Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Уральский государственный педагогический университет»
Институт физики, технологии и экономики
Кафедра физики и математического моделирования

Разработка мобильного приложения для развивающего образовательного центра

Выпускная квалификационная работа

Квалификационная работа
допущена к защите
Зав. кафедрой
Сидоров В.Е.

дата _____ подпись _____

Исполнитель:
Ягупов Руслан Сергеевич,
студент V курса,
группы БЭ-51z

_____
подпись

Научный руководитель:
Омельченко Сергей Владимирович,
кандидат технических наук,
доцент

_____
подпись

Екатеринбург – 2016

Содержание

ВВЕДЕНИЕ

В современном обществе стремительными темпами развиваются информационные и мобильные технологии, и все большее влияние на нашу жизнь оказывает мобильная техника и интернет. Теперь безграмотным человеком считается не тот, кто не умеет читать или писать, а тот, кто не умеет пользоваться мобильным телефоном, компьютером или интернетом, но таких людей с каждым годом становиться все меньше и меньше.

С появлением интернета на него сразу же обратил внимание бизнес, так как это место могло стать хорошей нищей для сбыта продукции и услуг. Так и случилось, обороты компаний увеличились в десятки и сотни раз, и это повлияло на дальнейшее развитие технологий и революционному созданию мобильных технологий.

С помощью мобильных технологий мы можем теперь заплатить за парковку прямо с телефона. Встроенные карты позволяют не заблудиться в любом районе незнакомого города. Нам доступно расписание транспорта по нажатию одной лишь кнопки. В конце концов, теперь мы можем просто узнавать и исследовать интересные места, поблизости от которых мы оказались.

Сейчас у нас в карманах и сумках лежит больше всевозможной информации о мире, чем когда-либо в истории. Фактически для сотен миллионов людей стало совершенно естественным и привычным сразу искать в смартфонах и планшетах информацию о любой деятельности. Наши гаджеты уже могут предвидеть, какая информация нас может заинтересовать, и подают её нам в удобной форме.

Подобное, на первый взгляд, поверхностное и легкомысленное повседневное использование мобильных технологий изменило человеческое сообщество. Оно сделало наши связи с другими людьми гораздо шире и теснее. В сети мы обретаем настоящих друзей, которых никогда в жизни не

встречали вживую. Мы можем узнавать и обсуждать любые актуальные вопросы и события, происходящие по всему миру. Это стимулирует возникновение сообществ и обмен мнениями, которые ранее были просто невозможны.

Помимо изменений в нашей социальной активности и расширении доступной информации, мобильные технологии в скором времени могут начать активно влиять на наше здоровье.

В нашу жизнь устойчиво вошли мобильные технологии, которые кардинальным способом улучшают и процессы производства, и процессы потребления информации. Использование мобильных технологий позволяет быть в курсе всех событий в мире, прилагая для этого минимум усилий. Кроме того, мобильные технологии позволяют снизить стоимость продукции для конечных потребителей за счет оптимизации процессов, сокращения производственных издержек и непроизводственных затрат.

С помощью мобильных устройств мы не привязаны к рабочему месту и легко можете получить доступ к информации в дороге, в цеху, на совещании и где-либо еще, затратив на это минимум денежных средств.

Всё это лишь малая часть того, как мобильные технологии трансформировали, улучшили и упростили жизнь миллиардов людей. И этот процесс идёт по мере развития самих технологий. Нам, без преувеличения, повезло жить в мобильную эпоху. Эти технологии за какие-то 15 лет стали незаменимыми помощниками. Кто из вас периодически не ощущал себя без телефона как без рук? Очень велика, стала роль гаджетов во множестве наших больших и крупных дел, ежедневной рутине и уникальных событиях.

Но новые возможности принесли с собой и новые трудности. У каждой технологии есть свои преимущества и недостатки, с которыми приходится либо мириться, либо искать пути их решения.

Актуальность темы дипломной работы заключается в том, мобильные технологии являются важной частью нашего быта, уже нельзя представить себе, что мы не пользуемся мобильным телефоном или планшетом, и тем

более интернетом, эти технологии окружают нас повсюду, они известны маленькому ребенку и пенсионеру. Это расширяет возможности донесения целенаправленной информации о товарах и услугах широким массам населения, рост прибыли, и снижение расходов, а также дает конкурентное преимущество для поставщика, и выгоду для потребителя.

Целью данной выпускной квалификационной работы является создание мобильного приложения для развивающего образовательного центра, для более эффективного привлечения новых клиентов, повышения информированности существующих клиентов о работе предприятия, а также закрепление практических навыков программирования для ОС Android в среде разработки Android Studio.

Для достижения целей дипломной работы поставлены следующие задачи:

1. Произвести расчет экономических показателей эффективности создания мобильного приложения
2. Произвести поиск и анализ способов повышения потока клиентов с помощью мобильного приложения
3. Произвести анализ требований к мобильному приложению
4. Проектировать пользовательский интерфейс
5. Создать мобильное приложение

Структура дипломной работы обусловлена предметом, целью и задачами исследования. Работа состоит из введения, двух глав, заключения, списка литературы и приложений.

ГЛАВА 1. РАСЧЕТ ЭКОНОМИЧЕСКИХ ПОКАЗАТЕЛЕЙ ЭФФЕКТИВНОСТИ СОЗДАНИЯ ПРИЛОЖЕНИЯ

1.1 Показатели экономической эффективности

Прибыль – основная цель предпринимательской деятельности предприятия. В условиях рыночных отношений – это превращенная форма прибавочной стоимости. Учет прибыли позволяет установить, насколько эффективно ведется деятельность. По своей экономической природе прибыль выступает как часть стоимости прибавочного продукта, созданного для общества трудом работников материального производства. В процессе формирования прибыли отражаются все стороны хозяйственной деятельности промышленного предприятия: уровень использования основных фондов, технологии, организация производства и труда[1].

В структуре прибыли наибольшая доля приходится на выручку от реализации. Реализация продукции является одним из показателей планирования, оценки хозяйственной деятельности промышленного предприятия, основным источником дохода и пополнения бюджета. Денежные средства, поступающие на расчетный счет предприятия за реализованную продукцию, называются выручкой. Из выручки возмещаются производственные затраты на израсходованные материальные ценности, амортизация. Оставшаяся часть – это чистая продукция или валовой доход. Исключив из валового дохода заработную плату, с учетом отчислений на социальное страхование, получим прибыль от реализации.

После реализации товарной продукции часть прибыли предприятия вносится в бюджет государства и местный бюджет.

Помимо прибыли от реализации результаты хозяйственной деятельности оцениваются по балансовой, валовой и чистой прибыли.

Балансовая прибыли включает дополнительно к прибыли от реализации прибыль подсобных и обслуживающих производств, не связанных с непосредственно с основной производственной деятельностью

промышленного предприятия, прибыль от долевого участия в совместных предприятиях, сдачи имущества в аренду, различные дивиденды, а также убытки от прочих хозяйственных операций. Балансовая прибыль учитывает также прибыль или убытки от реализации излишка основных фондов, сверхнормативных запасов оборотных средств и прочего имущества.

Валовая прибыль является базой для налогообложения. Дополнительно к балансовой прибыли она учитывает штрафы и пени, уплаченные за вычетом аналогичных видов оплаты, перечисленных в бюджет.

Показатели экономической эффективности определяется в расчете на год (условно-годовая экономия) и с учетом времени внедрения до конца календарного года (текущая экономия). В последнем случае годовую экономию (по численности, трудоемкости, себестоимости) делят на 12 и умножают на число месяцев, оставшихся до конца года после завершения внедрения мероприятия.

Эффективность представляет собой величину эффекта, приходящуюся на единицу осуществленных затрат. Она может быть определена как отношение экономического эффекта к затратам и ресурсам, потребленным и примененным средствам производства. Чем больше производственный результат по сравнению с осуществленными для его достижения затратами, тем выше эффективность производства. В то же время большая величина эффекта может и не свидетельствовать о высокой эффективности, если он получен ценой значительных затрат ресурсов. Чтобы подчеркнуть этот аспект проявления эффективности производства, для ее характеристики иногда используют величину, обратную уровню эффективности: отношение затрат и ресурсов к экономическому эффекту. В такой форме оценка эффективности дает представление о том, величиной каких затрат достигается тот или иной эффект.

Если результаты превышают затраты, то можно утверждать, что имеет место экономическая эффективность. Повышение экономической эффективности заключается в увеличении полезных результатов на единицу

затраченных ресурсов.

Повысить экономическую эффективность означает:

1. получить больший результат при одинаковых затратах ресурсов;
2. получить одинаковый результат при меньших затратах ресурсов;
3. достичь большего результата с меньшими затратами ресурсов.

Кроме экономической эффективности можно рассматривать социальную или другие виды эффективности. Главное отличие экономической эффективности от других видов заключается в том, что полезные результаты и затраты выражены в стоимостной форме.

Экономическая эффективность определяется по использованию отдельных видов ресурсов, поэтому применяется система локальных показателей.

Часто даже IT-проекты оказываются не прибыльными не из-за того, что у них некачественный дизайн или программный код, а из-за того, что на этапе разработки инвестиционного проекта по созданию приложения не учитываются экономические показатели его дальнейшей работы. Если больше внимания уделить анализу предполагаемой рентабельности и окупаемости проекта, то риск от инвестиций будет значительно ниже. Экономические показатели эффективности работы проекта необходимо просчитывать еще на этапе разработки создания мобильного приложения.

Если предполагаемое мобильного приложение рассчитано на большие бюджетные вложения, то залогом его эффективности будет участие в проекте специалистов, которые смогут проанализировать его экономические составляющие.

Самой распространённой ошибкой при разработке инвестиционного проекта является неправильной выбор самой идеи приложения. Поэтому, лучше всего инвестору проконсультироваться у специалистов, прежде чем начинать разработку проекта приложения.

Отсутствие или непоследовательность бизнес-плана тоже является частой ошибкой инвесторов, решивших вложить средства в IT-сферу. Оптимальным решением в этом случае будет разработка бизнес-плана еще до того, как инвестор начнет разрабатывать непосредственно проект.

Очень часто IT-проекты страдают от недостатка финансирования, поэтому инвестору стоит ориентироваться на уже имеющиеся денежные возможности.

Также не стоит инвестору слишком завышать технические требования к проекту. Лучше всего исходить из реальных финансовых возможностей и разумных сроков и только необходимого функционала приложения.

Не стоит откладывать старт приложения до полного его наполнения. Лучше всего приложение наращивать поэтапно, постепенно расширяя его функциональные возможности.

Разрабатывая стратегию совершенствования проекта, инвестору надо подходить к вопросу грамотно, учитывая реальные финансовые возможности и текущее состояние рынка. Не нужно недооценивать инвестору значение бухгалтерского учета работы проекта.

1.2 Методы расчета

Амортизация данного проекта рассчитывается как произведение стоимости основных фондов умноженных на норму амортизации и деленное на 100%.

Прибыль рассчитывается 0-30% от полной себестоимости единицы продукта. Полная себестоимость единицы продукции складывается из производственной суммы и непроизводственной себестоимости.

Рыночная цена рассчитывается сумма полной себестоимости, налога на добавочную стоимость и прибыли.

Рентабельность продаж рассчитывается как прибыль, деленная на полную себестоимость. Рентабельность основных фондов рассчитывается как прибыль, деленная на сумму основных и оборотных фондов.

К основным показателям рентабельности относятся: рентабельность авансированного капитала и рентабельность собственного капитала. Их экономическая интерпретация заключается в следующем: сколько рублей прибыли приходится на 1 руб. авансированного (собственного) капитала. При расчете можно использовать прибыль отчетного периода или чистую прибыль.

1.2.1 Показатели использования трудовых ресурсов

Для измерения производительности труда, эффективности использования трудовых ресурсов в промышленности используются два основных показателя: выработка и трудоемкость[11].

Выработка измеряется количеством продукции, приходящейся на одного работающего или рабочего в единицу рабочего времени. В зависимости от принятых единиц рабочего времени различают часовую, дневную, месячную, квартальную и годовую выработку — производительность труда. При изготовлении на предприятии разнородной продукции при расчете показателя выработки используют стоимостной

способ измерения объема производства продукции, т. е. в денежном выражении. Выработку (В) можно рассчитывать по товарной, реализованной или валовой продукции по формуле:

$$В = \frac{ТП}{Чср}$$

где ТП — объем производства продукции в стоимостном выражении (товарная продукция), руб.;

Чср — среднесписочная численность промышленно-производственного персонала, чел.

Трудоемкость - это затраты рабочего времени на производство единицы продукции. В зависимости от состава включаемых в нее трудовых затрат различают технологическую трудоемкость, трудоемкость обслуживания производства, трудоемкость управления производством, полную трудоемкость.

Полная трудоемкость единицы продукции (Тп) измеряется в человеко-часах и определяется по формуле:

$$Тп = \frac{Т}{Nв}$$

где Т — затраты труда промышленно-производственного персонала на изготовление продукции за период, в чел. ч;

Nв - выпуск продукции за период в натуральном выражении, шт.

1.2.2 Показатели рентабельности

Абсолютная величина прибыли предприятия при всей важности этого показателя не дает полного и качественного представления об эффективной деятельности предприятия, не может быть применена для сравнения производственной деятельности различных предприятий. Равная прибыль еще не свидетельствует об одинаково успешной работе, поскольку для ее получения могут быть использованы различные количественные величины ресурсов. Сравнение эффективности хозяйственной деятельности

предприятий разных масштабов, производственного назначения и форм собственности производится обычно не в абсолютных, а в относительных показателях. Поэтому для оценки эффективности работы предприятия применяются относительные показатели, один из которых называется рентабельностью. Для оценки конечных результатов деятельности предприятий широко используются различные показатели рентабельности.

Рентабельность - это показатель, который характеризует эффективность применения или потребления ресурсов; он показывает величину прибыли, полученной предприятием в расчете на единицу примененных или потребленных ресурсов. Под примененными ресурсами подразумевают внеоборотные активы, а под потребленными — в первую очередь затраты материалов, энергоносителей, труда и т. п. Это показатель эффективности производственной деятельности предприятия за определенный период[13].

Рассмотрим наиболее распространенные в практике отечественных предприятий показатели рентабельности.

Рентабельность производства (Рп) — это отношение прибыли за отчетный период (за год) к среднегодовой стоимости основных средств и нормируемых оборотных средств. Определяется по формуле:

$$\text{Рп} = \frac{\text{Пп}}{\text{Фср}} + \text{Фоб}$$

где Пп — прибыль за период, руб.;

Фср — среднегодовая стоимость основных средств, руб.;

Фоб — норматив оборотных средств, руб.

Рентабельность продукции (реализованной) рассчитывается как отношение прибыли от реализации продукции к затратам на годовой объем производства продукции:

$$\text{Рпрод} = \frac{\text{Пр}}{\text{Сп}} * 100\%$$

где Рпрод — рентабельность продукции;

Пр — прибыль от реализации продукции;

Сп — полная себестоимость реализованной продукции.

Разновидностью рентабельности продукции является рентабельность изделия, которая определяется как отношение прибыли, полученной от реализации изделия соответствующего наименования, к себестоимости ее производства. В производственных условиях рентабельность различных видов изделий (Ри) может быть рассчитана по формуле:

$$\text{Ри} = \frac{\text{Ци} - \text{Си}}{\text{Си} * 100\%}$$

где Ри — рентабельность изделия;

Ци, Си — соответственно цена и полная себестоимость изделия.

Рентабельность продаж (Рпродаж) характеризует прибыльность продаж и показывает величину прибыли на один рубль продаж. Определяется по формуле:

$$\text{Рпродаж} = \frac{\text{П}}{\text{ВР} * 100}$$

1.3 Расчет показателей экономической эффективности

Так как за основу берется бесплатное программное обеспечение Android Studio, в затратную часть создания приложения относятся такие расходы как:

- расходы по электроэнергии
- расходы на хостинг
- заработная плата программисту

и прочие всевозможные расходы на канцелярские товары, и расходные материалы для компьютера. Такие расходы как, амортизации компьютеров и оргтехники.

1.3.1 Расчет электроэнергии

Наименование	Кол-во	кВт/час	кВт/в день	кВт/в месяц
Компьютер	1	0.2	1.8	39,6
Энергообеспечение	1	0.5	4.5	99
ИТОГО	2	0.7	5.6	138,6

Таблица 1.1

Цена 1 кВт/час: 3,54 руб.

В месяц: $\frac{3,54}{138,6} = 490,64$ руб

1.3.2 Расчет ежемесячных расходов

Наименование	Сумма руб.	ЕСН руб.
Зарплата программиста приложения	35000	9100
Зарплата программиста сайта	5000	1300
Электроэнергия	490,64	
Хостинг	280,10	
Интернет	3000	
Печеньки/кофе для программистов	2000	
ИТОГО	45770,74	10300
Расходов всего	56070,74	

Таблица 1.2

Заработная плата программисту приложения: 35000 руб.

Заработная плата программисту сайта за внедрение функции: 5000 руб.

RПост = 56070,74 - постоянные ежемесячные расходы.

1.3.3 Расчет амортизации

Годовая сумма амортизационных отчислений рассчитывается по формуле:

$$A = \frac{Ф * NA}{100\%}$$

где Ф - первоначальная стоимость основных фондов по видам, руб.;

NA - норма амортизации по видам основных фондов, в процентах.

Элементы основных фондов	Кол-во	Стоимость Руб.	Сумма Руб.	Норма амортизации %	Амортизационные отчисления Руб.
Компьютер	1	48000	48000	20	9600
Офисная мебель	1	9000	9000	20	1800
Помещение	15,5 м^2	700	10850	5	542,5
ИТОГО					11942,5

Таблица 1.3

Таким образом, годовая сумма амортизационных отчислений составляет 11942,5 рублей (Таблица 1.3).

Исходя из того, что трудоёмкость создания приложения составляет 15 дней, рассчитываем амортизацию оборудования за этот период по формуле:

$$\text{Афакт} = \frac{\text{Агод} * \text{Тфакт}}{365}$$

Рассчитаем сумму амортизационных отчислений для перечисленной группы оборудования с учетом числа календарных дней на разработку приложения:

$$\text{А} = \frac{11942{,}5 * 15}{365} = 490{,}78 \text{ руб}$$

Общая сумма заработной платы программистов составляет 40000 руб. Соответственно, затраты на заработную плату включаемые в себестоимость программы с учетом работы над программой в течение 15 дней составят:

$$\text{ЗПпр} = \frac{\text{ЗПмес} * \text{Тфакт}}{\text{Д}}$$

где ЗПпр - заработная плата в месяц программиста, руб.;

Тфакт - число календарных дней на разработку;

Д - число дней в периоде (месяц).

$$ЗПпр = \frac{40000 * 15}{22} = 27272{,}72 \text{ руб}$$

Отчисления на социальное страхование составят:

$$ЕСН = \frac{ЗПпр}{100} * 26\% = \frac{27272{,}72}{100} * 26\% = 7090{,}90 \text{ руб.}$$

1.3.4 Расчет ежемесячных материальных затрат

Наименование	Сумма, руб./мес.
Электроэнергия	490,64
Хостинг	280,10
Интернет	3000
Прочие расходы	2000
ИТОГО	5770,74

Таблица 1.4

Зм = 5770,74 рублей в месяц

Следовательно, затраты на период разработки программного продукта рассчитаем по формуле:

$$Зпр = \frac{Зм * Тфакт}{Д}$$

где Зм - ежемесячные затраты, руб.;

Тфакт - число календарных дней на разработку;

Д - число дней в периоде (месяц).

$$Зпр = \frac{5770{,}74 * 10}{22} = 3934{,}59 \text{ руб.}$$

1.3.5 Расчет себестоимости

Рассчитаем себестоимость программного продукта по формуле:

Сст - себестоимость разработки программы

$$Сст = Зпр + ЗПпр + ЕСН + А$$

$$Сст = 3934{,}59 + 27272{,}72 + 7090{,}90 + 490{,}78 = 38788{,}99 \text{ руб.}$$

Данная себестоимость является приблизительной, так как в ней не учтены некоторые детали, которые существенно не повлияют на итог.

Сст ≈ 39000 рублей.

1.3.6 Расчет цены приложения

Исходя из нормального уровня рентабельности 20%, мы можем определить цену разработанной нами программы:

$$Ц = \frac{Сст * 2 * R}{100\%}$$

где Сст - себестоимость разработки программы;

R - планируемый уровень рентабельности.

$$Ц = \frac{39000 * 2 * 20}{100\%} = 15600 \text{ руб.}$$

1.3.7 Графическая часть

В данной главе рассмотрим графическое представление рассчитанных нами данных по финансовым показателям предприятия. Ежемесячные материальные затраты представлены на рис 1. По гистограмме видно, что наиболее затратным является интернет, прочие расходы.

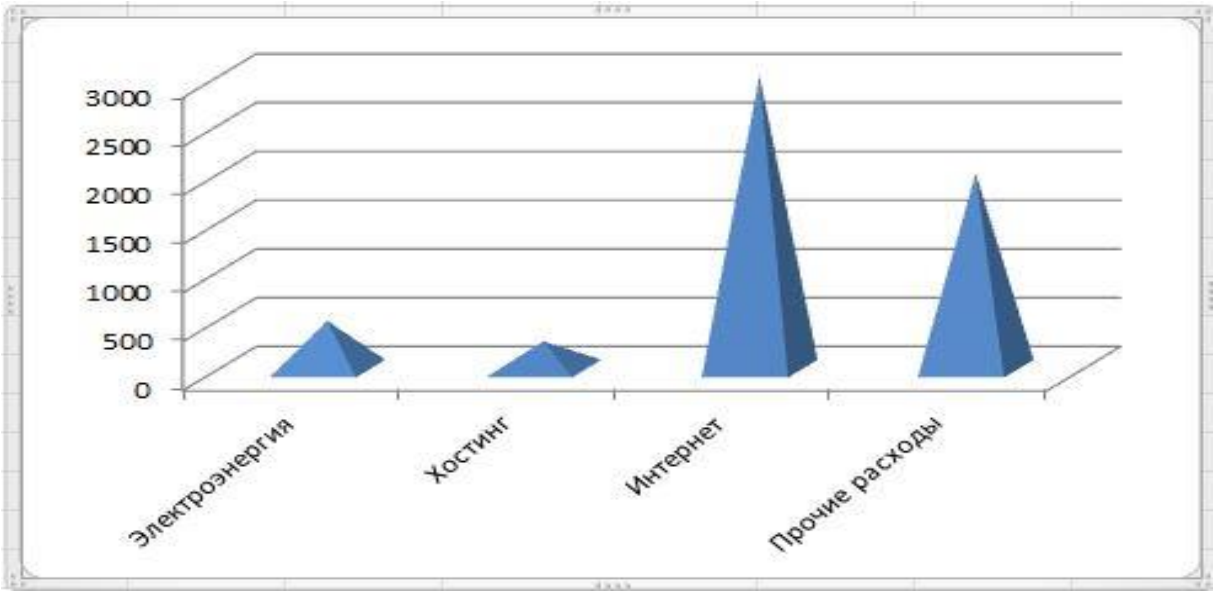

Рис 1. Ежемесячные материальные затраты

Годовая амортизация рис 2. показывает, что наиболее подвержена старению компьютерная техника, офисная мебель и помещения.

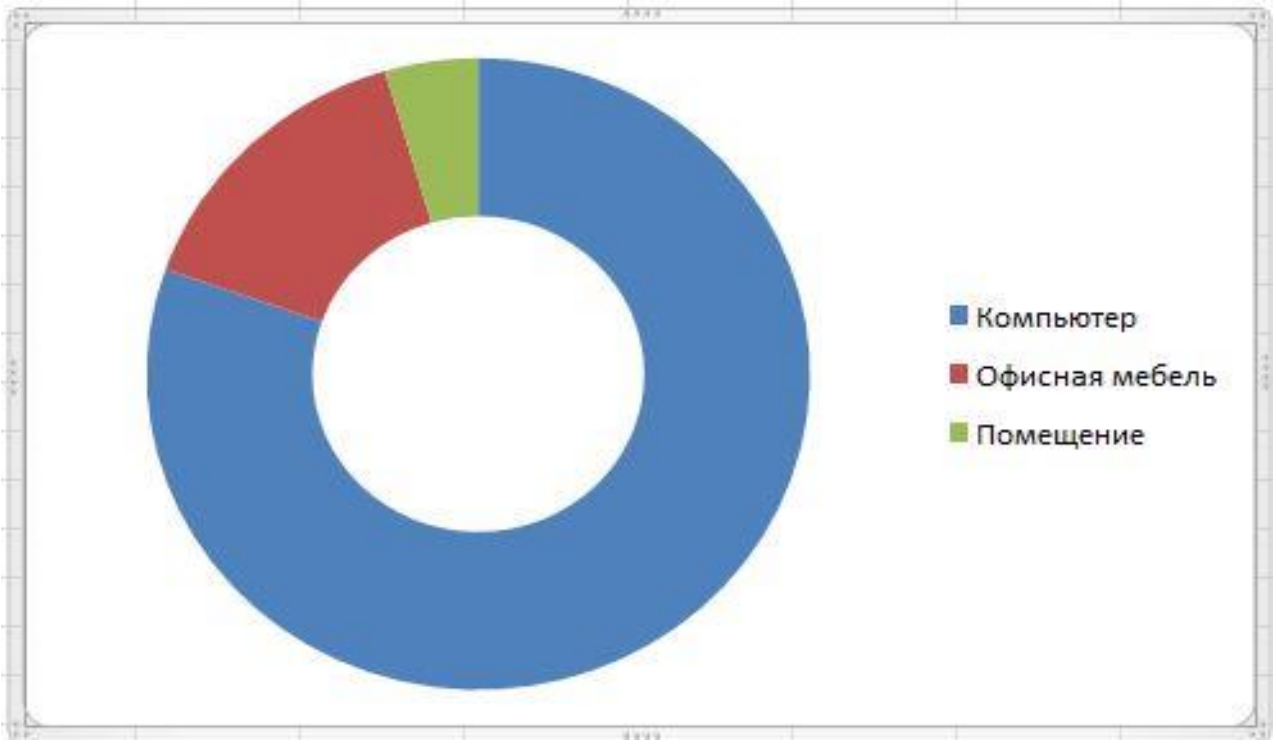

Рис.2 Годовая сумма амортизационных отчислений

Общие затраты рис 3. показывают как распределились затраты на разработку приложения.

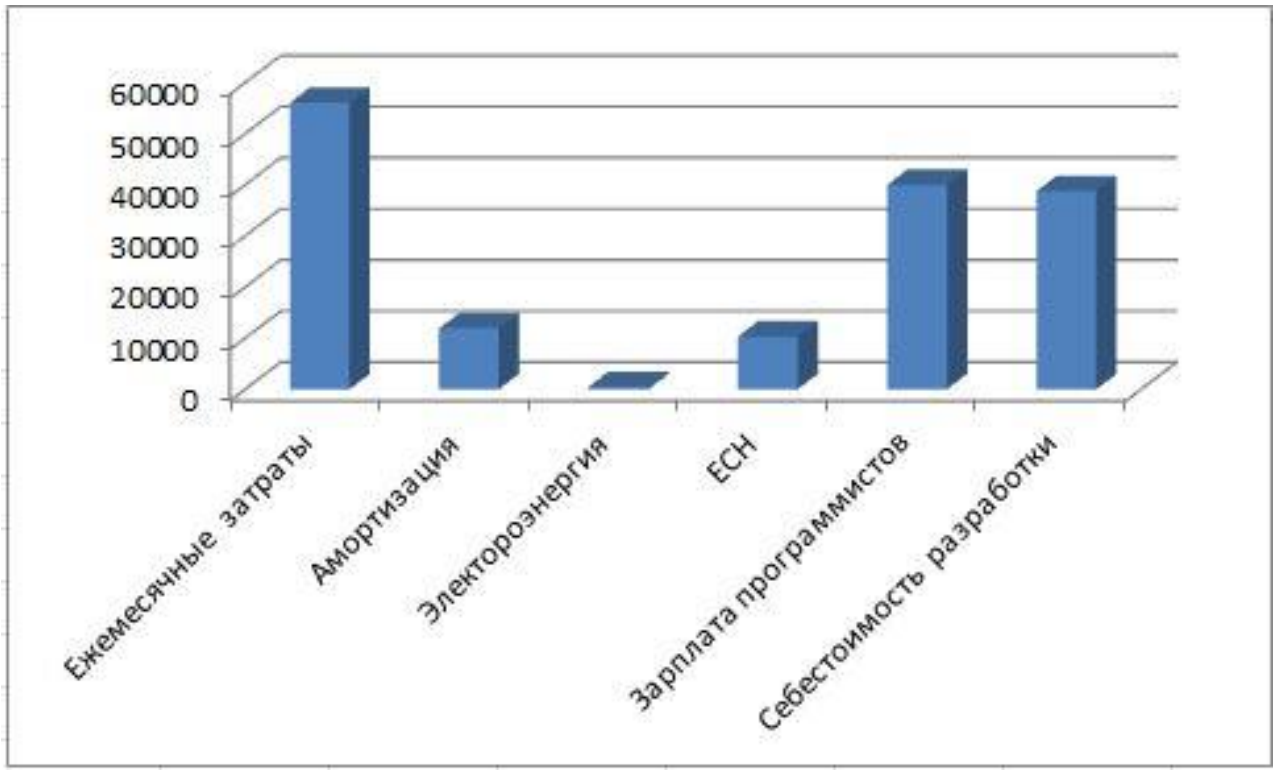

Рис 3. Общие затраты

1.4 Поиск и анализ способов повышения потока клиентов с помощью мобильного приложения

Для получения прибыли с помощью мобильных приложений есть целый арсенал инструментов, с помощью которых можно привлекать клиентов, стимулировать повторные покупки или устраивать прибыльные акции.

Отправка Push-уведомлений, наиболее полезная функция. Различные приложения часто уведомляют нас о тех или иных событиях – новое сообщение в чате, письмо в почте, напоминание купить аспирин, когда мы проезжаем мимо своей аптеки и т.д. Эти уведомления невозможно игнорировать – они не исчезнут, пока вы как-то на них не отреагируете.

Такие же уведомления мы можем посылать на экран своим клиентам с помощью специального push-функционала. При этом – что важно – мы можете отправить сообщение не просто всем клиентам, а, например, только жителям определенного района или города.

В своём приложении мы можем создавать мобильные GPS-купоны, которые автоматически активируются, когда клиент находится в непосредственной близости с нашим офисом.

Прямой звонок из мобильного приложения, нажав на специальную кнопку в приложении, пользователь может позвонить вам без необходимости запоминать и вручную набирать ваш номер телефона.

Помимо обычного номера телефона мы можем встроить возможность связаться с нами через форму обратной связи, по электронной почте, скайпу или сообщение в соц. сети.

Можно интегрировать многие популярные социальные сеть в своё приложение, получив тем самым ещё один канал воздействия на клиентов.

Мы можем:

- Встроить виджеты своих групп «Facebook» и «ВКонтакте», чтобы получить новых участников и усилить обратную связь от них;
- Встроить Twitter-канал своей компании;

- Встроить профиль социальной сети деловых контактов «LinkedIn»;

Предоставление любой информации о бизнесе и оперативное её обновление мобильного приложения, вы можете изменить или добавить любую информацию о компании или своих продуктах, и она автоматически обновится и в приложении клиента при следующем запуске. Ему не нужно будет обновлять приложение или предпринимать какие-то действия для этого – приложение само загрузит обновленную информацию без участия клиента, и таким образом у него всегда будет актуальная информация о нашей компании, даже если вы переехали или поменяли контактные телефоны. Для новостной и контентной информации можно использовать RSS-канал блога компании – тогда пользователь будет видеть свежие статьи при каждом использовании приложения.

Опросы – один из неплохих способов (гораздо лучше – это личное общение с несколькими конкретными клиентами и наблюдение за их действиями) получить реальное предоставление о том, что на самом деле хотят клиенты, чего им не хватает для полного удовлетворения, почему они перестали покупать и т.д.

После выявления, можно сделать выводы, что наиболее эффективными способами являются:

- Push-уведомления
- GPS-купоны(Check In)
- Прямой звонок из приложения
- Форма обратной связи
- Интеграция с социальными сетями
- Новости компании
- Опросы клиентов

В данной главе мы нашли и проанализировали способы повышения потока новых качественных клиентов и повышения прибыли предприятия.

ГЛАВА 2. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

2.1 Анализ требований мобильного приложения

Для того чтобы нам четко осознавать в каком направлении двигаться в дальнейшем, нам необходимо составить список требований которым должно соответствовать наше приложение.

Технические требования:

1. Соответствовать разработанному дизайну
2. Работать без серьезных ошибок
3. Работать с версии Android 4.3
4. Работать без визуальных задержек

Функциональные требования:

Пользователь заходит в приложение, без какой либо авторизации, как только приложение загрузиться должен отобразиться раздел Новости.

Для перехода в другой раздел должны быть кнопка соответствующая дизайну вверху слева и возможность открытия меню жестом по экрану слева на право. В открытом меню должны быть логотип компании в шапке меню, список разделов ниже, с интуитивно понятными картинками и названиями разделов.

В разделе новостей должна быть возможность прокрутки, как только загруженные новости кончаются в списке, должны загрузиться еще ограниченное количество новостей, если новостей больше нет, то больше загрузки происходить недолжно.

В разделе направления при его открытии должен появляется список направлений, при нажатии на элемент списка должно открыться новое окно либо список с подкатегориями с таким же функционалом. В новом окне должно отображаться содержимое, пришедшее с сервера.

В разделе контакты должны отображаться контактные данные предприятия.

В разделе о нас, отображается таблица с работниками организации с фото сотрудников, при нажатии на картинку открывается подробная информация о сотруднике в новом окне.

Согласно выше перечисленным требованиям, которые мы определяем для нашего мобильного приложения можно провести анализ который поможет нам эффективно, быстро и без лишних финансовых и человеческих затрат осуществить комплексную, самодостаточную разработку приложения. Приложение можно будет легко расширять, поддерживать, содержать легко читаемый программный код, будет понятен людям, которые будут его расширять и поддерживать в дальнейшем.

Для того чтобы более глубже понять принципы разработки, нужно понять для какой программной платформы мы будем разрабатывать приложение для этого нам нужно познакомиться ближе с операционной системой Android[26].

В 86 % смартфонов, проданных во втором квартале 2014 года, была установлена операционная система Android. При этом за весь 2014 год было продано более 1 миллиарда Android-устройств.

Продолжая усилия по привлечению к платформе Android новых разработчиков, Google начала проводить различные состязания. Первое из них, названное Android Developer Challenge (ADC), было проведено в 2008 году. Победившим проектам были обещаны щедрые денежные призы. ADC проводилось и в следующем году, в нем также приняло участие большое количество претендентов. В 2010 году ADC не проводилось, что может быть объяснено тем, что Android набрала необходимую базу разработчиков и поэтому необходимость в привлечении новых участников отпала. Кроме того, Google в начале 2010 года начал программу распространения устройств. Каждый разработчик, приложение (или приложения) которого было скачано более 5000 раз при среднем пользовательском рейтинге не менее 3,5 звезды, получал новый телефон Motorola Droid, Motorola Milestone или Nexus One. Эта акция вызвала бурную реакцию сообщества разработчиков, хотя

поначалу столкнулась с недоверием. Многие считали полученные по электронной почте уведомления продуманной мистификацией. К счастью, все оказалось правдой, и тысячи разработчиков по всему миру получили в подарок устройства — удачный шаг со стороны Google по привлечению новых разработчиков и поддержке уверенности уже привлеченных. Помимо этого Google предлагает разработчикам специальную версию аппаратов — Android Dev Phone (ADP). Первым ADP был вариант T-Mobile G1 для разработчиков (также известный как HTC Dream). Следующее поколение, ADP 2, было вариантом HTC Magic. К тому же Google продавала собственный телефон (Nexus One) конечным пользователям. Хотя изначально он не позиционировался как девелоперский, многие рассматривали его как потомка ADP 2. В итоге Google прекратила продажи Nexus One потребителям, и теперь он доступен только для ее партнеров и разработчиков. Этот аппарат компании Samsung, работающий на Android 2.3 (Gingerbread), называется Nexus S. ADP-устройства могут быть приобретены на Android Market (для этого необходимо иметь учетную запись разработчика). Ежегодная конференция Google I/O — мероприятие, на котором демонстрируются новейшие и лучшие технологии и проекты Google, и Android в последние годы занимает среди них особое место. На Google I/O обычно проводится несколько сессий, связанных с Android[25].

Android - операционная система для смартфонов, интернет-планшетов, электронных книг, цифровых проигрывателей, наручных часов, игровых приставок, нетбуков, смартбуков, очков Google, телевизоров и других устройств. В будущем планируется поддержка автомобилей и бытовых роботов. Основана на ядре Linux и собственной реализации виртуальной машины Java от Google. Изначально разрабатывалась компанией Android, Inc., которую затем купила Google. Впоследствии Google инициировала создание альянса Open Handset Alliance (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать Java-приложения, управляющие устройством через разработанные

Google библиотеки. Android Native Development Kit позволяет портировать библиотеки и компоненты приложений, написанные на Си и других языках[25].

Это не просто еще один дистрибутив Linux для мобильных устройств. При разработке для Android вам, скорее всего, не придется иметь дело с сами ядром Linux. С точки зрения программиста, Android — платформа, абстрагирующая разработчика от ядра и позволяющая ему создавать код на Java. Android обладает несколькими полезными возможностями. Во-первых, это фреймворк, предлагающий большой набор API для создания различных типов приложений и, кроме того, обеспечивающий возможности повторного использования и замены компонентов, которые предлагаются платформой и сторонними приложениями. Во-вторых, наличие виртуальной машины Dalvik, отвечающей за запуск приложений на Android. Кроме того, к услугам разработчика набор графических библиотек для 2D- и 3D-приложений, поддержка мультимедиа-форматов (Ogg Vorbis, MP3, MPEG-4, H.264, PNG), API для доступа к камере, GPS, компасу, акселерометру, сенсорному экрану, джойстику и клавиатуре. Имеется даже специальное API для воспроизведения фоновых звуковых эффектов, которое пригодится нам при разработке игр. Не все Android-устройства обладают всеми этими возможностями — налицо аппаратное разделение. Конечно, список возможностей Android не исчерпывается упомянутыми мной. Архитектура Android формируется из набора компонентов. Каждый компонент построен на основе элементов более низкого уровня. На рис. 1 представлен краткий обзор главных компонентов Android.

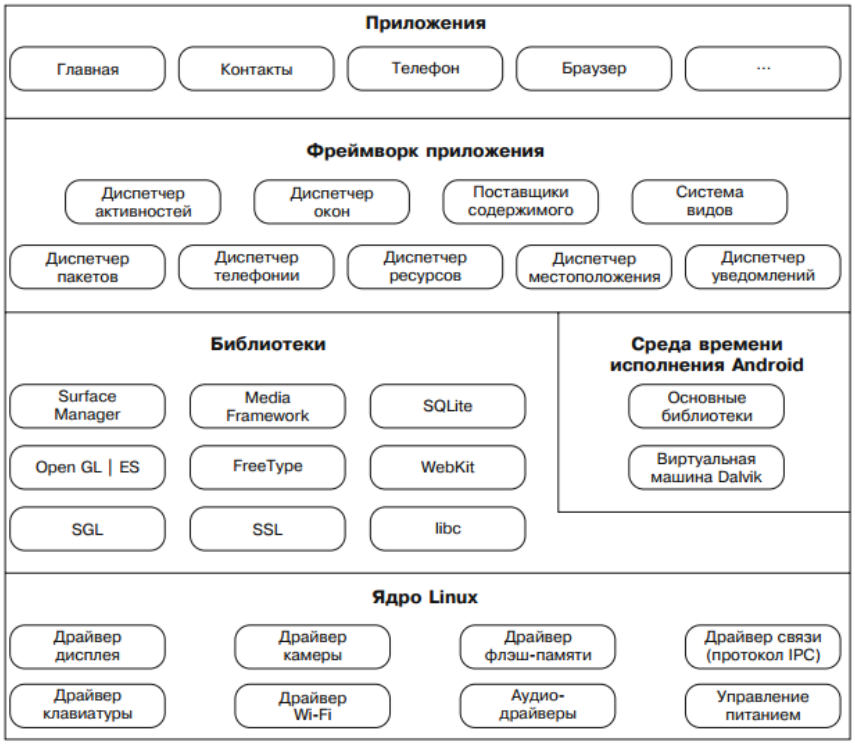

Рис 1. Архитектура Android

Фреймворк приложения связывает вместе системные библиотеки и среду выполнения, создавая, таким образом, пользовательскую сторону Android. Фреймворк управляет приложениями и предлагает продуманную среду, в которой они работают. Разработчики создают приложения для этого фреймворка с помощью набора программных интерфейсов на Java, охватывающих такие области, как разработка пользовательского интерфейса, фоновые службы, оповещения, управление ресурсами, доступ к периферии и т. д. Все ключевые приложения, поставляемые вместе с ОС Android (например, почтовый клиент), написаны с помощью этих API. Приложения, будь они с интерфейсом или с фоновыми службами, могут связываться с другими приложениями. Эта связь позволяет одному приложению использовать компоненты других. Простой пример — программа, делающая фотоснимок и потом обрабатывающая его. Приложение запрашивает у системы компонент другого приложения, обеспечивающий это действие. Далее первое приложение может повторно использовать этот компонент (например, от встроенного приложения камеры или от фотогалереи).

Подобный алгоритм снимает значительную часть ноши с программиста, а также позволяет настроить многообразие аспектов поведения Android.

В программировании для Android используются объектно-ориентированные технологии, поэтому мы приведем обзор объектных технологий. В условиях постоянно растущего спроса на новые мощные программные продукты достаточно трудно сочетать такие требования, как быстрота разработки, правильность работы и экономичность. Объекты, по сути, представляют собой повторно используемые программные компоненты. В качестве объектов могут использоваться дата, время, видео, человек, автомобиль и другие предметы материального мира. Практически каждое существительное может быть адекватно представлено программным объектом в понятиях атрибутов (например, имя, цвет и размер) и поведений (например, вычисление, перемещение и передача данных). Разработчики программ видят, что использование модульной структуры и объектно-ориентированного проектирования при разработке приложений повышает продуктивность работы. Этот подход пришел на смену применявшемуся ранее структурному программированию — объектно-ориентированный код проще понять и изменить[7].

Чтобы лучше понять суть объектов и их содержимого, воспользуемся простой аналогией. Представьте себе, что мы находимся за рулем автомобиля, и нажимаете педаль газа, чтобы набрать скорость. Что должно произойти до того, как мы получаем такую возможность? Прежде чем мы поведем автомобиль, кто-то должен его спроектировать. Изготовление любого автомобиля начинается с инженерных чертежей, которые подробно описывают устройство автомобиля. В частности, на этих чертежах показано устройство педали акселератора. За этой педалью скрываются сложные механизмы, которые непосредственно ускоряют автомобиль, подобно тому, как педаль тормоза скрывает механизмы, тормозящие автомобиль, а руль скрывает механизмы поворота. Благодаря этому люди, не имеющие понятия о внутреннем устройстве автомобиля, могут легко им управлять. Подобно

тому, как невозможно готовить пищу на кухне, которая лишь изображена на листе бумаги, нельзя водить автомобиль, существующий лишь в чертежах. Прежде чем мы сядем за руль машины, ее нужно построить на основе инженерных чертежей. Воплощенный в металле автомобиль имеет реальную педаль газа, с помощью которой он может ускоряться, но и это не все — он не может это делать самостоятельно, а только после того, как водитель нажмет на педаль.

Воспользуемся примером с автомобилем для демонстрации некоторых ключевых концепций объектно-ориентированного программирования. Для выполнения операции в программе требуется метод, в котором «скрываются» инструкции программы, непосредственно выполняющие операцию. Метод скрывает эти инструкции от пользователя подобно тому, как педаль газа автомобиля скрывает от водителя механизмы, вызывающие ускорение автомобиля. Программная единица, именуемая классом, включает методы, выполняющие задачи класса. Например, класс, представляющий банковский счет, может включать три метода, один из которых пополняет счет, второй снимает средства со счета, а третий запрашивает текущий баланс. С концептуальной точки зрения класс подобен инженерному чертежу, в котором изображено устройство педали газа, рулевого колеса и других механизмов.

Водитель не сядет за руль автомобиля, пока его не построят по чертежам, так и построение объекта класса необходимо для выполнения задач, определяемых методами класса. Этот процесс называется созданием экземпляра. Полученный при этом объект называется экземпляром класса.

На основе одних и тех же чертежей можно создать много автомобилей, а на основе одного класса можно создать много объектов. Использование существующих классов для создания новых классов экономит время и силы разработчика. Повторное использование также облегчает создание более надежных и эффективных систем, поскольку ранее созданные классы и компоненты обычно проходят тщательное тестирование, отладку и

оптимизацию. Подобно тому, как концепция использования взаимозаменяемых частей легла в основу промышленной революции, повторно используемые классы играют ключевую роль в программной революции, которая была инициирована внедрением объектных технологий.

Когда мы ведем машину, нажатие педали газа отправляет автомобилю сообщение с запросом на выполнение определенной задачи (ускорение автомобиля). Подобным же образом отправляются сообщения объекту. Каждое сообщение представляется вызовом метода, который «сообщает» методу объекта о необходимости выполнения некоторой задачи. Например, программа может вызвать метод deposit объекта банковского счета, чтобы пополнить банковский счет.

Любой автомобиль помимо возможности выполнять определенные операции также обладает атрибутами, такими как цвет, количество дверей, запас топлива в баке, показания спидометра и одометра. По аналогии с операциями атрибуты автомобиля представляются на инженерных диаграммах (в качестве атрибутов автомобиля могут выступать одометр и указатель уровня бензина). При вождении автомобиля его атрибуты перемещаются вместе с ним. Каждый автомобиль содержит собственный набор атрибутов. Например, каждый автомобиль «знает» о том, сколько бензина осталось в его баке, но ему ничего не известно о запасах горючего в баках других автомобилей. Объект, как и автомобиль, имеет собственный набор атрибутов, которые он «переносит» с собой при использовании этого объекта в программах. Эти атрибуты определяются как часть объекта класса. Например, объект bankaccount имеет атрибут баланса, представляющий количество средств на банковском счете. Каждый объект bankaccount «знает» о количестве средств на собственном счете, но ничего не «знает» о размерах других банковских счетов. Атрибуты определяются с помощью других переменных экземпляра класса.

Классы инкапсулируют атрибуты и методы в объекты (атрибуты и методы объекта между собой тесно связаны). Объекты могут обмениваться информацией между собой, но обычно они не «знают» о деталях реализации других объектов, которые скрыты внутри самих объектов. Подобное сокрытие информации жизненно важно в практике хороших программных архитектур.

С помощью наследования можно быстро и просто создать новый класс объектов. При этом новый класс наследует характеристики существующего класса, которые при этом могут частично изменяться. Также в новый класс добавляются уникальные характеристики, присущие только этому классу. Если вспомнить аналогию с автомобилем, «трансформер» является объектом более обобщенного класса «автомобиль», у которого может подниматься или опускаться крыша.

А теперь ответьте на вопрос, каким образом мы собираемся программировать? Скорее всего, таким же образом, как и большинство других программистов, — включим компьютер, и начнете вводить исходный код программы. Подобный подход годится при создании маленьких программ, но что делать в том случае, когда приходится создавать крупный программный комплекс, который, например, управляет тысячами банкоматов крупного банка? Либо если вам приходится возглавлять команду из 1000 программистов, занятых разработкой системы управления воздушным движением следующего поколения? В таких крупных и сложных проектах нельзя просто сесть за компьютер и начать набирать код. Чтобы выработать наилучшее решение, следует провести детальный анализ требований к программному проекту (то есть определить, что должна делать система) и разработать архитектуру, которая будет соответствовать этим требованиям (то есть определить, как система будет выполнять свои задачи). В идеале перед началом создания кода следует выполнить эту процедуру и тщательно проанализировать проект (либо поручить выполнение этой задачи другим профессионалам). Если в ходе выполнения этого процесса происходит анализ

и проектирование системы с применением объектно-ориентированного подхода, значит, мы имеем дело с процессом объектно-ориентированного анализа и проектирования (ООАП). Языки программирования, подобные Java, называются объектно-ориентированными. Программирование на таких языках, называемое объектно-ориентированным программированием (ООП), реализует объектно-ориентированные проекты в виде работоспособных систем.

2.2 Анализ пользовательского интерфейса

Дизайн пользовательского интерфейса является фактором, оказывающим влияние на три основных показателя качества программного продукта: его функциональность, эстетику и производительность.

Функциональность является фактором, на который разработчики приложений зачастую обращают основное внимание. Они пытаются создавать программы так, чтобы пользователи могли выполнять свои задачи, и им было удобно это делать. Функциональность важна, но, тем не менее, это не единственный показатель, который должен учитываться в ходе разработки.

Эстетичный внешний вид самого приложения и способа его представления позволяет сформировать у потребителя положительное мнение о программе. Однако эстетические характеристики весьма субъективны и описать их количественно гораздо труднее, чем функциональные требования или показатели производительности. Вся эстетика приложения зачастую сводится к простому выбору: соотносятся ли между собой используемые цвета, передают ли элементы интерфейса их назначение и смысл представляемых операций, что ощущает человек при использовании тех или иных элементов управления и насколько успешно он их использует.

Производительность, а равно и надежность, также влияют на перспективу применения программы. Если приложение хорошо выглядит,

имеет простое и удобное управление, но, к примеру, медленно прорисовывает экраны, регулярно «подвисает» на десяток-другой секунд или, того хуже, падает с критической ошибкой при некорректных действиях пользователя, у него, вероятно, будет мало шансов на длительную эксплуатацию. В свою очередь, быстрая и стабильная работа приложения могут отчасти компенсировать его не самый стильный дизайн или отсутствие каких-то вторичных функций.

Для обеспечения успешной работы пользователя от дизайнера интерфейса требуется соблюдать баланс между вышеперечисленными факторами на протяжении всего жизненного цикла разработки приложения. Это достигается последовательной и тщательной проработкой деталей интерактивного взаимодействия на каждом из этапов разработки пользовательского интерфейса включающих:

1. Проектирование
    - Функциональные требования: определение цели разработки и исходных требований
    - Анализ пользователей: определение потребностей пользователей, разработка сценариев, оценка соответствия сценариев ожиданиям пользователей
    - Концептуальное проектирование: моделирование процесса, для которого разрабатывается приложение
    - Логическое проектирование: определение информационных потоков в приложении
    - Физическое проектирование: выбор платформы, на которой будет реализован проект и средств разработки
2. Реализация
    - Прототипирование: разработка бумажных и/или интерактивных макетов экранных форм
    - Конструирование: создание приложения с учетом возможности изменения его дизайна

2.3 Прототипирование пользовательского интерфейса

Прототипы стали незаменимым этапом профессиональной разработки программного обеспечения. Прототипы облегчают жизнь всем людям, вовлеченным в процесс создания того или иного проекта: клиенту, менеджерам, дизайнерам и разработчикам. Прототип визуализирует конечный продукт и помогает выявить серьезные проблемы на самых ранних этапах, помогая избежать проблем на поздних стадиях разработки.

Конечно, первоначальные эскизы, выполненные, к примеру, на бумаге и воплощающие первые идеи дизайнера еще никто не отменял. Но всю дальнейшую работу, в которую будут вовлечены все члены рабочей группы, лучше всего проводить в специальном программном обеспечении, которое поддерживает редактирование, управление версиями и совместную работу.

Прототип приложения будет создаваться в достаточно популярной графической программе Adobe Photoshop[27]. Эта программа является достаточно удобной в использовании и быстром освоении для построения прототипа.

Начинать прототипирование мы начнем с разработки левого меню, так как данный функционал является главным в приложении. Первое, что будет добавлено в макет это так называемые StatusBar и Toolbar(Рис 3).

Рис 3. Сверху Statusbar, снизу Toolbar

Statusbar – это визуальный графический элемент системы Android для взаимодействия пользователя с системой.

Toolbar - также известный как панели действий, является одним из наиболее важных элементов дизайна в деятельности нашего приложения, так

как он обеспечивает визуальную структуру и интерактивные элементы, которые знакомы пользователям.

Для создания меню мы обратимся к Google Material Pattern[28], в котором описывается лучшие решения для построения дизайна приложения.

В рекомендациях материального дизайна Google говорится, что объекты интерфейса пользователя должны отбрасывать тени, как и объекты реального мира. Когда для представления задается свойство elevation, Android автоматически генерирует тень от этого представления. Чем больше значение elevation, тем четче выражена тень. В рекомендациях материального дизайна приведены желательные значения elevation для разных экранных компонентов — например, для диалоговых окон рекомендуемое значение elevation равно 24dp, а для меню — 8dp.

Разработчики приложений часто приводят цвета темы в соответствие с фирменным стилем компании. Если вам потребуется изменить цвета темы, рекомендации материального дизайна Google по использованию цвета советуют выбрать цветовую палитру, состоящую из основного цвета (не более чем с тремя оттенками) и акцентного цвета. Основные цвета обычно используются для окрашивания строки состояния и панели приложения в верхней части экрана; кроме того, они могут использоваться в графическом интерфейсе.

Согласно рекомендациям дизайнеров Google, меню должно содержать элементы виде списка с иконками и надписями, так же иметь заголовок с необходимой функциональностью. Мы выбрали так называемый Navigation Drawer для реализации навигации в приложении.

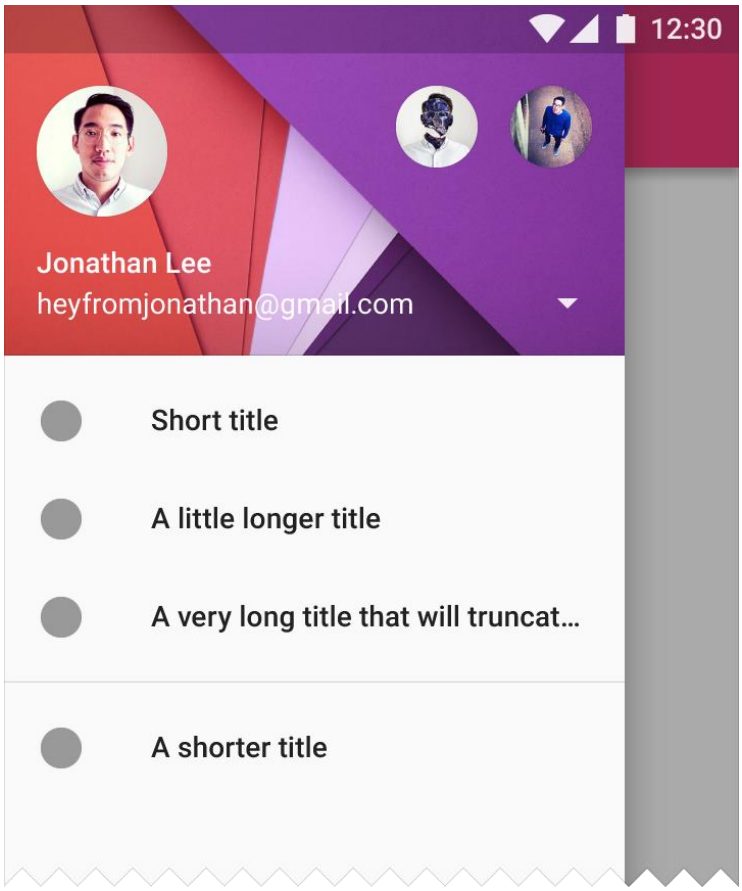

Рис 4. Типовой вид меню приложения

Значок в документации называется "гамбургером" (Hamburger menu). Это официальная позиция Google. При нажатии слева вылезет навигационная шторка. По высоте она занимает весь экран, включая системную область. Можете подвигать шторку вперёд-назад, чтобы увидеть, что верхняя кромка шторки в системной области полупрозрачна и не закрывает системные значки. Подобное поведение доступно на устройствах под Android 5 и выше. На старых устройствах шторка находится под системной панелью. Сама шторка состоит из двух основных частей - в верхней части находится картинка и текст, а в нижней - меню со значками. Меню в свою очередь разделено на две группы. В верхней части значки можно выбрать, и выбранный пункт останется выделенным. В нижней части меню пункты не выделяются. Уберите шторку обратно и вызовите теперь её не нажатием на значок гамбургера, а движением пальца от края экран в центр. Мы увидим, что во время движения значок трансформируется. К сожалению, шторка закрывает значок и непонятно, во что превращаются три полоски.

Наш Navigation Drawer содержит заголовок с логотипом компании в стиле Material Design[28], ниже идут список разделов приложения, которыми осуществляется навигация по разделам (Рис 4).

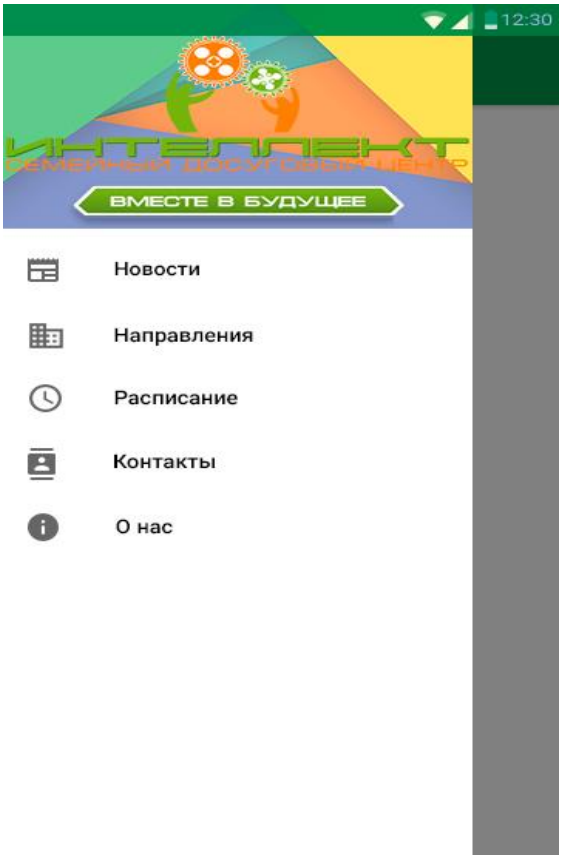

Рис 4. Navigation Drawer

Спрототипированный Navigation Drawer соответствует требованиям Material Design для Android и предоставляет пользователю приложения удобные возможности для навигации по разделам, также содержит логотип организации, что поддёргивает принадлежность приложения к этой организации, которая занималась разработкой данного приложения для удобства пользования пользователя.

Разрабатывая прототип раздела новостей, мы применили дизайн карточек, который используют многое приложения, в Android они называются CardView (Рис 5).

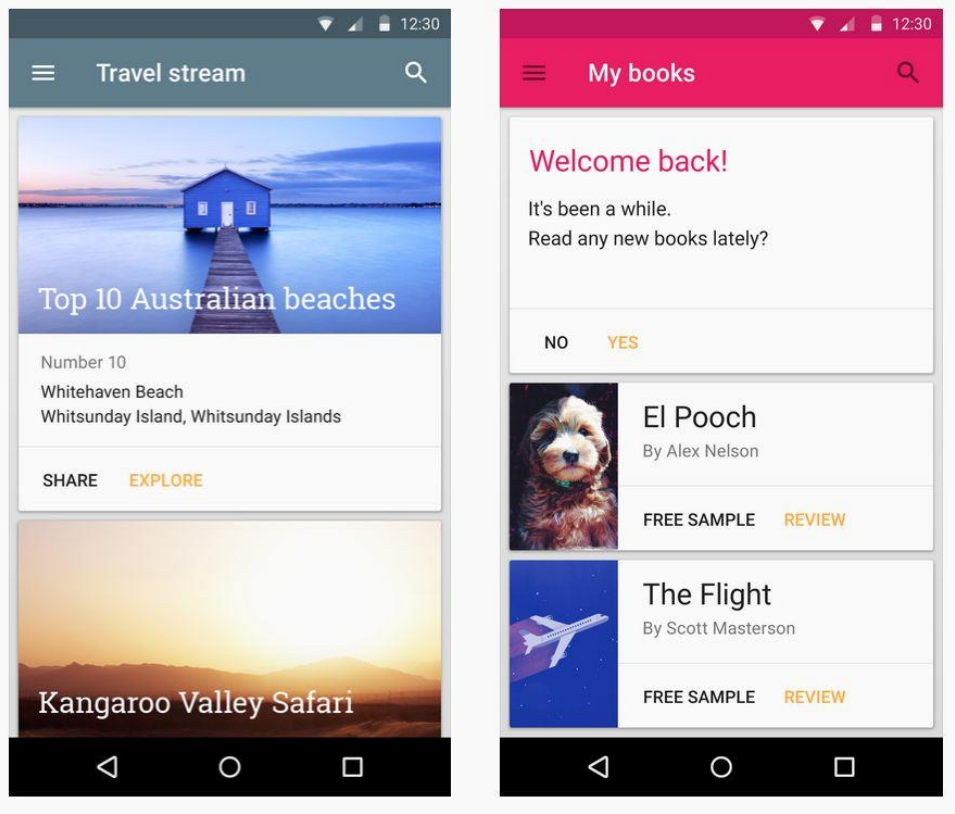

Рис 5. Пример использования CardView

Новый компонент CardView появился в Android Lollipop но благодаря библиотеке совместимости доступен и для старых устройств. Позволяет создавать красивую карточку с тенью и закруглёнными углами, который служит контейнером для других компонентов. Конечно же, он применим не во всех ситуациях, но отлично подходит при отображении какого-то пункта, состоящего из картинки, заголовка и небольшой вводной информации. Многие популярные приложения уже активно используют CardView.

В нашем случае как раз CardView подходит для создания списка раздела новостей, так как у каждой новости есть картинка, заголовок, дата события, краткое описание. Для подчеркивания заголовка использовалась полоса черного цвета с прозрачностью 56 процентов, данный процент прозрачности является наиболее приемлемым вариантом и является стандартным для данных заголовков (Рис 6).

Рис 6. Прототип ленты новостей

На дисплее элементов в списках является очень распространенной картины в мобильных приложениях. Пользователь видит список элементов, и можно прокручивать их. Если он выбирает один из элементов списка, это может обновить ActionBar или триггеры подробную экран для выбора.

Android предоставляет ListView класса, который способен отображать прокручивать список элементов. Эти элементы могут быть любого типа.

Раздел направления в связи с его структурой будет обладать стандартным списочным видом (Рис 7).

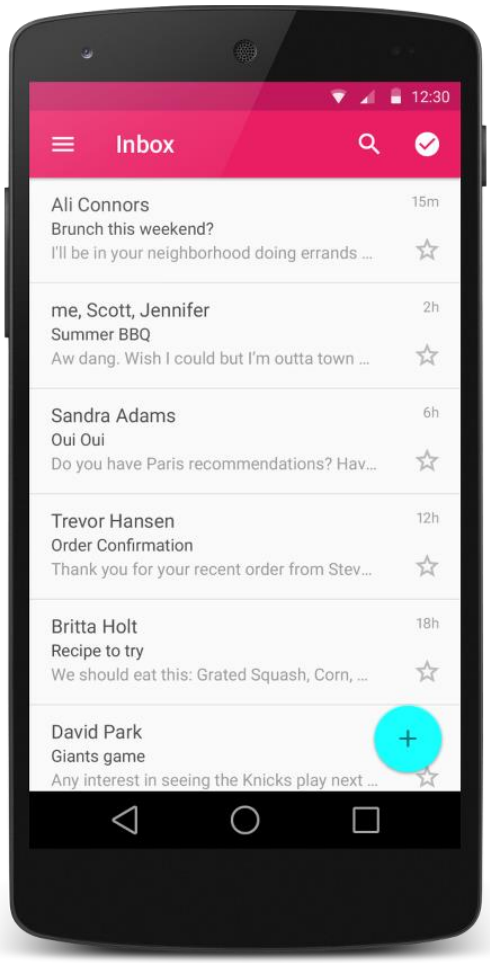

Рис 7. Пример использования ListView

После изучения структуры раздела направления на сайте предприятия был создан прототип раздела в приложении (Рис 8).

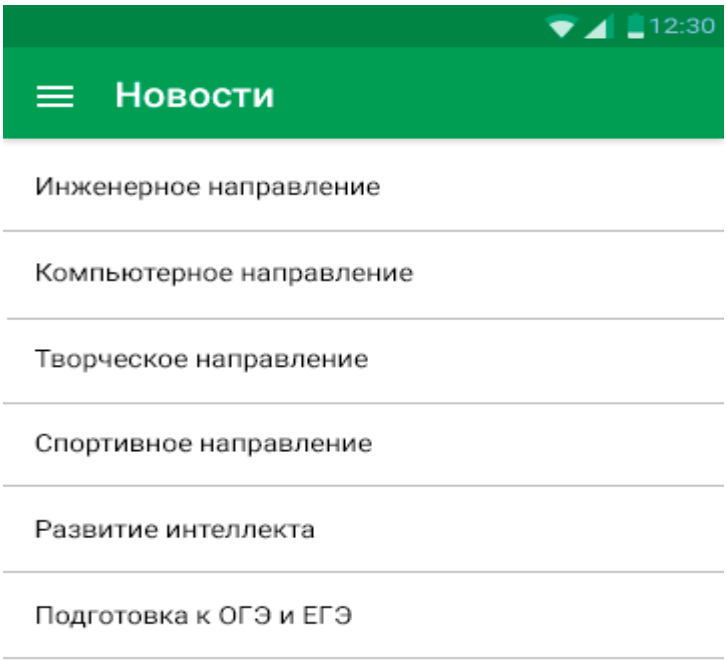

Рис 8. Раздел направления

Дальнейшее прототипирование раздела направления было направлено на окно с информацией о направлении, был выбран вариант с появление нового окна. В данном окне будет отображаться информация доступная пользователю. Сначала должна отображаться картинка с изображением заданным на удалённом сервере, снизу будет находиться подробная информация в определенном, удобном форматировании. Для удобной навигации сверху-слева находиться кнопка назад для закрытия окна с подробной информацией. Пользователь также должен понимать, что находиться в новом окне для этого меняется заголовок окна на название открытого направления (Рис 9).

Рис 9. Подробная информация о направлении

Раздел с контактной информацией имеет простую форму отображения текстовой информации (Рис 10).

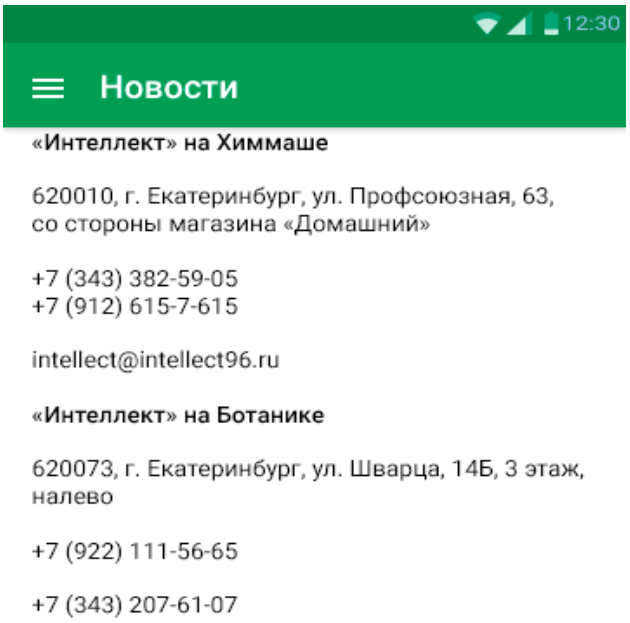

Рис 10. Раздел контакты

Последний раздел “О нас” сделан по образу и подобию раздела с сайта и имеет такой же дизайн. Сначала в разделе идут сотрудники предприятия в виде двух колонок и множестве строк (Рис 11).

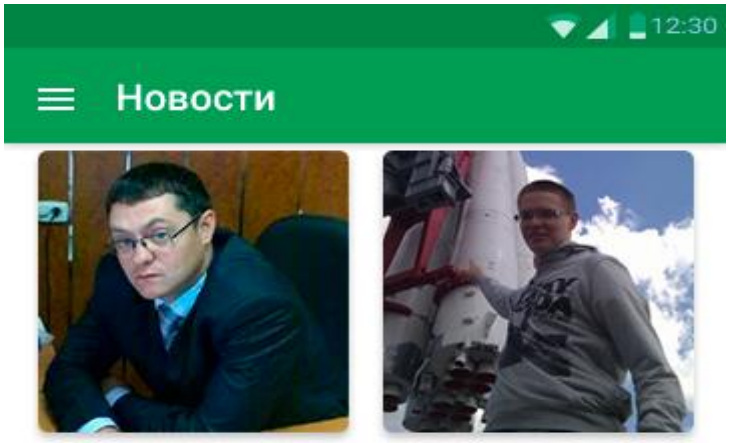

Рис 11. Раздел “О нас” – Список сотрудников

Когда пользователь нажмет на сотрудника в списке, то должно открыться новое окно с подробной информацией о сотруднике (Рис 12).

Рис 12. Подробная информация о сотруднике

2.4 Создание мобильного приложения

Стремительный рост продаж устройств на базе Android открывает выдающиеся возможности перед разработчиками приложений Android.

На платформе Android сейчас работают смартфоны, планшеты, электронные книги, роботы, реактивные двигатели, спутники NASA, игровые приставки, холодильники, телевизоры, камеры, медицинские устройства, «умные часы», автомобильные информационные системы (для управления радио, GPS, телефонами, термостатами и т. д.) и многие другие устройства.

По последним прогнозам, доходы от мобильных приложений (по всем мобильным платформам) к 2019 году достигнут 99 миллиардов долларов.

Одно из главных преимуществ платформы Android — ее открытость. Операционная система Android построена на основе открытого исходного кода и находится в свободном распространении. Это позволяет разработчикам получить доступ к исходному коду Android и понять, каким образом реализованы методы, свойства и функции приложений. Любой

пользователь может принять участие, где можно получить исходный код Android, узнать об идеологии, заложенной в основу операционной системы с открытым кодом, и получить лицензионную информацию.

Открытость платформы способствует быстрому обновлению. В отличие от закрытой системы iOS компании Apple, доступной только на устройствах Apple, система Android доступна на устройствах десятков производителей оборудования (OEM, Original Equipment Manufacturer) и телекоммуникационных компаний по всему миру. Все они конкурируют между собой, что идет на пользу конечному потребителю.

Для разработки приложений для ОС Android требуется установить Android Studio.

Установка Android SDK. Инструменты для разработки Android SDK можно загрузить на сайте для разработчиков. При установке можно выбрать требующиеся для разработки платформы и элементы SDK.

При разработке приложений Android используется Java — один из наиболее распространенных языков программирования. Использование Java стало логичным выбором для платформы Android, потому что это мощный, свободный и открытый язык, известный миллионам разработчиков. Опытные программисты Java могут быстро освоить Android-программирование, используя интерфейсы Google Android API (Application Programming Interface) и другие разработки независимых фирм.

Язык Java является объектно-ориентированным, предоставляет разработчикам доступ к мощным библиотекам классов, ускоряющих разработку приложений.

Программирование графического интерфейса пользователя управляется событиями, которые реагируют на инициируемые пользователями события, такие как касания экрана. Помимо непосредственного написания кода приложений можно воспользоваться средами разработки Eclipse и Android Studio, позволяющими собирать графический интерфейс из готовых объектов, таких как кнопки и текстовые

поля, перетаскивая их в определенные места экрана, добавляя подписи и изменяя их размеры.

Эти среды разработки позволяют быстро и удобно создавать, тестировать и отлаживать приложения Android.

Компоненты графического интерфейса в Android называются представлениями (views). Вертикальное представление LinearLayout используется для размещения текста и графики так, чтобы каждое представление занимало половину вертикального пространства LinearLayout. Компонент LinearLayout также позволяет размещать представления по горизонтали. Для вывода текста в приложении будет использоваться компонент TextView, а графика будет отображаться в компоненте ImageView. Графический интерфейс, созданный для приложения по умолчанию, содержит компонент TextView. Различные параметры этого компонента — текст, размер шрифта, цвет текста, размер относительно компонента ImageView в LinearLayout и т. д. — настраиваются в окне свойств среды разработки. Затем мы перетащим компонент ImageView с палитры в макет графического интерфейса и настроим его свойства, включая источник графических данных и позицию в LinearLayout.

Язык XML (eXtensible Markup Language, то есть расширяемый язык разметки) является естественным способом описания графических интерфейсов. Разметка XML хорошо читается как человеком, так и компьютером; в контексте Android она используется для описания макетов используемых компонентов и их атрибутов: размера, позиции, цвета, размера текста, полей и отступов. Android Studio разбирает разметку XML, чтобы отобразить макет в макетном редакторе и сгенерировать код Java, формирующий графический интерфейс на стадии выполнения. Также файлы XML используются для хранения ресурсов приложения: строк, чисел, цветов и т. д.

У каждого приложения существует тема, определяющая оформление стандартных компонентов, которые мы используем. Тема приложения

указывается в файле AndroidManifest.xml приложения. Мы можем настроить различные аспекты темы (например, составляющие цветовой схемы), определяя ресурсы в файле styles.xml, находящемся в папке res/values приложения.

Ресурсный файл style.xml содержит стиль с именем "AppTheme", ссылка на который включается в файл AndroidManifest.xml приложения для назначения темы. Этот стиль также определяет родительскую тему, которая может рассматриваться как аналог суперкласса в Java — новый стиль наследует атрибуты родительской темы и их значения по умолчанию. Как и субкласс Java, стиль может переопределить атрибуты родительской темы значениями, адаптированными для конкретных приложений (например, для использования в приложении фирменной цветовой гаммы компании). Мы используем эту концепцию для настройки трех цветов, используемых в теме приложения. Как упоминалось ранее, шаблоны приложений Android Studio теперь включают поддержку библиотек AppCompat, позволяющих использовать новые возможности Android в старых версиях платформы. По умолчанию Android Studio выбирает родительскую тему Theme.AppCompat.Light.DarkActionBar, одну из нескольких стандартных тем в библиотеке AppCompat — приложения, использующие эту тему, отображаются на светлом фоне, а в верхней части приложения располагается темная панель приложения. Все темы AppCompat используют рекомендации материального дизайна Google для оформления графических интерфейсов.

Установка JDK и JRE. Для разработки требуется среда исполнения Java Runtime Environment (JRE), комплект разработчика Java Development Kit (JDK), которые можно загрузить с официального сайта Oracle.

Создание виртуального устройства Android. Android tools включает в себя эмулятор «Android Virtual Device» (AVD). Эмулятор AVD позволяет тестировать приложения на виртуальном мобильном устройстве с ОС Android. Эмулятор позволяет создавать несколько виртуальных устройств с разными конфигурациями.

Проект — это группа связанных файлов (например, файлы кода, ресурсы и графические файлы), образующих приложение. Работа над приложением начинается с создания проекта. Чтобы создать приложение в среде Android Studio для операционной системы Android. Откроем Android Studio. Для создания нового проекта надо перейти к пункту меню File -> New-> New Project.... После этого у нас отобразится диалоговое окно создания нового проекта:

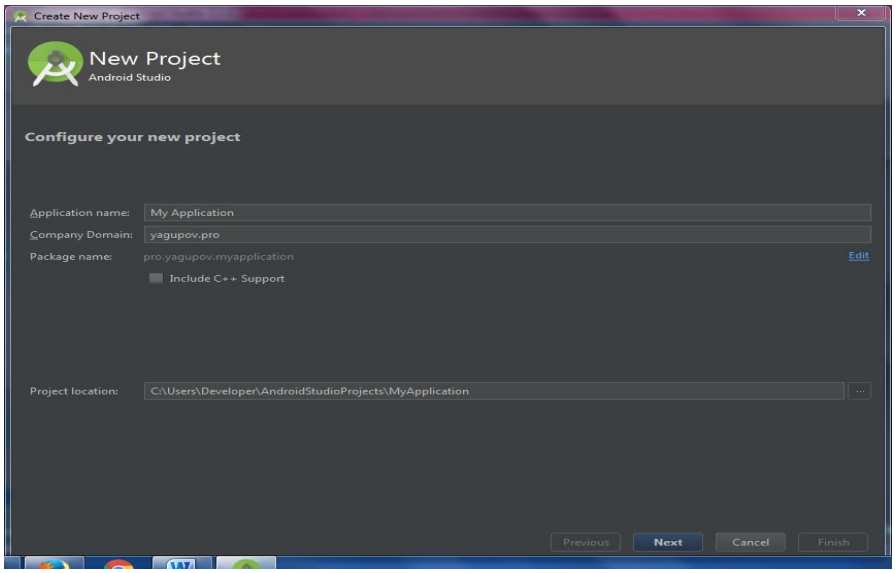

Рис 12. Окно создания нового проекта

На шаге Configure your new project мастера Create New Project (рис. 12) введем следующую информацию.

1. Application Name: — имя приложения.
2. Company Domain: — доменное имя веб-сайта компании. создания можно использовать имя example.com.
3. Package Name: — имя пакета Java для исходного кода приложения. Android и магазин Google Play используют это имя в качестве уникального идентификатора приложения, которое должно оставаться постоянным во всех версиях приложения, которые мы будем отправлять в магазин Google Store. Имя пакета обычно начинается с доменного имени вашей компании или учреждения, записанного в обратном порядке. Например, мы используем доменное имя deitel.com,

поэтому имена наших пакетов начинаются с префикса example.com. Далее обычно следует имя приложения. По общепринятым соглашениям в имени пакета используются только буквы нижнего регистра без пробелов. По умолчанию IDE выбирает имя пакета на основании текста, вводимого в полях Application Name и Company Domain. Чтобы изменить имя Package, щелкните на ссылке Edit справа от сгенерированного имени пакета.

4. Project Location: — путь к папке на вашем компьютере, в которой будет храниться проект. По умолчанию Android Studio размещает папки новых проектов во вложенной папке AndroidStudioProjects каталога учетной записи пользователя. Имя папки проекта состоит из имени проекта, из которого удаляются пробелы. Мы можем изменить путь к папке проекта; для этого введите путь или щелкните на кнопке (…) справа от поля и выберите папку для хранения проекта. После того как папка будет выбрана, щелкните на кнопке OK, а затем перейдите к следующему шагу кнопкой Next.

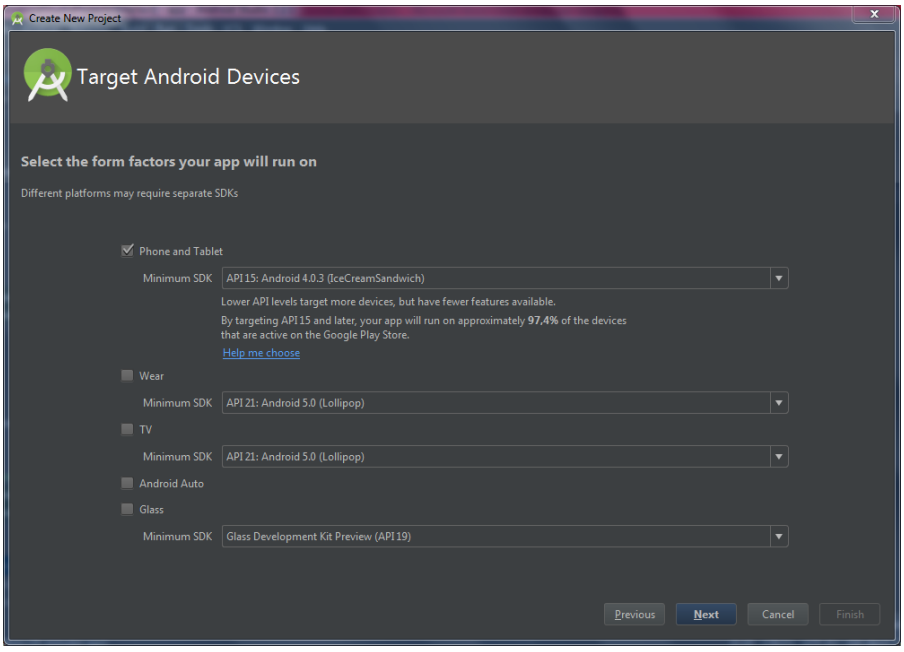

Рис 13. Выбор версии Android

На этом шаге будет предложено установить минимальную поддерживаемую версию проекта. По умолчанию устанавливается версия Android 4.0.3, что покрывает почти 97% устройств Android. Оставим по умолчанию и нажмем на кнопку Next.

На следующем шаге надо выбрать Activity, которая будет использоваться по умолчанию для главного экрана приложения:

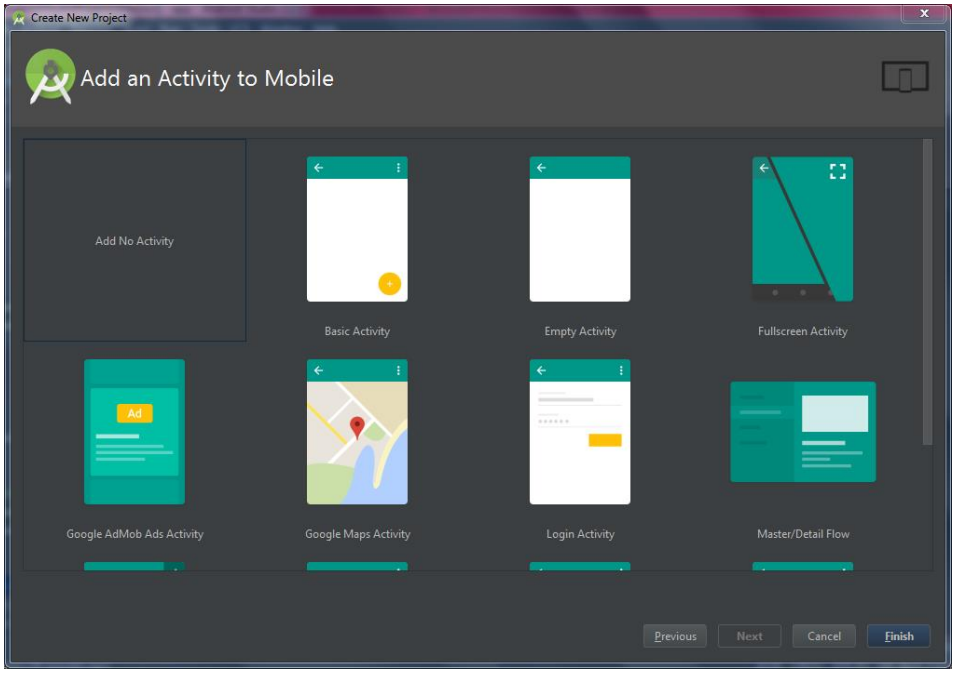

Рис 14. Выбор главной Activity

В нашем приложении мы не будем добавлять Activity обычным образом, потому что происходит не эффективная генерация программного кода. Позднее будет создана базовая реализация родительской Activity для остальных окон приложения, для соблюдения принципа DRY(*не повторяйся*)[29].

Рассмотрим структуру проекта приложения под ОС Android, которая создается по умолчанию.

Проект может включать различные модули. И все модули описываются файлом setting.gradle.

И если мы посмотрим на структуру проекта, то весь значимый код - файлы интерфейса, классы java и т.д. у нас по умолчанию находятся в папке (модуле) app

Файл build.gradle содержит информацию, которая используется при построении проекта.

Каждый модуль имеет свой файл build.gradle, который определяет конфгурацию построения проекта, специфичную для данного модуля. Так, если мы посмотрим на содержимое папки app, то, как раз найдем в ней такой файл. На начальном этапе данные файлы не столь важны, достаточно лишь понимать, для чего они нужны.

По умолчанию каждый проект включает один модуль - app. Собственно весь код, с которым мы будем работать, располагается внутри этого модуля.

В этом модуле мы можем увидеть несколько папок и файлов, из которых для нас важнейшими являются:

каталог libs - предназначен для хранения библиотек, используемых приложением

каталог src - предназначен для хранения исходного кода. Он содержит ряд подкаталогов. Исходные коды располагаются в папке main.

Папка main имеет сложную структуру:

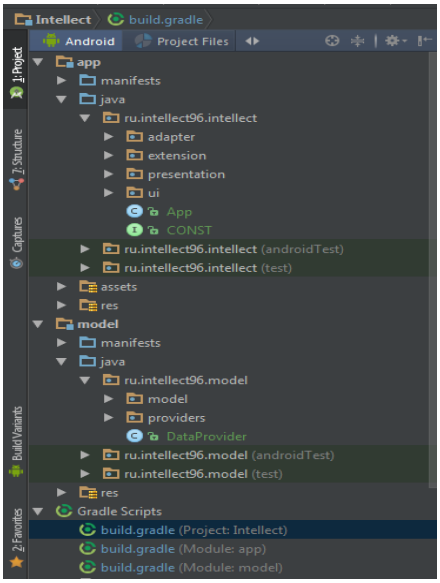

Рис 15. Структура Android проекта

После рассмотрения общих вопросов построения проекта мы уже можем приступить к самой реализации приложения. Начнем с создание со служебного класса App отвечающего за инициализацию вспомогательных компонентов:

```
public class App extends Application {

    @Override
    public void onCreate() {
        super.onCreate();

        DataProvider.onCreate(getApplicationContext());
    }
}
```

Код 1. Класс App

Класс App наследуется от служебного системного класса Application который реализует фундаментальную концепцию приложения , у которого переопределен метод onCreate, который вызывается при запуске приложения и происходит инициализация служебного класса нашего приложения.

Activity является основой для построения визуального интерфейса Android приложений, и мы в нашем случае разработаем каркас для дальнейшего использования, который будет облегчать нам модификацию приложения в дальнейшем, для этого мы создадим класс BaseActivity который будет объявлен абстрактным, что значит что создать на прямую без наследования будет невозможно:

```
public abstract class BaseActivity extends AppCompatActivity {

    @Nullable
    @BindView(R.id.toolbar)
    Toolbar mToolbar;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(getLayoutId());

        if (mToolbar != null) {
            mToolbar.setNavigationIcon(R.drawable.ic_arrow_back);
            setSupportActionBar(mToolbar);
```

```
            getSupportActionBar().setHomeButtonEnabled(true);
        }

        onAction();
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home: {
                finish();
                return true;
            }
        }

        return super.onOptionsItemSelected(item);
    }

    protected abstract void onAction();

    protected abstract int getLayoutId();
}
```

Код 2. Реализация базового класса Activity

В отличие от многих приложений Java, приложения Android не содержат метода main. Вместо этого в них используются четыре типа исполняемых компонентов — активности (activities), службы (services), провайдеры контента и широковещательные приемники (broadcast receivers). Приложение может иметь много активностей, одна из которых — первое, что мы видим при запуске приложения. Пользователи взаимодействуют с активностями через представления (views) — компоненты GUI, наследующие откласса View (пакет android.view).

До выхода Android 3.0 с каждым экраном приложения обычно связывалась отдельная активность. Активность может управлять несколькими фрагментами (fragments). На телефоне каждый фрагмент обычно занимает целый экран, а активность переключается между фрагментами на основании взаимодействий пользователя. На планшетах активности часто отображают несколько фрагментов на экран, чтобы более эффективно использовать доступное пространство.

На протяжении своего существования активность может находиться в одном из нескольких состояний — активном (то есть выполняемом), приостановленном или остановленном. Переходы активностей между этими состояниями происходят в ответ на различные события. «Активная активность» отображается на экране и «обладает фокусом» — то есть взаимодействует с пользователем. Приостановленная активность видна на экране, но не обладает фокусом (например, на время отображения диалогового окна с сообщением). Пользователь не может взаимодействовать с приостановленной активностью, пока она снова не станет активной — например, после того, как пользователь закроет диалоговое окно. Остановленная активность не отображается на экране и, вероятно, будет уничтожена системой, когда потребуется освободить занимаемую ею память. Активность останавливается, когда другая активность переходит в активное состояние. Например, когда мы отвечаем на телефонный звонок, приложение, управляющее звонками, становится активным, а предыдущее приложение останавливается. При переходах активности между этими состояниями исполнительная среда Android вызывает различные методы жизненного цикла (все эти методы определяются в классе Activity из пакета android.app). В приложениях для каждой активности будет переопределяться метод onCreate. Этот метод вызывается исполнительной системой Android при запуске активности — то есть когда ее графический интерфейс готов к отображению, чтобы пользователь мог взаимодействовать с активностью. Также у активностей существуют другие методы жизненного цикла: onStart, onPause, onRestart, onResume, onStop и onDestroy.

Каждый переопределяемый вами метод жизненного цикла активности должен вызывать версию метода из суперкласса; в противном случае происходит исключение. Вызов версии суперкласса необходим, потому что каждый метод жизненного цикла в суперклассе Activity содержит код, который должен выполняться помимо кода, определяемого вами в переопределенных методах жизненного цикла.

При использовании новых возможностей на более ранних платформах Android разработчик сталкивается с проблемой обеспечения совместимости. Теперь Google открывает доступ ко многим новым возможностям Android через Android Support Library — набор библиотек, благодаря которым разработчик может использовать эти возможности на современных и старых платформах Android. Одна из таких библиотек — AppCompat — обеспечивает поддержку панели приложения (которая прежде называлась панелью действий) и других возможностей на устройствах Android 2.1 (API 7) и выше; напомним, что панель приложения впервые появилась в Android 3.0 (API 11). Обновленные шаблоны приложений Android Studio тоже используют библиотеку AppCompat, поэтому новые приложения, которые мы создадим, будут работать почти на всех устройствах Android. Шаблон Android Studio Empty Activity определяет класс MainActivity приложения как субкласс AppCompatActivity (пакет package android.support.v7.app) — непрямого субкласса Activity, обеспечивающего использование новых средств Android на современных и старых платформах Android.

Android также поддерживает неявные (implicit) интенты — разработчик не указывает компонент для обработки интента. Например, можно создать интент для отображения содержимого URL-адреса и поручить Android запустить наиболее подходящую активность (браузер) в зависимости от типа данных. Если действие и информация, переданная startActivity, могут быть обработаны несколькими активностями, система выводит диалоговое окно, в котором пользователь выбирает активность. Если система не может найти активность для обработки действия, метод startActivity инициирует исключение ActivityNotFoundException. В общем случае рекомендуется предусмотреть обработку этого исключения в программах. Также можно предотвратить возникновение самого исключения — используйте метод resolveActivity класса Intent для определения того, существует ли активность для обработки интента.

Для любой Activity вызывается метод onCreate, он отвечает за создание окна. Для того чтобы можно было отобразить вид окна вызывается метод setContentView в который передается числовое значение идентификатора ресурса отображения, получаемого от класса потомка. Здесь так же устанавливается ActionBar, который есть у всех Activity приложения.

Следующий метод onOptionsItemSelected отвечает за обработку нажатия меню, которое находится в ActionBar в виде стандартной кнопки “Назад”. В случае если кнопка была нажата пользователем, то тогда происходит завершение работы находящейся на переднем плане Activity. Ниже идут два абстрактных метода onAction и getLayoutId, которые обязаны реализовать потомки данного класса, первый метод вызывается у потомка после инициализации предыдущих элементов метода onCreate, второй при запросе идентификатора ресурса вида.

Следующим ключевым элементом приложения является класс ViewFactory который отвечает за создание классов представлений приложения, которые имеют интерфейс BaseView который унифицирует работу с представлениями в приложении.

```
public class ViewFactory {

    public static final int
            BACK_CONTENT_VIEW       = 0,
            MAIN_VIEW               = 1,
            NEWS_VIEW               = 2,
            NEWS_ITEM_VIEW          = 3,
            DIRECTIONS_VIEW         = 4,
            DIRECTIONS_ITEM_VIEW    = 5,
            CONTACTS_VIEW           = 6,
            ABOUT_US                = 7;

    private ViewFactory() {
    }

    public static BaseView create(int pViewId) {
        switch (pViewId) {
            case BACK_CONTENT_VIEW:
                return new BackContentActivity();
            case MAIN_VIEW:
                return new MainActivity();
            case NEWS_VIEW:
```

```
                return new NewsFragment();
            case NEWS_ITEM_VIEW:
                return new NewsItemFragment();
            case DIRECTIONS_VIEW:
                return new DirectionsFragment();
            case DIRECTIONS_ITEM_VIEW:
                return new DirectionItemFragment();
            case CONTACTS_VIEW:
                return new ContactFragment();
            case ABOUT_US_VIEW:
                return new AboutFragment();
            default:
                throw new IllegalArgumentException("Can't find a view");
        }
    }
}
```

Код 3. Фабрика представлений

Данная фабрика является реализацией паттерна фабричный метод[30], который позволяет нам получить готовый экземпляр вида без явного инстанцирования объекта, что позволяет скрывать способ его создания. Так как классы вида у нас реализую один интерфейс BaseView мы можем создавать и возвращать их без каких либо проблем, потому что программист не будет знать с каким именно типом объекта он работает, что делает программу более гибкой, легко модифицируемой и расширяемой.

Приступим к рассмотрению интерфейса BaseView который представляет возможность работать универсально с классами представлений:

```
public interface BaseView {

    void show(@NonNull final Context pContext, @Nullable final Bundle
pArguments);

    void show(@NonNull final Context pContext, @Nullable final Bundle
pArguments, int pRequestCode);

    void show();

    void hide();

    void close();
}
```

Код 4. Интерфейс BaseView

Данный интерфейс является важной частью приложения, благодаря которому приложение можно будет легко расширять, добавляя новые представления без больших усилий, и временных затрат, что является важным для разработки мобильных приложений. Его реализую многие представления приложения, но реализация находятся в основном в базовых классах данных представлений. Как мы могли заметить BaseActivity не реализует данный интерфейс, и для этого есть некоторые причины, например стартовое окно приложения, вызывает система, и сами мы его запускать не будем, так зачем нам тогда реализовывать данный интерфейс?

Класс представления SplashActivity наследует от BaseActivity, но не реализует интерфейс BaseView, так как в данной ситуации это не нужно.

```
public class SplashActivity extends BaseActivity {

    private static final int DELAY_SEC = 0 * 1000;


    @Override
    protected int getLayoutId() {
        return R.layout.activity_splash;
    }

    @Override
    protected void onAction() {
        new Handler().postDelayed(() -> {
            ViewFactory.create(ViewFactory.MAIN_VIEW).show(this, null);
            finish();
        }, DELAY_SEC);
    }
}
```

Код 5. Окно загрузки приложения

В методе onAction мы создаем обработчик с задержкой на одну секунду для отображения окна приложения. После того как пройдет время, будет выполнен фрагмент кода в блоке который выполнен с применением замыканий[31] из синтаксиса Java 8. Произойдет запуск главного окна и закрытие текущего.

Не менее важным классом представления является BaseViewActivity который является родительским для остальным представлений реализующих интерфейс BaseView:

```
public abstract class BaseViewActivity<P extends BasePresenter> extends
BaseActivity implements BaseView {

    private P mPr;

    @Override
    public void show(@NonNull final Context pContext, @Nullable final Bundle
pArguments) {
        Intent intent = new Intent(pContext, getClass());
        if (pArguments != null) intent.putExtras(pArguments);
        pContext.startActivity(intent);
    }


    @Override
    public void hide() {
        moveTaskToBack(true);
    }

    @Override
    public void close() {
        finish();
    }

    @Override
    protected void onAction() {
        onSetup();

        mPr = PresenterFactory.create(this);
        mPr.onCreate();
    }

    protected P pr() {
        return mPr;
    }


    protected abstract void onSetup();
}
```

Код 6. Базовое представление

Реализованный метод show данного класса реализует стандартную логику отображения Activity в экосистеме Android, метод hide прячет окно

приложения, close закрывает окно, но остается в стеке приложений, onAction вызывает абстрактный метод onSetup и далее создает Presenter и вызывает его метод onCreate для его инициализации, метод pr возвращает Presenter.

С данного класса начитает работать пользовательского паттерн MVP[32].

MVP — шаблон проектирования пользовательского интерфейса, который был разработан для облегчения автоматического модульного тестирования и улучшения разделения ответственности в презентационной логике (отделения логики от отображения):

- Модель (англ. Model) — предоставляет данные для пользовательского интерфейса.
- Представление (англ. View) — реализует отображение данных (Модели) и маршрутизацию пользовательских команд или событий Presenter'у.
- Presenter — управляет Моделью и Представлением. Например, извлекает данные из Модели и форматирует их для отображения в Представлении.

Обычно экземпляр Представления создаёт экземпляр Presenter'а, передавая ему ссылку на себя. При этом Presenter работает с Представлением в абстрактном виде, через его интерфейс. Когда вызывается событие Представления, оно вызывает конкретный метод Presenter'а, не имеющего ни параметров, ни возвращаемого значения. Presenter получает необходимые для работы метода данные о состоянии пользовательского интерфейса через интерфейс Представления и через него же передаёт в Представление данные из Модели и другие результаты своей работы.

После рассмотрения архитектуры паттерна MVP мы можем представить базовый класс нашего презентера:

```
public class BasePresenterImpl<V extends BaseView> implements BasePresenter {

    private V mView;

    public BasePresenterImpl(V pView) {
```

```
        mView = pView;
    }

    protected V view() {
        return mView;
    }

    public void onCreate() {}

    public void onResume() {}

    public void onStop() {}

    public void onDestroy() {}
}
```

Код 7. Базовый презентер

В классе BasePresenterImpl(Код 7) находиться минимум кода, так как наше приложение не является огромной системой и не требует обширного функционала, и имеет заглушки для некоторых методов, содержит ссылку на представление для доступа к его функционалу. Остальные презентеры должны наследовать от базового класса, чтобы получить его функционал.

Презентеры также как представления создаются в фабрике, которая создает их по классу представления:

```
public class PresenterFactory {

    private PresenterFactory() {
    }

    @SuppressWarnings("unchecked")
    public static <P extends BasePresenter, V extends BaseView> P create(V
pView) {

        if (pView instanceof MainView) return (P) new
MainPresenterImpl((MainView) pView);
        if (pView instanceof NewsView) return (P) new
NewsPresenterImpl((NewsView) pView);
        if (pView instanceof DirectionView) return (P) new
DirectionPresenterImpl((DirectionView) pView);

        return (P) new BasePresenterImpl(pView);
    }
}
```

Код 8. Фабрика презентеров

Фабрика для распознавания класса представления использует внутренние механизмы языка Java – рефлексию. В информатике отражение или рефлексия означает процесс, во время которого программа может отслеживать и модифицировать собственную структуру и поведение во время выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов. Можно также выполнять операции над полями и методами, которые исследуются. Рефлексия в Java осуществляется с помощью Java Reflection API.

Теперь пришло время рассмотреть функциональность главного представления приложения, которым является класс MainActivity:

```
public class MainActivity extends BaseViewActivity<MainPresenter>
{

    @Override
    public void onSetup() {
        createNavigationMenu();
        onNavigationItemSelected(mNavigationView.getMenu().getItem(0));
    }

    private void createNavigationMenu() {
        mNavigationView.setNavigationItemSelectedListener(this);
        setDrawerToggle();
    }

    private void setDrawerToggle() {
        mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
                R.string.drawer_open, R.string.drawer_close);

        mDrawerLayout.setDrawerListener(mDrawerToggle);

        mDrawerToggle.setDrawerIndicatorEnabled(true);
        mDrawerToggle.syncState();
    }

    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.news:
                ViewFactory.create(ViewFactory.NEWS_VIEW).show(this, null);
                break;
            case R.id.directions:
                ViewFactory.create(ViewFactory.DIRECTIONS_VIEW).show(this,
null);
                break;
```

```
            case R.id.contacts:
                ViewFactory.create(ViewFactory.CONTACTS_VIEW).show(this,
null);
                break;
            case R.id.about_us:
                ViewFactory.create(ViewFactory.ABOUT_US_VIEW).show(this,
null);
                break;
        }
```

Код 9. MainActivity

Данный класс является отправной точкой приложения, отсюда можно открыть все представления приложения. Сначала создается левое меню приложения, далее устанавливается кнопка для открытия меню.

Пользователь может открыть меню двумя способами:

- Нажать на кнопку меню
- Свайпом вправо

Откроется меню и можно будет выбрать пункт меню, после чего в том же окне откроется другой раздел.

Пришло время приступить к созданию разделов приложения, которые базируются на новом для Android компоненте под названием Fragment. Фрагмент (класс Fragment) представляет поведение или часть пользовательского интерфейса в Activity. Разработчик может объединить несколько фрагментов в одну Activity для построения многопанельного пользовательского интерфейса и повторного использования фрагмента в нескольких Activity. Фрагмент можно рассматривать как модульную часть Activity. Такая часть имеет свой жизненный цикл и самостоятельно обрабатывает события ввода. Кроме того, ее можно добавить или удалить непосредственно во время выполнения Activity. Это нечто вроде вложенной Activity, которую можно многократно использовать в различных Activity.

Фрагмент (fragment) обычно представляет повторно используемую часть пользовательского интерфейса активности, но он также может представлять повторно используемый блок программной логики. Приложение использует фрагменты для создания частей графического интерфейса и управления ими. Фрагменты можно объединять для создания

интерфейсов, эффективно использующих размер экрана планшета. Кроме того, простой механизм замены фрагментов сделает интерфейс приложения более динамичным. Базовым классом всех фрагментов является класс Fragment (пакет android.app). При использовании субклассов AppCompatActivity с фрагментами необходимо использовать версию этого класса из библиотеки Android Support Library (пакет android.support.v4.app).

Каждый фрагмент, как и активность, имеет свой жизненный цикл и предоставляет методы, которые можно переопределять для обработки событий жизненного цикла. В этом приложении будут переопределены следующий метод:

 onCreateView — этот метод вызывается после onCreate; он должен построить и вернуть объект View с графическим интерфейсом фрагмента. Как мы вскоре увидем, он получает объект LayoutInflater, который используется для программного заполнения графического интерфейса фрагмента по компонентам, заданным в заранеео пределенном макете в формате XML.

Жизненный цикл фрагмента связывается с жизненным циклом его родительской активности. Существуют шесть методов жизненного цикла активности, у которых имеются соответствующие методы жизненного цикла фрагмента — onCreate, onStart, onResume, onPause, onStop и onDestroy. Когда система вызывает эти методы для активности, это приводит к вызову соответствующих методов всех присоединенных фрагментов активности (а возможно, и других методов жизненного цикла фрагментов). В данном приложении используются методы жизненного цикла фрагмента onResume и onPause. Метод onResume вызывается в тот момент, когда фрагмент находится на экране и готов к взаимодействию с пользователем. Когда активность, управляющая фрагментами, возобновляет работу, вызываются методы onResume всех ее фрагментов. Когда активность, управляющая фрагментами, приостанавливается, вызываются методы onPause всех ее фрагментов.

Фрагменты могут добавлять свои команды в меню управляющей активности. Как класс Activity, класс Fragment содержит метод жизненного цикла onCreateOptionsMenu и метод обработки события onOptionsItemSelected.

Метод onCreateOptionsMenu вызывается для инициализации стандартного меню активности — этот метод вместе с методом onOptionsItemSelected автоматически генерируется шаблоном Android Studio Blank Activity. Система передает объект Menu, в котором должны отображаться команды. В нашем приложении меню должно отображаться только при запуске приложения в портретной ориентации. Объект Resources класса Activity (возвращаемый унаследованным методом getResources) используется для получения объекта Configuration (возвращаемого методом getConfiguration), представляющего текущую конфигурацию устройства. Открытая переменная экземпляра orientation содержит признак Configuration.ORIENTATION_PORTRAIT или Configuration. ORIENTATION_LANDSCAPE. Метод getMenuInflater, унаследованный от Activity, возвращает объект MenuInflater, для которого вызывается метод inflate с двумя аргументами — идентификатором ресурса меню, используемого для заполнения меню, и объектом Menu, в который будут помещены команды меню. Если метод onCreateOptionsMenu возвращает true, это означает, что меню должно отображаться на экране.

Фрагмент всегда должен быть встроен в Activity, и на его жизненный цикл напрямую влияет жизненный цикл Activity. Например, когда операция приостановлена, в том же состоянии находятся и все фрагменты внутри нее, а когда Activity уничтожается, уничтожаются и все фрагменты. Однако пока Activity выполняется (это соответствует состоянию возобновления жизненного цикла), можно манипулировать каждым фрагментом независимо, например, добавлять или удалять их. Когда разработчик выполняет такие транзакции с фрагментами, он может также добавить их в стек переходов назад, которым управляет операция. Каждый элемент стека переходов назад

в операции является записью выполненной транзакции с фрагментом. Стек переходов назад позволяет пользователю обратить транзакцию с фрагментом (выполнить навигацию в обратном направлении), нажимая кнопку “ Назад”.

Родительская активность использует для управления своими фрагментами объект FragmentManager (пакет android.app), возвращаемый методом getFragmentManager класса Activity. Если активности потребуется взаимодействовать с фрагментом, который объявлен в макете активности и обладает идентификатором id, то для получения ссылки на заданный фрагмент активность может вызвать метод findFragmentById класса FragmentManager. FragmentManager может использовать объекты FragmentTransaction для динамического добавления, удаления и переключения между фрагментами.

Мы узнали, что такое фрагменты и теперь можно приступить к реализации базового класса для остальных фрагментов. Все фрагменты попадают в категорию представлений и поэтому реализуют интерфейс BaseView или реализуют потомков данного интерфейса. Ниже представлен код BaseViewFragment (Код 10) который будет базовым:

```
public abstract class BaseFragment<P extends BasePresenter> extends Fragment
implements BaseView {

    @Override
    public void show(@NonNull final Context pContext, @Nullable final Bundle
pArguments) {
        if (pArguments != null) setArguments(pArguments);
        ((BaseActivity) pContext)
                .getSupportFragmentManager()
                .beginTransaction()
                .replace(R.id.container, this)
                .addToBackStack(null)
                .commit();
    }

    @Override
    public void hide() {
        if (isAdded() && isVisible()) {
            getFragmentManager()
                    .beginTransaction()
                    .hide(this)
```

```
                    .commit();
            }
        }

        @Override
        public void close() {
            if (isAdded()) {
                getActivity()
                        .getSupportFragmentManager()
                        .popBackStack();
            }
        }

        @Override
        public void onViewCreated(View view, @Nullable Bundle savedInstanceState)
{
            super.onViewCreated(view, savedInstanceState);

            mPr = PresenterFactory.create(this);

            onSetup();

            mPr.onCreate();
        }

        protected void onSetup() {}

        @SuppressWarnings("unchecked")
        protected <T> T getArgument(String pArgName) {
            return (T) getArguments().get(pArgName);
        }

        protected boolean hasArgument(String pArgName) {
            return getArguments() != null &&
getArguments().containsKey(pArgName);
        }
}
```

Код 10. Фрагмент BaseViewFragment

Как можно заметить, некоторыми местами код фрагмента похож на код BaseActivityView, по этой причине они оба выбраны в одну категорию для реализации интерфейса BaseView, и каждый из них отвечает за свое отображение, но различия на уровне реализации существенны. Фрагменты зависимы от Activity поэтому всегда имеют к ним доступ. В методе show переданный контекст явно преобразуется к BaseView чтобы вызвать менеджер фрагментов для начала транзакции отображения фрагмента в

Activity. Остальные методы, которые есть в классе, похожи на методы, которые мы реализовывали в BaseActivityView, и особого интереса не представляют.

Можно приступить к реализации раздела с новостями, в котором у нас будет список новостей с картинками и полосой прокрутки:

```
public class NewsFragment extends BaseFragment<NewsPresenter> implements
        NewsView, LoadMoreSwipeRefreshLayout.OnUpRefreshListener,
        SwipeRefreshLayout.OnRefreshListener {


    @Override
    protected void onSetup() {
        initRefreshLayout();

        pr().getNews(mPage);
    }

    private void initRefreshLayout() {
        mRefreshLayout.setOnRefreshListener(this);
        mRefreshLayout.setOnUpRefreshListener(this);

        mRefreshLayout.setColorSchemeResources(R.color.colorPrimary,
R.color.colorAccent);
    }

    @Override
    public void showNews(List<News> pNews) {

        hideRefreshLoading();

        if (mNewsList.getCount() == 0) {
            mNewsAdapter = new NewsAdapter(pNews);
            mNewsList.setAdapter(mNewsAdapter);
        } else mNewsAdapter.addAll(pNews);
    }

    @Override
    public void onRefresh() {
        mPage = FIRST_PAGE;
        if (mNewsAdapter != null) mNewsAdapter.clear();
        pr().getNews(mPage);
    }
    @Override
    public void onUpRefresh() {
        mPage++;
        pr().getNews(mPage);
    }
```

```
    @OnItemClick(R.id.list)
    public void onItemClick(View pView) {
        Bundle args = new Bundle();
        args.putInt(CONST.VIEW_ID, ViewFactory.NEWS_ITEM_VIEW);
        args.putSerializable(CONST.CONTENT, ((News) ((BaseViewHolder)
pView.getTag()).item()));
        ViewFactory.create(ViewFactory.BACK_CONTENT_VIEW).show(getActivity(),
args);
    }
}
```

Код 11. Новостной фрагмент

Для работы раздела новостей необходим интернет, из которого будут загружаться данные. Сначала мы делаем запрос на сервер, откуда постранично получаем данные и на основе их формируем страницу. Новости можно обновить, потянув сверху экрана вниз и новости заново загрузиться. Чтобы пользователь понимал, что страница обновляется сверху находиться значок загрузки. Если пользователь пролистает новости до конца, то подгружаются еще десять или меньше новостей, при этом снизу отображается анимация загрузки. В случае если новости кончились, то загрузка прекращается. Если пользователь на элемент списка то происходит открытие нового окна, в котором находиться подробная информация о новости.

Чтобы картина разработки была четче, рассмотрим презентер новостного раздела:

```
public class NewsPresenterImpl extends BasePresenterImpl<NewsView> implements
NewsPresenter {

    public NewsPresenterImpl(NewsView pView) {
        super(pView);
    }

    @Override
    public void getNews(int pPage) {
        DataProvider.getInstance()
                .getNews(pPage)
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(pNewsList -> {
                    view().showNews(pNewsList);
                }, pThrowable -> {
```

```
                    view().error(pThrowable.getLocalizedMessage());
                });
    }
}
```

Код 12. Презентер новостного раздела

Исходный код презентера элегантен, содержит минимум кода, выполняет только поставленную задачу, запрашивает новости из провайдера в другом потоке и получает уже в главном потоке. В случае ошибки отображает ее в окне новостей.

ViewHolder является паттерном в Android для компактного хранения и отображения данных, является частью адаптера для ListView:

```
public class NewsHolder extends BaseViewHolder<News, BasePresenter>
implements NewsListItemView {

    @BindView(R.id.header)
    TextView header;
    @BindView(R.id.date)
    TextView date;
    @BindView(R.id.description)
    TextView description;

    public NewsHolder(View pView) {
        super(pView);
    }

    @Override
    protected void fill() {
        header.setText(Html.fromHtml(item().getTitle()));
        date.setText(item().getDate());

        if (TextUtils.isEmpty(item().getExcerpt())) {
            description.setText(Html.fromHtml(item().getTitle()));
        } else description.setText(Html.fromHtml(item().getExcerpt()));
    }
}
```

Код 13. Холдер списка новостей

Холдер занимается хранением и преобразованием сырых данных в представление.

Для отображения подробной информации открывается новое окно, в котором отображается контент. Данное окно реализовано с помощью

Activity, которое отображает в себе фрагмент, в котором уже отображается сам контент:

```
public class BackContentActivity extends BaseViewActivity {

    @Override
    protected void onSetup() {
        ViewFactory.create(getArgument(CONST.VIEW_ID)).show(this,
getArguments());
    }
}
```

Код 14. Окно с фрагментом внутри себя

Выбранная архитектура несет большие преимущества для дальнейшей разработки и поддержки приложения, так как мы можем отобразить любой фрагмент приложения, которое производиться в фабрике фрагментов, также у нас появляется возможность простой портации приложения на планшеты с низкими затратами на разработку.

Теперь можно рассмотреть сам фрагмент с подробной информацией о новости:

```
        public class NewsItemFragment extends
BaseWebViewFragment<BasePresenter> implements NewsItemView {

    @Override
    protected void onSetup() {
        News news = getArgument(CONST.CONTENT);
        getActivity().setTitle(Html.fromHtml(news.getTitle()));
        loadData(news.getContent());
    }
}
```

Код 14. Фрагмент с подробной информацией

Сам фрагмент базируется на другом фрагменте, который содержит в себе специальный компонент для отображения Html:

```
public abstract class BaseWebViewFragment<T extends BasePresenter> extends
BaseFragment<T> {

    @BindView(R.id.content)
    WebView mContentView;

    @Override
    protected int getLayoutId() {
        return R.layout.fragment_content;
    }
```

```
    protected void loadUrl(String pUrl) {
        mContentView.loadUrl(pUrl);
    }

    protected void loadData(String pData) {
        mContentView.loadData(pData, "text/html; charset=UTF-8", null);
    }
}
```

Код 15. Фрагмент для отображения контента

Данный фрагмент позволяет своим потомкам, отображать в себе различный контент который может быть загружен как локально, так и через интернет в виде ссылки на ресурс. Выбор пал на реализацию данной архитектуры из-за того что большинство контента у нас находится в формате Html и это удобно для работы.

Рассмотрим следующий раздел направлений, в котором отображается список направлений деятельности предприятия:

```
public class DirectionsFragment extends BaseFragment<DirectionPresenter>
implements DirectionView {

    @BindView(R.id.list)
    ListView mListView;

    @Override
    protected void onSetup() {
        if (hasArgument(CONST.PARENT))
pr().getDirection(getArgument(CONST.PARENT));
        else pr().getDirection(0);
    }

    @Override
    public void showDirections(List<Direction> pDirections) {
        mListView.setAdapter(new DirectionAdapter(pDirections));
    }

    @OnItemClick(R.id.list)
    void onDirectionClick(View view) {
        Direction direction = ((DirectionHolder) view.getTag()).item();
        Bundle args = new Bundle();
        if (direction.getType() == Direction.SUB_DIRECTION) {
            args.putInt(CONST.PARENT, direction.getId());

ViewFactory.create(ViewFactory.DIRECTIONS_VIEW).show(getActivity(), args);
        } else {
            args.putInt(CONST.VIEW_ID, ViewFactory.DIRECTIONS_ITEM_VIEW);
```

```
            args.putSerializable(CONST.CONTENT, direction);

ViewFactory.create(ViewFactory.BACK_CONTENT_VIEW).show(getActivity(), args);
        }
    }
}
```

Код 16. Фрагмент раздела направлений

Раздел работает рекурсивно, это значит, что если тип подраздел, тогда открывается этот же раздел с другим содержимым, а если тип раздел с контентом тогда откроется новое окно, в котором отображается информация.

Презентер направлений такой же, как и новостной:

```
public class DirectionPresenterImpl extends BasePresenterImpl<DirectionView>
implements DirectionPresenter {

    public DirectionPresenterImpl(DirectionView pView) {
        super(pView);
    }

    @Override
    public void getDirection(int pParent) {
        DataProvider.getInstance()
                .getDirection(pParent)
                .subscribe(directions -> {
                    view().showDirections(directions);
                });
    }
}
```

Код 17. Презентер направлений

Мы создали много различных классов для отображения, теперь пришло время создать классы провайдеров контента, и первый из них будет посредник между презентерами и интернет провайдером:

```
public class DataProvider implements RemoteProvider {

    private static DataProvider sDataProvider;

    private final Context mContext;

    private RemoteProvider mRemoteProvider;

    private DataProvider(Context pContext) {
        mContext = pContext;

        mRemoteProvider = new RemoteDataProviderImpl();
```

```
    }

    public static void onCreate(Context pContext) {
        sDataProvider = new DataProvider(pContext);
    }

    public static DataProvider getInstance() {
        return sDataProvider;
    }


    @Override
    public Observable<List<News>> getNews(int pPage) {
        return mRemoteProvider.getNews(pPage);
    }
}
```

Код 18. Провайдер данных

В данном случае провайдер принимает роль посредника (фасада) для интернет провайдера, которого мы рассмотрим далее:

```
public class RemoteDataProviderImpl implements RemoteProvider {

    private final IntellectService mService;


    public RemoteDataProviderImpl() {
        OkHttpClient client = new OkHttpClient
                .Builder()
                .addInterceptor(new
HttpLoggingInterceptor().setLevel(HttpLoggingInterceptor.Level.BASIC))
                .build();

        mService = new Retrofit.Builder()
                .baseUrl(BuildConfig.SERVER)
                .client(client)
                .addConverterFactory(GsonConverterFactory.create())
                .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .build()
                .create(IntellectService.class);
    }

    @Override
    public Observable<List<News>> getNews(int pPage) {
        return mService.getNews(pPage);
    }
    }
}
```

Код 19. Интернет провайдер

В интернет провайдере создается сервис для доступа к данным в интернете с помощью библиотеки Retrofit с применением Gson и RxJava, также производиться логирование операций, и применение нового Http клиента OkHttp который поддерживает безопасную передачу данных в сети.

А теперь создадим графический интерфейс пользователя для приложения. Макетный редактор (Layout Editor) позволяет создать графический интерфейс пользователя путем перетаскивания в окно приложения компонентов GUI, таких как Button, TextView, ImageView и др. По умолчанию описание макета для шаблона Empty App хранится в XML-файле с именем activity_main. xml в папке res/layout. Мы воспользуемся макетным редактором и окном Component Tree для построения приложения. Разметка XML в файле activity_main.xml будет отредактирована только для того, чтобы изменить способ размещения компонентов TextView и ImageView, используемых в приложении.

Так как экраны устройств Android обладают разными размерами, разрешением и плотностью пикселов (DPI, Dot Per Inch), разработчик обычно предоставляет изображения с разными разрешениями, а операционная система выбирает графику на основании плотности пикселов устройства. По этой причине папка res вашего проекта содержит несколько вложенных папок, имена которых начинаются с префикса drawable. Например, изображения для устройств с плотностью пикселов, близкой к плотности экрана телефона Google Nexus 6 (560 dpi) для нашего AVD, будут храниться в папке drawable-xxxhdpi.

Задание размеров в пикселах, независимых от плотности (dp или dip), позволяет платформе Android автоматически масштабировать графический интерфейс пользователя в зависимости от плотности пикселов экрана физического устройства. Размер пиксела, независимого от плотности, эквивалентен размеру физического пиксела на экране с разрешением 160 dpi (точек на дюйм). На экране с разрешением 240 dpi размер пиксела,

независимого от плотности, будет масштабироваться с коэффициентом 240/160 (то есть 1,5). Таким образом, компонент, размер которого составляет 100 пикселов, независимых от плотности, будет масштабирован до размера в 150 физических пикселов на таком экране. На экране с разрешением 120 точек на дюйм каждый независимый от плотности пиксел масштабируется с коэффициентом 120/160 (то есть 0,75). Значит, 100 независимых от плотностей пикселов превратятся на таком экране в 75 физических пикселов. Пикселы, независимые от масштабирования, масштабируются так же, как и пикселы, независимые от плотности, но их масштаб зависит также и от предпочитаемого размера шрифта, выбираемого пользователем (в настройках устройства).

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    android:id="@+id/drawer_layout"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <include layout="@layout/view_toolbar"/>

        <include layout="@layout/view_container"/>

    </LinearLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:headerLayout="@layout/view_drawer_header"
        app:menu="@menu/navigation_menu"/>

</android.support.v4.widget.DrawerLayout>
```

Код 20. Верстка MainActivity

В коде вёрстки используются вложенный элементы, всегда должен быть корневой элемент. В данном коде тулбар и контейнер с контентом вложены в линейный контейнер, ниже которого находится меню, но на самом деле он находится слева, это происходит, потому что все это внутри специального контейнера, который специальным образом визуально перемещает его в себе.

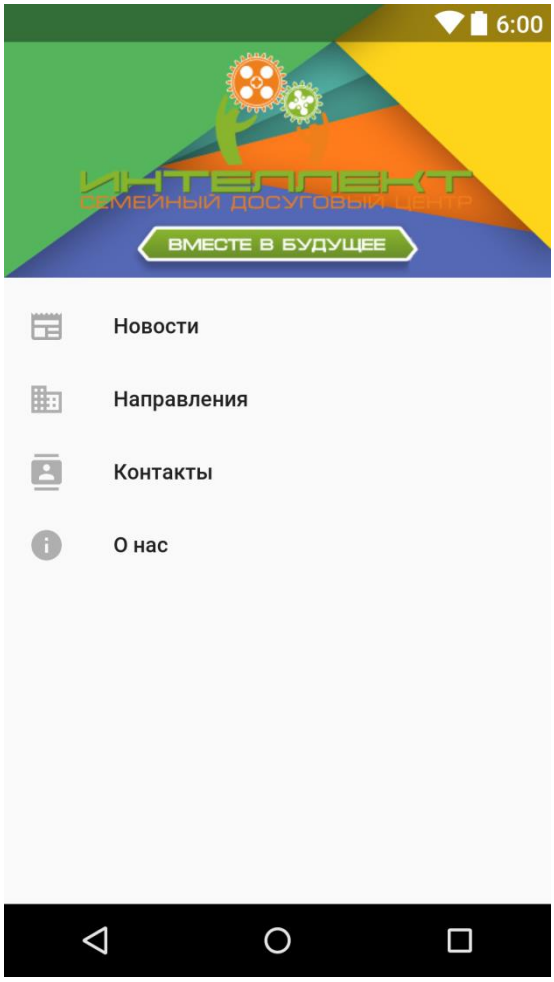

Рис 17. Верстка MainActivity

Следующий раздел для верстки это новости, который тоже свёрстан, но в себе содержит не стандартный элемент, который создан на базе стандартного элемента для обновления контента и теперь для дозагрузки контента для списков:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:app="http://schemas.android.com/apk/res-auto"
              android:orientation="vertical"
              android:layout_width="match_parent"
```

```
            android:layout_height="match_parent">

    <ru.intellect96.intellect.extension.LoadMoreSwipeRefreshLayout
        android:id="@+id/loadMore"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:loading_layout="@layout/loading_view">

        <ListView
            android:id="@+id/list"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="none"
            android:divider="@null"/>

    </ru.intellect96.intellect.extension.LoadMoreSwipeRefreshLayout>

</LinearLayout>
```

Код 21. Раздел со списком новостей

Наиболее объёмный по вёрстке стал элемент списка новостей, который содержит много контейнеров текстовыми отображениями, изображениями, большим количеством вложенных контейнеров, наиболее интересным является CardView, который придается свой неповторимый и пикантный вид разделу новостей, наполняя красочным сочетанием проверенного временем стиля и непревзойдённым новаторством в современном дизайне:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <android.support.v7.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/middle_margin"
        app:cardCornerRadius="4dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <FrameLayout
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <ImageView
                android:id="@+id/image"
                android:layout_width="match_parent"
                android:layout_height="220dp"
                android:scaleType="fitXY"
                android:src="@drawable/header"/>

            <TextView
                android:id="@+id/header"
                android:layout_width="match_parent"
                android:layout_height="56dp"
                android:layout_gravity="bottom"
                android:background="@color/black_60"
                android:ellipsize="end"
                android:gravity="center_vertical"
                android:padding="@dimen/middle_margin"
                android:textColor="@android:color/white"
                android:maxLines="2"
                android:textSize="14sp"
                android:textStyle="bold"
                tools:text="Международный чемпионат по робототехнике
среди любителей и профессионалов"/>

        </FrameLayout>

        <TextView
            android:id="@+id/date"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="@dimen/middle_margin"
            android:layout_marginTop="@dimen/middle_margin"
            android:textColor="@android:color/black"
            tools:text="16 Мая 2016"/>

        <TextView
            android:id="@+id/description"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="@dimen/middle_margin"
            android:layout_marginBottom="@dimen/middle_margin"
            android:textColor="@android:color/black"
            android:textSize="12sp"
            android:maxLines="4"
            android:ellipsize="end"
            tools:text="Вот и пришло время для очередного, пятого,
Международного чемпионата по робототехнике среди любителей и профессионалов,
который пройдёт 21 мая в стенах ИФТиЭ УрГПУ."/>
```

```
        </LinearLayout>

    </android.support.v7.widget.CardView>

</LinearLayout>
```

Код 22. Верстка элемента списка новостей

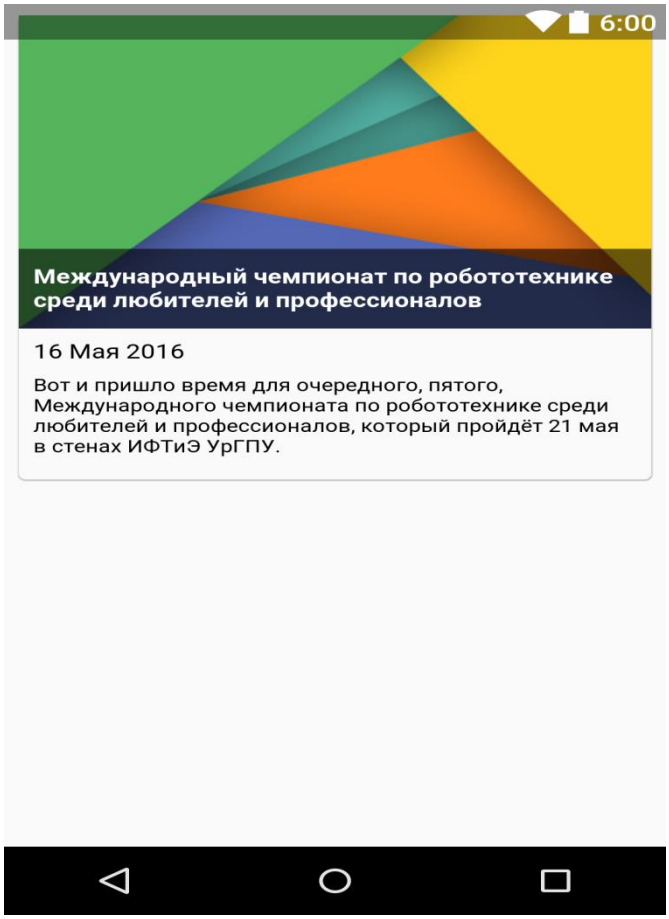

Рис 18. Верстка элемента списка новостей

Мы создали простой графический интерфейс в макетном редакторе и настроили свойства компонентов в окне свойств. В XML-файле разметки используемый по умолчанию компонент RelativeLayout был заменен компонентом LinearLayout, который затем был настроен для вертикального расположения представлений. Приложение выводит текст в компоненте TextView, а изображения — в компоненте ImageView. Мы изменили компонент TextView графического интерфейса по умолчанию, чтобы текст выравнивался по центру, увеличенным шрифтом и в одном из цветов стандартной темы. Компоненты ImageView перетаскивались мышью из

палитры компонентов макетного редактора. Как и положено, все строки, и числовые значения были определены в файлах ресурсов в папке res проекта.

В классе MainActivity задействованы многие средства объектно-ориентированного программирования на языке Java, включая классы, объекты, интерфейсы, анонимные внутренние классы и наследование. Также была представлена концепция заполнения графического интерфейса, то есть преобразования содержимого файла XML в его экранное представление. Мы познакомились с классом Android Activity и жизненным циклом активности. В частности, мы переопределили метод onCreate для инициализации приложения при запуске. В методе onCreate метод findViewById класса Activity использовался для получения ссылок на визуальные компоненты, с которыми приложение взаимодействует на программном уровне. Наконец, мы отредактировали файл AndroidManifest.xml, чтобы указать, что MainActivity поддерживает только портретную ориентацию, а класс MainActivity всегда должен отображать виртуальную клавиатуру. Заодно были представлены другие элементы, включенные Android Studio в манифест при создании проекта. Мы показали, как при помощи объекта Configuration определить, выполняется ли приложение на планшете в альбомной ориентации. Также в этой главе было показано, как управлять большим количеством графических ресурсов с использованием вложенных папок в папке assets приложения и как обращаться к этим ресурсам через AssetManager. Также были рассмотрены другие папки из папки res приложения — menu для хранения файлов ресурсов меню, xml для хранения файлов с разметкой XML. Также мы узнали, как использовать квалификаторы при создании папки для хранения макета, который должен использоваться только на больших устройствах в альбомной ориентации. Мы также показали, как использовать ресурс списка цветов состояний, для того чтобы гарантировать удобочитаемость текста на кнопках как для доступного, так и для заблокированного состояния.

ЗАКЛЮЧЕНИЕ

Целью настоящей дипломной работы заключается в создании мобильного приложения для развивающего образовательного центра, для более эффективного привлечения новых клиентов, повышения информированности существующих клиентов о работе предприятия, а также закрепление практических навыков программирования для ОС Android в среде разработки Android Studio.

Для достижения поставленной цели были сформулированы следующие задачи:

1. Произвести расчет экономических показателей эффективности создания мобильного приложения
2. Произвести поиск и анализ способов повышения потока клиентов с помощью мобильного приложения
3. Произвести анализ требований к мобильному приложению
4. Спроектировать пользовательский интерфейс
5. Создать мобильное приложение

В первой главе дипломной работы рассматривался расчет экономических показателей эффективности создания мобильного приложения, и поиск и анализ способов повышения потока клиентов с помощью мобильного приложения.

При решении задачи были изучены теоретические и практически методы расчета экономической эффективности программного проекта, в частности относящегося к сфере мобильной разработки.

С помощью расчета экономической эффективности были найдены следующие показатели:

- Использования трудовых ресурсов
- Рентабельности
- Потребления электроэнергии

- Ежемесячных расходов
- Амортизации
- Ежемесячных материальных затрат
- Себестоимости
- Цены приложения

По большинству показателей были сгенерированы графические представления для лучшего понимания распределения расходов на создание проекта приложения.

Были найдены и проанализированы и представлены способы повышения потока клиентов с помощью мобильного приложения.

Во второй главе был произведен анализ и предъявлены требования к мобильному приложению, проанализированы функциональные требования к интерфейсу, по которым был создан прототип пользовательского интерфейса. С помощью, которого в дальнейшем производилась разработка минимальной версии мобильного приложения для ОС Android с применением новейших технологий в области мобильных технологий.

При решении задачи были изучены теоретически методы предъявления и анализа требований к программному обеспечению, пользовательскому интерфейсу. Получен практический навык работы с графическим редактором Adobe Photoshop, в котором был разработан пользовательский интерфейс соответствующий требованиям стандартам. Было разработано приложение, которое соответствует предъявленным требованиями дипломной работой, и функционирует согласно описанным требованиям. Также были закреплены и получены новые навыки работы в среде разработки Android Studio.

Таким образом, задачи были решены в полном объёме, цель достигнута.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Алексеева А.И. Комплексный экономический анализ хозяйственной деятельности: учеб. пособие. М.: Финансы и статистика. 2009. 529 с.
2. Акулич, В.В. Анализ эффективности использования оборотных средств. Планово-экономический отдел - №3. - 2004 - С.75-77.
3. Амблер, Т. Практический маркетинг. СПб.: Питер, 2000. 213 с.
4. Бочаров, В.В. Финансовый анализ / В.В. Бочаров. СПб.: Питер, 2007. 240 с.
5. Экономический анализ [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=76557534 (дата обращения: 19.06.2016).
6. Баканов М.И. Теория экономического анализа. М.: Финансы и статистика, 2011. 416 с.
7. Брюс Эккель. Философия Java. СПб.: Питер, 2014. 640 с.
8. Java [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=82062183 (дата обращения: 02.11.2016).
9. Вера Иванова, Андрей Перерва. Путь аналитика. Практическое руководство IT-специалиста. Спб.: Питер, 2016. 304 с.
10. Головачев, А.С. Методологические основы конкурентоспособности предприятий и товара. Экономика и упр. 2005. № 1. С.4-8.
11. Горфинкель, В.Я. Экономика предприятия. Учебник для вузов по экономическим специальностям. М.: Банки и биржи, ЮНИТИ, 2009. 742 с.
12. Косолапова, М.В. Комплексный экономический анализ хозяйственной деятельности: учебник для высших учебных заведений, обучающихся по направлению подготовки "Экономика" и специальности "Бухгалтерский учет, анализ и аудит". Москва: Дашков и К°, 2011. 246 с.
13. Крейнина, М.Н. Финансовый менеджмент. М.: Дело и сервис, 2008. 400 с.

14. Карл И. Вигерс, Джой Битти. Разработка требований к программному обеспечению. БХВ-Петербург, Русская Редакция, 2016. 736 с.
15. Лобан, Л.А. Экономика предприятия: учеб. пособие. - М.: Веды, 2008. 280 с.
16. Маркетинг. Менеджмент: экспресс-курс. СПб.: Питер: Мир книг, 2012. 479 с.
17. Нехорошева, Л.Н. Экономика предприятия. М.: Высш. школа, 2006. 323 с.
18. Ноздрева, Р.Б. Маркетинг: как побеждать на рынке. М.: Финансы и статистика, 2006. 345 с.
19. Сергеев, И.В. Экономика предприятия: Учеб. пособие / И.В. Сергеев. М.: “Финансы и статистика”, 2009. 603 с.
20. Пястолов С.М. Экономический анализ деятельности предприятия: учебник. М.: Академический Проект, 2010. 576 с.
21. Прибыль [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=81983896 (дата обращения: 19.06.2016).
22. Райзберг Б. А. Современный экономический словарь. 5-е изд., перераб. и доп. М.: ИНФРА-М, 2009. 367 с.
23. Леонид Бугаев. Мобильный маркетинг. Как зарядить свой бизнес в мобильном мире. Альпина Паблишер. 2012. 214 с.
24. Пол Дейтел, Харви Дейтел, Александер Уолд. Android для разработчиков. 3-е издание. СПб.: Питер, 2016. 512 с.
25. Марио Цехнер. Программирование игр под Android. СПб.: Питер, 2013. 688 с.
26. Android [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=82147331 (дата обращения: 02.11.2016).
27. Adobe Photoshop [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=81488539 (дата обращения: 25.10.2016).
28. Material Design [Электронный ресурс] // Google. URL: https://material.google.com/ (дата обращения: 25.10.2016).

29. Don’t repeat yourself [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=75063200 (дата обращения: 29.10.2016).
30. Фабричный метод (шаблон проектирования) [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=82081042 (дата обращения: 05.11.2016).
31. Замыкание (программирование) [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=81810616 (дата обращения: 11.11.2016).
32. Model-View-Presenter [Электронный ресурс] // Википедия. URL: http://ru.wikipedia.org/?oldid=78041267 (дата обращения: 28.10.2016).