Оглавление

[Введение 2](#_Toc515431191)

[1. Анализ задания 5](#_Toc515431192)

[1.1. Актуальность задачи 5](#_Toc515431193)

[1.2. Анализ требований 5](#_Toc515431194)

[1.3. Обзор аналогов 6](#_Toc515431195)

[2. Подход к решению задачи 10](#_Toc515431196)

[3. Проектирование 13](#_Toc515431197)

[3.1. Диаграмма вариантов использования 13](#_Toc515431198)

[3.2. Диаграмма последовательности 14](#_Toc515431199)

[4. Реализация 15](#_Toc515431200)

[4.1. Клиентская часть 15](#_Toc515431201)

[4.1.1. Разработка 15](#_Toc515431202)

[4.1.2. Интерфейсы 19](#_Toc515431203)

[4.2. База данных 27](#_Toc515431204)

[4.3. Серверная часть 29](#_Toc515431205)

[Источники 30](#_Toc515431206)

# Введение

В последние десятилетия интернет технологии стали проникать в совершенно различные сферы деятельности человека. Их полезность, зачастую превращающаяся в необходимость, становится все более очевидной. По этой причине задача развития и распространения интернет технологий стала одной из самых приоритетных, актуальных и масштабных задач. Одной из особенностей того влияния, что интернет технологии оказали на наше время, является колоссальное увеличение объемов и видов информации, с которой мы вынуждены сталкиваться ежедневно. Как никогда становится важно уметь передвигаться в этой информационной среде. Умения искать информацию, вычленять в ней главное, а также игнорировать ненужную являются одними из ключевых для успешного существования человека сегодня.

Так что же такое информация? В общем случае, согласно определению Н.В. Макаровой, информация — это сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые уменьшают имеющуюся о них степень неопределенности, неполноты знаний.

Но правильно ли говорить лишь о важности в информации разбираться? Безусловно, нет, так как всегда, когда целью стоит передача какой-то конкретной или специфичной информации, имеет смысл говорить о качестве передаваемой информации, о том виде, в котором она передается. И в рамках контекста передачи информации по интернету оную можно разделить по способу ее хранения, представления и кодирования на пять видов: текстовую, числовую, графическую, звуковую и видеоинформацию. Поговорим о последней.

Видеоинформация это, по сути, последовательность часто-сменяющихся графических изображений. Изначально необходимость передачи видеоинформации была тесно связана исключительно с кинематографом. Сейчас же с распространением и популяризацией различных видеосервисов. этот вид информации стал еще более актуален и многочислен в своих проявлениях и сферах, в которых он используется. Существует и производится огромное количество видеороликов самых различных направленностей и тематик – обучающие, научные, развлекательные, новостные и т.п. И все эти ролики необходимо уметь передавать. И это рождает две проблемы: проблему качества контента и проблему доставки этого контента.

Видеосервисам важно понимать, насколько интересным их контент является, что привлекает пользователей, а что их отталкивает. Так при должном анализе в ситуации с роликами учебной направленности это поможет понять актуальность тех или иных предметов, удобство формата подачи материала. Возможно позволит установить закономерности для более продуктивного усваивания информации учениками и в соответствии с этим адаптировать содержимое роликов. Например, касательно длительности таких роликов, их разряженности текстовой информацией и т.п.

Помимо этого, необходимо уметь объективно оценивать качество услуг, выражающихся в качестве доставки видео. Ведь если пользователь не будет доволен качеством этих услуг, всегда ли он скажет об этом? Сможет ли он сам обозначить причины проблем, с которыми столкнется? Очевидно нет, поэтому так важно знать о возможных проблемах заранее – до момента, когда пользователь решит сменить сервис, которым пользуется.

Можно сказать, что зачастую стороны, ответственные за передачу и доставку информации, в частности видеоинформации, даже не подозревают о наличии каких-то проблем и еще чаще об их причинах. Проблемы, как и их причины, могут быть самыми различными. Выявить их, ориентируясь лишь на мнения своих пользователей, даже при определенном желании, практически невозможно. И это является серьезным вызовом.

Приходит понимание необходимости сбора информации. Причем информации доступной и специфичной для сервиса. Решением является встраивание некого API в плеер для получения данных о его работе. Эти данные представляют собой информацию как о качественных характеристиках видео, так и о просмотрах: геолокации, о буферизациях, их количестве, длительности, повторяемости, о скорости, битрейте, о количестве минут просмотра, о видео-отказах и т.д. Но эти «сырые» данные мало о чем могут сказать сами по себе, поэтому инструмент, способный обсчитывать, преобразовывать и отображать эти данные в удобно-представляемом формате и является ключевым для решения вышеобозначенных проблем.

Таким образом, создание такого инструмента и адаптация его под мобильную платформу является задачей, которую я буду решать в рамках выполнения своей работы.

# Анализ задания

## Актуальность задачи

На сегодняшний день, на фоне постоянно увеличивающийся потребности пользователей смотреть видеоролики в интернете, задача выявления уровня удовлетворения от просмотра, оценки качества содержания и доставки видеороликов актуальна как никогда. Видеосервисы заинтересованы в увеличении потребления их контента. Но при этом, инструменты для обсчета и анализа данных, которые можно получить с плеера видеосервиса, отсутствуют как таковые. Безусловно, сервисы для обсчета различных данных и их визуализации существуют, но в виду своей специфики и направленности они не являются пригодными для решения поставленной задачи.

## Анализ требований

1. Поскольку данные, которые необходимо анализировать и отображать, в некотором роде привязаны к другому продукту и имеет некоторую специфичность, существовала необходимость уметь разделять отображаемые данные в зависимости от того, кто эти данные предоставляет.

По этой причине необходимо было создать механизм проверки принадлежности пользователя к «предоставляемым» им данным.

То есть была выявлена нужда создания процедуры авторизации и аутентификации.

2. Следующим требованием являлось создание возможности представления данных в удобном для человека виде. А в виду количества и особенности данных оптимальным являлось отображение этих данных в виде различных графиков.

Создавать инструмент для визуализации данных самостоятельно было сочтено неразумным в виду доступности прекрасных и удовлетворяющих всем требованиям инструментов

Таким образом была сформирована необходимость нахождения и выбора средства, способного отрисовывать и представлять данные в виде графиков.

3. Далее при анализе требований было сочтено необходимым для пользователя уметь выбирать временной период и интервал запрашиваемой выборки данных для расширения потенциала информации для обработки.

В результате было выявлено требование о создании механизма выбора интервала и шага.

4. Еще одной необходимостью в рамках реализации задачи было добавление механизма настройки и фильтрации запрашиваемых данных.

Это выразилось в обязательстве создания страницы с соответствующими настройками для обеспечения вариативности выборок данных.

5. Помимо этого, было сочтено критичным для пользователя отсутствие возможности сохранять выбранные настройки с целью просмотра их в дальнейшем.

Поэтому надобность реализации средства сохранения настроек стала очевидной.

6. В виду необходимости в развитии инструмента еще одним требованием стало грамотное проектирование системы, позволяющее производить ее расширение и модификацию с наименьшими временными и рабочими затратами.

## Обзор аналогов

При сравнении с аналогами важно понимать, на что при сравнении ориентироваться, какие особенности и возможности считать ключевыми и необходимыми в рамках решения задачи. И именно такие особенности я сведу в таблицу ниже.

Таблица 1.1 – сравнение аналогов

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Разрабатываемый продукт** | **Google Analytics** | **Splunk** |
| **Возможность выбрать несколько параметров** | Да | Нет | Да |
| **Возможность смотреть аналитику за разные промежутки времени** | Да | Да | Да |
| **Необходимая специфичность данных** | Да | Нет | Да |
| **Разнообразный формат вывода данных** | Да | Да | Нет |
| **Бесплатно** | Да | Нет | Нет |

Google Analytics располагает большими возможностями для анализа и отображения данных, но формат входных данных там весьма ограничен, а потому продукт является плохо-пригодным. Помимо этого, там нельзя одновременно выбрать разные параметры и наблюдать корреляцию между ними, что тоже является достаточно большим минусом. Не говоря уже о том, что этот сервис платный.

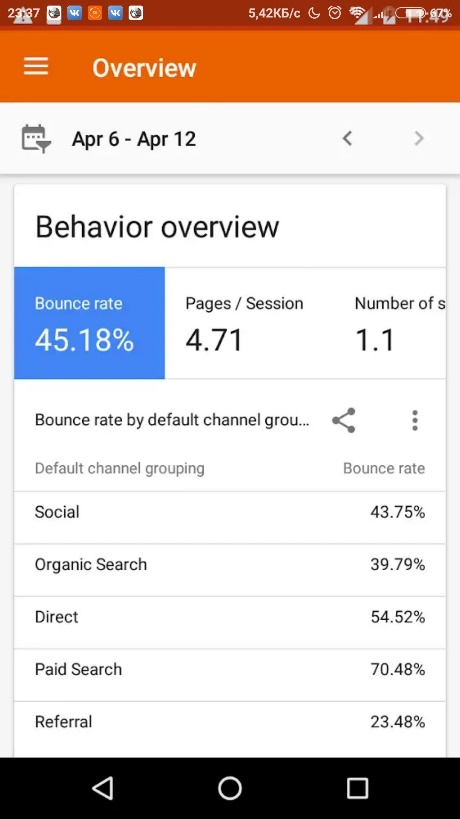
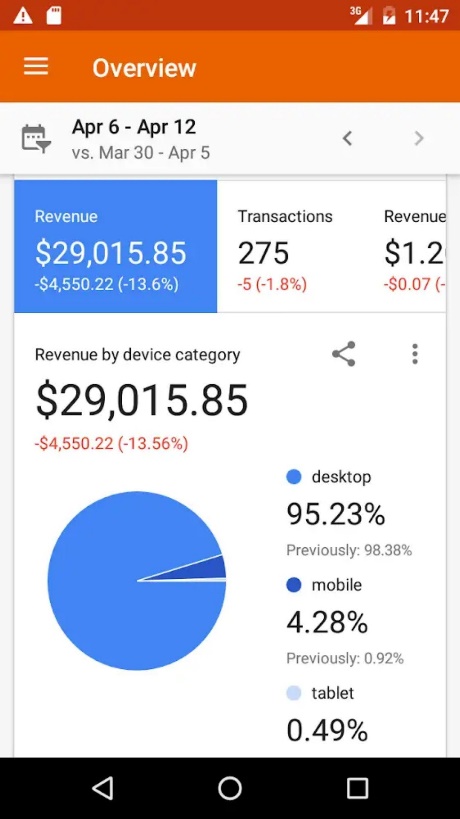


Рисунок 1.1 Рисунок 1.2

Splunk по сравнению с Google Analytics обладает большей гибкостью, касающейся входных данных, но при этом он ограничен в том, что касается разнообразия вывода этих данных. К тому же он сложен в настройке и не бесплатен.

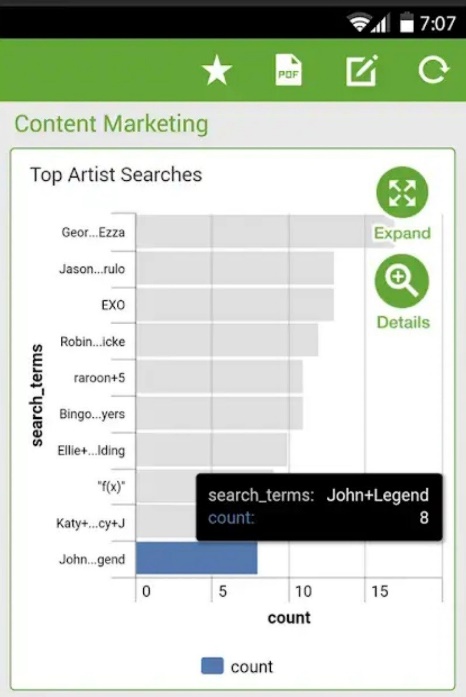


Рисунок 1.3

# Подход к решению задачи

Для решения задачи поставленной задачи был выбран следующий стек технологий:

1. React Native
2. Redux
3. Highcharts
4. Expo
5. NodeJS
6. MySQL
7. React Native – это адаптация JavaScript библиотеки React под нативный язык с возможностью написания кода под мобильные устройства.

Причины выбора были следующими:

* Используя React Native мы имеем возможность написать приложение сразу для двух, а то и трех платформ – Android, iOS & Windows Mobile. Это не значит, что эта технология кроссплатформенна, но она позволяет с легкостью создавать приложения для разных платформ, не требуя большего количества ресурсов на поддержку.

В связи с этим отпадает необходимость изучать множество технологий и языков для написания приложения под разные устройства.

* Возможность моментальных обновлений

Эта технология позволяет совершать изменения просто посредством изменения файла и последующей перезагрузкой приложения. Очевидно, что при нативной разработке для этого требуется перестраивать чуть ли не все приложение.

1. Redux – это JavaScript библиотека, используемая для управления состояниями. Зачастую используется совместно с такими библиотеками/фреймворками, как React и Angular. Является воплощением архитектуры Flux c некоторыми изменениями.

Мы будем использовать Redux как средство организации архитектуры, касающейся взаимодействий между компонентами системы. Смысл работы с использованием этой библиотеки заключается в хранении состояний компонентов в едином хранилище и передачи состояний в это хранилище.

Таким образом отпадает нужда в придумывании реализации межкомпонентного взаимодействия, что делает поток данных более понятным и избавляет от ошибок и «cпагетти-кода».

Схема наглядно отображает принцип работы Redux.

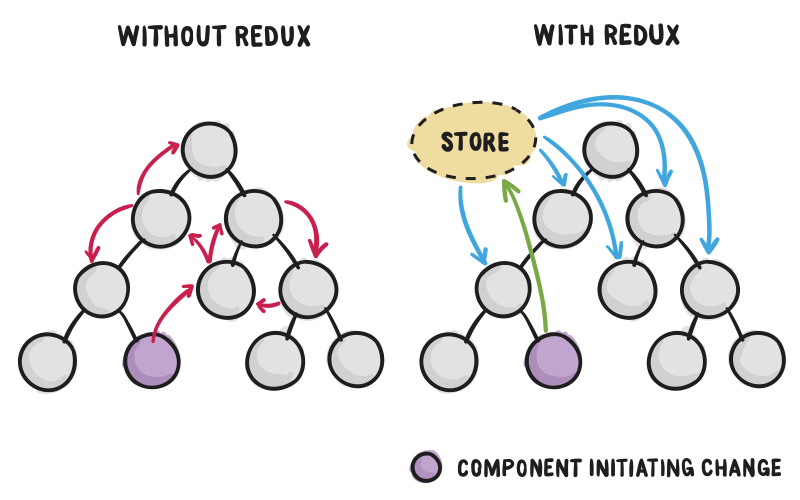


Рисунок 2.1

1. Highcharts (React-Native Highcharts) – JavaScript библиотека, позволяющая создавать различные анимированные и интерактивные графики на сайте или в приложении.

В связи с необходимостью представлять данные в понятном для человека виде было принято решение использовать соответствующую библиотеку. Highcharts был выбран по причине того, что он поддерживает большое количество видов графиков, таких как: line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, arearange, areasplinerange, columnrange and polar chart. При этом библиотека достаточно проста в использовании, обладает хорошим API и документацией к нему. Может работать с большим количеством данных. Помимо этого, она обновляемая и бесплатная для некоммерческого использования.

1. Expo – это по сути набор инструментов, выстроенных вокруг React Native. Он позволяет сходу создавать Andoid и iOS приложения, не задумываясь о платформе. Expo позволяет избежать необходимость установки таких приложений, как Android Studio и XCode. Достаточно установить лишь клиент Expo на телефон, чтобы следить за процессом разработки приложения. Кроме этого, Expo дает возможность публиковать и таким образом мгновенно обновлять уже работающее приложение.
2. NodeJS – программное средство, созданное на основе языка JavaScript. Представляет собой программный язык общего назначения. Работает на движке V8. В основе лежит событийно-ориентированное и асинхронное программирование.
3. MySQL – реляционная базой данных. Была выбрана по причине своей надежности, скорости работы, простоты использования и настройки.

# Проектирование

## Диаграмма вариантов использования

Диаграмма вариантов использования проектируется для описания прецедентов использования и их взаимосвязей актерами, такими как гость и пользователь. Прецеденты же представляют собой возможности, которые система предоставляет. Таким образом, в процессе создания подобной диаграммы мы выделяем основные действия в системе и их взаимозависимости (зависимость включения и зависимость расширения).

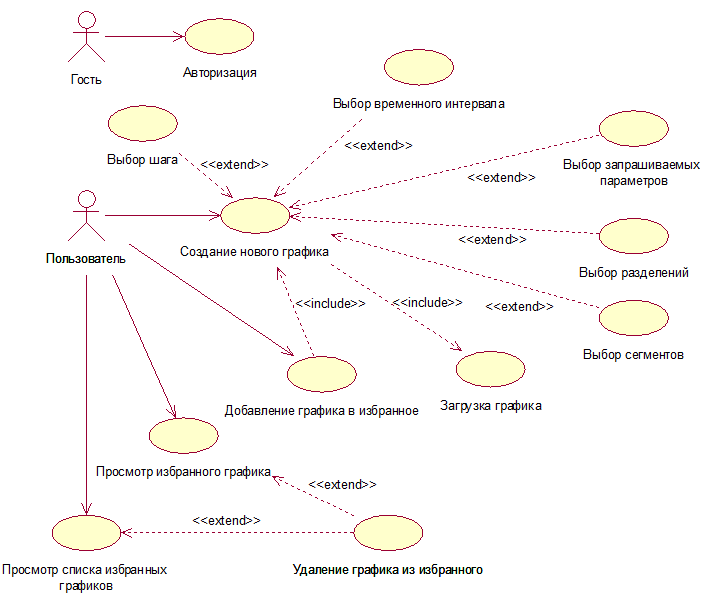


Рисунок 3.1.1 – диаграмма вариантов использования

## Диаграмма последовательности

Диаграмма последовательности строится для отображения череды действий, которые совершают пользователь и компоненты системы, в рамках выполнения какой-то конкретной операции.

На диаграмме ниже представлен процесс создания нового графика. Хочется отметить, что ввиду установленных по умолчанию настроек, операции, связанные с выбором параметров, сегментов, периода и шага, являются необязательными. Но отображены мной для полноты понимания работы системы.

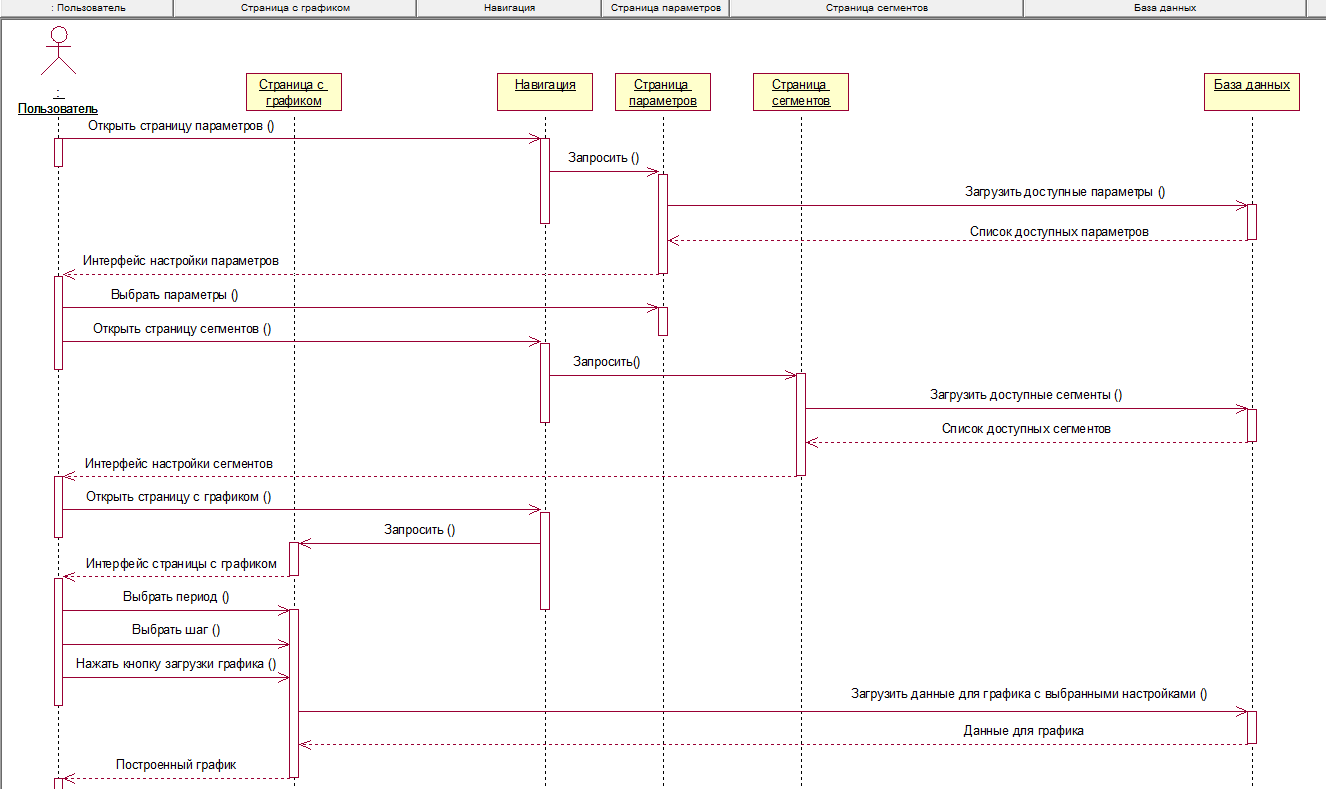


Рисунок 3.2.1 – диаграмма последовательности (Создание нового графика)

## Диаграмма классов анализа

Диаграмма классов анализа представляет собой описание системы, используя для этого три различных вида классов анализа, которые представляют собой абстракции, описывающие некоторые фрагменты системы на концептуальном уровне. Виды классов анализа можно соотнести с элементами модели MVC:

* Граничный класс – используется для обозначения элемента, моделирующего взаимодействие между актером и системой. Представляет собой некоторый интерфейс или View в MVC.
* Управляющий класс – является классом, регулирующим взаимодействия между элементами системы, либо выполняющий какие-то вычислительные операции или транзакции и т.п. Является Controller в MVC.
* Класс сущности – необходим для моделирования долгоживущий нередко сохраняемой информации. Является Model в MVC.

В таблице 3.3.1 обозначено соответствие классов их графическим отображениям.

|  |  |  |  |
| --- | --- | --- | --- |
| Варианты отображения | Граничный класс | Управляющий класс | Класс сущности |
| Графический стереотип | https://sites.google.com/site/anisimovkhv/_/rsrc/1500265181286/learning/pris/lecture/tema13/tema13_2/Analis_ClassGrStereotip.png Диалоговое окно «Нормативы» | https://sites.google.com/site/anisimovkhv/_/rsrc/1500265188014/learning/pris/lecture/tema13/tema13_2/Analis_ClassUprStereotip.png Расчет Vдоп | https://sites.google.com/site/anisimovkhv/_/rsrc/1500265173005/learning/pris/lecture/tema13/tema13_2/Analis_ClassEntityStereotip.png План |
| Стандартное обозначение со строкой-стереотипом | https://sites.google.com/site/anisimovkhv/_/rsrc/1500265177043/learning/pris/lecture/tema13/tema13_2/Analis_ClassGrStandard.png | https://sites.google.com/site/anisimovkhv/_/rsrc/1500265184611/learning/pris/lecture/tema13/tema13_2/Analis_ClassUprStandard.png | https://sites.google.com/site/anisimovkhv/_/rsrc/1500265168962/learning/pris/lecture/tema13/tema13_2/Analis_ClassEntityStandard.png |

Таблица 3.3.1

На диаграмме «Авторизация» приведен граничный класс интерфейс авторизации, которые включает в себя три других граничных класса, являющимся элементами формы. Помимо этого, мы можем увидеть на диаграмме управляющий класс «Проверка данных», которые взаимодействует с двумя классами сущности – «База данных» и «Объект Пользователь».

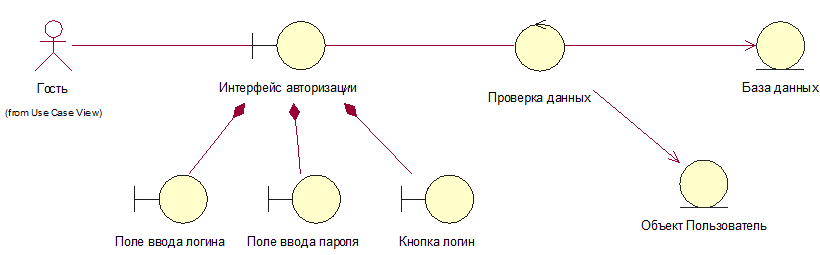


Рисунок 3.3.1 – диаграмма классов анализа. Авторизация

На диаграмме ниже изображены взаимодействия на странице с графиком. Тут мы можем увидеть главный интерфейс, включающий в себя ряд других.

Каждый из них впоследствии взаимодействует с управляющими и далее с граничными классами с целью изменения класса сущности «Объект График», который представляет собой хранилище настроек и данных отображаемого графика.

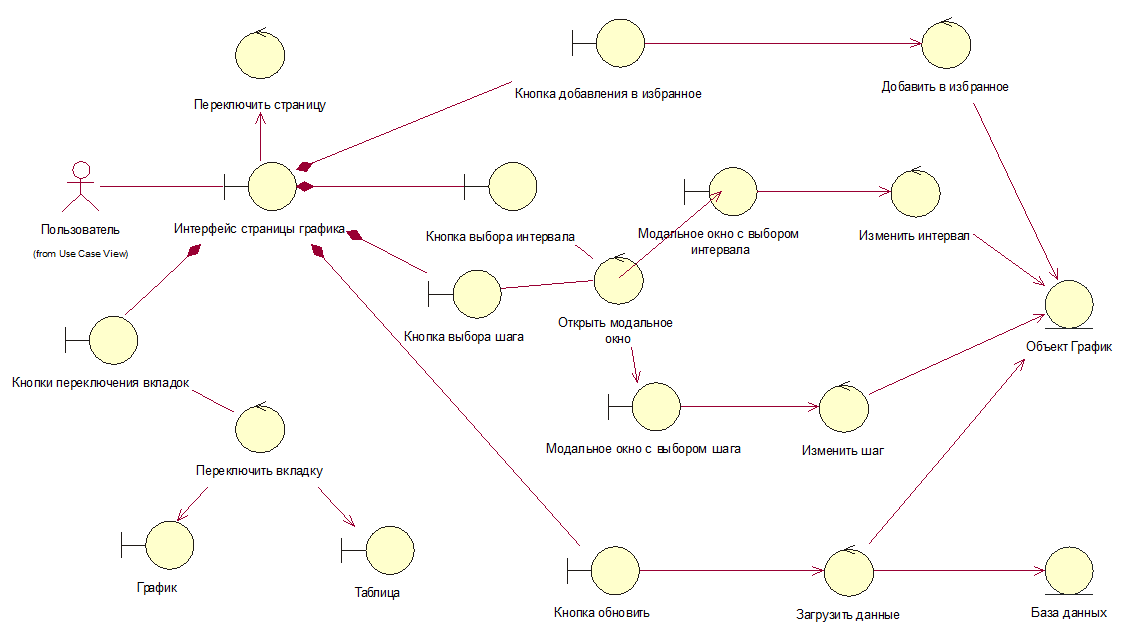


Рисунок 3.3.2 – диаграмма классов анализа. Страница с графиком

# Реализация

## Клиентская часть

### Разработка

Написание клиентской части происходило с использованием языка JavaScript на платформе NodeJS, по сути превращающую этот узкоспециализированный язык в язык широкого профиля. Это программная платформа расширяет возможности JavaScript, предоставляя ему собственный API, разрешающий последнему взаимодействовать с файлами, устройствами и т.п. Одной из ключевых возможностей является обеспечение способности подключать другие библиотеки, написанные в том числе и на других языках программирования. Это является хорошим подспорьем в разработке.

Основополагающим фреймворком для разработки является React Native, основанный на библиотеке React. Он, как и платформа NodeJS, является воплощением асинхронного и событийно-ориентированного программирования.

В React Native приложение строится из компонент платформы — это нативные модули, завернутые в React-компоненты. У фрейморвка, являющегося интерпретацией веб библиотеки React’a, есть на первых взгляд ряд неочевидных особенностей:

* Отсутствует HTML, в качестве альтернативы используется JSX.
* Нет CSS, вместо него используются похожие на CSS полифилы.
* DOM API отсутствует как таковой.
* Полная поддержка ES6/ES7. Код нативный, исполняется на NodeJS, поэтому нет нужды беспокоиться о поддержке браузерами, как это происходит в вебе.

Пример использования JSX в функциональной компоненте

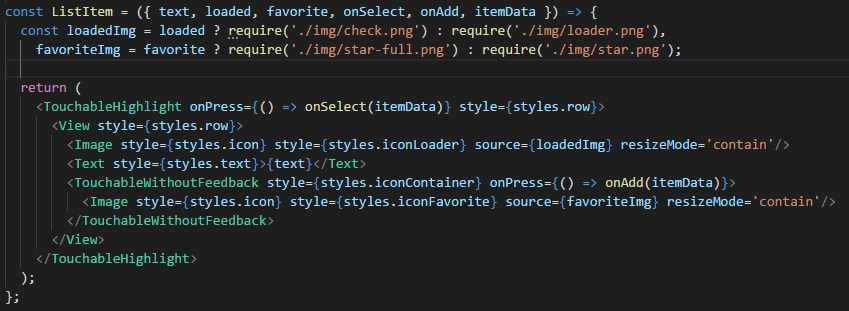


Рисунок 4.1.1.1

Пример использования CSS-полифила «react-native-extended-stylesheet»



Рисунок 4.1.1.2

По умолчанию каждая компонента является наследником стандартной React Native компоненты и обладает жизненным циклом, выражающимся в наличии методов-событий, позволяющими управлять компонентой. В частности, именно это и является проявлением событийно-ориентированного подхода в React Native.

Из соображений оптимизации приложения и упорядоченности структуры организации проекта имеет смысл делать компоненты функциональными, если они простые и не обладают серьезной поведенческой логикой, и объектно-ориентированными поддерживающими методы жизненного цикла стандартной React компоненты в другом случае.

Этот подход нашел применение в проекте и проявляется даже в структуре папок.

На рисунке 4.3 наиболее сложные компоненты, представляющие собой главные интерфейсы приложения находятся в папке screens, а более простые, те, из которых собираются сложные, находятся в папке components.

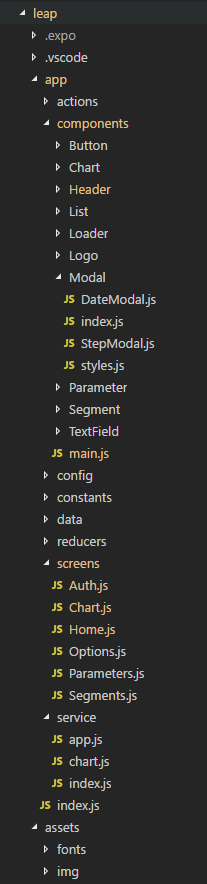


Рисунок 4.1.1.3

Полное описание структуры работы приложения содержится в таблице ниже.

|  |  |
| --- | --- |
| **Папка** | **Назначение** |
| app/components | Содержит компоненты, выступающими, условно говоря, строительными блоками, из которых собираются интерфейсы. Папка-компонент стандартно содержит 3 файла:   * styles.js – описывает стили компонента * componentName.js – файл с самим компонентом * index.js – файл, в который импортируются соседние файлы с последующим их экспортом. Существует для удобства импорта соседних файлов из других папок проекта   Помимо этого, там могут содержаться папки с изображениями, если те используются. |
| app/data | Содержит примеры данных для работы с приложением в качестве демонстрации |
| app/service | Содержит служебные функции по работе с данными, по выполнению запросов и т.п. Вынесены отдельно для использования в разных местах. |
| app/constants | Содержит основные константы для работы с проектом |
| app/config | Содержит конфигурационные файлы, являющимися ключевыми для работы приложения. В частности, там содержится 2 файла:   * store.js – инициализирует хранилище состояний, которое импортируется и используется повсеместно. Для своей работы использует файлы из папки reducers & actions. Является воплощением особенности работы с redux. * routes.js – инициализирует навигацию между интерфейсами проекта. |
| app/actions | Содержит файлы, описывающие возможные функции-события, возвращающие объекты. Эти функции используются для формирования объектов при транслировании событий в хранилище состояний. Является воплощением особенности работы с redux. |
| app/reducers | Содержит файлы, описывающие функции обработки событий. Возвращают новое состояние хранилища. Является воплощением особенности работы с redux. |
| app/screens | Содержит главные компоненты, описывающими основные интерфейсы приложения. |
| assets | Содержит шрифты и изображения, используемые для проекта. |

Таблица 4.1.1.1

### 4.1.2. Интерфейсы

* Экран загрузки

На рисунке 4.1.2.1 представлен экран загрузки. Он отображается в момент загрузки приложения. В этот момент загружаются шрифты, картинки и сохраненная на устройстве конфигурация. Сохранение последней необходимо в частности для того, чтобы пользователю при повторном заходе в приложение не нужно было бы авторизовываться повторно.



Рисунок 4.1.2.1

* Страница авторизации

После открытия приложения, если пользователь еще не авторизовывался, перед ним откроется соответствующая страница, где необходимо будет ввести свои учетные данные. При вводе неверных данных пароль будет обнулен. Страница авторизации представлена на рисунке 4.1.2.2

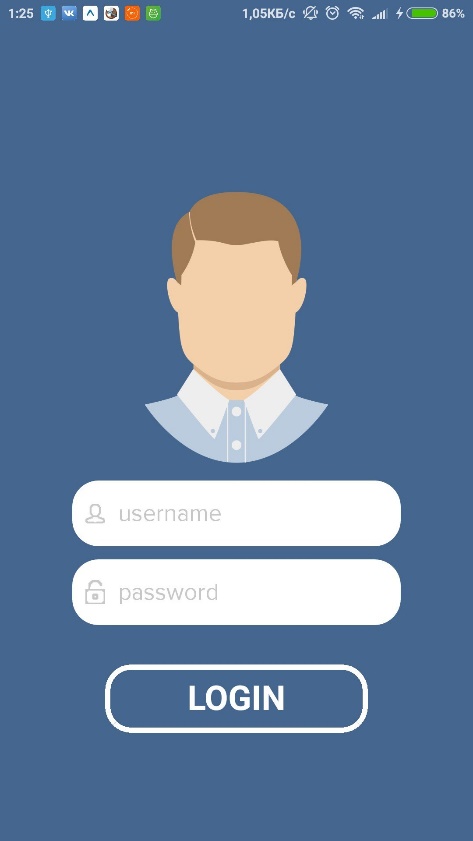


Рисунок 4.1.2.2

Стоит отметить, что при фокусе на элементе ввода, текстовые элементы приподнимутся относительно своего обычного положения, чтобы пользователь мог видеть вводимые им данные. Это изображено на рисунке 4.1.2.3

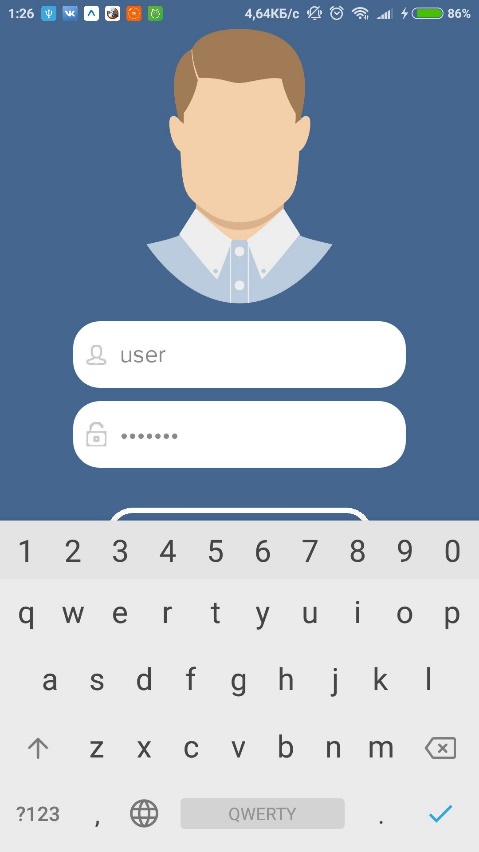


Рисунок 4.1.2.3

* Домашняя страница

В случае успешной авторизации или при открытии приложения с уже осуществленной авторизацией пользователь будет адресован на домашнюю страницу. Эта страница является стартовой и предназначена для управления избранными графиками и для создания новых.

Чтобы создать новый график необходимо нажать на кнопку «Новый график». После чего осуществится переход на страницу с новым графиком со стандартными настройками.

Помимо этого, тут есть список избранных графиков, слева от каждого элемента в виде иконки отображается его статус.

Если это иконка загрузки, значит данные для графика еще не загрузились, если это иконка с галочкой, то соответствующие данные уже загрузились и график можно посмотреть.

Соответственно, чтобы это сделать, необходимо нажать на график, чтобы осуществить переход на него. Стоит отметить, что даже если данные графика не загрузились, его все равно можно открыть.

При нажатии на звездочку справа от графика, его можно удалить из избранного и снова его туда добавить. Удаленные графики обновятся при следующей загрузке приложения и соответственно отображаться в списке не будут.

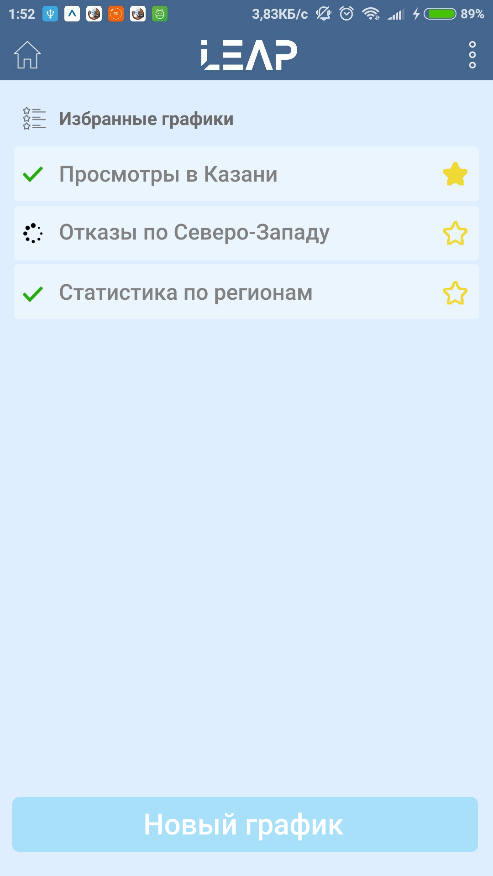


Рисунок 4.1.2.4

* Страница с графиком

На рисунке 4.1.2.5 представлен интерфейс страницы с графиком.

На верхней панели слева направо расположены следующие элементы:

1. Кнопка назад – при нажатии на нее происходит переход на домашнюю страницу
2. Название графика – то название, которое ввел пользователь при его редактировании
3. Иконка, отображающая статус загрузки графика.
4. Кнопка звездочка – элемент, одновременно отображающий состояние присутствия графика в избранном и позволяющей совершить соответствующие действия.
5. Кнопка меню. При нажатии на нее появляется окошко меню

На панели ниже находится интерфейс, отображающий выбранный период времени и шаг.

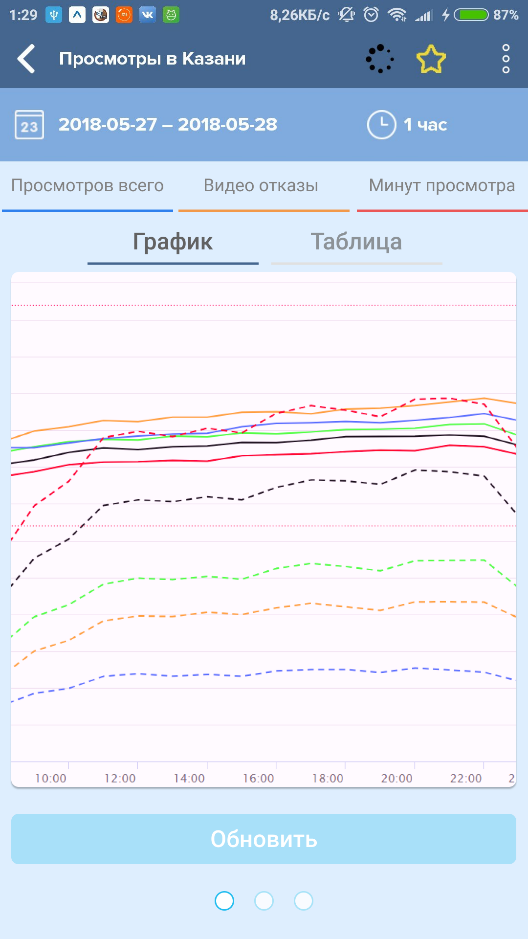


Рисунок 4.1.2.5

При нажатии на интервал появляется модальное окно с настройкой интервала. На нем выделяется текущий период. Чтобы выбрать новый период, можно воспользоваться одним из шаблонов в списке – достаточно просто кликнуть на элемент списка. Произойдет смена интервала и окно закроется. Либо можно сделать тоже самое вручную, выбрав произвольную дату на календаре.



Рисунок 4.1.2.6

При нажатии на шаг появляется похожее модальное окно теперь уже с выбором шага. Выбор шага ограничен стандартными значениями.

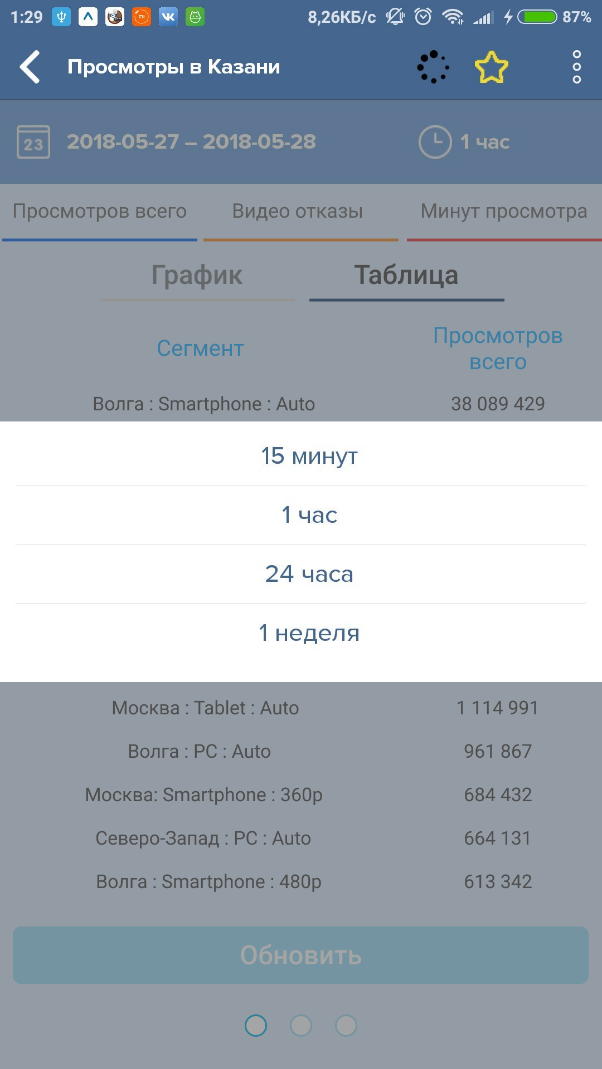


Рисунок 4.1.2.7

Также, стоит сказать о возможности переключиться на таблицу, где отображаются суммарные значения для абсолютного параметра и средние для относительного. Изменить параметр в таблице можно, кликнув по параметру из списка активных выше.



Рисунок 4.1.2.8

# 4.2. База данных

Для разработки базы данных использовалась СУБД MySQL.

MySQL относится к реляционным базам данных, поддерживает язык SQL и обладает рядом преимуществ:

1. Эта база данных является весьма надежной, не подвержена потерям данных.
2. При хранении небольшого количества информации является достаточно производительной.
3. Не требовательна к аппаратным ресурсам устройства.
4. Является кроссплатформенной СУБД.

Для реализации необходимого функционала клиентского приложения существовала необходимость в двух таблицах:

Таблица 4.2.1 - user

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Комментарий** | **Внешний ключ** |
| id | INT(10) | Идентификатор | - |
| login | VARCHAR(128) | Логин для входа в систему | - |
| password | VARCHAR(128) | Пароль для входа в систему | - |
| name | VARCHAR(128) | Полное имя | - |
| service\_name | VARCHAR(128) | Принадлежность к сервису | - |
| demo | enum(‘0’, ‘1’) | Будет ли пользователь получать пробные данные или настоящие | - |

Таблица 4.2.2 - favorite

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Тип** | **Комментарий** | **Внешний ключ** |
| id | INT(10) | Идентификатор | - |
| user\_id | INT(10) | Привязка к конкретному пользователю | user.id |
| name | VARCHAR(128) | Название | - |
| step | INT(10) | Выбранный шаг | - |
| date\_start | DATE | Начало периода | - |
| date\_end | DATE | Конец периода | - |
| segments | TEXT | JSON с информацией о параметрах | - |
| parameters | TEXT | JSON с информацией о сегментах | - |

# 4.3. Серверная часть

Целью работы являлась, безусловно, разработка именно мобильного приложения, но при этом взаимодействие с сервером было для него необходимостью для корректного и полноценного функционирования.

Например, для авторизации и взаимодействия с избранными графиками.

По этой причине было принято решение написать соответствующий серверный API.

* В качестве платформы был выбран NodeJS в виду того, что он сполна удовлетворяет необходимым потребностям.
* В качестве сервера был выбран Express, являющийся минималистичным и гибким веб-фреймворком для приложений Node.js и предоставляющий обширный набор функций для мобильных и веб-приложений. Он имеет в своем распоряжении множество служебных HTTP методов и различных обработчиков, что позволяет легко и просто создавать надежные API.
* В качестве инструмента документирования и тестирования API был выбран open source фреймворк Swagger, способный генерировать клиентский веб интерфейс, являющийся одновременно и документацией, и способом проверки работоспособности запросов. Генерация происходит на основе описываемых правил, оформленных определенным образом в виде комментариев к коду.

# Источники

React Native - URL: <http://facebook.github.io/react-native/>

Expo – URL: https://expo.io/