различни части на приложението. Той може да бъде използван, когато има много екрани и табовете не са подходящи.

За да използваме React Navigation, трябва да инсталираме компонента `NavigationContainer`, който ще ни помогне да осигурим правилната работа на навигацията в нашето приложение. Той е основен и обвива всички други навигационни компоненти, които използваме в приложението.

Основната опция, която можем да посочим в компонента `createStackNavigator`, е `initialRouteName`, която определя кой еcran да бъде показан първи при зареждане на приложението. Притежава и много други, като например това дали да се показва заглавната лента на екрана. Демонстрация на кода във *Фрагмент 3.21.* .

```

const {Navigator, Screen} = createStackNavigator();

const StackNavigator = () => {

  const [initialRoute, setInitialRoute] = useState('AmountOfPhoneNumbersScreen');
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const checkIfUserVisitedBefore = async () => {
      const hasVisitedBefore = await AsyncStorage.getItem('hasVisitedBefore');

      if (hasVisitedBefore) {
        setInitialRoute('BottomTabNavigator');
      } else {
        await AsyncStorage.setItem('hasVisitedBefore', 'true');
      }
    }

    setIsLoading(false);
  };

  checkIfUserVisitedBefore();
}, []);

if (isLoading) {
  return null; // render a loading indicator instead of the Navigator
}

return(
  <NavigationContainer>
    <Navigator
      initialRouteName={initialRoute}
      screenOptions={{headerShown: false}}
    >
      <Screen name='AmountOfPhoneNumbersScreen' component={AmountOfPhoneNumbersScreen}/>
      <Screen name='PhoneNumberEntryScreen' component={PhoneNumberEntryScreen}/>
      <Screen name='LocationPermissionScreen' component={LocationPermissionScreen}/>
      <Screen name='BottomTabNavigator' component={BottomTabNavigator}/>
      <Screen name='EmergencyOptionsScreen' component={EmergencyOptionsScreen}/>
      <Screen name='HospitalListScreen' component={HospitalListScreen}/>
      <Screen name='InstructionsScreen' component={InstructionsScreen}/>
    </Navigator>
  </NavigationContainer>
);
};

```

*Фрагмент 3.21.*

Когато приложението първоначално се зарежда, използваме променливите initialRoute и isLoading, представени на *Фрагмент 3.21.*, за да определим началния еcran, който ще се визуализира. Проверява се дали потребителят е посещавал приложението преди. Това ни дава възможност да управляваме поведение, като го пренасочваме потребителя към определен еcran от приложението.

*Фрагмент 3.22.* показва създаването на долен таб за навигация с две опции - начален еcran и еcran с настройки. За създаването на табовете се използва библиотеката за React Native - "@react-navigation/bottom-tabs". В началото на файла са импортирани необходимите компоненти - Text, View, createBottomTabNavigator и две библиотеки за икони - MaterialCommunityIcons и Ionicons.

Дефинира се основен компонент - BottomTabNavigator, който съдържа две променливи, представляващи имената на двата екрана, след това два компонента - TabBarIcon и TabBarLabel - за създаване на икони и текст за табовете. В BottomTabNavigator се създава Navigator с два Screen компонента - за началния еcran и за екрана с настройки. В опциите на всеки един от тях се задават функции за създаване на иконата и текста на таба, като се използват компонентите TabBarIcon и TabBarLabel.

```
const {Navigator, Screen} = createBottomTabNavigator();

const BottomTabNavigator = () => {
    let Home = 'Начало';
    let Settings = 'Настройки';

    const TabBarIcon = ({name, focused}) => (
        <View style={styles.tabBarIconContainer}>
            {name === Home ? (
                <MaterialCommunityIcons
                    name='home'
                    style={[styles.tabBarIcon, focused && styles.focusedTabBarIcon]}
                />
            ) : (
                <Ionicons
                    name='settings'
                    style={[styles.tabBarIcon, focused && styles.focusedTabBarIcon]}
                />
            )}
        </View>
    );
}

const TabBarLabel = ({name, focused}) => (
```

```

<View style={styles.tabBarLabelContainer}>
  <Text style={[styles.tabBarLabel, focused && styles.focusedTabBarLabel]}>
    {name}
  </Text>
</View>
);

return (
  <View style={styles.container}>
    <Navigator
      initialRouteName={Home}
      screenOptions={{
        headerShown: false,
        tabBarStyle: styles.tabBar,
      }}
    >
      <Screen
        name={Home}
        component={HomeScreen}
        options={() => ({
          tabBarIcon: ({focused}) => <TabBarIcon name={Home} focused={focused} />,
          tabBarLabel: ({focused}) => <TabBarLabel name={Home} focused={focused} />
        })}
      />

      <Screen
        name={Settings}
        component={SettingsScreen}
        options={() => ({
          tabBarIcon: ({focused}) => <TabBarIcon name={Settings} focused={focused} />,
          tabBarLabel: ({focused}) => <TabBarLabel name={Settings} focused={focused} />
        })}
      />
    
```

</Navigator>

</View>

);

}

*Фрагмент 3.22.*

## ЧЕТВЪРТА ГЛАВА

# РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ

### 4.1 Инсталiranе на необходимия софтуер

Преди да се стартира приложението за действие при здравословни проблеми, първоначалният подход е инсталация и настройка на локалната среда на изпълнение. В зависимост от типа на операционната система, се налага следването на различни стъпки. Необходима е инсталацията на:

#### 4.1.1 Node.js

React Native използва Node.js, среда за изпълнение на JavaScript от страна на сървъра извън браузъра. Това е платформа с отворен код. Предлага начин за лесно изграждане на бързи, мащабируеми програми. Дава се достъп до така наречения мениджър на пакети npm (node package manager), който ни позволява да инсталлираме и управляваме модули като зависимости.

Изтеглете node.js от nodejs.org. Уверете се, че това е най-новата LTS (дългосрочна поддръжка) версия. Също така с това изтегляне е включен сървър за разработка, наречен Metro bundler, който предоставя актуализации на живо при отстраняване на грешки.

#### 4.1.2 Android Studio емулатор

За най-добро изживяване трябва да използвате емулатора в Android Studio на компютър с поне следните спецификации:

- 16 GB RAM
- 64-битова операционна система Windows, macOS, Linux или Chrome OS
- 16 GB дисково пространство

Инсталирайте го, следвайки стъпките, изложени в линк в глава “Използвана литература” [12]. След успешна инсталация на Android Studio, последвайте указанията за създаване на виртуално мобилно Android устройство (AVD) [13].

#### **4.1.3 Изтегляне на приложението от Github хранилище и стартирането му върху Android Studio емулатор**

Първата необходима стъпка е изтеглянето на git. Помощен линк е посочен в глава “Използвана литература” [14]. Когато всичко е извършено успешно, преминете към следващата стъпка.

За да изтеглите проекта, е необходимо да имате профил в Github. Заредете виртуалното устройство, на което ще стартирате приложението. Отворете git bash.

Напишете следните команди:

- `cd Desktop` - промяна на текущата директория на Desktop
- `git clone https://github.com/ivona127/Diploma-project` - клониране на проекта
- `cd Diploma-project` - промяна на текущата директория на тази на проекта
- `npm install` - добавяне на Node.js
- `npm run android` - стартиране на приложението

След изпълнението на тези команди, трябва върху экрана на виртуалното устройство да бъде визуализирано приложението за действие при здравословни проблеми.

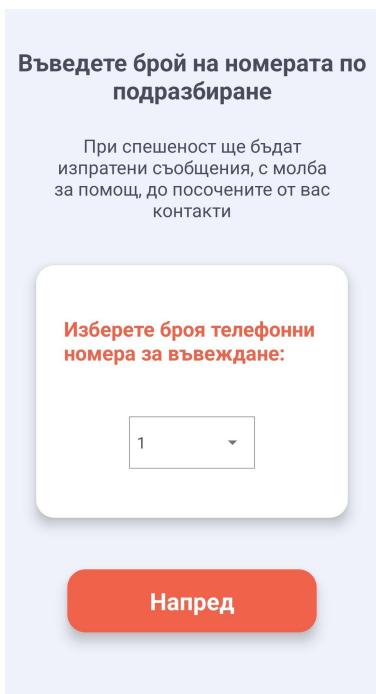
## **4.2 Инструкции за използване на приложението**

Когато приложението се стартира за първи път, потребителят бива въведен сред няколко экрана за настройка.

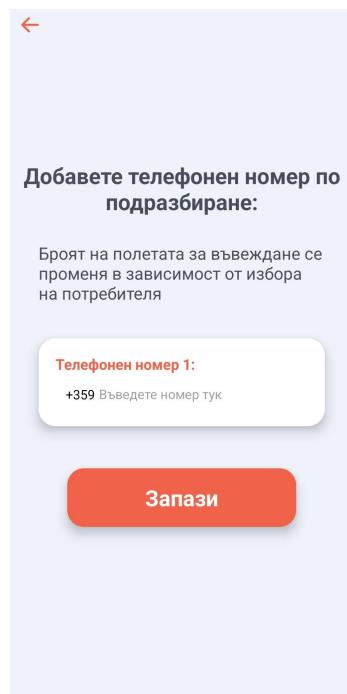
Първата стъпка, през която преминава е изборът какъв броя номера по подразбиране желае да въведе. Минималният брой е един, а максималния три. Не е определена точна бройка, тъй като зависи от предпочтитанието на самия потребител. Екранът, описващ гореспомената настройка, е показан на *Фигура 4.1.*.

Втората стъпка, пред която е изправен потребителя, е въвеждането на тези телефонни номера. Броя на полетата се променя динамично според предходния избор на потребителя.

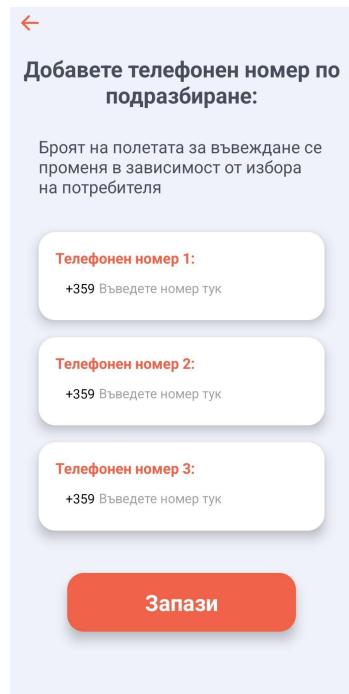
Ако той е преминал към еcran за въвеждане - на *Фигура 4.2.*, *Фигура 4.3.*, но настъпи промяна на мнението му, се позволява да се върне обратно и да избере друг брой.



Фигура 4.1.

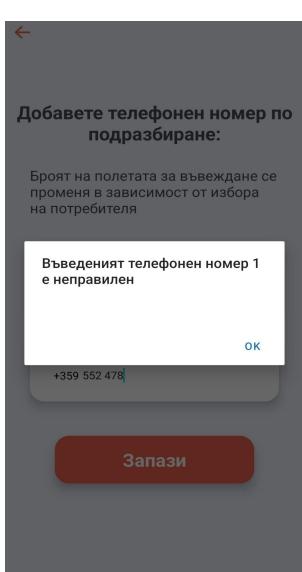


Фигура 4.2.

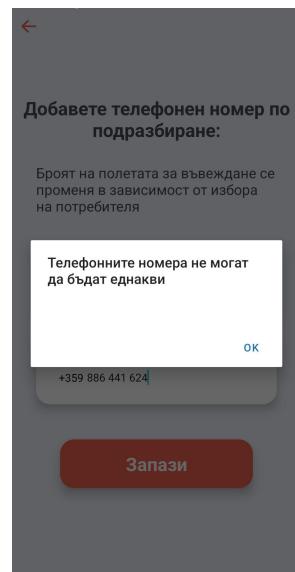


Фигура 4.3.

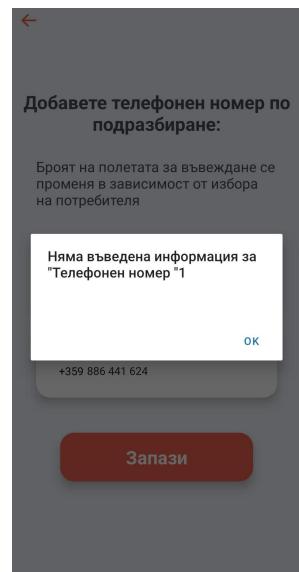
Ако въведената информация не е валидна, ще се изведе съобщение за грешка, уточнявайки, в точно което от полетата е проблемът - *Фигура 4.4.* . При желание да се продължи към следващата стъпка от предварителната настройка, но в една от клетките за въвеждане липсва каквато и да е информация, отново се показва грешката на потребителя *Фигура 4.5.* Не могат да бъдат въвеждани едни и същи номера. Всяка една от броя стойности трябва да бъде различна от останалите - *Фигура 4.6.* .



Фигура 4.4.

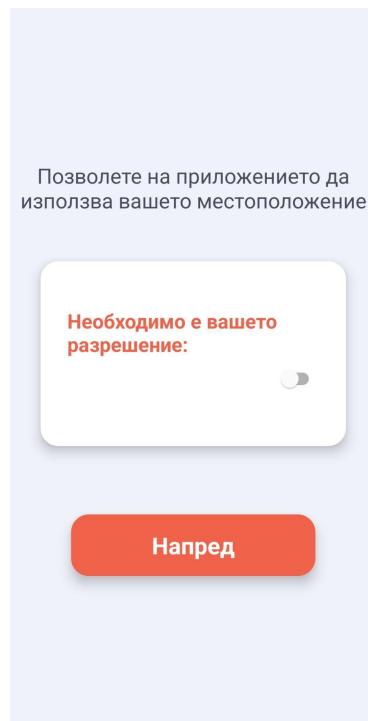


Фигура 4.5.

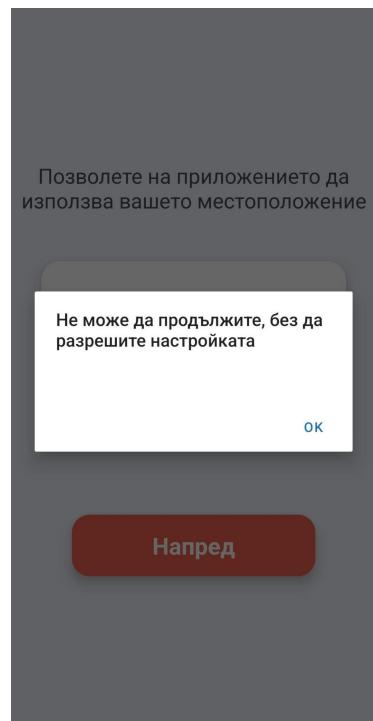


Фигура 4.6.

Последната стъпка от настройките на приложението е свързана с използването на местоположението на устройството. Необходимо е разрешението на потребителя, което позволява то да бъде използване по време на използване на приложението - *Фигура 4.7.* Ако то бъде отказано, ще се изпише съобщение за неуспешна настройка - *Фигура 4.8.*

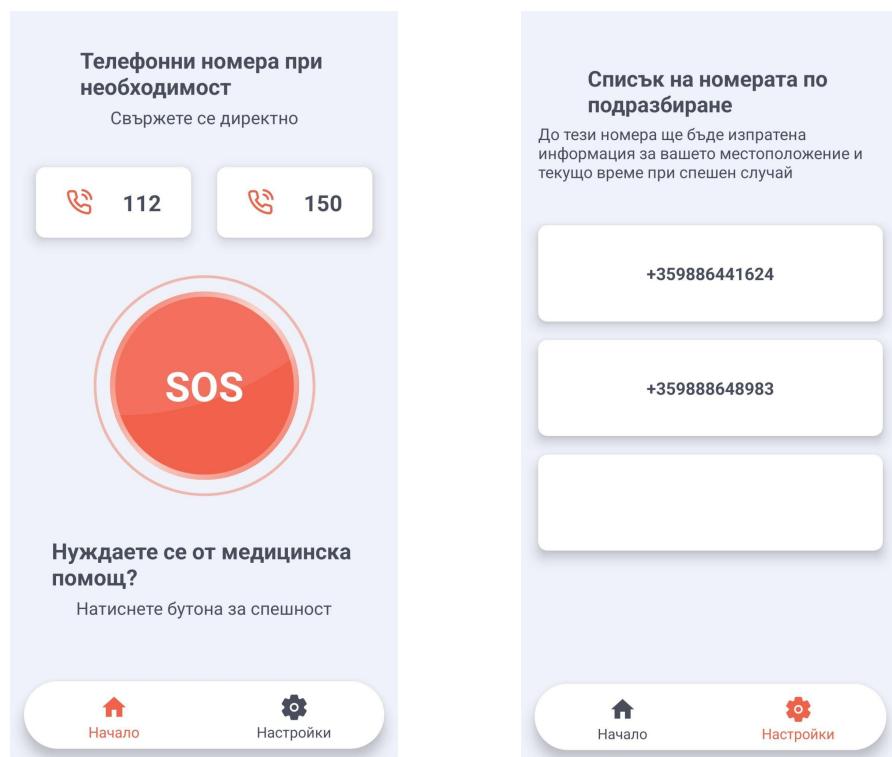


*Фигура 4.7.*



*Фигура 4.8.*

Преминал веднъж през еcranите за настройка, потребителят продължава към основната част на приложението. Тя представлява еcran с бутон за спешни медицински случаи (SOS бутон), както и се предоставя връзка със спешните телефони 112 и 150 - *Фигура 4.9.* . При всяко следващо отваряне на приложението това ще е първото, което потребителя вижда. Можем спокойно да го наречем начален еcran на приложението. В дъното на приложението има меню, с което да се отиде до еcran, наречен Настройки, където ще се визуализират, въведените по време на настройката от потребителя, телефонни номера - *Фигура 4.10.* . Все още те единствено се визуализират, но в бъдещата разработка на приложението, те ще се асоциират с дадено име и ще бъде доставена възможност за директна връзка с тях, както и добавянето на повече на брой от тях.



*Фигура 4.9.*

*Фигура 4.10.*

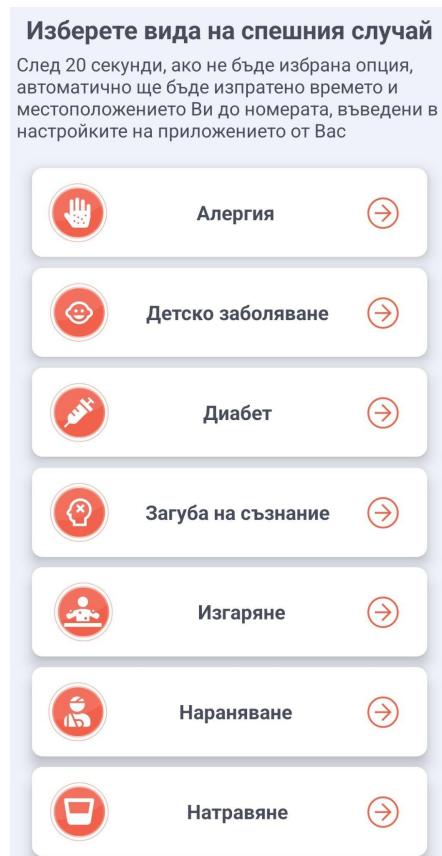
При възникване на спешен случай, когато се натисне бутона за спешност и устройството няма включен GPS локатор, се появява автоматичен прозорец, който позволява на потребителя да го зададе без да излиза от приложението - *Фигура 4.11.*.

След натискане на бутона се появява меню с опции, от които потребителят може да избере типа на събитието. Категориите са общо 10 на брой - алергия, детско заболяване, диабет, загуба на съзнание, изгаряне, нараняване, натравяне, раждане, сърдечни заболявания и ухапване - *Фигура 4.12.*

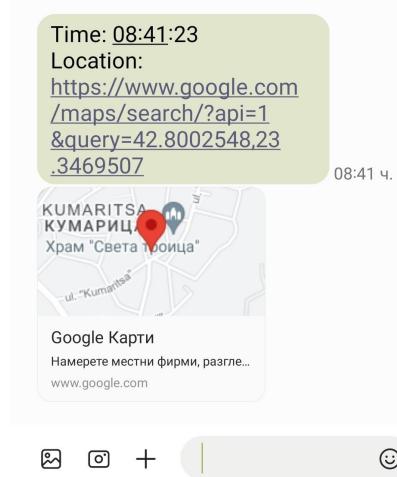
Ако до 20 секунди то не бъде избрано събитие от списъка за спешни случаи, се позволява на потребителя да изпрати СМС съобщение до номерата по подразбиране. То включва неговото местоположение и текущо време - *Фигура 4.13.*. Не зависимо дали бъде изпратено съобщение или не, потребителят се навигира отново до началния екран на приложението.



*Фигура 4.11.*



*Фигура 4.12.*

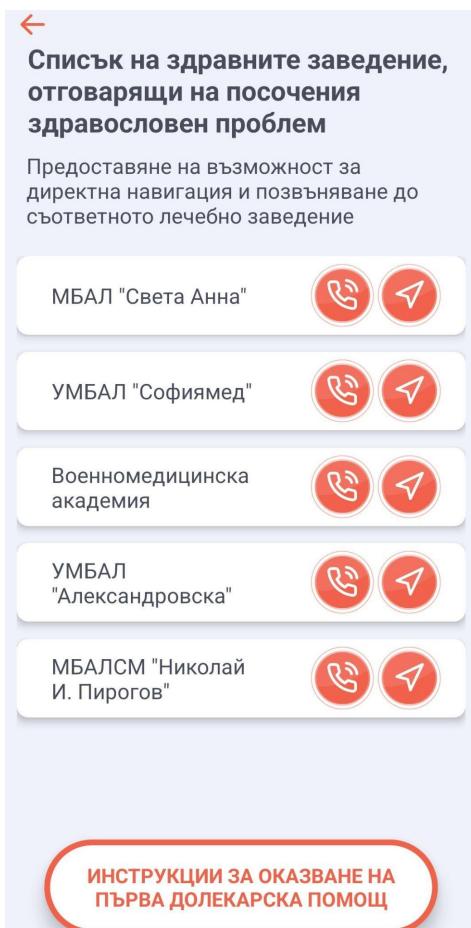


*Фигура 4.13.*

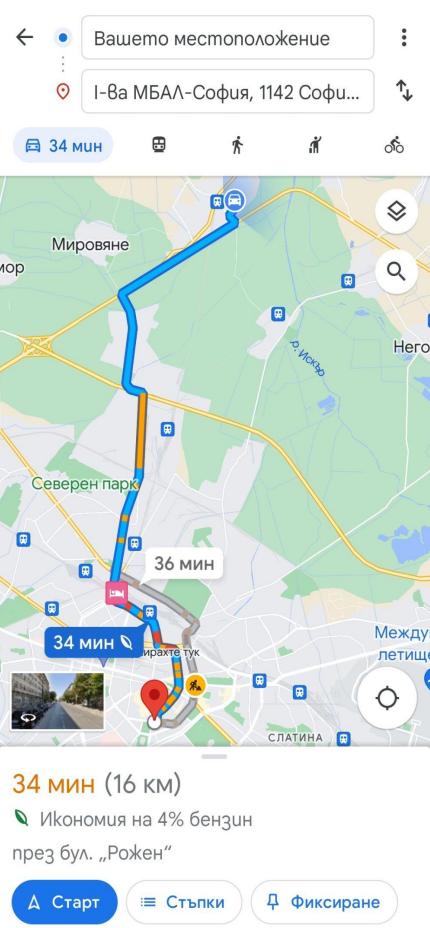
Според опцията, която е избрана от потребителя, се изобразява екран със списък от болнични заведения - *Фигура 4.14.*

Той може да бъде навигиран до всяко едно от тях - *Фигура 4.15.* или да им позвъни. В горния ляв ъгъл се поставя бутон, който връща потребителя в началния еcran на приложението.

В дъното на страницата е предоставен бутон, с натискането на който, се визуализира еcran със инструкции за прилагане на първа долекарска помощ според избраното събитие. Те са обединени от определен медицински проблем, но притежават второстепенно заглавие, което навлиза в конкретика - *Фигура 4.16.* В горния ляв ъгъл се поставя бутон, който връща потребителя към екрана със списъка на болничните заведения.



*Фигура 4.14.*



*Фигура 4.15.*



*Фигура 4.16.*

## **ЗАКЛЮЧЕНИЕ**

Успешно е разработено мобилно приложение за действие при здравословни проблеми. Изпълнени са всички поставени изисквания. Потребителят може първоначално да настрои своето приложение. Предоставят се избор за типа на спешния медицински случай, според който се изброяват здравни заведения. Изпълнена е функционалността, позволяваща задаване на навигация до всяко едно от тях, както и възможност за позвъняване. При нужда от спешна долекарска помощ се предоставят указания за действие в зависимост от медицинския случай. С натискането на разработения бутон за спешност се вземат точните географски координати на мобилното устройството и време. При липса на избор на спешен медицински случай в продължение на определен интервал от време, потребителят получава възможността да уведоми, избрани хора за нуждата си от тяхната помощ.

Бъдещото развитие на проекта включва следните функционалности и подобрения:

- Предоставяне на възможност за редакция на зададените телефонни номера по подразбиране и асоциирането им с име
- Сортиране на болничните заведения спрямо най-близкия до потребителя обект, както и в зависимост от тяхното работно време
- Предоставяне на възможност за изпращане на текстово, видео, аудио съобщение
- Възможност за позвъняване на личен лекар
- Създаване на автоматични тестове на приложението
- Деплоиване на приложението

## **ИЗПОЛЗВАНА ЛИТЕРАТУРА**

[1] Първа помощ БЧК

<https://www.redcross.bg/news/view?nwid=24013>

[2] iHelp с подкрепа от община Габрово

<https://gabrovo.bg/bg/page/1308>

[3] iHelp Bulgaria

<https://ihelp-bulgaria.com/>

[4] SOS Alert

<https://play.google.com/store/apps/details?id=com.rghvsapp.android.sosalert>

[5] Android Studio

<https://developer.android.com/studio>

[6] Android Emulator

<https://www.javatpoint.com/android-emulator>

[7] WebStorm

<https://alfasoft.com/software/development-tools/programming/jetbrains/webstorm/>

[8] TypeScript срещу JavaScript

<https://www.sanity.io/typescript-guide/typescript-vs-javascript>

[9] React Native

<https://www.netguru.com/glossary/react-native>

[10] React Bridge

<https://medium.com/@sannanmalikofficial/how-the-react-native-bridge-works-86171d9a7585>

[11] Flutter срещу React Native

<https://www.simplilearn.com/tutorials/reactjs-tutorial/flutter-vs-react-native>

[12] Инсталация на Android Studio

<https://developer.android.com/studio/install>

[13] Конфигуриране на емулятор

[Създаване и управление на виртуални устройства](#)

[14] Инсталиране на git

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

# СЪДЪРЖАНИЕ

<b>ОТЗИВ НА ДИПЛОМЕН РЪКОВОДИТЕЛ</b>	<b>3</b>
<b>УВОД</b>	<b>4</b>
<b>ПЪРВА ГЛАВА</b>	<b>5</b>
<b>    1.1    Обзор на съществуващите на пазара приложения</b>	<b>5</b>
1.1.1    Първа помощ БЧК	5
1.1.2    iHelp	6
1.1.3    SOS Alert   Emergency & Safety	8
<b>    1.2    Основни принципи, технологии за реализиране на мобилно приложение на операционната система Android и развойни среди</b>	<b>9</b>
1.2.1    Android Studio	10
1.2.2    WebStorm	11
<b>ВТОРА ГЛАВА</b>	<b>12</b>
<b>    2.1    Функционални изисквания към програмния продукт</b>	<b>12</b>
<b>    2.2    Съображения за избор на програмен език</b>	<b>13</b>
2.2.1    JavaScript срещу TypeScript	13
2.2.2    Какво представлява React Native	14
2.2.3    React Native срещу Flutter	15
<b>    2.3.    Локален запис на данни на мобилното устройство</b>	<b>16</b>
<b>ТРЕТА ГЛАВА</b>	<b>18</b>
<b>    3.1.    Основни концепции в едно React Native приложение</b>	<b>18</b>
3.1.1    useState	18
3.1.2    useEffect	18
3.1.3    useNavigation	19
<b>    3.2.    Предоставяне на първоначална настройка</b>	<b>20</b>
3.2.1    Променлив брой полета за въвеждане на телефонни номера според избора на потребителя	20

3.2.2	Запис и извлечане на въведената информация в локалното хранилище	22
3.2.3	Разрешение за използване на текущо местоположение	23
<b>3.3.</b>	<b>Библиотеката Redux</b>	<b>24</b>
<b>3.4.</b>	<b>Липса на действие от страна на потребителя след натискане на бутона за спешност</b>	<b>27</b>
3.4.1	Установяване на текущо време на натискане на бутона за спешност	27
3.4.2	Установяване на текущо местонахождение на потребителя	28
3.4.3	Отчитане на отмinalото време по зададен интервал	30
3.4.4	Оформяне и изпращане на СМС съобщения	31
<b>3.5</b>	<b>Реализация на телефонно обажддане</b>	<b>32</b>
<b>3.6</b>	<b>Навигиране до избрано болнично заведение</b>	<b>33</b>
<b>3.7</b>	<b>Прочитане на информация от локален файл с данни</b>	<b>34</b>
<b>3.8</b>	<b>Видове навигация между различните екрани</b>	<b>35</b>
<b>ЧЕТВЪРТА ГЛАВА</b>		<b>39</b>
<b>4.1</b>	<b>Инсталиране на необходимия софтуер</b>	<b>39</b>
4.1.1	Node.js	39
4.1.2	Android Studio емулатор	39
4.1.3	Изтегляне на приложението от Github хранилище и стартирането му върху Android Studio емулатор	40
<b>4.2</b>	<b>Инструкции за използване на приложението</b>	<b>40</b>
<b>ЗАКЛЮЧЕНИЕ</b>		<b>46</b>
<b>ИЗПОЛЗВАНА ЛИТЕРАТУРА</b>		<b>47</b>
<b>СЪДЪРЖАНИЕ</b>		<b>49</b>