Geekbrains

**"Разработка приложения для Android: система учета и подбора деталей конструктора"**

Разработчик — Android-разработка.

Техническая специализация

Слободчиков Сергей Игоревич

Ногинск

2024

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc165679990)

[**ВВЕДЕНИЕ** 4](#_Toc165679991)

[**1 Описание предметной области** 6](#_Toc165679992)

[**1.1 Классификация мобильных приложений** 6](#_Toc165679993)

[**1.2 Роль мобильных приложений в наши дни** 11](#_Toc165679994)

[**1.3 Обзор популярных операционных систем** 13](#_Toc165679995)

[**1.4 Раскрытие понятия «Собери конструктор из того что есть»** 15](#_Toc165679996)

[**1.5 Обзор аналогичных программных решений** 15](#_Toc165679997)

[**1.6 Техническое задание** 17](#_Toc165679998)

[**Выводы** 18](#_Toc165679999)

[**2 Выбор средств разработки** 19](#_Toc165680000)

[**2.1 Обзор и анализ ЯП** 19](#_Toc165680001)

[**2.2 Обзор и анализ СУБД** 21](#_Toc165680002)

[**2.3 Обзор и анализ сред разработки** 24](#_Toc165680003)

[**Выводы** 35](#_Toc165680004)

[**3 Проектирование** 36](#_Toc165680005)

[**3.1 Алгоритм работы приложения** 36](#_Toc165680006)

[**3.2 Разработка прототипа приложения** 37](#_Toc165680007)

[**3.3** **Настройка среды и создание проекта** 40](#_Toc165680008)

[**3.4** **Разработка приложения** 43](#_Toc165680009)

[**Выводы** 47](#_Toc165680010)

[**4 Планы по развитию приложения** 48](#_Toc165680011)

[**4.1 Завершить работу по алгоритму работы приложения**. 48](#_Toc165680012)

[**4.2 Планы по развитию** 49](#_Toc165680013)

[**Выводы** 49](#_Toc165680014)

[**ЗАКЛЮЧЕНИЕ** 50](#_Toc165680015)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 51](#_Toc165680016)

[**Приложение А. Листинг** 52](#_Toc165680017)

**ВВЕДЕНИЕ**

С развитием технологий жизнь современного человека заполонили все возможные мобильные устройства, роль которых сегодня сложно недооценить, ведь при помощи мобильного телефона можно в любой момент связаться со своими друзьями, родственниками или коллегами по работе, чтобы оперативно узнать или передать информацию. Помимо контактов на телефоне хранится масса другой информации: записывают свои идеи и мысли, номера кредитных карт и другие личные данные. А все возможные полезные программы, функции и опции, которыми оснащены современные сотовые телефоны, делают это небольшое по размеру устройство настолько многофункциональным, что, невозможно ими не восхищаться.

Мобильные приложения, в свою очередь, играют огромную роль в развитии современных информационных технологий посредством автоматизации многих процессов, связанных с упрощением рутинной деятельностью человека.

Для людей в деловой сфере польза сотового устройства заключается в возможности вести необходимые переговоры, достигать договоренности в любом месте и в любое время. Легко представить насколько бы упала продуктивность работа компаний, если бы предприниматели не имели бы возможности заключить контракт, дать распоряжение или решить какие-либо иные вопросы, находясь вне офиса.

Тема данной дипломной работы: Разработка мобильного приложения для смартфона «Набор конструктора».

Цель работы: разработать мобильное приложение, которое позволило бы при вводе количества имеющихся деталей конструктора подбирать наборы конструктора, которые вы можете собрать сейчас или получить сведения каких деталей не хватает чтобы собрать набор конструктора.

Задачи:

* проанализировать предметную область;
* провести анализ и выбор сред и средств разработки;
* выполнить проектирование и реализацию мобильного приложения.

Предмет работы: Разработка мобильного приложения.

Объект работы: Разработка мобильного приложения «Набор конструктора».

Актуальность данной работы обуславливается тем, что смартфон, в настоящее время, является одним из самых прогрессивных и необходимых устройств. Он может заменить даже компьютер или ноутбук при условии, если человек находится не дома, а ему нужно срочным образом внести правки в текстовом документе или графическом редакторе. Поэтому разработка мобильного приложения является востребованной и полезной задачей.

**1 Описание предметной области**

**1.1 Классификация мобильных приложений**

Приступая к разработке мобильных приложений важно иметь представление о том, какие виды приложений существуют. Если имеется представление о том, к какому типу относится приложение, то понятно на какие моменты в процессе его разработки необходимо обращать основное внимание. Можно выделить следующие виды приложений – рисунок 1.1:

Рисунок 1.1 – Классификация мобильных приложений

* Приложения переднего плана — это приложения, которые выполняют свои функции только тогда, когда они видимы на экране. В противном случае их выполнение приостанавливается. Такими приложениями являются, например, игры, текстовые редакторы, видеопроигрыватели. При разработке таких приложений необходимо очень внимательно изучить жизненный цикл активности, чтобы переключения в фоновый режим и обратно проходили гладко (бесшовно), т. е. при возвращении приложения на передний план было незаметно, что оно вообще куда-то пропадало. Для достижения этой гладкости необходимо следить за тем, чтобы при входе в фоновый режим приложение сохраняло свое состояние, а при выходе на передний план восстанавливало его. Еще один важный момент, на который обязательно надо обратить внимание при разработке приложений переднего плана, удобный и интуитивно понятный интерфейс.
* Фоновые приложения после настройки не предполагают взаимодействия с пользователем, большую часть времени находятся и работают в скрытом состоянии. Примерами таких приложений могут служить, службы экранирования звонков, SMS-автоответчики. В большинстве своем фоновые приложения нацелены на отслеживание событий, порождаемых аппаратным обеспечением, системой или другими приложениями, работают незаметно. Можно создавать совершенно невидимые сервисы, но тогда они будут неуправляемыми. Минимум действий, которые необходимо позволить пользователю: санкционирование запуска сервиса, настройка, приостановка и прерывание его работы при необходимости.
* Смешанные приложения большую часть времени работают в фоновом режиме, однако допускают взаимодействие с пользователем и после настройки. Обычно взаимодействие с пользователем сводится к уведомлению о каких-либо событиях. Примерами таких приложений могут служить мультимедиа-проигрыватели, программы для обмена текстовыми сообщениями (чаты), почтовые клиенты. Возможность реагировать на пользовательский ввод и при этом не терять работоспособности в фоновом режиме является характерной особенностью смешанных приложений. Такие приложения обычно содержат как видимые активности, так и скрытые (фоновые) сервисы, и при взаимодействии с пользователем должны учитывать свое текущее состояние. Возможно потребуется обновлять графический интерфейс, если приложение находится на переднем плане, или же посылать пользователю уведомления из фонового режима, чтобы держать его в курсе происходящего. И эти особенности необходимо учитывать при разработке подобных приложений.
* Виджеты – небольшие приложения, отображаемые в виде графического объекта на рабочем столе. Примерами могут служить, приложения для отображения динамической информации, такой как заряд батареи, прогноз погоды, дата и время. Разумеется, сложные приложения могут содержать элементы каждого из рассмотренных видов. Планируя разработку приложения, необходимо определить способ его использования, только после этого приступать к проектированию и непосредственно разработке [5].
* Мобильные сайты, веб-приложения – являются наиболее распространенным типом приложений, предназначенных для мобильных устройств. С помощью смартфона нынешних поколений можно посмотреть обычный сайт. Благодаря веб-приложению возможно в краткие сроки и за небольшую стоимость получить весомый результат, заключающийся в привлечении большего числа клиентов сайта и количества посещений.
* Нативное приложения – это приложение, доступное лишь на одной платформе и не доступное ни на каких других, например, приложение, которое доступно только для iPhone или телефона на базе Android. Такие приложения считаются самим ресурсоемкими, но также они дают возможность пользоваться функциями, предлагаемыми отдельными операционными системами, в максимальном объеме. В результате нативные приложения занимают первенство среди прочих видов приложений не только по функциональным особенностям, но также по скорости загрузки.
* Гибридные приложения – это смесь нативных и веб-приложений. Они устанавливаются как обычные приложения, но при этом используют WebView для загрузки веб-страниц. Это позволяет им иметь доступ к данным устройства и оставаться гибкими, не требуя установки отдельного браузера. Возможность использовать push-уведомления, возможность размещения в Google Play и App Store. При этом главный контент является платформа независимой страничкой с версткой, которая размещена на сервере. Благодаря этому можно внести косметические изменения в продукт без необходимости выпускать новую версию, ведь достаточно просто разместить проведенные изменения на сервер.[10]

Если говорить о разработке мобильных приложений под платформу Apple iOS, то нативным приложением будет называться решение, разработанное на языке программирования Objective-c или Swift (язык программирования Apple для разработки приложений под iOS и OS X, представленный на WWDC 2014), в рамках интегрированной среды разработки Xcode. С точки зрения пользователя, нативное приложение ничем не отличается от качественно выполненного кросс-плаформенного приложения, но все же стоит отметить, что нативные приложения под конкретную мобильную операционную систему, обычно, имеют понятный пользователю (нативный) интерфейс.

Удобство заключается в том, что при разработке нативных приложений связка UI/UX специалиста и программиста наиболее эффективна: дизайнер точно знает какие UI подходы привычны пользователям конкретной операционной системы, например кнопка «Back» и нижний Tab Bar в iOS, а iOS разработчик точно знает как реализовать те или иные UI фишки именно для мобильных устройств производства Apple. Как результат, пользователь открывает нативное приложение и интуитивно взаимодействует с новым интерфейсом.

Важно понимать, почему приложение называется кросс-платформенным. Дело в том, что, на самом деле, сам исполняющий файл, скомпилированный под одну из мобильных платформ, не может быть запущен в другой ОС. Т.е. если было разработано кросс-платформенное решение и скомпилировано под Android, это не означает, что можно взять файл с расширением «\*.apk» и запустить его на iPad’e. Идея кросс-платформенных решений не в удобстве для пользователя, а в оптимизации процессов разработки мобильного приложения.

Следовательно, кросс-платформенное приложение – это решение разработанное таким образом, чтобы иметь возможность, с минимальными усилиями, скомпилировать исходный код для исполнения на нескольких мобильных платформах, но результатом каждой отдельной компиляции будут отдельные исполняемые файлы. Например, под iOS исполняемый файл имеет расширение – «\*.ipa», под Android – «\*.apk» и т.д. [11]

Гибридные приложения – чаще всего это приложения, при разработке которых используются веб-технологии, а не разработка в нативной среде (пр. связка Xcode и Objective-c в случае разработки под iOS). В результате получается веб-приложение, которое запускается в «обертке» и подается не как веб-страница, а как отдельное приложение, требующее установки и имеющее отдельную иконку.

Гибридные решения достаточно популярны среди разработчиков кросс-платформенных движков, обуславливается это тем, что с функцией веб-браузера может одинаково справиться практически любая мобильная ОС, а значит если приложение уже работает под какой-либо мобильной OS, то запустить ее на другой не составит труда. Примером среды для создания таких приложений является PhoneGap [1].

**1.2 Роль мобильных приложений в наши дни**

Роль мобильных приложений в наши дни очень велика. Трудно представить себе современного человека в возрасте от 16 до 60, у которого не было бы мобильного телефона или смартфона.

С каждым днем мобильные приложения становятся популярнее среди рядовых интернет-пользователей. Причем приложения могут быть совершенно разными – от развлекательных и информационных программ, до мобильных интернет-магазинов и мобильного банкинга. Владельцы интернет-магазинов используют мобильные приложения как важный инструмент маркетинга, поддерживая с их помощью постоянную связь со своими клиентами.

Любое мобильное приложение – это визитка, которая вручается постоянным клиентам, и оно должно решать одну из трех задач пользователя:

* увлекательное времяпровождение;
* осуществление доступа в Интернет за неимением других способов;
* возможность получить нужную информацию в короткий промежуток времени.

С развитием технологий нашу жизнь заполонили все возможные мобильные устройства, роль которых сегодня сложно недооценить, ведь при помощи мобильного телефона мы можем в любой момент связаться со своими друзьями, родственниками или коллегами по работе, чтобы оперативно узнать или передать информацию. Помимо контактов на телефоне хранится масса другой информации: записывают свои идеи и мысли, номера кредитных карт и другие личные данные. А все возможные полезные программы, функции и опции, которыми оснащены современные сотовые телефоны делают это небольшое по размеру устройство настолько многофункциональным, что, невозможно ими не восхищаться.

Мобильные приложения, в свою очередь играют огромную роль в развитии современных информационных технологий посредством автоматизации многих процессов связанных с упрощением рутинной деятельностью человека.

Для людей в деловой сфере польза сотового устройства заключается в возможности вести необходимые переговоры, достигать договоренности в любом месте и в любое время. Легко представить насколько бы упала продуктивность работа компаний, если бы предприниматели не имели бы возможности заключить контракт, дать распоряжение или решить какие либо иные вопросы, находясь вне офиса.

По данным исследовательской компании Statista[9], в четвёртом квартале 2023 года Android сохранила свои позиции ведущей мобильной операционной системы в мире с долей рынка 70,1 %. Ближайший конкурент Android — iOS от Apple — занимал рыночную долю в 29,2 %. Отрыв лидера очевиден и велик.

**1.3 Обзор популярных операционных систем**

Индустрия компьютерных технологий считается одной из наиболее прогрессивных в современном мире. больше половины всего населения нашей планеты вне зависимости возраста и профессии, ежедневно пользуется гаджетом (мобильным телефоном, планшетом), компьютером или ноутбуком. Все аппаратные и программные процессы, заложенные в вышеперечисленных устройствах, управляются посредством комплекса единиц программного обеспечения, известного как операционная система (ОС) Генезис современных операционных систем начался в конце 1940-х годов. В это время начали разрабатываться служебные программы (загрузчики и мониторы), а также библиотеки часто используемых подпрограмм. Одной из самых первых операционных систем стала GM-NAA I/O, разработанная в 1955 году для компьютера IBM 704. 13 В 1960–1965 годах сформировались и были реализованы основные идеи, определяющие функциональность ОС: пакетный режим, разделение времени и многозадачность, разделение полномочий, реальный масштаб времени, файловые структуры и файловые системы.. Сегодня в число самых распространенных семейств операционных систем входят Windows, Linux, Android, iOS.

Вот некоторые операционные системы, которые пользуются наибольшей популярностью на четвертый квартал 2023 года:

* Android: 45,25%;
* iOS/iPadOS: 27,02%;
* Windows: 23,51%;
* macOS: 0,18%;
* Linux: 0,18%;
* Chrome OS: 0,036%;
* Unix: 0,002%;
* Windows Server: 0,0007%;
* FreeBSD: 0,0002%;
* IBM OS/400: 0,0002%.

Android: Это самая популярная мобильная операционная система, которая используется на большинстве смартфонов и планшетов. Она предлагает большой выбор приложений и гибкость настройки. Android также активно развивается и обновляется, что делает его привлекательным выбором для многих пользователей.

iOS: Это операционная система для мобильных устройств компании Apple, таких как iPhone и iPad. Она известна своим интуитивно понятным интерфейсом и высоким уровнем безопасности. iOS предлагает широкий выбор приложений и обновляется регулярно, что обеспечивает безопасность и актуальность устройства.

Windows: Это самая распространенная операционная система, разработанная компанией Microsoft. Она предоставляет широкий спектр функций и возможностей, включая поддержку множества приложений и игр. Windows обладает интуитивно понятным интерфейсом и поддерживает множество аппаратных платформ.

macOS: Это операционная система, созданная компанией Apple для своих компьютеров Macintosh. Она отличается высокой стабильностью и безопасностью, а также имеет уникальный интерфейс и набор встроенных приложений. macOS интегрирована с другими продуктами Apple, такими как iPhone и iPad, что обеспечивает удобство использования.

Linux: Это семейство операционных систем с открытым исходным кодом. Оно широко используется в серверных решениях благодаря своей надежности и гибкости. Существуют также версии Linux для настольных компьютеров, такие как Ubuntu и Mint. Linux известен своей стабильностью, безопасностью и возможностью тонкой настройки под нужды пользователя.

**1.4 Раскрытие понятия «Собери конструктор из того что есть»**

В настоящее время, подавляющее большинство людей имеет смартфон, поэтому мобильные приложения на нём могут являться удобными инструментами. Конструирование из деталей — это процесс создания различных моделей и конструкций из заранее подготовленных компонентов. Он требует от человека творческого подхода, логического мышления и умения работать руками.

Понятие «собери конструктор из того что есть» подразумевает использование тех деталей, которые уже имеются в наличии. То есть, если у вас есть определенный набор деталей, то ваша задача — придумать, как их можно соединить между собой таким образом, чтобы получить нужную модель или конструкцию. И в этом может помочь приложение.

В данной работе будет рассмотрена разработка мобильного приложения «Набор конструктора». Данное приложение должно показывать какие наборы конструктора можно собрать из имеющихся деталей, каких деталей не хватает чтобы собрать набор.

**1.5 Обзор аналогичных программных решений**

Brickit — приложение, которое позволяет находить инструкции небольших Lego-проектов, исходя из имеющихся у пользователя деталей. Оно использует технологию дополненной реальности и может анализировать коллекцию с помощью камеры смартфона.



Рисунок 1.2 – приложение Brickit

Плюсы приложения Brickit:

* Упрощает сборку фигур из деталей конструктора Lego. Достаточно разложить все имеющиеся детальки в один слой перед смартфоном, и приложение за несколько секунд определит детали и предложит подходящие инструкции.
* Помогает вернуть к жизни выброшенные наборы Lego
* Развивает визуальное восприятие.

Минусы приложения Brickit:

* Некоторые детали на фото система может не узнать
* Наборы только маленькие

Brick Instructions – приложение которое содержит большое количество различных схем наборов лего до 2015 года.

Плюсы приложения Brick Instructions:

* Большой набор схем лего конструктора

Минусы приложения Brick Instructions:

* Давно не обновлялось
* Плохое разрешение

LEGO Builder. Приложение содержит в электронном виде все схемы сборки LEGO наборов, выпускаемых с 2015 года. В каждой схеме указан рекомендуемый возраст, количество деталей, год выпуска набора и артикул.

Плюсы приложения LEGO Builder:

* Позволяет искать и изучать полную библиотеку инструкций LEGO для наборов с 2000 года по сегодняшний день.
* Позволяет заниматься 3D-моделированием, масштабируя и вращая наборы конструкторов LEGO.
* Создаёт цифровую коллекцию своих конструкторов LEGO и отслеживает, сколько кубиков у вас в коллекции.
* Сохраняет прогресс сборки и продолжает инструкции LEGO с того места, на котором остановились

Минусы приложения LEGO Builder:

* Для использования этого приложения понадобится стабильное подключение к Интернету.

**1.6 Техническое задание**

Название: «Мобильное приложение под OS Android «Набор конструктора».

Краткое название: «Набор конструктора».

Назначение: помогает собирать конструкторы из имеющихся деталей.

Потенциальные пользователи: возрастная категория 10+.

Среда разработки: Android Studio 7

Совместимость с Android: Android 7. и старше.

Верстка телефон книжная: Да.

Верстка телефон альбомная: Да.

Верстка планшет Книжная: Адаптивная от телефона.

Верстка планшет Альбомная: Адаптивная от телефона.

Логика работы: Пользователь добавляет в приложение информацию о деталях конструктора, которые у него есть. Приложение анализирует полученную информацию и определяет, какие наборы конструктора можно собрать из имеющихся деталей. Результаты анализа отображаются пользователю в виде списка возможных наборов. Пользователь может выбрать интересующий его набор и получить дополнительную информацию о нем, такую как количество деталей, необходимых для сборки, возрастную категорию и т.д. Если у пользователя нет каких-то деталей для выбранного набора, приложение может предложить ему альтернативные варианты наборов, которые можно собрать из имеющихся деталей. Приложение также может предоставлять пользователю рекомендации по дополнению его коллекции деталей, основываясь на наборах, которые он может собрать.

Основной функционал:

* Ручной ввод информации о деталях;
* Анализ базы данных деталей пользователя для определения возможных наборов конструктора, которые можно собрать из имеющихся деталей.
* Отображение результатов анализа в виде списка возможных наборов с указанием количества деталей, необходимых для сборки каждого набора.
* Предоставление дополнительной информации о каждом наборе, такой как возрастная категория, количество деталей, сложность сборки и т.д.
* Рекомендации по дополнению коллекции деталей пользователя на основе уже имеющихся деталей и возможных наборов.
* Функция отслеживания наличия деталей и уведомлений о появлении новых наборов, которые можно собрать из имеющихся деталей.

Основные требования к устройству: смартфон с ОС Android 7.0 и выше.

**Выводы**

Роль мобильных приложений очень велика. Будь то мобильная версия сайта или самостоятельный программный продукт, игра, «читалка» или справочник автолюбителя – сфер и тем для созданий приложений великое множество.

Большей популярностью пользуются кроссплатформенные приложения, так как они, во-первых, облегчают работу программиста при их создании, во-вторых, охватывают большее число пользователей.

В рамках Дипломного проекта будет начата разработка приложения «Набор конструктора», которая поможет собирать конструкторы из имеющихся деталей.

**2 Выбор средств разработки**

**2.1 Обзор и анализ ЯП**

1) Java – строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Java – это язык программирования общего назначения, основанный на классах, объектно-ориентированный и спроектированный таким образом, чтобы иметь как можно меньше зависимостей реализации. Он предназначен для того, чтобы позволить разработчикам приложений писать код один раз, выполнять где угодно (WORA), что означает, что скомпилированный код Java может работать на всех платформах, поддерживающих Java, без необходимости перекомпиляции. Java-приложения обычно компилируются в байт-код, который может работать на любой виртуальной машине Java (JVM) независимо от базовой компьютерной архитектуры. Синтаксис в Java похож на C и C ++ , но она имеет меньше низкоуровневых объектов, чем любой из них. По состоянию на 2019 год , Java был один из самых популярных языков программирования в использовании в соответствии с GitHub, особенно для клиент-серверных веб-приложений, по мнению 9 миллионов разработчиков. Платформа: Android, Android Wear. Основная IDE: Android Studio.

2) Swift – язык, разработанный компанией Apple и предназначенный для разработки приложений под iOS и OS X. Swift заимствовал довольно многое из C++ и Objective-C. Платформа: iOS, macOS, watchOS и tvOS. Основная IDE: Xcode.

3) JavaScript – прототипно-ориентированный сценарный язык программирования. Наиболее широкое применение нашел в браузерах как язык сценариев для придания интерактивности веб-страницам, а также в кроссплатформенных фреймворках (React Native, Ionic, Sencha и т.п.). Платформа: iOS, Android, Tizen и практически любая другая. Основная IDE: Eclipse, блокнот Notepad++.

4) C# – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах группой инженеров в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework. В области разработки мобильных приложений и используется во фреймворке Xamarin. Платформа: iOS, Android, Windows 10. Основная IDE: Visual Studio.

5) Objective-C – компилируемый объектно-ориентированный язык программирования корпорации Apple, построенный на основе языка C и парадигм Smalltalk. Язык был создан Брэдом Коксом в начале 1980 и сейчас уже морально устарел, его заменяет новый и более простой Swift. Тем не менее, еще 3-5 лет разработчики на Objective-C будут очень востребованы на рынке. Платформа: iOS, macOS, watchOS и tvOS. Основная IDE: Xcode

6) Python. Строго говоря, Python не является языком мобильной разработки, а больше используется в веб-проектах. Однако в ближайшем будущем боты заменят приложения, а с помощью Python как раз и можно их создавать + язык широко используется и в других областях, например, в машинном обучении. К тому же он легко постигаем новичками, у него много библиотек, синтаксис легко читаемый и довольно аккуратный код. Платформа: боты. Основная IDE: Eclipse, Visual Studio, PyCharm.

7) Kotlin является кросс-платформенным, статический типизированным, объектно-ориентированным языком программирования общего назначения, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM. Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция и лучшая поддержка языка в IDE. Язык полностью совместим с Java, что позволяет java-разработчикам постепенно перейти к его использованию; в частности, в Android язык встраивается с помощью Gradle, что позволяет для существующего android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

В Kotlin нет поддержки неявных расширяющих преобразований для данных. Таким образом, меньшие типы не могут быть преобразованы в большие типы. В то время как Java поддерживает неявные преобразования, Kotlin требует выполнить именно явное преобразование. Ряд разработчиков воспринимает это как минус Kotlin. Инкапсуляция необходима в любой программе для достижения желаемого уровня управляемости. Посредством инкапсуляции, представление объекта может быть установлено исходя из того, как вызывающие стороны взаимодействуют с ним. Кроме того, можно изменить представление без необходимости изменения вызывающих абонентов, если публичный API остается неизменным. Неприватные поля или public поля в Java полезны в сценариях, где вызывающие объекты должны меняться в соответствии с их представлением. Это означает, что такие поля предоставляют представление объекта вызывающим объектам. У Kotlin нет не-private полей. Это достаточно интересное отличие при сравнении Kotlin и Java.

**2.2 Обзор и анализ СУБД**

PostgreSQL – это мощная объектно-реляционная база данных с открытым исходным кодом, которая использует и расширяет язык SQL в сочетании с множеством функций, которые безопасно хранят и масштабируют самые сложные рабочие нагрузки с данными. Истоки PostgreSQL восходят к 1986 году в рамках проекта POSTGRES в Калифорнийском университете в Беркли и более 30 лет активно развиваются на базовой платформе.

PostgreSQL заработал прочную репутацию благодаря своей проверенной архитектуре, надежности, целостности данных, надежному набору функций, расширяемости и приверженности сообщества открытого исходного кода, стоящего за программным обеспечением, для последовательной разработки эффективных и инновационных решений. PostgreSQL работает на всех основных операционных системах, соответствует требованиям ACID с 2001 года и имеет мощные надстройки, такие как популярный расширитель баз данных геопространственных данных PostGIS. Неудивительно, что PostgreSQL стала реляционной базой данных с открытым исходным кодом, которую выбирают многие люди и организации.

PostgreSQL имеет множество функций, направленных на то, чтобы помочь разработчикам создавать приложения, администраторам – защищать целостность данных и создавать отказоустойчивые среды, а также помогать управлять данными независимо от размера набора данных. PostgreSQL не только бесплатная и открытая, но и расширяемая. Например, можно определять свои собственные типы данных, создавать собственные функции и даже писать код на разных языках программирования без перекомпиляции базы данных.

PostgreSQL пытается соответствовать стандарту SQL, где такое соответствие не противоречит традиционным функциям или может привести к неправильным архитектурным решениям. Поддерживаются многие функции, требуемые стандартом SQL, хотя иногда и с немного другим синтаксисом или функцией. Со временем можно ожидать дальнейшего продвижения к соответствию. Начиная с версии 13, выпущенной в сентябре 2020 года, PostgreSQL соответствует по крайней мере 170 из 179 обязательных функций для соответствия SQL: 2016 Core.

SQLite – это система управления реляционными базами данных, похожая на Oracle, MySQL, PostgreSQL и SQL Server. Она реализует большую часть стандарта SQL, но в отличие от четырех упомянутых выше СУБД она не поддерживает модель «клиент-сервер». Скорее, она встроена в конечную программу. Это означает, что можно связать базу данных SQLite с приложением и получить доступ ко всем возможностям БД в своем приложении.

Данная СУБД совместима как с Android, так и с iOS, и каждое приложение может создавать и использовать базу данных SQLite. В Android контакты и медиа хранятся и ссылаются на БД SQLite. Она является наиболее используемой СУБД в мире и самым распространенным программным обеспечением.

SQLite — это внутрипроцессная библиотека, которая реализует автономный, безсерверный, транзакционный механизм базы данных SQL с нулевой конфигурацией. Это база данных с нулевой конфигурацией, что означает, что, как и другие базы данных, вам не нужно настраивать ее в вашей системе.

Движок SQLite не является автономным процессом, как другие базы данных, вы можете связать его статически или динамически в соответствии с вашими требованиями к вашему приложению.

Преимущества:

* SQLite получает доступ к своим файлам хранения напрямую.
* SQLite не требует отдельного серверного процесса или системы для работы (без сервера).
* SQLite поставляется с нулевой конфигурацией, что означает отсутствие необходимости в настройке или администрировании.
* Полная база данных SQLite хранится в одном межплатформенном файле на диске.
* SQLite очень маленький и легкий, менее 400 КБ полностью сконфигурирован или менее 250 КБ без дополнительных функций.
* SQLite является автономным, что означает отсутствие внешних зависимостей.
* Транзакции SQLite полностью совместимы с ACID, что обеспечивает безопасный доступ из нескольких процессов или потоков.
* SQLite поддерживает большинство функций языка запросов, представленных в стандарте SQL92 (SQL2).
* SQLite написан на ANSI-C и предоставляет простой и удобный API.
* SQLite доступен в UNIX (Linux, Mac OS-X, Android, iOS) и Windows (Win32, WinCE, WinRT).
* SQLite не требует отдельного серверного процесса или системы для работы (без сервера).
* SQLite поставляется с нулевой конфигурацией, что означает отсутствие необходимости в настройке или администрировании.
* Полная база данных SQLite хранится в одном межплатформенном файле на диске.
* SQLite очень маленький и легкий, менее 400 КБ полностью сконфигурирован или менее 250 КБ без дополнительных функций.
* SQLite является автономным, что означает отсутствие внешних зависимостей.
* Транзакции SQLite полностью совместимы с ACID, что обеспечивает безопасный доступ из нескольких процессов или потоков.
* SQLite поддерживает большинство функций языка запросов, представленных в стандарте SQL92 (SQL2).
* SQLite написан на ANSI-C и предоставляет простой и удобный API.
* SQLite доступен в UNIX (Linux, Mac OS-X, Android, iOS) и Windows (Win32, WinCE, WinRT).

**2.3 Обзор и анализ сред разработки**

1) AppyPie – единственная платформа, которая поможет всем тем, кто не знает или не хочет углубляться в программирование – рисунок 2.1. AppyPie – это отличная платформа для создания мобильных приложений, которые можно будет опубликовать в Google Play, Apple App Store, Windows App Store или в любом другом магазине. Самое приятное в Appy Pie заключается в том, что не придется писать ни единой строчки кода, а просто кликнуть мышью и добавить функции, и можно в любое время создать уникальное и индивидуальное приложение. Отличительной особенностью платформы является то, что она имеет сотни удивительных функций, и можно добавить их в свое мобильное приложение.

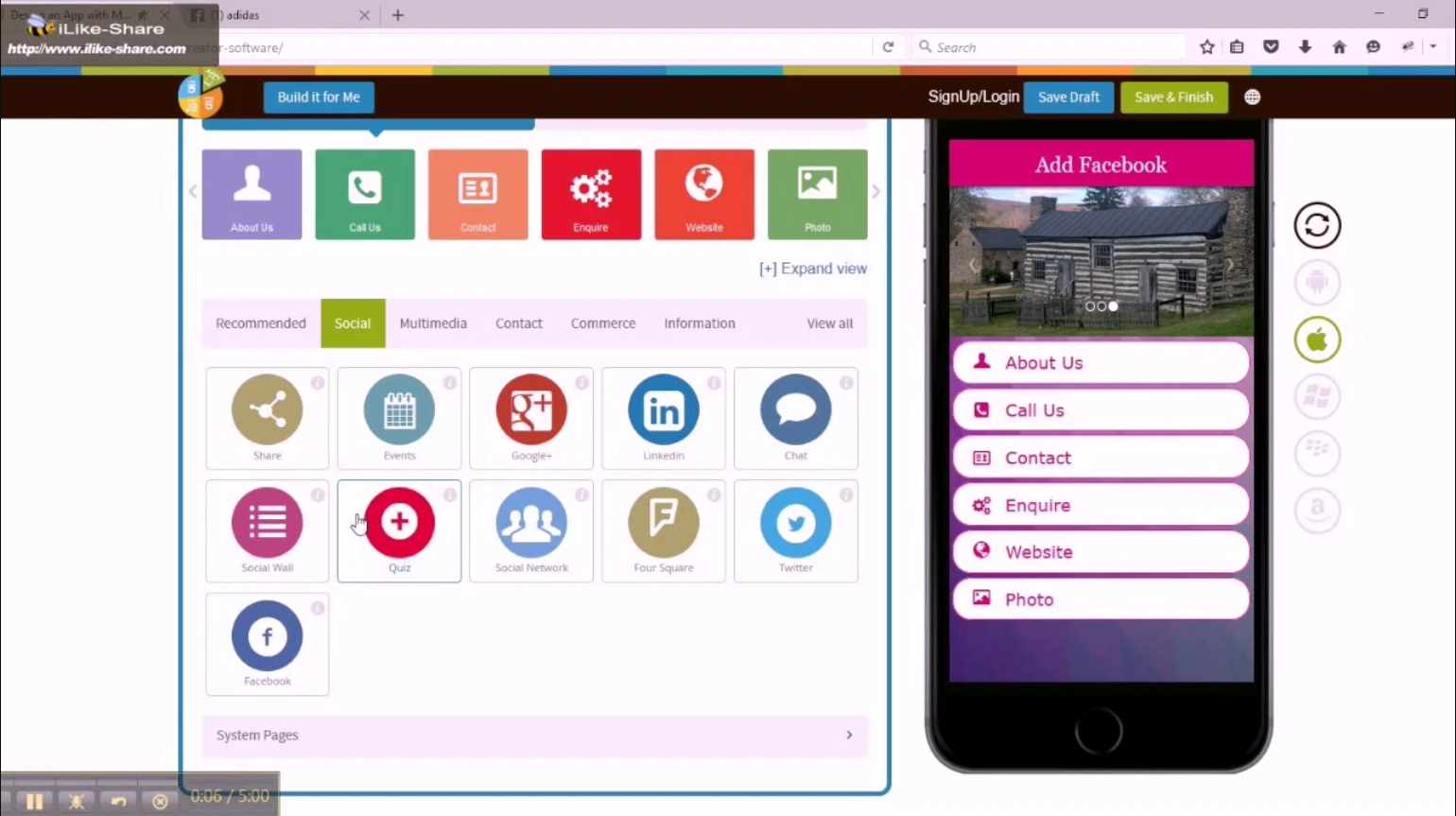


Рисунок 2.1 – AppyPie

2) PhoneGap – благодаря универсальности конкретно этого конструктора, он попадает в категорию востребованных кроссплатформенных инструментов разработки рисунок 2.2.

У этого инструмента есть свои преимущества, которые делают его привлекательным для современного разработчика, но самым полезным из них является возможность создания одного приложения, которое будет работать на всех мобильных устройствах. Приложения, созданные с помощью PhoneGap, имеют большой плюс – они совместимы с HTML5, CSS3 и JavaScript. Это прекрасное средство разработки от Adobe доступно для нескольких платформ, включая наиболее распространенные, а именно: Android, Windows и iOS.

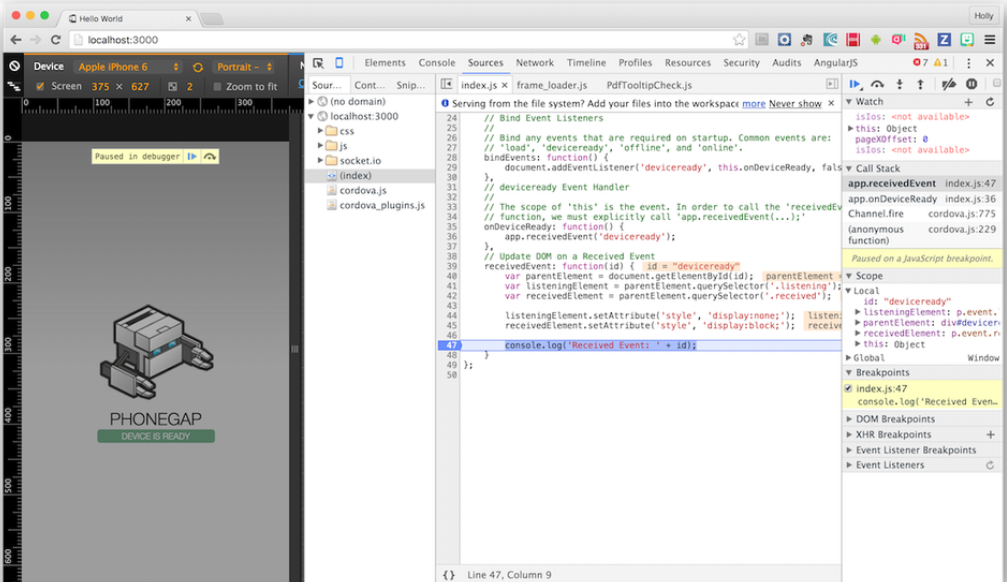


Рисунок 2.2 – PhoneGap

Помимо этих популярных платформ, работая с PhoneGap, разработчики могут также создавать приложения для устройств на Symbian, Palm, BlackBerry, iTouch и iPad. Данный универсальный инструмент уникален благодаря присущей ему гибкости, что помогает разработчикам создавать различные типы мобильных приложений при затрате значительно меньших усилий. Кроме того, разработчик, работающий с этим инструментом, может расширять функциональность и удобство использования мобильного приложения с помощью подключения, по мере необходимости, различных архитектур. Мобильные приложения, разработанные с помощью такого инструмента, как PhoneGap, обладают способностью работать безупречно на множестве платформ, поддерживая схожесть внешнего вида и ощущений от работы приложения. Фреймворк также усилен довольно мощной серверной частью, а это означает, что мобильные приложения, разработанные с помощью PhoneGap, обладают повышенной скоростью работы.

3) Xamarin, другой популярный инструмент для разработчиков приложений – это инструмент на C#. Фреймворк этого инструмента уникален тем, что позволяет разработчикам работать с собственными IDE (интегрированные среды разработки), API (интерфейсами прикладного программирования) и языками – рисунок 2.3.

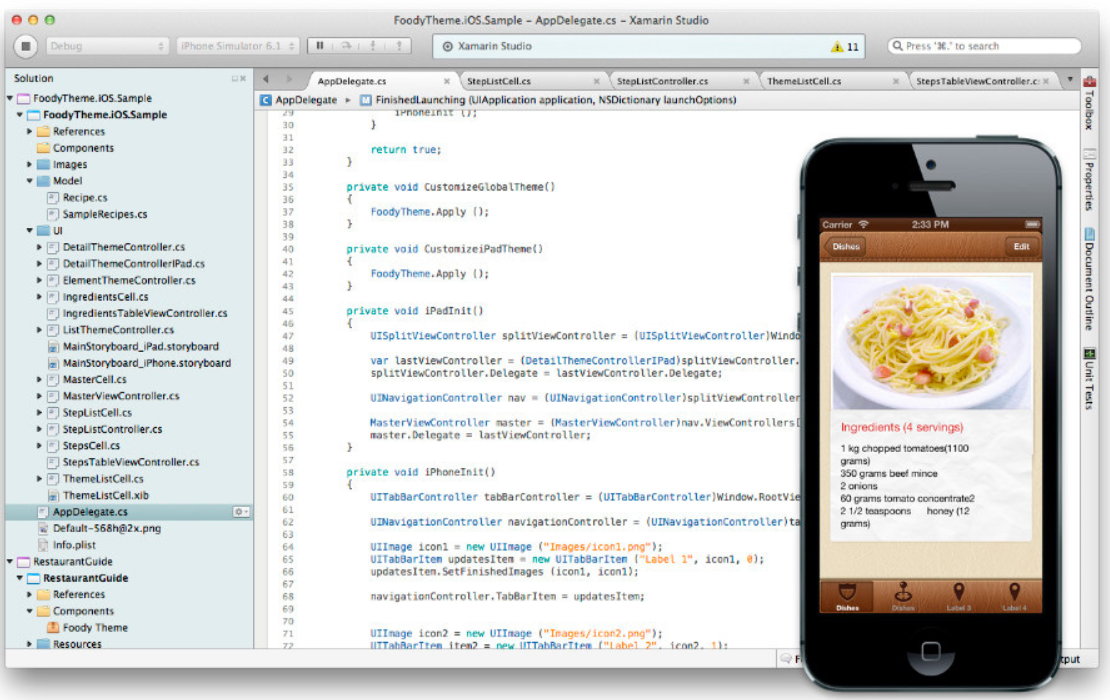


Рисунок 2.3 – Xamarin

Именно этот набор характеристик делает Xamarin предпочтительным выбором для нативных приложений. Уникальная функция, которая делает Xamarin инструментом для прибыльной разработки, состоит в том, что приложения, созданные с помощью этого инструмента, часто содержат меньше ошибок по сравнению с другими и, таким образом, обеспечивают намного более быстрое время выхода на рынок. У инструмента меньше программного кода, а это уменьшает общее количество ошибок. Это значит, что с помощью всего лишь одного теста код всех платформ может быть проверен с большой эффективностью. Философия Xamarin как инструмента заключается в совместной работе, поэтому инструмент распространяется для разработчиков с возможностью выбирать из ряда компонентов, некоторые из которых платные, а другие – бесплатны.

4) Appcelerator – ускоряет процесс разработки приложений, поскольку позволяет разработчикам создавать потрясающие приложения с меньшим количеством строк кода – рисунок 2.4.

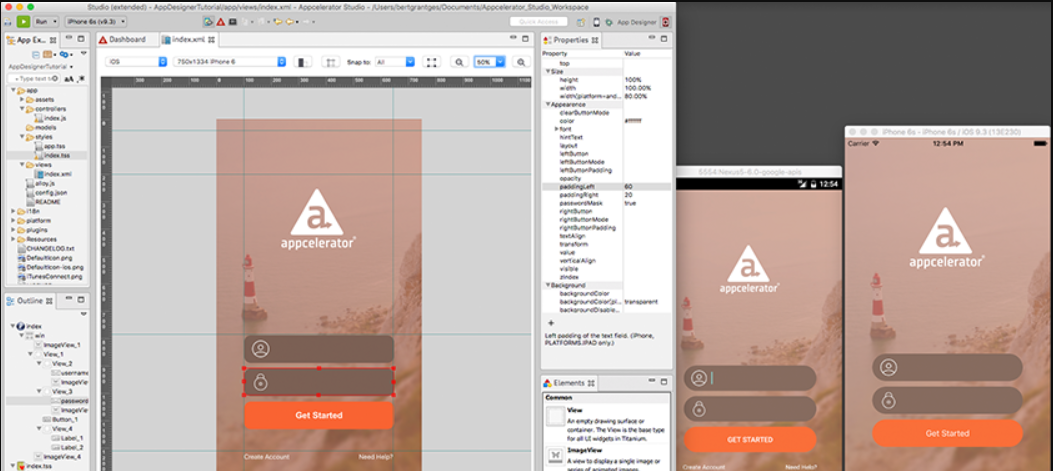
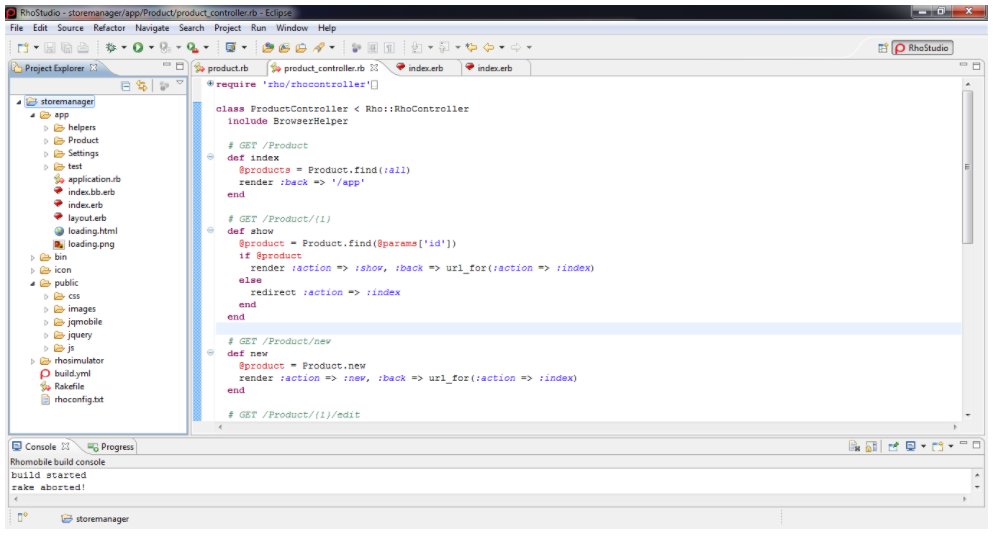


Рисунок 2.4 – Appcelerator

Appcelerator от Titanium является платформой разработки, которая пользуется огромной популярностью среди сообщества Twitterati, помогая разработчикам создавать собственные приложения для мобильных телефонов, планшетов и даже настольных компьютеров. Инструмент совместим с такими языками, как JavaScript, HTML, PHP, Ruby и Python. Инструмент не только популярен в Твиттере, хотя; Подсчитано, что этот инструмент ежемесячно становится основной более тысячи нативных приложений. Приложения, созданные с помощью этого инструмента, могут быть полностью аппаратными, предлагая возможность хранить все данные как на устройстве, так и в «облаке». Дизайнер приложений Appcelerator имеет встроенную функцию перетаскивания для легкого размещения объектов, а встроенная в него функция Hyperloop позволяет использовать JavaScript для получения прямого доступа к собственным API-интерфейсам на iOS и Android. Этот инструмент особенно подходит тем организациям, которые используют опыт веб-разработки при создании приложений для нескольких ОС.

5) RhoMobile – фреймворк с открытым исходным кодом, основанный на языке программирования Ruby, он является основой для RhoMobile – рисунок 2.5.

Рисунок 2.5 – RhoMobile

Именно Rhodes позволяет разработчику создавать нацеленные на обработку данных кросс-платформенные нативные приложения, которые будут совместимы с огромным спектром операционных систем и смартфонов. По своей основной структуре или архитектуре RhoMobile очень похож на PhoneGap. RhoMobile – это желательный инструмент для работы, поскольку он имеет одну определяющую черту – нужно написать код всего лишь один раз. Этот код затем можно использовать для создания мобильных приложений практически для всех основных смартфонов. В стремлении хранить только самые последние и самые свежие данные о мобильных телефонах пользователей, RhoMobile поставляется с встроенной средой разработки – RhoHub и RhoSync.

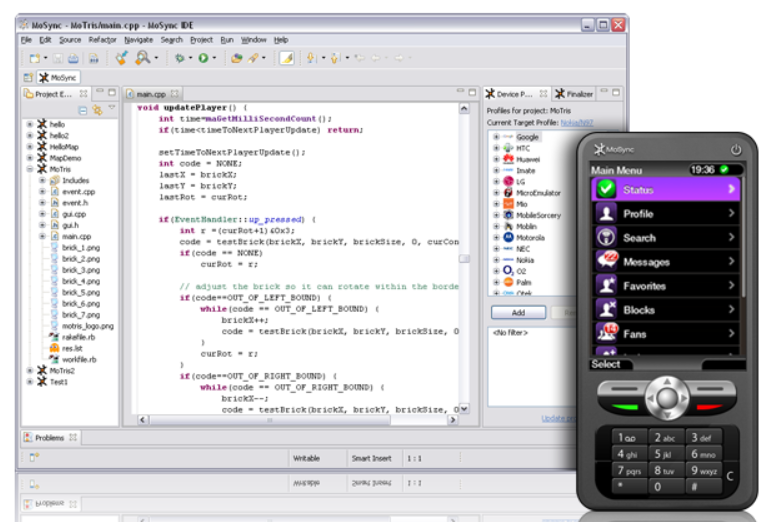
6) Фреймворк Ionic основан на языке Sass CSS и также поддерживает кросс-платформенность, что означает, что он может работать на различных операционных системах – рисунок 2.6.



Рисунок 2.6 – Ionic

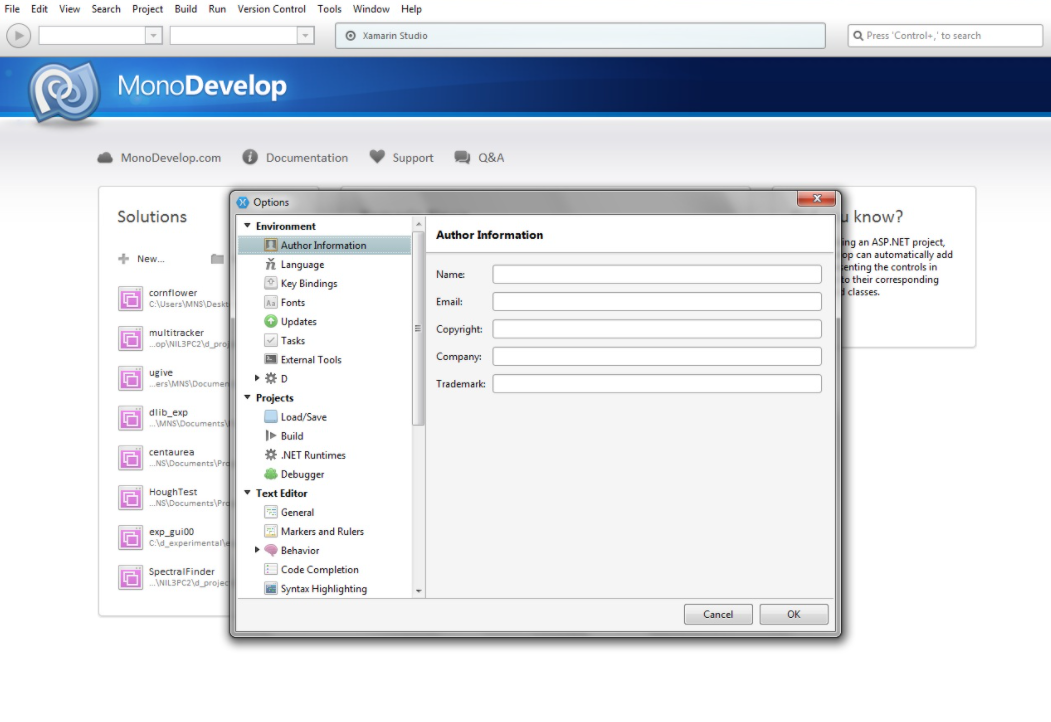
Ionic – один из наиболее простых в использовании инструментов, который можно интегрировать с Angular JS для разработки более сложных приложений. Ionic преимущественно используется для создания гибридных мобильных приложений. Так как это полный SDK с открытым исходным кодом (Software Development Kit), Ionic идеально подходит для создания гибридных мобильных приложений с использованием веб-технологий, включая, помимо прочего, CSS, HTML5 и SASS.

7) MoSync – отличный инструмент, основанный на стандартном веб-программировании, который также является набором для разработки программного обеспечения (SDK) для мобильных приложений – рисунок 2.7.

Рисунок 2.7 – MoSync

Этот инструмент довольно уникален, поскольку он оснащен библиотеками, компиляторами, профилями устройств, средами выполнения и другими полезными инструментами. MoSync совместим с языками программирования, такими как PHP, JavaScript, Python, Ruby и т.д. А недавно он также приобрел совместимость с интегрированной средой разработки на основе Eclipse для программирования на C/C++.

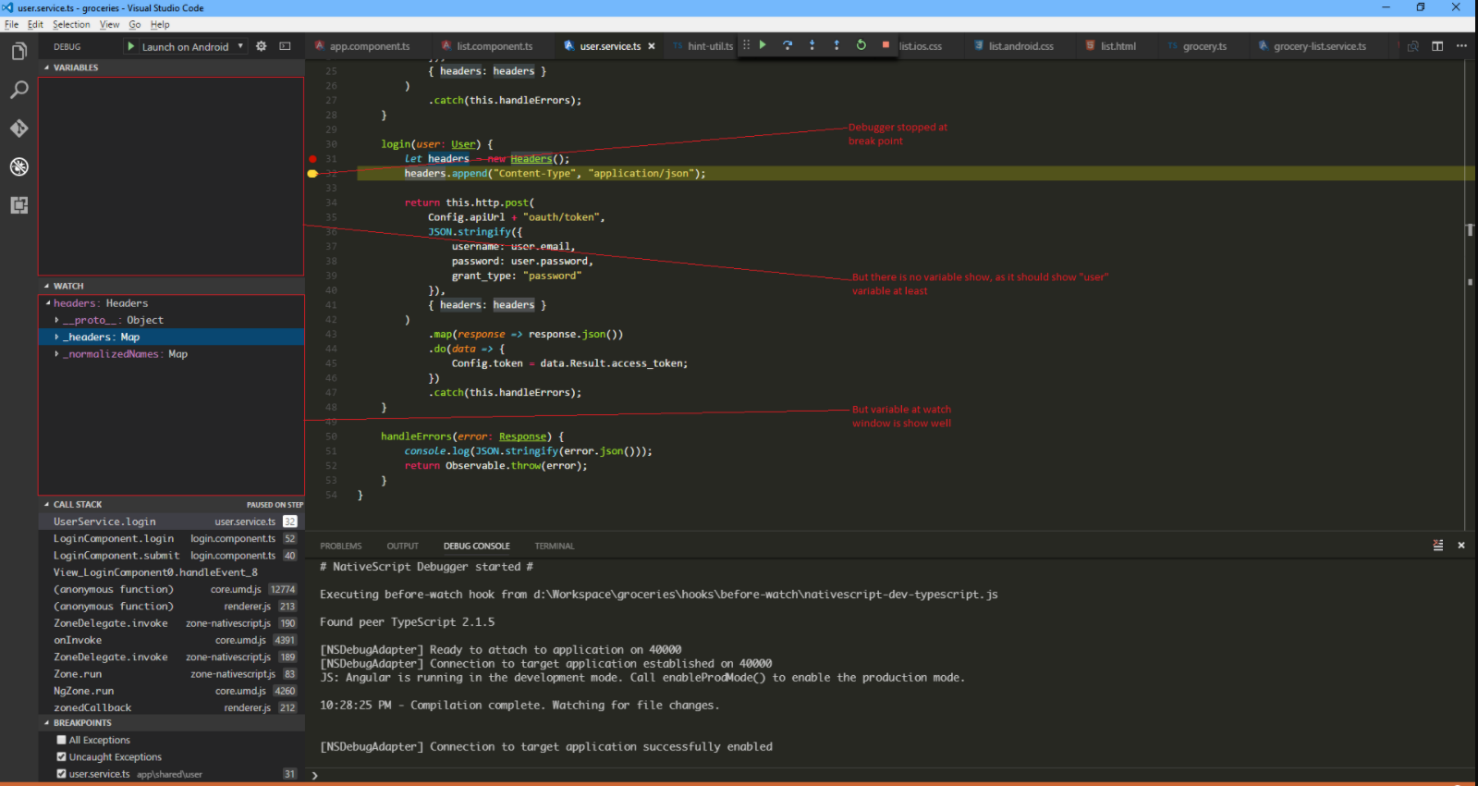
8) Monocross – еще один кросс-платформенный мобильный фреймворк с открытым исходным кодом, Monocross использует C#, Microsoft .NET и среду Mono для разработки приложений, которые поддерживаются на различных платформах, включая смартфоны и планшеты Android, iPad и iPhone, телефоны и устройства Windows с поддержкой Webkit – рисунок 2.8.

Рисунок 2.9 – Monocross

Одна из главных особенностей этого инструментария заключается в том, что он значительно упрощает перенос приложений.

9) NativeScript также является кросс-платформенным инструментом разработки с открытым исходным кодом, но, в отличие от многих других конкурентов, его можно использовать бесплатно – рисунок 2.9.

У инструмента даже нет многоуровневой системы с определенными уровнями Pro или Paid для расширенных опций. При работе с NativeScript разработчики могут использовать Angular, JavaScript или TypeScript и создавать мобильные приложения для Android и iOS. NativeScript интегрирован с Vue.JS, в дополнение к его способности поддерживать сотни плагинов для расширенной функциональности.

Рисунок 2.9 – NativeScript

10) Когда Kony объединен с какой-либо IDE, то он даёт разработчику возможность создавать приложения JavaScript, которые могут работать на всех платформах – рисунок 2.10.

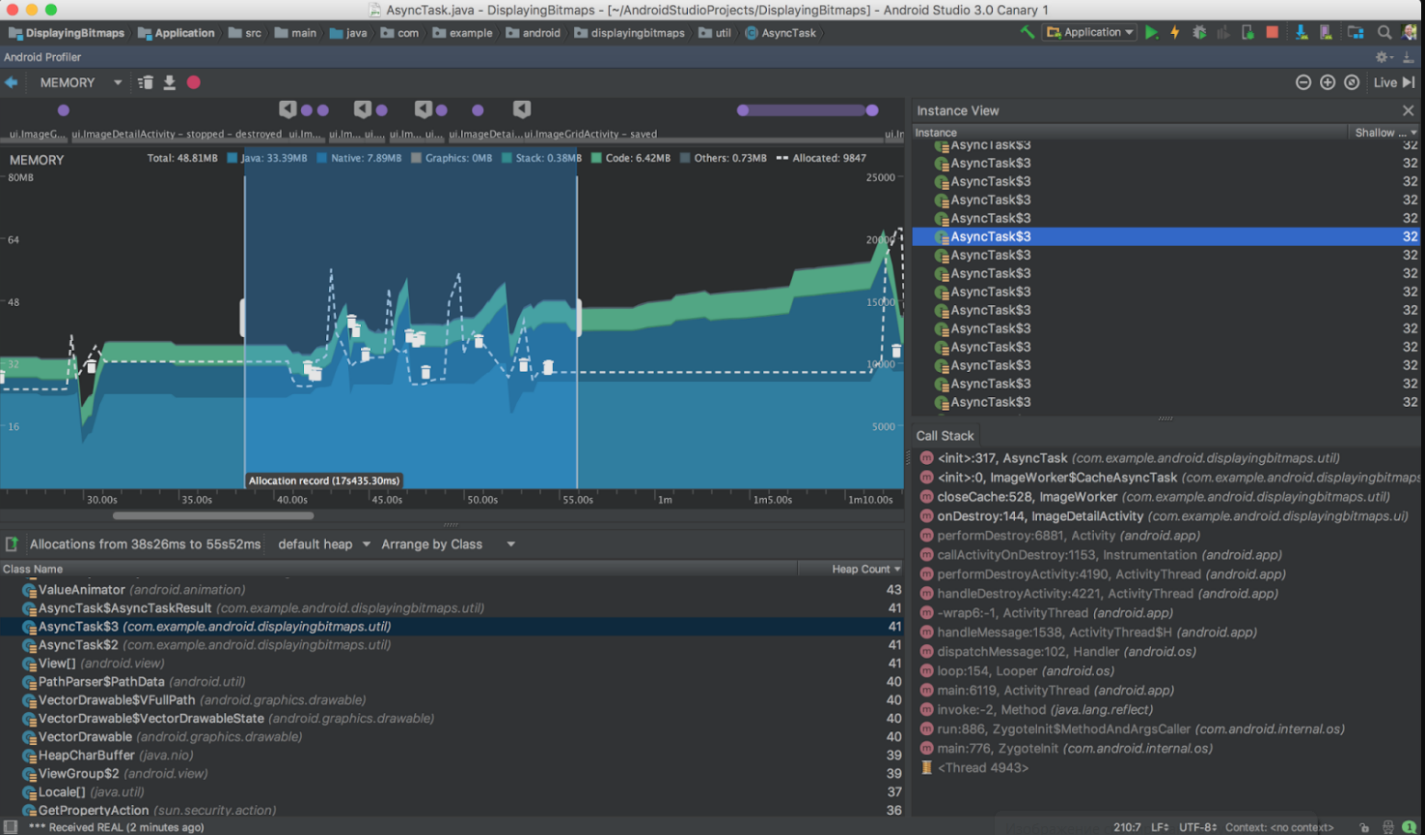


Рисунок 2.10 – Kony

Этот инструмент, однако, заставит немного потратиться, особенно если необходимо создать более одного приложения. Инструмент разработки приложений Kony поддерживает широкий спектр функций, включая управление API, чат-боты, голосовые сообщения, отчеты клиентов, дополненную реальность, справочные приложения и многие другие элементы. Хотя сам инструмент можно установить как в компьютерных системах Windows, так и в Mac, сопутствующие мобильные приложения используются для предварительного просмотра и тестирования на реальном устройстве, на котором он должен работать.

11) Android Studio является официальной интегрированной средой разработки (IDE) для разработки приложений для Android, основанной на IntelliJ IDEA – рисунок 2.11.

В дополнение к мощному редактору и инструментам разработчика IntelliJ, Android Studio предлагает еще больше возможностей, повышающих производительность при создании приложений для Android. Вместо использования предустановленных перспектив Android Studio следует контексту и автоматически создает соответствующие окна инструментов во время работы.

По умолчанию наиболее часто используемые окна инструментов прикрепляются к панели инструментов окна по краям окна приложения. Android Studio имеет три типа завершения кода, к которым можно получить доступ, используя сочетания клавиш. Android Studio автоматически применяет форматирование и стили, которые указаны в настройках стиля кода.

Но можно настроить параметры стиля кода с помощью языка программирования, включая указание условностей для вкладок и отступов, пробелов, фигурных скобок и пустых строк. Android Studio поддерживает различные системы управления версиями (VCS), включая Git, GitHub, CVS, Mercurial, Subversion и Google Cloud Source Repositories.

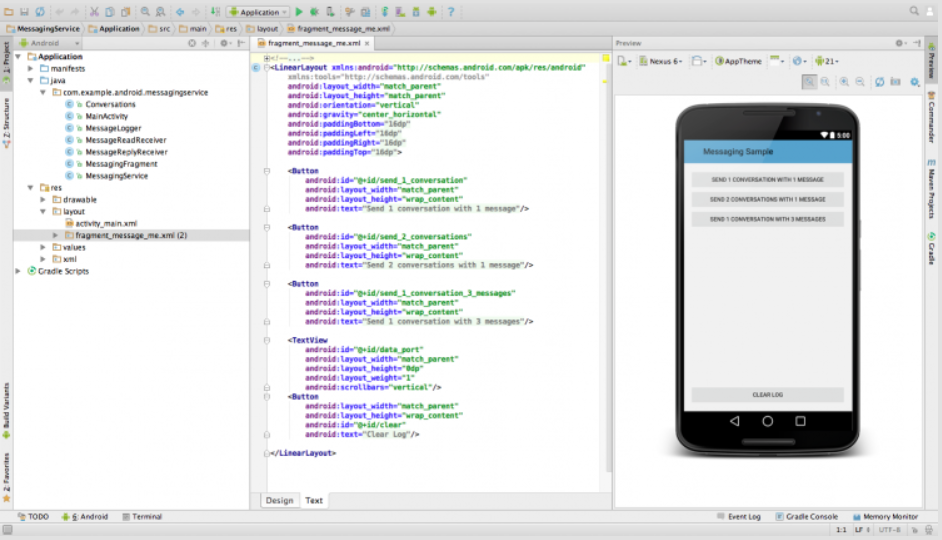


Рисунок 2.11 – Android Studio

**Выводы**

В данной главе были рассмотрены языки программирования, системы управления базами данных, подходящие для использования с Android и среды разработки. Для решения задач данной работы была выбрана платформа Android Studio, т.к. целью работы было создание приложения под Android, а не кроссплатформенного. Поэтому именно «родная» среда будет самой удобной. Не будет включать «лишние» библиотеки и минимизирует количество ошибок при компиляции с других языков (например, как C#).

После выбора IDE Android Studio, языком программирования был выбран Java. Основной выбор был между Java и Kotlin, И так как Kotlin более современный язык программирования и я начинаю его изучать я выбрал его.

**3 Проектирование**

**3.1 Алгоритм работы приложения**

Открытие приложения:

* При открытии приложения на первой странице пользователь видит выбор из трех кнопок:

1. Выбор имеющихся деталей LEGO.
2. Выбор имеющихся наборов LEGO.
3. Поиск наборов.

Выбор имеющихся деталей LEGO:

* При нажатии на эту кнопку пользователь видит список доступных деталей.
* Пользователь добавляет информацию о деталях конструктора, которые у него есть.
* После этого он возвращается на первую страницу.

Выбор имеющихся наборов LEGO:

* При нажатии на эту кнопку пользователь видит список доступных наборов.
* Пользователь добавляет информацию о наборах конструктора, которые у него есть.
* После этого он возвращается на первую страницу.

Поиск наборов:

* При нажатии на эту кнопку приложение анализирует полученную информацию о имеющихся деталях и наборах.
* Определяется, какие наборы конструктора можно собрать из имеющихся деталей.
* Также определяется, каких деталей не хватает для сборки других наборов.

Блок-схема представлена на рисунке 3.1.

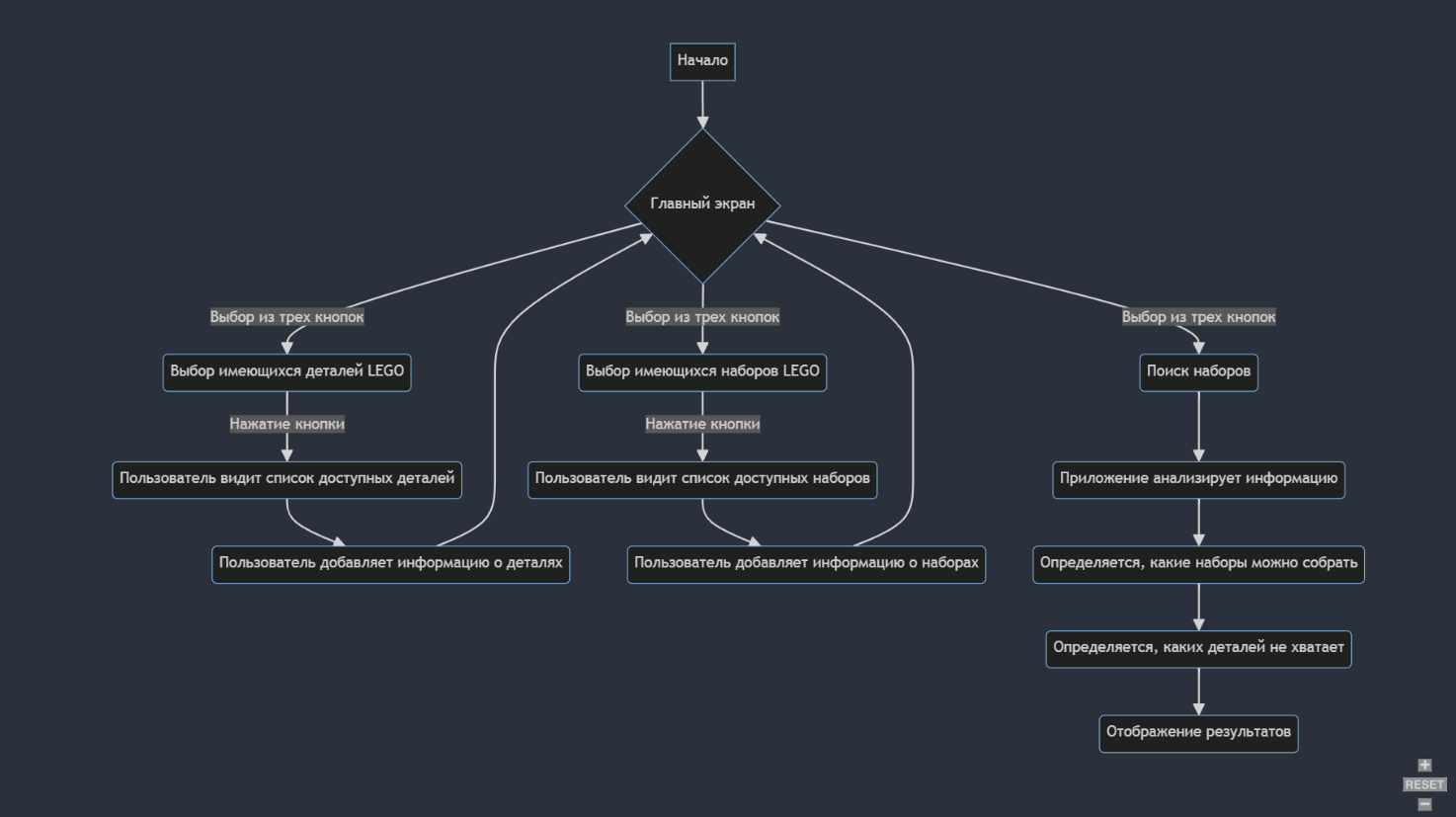
****

Рисунок 3.1 – Блок-схема работы приложения

**3.2 Разработка прототипа приложения**

Примерный интерфейс приложения представлен на рисунках 3.2 – 3.5 соответственно. Нажав на кнопку на главном экране Рисунок 3.2 “Выбор имеющихся деталей” или “Выбор имеющихся наборов” мы попадем на страницу с списком деталей или наборов Рисунок 3.3 введя информацию о имеющихся наборах или деталей подтверждаем и переходим на главный экран выбрав кнопку Поиск наборов мы получаем список наборов которые мы можем собрать и которым не хватает деталей. Нажав на этот набор мы перейдем на страницу с деталями из набора где узнаем каких деталей у нас не хватает рисунок 3.5.

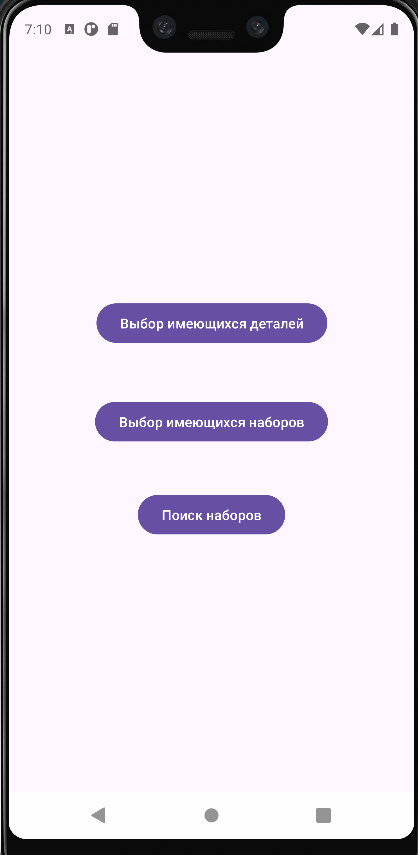


Рисунок 3.2 – Главный Экран

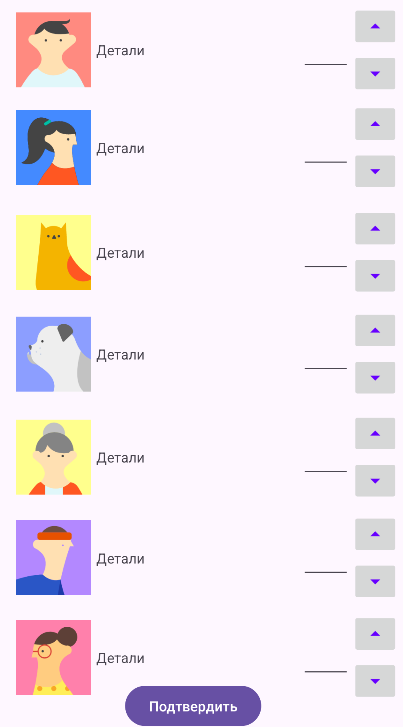


Рисунок 3.3 – Экран списка деталей и наборов

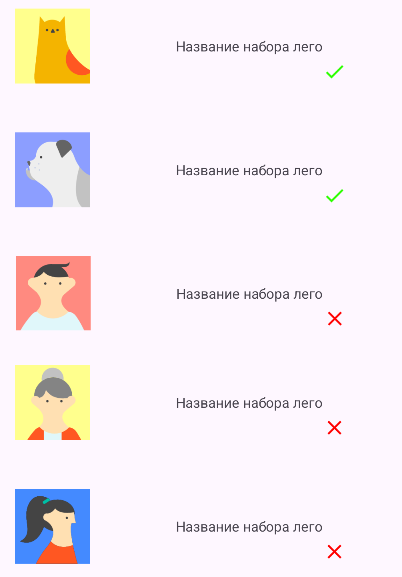


Рисунок 3.4 – Результаты наборов лего

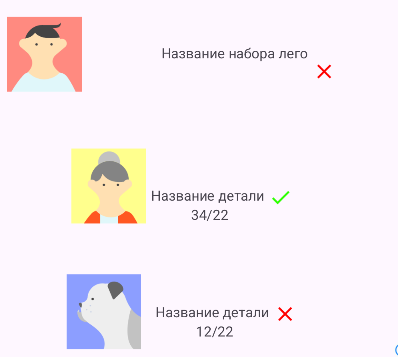


Рисунок3.5 - Экран деталей из набора

* 1. **Настройка среды и создание проекта**

Установка Android Studio:

Скачайте и установите Android Studio с официального сайта.

Запустите установщик и следуйте инструкциям. В одном установщике будет все необходимое — сама IDE, Android Emulator, Android SDK. То, чего нет в комплекте, инсталлятор докачает самостоятельно

Установка стандартная — обычный диалог инсталлятора. В процессе нужно будет ответить лишь на один важный вопрос, и то это опционально – рисунок 3.6

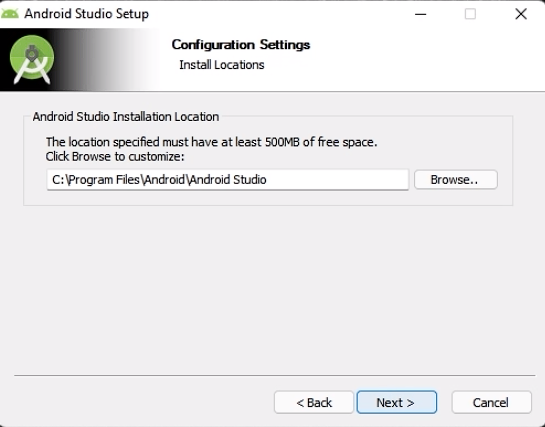


Рисунок 3.6 – Установка

После этого понадобится стандартно несколько раз нажать на кнопочку «далее», и на этом установка Android Studio завершена.

При первом запуске Android Studio задаст стандартный вопрос об импорте конфигурации – рисунок 3.7.

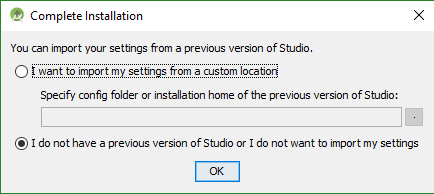


Рисунок 3.7 – Импорт конфигураций

 По умолчанию будет выбран тот же чекбокс, просто нажмите на кнопку «ОК». После этого Android Studio начнет качать Android SDK. Это может занять некоторое время. Если загрузка завершится неудачей, IDE предложит попробовать еще раз — обязательно нажмите «Retry»

После окончания загрузки нажмите «Finish».

В принципе, на этом установка закончена.

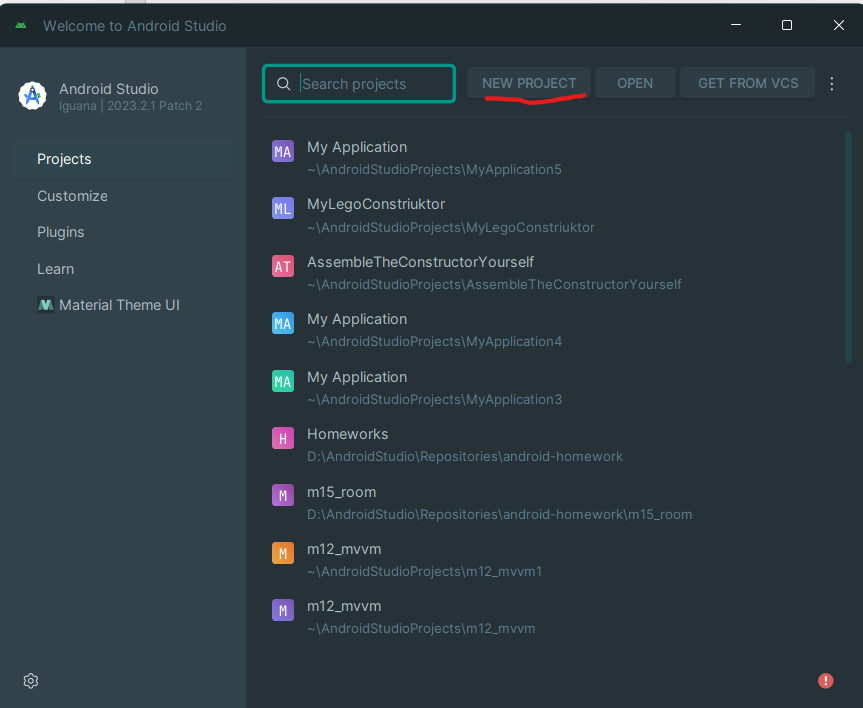
Для создания проекта нужно в главном окне Android Studio нажать на «NewProject» Там есть различные варианты базовых настроек для приложения  


Рисунок 3.7 – Создание проекта

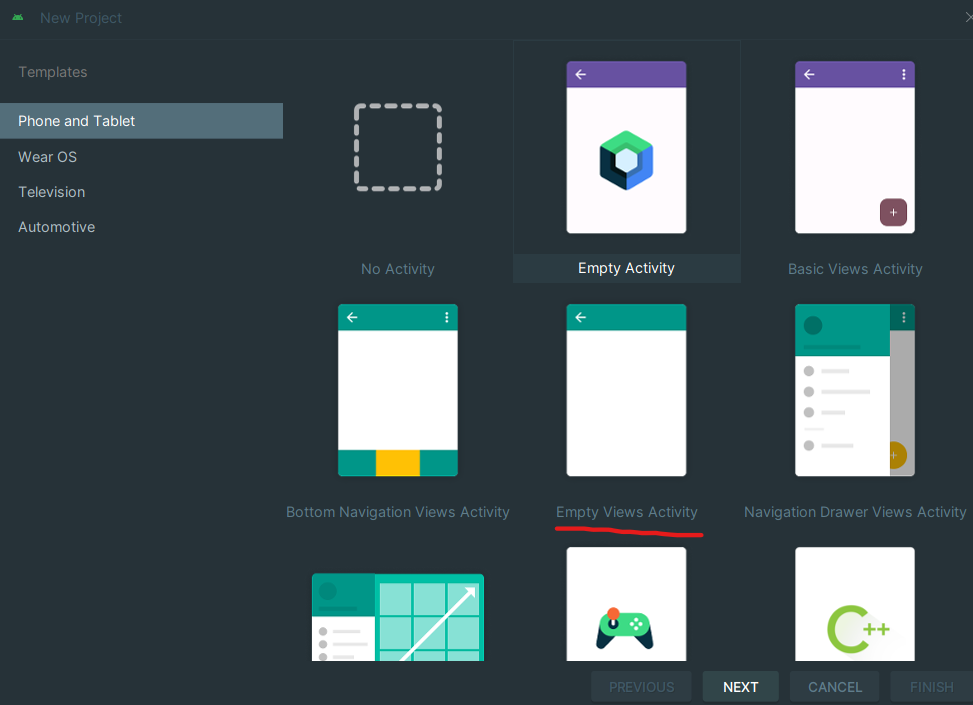


Рисунок 3.8 - Выбор компонентов

Мы выберем самое простое Empty Views Activity она пуста и позволяет работать с разметками xml. Дальше мы выбираем папку для хранения, имя нашего проекта и выбираем Minimum SDK и выбираем минимальную версию мы решили что это будет Android 7.0 ее поддерживает более 95% девайсов.

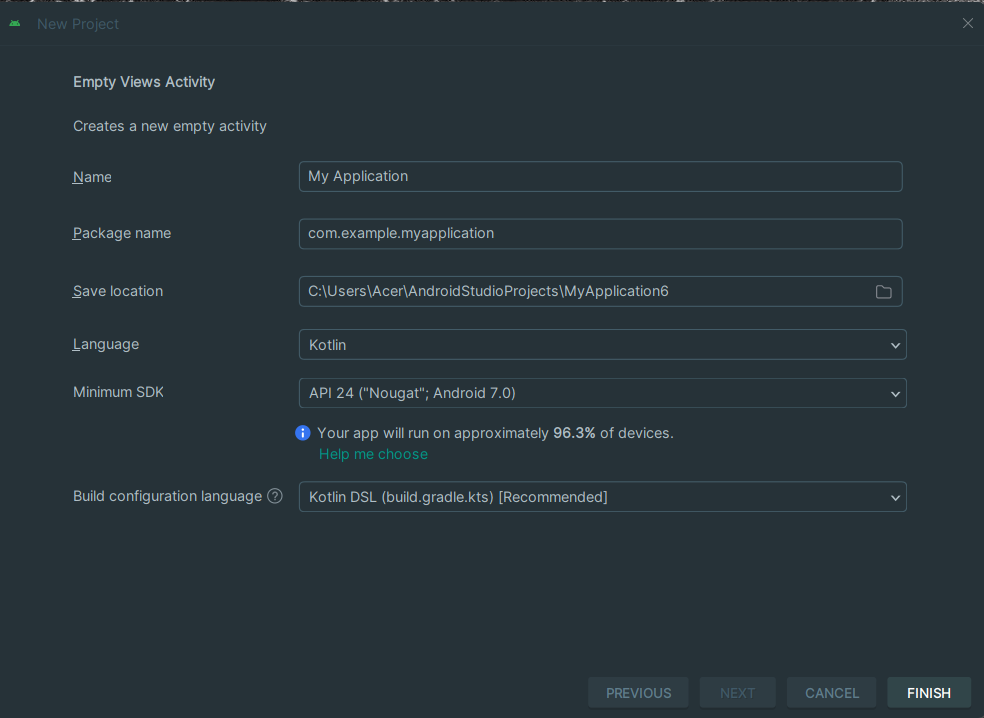


Рисунок 3.9 Настройка проекта

* 1. **Разработка приложения**

В данной главе будет представлено описание разработки приложения на данный момент для управления деталями конструктора Lego. В основе приложения лежит использование архитектурного паттерна MVVM (Model-View-ViewModel) с применением языка программирования Kotlin и фреймворка Android Jetpack.

Архитектура приложения

Приложение разработано с использованием следующих ключевых компонентов:

Model: В качестве модели данных используется база данных SQLite с использованием Room Persistence Library для взаимодействия с базой данных. Данные о деталях Lego представлены сущностью LegoParts, которая отображает таблицу в базе данных.

View: Интерфейс пользователя представлен фрагментами (Fragments) в соответствии с архитектурными принципами Single Activity. Каждый фрагмент отвечает за отображение определенного экрана пользовательского интерфейса.

ViewModel: Для управления данными и бизнес-логикой используется класс MainViewModel, который обеспечивает связь между пользовательским интерфейсом и моделью данных.

Основные компоненты приложения

Классы и интерфейсы

App и AppDataBase

Класс App представляет приложение Android и отвечает за инициализацию базы данных при запуске. Класс AppDataBase представляет базу данных приложения и содержит методы доступа к данным.

DatabaseCopier

Объект DatabaseCopier отвечает за копирование базы данных SQLite из ресурсов приложения при первом запуске.

FirstPageFragment

Фрагмент FirstPageFragment представляет первый экран приложения и содержит кнопку для добавления деталей.

LegoParts, LegoPartsDao, LegoPartsAdapter, LegoPartsViewHolder, LegoPartsDiffCallback

Эти классы отвечают за работу с данными о деталях Lego. LegoParts представляет сущность детали, LegoPartsDao - интерфейс для доступа к данным, LegoPartsAdapter и LegoPartsViewHolder - адаптер и ViewHolder для RecyclerView, а LegoPartsDiffCallback - класс для оптимизации обновления элементов RecyclerView.

MainActivity

Класс MainActivity представляет основную активность приложения.

MainViewModel и MainViewModelFactory

MainViewModel отвечает за управление данными и бизнес-логикой. MainViewModelFactory используется для создания экземпляра MainViewModel.

SecondPageFragment

Фрагмент SecondPageFragment представляет второй экран приложения.

Ресурсы макетов

В ресурсах приложения содержатся макеты XML для размещения пользовательского интерфейса на экранах приложения.

Реализация Создание базы данных

Для начала мы определим структуру базы данных и создадим классы для работы с ней.

// Класс для определения сущности LegoParts

@Entity(tableName = "lego\_parts")

data class LegoParts(

@PrimaryKey

@ColumnInfo(name = "id")

val id: Int,

@ColumnInfo(name = "name")

val name: String,

@ColumnInfo(name = "quantity")

var quantity: Int

)

// Абстрактный класс для работы с базой данных

@Database(entities = [LegoParts::class], version = 1)

abstract class AppDataBase: RoomDatabase() {

abstract fun legoPartsDao(): LegoPartsDao

}

Реализация ViewModel

Затем мы создадим ViewModel, которая будет обеспечивать взаимодействие между пользовательским интерфейсом и базой данных.

class MainViewModel(private val legoPartsDaoView: LegoPartsDao): ViewModel() {

// Методы для взаимодействия с базой данных и бизнес-логикой

}

Создание пользовательского интерфейса

Для отображения данных и взаимодействия с пользователем мы будем использовать фрагменты. На первом экране (FirstPageFragment) будет кнопка "Добавить детали", которая перенаправит пользователя на второй экран (SecondPageFragment), где будут отображаться Lego-детали.

<!-- layout/fragment\_first\_page.xml -->

<androidx.constraintlayout.widget.ConstraintLayout>

<Button

android:id="@+id/addDetailsButton"

android:text="@string/add\_a\_detail"

/>

</androidx.constraintlayout.widget.ConstraintLayout>

<!-- layout/fragment\_second\_page.xml -->

<androidx.constraintlayout.widget.ConstraintLayout>

<RecyclerView

android:id="@+id/recyclerView"

/>

<Button

android:id="@+id/confirmButton"

android:text="@string/confirm\_button"

/>

</androidx.constraintlayout.widget.ConstraintLayout>

Интеграция ViewModel и RecyclerView

Во втором фрагменте (SecondPageFragment) мы будем использовать RecyclerView для отображения списка Lego-деталей. Для этого нам понадобится адаптер LegoPartsAdapter, который будет связывать данные с макетом элемента списка.

class SecondPageFragment : Fragment() {

private lateinit var mainViewModel: MainViewModel

private lateinit var adapter: LegoPartsAdapter

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

// Инициализация ViewModel и RecyclerView

val application = requireActivity().application as App

val legoPartsDao = application.db.legoPartsDao()

mainViewModel = ViewModelProvider(this, MainViewModelFactory(legoPartsDao)).get(MainViewModel::class.java)

adapter = LegoPartsAdapter()

binding.recyclerView.adapter = adapter

// Наблюдение за изменениями данных и их обновление в RecyclerView

mainViewModel.allLegoParts.observe(viewLifecycleOwner) { legoParts ->

adapter.submitList(legoParts)

}

}

}

**Выводы**

В данной главе мы рассмотрели основные компоненты и реализацию приложения для работы с Lego-деталями. Мы создали структуру базы данных, настроили ViewModel для управления данными и попытались создать пользовательский интерфейс с использованием фрагментов и RecyclerView.

**4 Планы по развитию приложения**

## 4.1 Завершить работу по алгоритму работы приложения.

Приложение "MyLegoConstructor" создается с целью помочь любителям конструкторов LEGO эффективно управлять своими деталями и наборами. В данной главе обсудим алгоритм работы приложения и планы по его развитию.

При открытии приложения пользователь видит главный экран с выбором из трех кнопок:

* Выбор имеющихся деталей LEGO: Пользователь может просмотреть список доступных деталей и добавить информацию о деталях, которые у него есть.
* Выбор имеющихся наборов LEGO: Пользователь может просмотреть список доступных наборов и добавить информацию о наборах, которые у него есть.
* Поиск наборов: Пользователь может найти наборы конструктора, которые можно собрать из имеющихся деталей, а также определить, каких деталей не хватает для сборки других наборов.

1. Выбор имеющихся деталей LEGO

При нажатии на кнопку "Выбор имеющихся деталей LEGO" пользователь видит список доступных деталей. Он может добавить информацию о деталях конструктора, которые у него есть, а затем вернуться на главную страницу.

1. Выбор имеющихся наборов LEGO

При нажатии на кнопку "Выбор имеющихся наборов LEGO" пользователь видит список доступных наборов. Он может добавить информацию о наборах конструктора, которые у него есть, а затем вернуться на главную страницу.

1. Поиск наборов

При нажатии на кнопку "Поиск наборов" приложение анализирует полученную информацию о имеющихся деталях и наборах. Оно определяет, какие наборы конструктора можно собрать из имеющихся деталей, а также каких деталей не хватает для сборки других наборов.

## 4.2 Планы по развитию

Улучшение пользовательского интерфейса: Повышение удобства использования приложения путем улучшения дизайна и навигации.

Добавление функциональности: Расширение возможностей приложения, таких как возможность создания пользовательских списков деталей и наборов, добавление фильтров для поиска наборов и деталей.

Расширение базы данных: Пополнение базы данных деталей и наборов LEGO для более полного покрытия ассортимента продукции.

Локализация: Добавление поддержки Английского языка

Увеличение способов сборки одного набора, ведь из разных деталей можно собрать один и тот же объект. Например из 3 деталей “plate\_1\_x\_2” получится полноценный “ brick\_1\_x\_2“

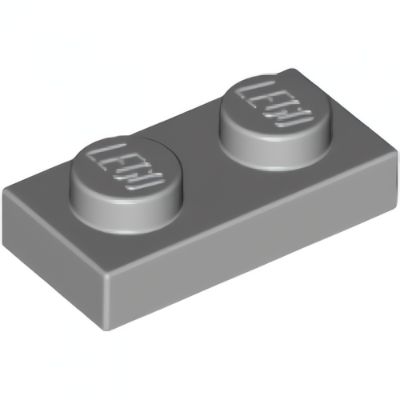
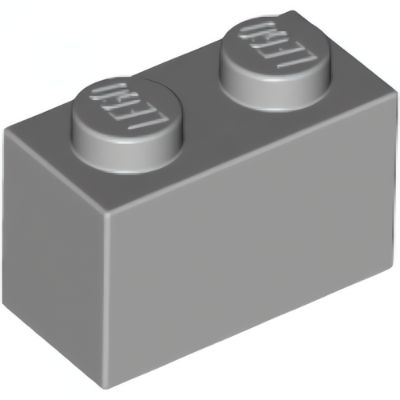
 

Рисунок 4.1 “plate\_1\_x\_2” Рисунок 4.2 “brick\_1\_x\_2”

## Выводы

Развитие приложения "MyLegoConstructor" направлено на повышение его функциональности, удобства использования и доступности для широкого круга пользователей. Планируемые улучшения помогут сделать приложение более полезным инструментом для управления коллекцией LEGO и создания новых моделей.

**ЗАКЛЮЧЕНИЕ**

Мобильные приложения захватывают практически все сферы жизни. И потенциальный пользователь уже не представляет своего обычного дня без них. Человек может первым узнавать о каких-либо акциях, скидках, новостях, достаточно лишь скачать приложение на мобильный телефон и разрешить присылать уведомления. Это дает бизнесу практически безграничные возможности, чтобы захватить внимание аудитории по всем каналам.

С помощью приложения MyLegoConstructor" человек имея представления какие детали есть увидит что он может собрать, а также купив наборы или детали что он сможет в дополнение. Это возможно при помощи мобильного приложения данной работы.

В рамках данной работы было начата разработка мобильного приложение под Android, позволяющее вводить детали конструктора или наборы имеющиеся у пользователя, чтобы узнать какие наборы мы сможем собрать сейчас и какие не можем если не купим набор или деталей.

Приложение работает с базой данных SQLite, в которой хранятся сохранённые детали.

Приложение предназначено для обычного пользователя для использования в повседневной жизни. Оно поможет улучшить выходные дни чтобы быстро найти какие наборы можно собрать в данный момент или что нужно купить чтобы собрать еще больше разных наборов конструктора.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Блох Д. «Эффективное программирование», 2017 г. – 485 с.

2. Автор: Дон Гриффитс, Дэвид Гриффитс- Программирование для Android на Kotlin. 3 изд (2023 г) 905c

3. Герберт Шилдт. «Java 8. Полное руководство 9-е издание», 2019 г. – 560 с.

4. Г. Шилдт «Java 8. Руководство для начинающих» (6-е издание), 2019 г. – 720 с.

5 Настя Гейн “Виджет”. https://secrets.tinkoff.ru/glossarij/vidzhet/(дата обращения: 17.04.2024)

6. Пантилейкин Н. В. Мобильные приложения и их виды // Научно-методический электронный журнал «Концепт». – 2017. – Т. 26. – С. 776–780. – URL: http://e-koncept.ru/2016/46956.htm.

7. П. Дейтел, Х. Дейтел, А. Уолд. Android для разработчиков. 3-е издание (2019 г., PDF)

8. Рето Майер. Android 2. Программирование приложений для планшетных компьютеров и смартфонов,Эксмо, 2019 г – 360 с

9. ОТЧЕТ ПРО РЫНОК МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 2023 https://asomobile.net/blog/rynok-mobilnyh-prilozhenij-2023/ (дата обращения: 18.04.2024).

10. Типы мобильных приложений и их влияние на успех in-app продвижения. URL: https://www.byyd.me/ru/blog/2023/09/types-of-apps/ (дата обращения: 18.04.2024).

11. Что такое нативные и кроссплатформенные приложения? плюсы и минусы. URL: https://itvdn.com/ru/blog/article/native-cross-platform (дата обращения: 20.04.2024).

**Приложение А. Листинг**

package com.example.mylegoconstriuktor

import android.app.Application

import androidx.room.Room

class App : Application() {

lateinit var db: AppDataBase

override fun onCreate() {

super.onCreate()

db = Room.databaseBuilder(

applicationContext,

AppDataBase::class.java,

"db").build()

}

}

import androidx.room.Database

import androidx.room.RoomDatabase

@Database(entities = [LegoParts::class], version = 1)

abstract class AppDataBase:RoomDatabase() {

abstract fun legoPartsDao(): LegoPartsDao

}

import android.content.Context

import java.io.File

import java.io.FileOutputStream

object DatabaseCopier {

fun copyDatabase(context: Context) {

val databasePath = context.getDatabasePath("db").path

if (!File(databasePath).exists()) {

try {

val inputStream = context.assets.open("sqlite.db")

val outputStream = FileOutputStream(databasePath)

val buffer = ByteArray(1024)

var length: Int

while (inputStream.read(buffer).also { length = it } > 0) {

outputStream.write(buffer, 0, length)

}

outputStream.flush()

outputStream.close()

inputStream.close()

} catch (e: Exception) {

e.printStackTrace()

}

}

}

}

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import androidx.fragment.app.Fragment

import androidx.fragment.app.activityViewModels

import androidx.navigation.fragment.findNavController

import com.example.mylegoconstriuktor.databinding.FragmentFirstPageBinding

class FirstPageFragment : Fragment() {

private val mainViewModel: MainViewModel by activityViewModels()

private var \_binding: FragmentFirstPageBinding? = null

private val binding get() = \_binding!!

override fun onCreateView(

inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

// Inflate the layout for this fragment

\_binding = FragmentFirstPageBinding.inflate(inflater)

return binding.root

}

override fun onDestroyView() {

super.onDestroyView()

\_binding = null

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

// Обработчик нажатия кнопки "Добавить детали"

binding.addDetailsButton.setOnClickListener {

mainViewModel.onAddBtn()

findNavController().navigate(R.id.secondPageFragment)

}

// Переход на вторую страницу

}

}

import androidx.room.ColumnInfo

import androidx.room.Entity

import androidx.room.PrimaryKey

@Entity(tableName = "lego\_parts")

data class LegoParts(

@PrimaryKey

@ColumnInfo(name = "id")

val id: Int,

@ColumnInfo(name = "name")

val name: String,

@ColumnInfo(name = "quantity")

var quantity: Int

)

import android.view.LayoutInflater

import android.view.ViewGroup

import androidx.recyclerview.widget.DiffUtil

import androidx.recyclerview.widget.ListAdapter

import androidx.recyclerview.widget.RecyclerView

import com.example.mylegoconstriuktor.databinding.ItemDetailBinding

class LegoPartsAdapter : ListAdapter<LegoParts, LegoPartsViewHolder>(LegoPartsDiffCallback()) {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LegoPartsViewHolder {

val binding = ItemDetailBinding.inflate(LayoutInflater.from(parent.context), parent, false)

return LegoPartsViewHolder(binding)

}

override fun onBindViewHolder(holder: LegoPartsViewHolder, position: Int) {

val legoPart = getItem(position)

holder.bind(legoPart)

}

}

class LegoPartsViewHolder(private val binding: ItemDetailBinding) : RecyclerView.ViewHolder(binding.root) {

fun bind(legoPart: LegoParts) {

binding.nameDatail.text = legoPart.name

binding.amountOfDetail.setText(legoPart.quantity.toString())

}

}

class LegoPartsDiffCallback : DiffUtil.ItemCallback<LegoParts>() {

override fun areItemsTheSame(oldItem: LegoParts, newItem: LegoParts): Boolean {

return oldItem.id == newItem.id

}

override fun areContentsTheSame(oldItem: LegoParts, newItem: LegoParts): Boolean {

return oldItem == newItem

}

}

@Dao

interface LegoPartsDao {

@Query("SELECT \* FROM lego\_parts")

fun getAll(): Flow<List<LegoParts>>

@Insert

suspend fun insert(legoParts: LegoParts)

@Delete

suspend fun delete(legoParts: LegoParts)

@Update

suspend fun update(legoParts: LegoParts)

@Query("SELECT \* FROM lego\_parts WHERE id = :id")

suspend fun getLegoPartById(id: Int): LegoParts?

}

class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

}

}

class MainViewModel(private val legoPartsDaoView: LegoPartsDao): ViewModel() {

val allLegoParts = this.legoPartsDaoView.getAll()

.stateIn(

scope = viewModelScope,

started = SharingStarted.WhileSubscribed(5000L),

initialValue = emptyList()

)

val legoPartsList = listOf(

"antenna\_small\_base\_with\_lever",

"arch\_1\_x\_6\_raised\_arch",

"bracket\_1\_x\_2\_\_\_2\_x\_2",

"brick\_\_modified\_1\_x\_1\_with\_open\_o\_clip\_vertical\_grip\_\_\_hollow\_stud",

"brick\_\_modified\_1\_x\_1\_with\_scroll\_with\_hollow\_stud",

"brick\_\_modified\_1\_x\_14\_with\_groove",

"brick\_\_modified\_1\_x\_2\_with\_grille\_fluted\_profile",

"brick\_\_modified\_1\_x\_2\_with\_masonry\_profile",

"brick\_\_modified\_facet\_2\_x\_2",

"brick\_\_round\_1\_x\_1\_open\_stud",

"brick\_\_round\_2\_x\_2\_dome\_bottom",

"brick\_1\_x\_1",

"brick\_1\_x\_1\_x\_3",

"brick\_1\_x\_12",

"brick\_1\_x\_16",

"brick\_1\_x\_2",

"brick\_1\_x\_2\_x\_2\_with\_inside\_stud\_holder",

"brick\_1\_x\_3",

"brick\_1\_x\_4",

"brick\_1\_x\_6",

"brick\_2\_x\_2\_corner",

"door\_\_frame\_1\_x\_6\_x\_7\_arched\_with\_notches\_and\_rounded\_pillars",

"door\_1\_x\_5\_x\_3\_with\_3\_studs\_and\_handle",

"pane\_for\_window\_1\_x\_2\_x\_2\_lattice\_diamond",

"panel\_1\_x\_2\_x\_1\_with\_rounded\_corners\_and\_2\_sides",

"panel\_1\_x\_4\_x\_5\_wall\_with\_window",

"plate\_\_modified\_1\_x\_1\_with\_light\_attachment\_\_\_thick\_ring",

"plate\_\_modified\_1\_x\_1\_with\_open\_o\_clip\_thick\_vertical\_grip",

"plate\_\_modified\_1\_x\_2\_with\_bar\_arm\_up\_horizontal\_arm\_5mm",

"plate\_\_modified\_1\_x\_2\_with\_bar\_handle\_on\_end",

"plate\_\_modified\_1\_x\_4\_offset",

"plate\_\_modified\_1\_x\_4\_with\_2\_studs\_with\_groove",

"plate\_\_round\_1\_x\_1",

"plate\_1\_x\_1",

"plate\_1\_x\_2",

"plate\_1\_x\_4",

"plate\_2\_x\_2",

"plate\_2\_x\_3",

"plate\_2\_x\_4",

"plate\_2\_x\_6",

"plate\_4\_x\_6",

"slope\_\_inverted\_45\_2\_x\_1",

"slope\_\_inverted\_45\_2\_x\_2\_with\_flat\_bottom\_pin",

"slope\_30\_1\_x\_1\_x\_0\_66",

"slope\_45\_1\_x\_1\_x\_0\_66\_quadruple\_convex\_pyramid",

"slope\_45\_2\_x\_1\_double\_with\_bottom",

"support\_1\_x\_1\_x\_6\_solid\_pillar",

"technic\_\_brick\_1\_x\_1\_with\_hole",

"technic\_\_brick\_1\_x\_2\_with\_holes",

"technic\_\_pin\_without\_friction\_ridges",

"tile\_\_modified\_1\_x\_1\_with\_open\_o\_clip",

"tile\_\_modified\_2\_x\_2\_triangular",

"tile\_\_modified\_facet\_2\_x\_2",

"tile\_\_round\_2\_x\_2\_with\_open\_stud",

"tile\_1\_x\_1\_with\_groove",

"tile\_1\_x\_2\_with\_groove",

"tile\_1\_x\_3",

"tile\_1\_x\_4",

"tile\_1\_x\_6",

"tile\_2\_x\_6",

"wedge\_\_plate\_2\_x\_4",

"window\_1\_x\_2\_x\_2\_flat\_front"

)

fun onAddBtn() {

viewModelScope.launch {

for (i in 1..62) {

// Проверяем, существует ли запись с таким идентификатором

if (legoPartsDaoView.getLegoPartById(i) == null) {

val name = legoPartsList.getOrElse(i - 1) { "" } // Получаем имя из списка, используя индекс i-1

legoPartsDaoView.insert(

LegoParts(

id = i,

name = name,

quantity = 0

)

)

}

}

}

}

fun onUpdateBtn(){

viewModelScope.launch {

allLegoParts.value.lastOrNull()?.let {

val legoParts = it.copy(

name = "деталь 4 на 4"

)

legoPartsDaoView.update(legoParts)

}

}

}

fun onDeleteBtn(){

viewModelScope.launch {

allLegoParts.value.lastOrNull()?.let {legoPartsDaoView.delete(it)

}

}

}

fun showAllTable() {

viewModelScope.launch {

val allLegoParts = legoPartsDaoView.getAll().firstOrNull()

val result = allLegoParts?.joinToString(separator = "\n") { "ID: ${it.id}, Name: ${it.name}, Quantity: ${it.quantity}" }

?: "No Lego parts found."

Log.d("AllLegoParts", result)

}

}

}

import androidx.lifecycle.ViewModel

import androidx.lifecycle.ViewModelProvider

import androidx.lifecycle.viewmodel.CreationExtras

class MainViewModelFactory(private val legoPartsDao: LegoPartsDao) : ViewModelProvider.Factory {

override fun <T : ViewModel> create(modelClass: Class<T>, extras: CreationExtras): T {

if (modelClass.isAssignableFrom(MainViewModel::class.java)) {

return MainViewModel(legoPartsDao) as T

}

throw IllegalArgumentException("Unknown ViewModel class")

}

}

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import androidx.fragment.app.Fragment

import androidx.fragment.app.activityViewModels

import androidx.lifecycle.LifecycleOwner

import androidx.lifecycle.ViewModelProvider

import androidx.lifecycle.observe

import androidx.recyclerview.widget.LinearLayoutManager

import com.example.mylegoconstriuktor.databinding.FragmentSecondPageBinding

class SecondPageFragment : Fragment() {

private lateinit var mainViewModel: MainViewModel

private var \_binding: FragmentSecondPageBinding? = null

private val binding get() = \_binding!!

private lateinit var adapter: LegoPartsAdapter

override fun onCreateView(

inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

// Inflate the layout for this fragment

\_binding = FragmentSecondPageBinding.inflate(inflater)

return binding.root

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

adapter = LegoPartsAdapter() // Initialize adapter

// binding.recyclerView.adapter = adapter // Set adapter to RecyclerView

val application = requireActivity().application as App

val legoPartsDao = application.db.legoPartsDao()

// mainViewModel = ViewModelProvider(this, MainViewModelFactory(legoPartsDao)).get(MainViewModel::class.java)

//

// mainViewModel.allLegoParts.observe(viewLifecycleOwner) { legoParts ->

// adapter.submitList(legoParts)

// }

}

override fun onDestroyView() {

super.onDestroyView()

\_binding = null

}

}

private fun Any.observe(owner: LifecycleOwner, onChanged: (List<LegoParts>?) -> Unit) {

TODO("Not yet implemented")

}