МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа экономики и управления
Кафедра «Информационные технологии в экономике»

РАБОТА ПРОВЕРЕНА
Рецензент, _____
_____/ _____ /
«_____» _____20____ г.

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой, д.т.н., с.н.с.
_____/ Б.М. Суховилов /
«_____»_____20____ г.

Автоматизация внеклассной жизни школьников и их родителей

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.03.2018. 1394. ВКР

Консультант, доцент, к.т.н.
_____/ О.И. Галичин /
«_____» _____20____ г.

Руководитель, доцент, к.т.н.
_____/ Е.М. Сартасов /
«_____»_____20____ г.

.

Автор
студент группы ВШЭиУ-469
/ Д.В. Розенфельд /
«_____» _____20____ г.

Нормоконтролер, доцент, к.т.н.
/ Е.А. Конова /
«_____»_____20____ г.

Челябинск 2018

АННОТАЦИЯ

Розенфельд, Д.В. «Автоматизация внеклассной жизни школьников». – Челябинск: ЮУрГУ, ВШЭУ–469, 62 с., 25 ил., 5 табл., библиогр. список – 4 наим.

Выпускная квалификационная работа выполнена с целью разработки программного обеспечения, предоставляющего возможности для организации различных детских мероприятий в рамках класса или школы, а также для упрощения жизни родителей. В работе обоснована актуальность выбранной темы, сформулирована цель и задачи.

В результате, разработана система, которая успешно выполняет поставленные задачи, а также прошла все стадии тестирования и была успешно сдана в эксплуатацию, в частности, опубликована в магазины приложений «AppStore» и «GooglePlay».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью является разработка программного обеспечения, предоставляющего возможность для организации различных детских мероприятий в рамках класса или школы, а также для упрощения жизни родителей.

Задачи:

- Обследование предметной области;
- Изучение мобильной разработки, в частности фрэймворка Xamarin.Forms [4];
- Разработка мобильного клиента, для сервиса «Классная.Москва»;
- Реализовать для отдела «Quality Assurance» инструменты, позволяющие покрыть приложение интеграционными тестами;
- Доработка серверной части, для работы с мобильным приложением.

Перечень задач, которые должен выполнять мобильный клиент, представлен в следующем разделе.

1 ПОСТАНОВКА ЗАДАЧИ

Разработать полноценную клиент-серверную систему, которая позволит пользователям участвовать в классной жизни своих детей без отрыва от работы или повседневных дел. Эта система, должна совмещать в себе все возможные активности, которые обычно проводятся на родительских собраниях. При этом, предоставлять их в удобной и доступной форме, чтобы родители не испытывали необходимости, в дополнительных сервисах или личных встречах.

Такие базовые функции, как проведение голосований и сборов денег, безусловно, должны поддерживаться нашим продуктом. Данный функционал доступен для родителей, которые объединились в один класс. Также, председатель этого класса имеет возможность раздавать поручения остальным родителям.

Из дополнительных функций, предоставляемых нашей системой, нужно отдельно выделить организацию и проведение всевозможных событий или мероприятий для детей, с привлечением заинтересованных в этом организаций. Для оповещения всех участников класса можно использовать объявления, или же чат.

Приложение подразделяется на две части: клиентская и серверная. Серверная часть отвечает за хранение и обработку данных. Сохраняются данные пользователя – его телефон, e-mail, пароль, список девайсов, аватар.

Пользователю предоставляется возможность регистрации в приложении по номеру телефона. После регистрации пользователь может создать класс, в котором он автоматически становится председателем.

Пользователи подразделяются на роли: Председатель, Казначей, Родитель.

Приложение должно хранить данные о пользователях и созданных им сущностях.

Вывод по разделу

Был проведён анализ предметной области, и выбраны технологии, для реализации поставленной задачи.

2 СРАВНИТЕЛЬНЫЙ АНАЛИЗ

Для анализа похожих решений на рынке, проведён сравнительный анализ нескольких сервисов.

2.1 Сетевой город

Сетевой город – сайт для взаимодействия школы и родителей, позволяет отслеживать успеваемость детей, получать письма от учителей и т.д. На деле, сильно подвисающий и тормозящий сервис, использующий устаревшие технологии. Не имеет своего мобильного приложения. Внешний вид интерфейса представлен на рисунке 1.

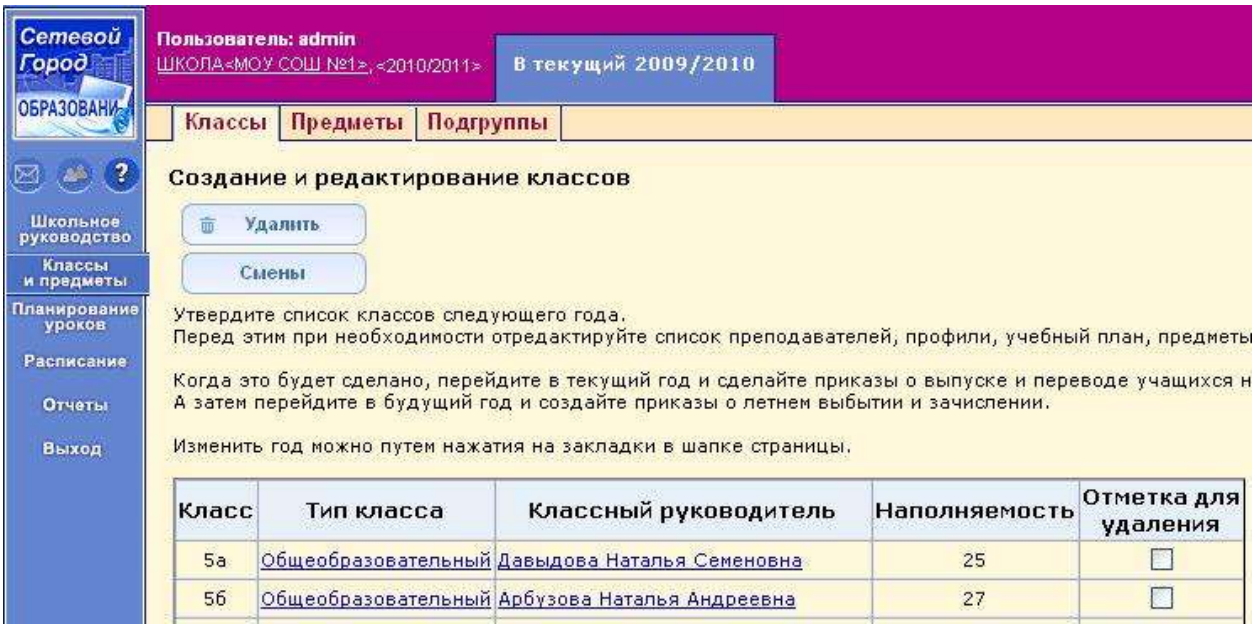

Рисунок 1 – Сетевой город

2.2 Вконтакте

«Вконтакте» российская социальная сеть со штаб-квартирой в Санкт–Петербурге, крупнейшая в Европе. Сайт доступен на многих языках, особенно популярен среди русскоязычных пользователей. «ВКонтакте» позволяет пользователям отправлять друг другу сообщения, создавать группы, публичные страницы и события, обмениваться изображениями, аудио, видео, тегами, а также играть в браузерные игры. Внешний вид интерфейса представлен на рисунке 2.

Рисунок 2 – Вконтакте

2.3 Одноклассники

Можно назвать аналогом «Вконтакте», рассчитанном на немного другую возрастную группу пользователей. Российская разработка, от компании «Mail.Ru Group». Функции аналогичны предыдущему пункту, однако присутствует большее количество рекламы, по отношению к конкурентам. Внешний вид интерфейса представлен на рисунке 3.

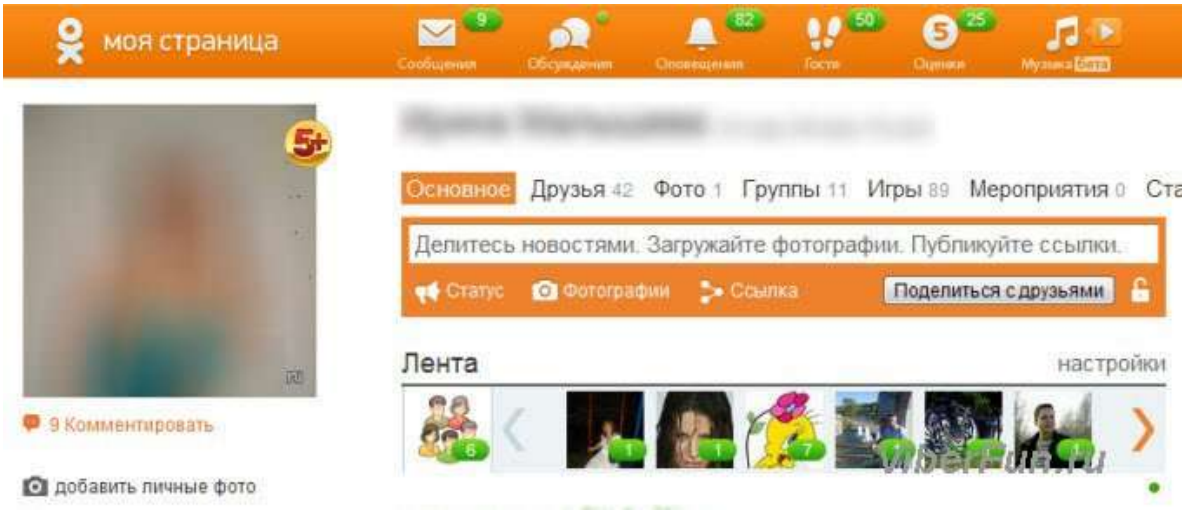

Рисунок 3 – Одноклассники

2.4 WhatsApp

WhatsApp – популярная бесплатная система мгновенного обмена текстовыми сообщениями для мобильных и иных платформ с поддержкой голосовой и видеосвязи. Позволяет пересылать текстовые сообщения, изображения, видео и аудио через Интернет. Клиент работает на платформах Android, iOS (iPhone), Windows Phone, Nokia Symbian, Nokia S40, а также ОС Windows. Внешний вид интерфейса представлен на рисунке 4.

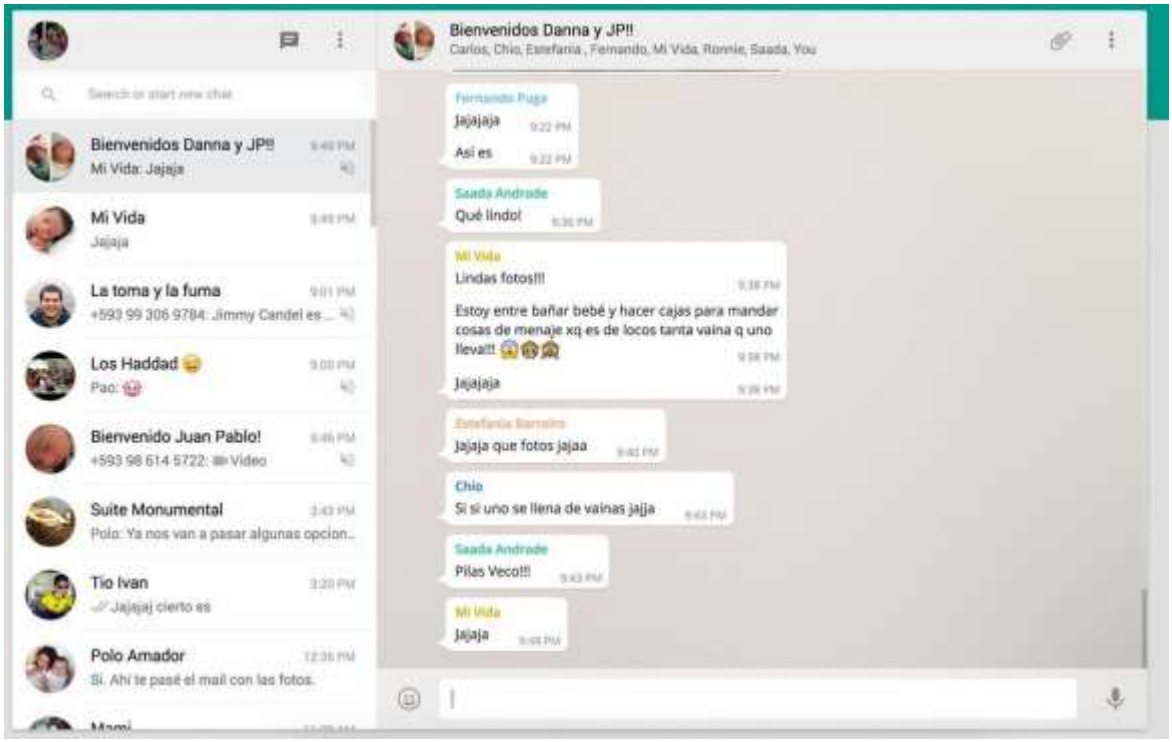

Рисунок 4 – WhatsApp

Результат сравнительного анализа приведён в таблице 1.

Таблица 1 – Сравнительный анализ

	Сетевой город	**Вконтакте**	**Одноклассники**	**WhatsUp**	**Классная.Москва**
Кроссплатформенность	–	+	+	+	+
Комфортное общение	–	+	+	+	+
Возможность создать голосование	–	+	–	–	+
Возможность организации событий	–	+\–	–	–	+
Наличие встроенной календарной ленты событий	–	–	–	–	+
Возможность раздавать поручения	–	–	–	–	+
Возможность проводить сборы денег	–	+\–	–	–	+

Вывод по разделу

Данная таблица показывает, что разработка запланированного программного обеспечения является целесообразной, так как не имеет полноценных аналогов на рынке.

3 СТРУКТУРА ПРОГРАММЫ

3.1 Структура базы данных

В качестве базы данных, в моём проекте выступает нереляционная база данных MongoDB. Она была выбрана на основании того, что приложение рассчитано на одновременное использование большим количеством пользователей. А также, исходя из условий поставленной задачи – у нас много запросов на выборку списков документов одинакового типа, по какому-либо признаку, с чем данное решение справляется быстрее и лучше, чем стандартные SQL базы данных. Помимо этого, MongoDB обладает встроенными средствами масштабирования, что позволяет развернуть сервер приложения в кластере из нескольких машин, тем самым получая прирост в производительности.

MongoDB – это база данных ориентированная на хранение документов. Все документы подразделяются на коллекции. Пример коллекций приведён на рисунке 5.

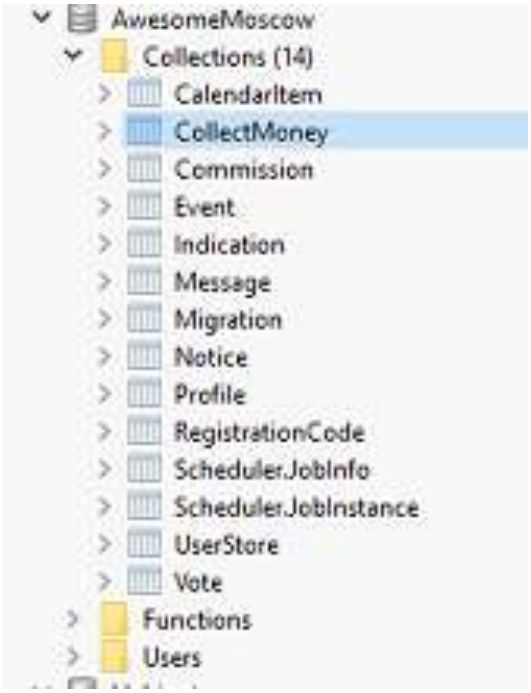

Рисунок 5 – Коллекции MongoDB

Пример документа, который хранится в базе данных, продемонстрирован на следующем рисунке, под номером 6.

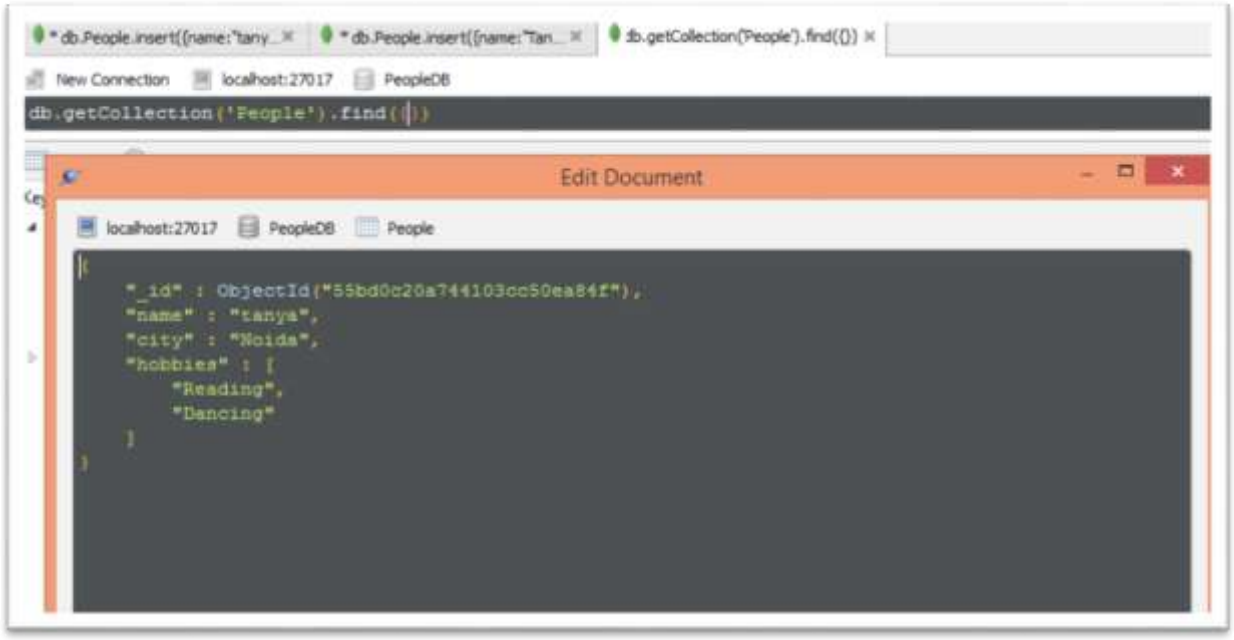

Рисунок 6 – Документ MongoDB

Типы документов и соответствующее разделение на коллекции представлено в таблице 2.

Таблица 2 – Структура БД

Тип	Описание	Поля
Class	Коллекция классов, в которой хранятся документы, имеющие тип «Class».	string ClassName – Имя класса FiasAdrObj City – Город string CreatorId – ID создателя
CollectMoney	Коллекция сборов денег, в которой хранятся документы, имеющие тип «CollectMoney».	string Name – имя сбора денег string Description – описание DateTime EndDate – дата окончания List<Particiapnt> Particiapnts – участники сбора денег int SumPerson – сумма с человека string AuthorId – Id создателя string ClassId – Id класса

Продолжение таблицы 2

Тип	Описание	Поля
Commission	Коллекция поручений, в которой хранятся документы, имеющие тип «Commission».	string Name – Название поручения string Description – Описание поручения DateTime EndDate – дата окончания string ResponsibleUserId – Id ответственного за поручение String CreatorId – Id создателя string ClassId – Id класса
Event	Коллекция событий, в которой хранятся документы, имеющие тип «Event».	string Name – Название события string Description – Описание события string Place – Место проведения события DateTIme Date – Время проведения IEnumerable<EventParticipant> Participants – Участники события string AuthorId – Id создателя string ClassId – Id класса

Продолжение таблицы 2

Тип	Описание	Поля
Indication	Коллекция индикаций для каждого пользователя, хранит документы с типом «Indication».	String ProfileId – Id профиля String ClassId – Id класса List<string> UnreadEvents – Непрочитанные события List<string> UnreadVotings – Непрочитанные голосования List<string> UnreadNotices – Непрочитанные объявления List<string> UnreadParents – Новые родители в классе List<string> UnreadCollectMoneys – Непрочитанные сборы денег List<string> UnreadCommissions – Непрочитанные поручения
Message	Коллекция сообщений, хранит документы с типом «Message».	object ClassId – Id класса string UserName – имя отправителя object UserId – Id отправителя DateTime DateTime время отправки string MessageText – Текст сообщения BlobInfo Attachment – Прикреплённый файл

Продолжение таблицы 2

Тип	Описание	Поля
Notice	Коллекция объявлений, хранит документы с типом «Notice».	String CreatorId – Id создателя DateTime – дата создания String ClassId – Id класса String Body – текст объявления Bool IsImportant – важность объявления IEnumerable<BlobInfo> Attachments – прикреплённые файлы
Profile	Коллекция профилей пользователей, хранит документы с типом «Profile».	String FullName – имя пользователя String Phone – телефон пользователя String Email – электронная почта List<UserRole> Roles – роли пользователя во всех классах Bool ShowPhone – публичность телефона List<DeviceDescription> Devices – Список устройств пользователя BlobInfo Photo – Фото пользователя

Продолжение таблицы 2

Vote	Коллекция голосований, хранит документы с типом «Vote».	String VoteName – Название голосования IEnumerable<Variant> Variants – варианты голосования String ClassId – Id класса String AuthorId – Id создателя List<string> Users – список проголосовавших DateTime StartDate – дата начала голосования DateTime EndDate – дата окончания Bool IsClosed – завершено ли голосование

3.2 Серверная часть

Взаимодействие между сервером и клиентами происходит посредством REST и SignalR.

REST (сокр. от англ. Representational State Transfer «передача состояния представления») – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы. В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине.

В сети Интернет вызов удалённой процедуры может представлять собой обычный HTTP-запрос (обычно «GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса.

Для веб-служб, построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «RESTful».

В отличие от веб-сервисов (веб-служб) на основе SOAP, не существует «официального» стандарта для RESTful веб-API. Дело в том, что REST является архитектурным стилем, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют стандарты, такие как HTTP, URL, JSON и XML.

SignalR – это набор серверных и клиентских библиотек, облегчающих двухстороннее взаимодействие реального времени между сервером и клиентом. Не только клиент может инициировать контакт с сервером, как в случае веб-разработки, но и сервер может связываться с клиентом. При этом используются отнюдь не просто HTTP-ответы. На самом деле, это вызовы методов с

сервера на клиенте по аналогии с технологией оповещения (push technology). Клиент может даже связываться даже с другими клиентами через серверный компонент SignalR. Все это возможно потому, что SignalR создает постоянное соединение между сервером и клиентом.

Работу SignalR можно увидеть на примере чата и индикации о новых сущностях. Как только, один из пользователей написал сообщение, или создал сущность, другие участники класса получают оповещение, чтобы увеличить счётчик непрочитанных сообщений/документов.

Регистрация и авторизация пользователей

Происходит с помощью взаимодействия клиентов с сервисом UserManager. Регистрация осуществляется по номеру телефона, так же у пользователя просят ввести пароль, минимум 6ть символов, а также псевдоним. Для подтверждения номера телефона пользователь, получает по смс код, который действителен в течении двух минут. После успешной авторизации, на web клиенте, мы запоминаем сессию пользователя, чтобы в последующем не просить его каждый раз вводить свои данные. На мобильном клиенте данные сохраняются в локальное хранилище приложения. Стираются после выхода из профиля или после удаления приложения.

Создание класса

При создании класса, пользователь должен указать город и название класса. Город можно выбрать из базы данных ФИАС. Сохраняется город ввиде объекта, в котором хранится фиас-код, префикс населённого пункта и название. При сохранении документа, на сервере к документу дописывается коллекция участников. Пользователю, который создал класс, автоматически проставляется роль – «Председатель».

3.3 Мобильный клиент

Регистрация и аутентификация пользователей в системе

Для регистрации пользователя в системе используется его номер телефона. А также пароль, который должен быть не менее 6ти символов. Пользователь может указать своё имя или псевдоним, под которым его будут видеть другие клиенты. После ввода этих данных, на указанный пользователем номер телефона отправляется СМС с кодом, для подтверждения номера.
Внешний вид формы регистрации представлен на рисунке 7.

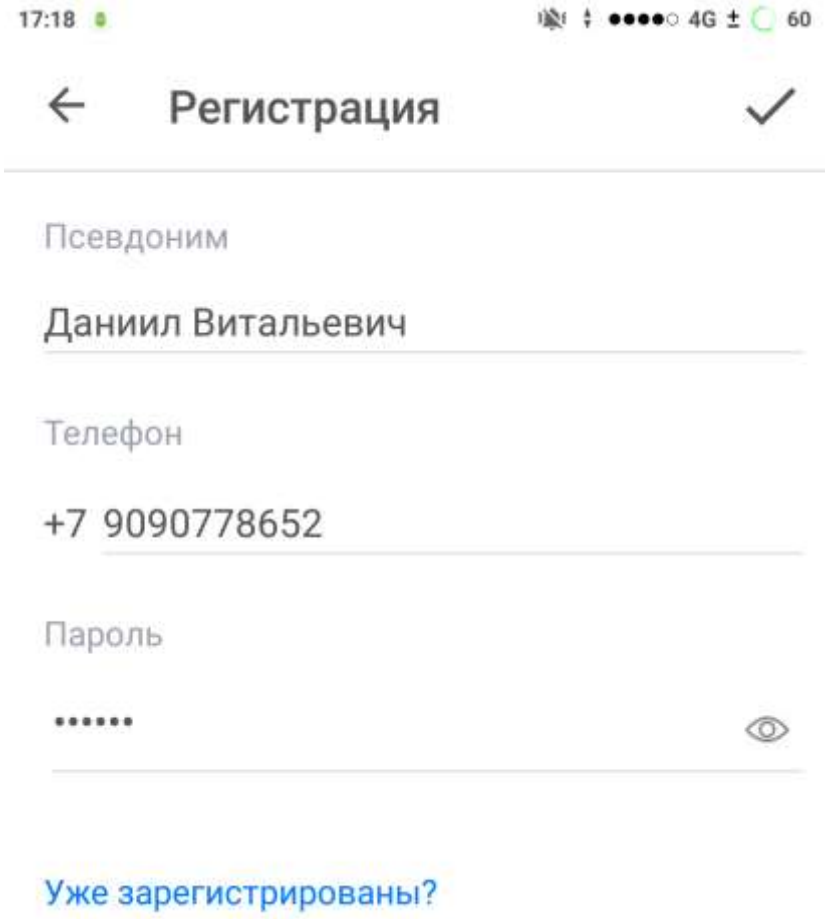

Рисунок 7 – Страница регистрации.

Форма имеет два состояния:

- Пароль, введённый пользователем, скрыт;
- Пароль виден.

Переключение между состояниями происходит с помощью кнопки, встроенной в текстовое поле.

Помимо полей ввода на форме находится кнопка подтверждения – галочка в тулбаре, правый верхний угол. По нажатию на которую происходит отправка данных на сервер, их валидация и, при корректности оных, вызывается отправка смс сообщения на номер, указанный пользователем. Кнопка подтверждения становится доступной только после корректного заполнения пользователем всех полей.

Последняя кнопка на данной форме – «Уже зарегистрированы?» возвращает пользователя на страницу входа в приложение.

Если пользователь ввёл корректный номер телефона, к которому у него есть доступ, то он получает смс с кодом и вводит его в приложении. После этого ещё раз вызывает подтверждение, тем самым отправляя код на сервер. Он проверяется на соответствие высланному коду, и при успешном прохождении проверки, на основе данных, ранее отправленных пользователем, создаётся документ в коллекции Profile, и аналогичный документ User (в данном документе хранится пароль в зашифрованном виде, идентификатор пользователя и его номер телефона). Данный документ основан на дефолтных средствах идентификации пользователей, предоставляемых Microsoft.

Дополнительно, для того, чтобы при каждом открытии приложения не заставлять пользователя снова и снова вводить номер и пароль, эти данные кэшируются в локальном хранилище приложения и запрашиваются оттуда при запуске, тем самым улучшая «User Expirience».

Предоставить пользователю возможность ввести дополнительную информацию о себе для идентификации его другими клиентами

Пользователю предоставляется возможность выбрать изображение, которое будет отображаться для других родителей, а также ввести e-mail.

Внешний вид страницы профиля представлен на рисунке 8.

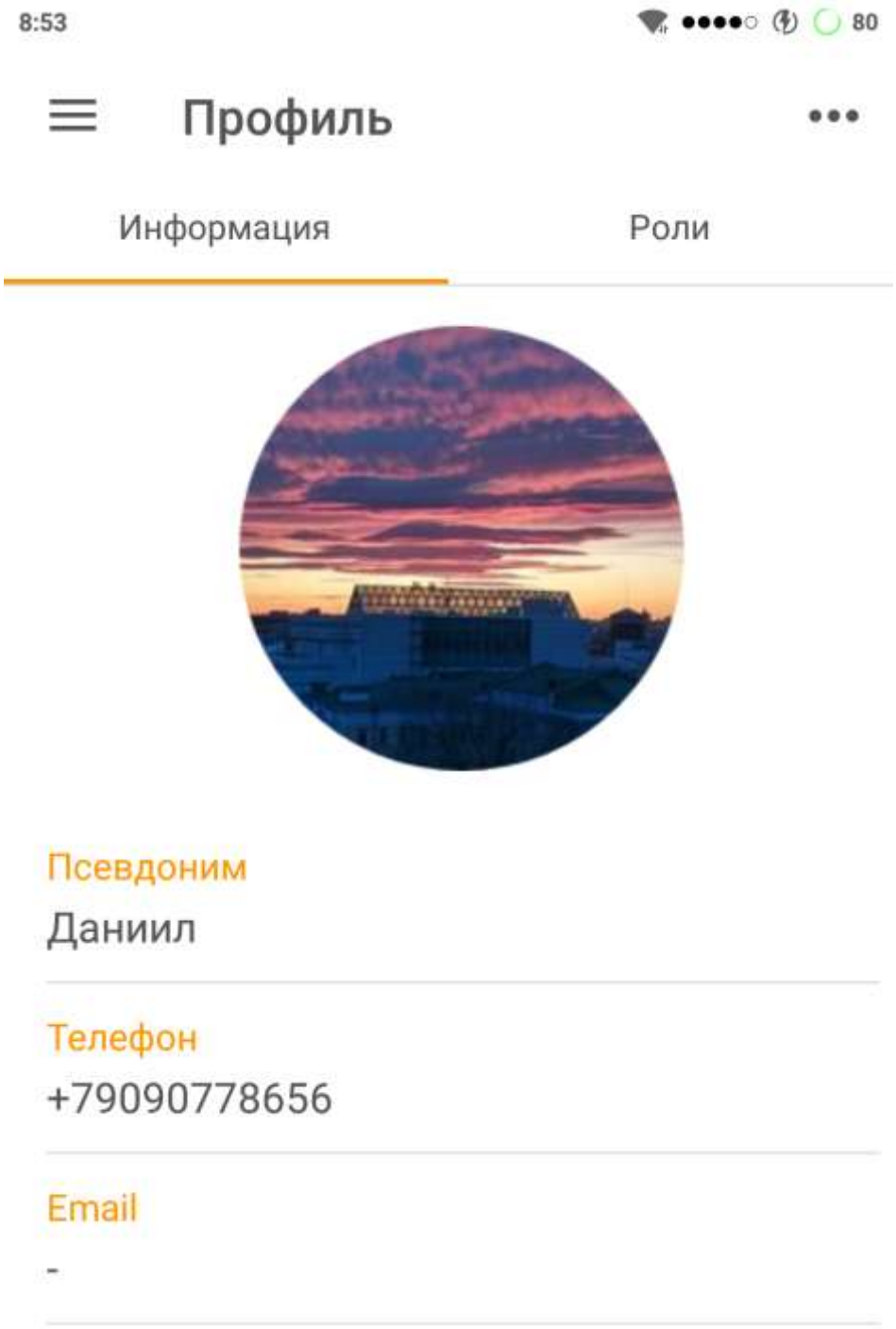

Рисунок 8 – Профиль пользователя

На примере этой странице стоит рассмотреть верстку страницы в Xamarin.Forms. Для этого существует два варианта. Первый – описать верстку страницы в xaml файле. Данный вариант представлен в листинге 1.

Листинг 1 – Код страницы профиля

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:controls="clr-
namespace:HabinetMobileClient.Controls;assembly=HabinetMobileClient"
             xmlns:forms="clr-
namespace:FFImageLoading.Forms;assembly=FFImageLoading.Forms"
             xmlns:transformations="clr-
namespace:FFImageLoading.Transformations;assembly=FFImageLoading.Transformations"
             x:Class="HabinetMobileClient.Views.Common.MyProfilePage"
             Title="Профиль"
             NavigationPage.BackButtonTitle=""
             BackgroundColor="{StaticResource White}">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style x:Key="PlaceholderTextStyle" TargetType="Label"
BasedOn="{StaticResource PlaceholderStyle}"/>
        </ResourceDictionary>
    </ContentPage.Resources>
    <ContentPage.ToolbarItems>
        <controls:SvgToolbarItem AutomationId = "MenuIcon" SvgIcon = "ic_more.svg"
Clicked="MenuItem_OnClicked" />
    </ContentPage.ToolbarItems>
    <ContentPage.Content>
        <StackLayout x:Name="RootStackLayout"
                 VerticalOptions="FillAndExpand"
                 HorizontalOptions="FillAndExpand"
                 Orientation="Vertical">
            <controls:TabControl x:Name="TabControl" FirstTabText="Информация"
SecondTabText="Роли" />
            <StackLayout x:Name="InfoStackLayout"
                   VerticalOptions="FillAndExpand"
                   HorizontalOptions="FillAndExpand"
                   Orientation="Vertical"
                    Padding="17.5,5,5,0">
                <ScrollView>
                    <StackLayout Spacing="3">
                        <Frame HeightRequest="150" WidthRequest="150" HasShadow="False"
HorizontalOptions="Center"  Margin="0,0,0,20">
                            <Frame.GestureRecognizers>
                                <TapGestureRecognizer x:Name="PhotoTapGesture"/>
                            </Frame.GestureRecognizers>
                            <forms:CachedImage x:Name="PhotoImage" AutomationId="Photo"
Aspect="AspectFill" Source="{Binding PhotoUrl}"  Margin="-15">
                                <forms:CachedImage.Transformations>
                                    <transformations:CircleTransformation />
                                </forms:CachedImage.Transformations>
                            </forms:CachedImage>
                        </Frame>
                        <Label Text="Псевдоним" FontSize="{StaticResource Display5}"
TextColor="{StaticResource Primary1}"/>
                        <Label Text="{Binding FullName}" AutomationId="Nickname"
FontSize="{StaticResource Display4}" TextColor="{StaticResource Primary2}" />
                        <BoxView HeightRequest="1" Color="{StaticResource Primary4}" Mar-
gin="0,8,0,8"/>
                        <Label Text="Телефон" FontSize="{StaticResource Display5}"
TextColor="{StaticResource Primary1}"/>
                        <Label Text="{Binding Phone}" AutomationId="PhoneNumber"
FontSize="{StaticResource Display4}" TextColor="{StaticResource Primary2}"/>
                        <BoxView HeightRequest="1" Color="{StaticResource Primary4}" Mar-
gin="0,8,0,8"/>
```

```
                        <Label Text="Email" FontSize="{StaticResource Display5}"
TextColor="{StaticResource Primary1}"/>
                        <Label x:Name="EmailLabel" AutomationId="EmailEntry"
Text="{Binding Email}" FontSize="{StaticResource Display4}" TextColor="{StaticResource
Primary2}"/>
                        <BoxView HeightRequest="1" Color="{StaticResource Primary4}" Mar-
gin="0,8,0,8"/>
                    </StackLayout>
                </ScrollView>
            </StackLayout>
            <StackLayout x:Name="RolesStackLayout"
                   VerticalOptions="FillAndExpand"
                   HorizontalOptions="FillAndExpand"
                   Orientation="Vertical"
                         Padding="17.5,5,5,0">
                <ListView x:Name="RolesListView" ItemsSource="{Binding Roles}"
SeparatorVisibility="None" HasUnevenRows="True" RowHeight="-1"
CachingStrategy="RecycleElement">
                    <ListView.Header>
                        <Label Text="Ваши роли в классах (назначаются председателем)"
Style="{StaticResource PlaceholderTextStyle}"/>
                    </ListView.Header>
                    <ListView.ItemTemplate>
                        <DataTemplate>
                            <controls:CustomViewCell SelectedBackgroundColor="White">
                                <StackLayout Spacing="0">
                                    <StackLayout Padding="0,8,0,8" Spacing="5">
                                        <Label Text="{Binding ClassName}"
AutomationId="ClassName" TextColor="{StaticResource Primary2}" FontSize="{StaticResource
Display4}"/>
                                        <Label Text="{Binding RoleName}"
AutomationId="RoleName" TextColor="{StaticResource Primary1}" FontSize="{StaticResource
Display5}"/>
                                    </StackLayout>
                                    <BoxView HeightRequest="1" Color="{StaticResource
Primary4}" />
                                </StackLayout>
                            </controls:CustomViewCell>
                        </DataTemplate>
                    </ListView.ItemTemplate>
                    <ListView.Footer>
                        <Label/>
                    </ListView.Footer>
                </ListView>
            </StackLayout>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

В контексте данной страницы разберём такой паттерн проектирования как MVVM (Model View ViewModel) [2]. Его схема представлена на рисунке 9.

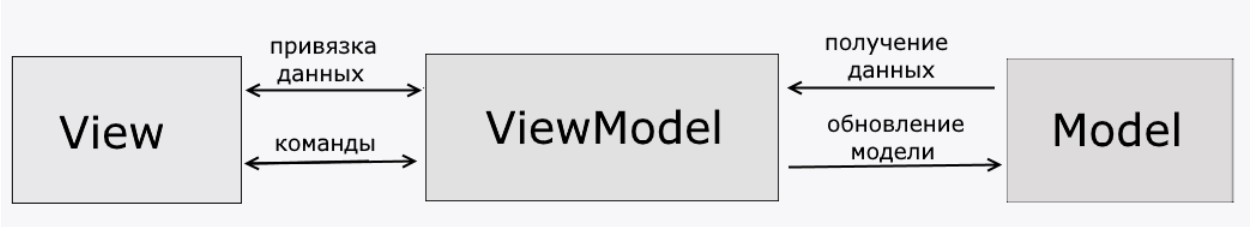

Рисунок 9 – Паттерн MVVM

В листинге 1 можно увидеть ключевое слово «Binding» при описании свойств контролов. Данная конструкция указывает, что значение свойства нужно искать в «BindingContext» или, говоря русским языком – контекст привязки. В рамках паттерна MVVM этим контекстом будет ViewModel.

Для прикрепления какого-либо файла из памяти устройства, нам понадобятся соответствующие разрешения, а также нативные реализации данного функционала, для каждой платформы, так как работа с памятью и файловой системой сильно отличаются в зависимости от операционной системы. Так на Android пользователю предоставляется свободный доступ к памяти устройства, с некоторыми ограничениями на системные файлы и папки. В тоже время, файловая система на iOS практически полностью закрыта от пользователя, а доступ предоставляется лишь к некоторым абстракциям, таким как «Фото», «Музыка» и т.д.

Функционал, описанный выше, реализован с помощью механизма Dependency Injection. В общем виде, это выглядит как интерфейс, объявленный в основном проекте, и его наследники в каждом из платформенных проектов. Схему данного паттерна [3] можно увидеть на рисунке 10.

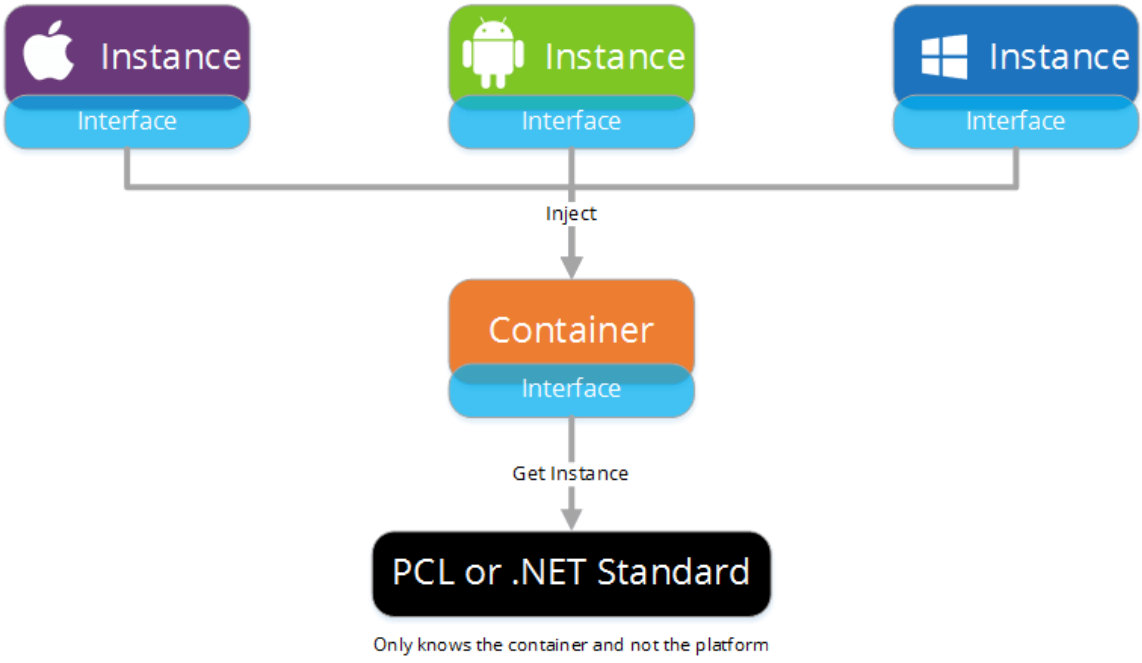

Рисунок 10 – Dependency Injection

Интерфейс регистрируется в IoC контейнере, и при вызове методов, в зависимости от платформы на которой запущено приложение, происходит выполнение разного кода, для каждой ОС. В листинге 2 представлена реализация Dependency Injection на примере отправки пользователю локального push-уведомления. Как правило, данные уведомления инициируются с сервера, и в таком варианте могут полностью обрабатываться операционной системой, без вмешательства разработчика приложения, однако бывают ситуации, продиктованные техническим заданием, когда нужно вручную вывести для пользователя push-уведомление.

Листинг 2 – Отправка локального push-уведомления.

```
namespace HabinetMobileClient.Services
{
    public interface IPushNotificationService
    {
        void SendLocalNotification(string message, string title, string document = null,
           string id = null, string view = null);
    }
}


[assembly: Dependency(typeof(PushNotificationService))]
namespace HabinetMobileClient.iOS.Services
{
    public class PushNotificationService : IPushNotificationService
    {
        public void SendLocalNotification(string message, string title,
              string document = null, string id = null, string view = null)
        {
            var dic = new NSMutableDictionary
            {
                {new NSString("type"), new NSString("local")},
                {new NSString("action"), new NSString("preview")}
            };

            if (!string.IsNullOrEmpty(document))
            {
                dic.Add(new NSString("document"), new NSString(document));
            }

            if (!string.IsNullOrEmpty(id))
            {
                dic.Add(new NSString("id"), new NSString(id));
            }

            if (!string.IsNullOrEmpty(view))
            {
                dic.Add(new NSString(view), new NSString(view));
            }
```

```
            var notification = new UNMutableNotificationContent
            {
                Title = title,
                Body = message,
                Sound = UNNotificationSound.Default,
                UserInfo = dic
            };

            var trigger = UNTimeIntervalNotificationTrigger.CreateTrigger(1, false);
            var requestId = "localPush";

            var request = UNNotificationRequest.FromIdentifier(requestId, notification,
trigger);

            UNUserNotificationCenter.Current.AddNotificationRequest(request, err =>
            {
                if (err != null)
                {
                    Debug.WriteLine(err);
                }
            });
        }
       }
}


[assembly: Dependency(typeof(PushNotificationService))]
namespace HabinetMobileClient.Droid.Services
{
    public class PushNotificationService : IPushNotificationService
    {
        private readonly NotificationManager _notificationManager;

        public PushNotificationService()
        {
            _notificationManager =
                (NotificationManager)Forms.Context.GetSystemService(
                    Context.NotificationService);
        }

        public void SendLocalNotification(string message, string title, string document =
null, string id = null,
            string view = null)
        {
            var pushIntent = new Intent(MainActivity.Context, typeof(MainActivity));

            if (!string.IsNullOrEmpty(document))
            {
                pushIntent.PutExtra("document", document);
            }

            if (!string.IsNullOrEmpty(id))
            {
                pushIntent.PutExtra("id", id);
            }

            if (!string.IsNullOrEmpty(view))
            {
                pushIntent.PutExtra("view", view);
            }
```

```
            pushIntent.PutExtra("type", "local");

            var stackBuilder = TaskStackBuilder.Create(MainActivity.Context);
            stackBuilder.AddParentStack(Class.FromType(typeof(MainActivity)));
            stackBuilder.AddNextIntent(pushIntent);

            var pendingIntent =
                stackBuilder.GetPendingIntent((int)(Math.Random() * 100),
PendingIntentFlags.UpdateCurrent);

            var notificationBuilder = new
NotificationCompat.Builder(MainActivity.Context)
                .SetContentTitle(title)
                .SetContentText(message)
                .SetStyle(new NotificationCompat.BigTextStyle().BigText(message))
                .SetSmallIcon(Resource.Drawable.status_bar_icon)
                .SetColor(Color.ParseColor("#ff0000"))
                .SetPriority((int)NotificationPriority.Max)
                .SetContentIntent(pendingIntent);

            notificationBuilder.SetDefaults(NotificationCompat.DefaultAll);

            string channelId = "default_channel_id";
            string channelDescription = "Default Channel";
            //Check if notification channel exists and if not create one
            if (Build.VERSION.SdkInt >= BuildVersionCodes.O)
            {
                NotificationChannel notificationChannel =
_notificationManager.GetNotificationChannel(channelId);
                if (notificationChannel == null)
                {
                    notificationChannel = new NotificationChannel(channelId,
channelDescription, NotificationImportance.Max);
                    notificationChannel.EnableLights(true);
                    notificationChannel.EnableVibration(true);
                    _notificationManager.CreateNotificationChannel(notificationChannel);
                }
                notificationBuilder.SetChannelId(channelId);
            }

            _notificationManager.Notify(0, notificationBuilder.Build());
        }
      }
}
```

Вернёмся к установке «аватара» в профиле пользователя. После выбора файла, он отправляется на сервер, где записывается в хранилище blob'ов, а на клиент возвращается ссылка на изображение. По этой ссылке картинка подтягивается в Image контролл и отображается для пользователя.

Пользователь должен иметь возможность создать класс и пригласить других людей присоединиться к нему.

Класс является базовой сущностью, по которой пользователи разбиваются на группы. Внутри своего класса пользователи получают доступ к созданию остальных сущностей, таких как события, голосования и т.д.

Итак, для того чтобы создать класс, клиент должен открыть главное меню приложения, нажать на «Мои классы», при этом он попадёт на страницу «Список классов».

Здесь нужно подробнее рассмотреть такой способ навигации по приложению как «Главное меню» или Master-Home Page. В данном случае, мастером является шторка, которую можно вызвать на любой странице приложения, сделав свайп в право. Она предоставляет доступ ко всем основным разделам программы, чтобы пользователь мог в любой момент перейти к интересующим его действиям. В роли home page выступает страница, которая открывается при переходе по одну из пунктов меню. Стоит уточнить, что данный подход к навигации не является единственным. Так, за последнее время, стал очень популярен метод TabPage, когда главные подразделы приложения вынесены, в виде «табов» или закладок, в нижней части экрана. Данный метод, не так давно, применили «Вконтакте», хотя он и был встречен пользователями не слишком позитивно.

Создание класса. Для этого пользователю необходимо ввести его название и выбрать город. По умолчанию установлена Москва, однако есть возможность выбрать любой другой населённый пункт на территории Российской Федерации, список взят из справочника ФИАС. Интерфейс выбора города можно увидеть на рисунке 11.

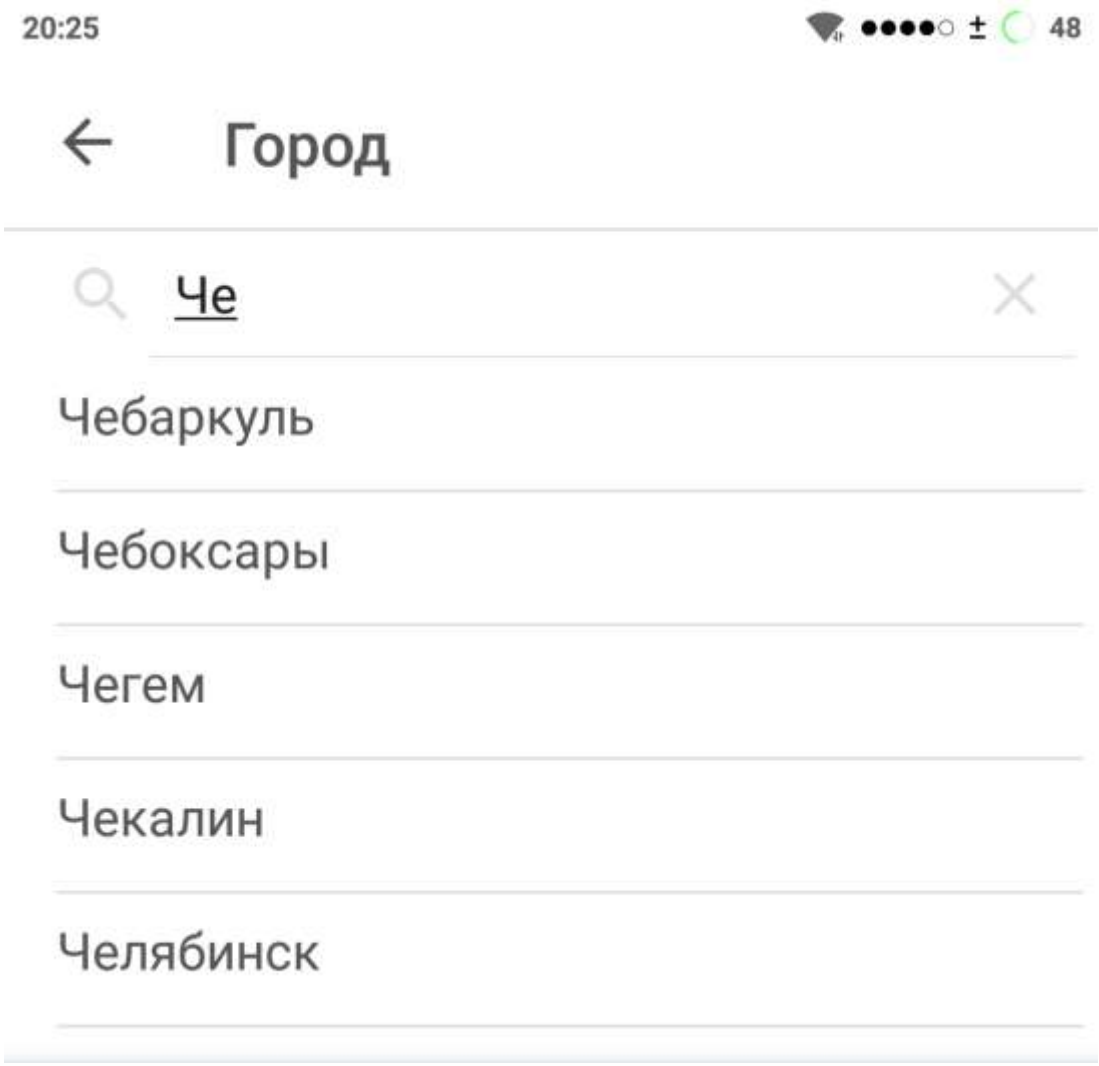

Рисунок 11 – Страница выбора города

Пользователь, создавший класс, автоматически получает роль председателя и может пригласить других родителей присоединиться. Также, ему становятся доступны все функции приложения.

Теперь о функции приглашения в класс – под приглашением, понимается отправка какому-либо человеку ссылки, при переходе по которой, он сможет зарегистрироваться в сервисе и вступить в класс. Отправка может быть сделана любым, доступным пользователю способом – смс-сообщение, e-mail или один из множества мессенджеров, установленных на устройстве. Внешний вид приведён на рисунке 12.

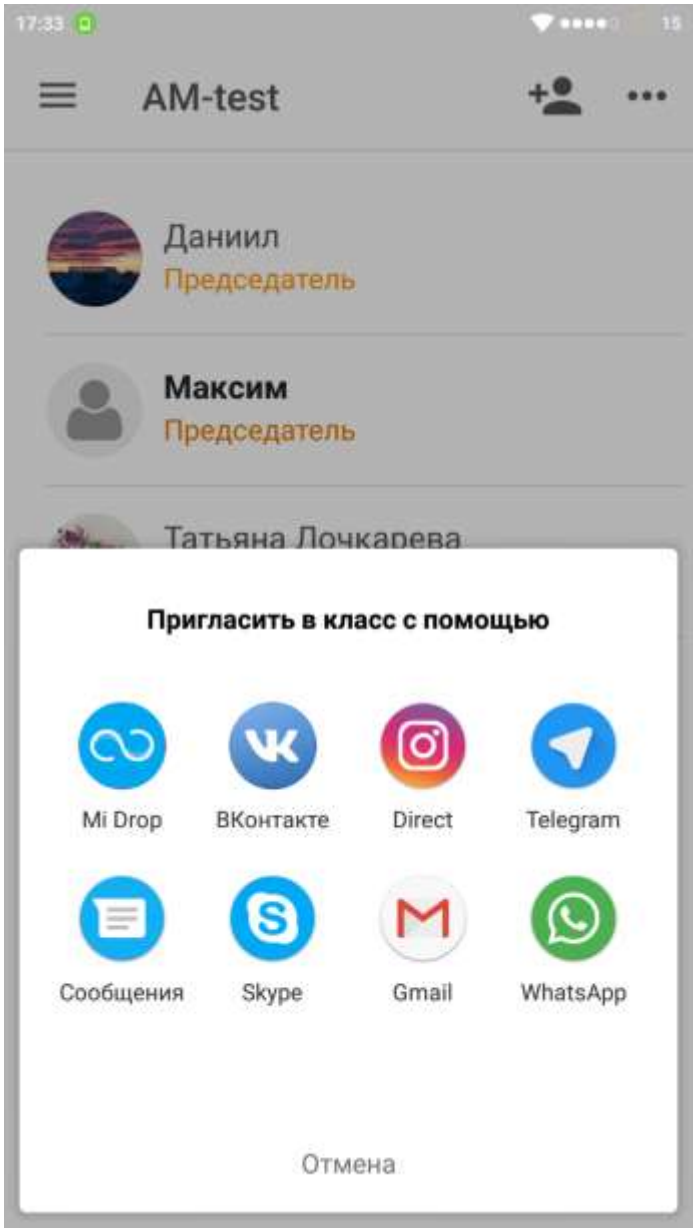

Рисунок 12 – Приглашение в класс

Если человек, получивший данную ссылку, ещё не установил мобильное приложение, то ссылка откроется в браузере. Однако, самое интересное начинается, если человек, которому отправили ссылку на вступление в класс, уже имеет мобильный клиент. В этом случае, система предложит ему открыть ссылку непосредственно в нашем приложении. При положительном ответе запустится мобильный клиент и сразу же откроет страницу вступления в класс. Данная страница представлена на рисунке 13.

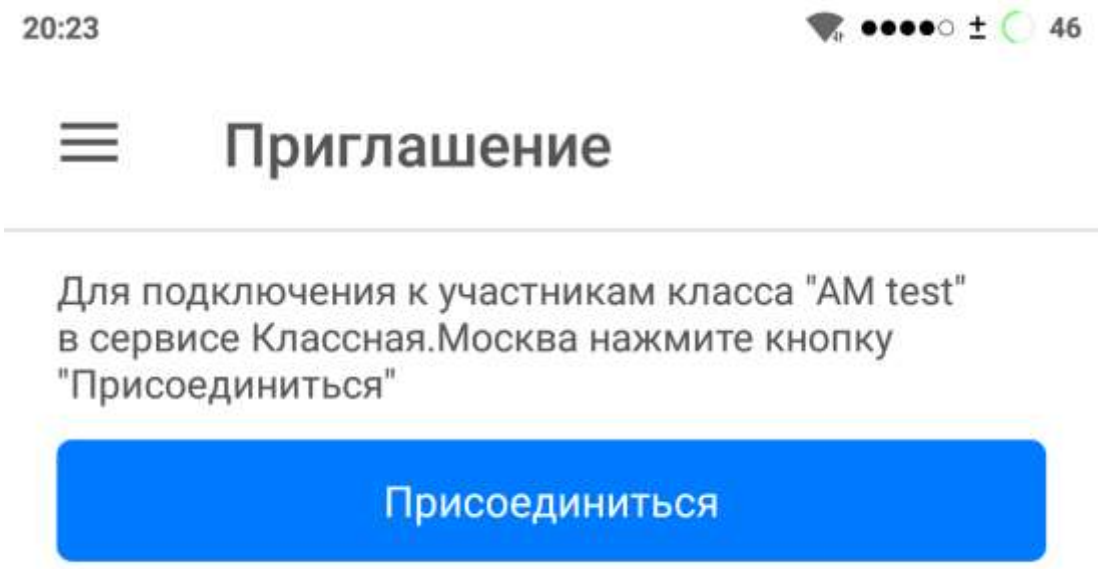

Рисунок 13 – Страница присоединения к классу

Эту функцию принято называть Deep Link или «глубокие» ссылки. Для того, чтобы определённый домен (в нашем случае «классная.москва») ассоциировался с нашим приложением, нужно произвести соответствующие настройки в аккаунте разработчика, в случае с iOS. Для Android, необходимо сгенерировать fingerprint, проще говоря, уникальный ssh-ключ приложения и разместить его на нашем сайте, после чего ожидать проверки от Google, после которой мы уже сможем получать ссылки в мобильном приложении. Далее, вся обработка происходит уже на стороне клиента, ссылка передаётся в приложение, как параметр запуска и в зависимости от её содержания, мы можем открыть для пользователя соответствующую страницу в мобильном клиенте.

3.3.1 «Идеи»

Итак, у пользователя есть класс. Теперь ему доступны все функции приложения. Перейдём к такой сущности как «Идея». В общем виде, это какое-либо мероприятие, заранее выбранное нашими модераторами. Любой пользователь может выбрать понравившуюся ему идею и предложить её для проведения в своём классе. Или же, поделиться ей и предложить людям, ещё не зарегистрированным в нашем сервисе. Внешний вид списка идей можно увидеть на рисунке 14.

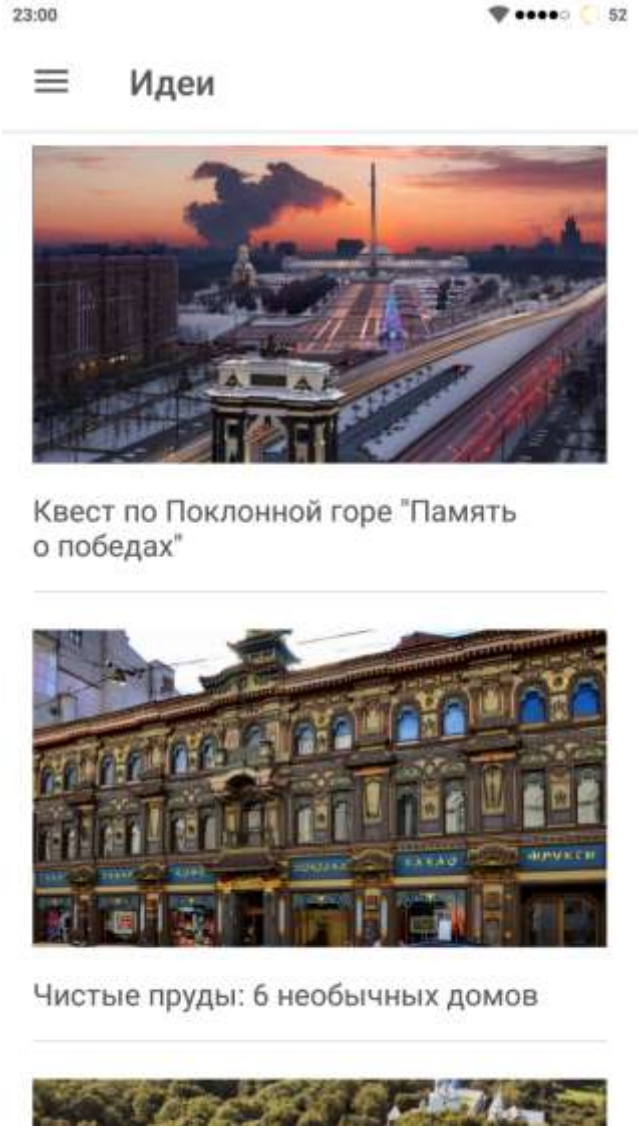

Рисунок 14 – Список идей

Перейдя к одной из идей, пользователь увидит подробную информацию, об этом мероприятии – время проведения, стоимость, описание. Если

данной информации будет недостаточно, он всегда может перейти непосредственно на сайт, где сможет увидеть дополнительное описание, а также приобрести билеты на данное событие. Страницу просмотра идеи можно увидеть на рисунке 15.

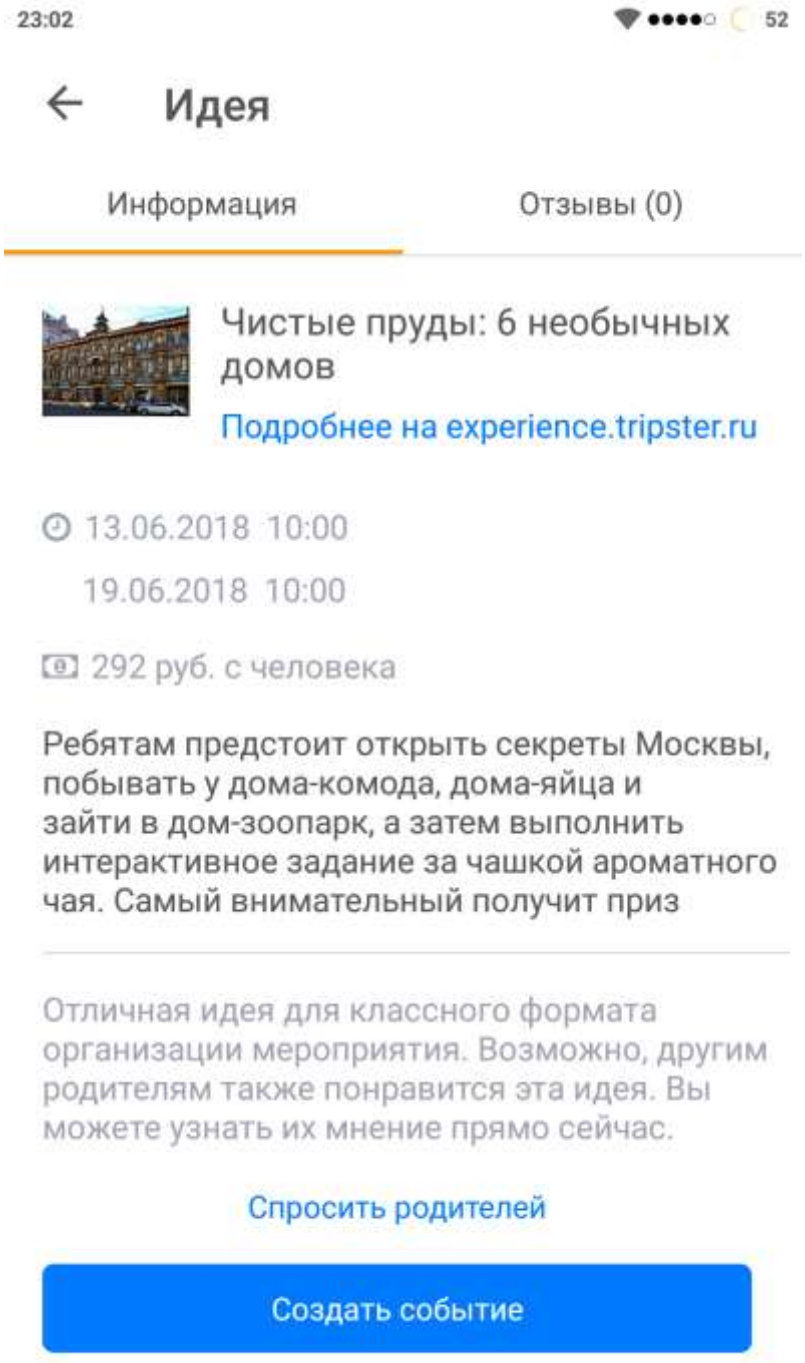

Рисунок 15 – Просмотр идеи

На данной странице, помимо описания мероприятия присутствуют две кнопки: «Спросить родителей» и «Создать событие». Эти кнопки пред-

ставляют ещё две самостоятельные части системы: «Голосование» и «Событие».

3.3.2 «Голосования»

Голосование, в первую очередь, необходимо, чтобы узнать мнение участников класса, по какому-либо вопросу. Для создания голосования нужно перейти в соответствующий пункт меню. Откроется список, состоящий из двух вкладок: «Активные» и «Завершенные». Соответственно, в первой расположены голосования, которые ещё проводятся, а на второй те, которые уже закрыты. В списке, отображается часть названия голосования, имя пользователя, который его создал, а также дата завершения. При нажатии, на одно из них откроется страница просмотра голосования.

Для создания голосования нужно нажать на икону «плюс» в правом вернем углу страницы. После данного действия откроется форма создания голосования. На ней доступны для заполнения поля, представленные в таблице 3.

Таблица 3 – Поля голосования

Наименование поля	**Описание**	**Обязательное**
Вопрос голосования	По сути, это поле является названием голосования и оно будет отображаться в списке.	Да
Описание	Дополнительная информация, которая может понадобится родителям, чтобы ответить на поставленный вопрос.	Нет

Продолжение таблицы 3

Дата завершения голосования	Дата, когда голосование закроется автоматически. Не может быть меньше текущей даты плюс один день. При выборе даты, меньше необходимой, пользователь получит соответствующее всплывающее уведомление.	Да
Варианты ответов (до 10ти).	Собственно, варианты, которые будут доступны пользователям, для ответа на поставленный в голосовании вопрос.	Два обязательны, остальные опционально.
Класс	Класс, которому будет принадлежать данное голосование. Данное поле становится доступно для изменения пользователю, если на момент создания голосования он состоит в двух или более класс. В противном случае, данное поле заполняется автоматически.	Да

Таблица 3 – Создание голосования

Внешний вид представлен на рисунке 16.

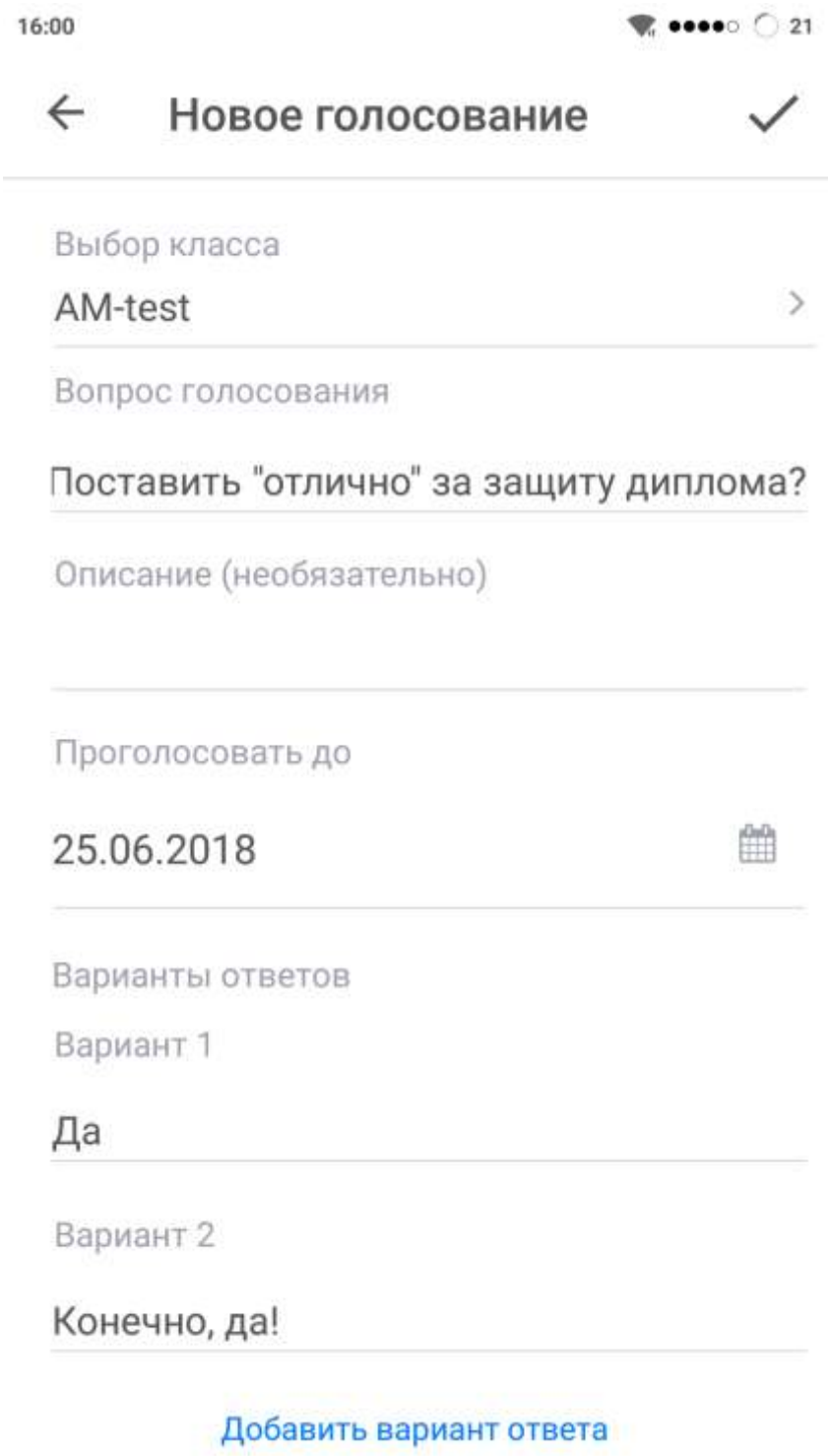

Рисунок 16 – Создание голосования

На данной странице также присутствует кнопка «Добавить вариант ответа». По нажатию на неё, для пользователя появляется дополнительное поле ввода. При достижении десяти вариантов ответа, кнопка исчезает.

После заполнения всех обязательных полей, кнопка подтверждения, она же «галочка» в правом верхнем углу, становится активной, тем самым,

позволяя пользователю сохранить голосование. При сохранении, система отправляет сообщение в чат класса, которое оповещает пользователей о наличии нового вопроса. Сообщение в чат, также инициирует отправку push-уведомления.

Итак, пользователь завершил создание, и попал на страницу просмотра голосования, которую можно увидеть на рисунке 17.

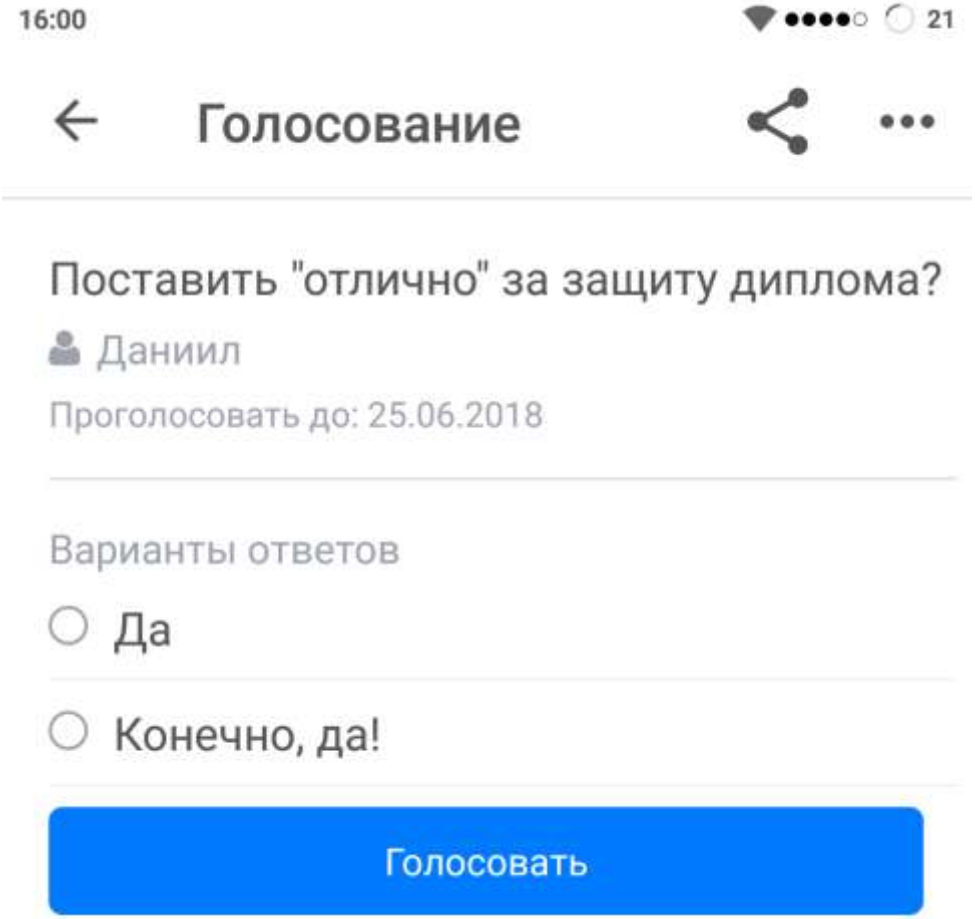

Рисунок 17 – Просмотр голосования

На ней отображаются, все заполненные создателем поля, а также, доступные для выбора варианты ответов. И, конечно же, кнопка «Голосовать». Также, в тулбаре появляются две иконки: «Поделиться» и «Контекстное меню». Стоит отметить, что последняя доступна только создателю голосования, либо председателю класса, от рядового участника данная кнопка будет скрыта. Сразу, после создания, в контекстном меню доступны два пункта: «Изменить» и «Удалить». Изменение голосования возможно, только пока в нём никто не проголосовал. После появления хотя бы одного голоса, данный пункт

меню заменяется на «Продлить», в котором уже можно изменить лишь дату завершения голосования. Это было сделано, чтобы сохранить целостность информации, дабы избежать ситуаций, когда пользователи голосовали за один вопрос или вариант ответа, а создатель, в последний момент, решил поменять исходные данные.

Вернёмся к кнопке «Голосовать». Если нажать на неё, не выбрав один из предложенных вариантов, появится информационное сообщение, уведомляющее пользователя, о необходимости выбора. В случае, если пользователь выбрал один из вариантов, его голос будет учтён, и страница просмотра изменит свой внешний вид. Пользователю станут доступны результаты голосования, на текущий момент – общее кол-во голосов и их процентное соотношение в наглядном виде, как показано на рисунке 18.

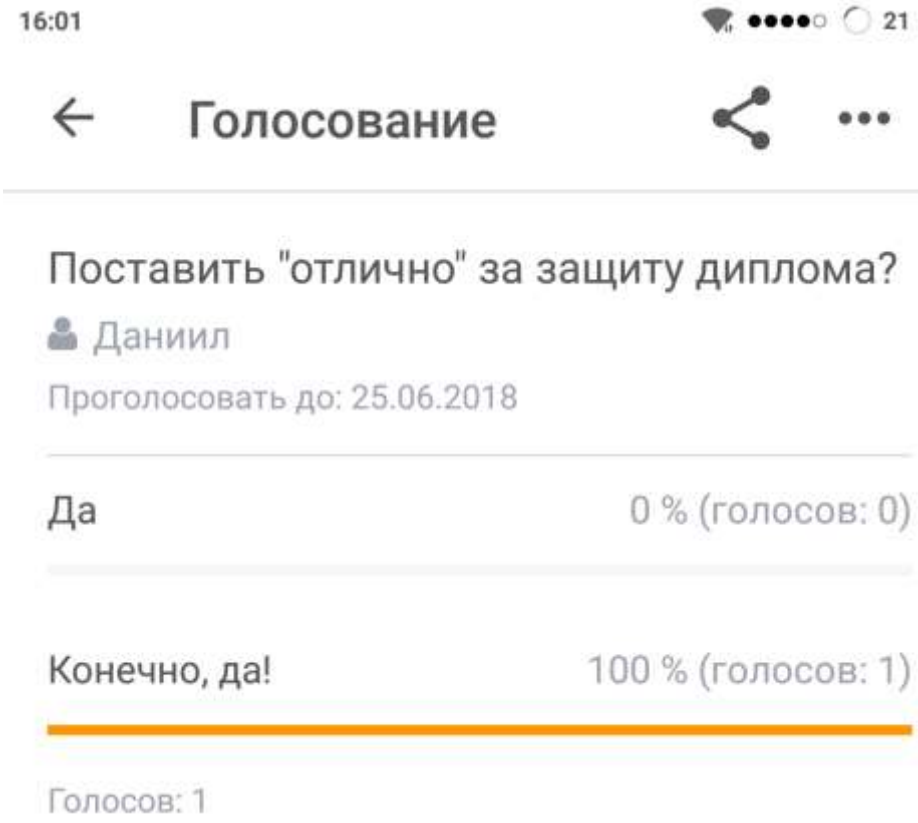

Рисунок 18 – Результаты голосования

Теперь создателю голосования стал доступен пункт в контекстном меню – «Продление». При нажатии на него, пользователь будет перенаправлен на страницу, где сможет выбрать новую дату завершения. Тут действуют

такие же ограничения, как и при создании – нельзя выбрать дату меньше текущей. После сохранения, он вернётся обратно на страницу просмотра, где уже будут обновлённые данные о голосовании. Интерфейс продления показан на рисунке 19.

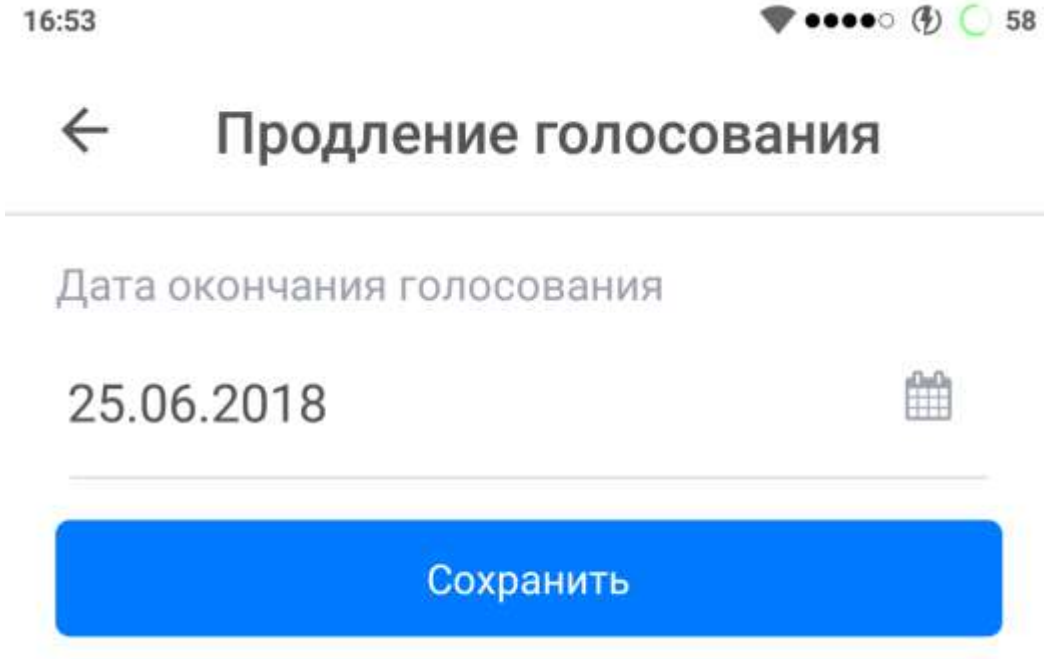

Рисунок 19 – Продление голосования

Последнее действие, которое можно выполнить с голосованием, это удалить его. При этом, пользователь получит всплывающее сообщение, с вопросом, действительном ли он хочет удалить это голосование. При утвердительном ответе, голосование будет удалено, а пользователь попадёт на страницу списка голосований. Если с момента создания или последнего изменения голосования прошло не меньше десяти минут, то в чат класса будет отправлено сообщение, о его удалении. Взаимодействие с пользователем представлено на рисунке 20.

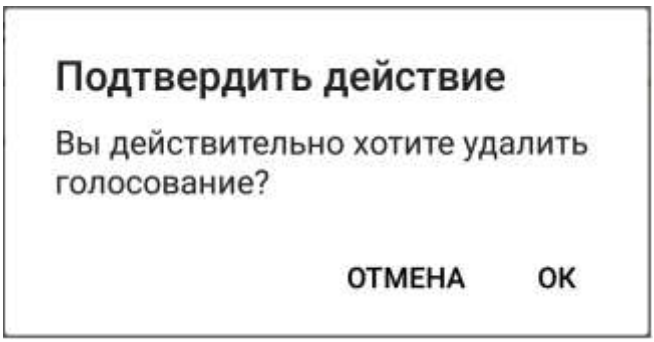

Рисунок 20 – Подтверждение удаления голосования

3.3.3 «События»

Итак, мы рассмотрели базовую функциональность голосований, и теперь можем перейти к такому документу как «Событие». Данная сущность предназначена, для организации каких-либо мероприятий. Это может быть, как родительское собрание, так и совместный поход в театр. В общем случае, событие подходит, чтобы собрать для людей проведения разного рода активностей. Перейдём непосредственно, к реализации данного функционала в приложении. Для этого в главном меню выберем пункт «События». После этого откроется страница со списком событий, разделённая на два подраздела: «Предстоящие» и «Завершённые». Таким образом, мы акцентируем внимание пользователя на событиях, в которых он ещё может принять участие, но при этом оставляем ему возможность просмотреть ранее проведённые события. В списке отображается лишь малая информация о мероприятии, только название и дата проведения. Создаётся событие по тому же принципу, что и другие сущности – нажимаем на «плюс» в правом верхнем углу и попадаем на страницу создания события. Внешний вид этого экрана можно наблюдать на рисунке 21.

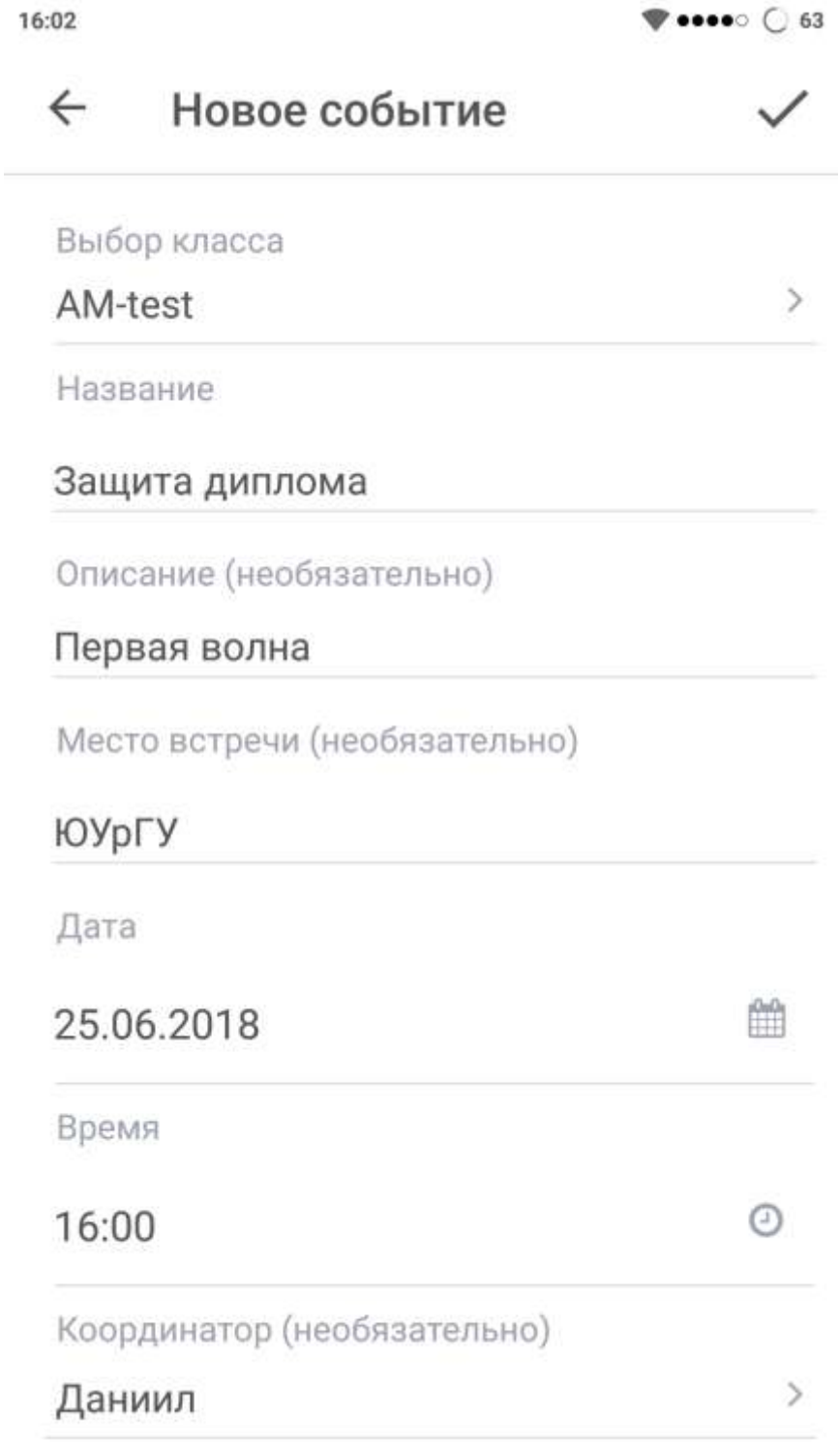

Рисунок 21 – Создание события

Описание полей, представленных на рисунке выше, приведено в таблице 4, ознакомиться с которой можно ниже. Все поля разделены на обязательные и необязательные для заполнения.

Таблица 4 – Поля события

Наименование поля	**Описание**	**Обязательное**
Название события	Название события, которое будет отображаться в списке и календаре.	Да
Описание	Дополнительная информация по событию, возможно подробное его описание или указание стоимости участия и т.д.	Нет
Место встречи	Информация, о месте проведения события.	Нет
Дата	Дата, в которую будет проходить событие. Не может меньше текущей даты.	Да
Время	Время проведения, соответственно, нельзя выбрать дату и время меньше, чем текущее плюс один час.	Да
Координатор	Человек, отвечающий за организацию мероприятия. Может быть выбран из участников класса.	Нет
Класс	Класс, которому будет принадлежать данное событие. Это поле становится доступно для изменения пользователю, если на момент создания события он состоит в двух или более классах. В противном случае, данное поле заполняется автоматически.	Да

Из особенностей данного документа стоит выделить поле «Координатор». По умолчанию, в данное поле записывается создатель события, но так-

же можно выбрать любого другого участника класса, при этом, он получит такие же права на это событие, как и его создатель. Внешний вид выбора координатора представлен на рисунке 22.

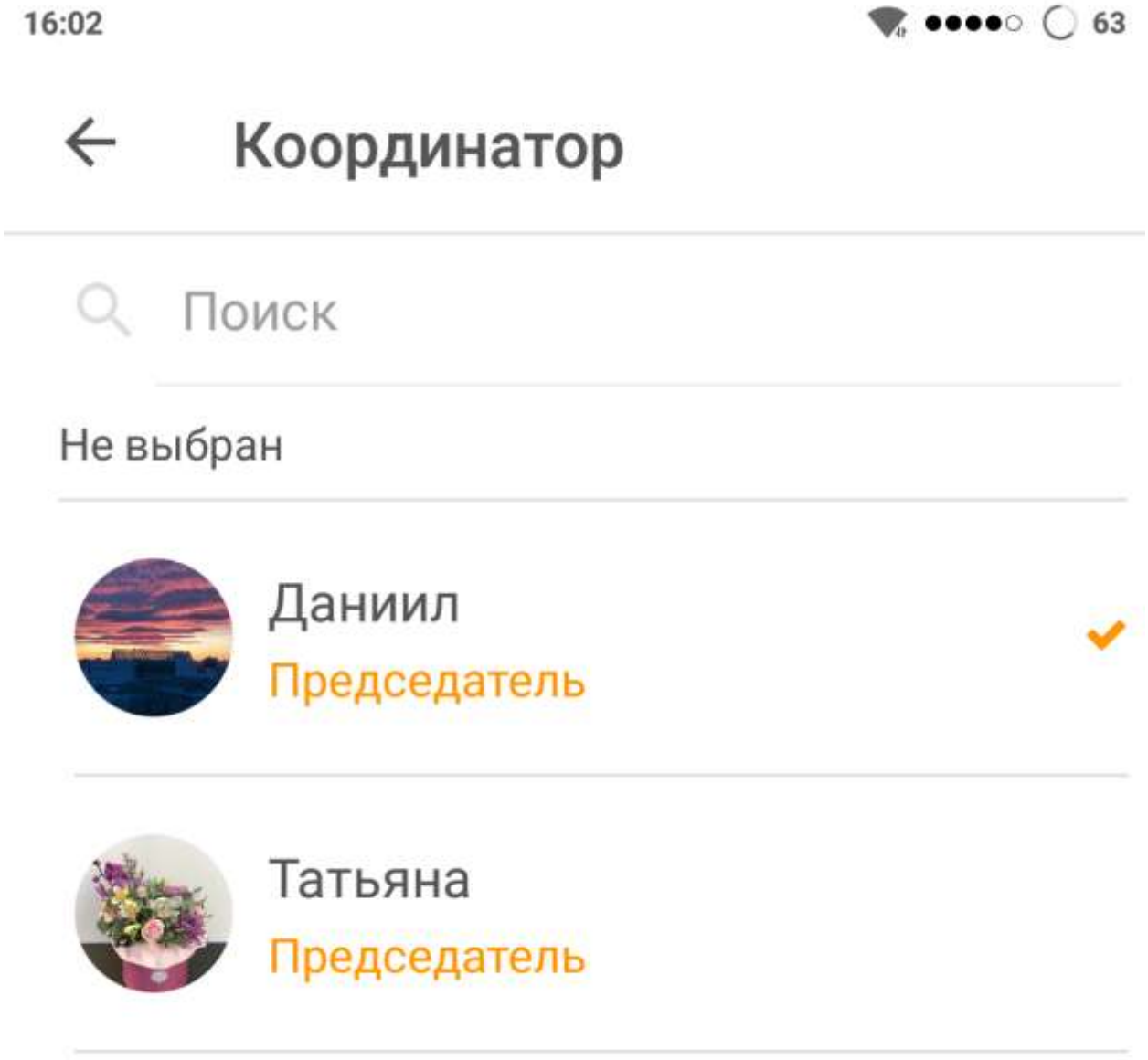

Рисунок 22 – Выбор координатора

Итак, после заполнения всех обязательных полей мы можем сохранить документ. Нажимаем на «галочку» в правом верхнем углу, и если все серверные валидации (например, на количество символом в названии и т.д.) успешно прошли, то пользователь переходит на страницу просмотра только что созданного события, внешний вид которой представлен на рисунке 23.

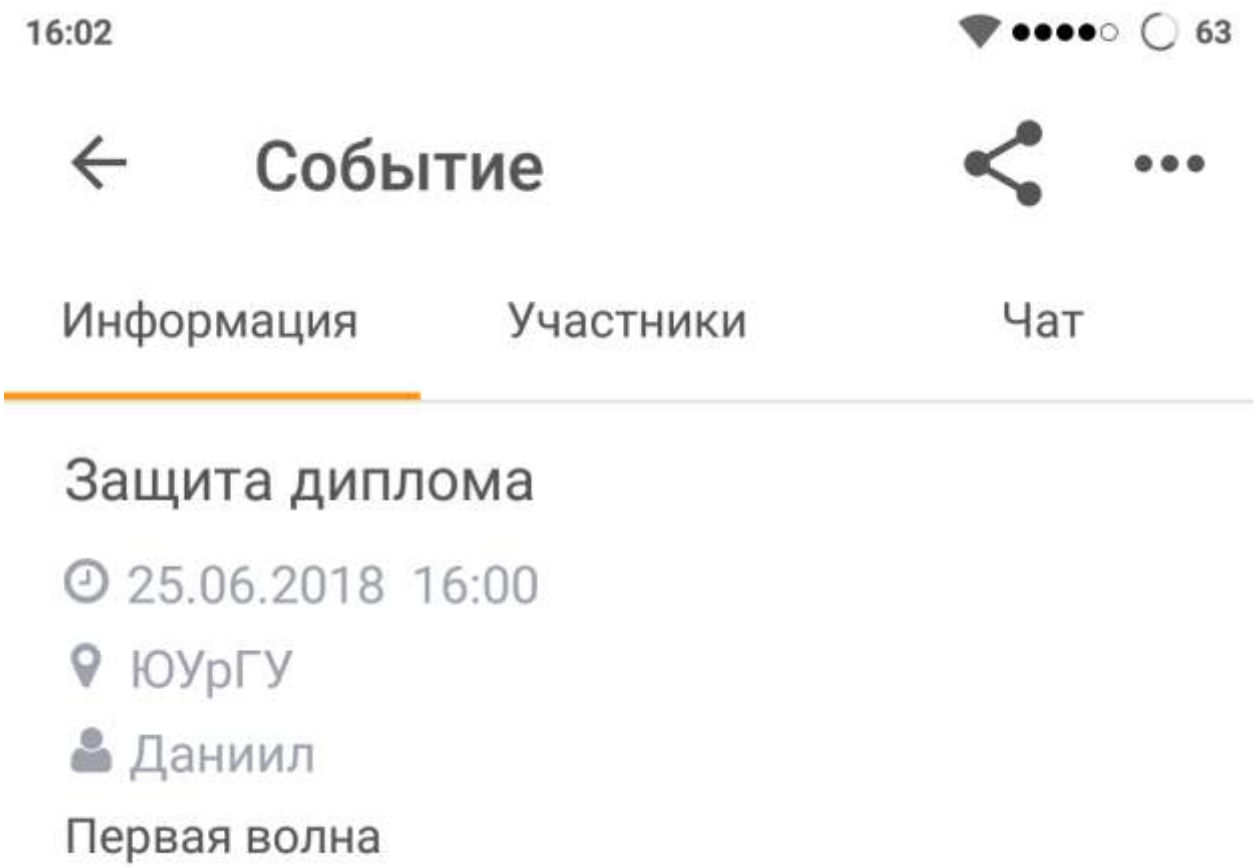

Рисунок 23 – Просмотр события

На данной странице присутствуют три вкладки: «Информация», «Участники» и «Чат». Разберём каждую в отдельности. В первом разделе пользователю доступна вся информация о событии – название, описание, дата и время проведения, а также имя координатора. Помимо этого, если событие откроет пользователь, не являющийся его создателем, то у него будет отображена кнопка «Принять участие», нажав которую, он станет участником этого события. Далее, перейдём на вторую вкладку, а именно «Участники». Тут пользователь видит список родителей, которые уже вступили в данное событие.

Ну и третья вкладка, чат – это экземпляр основного чата класса, который доступен в главном меню и о котором мы поговорим чуть позже. Единственное ограничение, данная вкладка доступна только участникам события, при попытке перейти на неё, не вступив в событие, пользователь получит соответствующее уведомление.

Теперь поговорим о контекстном меню в событии. Прежде всего, рядовому пользователю, который не являются участником или создателем, меню попросту недоступно. Далее, когда родитель присоединяется к мероприятию, у него появляется соответствующая кнопка, которая даёт возможно отказаться от участия в событии. При этом, после отказа, на вкладке информация у него снова появляется кнопка «Принять участие», а контекстное меню пропадает. Идём дальше, у создателя события и председателя класса прав чуть больше, поэтому им доступны такие пункты меню как «Изменить» и «Удалить». При выборе варианта изменит, пользователь перенаправляется на страницу редактирования события. По большому счёту, это такая же страница создания, но уже с пред заполненными полями. Значения для полей берутся из того события, которое было открыто на редактирование. После внесения необходимых изменений пользователь может сохранить изменения, после чего автоматически вернётся на страницу просмотра, где уже будут отображены новые данные. Ну и последний пункт контекстного меню – это удаление документа. При его выборе пользователя попросят подтвердить действие, так же, как это было ранее описано в главе «Голосования». Если с момента создания или редактирования прошло больше десяти минут, то сообщение об этом будет отправлено в чат класса.

3.3.4 «Объявления»

Следующий механизм, который предоставляет приложение – это «Объявления». Основная функция данного документа – это донести до класса какую-либо информацию, с дополнительной возможностью прикрепления материалов в виде электронных файлов. Собственно, открываем главное меню и переходим в список объявлений. Эти документы могут быть помечены автором как важные и такие объявления будут выделены в списке оранжевой полоской слева от информации. Также, на странице присутствует специальный свитч, которой позволяет просмотреть только важные объявления, игнорирую все остальные. В списке, у каждого документа отображается его автор, дата и время создания, а также часть текста. Помимо прочего, при создании класса, автоматически генерируется системное объявление, предлагающие пользователям оставлять обратную связь о работе приложения. Список объявлений показан на рисунке 24.

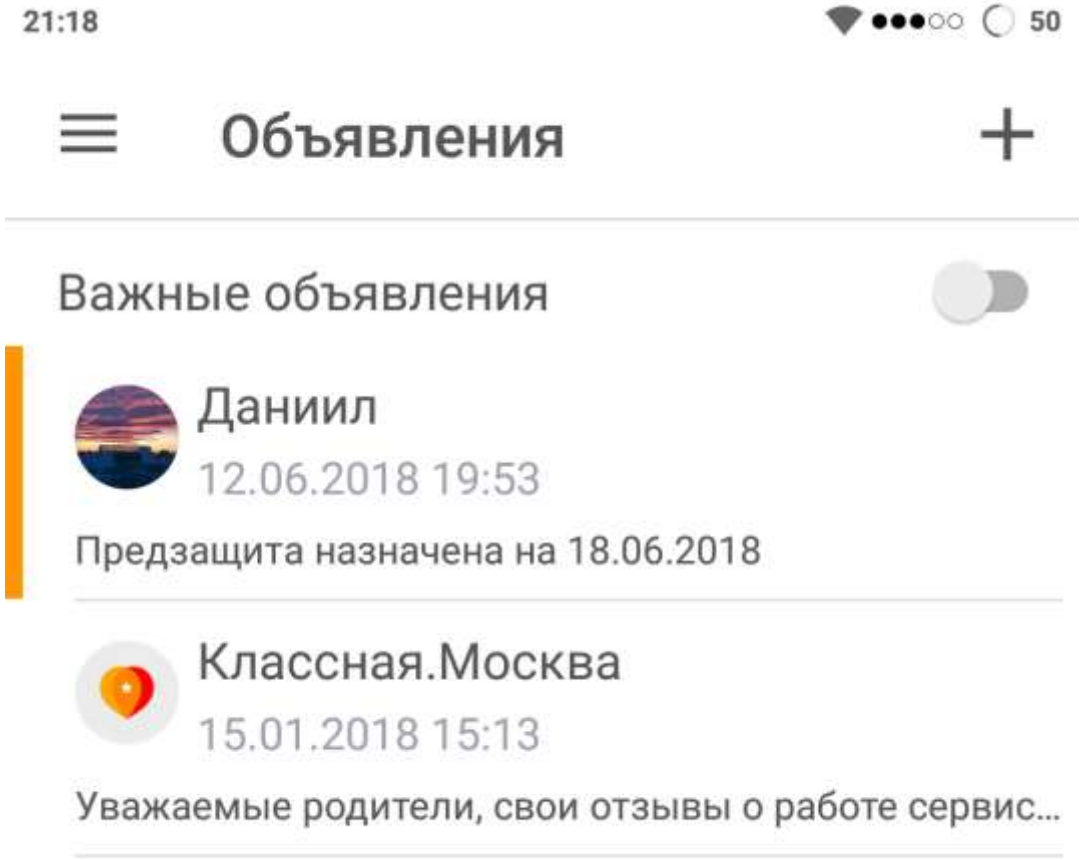

Рисунок 24 – Список объявлений

Перейдём к созданию объявления. Тут всё довольно лаконично, всего одно обязательное поле – текст объявления. Описание всех полей представлено в таблице 5.

Таблица 5 – Поля объявления

Наименование поля	**Описание**	**Обязательное**
Текст объявления	Это может быть любая информация, которую необходимо донести до класса.	Да
Важное	Индикатор того, важное это объявление или нет, имеет всего два значения, по этому устанавливается с помощью свича.	Нет
Класс	Класс, которому будет принадлежать данное объявление. Это поле становится доступно для изменения пользователю, если на момент создания объявления он состоит в двух или более классах. В противном случае, данное поле заполняется автоматически.	Да
Файл	Пользователь может прикрепить к объявлению документы или картинки, которыми он хочет поделиться с классом. Размер одного файла ограничивается пятнадцатью мегабайтами, всего к объявлению может быть прикреплено до десяти файлов.	Нет

Внешний вид страницы создания объявления представлен на рисунке 25.

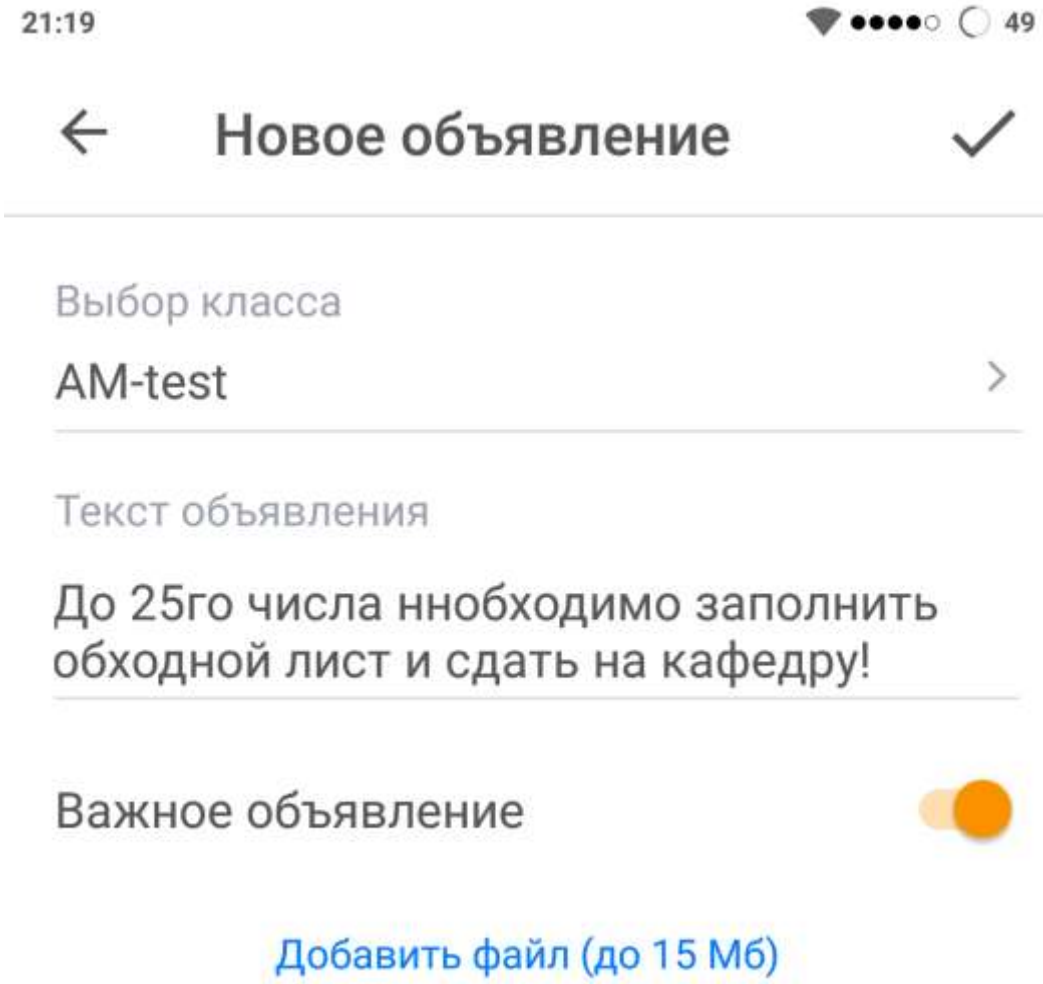

Рисунок 25. – Создание объявления

Из особенностей конкретно этого документа стоит выделить возможность прикрепления файлов. Для этого на странице имеется специальная кнопка «Добавить файл», по нажатию на которую открывается меню выбора файла, подробнее смотрите в разделе работа с файлами. После того, как выбранный документ проходит проверку на размер, начинается его отправка на серверное хранилище блобов, по окончанию которой клиент получает id, который уже прикрепляется непосредственно к документу. Как только количество документов достигает десяти, кнопка пропадает и добавление становится не доступно. После заполнения всех обязательных полей пользователь

может сохранить документ. При удачном сохранении, он будет перенаправлен на страницу просмотра только что созданного объявления.

3.3.5 Unit-тесты

Написание unit тестов – это мощный инструмент, позволяющий поддерживать проект в рабочем состоянии, на протяжении всего цикла разработки. [1] Также, покрытие кода тестами позволяет дополнительно проанализировать его на наличие ошибок или возможных доработок. Часто, в процессе написания тестов, приходится заниматься рефакторингом – переработкой ранее написанного кода в более читаемый и поддерживаемый. Помимо стандартных тестов, которые проверяю сервисы и хелперы, паттерн MVVM позволяет проверить корректность работы бизнес логики, которая выполняется непосредственно при взаимодействии пользователя с интерфейсом. А также, корректность значений, например, текстовых полей, которые отображаются на страницах. Пример покрытия тестами View Model приведён в листинге 3.

Листинг 3 – Код unit-тестов обратной связи

```
public class FeedbackViewModelTest
    {
        [Fact]
        public void IsAvailable_withoutValidEmail_should_return_false()
        {
            //Given
            var feedbackMessage = new Mock<FeedbackMessage>();
            var isValid = new Mock<IIsValidHelper>();

            var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

            //When
            sut.Email = "info@inf";
            sut.Assessment = FeedbackAssessmentEnum.Good;
            isValid.Setup(helper => helper.IsValidEmail(sut.Email)).Returns(false);

            //Then
            Assert.False(sut.IsAvailable());
        }

        [Fact]
        public void IsAvailable_withAssessment_should_return_true()
        {
            //Given
            var feedbackMessage = new Mock<FeedbackMessage>();
            var isValid = new Mock<IIsValidHelper>();

            var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

            //When
            sut.Assessment = FeedbackAssessmentEnum.Good;

            //Then
            Assert.True(sut.IsAvailable());
```

```
}

[Fact]
public void IsAvailable_withAttachment_should_return_true()
{
    //Given
    var feedbackMessage = new Mock<FeedbackMessage>();
    var isValid = new Mock<IIsValidHelper>();
    var email = "info@inf.com";
    isValid.Setup(helper => helper.IsValidEmail(email)).Returns(true);

    var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

    //When
    sut.Description = "Description";
    sut.Category = IssueTypeEnum.Bug;
    sut.Email = email;
    sut.CategoryName = IssueTypeEnum.Bug.ToString();
    sut.Attachments = new List<BlobInfo> {new BlobInfo(), new BlobInfo()};

    //Then
    Assert.True(sut.IsAvailable());
}

[Fact]
public void IsAvailable_withAttachment_withoutCategory_should_return_false()
{
    //Given
    var feedbackMessage = new Mock<FeedbackMessage>();
    var isValid = new Mock<IIsValidHelper>();
    var email = "info@inf.com";
    isValid.Setup(helper => helper.IsValidEmail(email)).Returns(true);

    var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

    //When
    sut.Description = "Description";
    sut.Email = email;
    sut.Attachments = new List<BlobInfo> {new BlobInfo(), new BlobInfo()};

    //Then
    Assert.False(sut.IsAvailable());
}

[Fact]
public void IsAvailable_withAttachment_withoutValidEmail_should_return_false()
{
    //Given
    var feedbackMessage = new Mock<FeedbackMessage>();
    var isValid = new Mock<IIsValidHelper>();
    var email = "info@inf";
    isValid.Setup(helper => helper.IsValidEmail(email)).Returns(false);

    var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

    //When
    sut.Description = "Description";
    sut.Category = IssueTypeEnum.Bug;
    sut.Email = email;
    sut.CategoryName = IssueTypeEnum.Bug.ToString();
    sut.Attachments = new List<BlobInfo> {new BlobInfo(), new BlobInfo()};

    //Then
```

```
            Assert.False(sut.IsAvailable());
        }

        [Fact]
        public void
IsAvailable_withAttachment_withoutValidEmailAndDescription_should_return_false()
        {
            //Given
            var feedbackMessage = new Mock<FeedbackMessage>();
            var isValid = new Mock<IIsValidHelper>();
            var email = "info@inf";
            isValid.Setup(helper => helper.IsValidEmail(email)).Returns(false);

            var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

            //When
            sut.Category = IssueTypeEnum.Bug;
            sut.Email = email;
            sut.CategoryName = IssueTypeEnum.Bug.ToString();
            sut.Attachments = new List<BlobInfo> {new BlobInfo(), new BlobInfo()};

            //Then
            Assert.False(sut.IsAvailable());
        }

        [Fact]
        public void IsAvailable_withAttachmentAndAssessment_should_return_true()
        {
            //Given
            var feedbackMessage = new Mock<FeedbackMessage>();
            var isValid = new Mock<IIsValidHelper>();
            var email = "info@inf.com";
            isValid.Setup(helper => helper.IsValidEmail(email)).Returns(true);

            var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

            //When
            sut.Description = "Description";
            sut.Assessment = FeedbackAssessmentEnum.BestOfTheBest;
            sut.Category = IssueTypeEnum.Bug;
            sut.Email = email;
            sut.CategoryName = IssueTypeEnum.Bug.ToString();
            sut.Attachments = new List<BlobInfo> {new BlobInfo(), new BlobInfo()};

            //Then
            Assert.True(sut.IsAvailable());
        }

        [Fact]
        public void IsAvailable_withCategory_should_return_false()
        {
            //Given
            var feedbackMessage = new Mock<FeedbackMessage>();
            var isValid = new Mock<IIsValidHelper>();
            var email = "info@inf.com";
            isValid.Setup(helper => helper.IsValidEmail(email)).Returns(true);

            var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

            //When
            sut.Category = IssueTypeEnum.Bug;
            sut.Email = email;
            sut.CategoryName = IssueTypeEnum.Bug.ToString();
```

```
        //Then
        Assert.False(sut.IsAvailable());
    }

    [Fact]
    public void IsAvailable_withDescription_should_return_false()
    {
        //Given
        var feedbackMessage = new Mock<FeedbackMessage>();
        var isValid = new Mock<IIsValidHelper>();
        var email = "info@inf";
        isValid.Setup(helper => helper.IsValidEmail(email)).Returns(false);

        var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

        //When
        sut.Description = "Description";
        sut.Category = IssueTypeEnum.Bug;
        sut.Email = email;
        sut.CategoryName = IssueTypeEnum.Bug.ToString();

        //Then
        Assert.False(sut.IsAvailable());
    }

    [Fact]
    public void IsAvailable_withDescription_should_return_true()
    {
        //Given
        var feedbackMessage = new Mock<FeedbackMessage>();
        var isValid = new Mock<IIsValidHelper>();

        var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

        //When
        sut.Assessment = FeedbackAssessmentEnum.Good;
        sut.Description = "Description";
        sut.Category = IssueTypeEnum.Bug;
        sut.Email = "info@inf.com";
        isValid.Setup(helper => helper.IsValidEmail(sut.Email)).Returns(true);
        sut.CategoryName = IssueTypeEnum.Bug.ToString();

        //Then
        Assert.True(sut.IsAvailable());
    }

    [Fact]
    public void IsAvailable_withDescription_withoutCategory_should_return_false()
    {
        //Given
        var feedbackMessage = new Mock<FeedbackMessage>();
        var isValid = new Mock<IIsValidHelper>();
        var email = "info@inf.com";
        isValid.Setup(helper => helper.IsValidEmail(email)).Returns(true);

        var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

        //When
        sut.Description = "Description";
        sut.Email = email;

        //Then
```

```
            Assert.False(sut.IsAvailable());
        }

        [Fact]
        public void IsAvailable_withDescription_withoutAssessment_should_return_true()
        {
            //Given
            var feedbackMessage = new Mock<FeedbackMessage>();
            var isValid = new Mock<IIsValidHelper>();
            var email = "info@inf.com";
            isValid.Setup(helper => helper.IsValidEmail(email)).Returns(true);

            var sut = new FeedBackVeiwModel(feedbackMessage.Object, isValid.Object);

            //When
            sut.Description = "Description";
            sut.Category = IssueTypeEnum.Bug;
            sut.Email = email;
            sut.CategoryName = IssueTypeEnum.Bug.ToString();

            //Then
            Assert.True(sut.IsAvailable());
        }
    }
```

Для написания этих тестов использовались библиотеки xUnit и Moq. Они были выбраны, так как поддерживают проекты .Net Standart. В совокупности, данные библиотеки позволяют тестировать даже те классы, которые имеют в конструкторе внешние зависимости. Для этого создаются «заглушки» данных зависимостей, которые и передаются тестируемому классу. При этом, их можно настроить – т.е. задать значения, которые будут возвращать те ли иные методы, а впоследствии проверить, были ли они вызваны.

Вывод по разделу

Все заявленные в техническом задании функции были успешно реализованы и приняты заказчиком. Код покрыт тестами и поддерживается в рабочем состоянии, что позволяет продолжать активную разработку данного проекта.

ЗАКЛЮЧЕНИЕ

В ходе реализации проекта, выполнены все поставленные заказчиком задачи. Создан мобильный клиент «Классная.Москва». В приложении реализованы следующие функции:

- Регистрация/авторизация;
- Редактирование профиля;
- Создание класса;
- Приглашение родителей в класс;
- Голосования;
- События;
- Объявления;
- Поручения;
- Календарь;
- Чат;
- Push-уведомления;
- Deep-links.

Серверная часть доработана для взаимодействия с мобильным клиентом. Добавлен функционал по отправке push-уведомлений.

Приложение было успешно реализовано, в частности, были протестированы, приняты и сданы в эксплуатацию несколько релизов. Новые версии приложения регулярно публикуются в магазины «AppStore» и «GooglePlay». На момент написания, актуальной версией проекта является релиз под номером 1.6. Также, уже ожидает тестирования 1.7 и ведётся разработка для версии 1.8. Это говорит о том, что проект всё ещё актуален и активно развивается.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Roy Osherove, The Art of Unit Testing, Second Edition with examples in C#, Forewords by Michael Feathers and Robert C. Martin, 296 pages.
2. Charles Petzold, Creating Mobile Apps with Xamarin.Forms, 448 pages.
3. Ed Snider, Mastering Xamarin.Forms – Second Edition, 192 pages.
4. https://docs.microsoft.com/en-us/xamarin/ – официальная документация по Xamarin от Microsoft.