**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Мобилно приложение социална мрежа за готвачи и любители на готвенето – „EverydayChef“

Дипломант: Научен ръководител:

*Мартин Пенев Александър Илиев*

СОФИЯ

2020

Дата на заданието: 15.11.2019 г. Утвърждавам:..............................

Дата на предаване: 15.02.2020 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Мартин Василев Пенев 12A клас

Тема: Мобилно приложение социална мрежа за готвачи и любители на готвенето - “EverydayChef”

1. Изисквания:
   1. Регистрация и логин чрез имейл и чрез фейсбук
   2. Създаване и менажиране на семейства или взимане на съществуващи такива от фейсбук
   3. Списък за пазаруване за всяко семейство
   4. Налични продукти за всяко семейство
   5. Създаване и менажиране на рецепти
   6. Оценяване на рецепти
   7. Любими рецепти на всеки един потребител
   8. Търсачка на рецепти по налични продукти
   9. Нотификации при различни събития
2. Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :...........................................

Ръководител:..........................................

/Александър Илиев/

Директор:................................................

/ доц. д-р инж. Ст. Стефанова /

**УВОД**

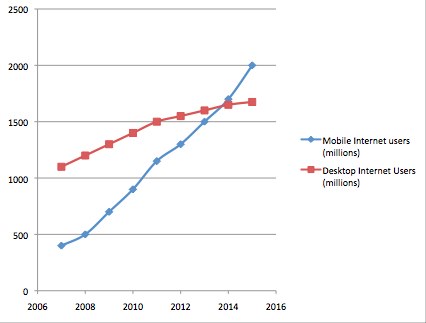
В ежедневието си човек много пъти на ден си задава въпроса: „Какво ще закусвам/обядвам/вечерям днес?“. Понякога липсват продукти за желаното от него ястие, друг път липсват идеи за това какво да си сготви или знания за това как да го направи, а често липсват и трите. Приложението „EverydayChef“ има за цел да отсрани подобни проблеми и да уле сни доколкото е възможно живота в кухнята. Свързвайки готвачи, любители на готвенето и хора без много опит в сферата, софтуерът се стреми да изгради социална мрежа, специално създадена за изкуството на приготвяне на храна. В нея всеки участник може да черпи от опита на другите и да споделя своя собствен, да получи обратна връзка за работата си и да оценява от своя страна тази на другите. Това основнo се изразява в създаване на рецепти и коментирането и оценяването на такива. / Следва дописване /

**ПЪРВА ГЛАВА**

**Проучвателна част**

**Технологии и подобни приложения**

* 1. **Защо мобилно приложение**
     1. **Мобилни устройства**

Днешно време технологиите са навсякъде около нас и съответно приложение като „EverydayChef“ може да бъде разработено за всякакви устройства. Това повдига въпроса защо е избрана точно мобилната среда.

*Фиг. 1.1 - Използваемост на мобилни и настолни устройства за достъп до интернет*

Както се вижда на графиката, бройката на потребителите на мобилни устройства през 2016 е вече по-голяма от тази на хората, използващи настолни такива. Проучвания показват, че само една година след това, през 2017 вече 67% от глобалния трафик идва от джобната технология, което вече ясно изгражда тенденцията и за в бъдеще – използваемостта на мобилните устройства за връзка с интернет расте, за разлика от тази на десктоп компютрите. Много по-удобно е за един човек да използва телефона си, за да свърши каквато работа му е нужна, където и когато му е нужна. Това е главната причина да избера тази среда като основна за моето приложение.

* + 1. **Мобилно приложение или мобилен уеб[1]**

Връзката през мобилното устройство до интернет се осъществява чрез мобилно приложение или чрез мобилен уеб. Ако компании с по-стабилни финанси могат да си позволят разработката и на двете за техния продукт, то в повечето случаи, особено в началото на реализацията на проект, се налага да се избира една от двете опции. Изборът обикновено се основава на разликите в цената на изпълнение, целта на продукта и клиентите, за които е предназначен. Въпреки това изследвания показват, че хората предпочитат мобилните приложения пред мобилните уеб сайтове. Ето защо:

* По-добра персонализация

Когато се очаква продуктът да се използва ежедневно, мобилното приложение е най-добрият вариант поради възможнстта за настройването му по вкус на потребителя и поддръжката на важна информация под ръка.

* Push notifications (Директни съобщения)

Вградена функция на мобилните приложения е изпращането на директни съобщения към потребителя при различни събития. Въпреки че такива съобщения могат да се изпращат и от уеб приложенията, тези от мобилните приложения предоставят доста по-богато оформление, както и интерактивност на съобщениетo. Поради това може да се каже, че в този аспект мобилните приложения превъзхождат тези в уеб.

* Лесен достъп до устройството и неговите функционалности

През мобилното приложение много лесно се използват камерата, контактите, историята на обажданията и подобни функции на телефона. По този начин може да се осигури много по-приятна и улеснена работа с продукта за потребителя, както и се намалява времето, за което той би свършил нещо, изискващо тези функции, чрез мобилния уеб.

* Няма нужда от интернет връзка

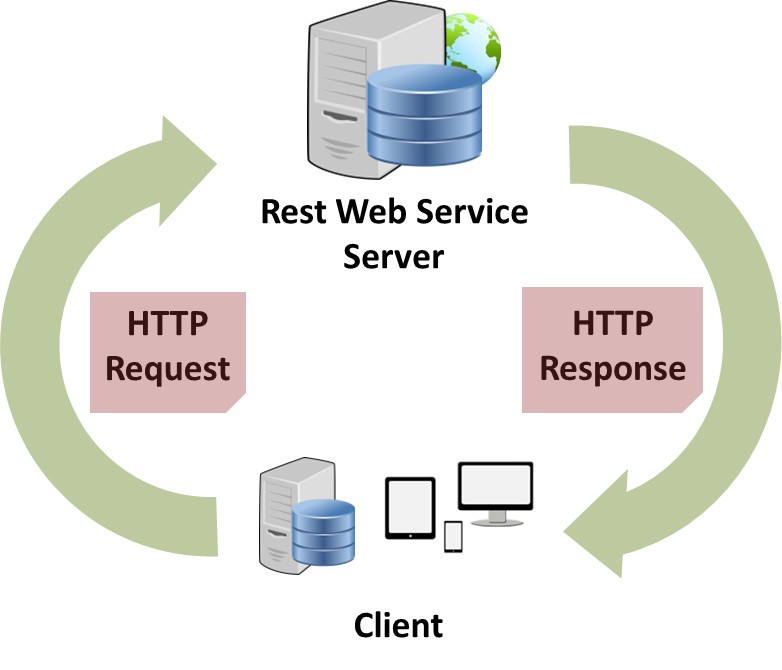
Много полезно предимство на мобилните приложения е това, че могат да се ползват и при липса на обхват. Това е, защото понеже приложението е инсталирано на самото устройстройство то запазва локално последното заредено съдържание и винаги показва него. При наличие на интернет връзка това съдържание се обновява.

* По-висока скорост

Едно добре направено приложение може да процедира много по-бързо от една уеб страница. Една от причините е именно локално запазената (кешираната) информация в приложението.

* 1. **Използвани технологии**
     1. **REST**

„EverydayChef“ е имплементирано на базата на REST. REST[1] (Representational state transfer) представлява софтуерна архитектура, предоставяща дефинирани стандарти за връзката между компютърните устройства в мрежата, за по-лесна комуникация помежду им. Основава се на HTTP – мрежов протокол, който е в основата на обмена на данни в глобалната мрежа, която познаваме днес.

Характерно за създадените по този начин системи (наричани RESTful системи) е, че при тях са разделени понятията клиент и сървър, както и на сървъра не му трябва да знае какво е състоянието на клиента във всеки един момент от време и обратното.

*Фиг. 1.2 - Визуализация на REST архитектура*

* + - * **Разделение на клиент от съръвър**

При REST архитектурата клиентът и сървърът се изграждат напълно отделно без едното да се интересува от другото. Това значи, че могат да бъдат променяни по всяко време и това да не засяга другата страна.

По този начин интерфейсът по-лесно може да бъде интегриран на друга платформа и има много възможности за подобрение.

Чрез REST архитектурния стил различни клиенти могат да се закачат за същите пътища на интерфейса, да извършват еднакви операции, както и да получават еднкави данни.

* + - * **Не се знае текущото състояние на отсрещната страна**

При всички приложения, следващи REST, на сървърът не му трябва да знае текущото състояние на клиента и обратното. По този начин и двете страни разбират всяко едно съобщение помежду си без да се налага да виждат предишните. Това се осъществява чрез използването на ресурси вместо на команди. Това са обекти, документи или просто информация, която трябва да бъде запазена или обработена по някакъв начин и/или изпратена към друго устройство.

Това спомага за ефективността и скоростта на RESTful приложенията, както и за тяхното лесно разрастване, тъй като отделните компоненти могат да бъдат променяни и преизползвани без това да се отразява на системата като цяло.

* + - * **Комуникация между клиент и сървър**

В REST архитектурата клиентите изпращат заявки (requests), за да получат или да променят някакви ресурси, а сървърите връщат отговори (responses) на тези заявки. Цялата комуникация се извършва във вид на JSON(JavaScript Object Notation) – формат за обмен на данни, който е независим от езика, лесен за разбиране от хора и машини и пестелив откъм размер.

* **Създаване на заявки**

REST изисква клиента да изпрати заявка, за да получи или да редактира информация на сървъра. Заявката обикновено се състои от:

* HTTP Глагол – заявява каква операция трябва да се извърши
* HTTP header – позволява на клиента да даде допълнителна инфорамция за заявката
* Път (path) към ресурс
* Тяло на заявката (само за някой глаголи) – не е задължително
  + **Глаголи в HTTP**

Най-основните глаголи които се срещат в HTTP са:

* GET – получаване на специфичен ресурс (чрез неговото id) или на множество ресурси
* POST – създаване на нов ресурс
* PUT – редакция във вече съществуващ ресурс
* DELETE – изтриване на ресурс

Същестуват още глаголи, но изброените са основните и необходимите за създаването на RESTful приложение.

* + **Пътища (Path)**

Всяка заявка трябва да има път към ресурс, който трябва да се върне или промени по указан начин. В RESTful приложенията пътищата са така направени, че да е ясно накъде сочат.

По конвенция, първата част трябва да бъде множественото число на ресурса. Това прави вложените пътища лесни за четени и разбиране. За път като *somesitedomain.com/books/12/pages/3* ясно се разбира, че ще се извършва някаква операция с страницата с *id = 3* на книгата с *id = 12.* Пътищата трябва да съдържат толкова информация, колкото е необходима за да се локализира ресурсът. За операция *POST,* например, няма нужда да се записва *id*, тъй като сървърът автоматично ще сложи такова на новия създаден ресурс.

* **Изпращане на отговори**

Както вече се спомена, в зависимост от получената от клиента заявка, сървърът връща отговор, в който се съдържа информация за това как е била обработена заявката, както и желаните данни ако има такива, в какъв формат са те и други.

* + **Кодове на състоянието (Status codes)**

|  |  |
| --- | --- |
| 200 (OK) | Успешно изпълнена HTTP завка. |
| 201 (CREATED) | Успешно изпълнена HTTP заявка за съдаване на ресурси. |
| 204 (NO CONTENT) | Успешно изпълнена HTTP заявка, при която няма нищо в тялото на отговора. |
| 400 (BAD REQUEST) | HTTP заявката не може да бъде обработена поради грешка в синтаксиса, твърде голям размер или друг проблем от страната на клиента |
| 403 (FORBIDDEN) | Клиентът няма право да достъпи този ресурс |
| 404 (NOT FOUND) | Ресурсът не може да бъде намерен. Възможно е да е изтрит или преместен. |
| 500 (INTERNAL SERVER ERROR) | Грешка от страна на сървъра. |

За да се извести клиента за това дали поисканата от него операция е извършена успешно или не, в отговора на сървъра се съдържа т. нар. *status code* (на български: „код на състояниетo“). Такива кодове има много на брой, но най-известните от тях са:

*Табл. 1.1 - Често срещани статус кодове*

За всеки HTTP глагол има очаквани статус кодове, които трябва да бъдат върнати от сървъра при успешно изпълнение:

* GET — връща 200 (OK)
* POST — връща 201 (CREATED)
* PUT — връща 200 (OK)
* DELETE — връща 204 (NO CONTENT)

Ако при обработка на HTTP заявката възникне грешка, сървърът трябва да върне най-специфичния възможен статус код за грешка, който да отговаря на изникналия проблем.

* + 1. **Android studio – frontend на приложението**

Частта от приложението, която потребителя ще вижда и с която ще работи(frontend), е писана чрез Android Studio [3]. Това е официалното IDE(integrated development environment - среда за разработка), специализирано за операционната система андроид. Излиза за първи път на 16.05.2013г., а първата стабилна версия е обявена декември 2014 година.

Поддържа се както новоизлезлият и обявен за предпочитан от Google език Kotlin, така и старите основни езици като Java и C++. Предоставени са множество функционалности, които улесняват разработката на приложения и увеличават удобството на използване като:

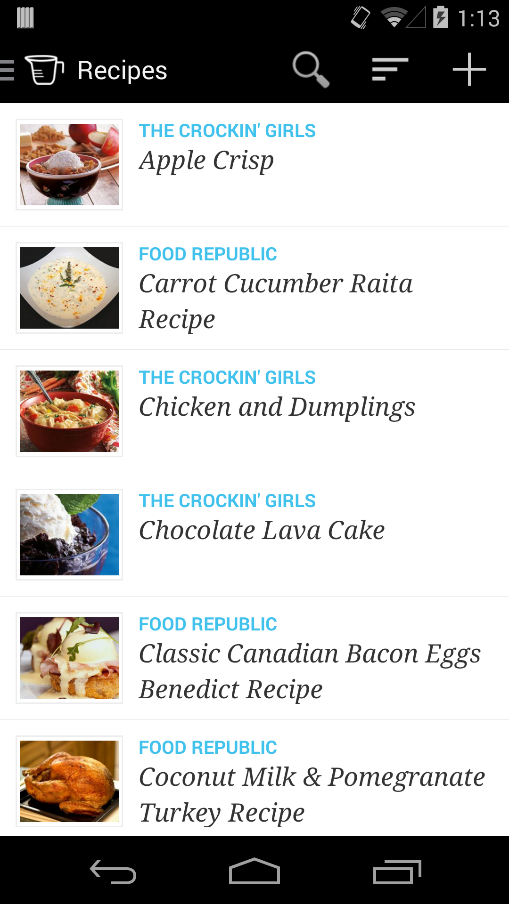
* Gradle build
* Специфични за андроид рефактори и quick fix опции
* Lint средства за засичане на различни проблеми
* ProGuard интеграция
* Темплейт-базирана обработка на дизайна и компонентите на приложенията
* Среда за обработка на дизайна
* Виртуален емулатор
  + 1. **Spring boot – backend на приложението**

Другата част от приложението – сървъра(backend), е написана чрез Spring Boot. Това e open-source micro framework, поддържан от компания на име Pivotal. Разработен е върху framework на име Spring. Чрез него се осигурява автоматична конфиргурация на Spring проект, с което се спестява много време и усилия на програмистите.

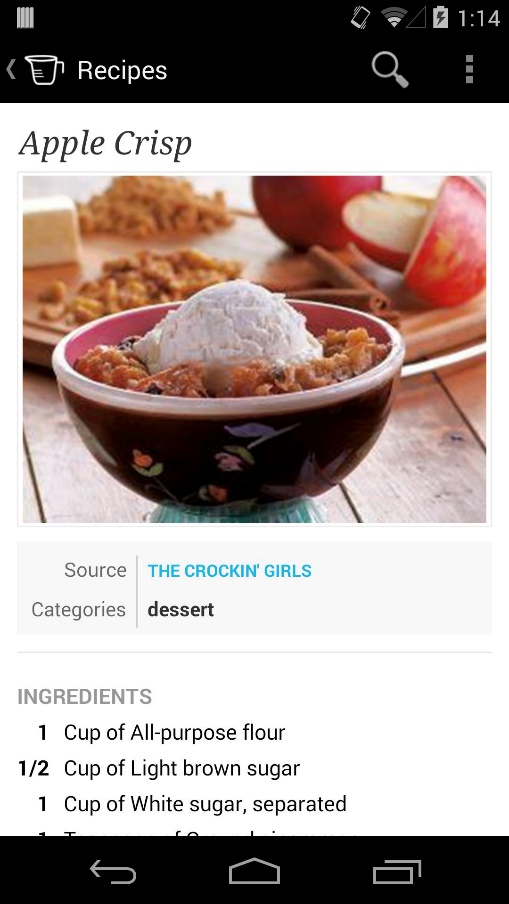
Spring излиза през 2002 година и много бързо се утвърждава като доминиращ начин, по които се пише Java код. Разработен е open-source (целият код е достъпен) и има за цел да направи писането на сървърната част на Java по-лесен и удобен. През 2012 година започват дискусии по начин за подобрение на framework-a. 06.07.2013 Фил Уеб пише, че вместо да се променят вече съществуващия Spring, са започнали да работят по нов проект, наречен Spring Boot, който да оптимизира стария. По тази причина може да се каже, че Spring Boot излиза за да се направи разработката на уеб приложения с Spring по-достъпна и разбираема.

* 1. **Подобни приложения**

Приложения, които да са от полза на човек при ежедневните му проблеми съвсем не са рядкост. Такива има много и то най-различни. Честата нужда от помощ при готвене вече е зародила различни идеи в главите на различни хора за това как може така разпространената технология да помогне в такива ситуации. Ето защо ще разгледам някои приемери за приложения, които са близки до моето.

* + 1. **Pepperplate**

Това приложение е много добър пример за мобилен помощник в кухнята, какъвто представлява и “EverydayChef”. Потребителят може да създаде свои рецепти, да ги организира в менюта, които в последствие да включи в хранителния си план за седмицата или за месеца. При липса на желаните за съответната рецепта продукти той може да ги сложи в списъка с нужни покупки, за да е наясно какво му е необходимо при следващото му пазаруване.

*Фиг. 1.3 – “Pepperplate“ - страница за рецепти*

Предимства:

* Интуитивен дизайн
* Организация в менюта
* Import на рецепти

Недостатъци:

* Нужна е подготовка от страна на потребителя, за да използва приложението
* Липса на свързаност с други хора
* Работата с приложението отнема много време

*Фиг. 1.4 – “Pepperplate“ - страница с детайли за една рецепта*

**ВТОРА ЧАСТ**

***Изисквания***

***Структура на базата и описание на алгоритъма***

* 1. **Изисквания към програмния продукт**

Приложението ‘EverydayChef’ трябва да поддържа:

* Регистрация и логин чрез имейл и чрез фейсбук
* Създаване и менажиране на семейства или взимане на съществуващи такива от фейсбук
* Списък за пазаруване за всяко семейство
* Налични продукти за всяко семейство
* Създаване и менажиране на рецепти
* Оценяване на рецепти
* Любими рецепти на всеки един потребител
* Търсачка на рецепти по налични продукти
* Нотификации при различни събития
  1. **Структура на базата данни**
  2. **Описание на алгоритъма**
     1. **Логин и регистрация**

При влизане в приложението се показва страница, на която потребителят трябва да избере една от опциите за логин (чрез gmail или чрез facebook) или съответно да се регистира, ако все още няма акаунт за EverydayChef. – LoginActivity

След успешен логин се извлича информацията за този акаунт чрез заявки до сървъра и в различните страници (фрагменти) на основната част от приложението (MainActivity) се показва семейството, към което принадлежи, харесани и създадени рецепти, както и списъците с продукти за пазаруване.

* + 1. **Семейство**

Всеки потребител на EverydayChef може да участва в едно семейство. Участниците в семейството споделят общи налични продукти, харесани рецепти, списъци с нужни покупки и списък с рецепти за скоро готвене.

Чрез отварящото се отдясно меню (NavigationDrawer) потребителят може да избере страницата за профил (ProfileFragment). В нея той може да редактира информацията си, както и да управлява семейството, към което принадлежи в случай, че то е създадено от него. Ако това семейство е създадено от някой друг, потребителят може само да преглежда информацията свързана с него или да го напусне.

* + 1. **Рецепти**

Могат да се създават рецепти от всеки потребител, да се променят и премахват чрез съответните заявки към бекенда. Могат да се коментират, харесват и да се добавят в списък с рецепти за скоро готвене. Рецептите съдържат нужните за тях продукти.

От NavigationDrawer потребителят може да избере страницата за профил, където да види собствените си рецепти, или страницата за готвене (CookFragment), откъдето може да намира други рецепти според това какви продукти има или това какво е търсил в търсачката. При влзиането в страницата за специфична рецепта (RecipeFragment) се извличат от бекенда нужните продукти за нея, описанието на приготвянето и, снимката на рецептата, ако има такава, и други. В края на страницата има бутон Cook, който залага рецептата в списъка за готвене на семейството, както и добавя нейните съставки в списък за покупки в случай, че семейството няма тези продукти в наличност.

* + 1. **Списъци с нужни покупки**

Всяко семейство има възможност да си създава списъци с продукти, които са споделени за всеки един член на семейството.

Отново от NavigationDrawer потребителят може да достъпи страницата със списъците (ShoppingListsFragment). На нея се показват всички различни списъци, които принадлежат на съответното семейсто. Отдолу на страницата има бутон за създаване на нов списък, при натискането на който потребителят въвежда име на списъка, след което е препратен към страницата на единичен списък (ShoppingListFragment). В нея могат да се добавят продукти за купуване както ръчно - чрез бутона на страницата, така и динамично чрез добавяне на рецепта в списъка за готвене на семейството. Отбелязаните продукти за купени се добавят в наличните продукти на семейството.

* + 1. **Търсачка на рецепти**

**/Това ще е забавно, ама хубаво е да го направя преди да почна да лафя за него.../**

* 1. **Аргументация за използвани технологии** 
     1. **Android Studio**

Освен Android Studio (АS), хората разработващи приложения за Android света познават и друго популярно, но по-старо IDE – Eclipse. Причините, поради които моят избор е това на Google (AS), са няколко. На първо място, продуктът предлага вградена Gradle build система – удобен помощник в управлението на начина, по който приложението се сглобява (build). Освен това интерфейсът е много интуитивен и лесно се използват отделните компоненти. За разлика от Eclipse, AS предлага и графичен интерфейс, чрез който процесът на разработка и стилизация на приложения е много улесенен. Друга важна причина е, че Android Studio е създадено специално за Android, докато Eclipse се използва за всяка платформа и всякакъв език. Студиото предлага много по-стабилно представяне от своя опонент, както и за работа по желания начин се изисква значително по-малко оперативна памет и процесорна скорост. И двете IDE-та се славят с оптимална code auto completion функционалност – автоматични предложения за допълване или поправяне на грешка в кода.

Всички изброени причини са в основната на моят избор да разработвам EverydayChef на Android Studio. Това също са и причините за това, че от неговото обявяване през 2013 година IDE-то е най-използваният продукт за разработка на Android приложения.

* + 1. **Spring boot**

Що се отнася до бекенд технологиите, въпросът какво да бъде използвано беше малко по-труден заради по-големия брой варианти като NodeJS - Express, RubyOnRails, Laravel, Flask и много други. Аз обаче се спрях на Spring Boot по следните причини:

* Разработван от много време – Spring Boot е базиран на Spring Framework, който сам по себе си е разработван над 15 години като алтернатива на JEE стакът.
* Стабилен – Основните модули на Spring екосистемите са стабилни за дълго време и повечето промени лесно могат да бъдат отменени
* JVM - Spring Boot е базиран на Виртуалната машина на Java, което осигурява по-голяма защита и независимост от използваната платформа.
* Приятен за разработка – Spring Boot предоставя доста удобства за да улесни разработката, с което привлича доста софтуеристи.

**ТРЕТА ЧАСТ**

***Същинска част***

***Начин на разрабока на алгоритъма***

* 1. **Част приложение – фронтенд**
     1. **Логин и регистрация**
  2. **Част сървър - бекенд**

**ИЗТОЧНИЦИ**

1. Приложение или уеб - <https://www.whizsolutions.co.uk/7-reasons-mobile-apps-better-website/>
2. REST - <https://www.codecademy.com/articles/what-is-rest>
3. “Android Studio” - <http://developer.android.com/tools/studio/index.html>
4. „Spring Boot“ - <https://dzone.com/articles/history-of-spring-framework-spring-boot-framework>