1. **Тема:** 3DPernikGuide

2. **Автори:**

Алекс Василев Милчов, ЕГН: 0246043829, Телефон: 0893762009; гр. Перник, ул. Отец Паисий №61; ПГ по икономика – гр. Перник, XII клас

Давид Димитров Иванов, ЕГН: 0250313781, Телефон: 0896239867; гр. Перник, ул. Струма №19; ПГ по икономика – гр. Перник, XII клас

3. **Ръководител:** Людмил Велинов Любомиров – Старши учител по теоретично обучение в ПГ по икономика – гр. Перник, тел.: 0895481106, e-mail: lusivelinov@abv.bg

4. **Резюме:** Проектът представлява една виртуална разходка из най-известните забележителности на град Перник. В нея се включват 3D модели на находки и исторически артефакти, които могат да бъдат разгледани и да се прочете повече информация за тях с помощта на сканиране на QR код или засичане на геолокацията на потребителя.

4.1 **Цели:** Приложението цели да се покажат най-известните забележителности на град Перник, чрез едни от най-модерните и интерактивни технологии на двадесет и първи век, а именно - 3D моделите и тяхната визуализация.

4.2 **Ниво на сложност:** Проектът е разработван в продължение на 5 месеца.

Екип:

1. Алекс Василев Милчов – разработка, тестване, конфигуриране и управление на приложението.

2. Давид Димитров Иванов – разработка, тестване, конфигуриране и управление на сървърната част и уебсайт.

4.3. **Основни етапи в реализирането на проекта:**

1. Установяване на тема за проекта.

2. Сключване договор с историческия музей.

3. Набиране на необходимите ни материали.

4. Проучване на забележителностите, артефактите и находките.

5. Изграждане на функционален план.

6. Избор на технологии спрямо изградения функционален план.

7. Изграждане на логически план спрямо избраните технологии.

8. Разработка и тестване.

9. Публикация.

4.4 **Езици и технологии:**

- езици:

PHP - PHP е един от най-широко използваните програмни езици за създаване и реализиране на уеб услуги. Поради отличната поддръжка на редица бази данни като MySQL, Oracle и PostgreSQL, PHP много често се използва съвместно с бази данни. В добавка, вградените над 800 функции и наличието на отделни библиотеки с фукнции правят PHP език с широки възможности.

Java - Java или Джава е обектно-ориентиран език за програмиране. Кодът, написан на Java не се компилира до машинен код за определен процесор, а до специфичен за езика код, наречен байт код. Поради това за изпълнението на програма, написана на Java е необходима т. нар. Виртуална машина (JVM - Java Virtual Machine).

JSON - JSON (JavaScript Object Notation) е опростен формат за обмяна на данни, удобен за работа както за хората, така и за компютрите. Той е базиран на едно подмножество на езика за програмиране JavaScript. JSON има текстов формат, напълно независим от реализацията на езика, но използва конвенции, които са познати на програмистите на C-подобни езици, включително C, C++, C#, Java, JavaScript, Perl, Python, и много други. Тези свойства правят JSON идеален език за обмяна на данни.

XML - XML представлява език за маркиране. Предназначен е да описва данни в текстов вид, по един неутрален по отношение на платформата и операционната система начин. Езикът използва тагове, които силно приличат на HTML таговете. Един XML документ представлява един текстов файл, в който се съдържат различни XML декларации и тагове, както и текст.

SQL - SQL е структурен език за заявки. Абревиатура от Structured Query Language. С негова помощ можем лесно да правим запитвания и заявки към релационни бази данни. Това е език с две основни функции - с него можем да дефинираме структури от данни и също така можем да обработваме данните, които искаме да запишем в базата дани и/или вече са записани.

- технологии:

Android SDK - Всеки път, когато Google пусне нова версия на Android, се издава и съответната SDK. За да могат да пишат програми с най-новите функции, разработчиците трябва да изтеглят и инсталират SDK на всяка версия за конкретния телефон. Платформите за разработка, които са съвместими с SDK, включват операционни системи като Windows (XP или по-нова версия), Linux (всяка скорошна дистрибуция на Linux) и Mac OS X (10.4.9 или по-нова версия). Компонентите на Android SDK могат да бъдат изтеглени отделно. Въпреки, че SDK може да се използва за писане на Android програми в командния ред, най-разпространеният метод е чрез използване на интегрирана среда за разработка (IDE). Препоръчителният IDE е Android Studio. Въпреки това, други IDE, като NetBeans или Eclipse, също ще работят. Повечето от тези IDE предоставят графичен интерфейс, който позволява на разработчиците да изпълняват задачи за разработка по-бързо. Тъй като приложенията за Android са написани на Java код, потребителят трябва да има инсталиран Java Development Kit (JDK).

Laravel - Laravel е PHP фреймуърк (framework) с отворен код, базиран на шаблона MVC (model–view–controller), който дава възможност за разделяне на бизнес логиката от графичния интерфейс. Архитектурата разделя изходния код на три отделни части – модел, изглед и контролер, което позволява лесна модификация и структуриране на кода.

Eloquent – Eloquent е ORM, който е включен в Laravel. Чрез него SQL-а става доста по-организиран, но същевременно и доста по-прост. Всяка таблица от базата данни има съответния "Модел", който се използва за взаимодействие с тази таблица. По този начин релациите стават по-качествени и по-лесни за разбиране.

MySQL - MySQL е релационна база данни. Използва SQL (Structured Query Language) - най-популярния език за добавяне, достъпване и обработване на информация в база от данни. Базата е с отворен код (open source), може да бъде свалена от Уеб и използвана от всеки според неговите нужди, без да има нужда от лицензиране. Предимството на MySQL е, че специфичната информация може да бъде съхранявана в отделни, много на брой малки таблици с установени между тях се връзки (релации), вместо в една единствена огромна таблица. С това много се улеснява и ускорява обработката на информацията.

RxJava - RxJava е реализация на Java VM на ReactiveX (Реактивни разширения): библиотека за съставяне на асинхронни и базирани на събития програми чрез използване на наблюдавани (Observables) последователности.

Android Jetpack - Android Jetpack е набор от библиотеки, инструменти и насоки, които да помогнат на разработчиците да пишат по-лесно висококачествени приложения. Тези компоненти ви помагат да следвате най-добрите практики, освобождават ви от писане на кодов панел и опростяват сложни задачи, така че да можете да се съсредоточите върху кода, който ви интересува.

Retrofit - Retrofit е REST клиент за Java и Android. Той прави сравнително лесно извличането и качването на JSON (или други структурирани данни) чрез базирана на REST уебсервиз. В Retrofit всеки има право да избере кой конвертор да използва за сериализиране на данни. Обикновено за JSON използвате GSON, но можете да добавите персонализирани конвертори за обработка на XML. Retrofit използва библиотеката OkHttp за HTTP заявки.

Room - Room е ORM, библиотека за релационно конфигуриране на обекти. С други думи, Room ще конфигурира обектите на нашата база данни към Java обекти. Room предоставя абстракционен слой над SQLite, за да позволи плавен достъп до базата данни, като в същото време използва пълната мощност на SQLite.

SQLite - SQLite е компактна библиотека за релационни бази данни, подходяща както за по-малки уеб приложения с опростен интерфейс, така и за реализация на функционалности от големи проекти, целящи по-високо бързодействие и гъвкавост на приложенията. Поддържа езика SQL и транзакционния модел за манипулация на данните, които са неизменна част от уеб приложенията, използващи бази данни. SQLite излиза извън модела клиент-сървър и не се нуждае от осъществяване на връзка със сървър за бази данни за нейното функциониране. Програми, които използват SQLite база данни, могат да осъществяват достъп до базата, без да стартират отделен RDBMS процес. Това е постигнато чрез възможността на библиотеката за писане и четене на база данни директно от файлове върху диска. Цялостна SQL база, съдържаща таблици, индекси, тригери и изгледи се представя от един самостоятелен файл върху диска. Файловия формат на базата данни е съвместим с множество платформи, включително Андроид. Файловия формат може свободно да бъде копиран и изпълняван между различни операционни системи и архитектури.

View Binding – View Binding е функция, която ви позволява по-лесно да пишете код, който взаимодейства с изгледите (views). След като обвързването на изглед (view) е активирано в модул, той генерира инстанция на всеки файл на XML оформление, присъстващ в

този модул. Инстанцията на обвързващия клас съдържа директни препратки към всички изгледи, които имат идентификатор в съответното оформление.

ZXing – ZXing е многоформатна 1D/2D библиотека за обработка на изображения с баркод (Barcode, QR Code, MaxiCode, Data Matrix…) с отворен код, реализирана в Java, с портове към други езици.

MPAndroidChart – MPAndroidChart е мощна библиотека за създаване и конфигуриране на диаграми за платформата Андроид. Могат да се създадат диаграми от различни видове като: Pie Chart, Bar Chart, Linear Chart и др.

Glide – Glide е бърза и ефективна рамка за управление на отворен код и зареждане на изображения за Android, която обхваща функционалности като - декодиране на медии, кеширане на изображения в памет, обединяване на ресурси в прост и лесен за използване интерфейс.

OpenGL – OpenGL (**Open G**raphics **L**ibrary) е мощен приложно-програмен интерфейс (API) за реализиране на лесно преносими графични приложения. Създаден през 1992 г., OpenGL бързо става един от най-популярните програмни интерфейси за реализиране на 2D и 3D графика. За това допринасят широката му достъпност, съвместимостта му с различни операционни системи и с различни компютърни платформи. Тази библиотека е подходяща за приложения изискващи високо качество на изображението, комбинирано с добра производителност, за да бъде възможно генерирането му в реално време.

4.4. **Главни проблеми при разработка:**

- сървър:

1. Инсталацията и конфигурирането на Laravel върху shared hosting (споделен хостинг). Качването, инсталацията и конфигурирането на Laravel беше трудно за нас, тъй като за първи път качваме framework на хостинг. Първо трябваше да проучим дали Laravel се поддържа на нашия хостинг. След като разбрахме, че се поддържа, инсталирахме Laravel през cmd с помощта на Composer на нашата машина. След това го инсталирахме на хостинга. Решението на проблема - прехвърлихме всички файлове, с които Laravel идва чрез програмата FileZilla върху хостинга. Трябваше да конфигурираме базата данни, така че да може тя да работи с Laravel.

2. Структура на базата данни – Структурирането на базата данни също не беше лесно нещо за нас, защото трябваше да помислим за добър начин, по който можем да направим таблиците в базата данни. Искахме да я направим възможно най-проста, за да може да работим много по-лесно с нея. Успяхме да го направим чрез моделите в Laravel, с които установихме, че ще ни бъде много по-лесно.

3. Подредба на код – Подреждането на кода и спазването на конвенциите за правилно написан код не беше толкова просто задача, защото добрите практики са много. Трябваше да измислим начин, с който да постигнем ефект на гъвкавост, за да можем в бъдеще да разширим много по-лесно сървърната част. Решихме този проблем с помощта на най-различни шаблони за дизайн (design patterns).

- приложение:

1. Дизайн на потребителския интерфейс (UI)

Главната ни цел при разработката на качествен UI бе да достави максимално удобство на потребителя, без това да се отразява негативно на неговата ефективност. Ето кои са стъпките, през които протича процесът при създаването на UI дизайна на приложение:

- Събиране на изискванията на функционалността - създадохме списък с функционалностите, които ще поддържа нашата системата, за да се постигнат максимално целите на проекта и потенциалните нужди на потребителя.

- Анализ на потребителите и задачите – на тази стъпка трябваше да проведем проучване, което включва анализ на потенциалните потребители на системата, като трябваше да изучим начина, по който те извършват задачите, които функционалностите на дизайна

предоставя.

- Информационна архитектура - разработката на процеса и/или информационния поток на системата (Three flowchart), който показва йерархията на отделните екрани, от които е създадено мобилното приложение.

- Прототип - разработихме кадри, които показват визуално елементи, от които ще се състои интерфейса. На лист хартия начертахме основен дизайн, както и отделни компоненти, от които ще се състои нашия потребителски интерфейс на приложението.

- Тестване на използваемостта - изпробване на прототипи върху обикновен потребител – често се използва техника, наречена „think aloud protocol“, където вие карате потребителя да сподели своите впечатления по време на тестването. По този начин UI тестването позволява на дизайнера да разбере дизайна си, по начина, по който го разбира потребителя и така по-лесно да създаде успешни приложения.

- Графичен UI дизайн – тук вече изготвихме актуалния изглед към финалния графичен потребителски дизайн (Graphic user interface – GUI). Той се базира на данните, придобити по време на етап “Анализ на потребителите и задачите”, но претърпя известни промени, наложени от резултатите от “Тестването на използваемостта”. В зависимост от типа на интерфейса, който създавахме, процесът включваше и програмиране, чрез което накарахме системата да изпълни дадено действие.

- Софтуерна поддръжка – След пускането на новия интерфейс, регулярна поддръжка може да бъде необходима, за да се отстранят софтуерни бъгове, да се променят опции или да се направи цялостен ъпгрейд на системата. Ако обаче се стигне до ъпгрейд на интерфейса, той задължително трябва да премине наново през всички гореописани стъпки. Потребителският интерфейс е графичното оформление. Състои се от бутоните, оформлението на текста, изображенията, плъзгачите, цветовете, шрифтовете и всички останали елементи, с които потребителят взаимодейства в приложението. UI дизайнът включва оформление на екрана, преходи, анимация на интерфейса и всяко отделно микро-взаимодействие. Тази работа се оказа изключително сложна, тъй като трябваше да доставим максимално добро изживяване на обикновения потребител. Тук предоставяме линкове към помощните материали, които ни помогнаха да се справим с тази трудност:

https://developer.android.com/guide/topics/ui/look-and-feel

https://developer.android.com/guide/topics/ui/controls/button

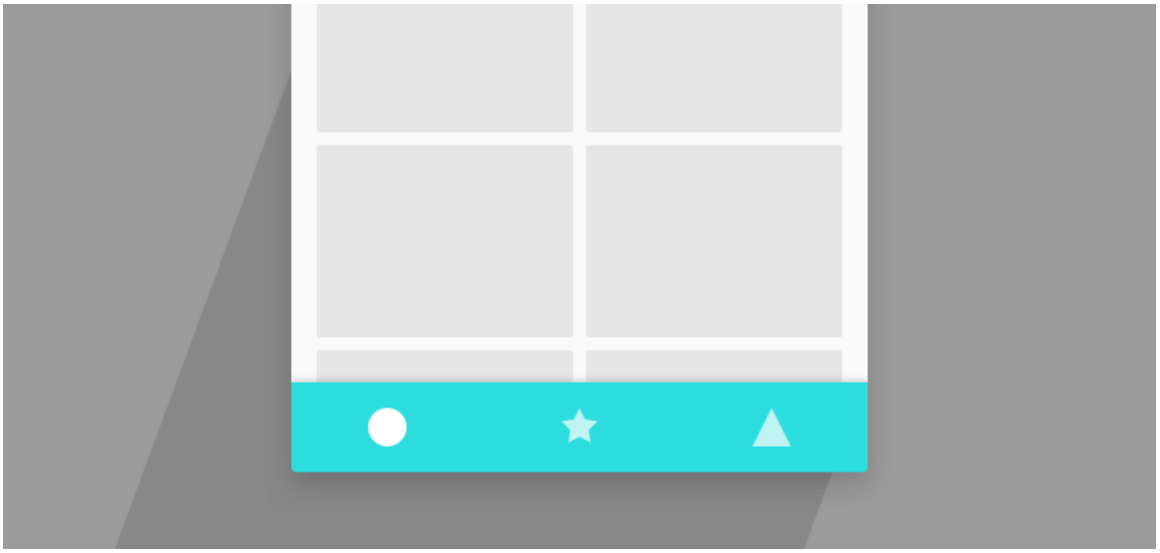
https://developer.android.com/guide/topics/ui/floating-action-button

https://bit.ly/3a9ZEPe

https://bit.ly/3elIhhL

2. Навигационна система – Голяма роля в изработката на мобилно приложение е начина на навигация. Всяко приложение само по себе си е една презентация на услуги, продукти или информация. Ако потребителите изпитват трудност да намерят някаква препратка към желаната от тях информация, то това е лошо потребителско изживяване. Това разбира се не означава половината екран да бъде в менюта или да се прекали с така наречените „залепени менюта“, които да закриват важната за потребителя информация. Това е част от така нареченото понятие “Потребителско изживяване (User experience - UX)”. Проектът “3DPernikGuide” се сблъска с оформянето, конфигурирането, създаването и поддръжката на една навигационна система. Първоначалният ни замисъл бе да използваме така наречената “Bottom Navigation” система в Андроид. “Bottom Navigation” създава ленти за навигация в долната част на екрана, което улеснява изследването и превключването отделните екрана или слоеве (Fragment) на приложението само с едно докосване. Този модел може да се използва, когато има между 3 и 5 главни дестинации. Тази навигационна система не оправда очакванията ни, защото приложението стана твърде хаотично и неподредено, затова преценихме да опитаме нещо ново.

Bottom Navigation Menu (Пример):



След дълго изследване и проучване на информация в интернет, засягаща тази обширна тема, стигнахме до извода, че трябва да използваме така наречаното навигационно меню “Navigation Drawer”. Този модул осигурява достъп до дестинации и функционалности на приложението, като например: превключване на акаунти. Модулът може да бъде конролиран чрез иконата на навигационното меню, което се създава по подразбиране. Главното тук е, че се предоставя на потребителя по-добро изживяване, защото Навигационното меню невинаги е на екрана, което не пречи за показването на функционалностите на приложението “3DPernikGuide”.

Navigation Drawer (Пример):

Тук даваме линкове към помощните материали, които ни помогнаха да се справим с избора на навигационна система:

- https://material.io/components/navigation-drawer/
- https://material.io/components/bottom-navigation/
- https://bit.ly/2VxJutO

3. Navigation Component – предишните андроид приложения, които сме разработвали, използвахме custom (персонализирана) система за навигация между различните слоеве и екрани на приложението. В последната година, Google заедно с екипа на Android Developers, представиха нов набор от библиотеки - Android Jetpack. Android Jetpack е разделен на четири основни части:

- “Foundation”
- “Architecture”
- “Behavior”
- “UI”

Частта “Architecture” има подвидове, част от която е “Navigation”. “Navigation” се отнася до взаимодействията, които позволяват на потребителите да се придвижват, навлизат и да се връщат назад от различните части и съдържание в приложението. Избрахме да заменим използваната от нас досега персонализирана навигация с навигацията на набора от компоненти Android Jetpack, защото навигационният компонент на Android Jetpack ни помогна да внедрим навигация, от по-прости кликвания на бутони до по-сложни модели, като ленти за приложения и навигационната система “Navigation Drawer”. Навигацията може да бъде конфигурирана с визуализатор (Navigation Graph), който е препоръчително да се използва при връзката между отделните слоеве (Fragments) на даден екран. Друг начин за оформяне на навигацията между отделните екрани е чрез програмен код. Трябваше ни време да разучим компонента, тъй като той е все още нов и няма голям набор от информация, уроци и примери за използването му. Тук оставяме линкове към помощните материали, които ни помогнаха да разучим новата навигационна система:

- https://bit.ly/2XADdjL
- https://bit.ly/2XA3Q8p
- https://bit.ly/3cb73PY

4. Правилно използване на избрания от нас архитектурен шаблон.

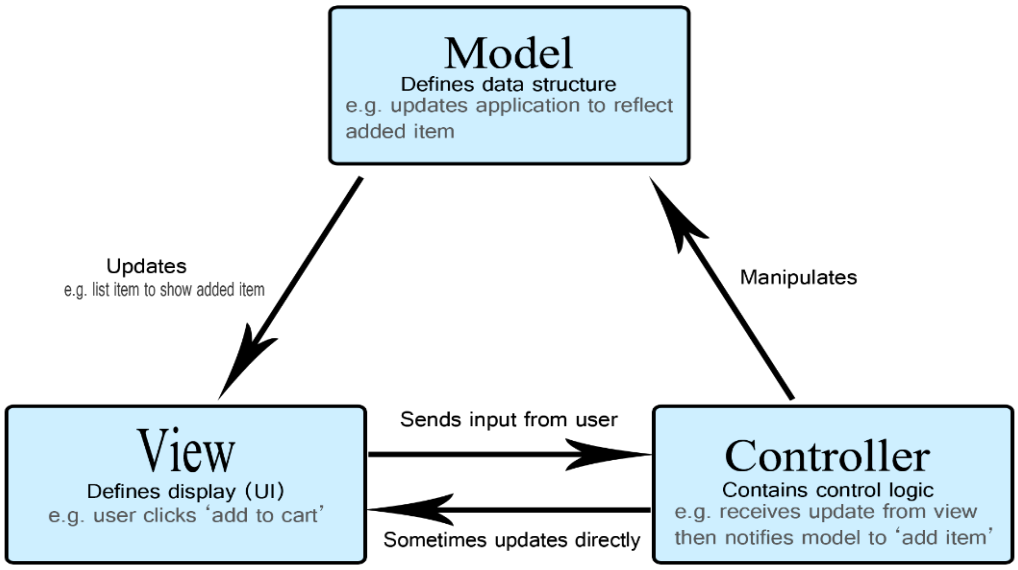
В Android имаме проблем, възникващ от факта, че дейностите на Android са тясно свързани както с потребителския интерфейс, така и с механизмите за достъп до данни. Можем да намерим крайни примери като CursorAdapter, който смесва адаптери, които са част от изгледа, с курсори, нещо, което трябва да бъде пренесено в дълбочината на слоя за достъп до данни. За да може приложението да бъде лесно разширяващо се и поддържано, трябва да определим добре разделени слоеве. Какво правим утре, ако вместо да извлечем същите данни от база данни, трябва да го направим от обширна уеб услуга? Ще трябва да преработим целия си изглед, затова избрахме една от най-доказаните архитектури в областта на андроид приложенията - MVP (Model - View - Presenter). Този архитектурен шаблон перфектно

отговаряше за изпълнението на поставените ни цели и фукционалности. MVP прави изгледите (views) независими от нашия източник на данни. Разделяме приложението на поне три различни слоя, което ни позволява да ги тестваме независимо. Тук възниква проблемът с правилното структурирането на файловете и кода. В минали проекти сме използва единствено архитектурата MVC (Model - View - Controller). Сигурно се питате: Защо този проект не е разработен на MVC? Отговорът е следният: Традиционно написаното приложение за Android, поддържано от MVC архитектура се състои от три основни компонента:

- Model (Модел) - Моделите съдържат показните данни. Обикновено тези данни се получават от мрежата или от локална база данни. След това данните се поставят в малки, прости класове, които другите компоненти могат да използват.

- View (Изглед): Изгледите са това, което се показва на потребителя. Изгледите могат да реагират на всяко взаимодействие, което може да има потребител с екрана. Изгледът трябва да носи отговорност само за показване на неща и не трябва да съдържа никаква бизнес логика. По този начин изгледът е лек компонент в сравнение с контролера и обикновено не съдържа много код. В Android отговорността за изгледа често пада върху дейности и фрагменти.

- Контролери: Контролерите са начин за свързване на модели и изгледи заедно. Контролерът актуализира модела, когато нещо се случи в изгледа. Контролерът също ще актуализира изгледа, когато моделът се промени. Твърде често отговорностите на контролера се намират и в дейности и фрагменти.



Въпреки, че при липсата на каквато ѝ да е друга архитектура, MVC е вършила чудесна работа, тя е имала своите недостатъци. Когато става въпрос за внедряване на MVC на платформата Android, нещата стават трудни. За начало по-голямата част от логиката се озовава в контролера. Често срещан проблем с Android е работата с контролер, която съдържа цялата логика. Тогава контролерът носи цялата отговорност за това, което се показва на екрана. За прост екран това може да бъде управляемо, но тъй като се добавят функции, този файл ще продължи да расте.

Освен това, дейностите с Android са тясно свързани както с потребителския интерфейс, така и с механизмите за достъп до данни. По този начин е лесно да попаднете в капана на поставяне на контролер и да видите логиката в дейност или фрагмент. Това обаче създава плътно свързани компоненти, което затруднява рефакторирането и промяната.

Изгледният слой трябва да се отнася само за изгледите. Не трябва да се знае за бази данни или мрежови обаждания. Освен това е трудно да се тестват компоненти, когато са плътно свързани. Трябва да тествате кода, който съществува в контролера. Голяма част от тествания код ще се нуждае от стартиране на Android SDK компоненти, като дейности, които се нуждаят от пълно работещо устройство или емулатор. Основното тестване на единица няма да помогне;

ще трябва да използвате скъпи инструментални тестове на единици, за да тествате основните части на приложението.

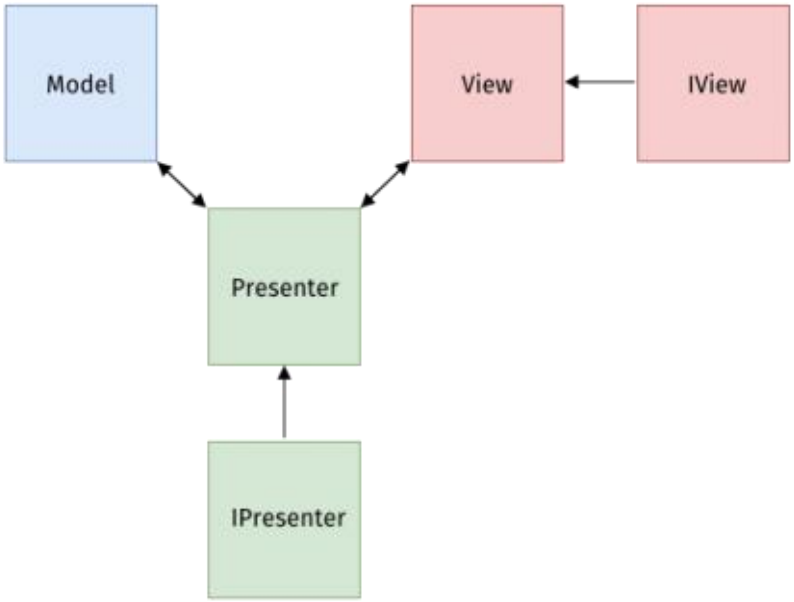
Една алтернатива на проблемите, представени от MVC, е отделянето на някои части един от друг. MVP е архитектурен модел, който можете да използвате, за да се справите с някои от недостатъците на MVC, и е добра алтернативна архитектура. Той предоставя лесен начин да помислите за структурата на приложението си. Той осигурява модулност, тестируемост и като цяло по-чиста и поддържана база данни с кодове. MVP се състои от следните компоненти:

- Model (Модел): Моделът ще продължи да съдържа данните в прости класове, така че всъщност нищо не се променя.

- View (Изглед): Изгледът ще продължи да се прилага с класове, дейности или фрагменти, но ще променим обхвата на това, което изгледът контролира.

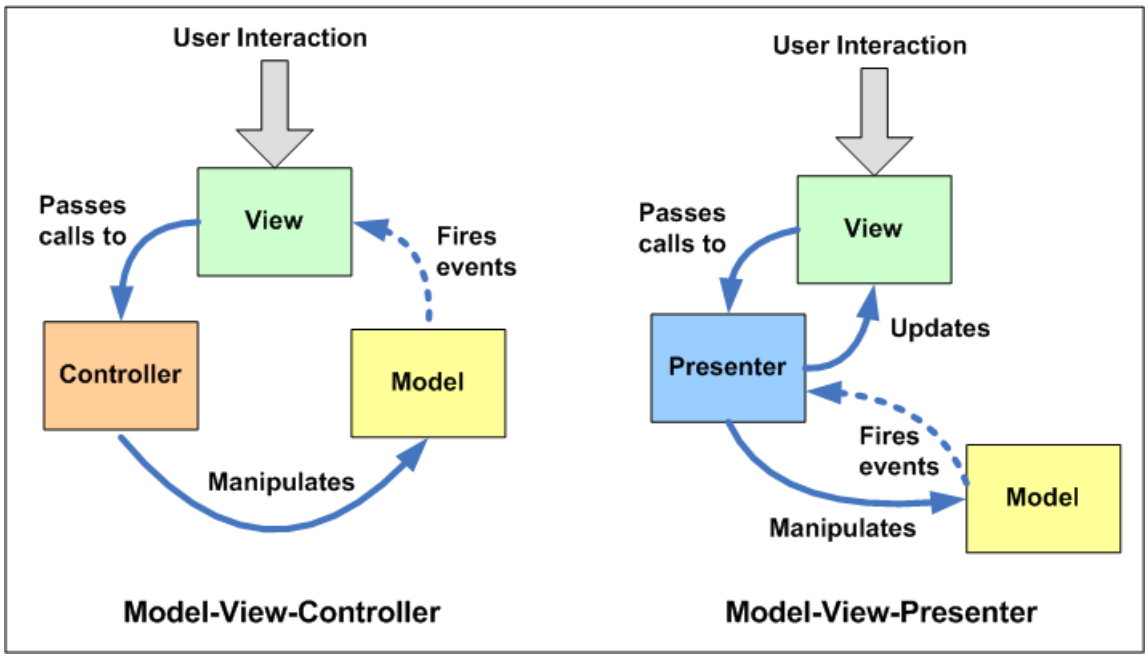
- Presenter (Презентатор): Последната част е презентаторът, който обработва актуализации на потребителския интерфейс въз основа на промените в модела на данни и също обработва входовете на потребителите. Презентаторът ще съдържа голяма част от бизнес кода и ще замени контролера от MVC.

В MVP, вместо да има контролен клас, който обработва както промените в модела, така и това, което се показва на екрана, частите на контролера и изгледа са разделени, а презентаторът и изгледът стават по-леки.



В кода ще видите интерфейси, които определят презентатора и изгледа. Интерфейсите помагат за отделяне на частите от архитектурата. Интерфейсът формира договор между водещия и изгледа. Те също така ще ви помогнат да определите и напишете тестови случаи по-късно. Когато има промени в данните, водещият се уведомява, че данните са променени. След това изгледът ще получи тези данни през презентатора и ще се актуализира, използвайки данните от презентатора. Можете също да отидете в обратна посока със събития от гледката. Когато потребител взаимодейства с изгледа, се извиква метод на презентатора. След това презентаторът извиква подходящия метод за това действие за актуализиране на модела.

Разликата между MVC и MVP



Тук оставяме линкове към помощните материали, които ни помогнаха да направим правилния избор за архитектурен шаблон:

- https://bit.ly/2ycglMS
- https://bit.ly/3cmJ4xp
- https://bit.ly/3cb73PY

- **engine**:

1. Прочитане на Wavefront .obj файловете

OBJ файлът представлява 3D формат на изображения, който може да бъде експортиран и отворен от различни програми за редактиране на 3D изображения. Той съдържа триизмерен обект, който включва 3D координати, текстурни карти, полигонални лица и друга информация за обекта. OBJ файловете могат също да съхраняват препратки към един или повече .MTL файлове, които съдържат материали или цветове за обекта. Прочитането и редактирането на един такъв файл беше наистина трудна задача, защото този формат е сложен за прочитане с програмен код.

OpenGL ни помогна за прочитането и модифицирането на такъв тип файлове, но на много базово ниво. Когато отворим OBJ файла с помощта на текстов редактор, виждаме най-различни символи. Тези символи са типове данни и координати на 3D обекта. Тук ще представим най-често срещаните символи като ги разделим на няколко секции:

- Данни за върха:
  - geometric vertices (v)
  - texture vertices (vt)
  - vertex normals (vn)
  - degree (deg)
  - basis matrix (bmat)
  - step size (step)
- Елементи:
  - point (p)
  - line (l)
  - face (f)
  - curve (curv)
  - 2D curve (curv2)
  - surface (surf)
- Групиране:
  - group name (g)
  - object name (o)

- Атрибути за визуализация:
    - material name (usemtl)
    - material library (mtllib)

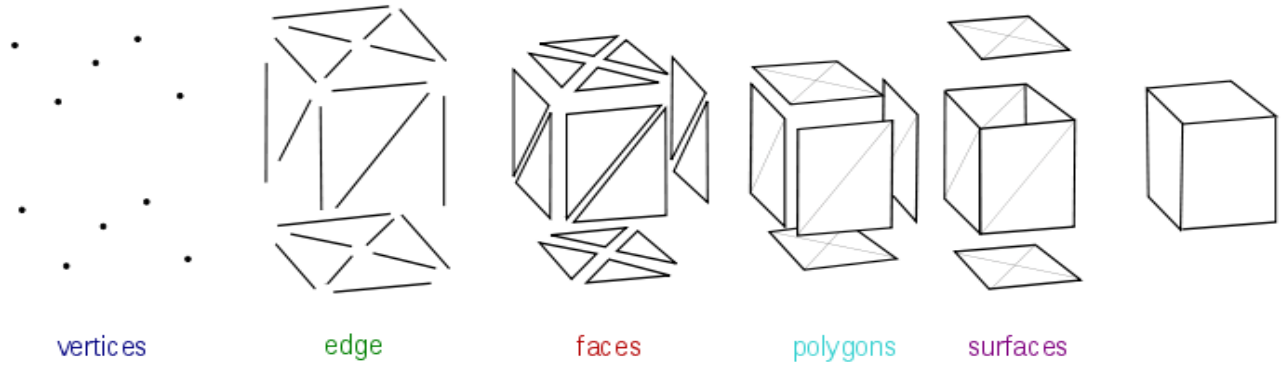
Това са част от типовете, които можете да срещнете в един OBJ файл. Комплексната структура на тези файлове могат да объркат всеки разработчик, готов да работи с тях – ръчна преработка или с помощта на програмен код. Избрахме да работим с този тип данни в нашето приложение, защото текстурите се намират в отделен файл и това спомага за бързото им редактиране.

Пример за Wavefront OBJ File

```
mtllib church.mtl
o Cube_Cube.001
v -4.694561 1.406826 -13.434690
v -5.311534 1.406826 -13.434690
v -5.483025 1.406826 -13.384379
v -5.554059 1.406826 -13.262917
v -5.554059 1.406825 -5.738897
v -5.483025 1.406825 -5.617435
```

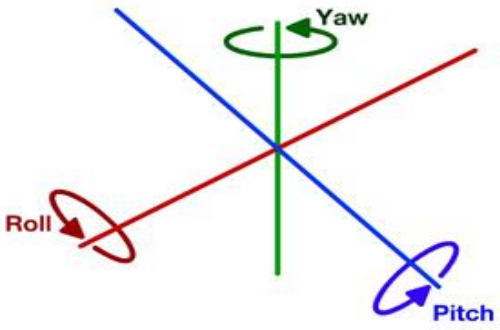
Тук предоставяме линкове към помощните материали, които ни помогнаха с изучаването на структурата на файловете от типа - OBJ:

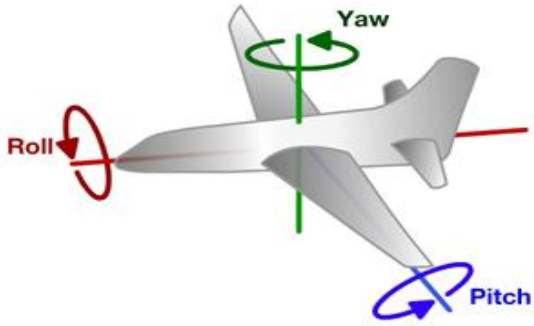
- https://wblut.com/reading-an-obj-file-in-processing/
- https://bit.ly/3cs4h94
- https://en.wikipedia.org/wiki/Wavefront_.obj_file



2. Въртене на 3D обекта – След като успяхме напълно да визуализираме един от 3D моделите, се изискваше да създадем система, която да завърта модела във всяка една посока, посочена от потребителя. Тази част от разработката на визуализатора беше най-трудна, тъй като OpenGL не предоставя вградена функция, която да използваме и трябваше да създадем собствена такава. Първоначално трябваше да разучим как изчисляваме и ориентираме даден 3D модел. За тази цел трябваше да притежаваме солидни знания по математика, но за щастие бяхме добре подготвени и теорията не ни затрудни. Първо изучихме какво е 3D ротация. Навлязохме в дълбочина, за да можем адекватно да разработим нашата система. 3D ротациите имат няколко варианта:

- около оста x – pitch
- около оста y – yaw
- около оста z – roll





Разгледахме два начина за въртене на обект. Първият, който не се препоръчва, е “Ъгли на Ойлер”. Ъглите на Ойлер са подходящи за по-лесни операции, но когато нещата станат по-сложни, този начин не е адекватен за работа. След като потърсихме повече информация, стигнахме до извода, че най-добре за ротацията във визуализатора да използваме “кватерниони”. Кватернионът е четириелементен вектор, който може да се използва за кодиране на всяко въртене в 3D координатна система. Технически кватернион е съставен от един реален елемент и три сложни елемента и може да се използва за много повече от въртене, но в нашият визуалитатор се използва на базово ниво.

В проекта извършваме проста операция на нормализиране на кватернионите.

Нека разгледаме следния кватернион: $Q = a + b.i + c.j + d.k = [a \quad b \quad c \quad d]$

Нормализирането се дава по следната формула: $Q_{normalized} = \frac{Q}{|Q|}$

Където: $|Q| = \sqrt{Q.\overline{Q}} = \sqrt{\overline{Q}.Q} = \sqrt{a^2 + b^2 + c^2 + d^2}$

Краен резулат се дава:

$$Q_{normalized} = [\frac{a}{\sqrt{a^2+b^2+c^2+d^2}} \quad \frac{b}{\sqrt{a^2+b^2+c^2+d^2}} \quad \frac{c}{\sqrt{a^2+b^2+c^2+d^2}} \quad \frac{d}{\sqrt{a^2+b^2+c^2+d^2}}]$$

Формулата, преобразена в програмен код на Java:

```
private float x, y, z, w;

public Quaternion(float x, float y, float z, float w) {
    this.x = x;
    this.y = y;
    this.z = z;
    this.w = w;
}

public void normalize() {
    float mag = (float) Math.sqrt(w * w + x * x + y * y + z * z);
    w /= mag;
    x /= mag;
    y /= mag;
    z /= mag;
}
```

Ресурсите, които използвахме, за да създадем системата на въртене в нашия engine:

- https://bit.ly/34UZACg
- https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
- https://www.youtube.com/watch?v=q0jgqeS_ACM
- https://www.wikiwand.com/en/Quaternions_and_spatial_rotation
- https://youtu.be/d4EgbgTm0Bg

- https://youtu.be/zjMuIxRvygQ
- https://youtu.be/SCbpxiCN0U0

4.5 **Логическо и функционално описание на решението:**

- **сървър**: Като за начало инсталирахме Laravel на нашата машина чрез Composer. Файловата структура - стандартна за фреймуърк (framework) като Laravel. За по-качествен код в допълнително създадените файлове от нас, използвахме MVC (Model-View-Controller) архитектура. Тук са описани по отделно допълнително създадените папки и файлове:

- **app –** централната точка, която съдържа основния код на сървъра въпреки това, почти всички класове се намират в тази директория

- **bootstrap –** директория за зареждане на файла app.php, който зарежда Laravel. Тази директория съдържа и кеш директория, която съдържа файлове, генерирани от Laravel за оптимизация на производителността, като файловете за кеш (cache), маршрута (route) и услугите (services)

- **config –** конфигурационната директория, както подсказва името, съдържа всички конфигурационни файлове за нашето приложение

- **database –** директорията за базата данни

- **public –** публичната директория съдържа файла index.php, който е входната точка за всички заявки, влизащи в приложението ни

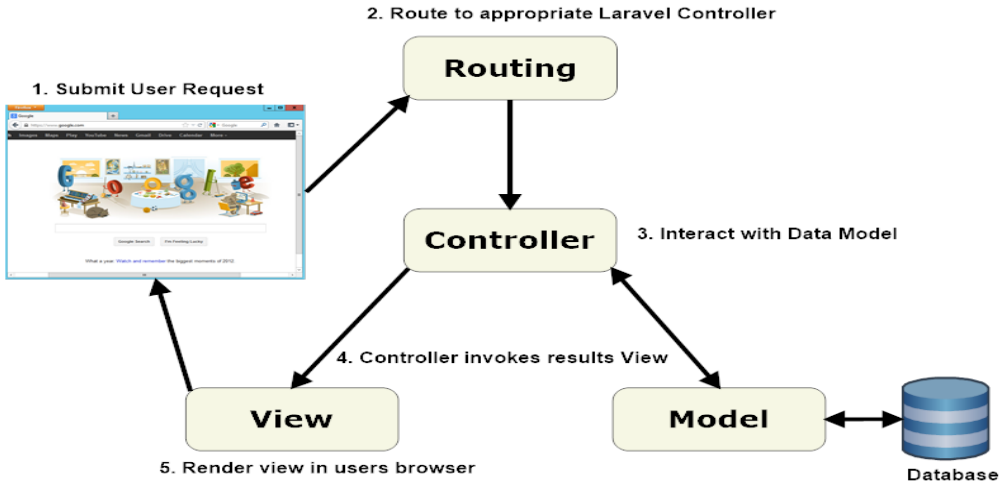
- **resources –** директория с ресурси и езикови файлове

- **routes –** директория на маршрутите (routes). Най-важният файл в тази директория е api.php, който съдържа всички мрежови адреси на дадения ресурс за Application programming interface (API)

- **storage –** директория за съхранение на компилираните шаблони на Blade темплейтната система

- **tests –** тестова директория за автоматизирани тестове (Unit Tests)

- **vendor –** директория, която съдържа composer зависимости



- **приложение**: При започване разработката на приложението, проверихме текущите версии на Android Studio и Gradle, защото често се появяват грешки във файловете, свързани с различните версии. Файловата структура е стандартна за андроид приложение, написано на MVP (Model-View-Presenter) архитуктура. Трябва да бъдат описани поотделно допълнително създадените папки и файлове:

- **base** - файловете, принадлежащи към тази папка, могат да се опишат като базови за приложението. Съдържат методи, които имат за цел да спестят кода в другите класове на приложението.

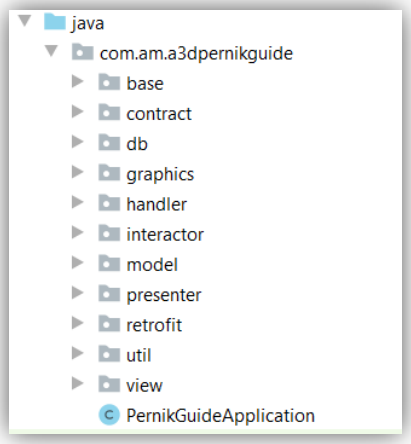
- **contract** - интерфейси, които участват активно в комуникацията на отделните слоеве на MVP архитектурата.

- **db** - директория, съдържаща файлове, които са стандартни за конфигурацията на SQLite локална база данни с допълнение Room ORM.

- **graphics –** една от най-важните директории в андроид приложението. Файловете са отправна точка към нашата библиотека за визуализация на 3D модели. Единствената директория, която извършва комуникация с визуализатора.

- **handler –** класове, отговорни за различна комуникация и случващи се събития в приложението. Най-често те са отговорни за връзката между Model и трите услуги за съхранение на данни (сървърна част – глобална база данни, Shared Preferences – съхрание на данни като кеш в приложението, локална база данни - Room + SQLite).

- **interactor** – Interactor или по-познат като Model, най-често взаимодейства с презентационния слой на MVP архитектурата и с класовете, отговорни за комуникацията към услугите за съхранение на данни (handlers).



- **model** – класове, използвани при WEB комуникацията с помощта на Gson за преобразуване /Parsing/ на JSON към обикновени Java обекти /POJO/.

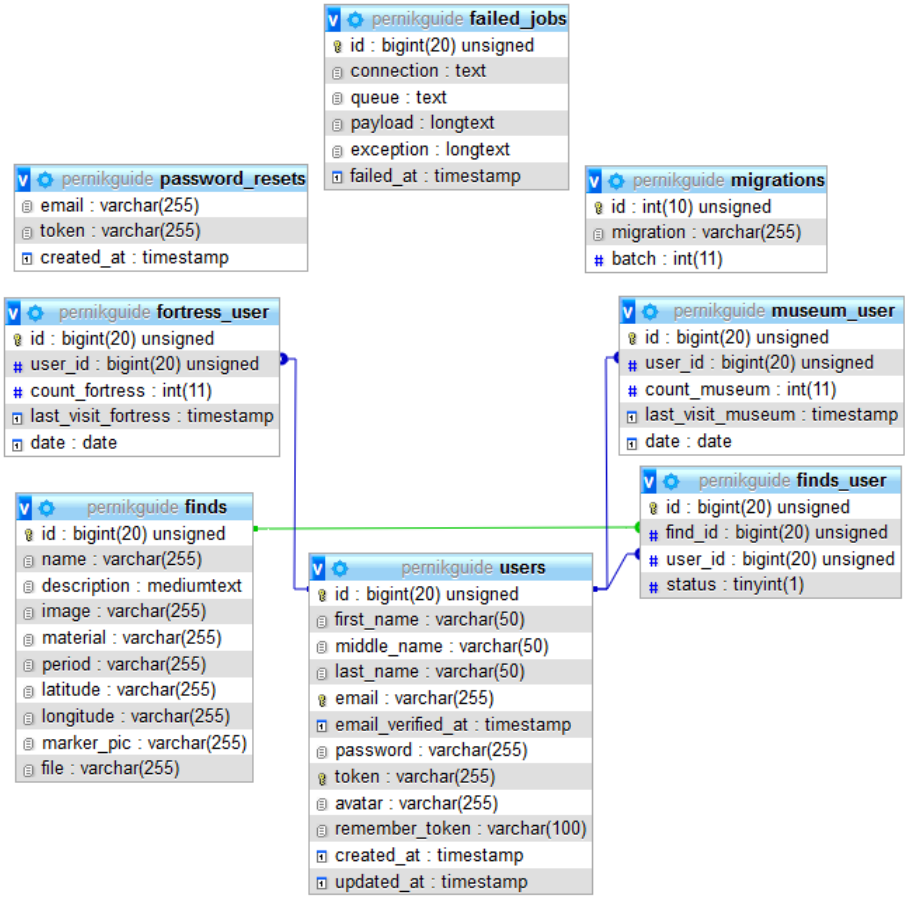
- **presenter** – файлове, които играят ролята на презентационен слой в андроид приложението. Тези файлове комуникират с визуалните файлове на приложението и с така наречените – Модели (Model).

- **retrofit** – централна точка за комуникацията между приложението и WEB средата.

- **util** – разнообразни класове, управляващи различни части от визуалните компоненти на приложението.

- **view** – Дирuктория, която управлява View частта от MVP архитектурата. Тук се съдържат файлове, свързани с визуалните компоненти на приложение – екрани и фрагменти.

База Данни (БД):



Базата данни се състои от 8 таблици (users, fortress_user, museum_user, finds, finds_user, password_resets, failed_jobs, migrations):

- **users** – Таблица, която отговаря за потребителите. В нея се съдържат регистрираните потребители и чрез нея потребителите влизат в профила си. Тя притежава следните полета: (id, first_name, middle_name, last_name, email, email_verified_at, password, token, avatar, remember_token, created_at, updated_at).

- **fortress_user** – Таблица, която отговаря за посещенията на потребителите на туристическия обект „Средновековна Пернишка Крепост“. В нея се съдържат броя посещения на потребителите, последното посещение на потребителите и датата на посещение на потребителите. Също така, чрез нея се изчислява класацията за потребители с най-много посещения. Тя притежава следните полета: (id, user_id, count_fortress, last_visit_fortress, date).

- **museum_user** – Таблица, която отговаря за посещенията на потребителите на туристическия обект „Подземен минен музей“. В нея се съдържат броя посещения на потребителите, последното посещение на потребителите и датата на посещение на потребителите. Също така чрез нея се изчислява класацията за потребители с най-много посещения. Тя притежава следните полета: (id, user_id, count_museum, last_visit_museum, date).

- **finds** – Таблица, която отговаря за всички находки свързани с потребителите в туристическия обект „Средновековна Пернишка Крепост“. В нея се съдържат наименованието, описанието, снимката, материала, периода, координатите, марките и 3D модела. Тя притежава следните полета: (id, name, description, image, material, period, latitude, longitude, marker_pic, file).

- **finds_user** – Таблица, която отговаря за всички находки, които са посетени от потребителите в туристическия обект „Средновековна Пернишка Крепост“. В нея се съдържат идентификационния номер на находката, идентификационния номер на потребителя и дали е била посетена или не. Тя притежава следните полета: (id, find_id, user_id, status).

- **migrations** – Таблица, която идва по подразбиране с Laravel и отговаря за миграциите (таблиците). В нея се съдържат имената на таблиците, които сме създали. Тя притежава следните полета: (id, total_count, last_visit, date).

- **failed_jobs** – Таблица, която идва по подразбиране с Laravel и отговаря за задачите (jobs) в опашките (queues), които също са част от Laravel. Тя притежава следните полета: (id, connection, queue, payload, exception, failed_at).

- **password_resets** – Таблица, която идва по подразбиране с Laravel и отговаря за паролите и подновяването им. Тя притежава следните полета: (email, token, created_at).

4.6 **Реализация:** За изработката на приложението е използвана среда за разработка Android Studio, включваща Java и XML редактор, както и виртуално Android устройство (емулатор) за тестване на приложението по време на разработката. Също така ADB (android debug bridge) за своевременно тестване върху реални устройства.

При писането на PHP кода е използван редактор Sublime Text 3, в комбинация с различни разширения за оптимизиран работен процес. Базата данни се администрира с phpMyAdmin.

Системата е изградена с повишено внимание към файловата и логическа структура. Тъй като имаме изключително много идеи за доразвитие на системата, беше нужна солидна и лесна

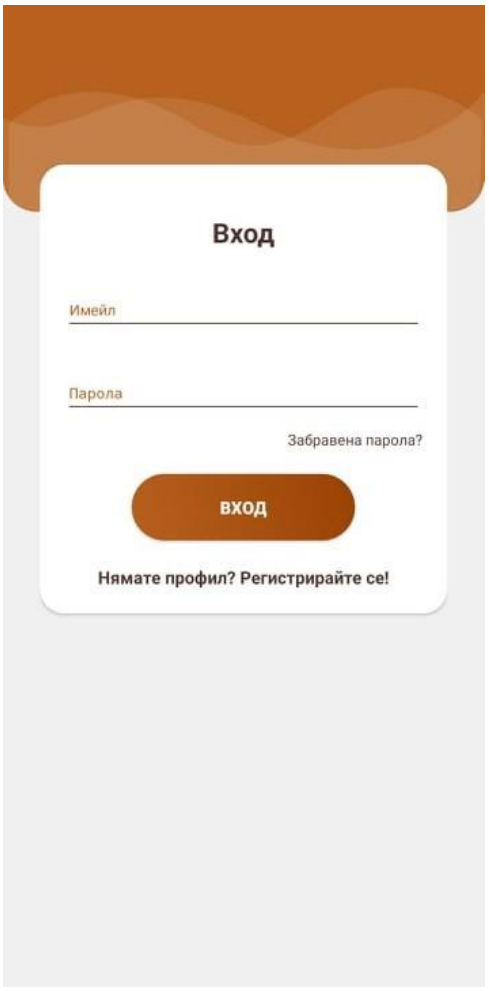
за поддръжка основа, каквато смятаме, че сме изградили. За да работим съвместно използваме система за управление на версиите Git /Version Control/ и място за съхранение на сорс кода GitHub. По време на реализацията много акцентираме върху комуникацията, тъй като всеки от нас се специализира в конкренти области и често се налага взаимно да обменяме опит, както и да вземаме важни решения, чийто последствия да бъдат решаващи за в бъдеще.

4.7 **Описание на приложението:**

Приложението може да бъде инсталирано от папка **apk** или от нашия информативен сайт: https://pernikguide.noit.eu/, като за в момента очакваме одобрение от Google Play Service за публикуване на приложението.

„3DPernikGuide" е приложение, което печели своята широка потребителска аудитория, като предоставя отличаващ се продукт на пазара. Проектът е уникален от останалите подобни продукти, свързани с туристическите услуги, тъй като предоставя едно по-реалистично преживяване при посещаване на дадена забележителост в град Перник. Приложението притежава интерфейс, изцяло изграден чрез визуалния език Material Design. Добре познатият дизайн на ежедневно използваните продукти на Google. Внушава усещането у потребителя, че той е вече запознат с облика на приложението ни, макар и да го вижда за първи път. Това спестява неудобството от неясни функционалности и потвърждава универсалността на продукта.

При стартиране на приложението потребителят се намира в началната страница, където има възможност да влезе в своя профил или да създаде такъв. При регистрацията на потребителски профил в приложението, потребителят въвежда данни относно своето Име, Презиме, Фамилия, Е-мейл адрес, парола. Има поле и повтаряне на паролата, където се сравняват двете пароли, за да сме сигурни, че потребителят записва исканата от него парола. При наличие на вече създаден профил, потребителят е необходимо да въведе своите данни и да влезе в приложението. След като потребителят влезне в профила, неговите данни се запазват в кеш и няма нужда да влиза в собствения си акаунт отново при повторно зареждане на приложението.



Страницата, в която потребителят влиза, показва 5 основни контейнера, които се отличават от заобикалящата обстановка на приложението:

- поздравителен контейнер – в тази част се изписва поздравително съобщение с вашите име и фамилия.

- брой посещения (Крепост) – изписва колко пъто сте посетили Средновековната пернишка крепост

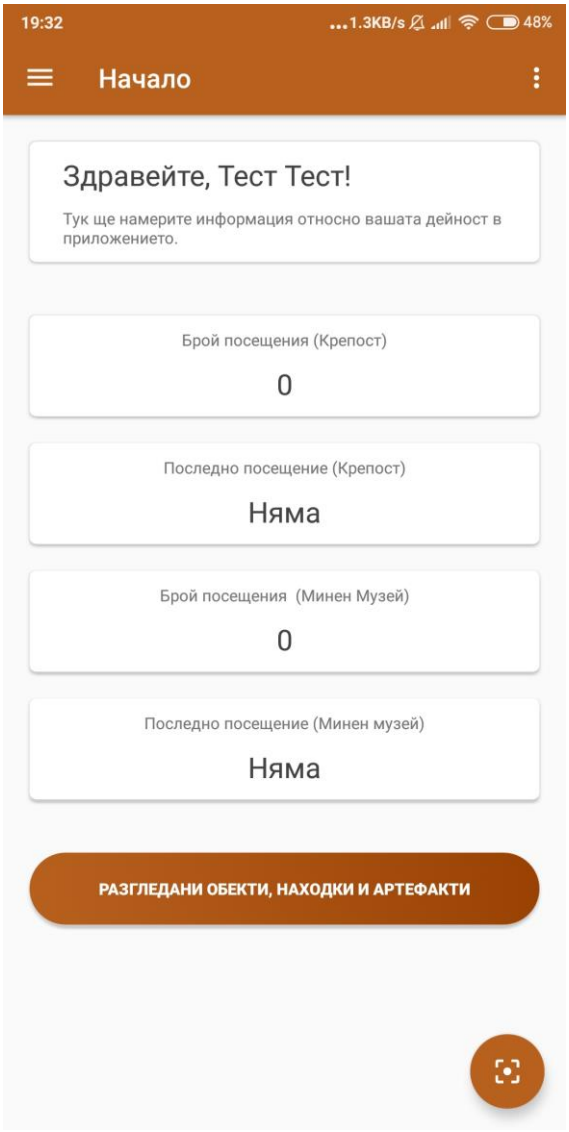
- последно посещение (Крепост) – в този контейнер се изписва кога последно сте посетили Средновековната пернишка крепост. Пример:
2020-04-19 13:43:55.

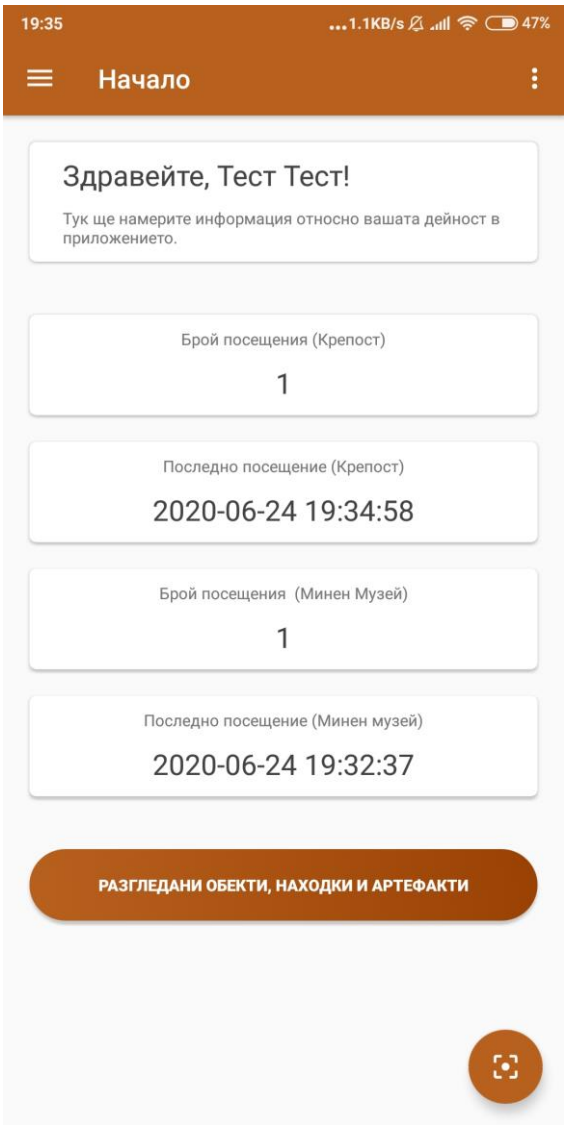
- брой посещения (Минен музей) - изписва колко пъто сте посетили Подземния минен музей.

- последно посещение (Минен музей) – в този контейнер се изписва кога последно

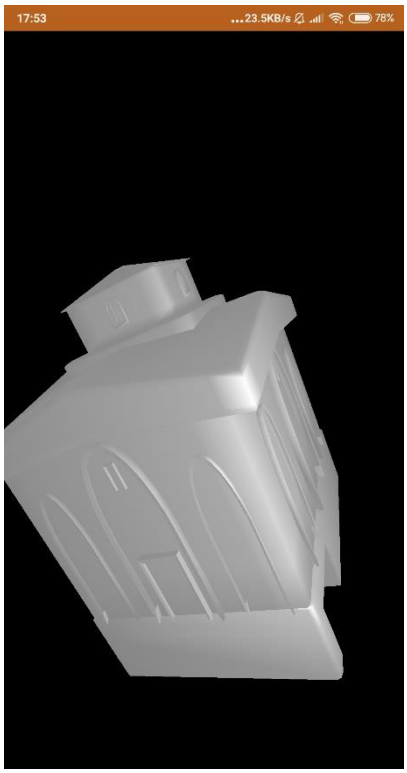
сте посетили Подземния минен музей.

Пример: 2020-04-19 13:43:55.





В долния десен ъгъл на екранна се намира така нареченият “Floating Action Button”. Когато натиснем на него, той ни отвежда към нов екран изцяло запълнен с QR code скенера. Този скенер има две функции в двата горни края на екрана – в левия ъгъл се намира функцията, която спира фокусирането на камерата, а в другия край се намира функцията за светкавицата. Тази функция позволява на потребителя да сканира QR кодове без светлина, тъй като включва вградената изкуствена външна светлина на телефона. Скенерът може да сканира единствено QR кодове, генерирани от нас. Кодът е съставен от 16 символен низ, който е запазен локално в базата данни на всяко едно устройство, на което е инсталирано нашето приложение. Когато сканираме код се отваря нов екран с 3D модел, отговарящ на съответния код в локалната база данни. Моделът може да се върти от потребителя, както да се увеличава и намалява, за да може реалната картина на обекта да е абсолютно детайлна. Извънредното положение ни попречи да залепим QR кодовете на предназначените за тях места, но след края на извънредното положение тези кодове ще бъдат поставени на мястото си.



Препоръчително е кодовете да се сканират от хартиен носител!

Църква

Крепостна стена

Селище

Водна помпа

Машина за дробене

Табло

След като се върнем на началния екран, най-отгоре на екрана имаме така наречения **Toolbar.** Той е част от Material Design. В него се съдържат: иконка за отваряне на навигационната система, името на екрана, иконка за повече менюта, откъдето се отваря и менюто за изход от профила. При натискане на иконката за навигационната система, виждаме три менюта:

- Начало - Това е началният екран на нашето приложение, където се намират контейнерите за информация, относно дейността на потребителя. Това са гореспоменатите контейнери, които описахме подробно.

- Забележителности – препратка към двете най-известни забележителности в град Перник.

– **Средновековната пернишка крепост** - Средновековната пернишка крепост е разположена на неголямо скалисто плато в югозападната част на Перник. Запазените останки очертават многоъгълна крепост, за която се смята, че е построена по времето на хан Омуртаг. Археолозите откриват останки, които показват, че крепост на това място е съществувала още през VI – V в. пр. н. е. Крепостната стена е с дебелина 2 метра. Очертанията й следват естествените извивки на платото.

Тази тракийска крепост е една от най-значимите в българските земи, засвидетелствана по археологически път. В началото на XI век владетел на крепостта и селището Перник е болярин от прабългарски род Кракра. Легендите и песните за непобедимия войвода Кракра се разказват и пеят до ден днешен. Пернишката крепост придобива международна известност, когато театърът на военните действия между България и Византия се преместил в западните български земи. Византийският хронист Скилица е описал борбата на българите, водени от управителя на Пернишката крепост – Кракра, срещу византийския император Василий II. Крепостта издържала на яростните обсади през 1004 и 1016 г.

Крепостта е завладяна последна и средновековният Перник не е разрушен. И през византийското владичество запазва стратегическото си значение.

От крепостта се открива изглед към местността “Кървавото”. Според легендата през 1016г. Кракра устоява на 88-дневна обсада, при която загиват много византийци. Оттук идва името на местността. Според едно от житията на закрилника на града – Свети Иван Рилски, в тази местност се е намирала пещерата на отшелника.

Освен запазените основи на жилищни сгради, посетителите могат да различат и основите на три християнски храма – кръстокуполната църква, голяма трикорабна базилика и една от малкото известни двуетажни припортни църкви-гробници. Тук археолозите откриват и сребърен печат на цар Петър – единственият намерен досега сребърен печат на владетел от това време. Според легендата именно тук царят е прекарал известно време на път за срещата си със Свети Иван Рилски. През 2013г. Община Перник реализира проект „Подобряване на туристическата

атракция - Средновековна крепост „Кракра Пернишки“ в гр. Перник и свързаната с нея инфраструктура“ по ОП“Регионално развитие“ 2007-2013.

Ъгловата кула и Западната порта на крепостта са визуализирани с алтернативни материали. Отремонтирани са всички подходи към Средновековната крепост , а пътят до нея е изцяло реновиран. Изграден е атракцион – наблюдателна кула.

Удобни и добре изградени алеи осигуряват достъп на посетителите до допълнителната атракция за туристите - катапулти и стенобойни оръдия. Върху крепостните стени са монтирани панорамни платформи, които предлагат на посетителите едно неповторимо преживяване – да видят целия град и околностите от птичи поглед.

- **Подземен минен музей** - Подземният минен музей е уникален за Балканския полуостров. Разположен е в две от запазените галерии на първия подземен рудник, който е работил от 1891 до 1966 г.

Посетителите имат възможност да проследят историята на въгледобива в автентична руднична атмосфера.Тридесетте експозиционни камери проследяват развитието на въгледобива от ръчния добив, през извоза на въглища с коне до съвременните механизирани комплекси. В една от камерите е разположен иконостас на небесния покровител на миньорите Св. Иван Рилски, с когото е свързана една от най-старите легенди за "Старите рудници". В екрана с туристическите обекти се съдържат два контейнера за всяка забележителност. Всеки контейнер съдържа снимка и името на туристическия обект. След като изберем някой обект, той ни навигира към нов екран с избраната от нас забележителност. Най-отгоре в менюто можем да видим четири компонента: иконка за връщане към предходното меню, името на туристическия обект, иконка за информация и отново менюто за изход. Когато натиснем върху менюто за информация, екранът показва компонент с подробна информация за съответния туристически обект. След като разгледаме нужната информация, можем да се върнем към екрана със забележителността. На пръв поглед виждаме отново снимка на обекта. Под нея се намират три контейнера:

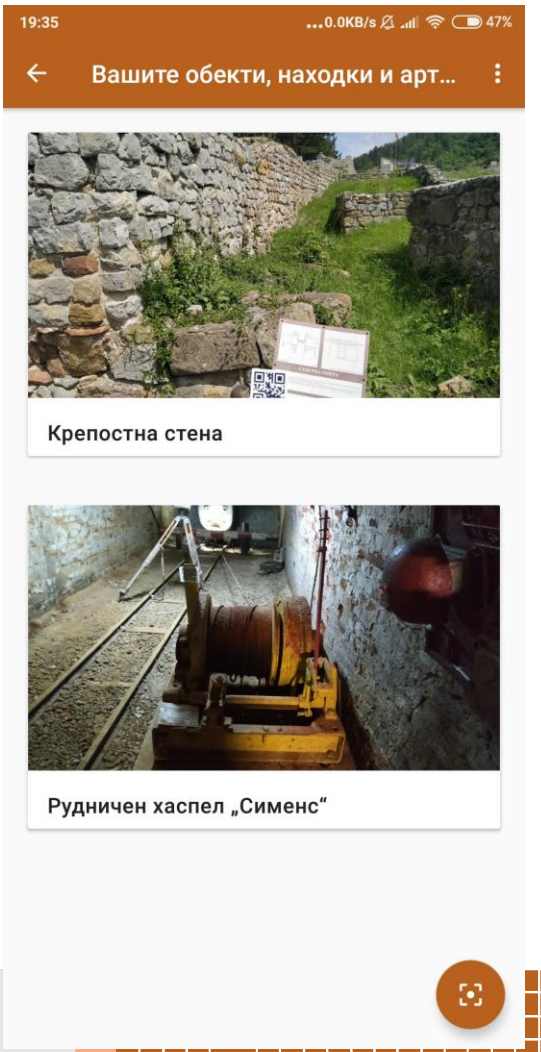
- Общ брой посетители – общият брой посещения на дадената забележителност от всички потребители.

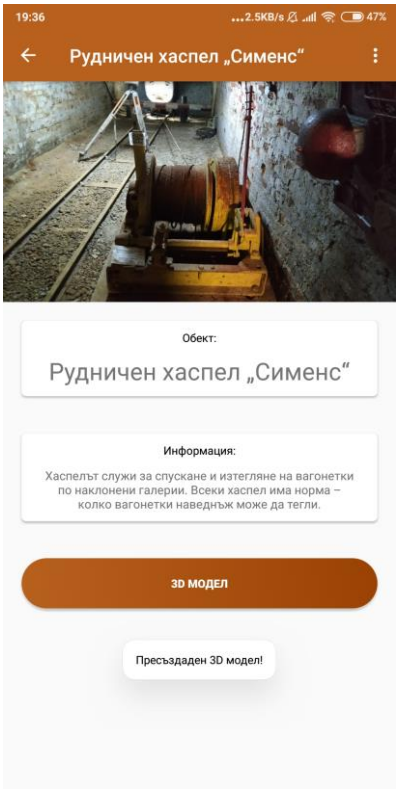
- Последно посещение от потребител – показва последното посещение от даден потребител. Пример: 2020-04-20 17:48:44

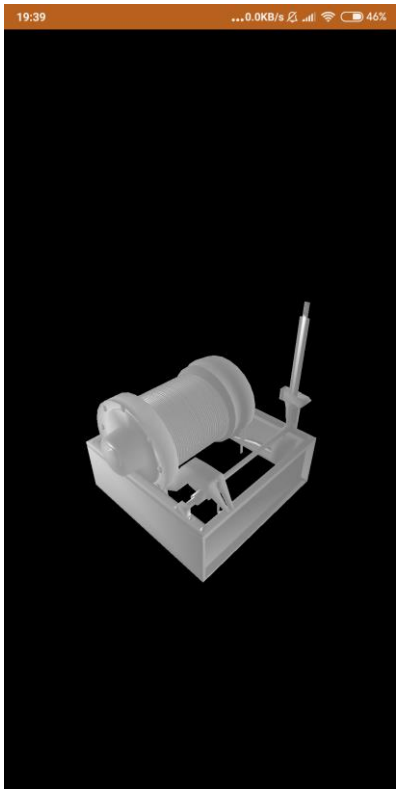
- диаграма – Тази колонна диаграма (Bar Chart) е част от голямото семейство от диаграми на библиотеката MPAndroidChart. С нея представяме петте потребителя, които най-много са посетили зададената от нас туристическа местност.

- Информация – този екран дава ясна представа за нас и нашите цели. Екранът съдържа три основни контейнера: За нас, За проекта – кратко описание за работата ни над проекта, Цел на приложение – кратка информация за целта на нашето приложение.

Бутонът на началния екран “Разгледани обекти, находки и артефакти” ни отвежда до страницата “Вашите обекти, находки и артефакти”, която отговаря за посетените от нас обекти и артефакти на двете забележителности от град Перник. След натискане на даден обект се отваря нов екран, при който се вижда снимката на избраната находка или артефакт, наименуването му, кратка информация и бутон, с който можете да разгледате отново 3D модела на обекта. Обектите и артефактите могат да бъдат посетени с помощта на нашия скенер. При сканиране на QR кода, системата автоматично запазва вашия обект.




19:36
...2.5KB/s 47%
Рудничен хаспел „Сименс“
Обект:
Рудничен хаспел „Сименс“
Информация:
Хаспелът служи за спускане и изтегляне на вагонетки по наклонени галерии. Всеки хаспел има норма – колко вагонетки наведнъж може да тегли.
3D МОДЕЛ
Пресъздаден 3D модел!

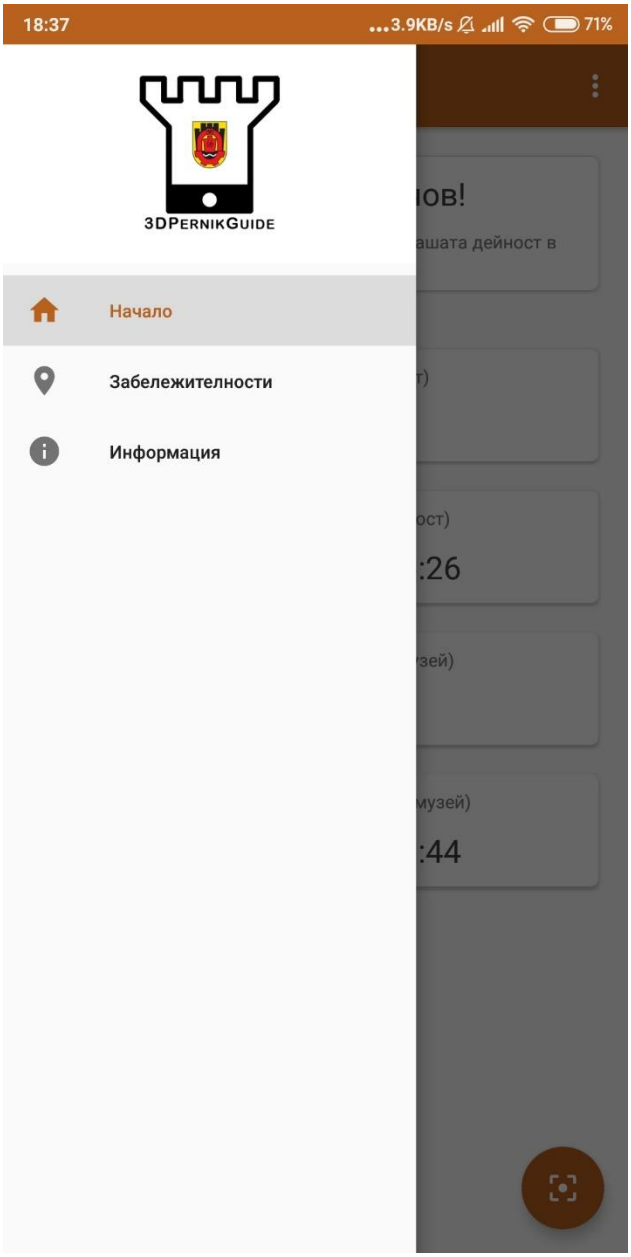

19:39
...0.0KB/s 46%


20:10
...0.0KB/s 39%
Крепостна стена
Обект:
Крепостна стена
Информация:
Оградени от крепостните стени са били около 50 декара. Каменната стена, обграждаща крепостта, е изградена от две зидарии с ширина 1 м и върви по естествените очертания на хълма, върху който някога се е разполагал центърът на административна област Перник. Тя е с обща дължина около 800 м. За съжаление при възстановяването на стената, новата зидария е вървяла ту по външната, ту по вътрешната зидария на стената и е загубен уникалният и автентичен изглед на крепостната стена. В североизточната част на крепостта се намира нейната главна порта, зад която започва и основна улица на „града“.
3D МОДЕЛ

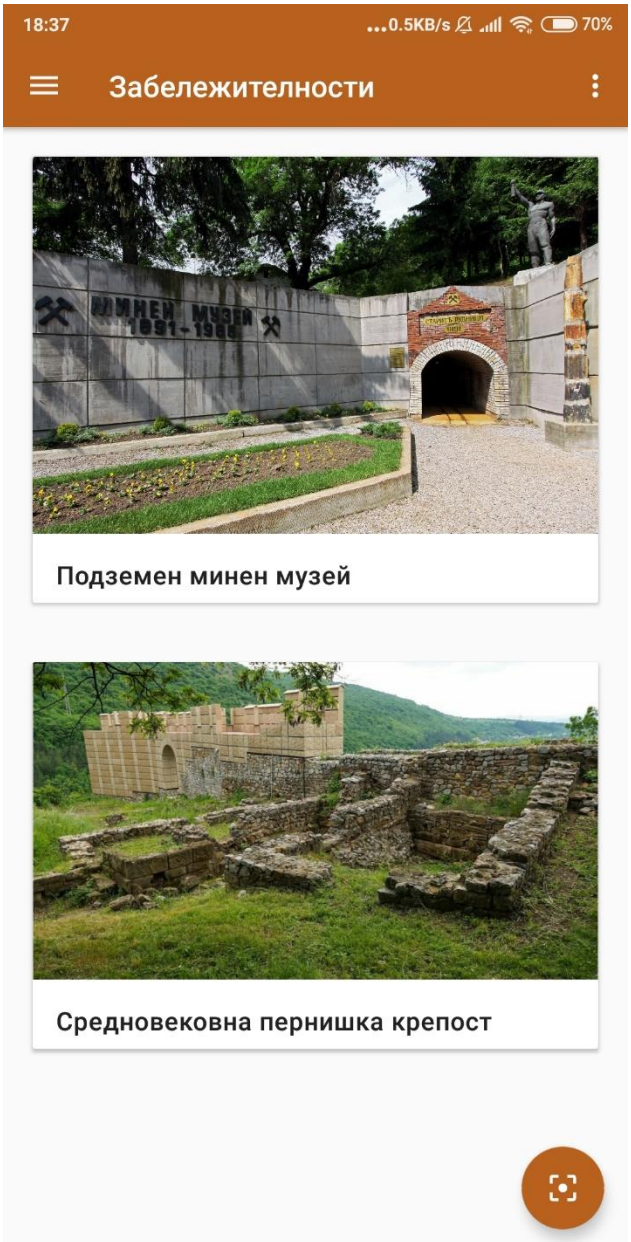

20:10
...0.0KB/s 39%

4.8 **Заключение:** Можем да заключим, че въпреки пренаситения пазар на софтуерни продукти, ориентирани към туристическите услуги, ние намерихме начин да бъдем уникални и все пак да решим реален проблем, като представим иновативен продукт. Проектът е изготвен от нашия екип със съдействието на Регионалния исторически музей. Приложението е безвъзмездно предоставено на РИМ – гр. Перник. Това е описано и в подписания договор за сътрудничество между директорите на Професионална гимназия по икономика и РИМ – Перник.

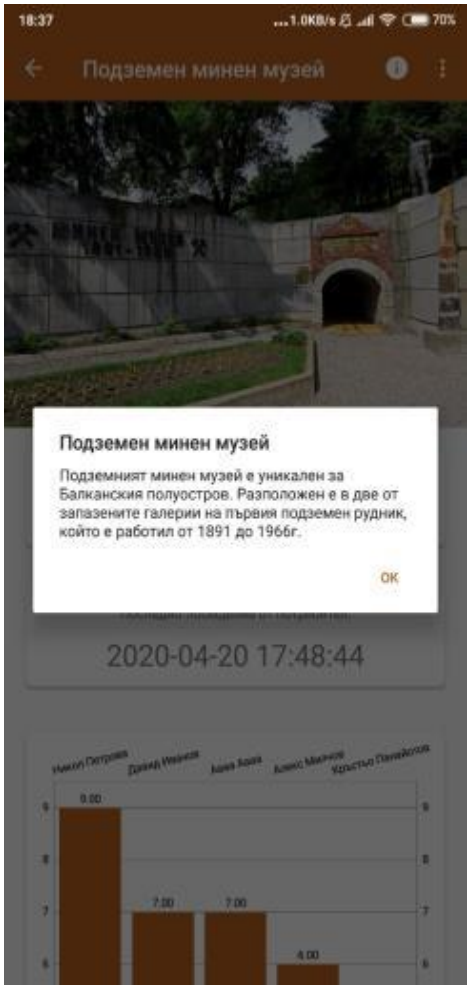
Навигационна система



Екран със забележителности



ПОДЗЕМЕН МИНЕН МУЗЕЙ



СРЕДНОВЕКОВНА ПЕРНИШКА КРЕПОСТ