MINISTERUL EDUCAŢIEI AL REPUBLICII MOLDOVA

UNIVERSITATEA DE STAT „ALECU RUSSO” DIN BĂLŢI

FACULTATEA DE ŞTIINŢE REALE, ECONOMICE ȘI ALE MEDIULUI

CATEDRA DE MATEMATICĂ ȘI INFORMATICĂ

**UTILIZAREA FRAMEWORKULUI JS PENTRU CREAREA APLICATIEI MOBILA HIBRIDA**

**TEZĂ DE LICENTA**

**Autor:**

Studenta al grupei IS31Z

**Stanislava VOVOD**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnatura)

**Conducător științific:**

**Corina NEGARA**

dr., conf., univ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnatura)

**Dumitru STOIAN**

asist. univ.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnatura)

**BĂLȚI, 2021**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ, КУЛЬТУРЫ И ИССЕЛЕДОВАНИЯ

РЕСПУБЛИКИ МОЛДОВА

БЕЛЬЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. «АЛЕКУ РУССО»

ФАКУЛЬТЕТ ТОЧНЫХ НАУК, ЭКОНОМИКИ И ОКРУЖАЮЩЕЙ СРЕДЫ

КАФЕДРЫ МАТЕМАТИКИ И ИНФОРМАТИКИ

**ИСПОЛЬЗОВАНИЕ ФРЕЙМВОРКА JS ПРИ СОЗДАНИИ ГИБРИДНОГО МОБИЛЬНОГО ПРИЛОЖЕНИЯ**

**ЛИЦЕНЗИОННАЯ РАБОТА**

**Автор:**

Студентка группы IS31Z

**Станислава ВОВОД**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Научный руководитель:**

**Корина НЕГАРА**

др., конф., унив.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Дмитрий СТОЯН**

асист. унив.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**БЭЛЦЬ, 2021**

Controlată:

Data „\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021

Conducător ştiinţific: dr., conf. univ., Corina NEGARA

asist. univ., Dumitru STOIAN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Aprobată

şi recomandată pentru susţinere

la şedinţa Catedrei de ştiinţe economice

Proces-verbal nr. \_\_\_\_\_\_ din „\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_202\_\_

Şeful catedrei dr., conf. univ. Corina NEGARA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnatura)

*ADNOTARE*

Lucrare autorizată. „ Utilizarea frameworkului JS pentru crearea aplicației mobila hibrida.”, Vovod Stanislava. Specialitate: 0444.1-Informatică (științe exacte). Bălți, 2021. Structura operei. Lucrarea autorizată conține o introducere, două capitole, concluzii, o listă de referințe din 20 de surse, 44 de pagini din textul principal, 10 figuri.

Cuvinte cheie: frameworkuri JS, comparare frameworkurilor, aplicație web, aplicație mobila, aplicație mobile hibrida.

Scopul lucrării este analizarea si compararea frameworkurilor JS; alegerea unui framework la care se dezvolta o aplicație mobila hibrida demo care utilizează cadrul Vue.js.

*АННОТАЦИЯ*

Лицензионная работа. «Использование фреймворкa JS при создании гибридного мобильного приложения.», Вовод Станислава. Специальность: 0444.1-Информатика (точные науки). Бэлць, 2021. Структура работы. Лицензионная работа содержит введение, три главы, выводы, список использованной литературы из 20 источников, 40 страниц основного текста, 10 рисунков, одну таблицу.

Ключевые слова: JS-фреймворки, сравнение фреймворков, веб-приложение, мобильное приложение, гибридное мобильное приложение.

Целью работы является анализ и сравнение JS-фреймворков; выбор фреймворка для разработки гибридного мобильного демонстрационного приложения, использующего фреймворк Vue.js.

*ANNOTATION*

Licensed work. " Using Google Services in hybrid mobile applications JS on Vue.js", Vovod Stanislava. Specialty: 0444.1-Computer science (exact sciences). Balti, 2021. The structure of the work. The licensed work contains an introduction, two chapters, conclusions, a list of references from 20 sources, 44 pages of the main text, 10 figures.

Keywords: JS frameworks, framework comparation, web application, mobile application, hybrid mobile application.

The scope of this work is to analyze and compare JS frameworks; choosing a framework to develop a hybrid mobile demo application that uses the Vue.js framework.

Aprobat:

Șeful catedrei de matematică și informatică

dr., conf. univ. C.Negara

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_\_

**Graficul calendaristic de executare a tezei de licență**

Tema tezei de licența: UTILIZAREA FRAMEWORKULUI JS PENTRU CREAREA APLICATIEI MOBILA HIBRIDA

confirmată prin ordinul rectorului USARB nr.\_\_\_ din „\_\_\_\_\_\_\_\_\_\_\_\_\_”

Termenul limită de prezentare a tezei de licență la Catedra de matematică și informatică „\_\_\_\_\_\_\_\_\_\_\_”.

**Etapele executării tezei de licență:**

|  |  |  |
| --- | --- | --- |
| Etapele | Termenul de realizare | Viza de executare |
| 1. Stabilirea temei; fixarea obiectivelor; selectarea surselor de informare. | 11.09.20 | realizat |
| 1. Investigația cadrului teoretic al cercetării; expunerea cadrului teoretic al cercetării. | 03.10.20 | realizat |
| 1. Întocmirea problemei cercetării; stabilirea tipului de cercetare. | 21.10.20 | realizat |
| 1. Specificarea unităților studiate. | 10.11.20 | realizat |
| 1. Alegerea metodelor de cercetare. | 18.11.20 | realizat |
| 1. Culegerea datelor; selectarea modalităților de prelucrare a datelor; stocarea datelor; analiza datelor | 21.01.21 | realizat |
| 1. Elaborarea concluziilor și a recomandărilor practice. | 15.03.21 | realizat |
| 1. Susținerea preventivă a tezei. | 30.04.21 | realizat |

Student (ă) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura)

Conducători științifici \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(*semnătura*)

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc41253564)

[ВВЕДЕНИЕ 3](#_Toc41253565)

[1. ИСПОЛЬЗОВАНИЕ ГУГЛ СЕРВИСОВ В СОВРЕМЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ 5](#_Toc41253566)

[1.1. Один из многофункциональных сервисов гугл - Firebase 6](#_Toc41253567)

[1.1.1. Аутентификация в Firebase 6](#_Toc41253568)

[1.1.2. Firebase Хостинг 7](#_Toc41253569)

[1.1.3. Cloud Storage 8](#_Toc41253570)

[1.1.4. База данных реального времени Firebase 9](#_Toc41253571)

[1.2. Google OAuth 10](#_Toc41253572)

[1.2.1. Абстрактное описание протокола 11](#_Toc41253573)

[1.2.2. Регистрация приложения 12](#_Toc41253574)

[1.2.3. Идентификатор клиента и секрет клиента 12](#_Toc41253575)

[1.2.4. Итог по первой главе 12](#_Toc41253576)

[2. ГИБРИДНЫЕ ПРИЛОЖЕНИЯ ИСПОЛЬЗУЯ VUEJS И VUETIFY 13](#_Toc41253577)

[2.1. VueJS при создании пользовательских интерфейсов………………………...14](#_Toc41253578)

[2.2. Vuetify - Framework для создания гибридных мобильных приложений 15](#_Toc41253579)

[3. ВНЕДРЕНИЕ ГУГЛ-СЕРВИСОВ В СИСТЕМЫ, ИСПОЛЬЗУЮЩИЕ VUEJS 17](#_Toc41253580)

[3.1. Описание структуры приложения 17](#_Toc41253581)

[3.2. Установка и настройка среды разработки 17](#_Toc41253582)

[3.2.1. Node Package Manager (NPM) 17](#_Toc41253583)

[3.2.2. Инструменты командной строки (CLI) 18](#_Toc41253584)

[3.2.3. Использование Cordova для определения гибридной трансляции 18](#_Toc41253585)

[3.2.3. Создание приложения 18](#_Toc41253586)

[3.3. Создание проекта в Firebase 19](#_Toc41253587)

[3.4. Интеграция Firebase 19](#_Toc41253588)

[Заключение 24](#_Toc41253589)

[Библиография 25](#_Toc41253590)

# ВВЕДЕНИЕ

В наше время прогресс достиг невиданных высот. С появлением интернета технологии развиваются с такой скоростью, что уследить за всеми практически невозможно. Каждый год появляются все новые и более инновационные площадки, сервисы, приложения, позволяющие делиться информацией, обучаться, работать, веси бизнес и многое другое, не выходя из дома. Однако, все возрастающая популярность приложений привела к появлению их различных вариаций, а именно, появились такие понятия, как нативные, гибридные и веб приложения.

Согласно последнему опросу Red Hat, одного из мировых провайдеров программных решений с открытым исходным кодом, сегодня 50% компаний нуждаются в специалистах по мобильной разработке или менеджерах мобильных проектов. Интересно, что в этой доле компаний нативные разработчики слабо востребованы. Лидирующие позиции занимают front-end специалисты — 32% опрошенных голосуют за front-end.

Данные исследований Gartner также подтверждают повышение интереса со стороны бизнеса к мобильной разработке. По их прогнозам, к концу 2017 года спрос на разработку мобильных приложений будет расти в 5 раз быстрее, чем штат IT компаний, способных удовлетворить этот спрос.

Исходя из динамики развития мобильных приложений, мировые тенденции подтверждаются увеличением спроса на мобильную разработку, растет скорость создания приложений, повышается интерес к простым решениям с доступной стоимостью разработки и удобным, понятным интерфейсам. Всё это открывает возможности для совершенствования гибридных приложений, которые все увереннее закрепляются на мировом рынке и не сдают свои позиции. Отдавая предпочтение тем или другим, важно отслеживать состояние рынка и правильно расставлять приоритеты по скорости запуска проекта, стоимости и производительности.

*Цель*данной работы состоит в исследовании гибридных web технологий и возможностей фреймворков JS для их сравнительного анализа и последующей разработки демонстрационного гибридного мобильного приложения на фреймворке Vue.js.

Для выполнения поставленной цели были спланированы следующие *задачи*:

* Изучение основ гибридных web технологий;
* Исследование возможности использования гибридных web технологий в качестве замены нативных технологий;
* Сравнительный анализ возможностей гибридных технологий в разных фреймворках;
* Аргументы в пользу выбора фреймворка Vue.js для дальнейшей разработки;
* Создание мобильного приложения используя гибридные web технологии фреймворка Vue.js.

*Ценность работы:*Данная работа полезна своей теоретической частью, так как было произведено исследование основных технологий, используемых для создания гибридных web приложений, а также сравнительный анализ фреймворков JS и выбора самого оптимального для разработки гибридного мобильного приложения. Практическая же часть будет полезна тем, что в ней описано назначение и процесс создания гибридного мобильного приложения на выбранном фреймворке JS – Vue.js.

*Актуальность:* данная работа является актуальной и будет таковой еще долгое время, так как мобильные устройства в нынешнее время предпочитает все большее количество потребителей. Они портативные, компактные и позволяют пользователям использовать их в любом месте и в любое время. С этим так же растет и спрос на мобильные приложения, а гибридные мобильные приложения - это набор технологий, которые призваны сочетать в себе основные преимущества мобильной платформы будучи обычным web приложением, что дает им большое преимущество на ряду с другими технологиями.

Основываясь на поставленных целях, работа была разделена на 3 главы:

В первой главе было проведено исследование технологий в составе гибридных мобильных приложений таких как Firebase API, Google OAuth и другие.

Во второй главе описаны гибридные web приложений, и таких технологий так Vue, Vuetify и Cordova для создания мобильного веб приложения.

В третьей главе описывается процесс создания гибридного web приложения используя данный набор технологий:

В ходе данной работы будет разработано приложение, которое является аналогом twitter и включает в себя набор функций для создания и размещения постов. Это будет гибридное web приложение, включающее в себя авторизацию, добавление постов, просмотр карт и геолокацию.

# ГИБРИДНЫЕ МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ

* 1. Гибридные мобильные приложения

Гибридные мобильные приложения - это приложения, которые создаются с единой базой кода, позволяющей запускать их на нескольких операционных платформах и устанавливаются на устройство, как и любое другое приложение. Они являются одновременно нативными и веб-приложениями. По своей структуре такие приложения состоят из нативных элементов собственных приложений, разработанных для конкретной платформы, такой как iOS или Android, но включают в себя элементы веб-приложений, веб-сайтов, которые встроены в приложение, но не установлены на устройстве, таким образом они становятся доступными только с интернет-подключением и без него не работают [9].

Гибридные приложения развертываются в собственном контейнере, который использует мобильный объект WebView. При использовании, данный объект отображает веб-контент с помощью веб-технологий (CSS, JavaScript, HTML, HTML5). В сущности, это адаптированные веб-страницы, отображающиеся с помощью технологии WebView. Веб-контент может отображаться либо сразу после открытия приложения, либо только для определенных частей приложения [10]. Для доступа к аппаратным функциям устройства (акселерометр, камера, контакты ...) добавляются собственные элементы UI для каждой платформы, чтобы пользователь удобного взаимодействовал с приложением. Гибридные приложения также могут полагаться на платформы, которые предлагают API-интерфейсы JavaScript, если эти функции вызываются в WebView. По итогу, смысл гибридной разработки заключается в том, что разработка происходит на WebView мобильного телефона с помощью HTML5/Javascript/СSS, затем просто «пакуем» подобного рода разработку в нативную «обертку».

Родство гибридных приложений с веб-приложениями, позволяет просто и быстро вносить коррективы. Таким образом, разработчикам не приходится повторно размещать приложение в магазине ради устранения ошибок предыдущей версии, как это происходит в случае с нативными приложениями.



Рис.2.1. Гибридное приложение.

Преимущества гибридных приложений:

1. Стоимость и скорость создания приложения;
2. Количество задействованных в разработке людей;
3. Кроссплатформенность, позволяющая обратить внимание на продукт большего количества пользователей за счет привлечения людей с различными предпочтениями и возможностями;
4. Присутствует опция автономного обновления, что позволяет пользователям не проверять наличие обновлений вручную и значительно улучшает показатель UX.
5. Наличие наибольшей функциональности и персонализации для пользователя;
6. При разработке гибридного мобильного приложения есть возможность создавать его на нескольких платформах, а не ограничиваться одной, как в случае с нативными приложениями;
7. Гибриды - хорошая опция для разработчиков, которые создают визуально насыщенные приложения, например, игры (которые не будут хорошо работать в виде веб-приложений).

Выбор подходящей модели мобильного приложения - это очень важный этап в его разработке, на который влияют несколько факторов, таких как:

1. техническая оценка разработчиков;
2. потребность в доступе к информации на устройстве;
3. влияние скорости интернета на приложение;
4. является одноплатформенным или мультиплатформенны приложением.
5. некорректная работа при отсутствии интернет-соединения;
6. средняя скорость работы на фоне нативных;
7. минимализм в отношении визуальных элементов.

Процесс разработки гибридного приложения [11] соответствует процессу разработки веб-сервиса с дополнительным этапом разработки интерфейса обложки мобильного приложения. В результате владелец проекта имеет и сайт веб-сервиса, и мобильное приложение.

* 1. Сравнение гибридных, нативных и веб приложений

Широкая популярность гаджетов вполне закономерно приводит к тому, что на рынок приходят новые производители. А вместе с ними – их собственные разработки, которые находят себя в приложениях. Это так называемые нативные приложения, то есть доступные только на одной определенной платформе или операционной системе. Простой пример: площадка Android на своих использует в качестве стартового браузера сервисы Google, а тот же IOS – браузер под названием Safari. Такой подход касается не только браузера, но и многих других приложений, возможность установить которые ограничивается их «материнской» системой.

Еще недавно компании ориентировались только на опыт в разработке нативных приложений под Android и iOS, но сейчас лучшей заменой для них становятся кросс-платформенные решения, так как для бизнеса в приоритете находится разработка относительно простых и недорогих приложений за короткое время. Компании замотивированы на конкурентную борьбу в своей сфере бизнеса, поэтому они предпочитают приложения тактического, а не стратегического характера. Другими словами, в условиях высокой конкуренции компаниям важно быть первыми и максимально доступными в мобильных решениях для своих потребителей. Следуя этой логике, они расставляют приоритеты в пользу быстрого создания приложений.

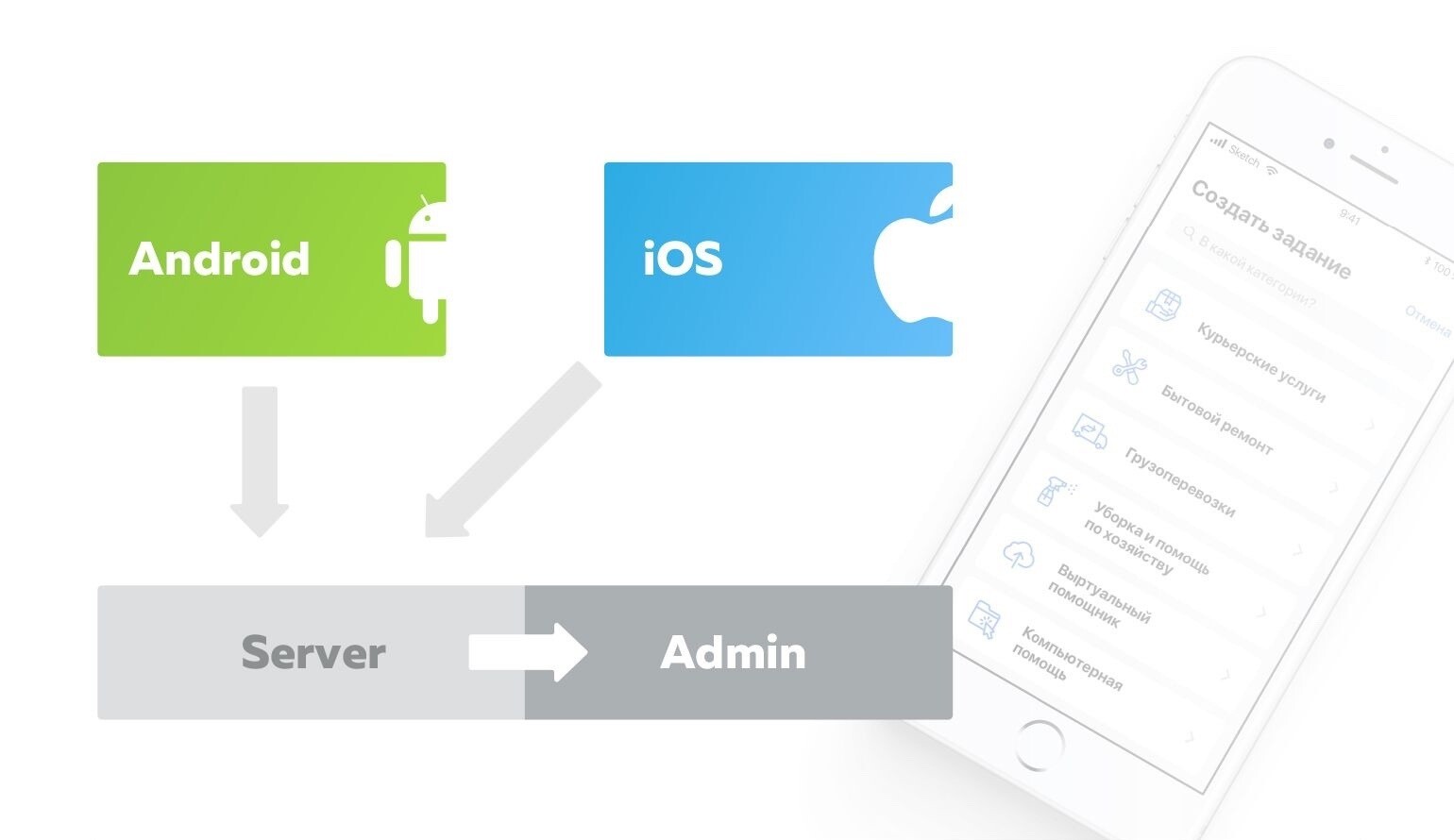
Фактически гибридные мобильные приложения это мобильные приложения с использованием веб-технологий. Обычный пользователь даже не замечает разницы, поскольку гибридное приложение устанавливается на устройство и выглядит в точности, как нативное. Тем не менее, разница существует, и это необходимо учитывать, при выборе подхода.

Гибридные приложения по своей сути более универсальны и чаще всего подходят для разных систем. Во многом, гибридное приложение – это своего рода синтез мобильной направленности и веб-приложений, а именно форм, доступных только для одной системы, но уже браузерной или компьютерной.

Нативные приложения загружаются через магазины приложений (App Store, Google Play или его аффилиаты, магазин приложений Windows и т.д.) и устанавливаются в ПО смартфона. Важным отличием является то, что нативные приложения разрабатываются специально под конкретную платформу (например, под iOS для iPhone, под Android для устройств под управлением ОС Android или под Windows для Windows Phone и т.д.) и требуют от разработчика специальных знаний и умений для работы в конкретной среде разработки (xCode для iPhone, eclipse, Android Studio или IntellijIdea для устройств на Android); более того, используется только «родные» языки программирования для написания таких приложений. Естественно, сам процесс при этом более трудоемкий. Таким образом, нативные приложения всегда «заточены» под конкретную ОС и органично выглядят на смартфоне. Такие приложения с легкостью могут использовать все функции ПО смартфона (камера, микрофон, акселерометр, геолокация, адресная книга, плеер и т.д.), и при этом более бережно расходуют ресурсы телефона (аккумулятор, память). В зависимости от назначения приложения предполагают или не предполагают наличие интернет-соединения

Веб-приложения не случайно называют html5-приложениями. Это, по сути, сайт, оптимизированный под смартфон. Пользовательский интерфейс создается при помощи стандартных веб-технологий. Их не нужно загружать из магазина приложений, но они могут находиться в специальных магазинах веб-приложений, которые есть у некоторых современных браузерах, например у Chrome. Веб-приложения используют для работы браузер телефона. Главной особенностью таких приложений является их кроссппатформенность — возможность работать на всех устройствах, без дополнительной адаптации.

Любой многопользовательский сервис состоит из клиентских приложений и серверной части. Если смотреть совсем широко, то туда ещё подключаются сайт, телефония, api сторонних проектов, но для того, чтобы сравнить натив с гибридом нам это не нужно, поэтому рассмотрим самый простой вариант:



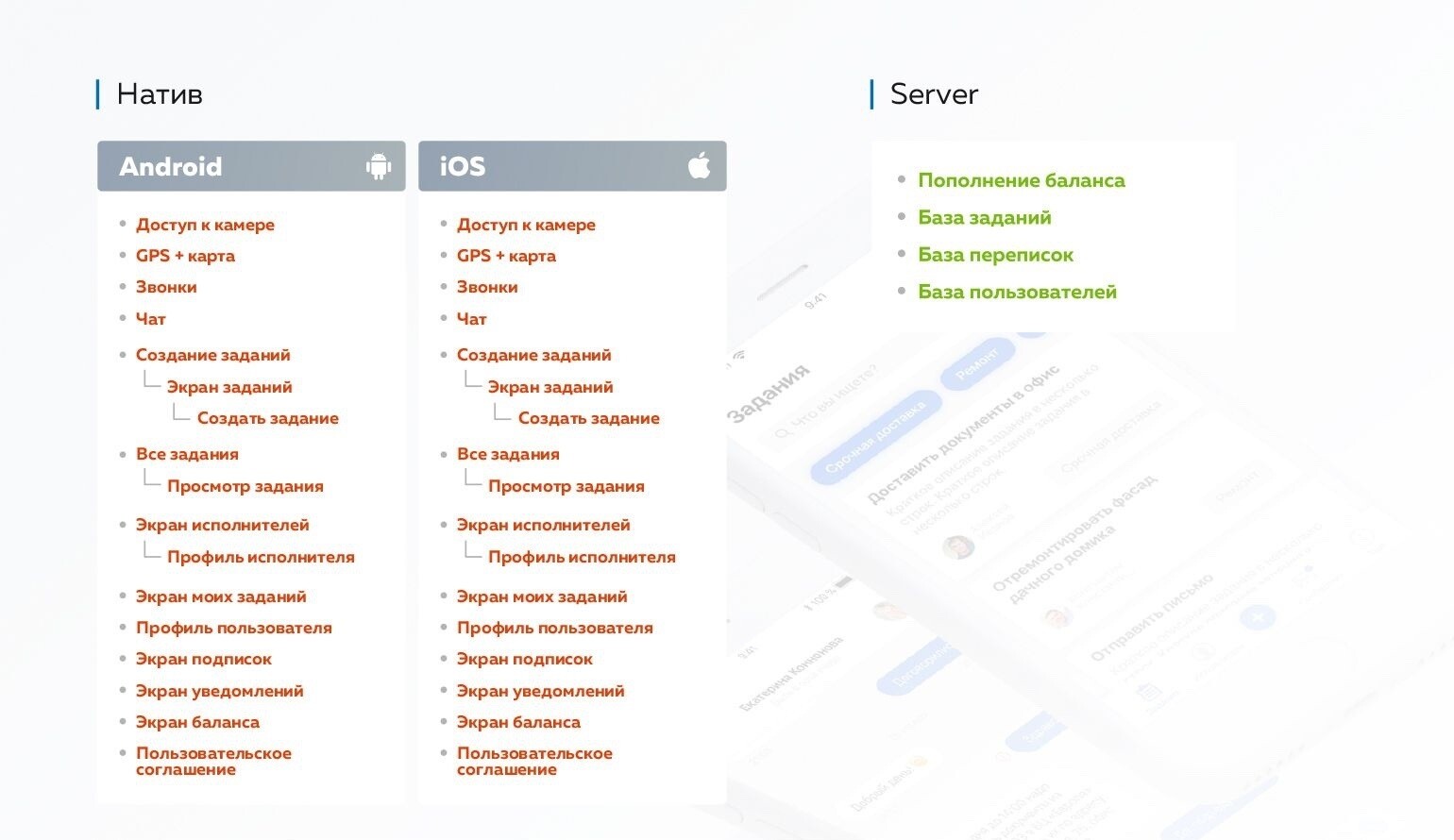
На стороне клиентской части располагается приложение iOS или Android, а со стороны сервера, собственно, серверное ПО и админка для управления данными.

Нативное приложение можно сравнить с приложениями для ПК. Купили MS Word, скачали установщик на компьютер, установили его и пользуетесь. По большому счёту вам уже без разницы есть интернет или нет - программа запускается с вашего ПК. Тут тоже самое - скачали со сторов, установили и весь функционал доступен с вашего телефона. То есть приложение скачивается "от и до" и, если это не заложено логикой приложения, интернет ей не нужен. Самый простой пример - это одиночные игры. Скачал и играешь.

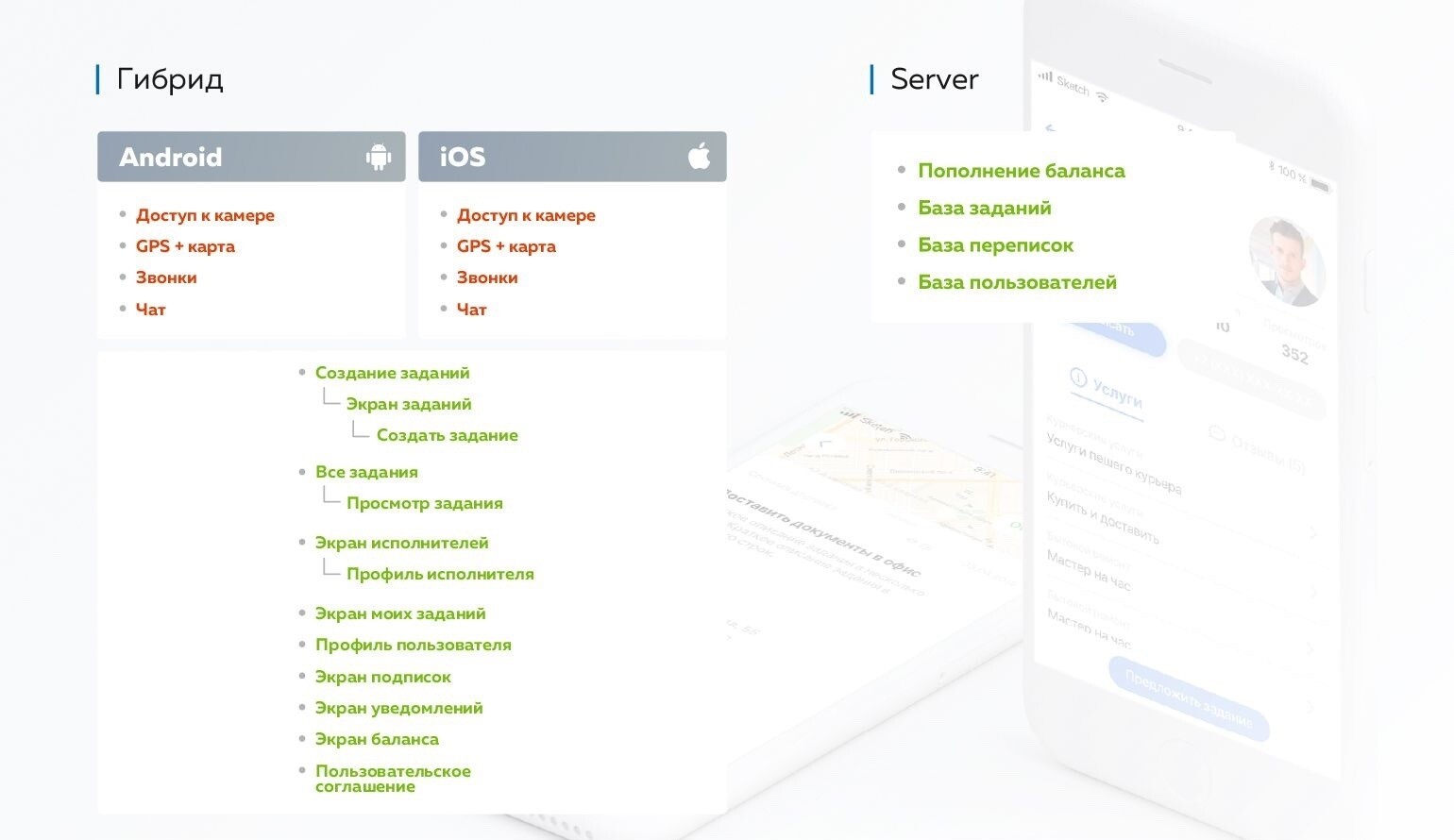
Гибрид - это скорее подход к программированию, чем какой-то особенный вид приложения. Его принцип заключается в том, что всё что можно (читай "все функции не связанные с железом") программируется на стороне сервера, а на стороне клиента остаётся только необходимый минимум. Если рассматривать типовой мобильный магазин, то GPS и функция отображения данных будет на стороне приложения, а каталог, карточки товаров и т.д. будут грузиться с сервера.

Давайте для примера рассмотрим как бы в гибриде и нативе выглядела наша платформа для создания сервиса поиска исполнителей "Сервис ПИ". Красным выделено то, что нужно делать на стороне приложения для телефона, а зелёным - на стороне сервера.

В нативе:



В гибриде:



Отсутствие требований к постоянному наличию интернета (приложения шагомеры, пульсомеры, игры-кликеры, органайзеры и т.д.)

Вообще, как я и писал выше, моя команда занимается разработкой гибридных приложений. Это и не случайно, ведь у бизнес-приложений нет требований ни к вычислительной мощности телефона (нагрузка не выше, чем у браузера), ни к отсутствию интернета (для того, чтобы сделать заказ в мобильном магазине в любом случае нужен интернет). Если же клиент приходит к нам с заказом игры, то, конечно же, мы отправляем его к нативщикам. Самое важное, что я рекомендую - это грамотно выбрать подход к разработке, чтобы потом не было мучительно больно за ошибки, сделанные на старте.

Как видно из картинок единственная разница между нативом и гибридом - это объём функционала, который реализуется на стороне телефона. Но есть большое "НО". Весь "зелёный" функционал, который мы в гибриде перенесли на сервер делается 1 раз для обеих платформ, а красный - для каждой платформы пишется отдельно. Давайте разберём какие из этого можно сделать финансовые выводы.

Таблица 2. Преимущества и недостатки гибридного, веб и нативного приложения

|  |  |  |
| --- | --- | --- |
| **Преимущества** | **Недостатки** | |
| **Нативное** | * Максимальная функциональность и скорость работы * Не требуется интернет- соединение для использования * Имеет доступ к ПО смартфона (GPS, плеер, камера) Распространение через магазины приложений | * Выше стоимость и длиннее сроки разработки * Требует от разработчика знаний определенной среды программирования * Работает только с одной платформой * При косметических изменениях необходимо выпускать обновление |
| **Веб (HTML5)** | * Кроссплатформенность * Не требует загрузки из магазина мобильных приложений * Можно легко адаптировать обычный сайт * Легче найти веб-разработчика нежели разработчика под определенную платформу Простота создания и поддержки | * Требует подключения к интернету * Не имеет доступа к ПО смартфона * Не может отправлять push-уведомления * Должен быть запущен интернет-браузер * При продаже требуется использование своей платежной системы |
| **Гибридное** | * Функциональность нативного приложения на независимой платформе * Запускается не из браузера в отличии от веб приложения * Возможность независимого обновления * Распространение через магазины приложений | * Загружается из магазина мобильных приложений (необходимо соответствовать требованиям) * Разработчик должен быть знаком с разными API |

Таблица 1

Если для работы приложения необходимо использовать мощности устройства, о которых мы писали ранее, или для его работы очень важна скорость обработки информации (игры, социальные сети, геолокационные сервисы, сервисы обмена фотографиями и т.д.), то необходимо делать нативное приложение. Если скорость работы не так важна, то делайте лучше гибридное приложение. А если вам просто важно, чтобы у пользователя была возможность получить от вас необходимую информацию при наличии интернета через мобильный телефон, и его это устраивает, то спокойно делайте веб-приложение!

Часто производители делают нативные приложения платными. И если вы планируете последовать этому плану, помните, что люди в принципе не очень любят платить деньги за какой-либо софт, пускай и качественный. В то время как разработка качественного приложения под конкретную платформу действительно стоит денег. Так что придется хорошо продумать процесс монетизации, а лучше его протестировать перед официальным запуском.

Учитывая свои нюансы разработки нативного приложения, акцентируют внимание на применяемом языке программирование. Например, для платформы IOS актуально и правильно применить Objective-c (реже встречается использование Swift).

Понимая особенность реализованного во время разработки приложения, можно быстро определиться и решить для себя гибридное или нативное приложение будет приносить пользу. При этом учитывается сразу несколько факторов: стоимость самой разработки, преследуемая цель, необходимость работы на нескольких платформах и, конечно же, целевая аудитория.

Плюсы и минусы гибридного подхода

По большему счёту, к основным достоинствам гибридного подхода относятся:

1. Кросс-платформенность. Сделав одно приложение, можно экспортировать его под любую операционную систему – iOS, Android, Windows Phone, BlackBerry.
2. Доступная стоимость разработки – в разы меньше, чем при нативном подходе.
3. Использование одного языка – JavaScript.
4. Доступ к основным данным мобильного устройства: GPS, камера, телефонная книга и т.д.
5. Работа в оффлайн-режиме.
6. Возможность распространять приложение через официальные магазины приложений.
7. Быстрый вывод на рынок: гибридные приложения разрабатываются быстрее, так как в ходе процесса используются стандартные веб-технологии, которые легко обслуживать в долгосрочной перспективе.
8. Дистрибуция для нескольких платформ: этот вид приложений распространяется через оба магазина, что позволяет охватить большее число пользователей.

Однако у гибридной разработки есть свои слабые места:

1. Сниженная скорость работы в отличие от нативных приложений.
2. Меньше возможностей для интеграции с «начинкой» мобильного устройства.
3. Проблематичная верстка адаптивного дизайна. Несмотря на наличие различных веб-фреймворков для построения приложений, мы столкнулись с тем, что все они по-разному ведут себя на различных девайсах.
4. Интеграция сторонних сервисов: Вы не можете разрабатывать гибридное приложение на одном только JavaScript. Вам потребуется интегрировать такие фреймворки для гибридной разработки, как Cordove, Ionic или React Native, каждый из которых требует определенных усилий для освоения.

Появление ряда багов, которые могут приводить к определенным нарушениям нормальной работы приложения. Во время тестирования, исходя из различного поведения приложения на разных устройствах, правка одного бага влечет за собой появление других под новыми версиями операционных систем.

Ограниченное представление визуальных и графических элементов, в частности, анимации.

Таким образом, использовать гибридный подход можно в тех случаях, когда:

1. Стоит задача быстрой разработки приложения под более, чем 2 мобильные платформы;
2. Есть необходимость сэкономить на бюджете;
3. Требуется относительно простое приложение без сложной анимации;
4. Можно исключить поддержку множества параметров нативного функционала.
   1. Примеры

Гибридная разработка пользуется популярностью у крупных мировых компаний. Множество широко известных приложений являются кросс-платформенными. Среди них: Wikipedia, Health tap, BBC Sport App и Evernote. Пользователи скачали каждое из этих приложений более миллиона раз, что говорит о высоком качестве разработанных решений.

Приложение HeartCamera для iOS, позволяющее украсить фотографию рисованными сердцами и т.п.

1. Загружается из магазина;
2. Использует камеру телефона;
3. Необходимо подключение к интернету при желании поделиться результатом своей работы;
4. Можно настроить push-уведомления.

«Heart Camera» — приложение, которое адаптировано для IOS. Его применяют с целью украсить и «оживить» фотоснимки. Здесь есть много фильтров, наклеек и других дополнений, которые пользуются спросом у любителей мобильной обработки. Загрузить программу можно быстро, используя магазин. Для полноценной работы применяется камера мобильного телефона, а в качестве дополнения доступна настройка push-уведомлений. Если есть желание поделиться результатами обработки в социальных сетях или мессенджерах, то можно подключить Интернет для рассылки изображения (остальной функционал работает в офлайн режиме).

Приложение TripCase — органайзер для планирования путешествий.

1. Загружается из магазина;
2. Может использовать геолокацию;
3. Необходимо подключение к интернету;
4. Может использовать сотовую сеть;
5. Можно настроить push-уведомления

«TripCase» — это уникальное гибридные приложение для мобильных устройств, которое предоставляет доступ к планированию своих путешествий. Доступно множество функций для быстрого сбора вещей и даже заказа билетов. Загружается стандартно, используя markets. Для работы нужно проверить стабильное подключение к Интернету. При желании настраиваются push-уведомления. Также стоит выделить важность доступа к таким функциям телефона, как сотовая связь и геолокация.

**Это в 3 главу**

Разработка серверной части - один из самых трудных этапов создания приложения. Зачастую, при планировании разработки проекта недооценивается необходимый объем ресурсов и время создания бэкенда (Рис.1.1.). Еще одна проблема - ограниченность доступных команде ресурсов. Чаще всего разрабатывать бэкенд приходится с помощью тех инструментов и технологий, которыми владеют члены команды. Процесс получается длительным, а само приложение — достаточно сложным и дорогим с точки зрения сопровождения. А длительная разработка бэкенда на даже относительно простых проектах ведет к увеличению расходов и другим рискам: расход денег без визуально видимого результата.

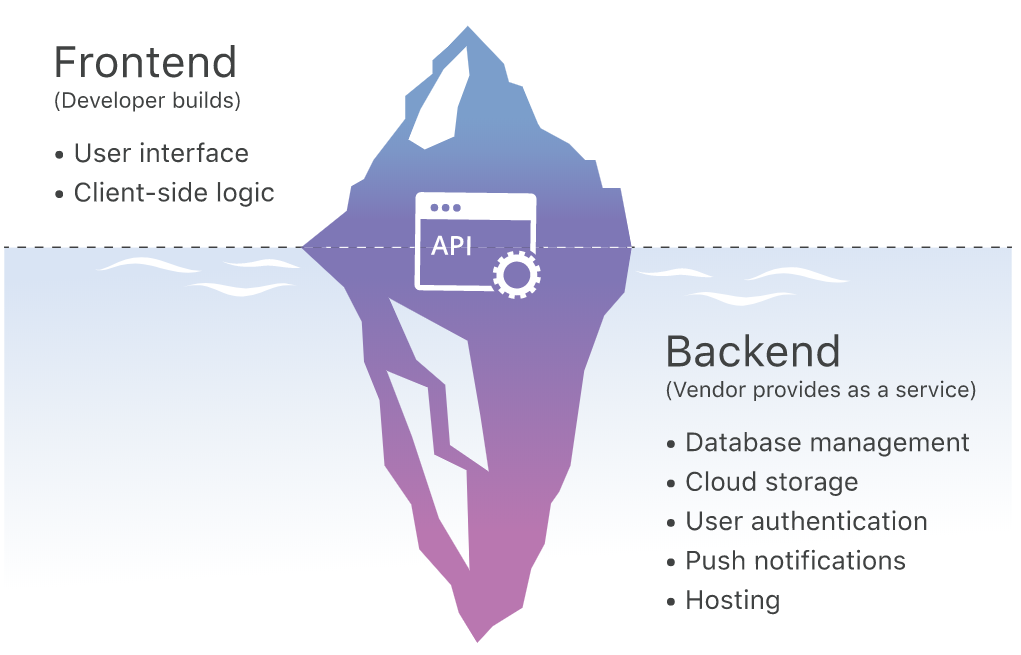


Рис.1.1. Принцип работы современного приложения.

Решить эту проблему возможно с помощью **Backend as a Service (BaaS**) - готовых серверных сервисов, собрав которые вместе, получается необходимый универсальный бэкенд для любого проекта. BaaS является хорошим решением, избавляющем от существенных затрат, а именно написания и поддержки серверного бэкенда. BaaS дает возможность реализовать простой сервис силами одного мобильного разработчика [1]. Такое решение будет особо уместно для небольших стартапов, которые сначала хотят проверить бизнес идею на её работоспособность, не планируя очень сложной логики и взаимодействия со сторонними сервисами — хотя с BaaS такое тоже возможно реализовать.

Преимущества бэкенда на BaaS, в основном, заключаются в том что, используя BaaS, разработчики способны быстро построить необходимый бэкенд и платформу для обработки данных, поступающих из мобильных приложений.

BaaS позволяет избегать разработчикам необходимости иметь дело с:

* физическим сервером приложения;
* базой данных;
* клиент-серверной библиотекой;
* написанием админки;
* дизайном своего API;
* хостингом.

BaaS-функции включают в себя облачное хранилище, push-уведомления, управление пользователями и файлами, службы определения местоположения и др [2]. Все эти сервисы имеют собственный API, чтобы легко встраиваться в приложения.

* 1. Один из многофункциональных сервисов гугл - Firebase

Firebase помогает быстро создавать качественные приложения, увеличивать аудиторию вовлеченных пользователей и повышать доходы. Платформа содержит множество полезных функций для приложения, в том числе серверный код для мобильных сервисов, статистику, а также инструменты для монетизации и расширения аудитории.

* + 1. Аутентификация в Firebase

Большинство приложений должны идентифицировать личность пользователя. Знание личности пользователя позволяет приложению безопасно сохранять пользовательские данные в облаке и обеспечивать одинаковую персонализированную работу на всех устройствах пользователя.

Аутентификация Firebase предоставляет бэкэнд-сервисы, простые в использовании SDK и готовые библиотеки пользовательского интерфейса для аутентификации пользователей в приложении. Он поддерживает аутентификацию с использованием паролей, телефонных номеров, популярных провайдеров федеративной идентификации, таких как Google, Facebook и Twitter, и многого другого. Аутентификация Firebase тесно интегрируется с другими сервисами Firebase и использует отраслевые стандарты, такие как OAuth 2.0 и OpenID Connect, поэтому ее можно легко интегрировать с пользовательским бэкэндом.

В приложение с использованием Firebase возможно войти либо с помощью FirebaseUI в качестве полного решения для проверки подлинности, либо с помощью пакета Firebase Authentication SDK для ручной интеграции одного или нескольких методов входа в приложение (Рис.1.2.).

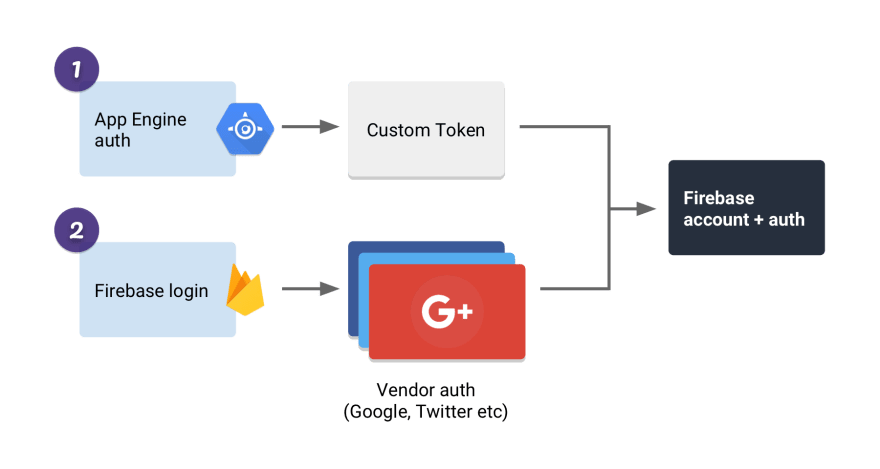


Рис.1.2. Принцип работы аутентификации*.*

Firebase предоставляет следующие возможности по аутентификации:

* **Аутентификация по электронной почте и паролю;**
* **Интеграция с федеративным поставщиком удостоверений;**
* **Проверка подлинности номера телефона;**
* **Кастомная аутентификация системной интеграции;**
* **Анонимная авторизация.**

Также стоит отметить, что по умолчанию аутентифицированные пользователи могут читать и записывать данные в базу данных Firebase Realtime и облачное хранилище [3].

* + 1. Firebase Хостинг

Firebase Hosting - это хостинг для веб-сайтов промышленного уровня для разработчиков. С помощью одной команды появляется возможность быстро развернуть веб-приложения и предоставлять как статический, так и динамический контент в глобальную CDN (сеть доставки контента). Также можно соединить Firebase Hosting с Cloud Functions или Cloud Run для создания и размещения микросервисов в Firebase.

Firebase Hosting создан для современного веб-разработчика. Веб-сайты и приложения стали более мощными, чем когда-либо, благодаря появлению интерфейсных JavaScript-фреймворков, таких как Angular и статических генераторов, таких как Jekyll. Он предоставляет инфраструктуру, функции и инструменты, адаптированные для развертывания веб-сайтов, приложений и управления ими не зависимо от того, простая ли это целевая страница приложения либо сложное Progressive Web App (PWA).

Ключевые возможности:

* Передача контента через безопасное соединение;
* Статический и динамический контент + микросервисы;
* Быстрая доставка контента;
* Развертывание новых версий одной командой;
* Откат одним кликом.

Firebase Hosting обеспечивает быстрый и безопасный хостинг для веб-приложения, статического и динамического контента, микросервисов.

* + 1. Cloud Storage

Облачное хранилище для Firebase - это мощный, простой и экономически эффективный сервис хранения объектов, созданный для масштаба Google. Пакеты Firebase SDK для облачного хранилища повышают безопасность Google при загрузке и выгрузке файлов для приложений на основе Firebase, независимо от качества сети [4]. Помимо этого, доступны такие инструменты, как SDK для хранения изображений, аудио, видео или другого пользовательского контента. На сервере существует возможность использовать Google Cloud Storage , чтобы получить доступ к тем же файлам.

Ключевые возможности:

* Надежные операции;
* Высокий уровень безопасности;
* Высокая масштабируемость.

Разработчики используют Firebase SDK для облачного хранилища, чтобы загружать и скачивать файлы непосредственно с клиентов [5]. Если сетевое соединение плохое, клиент может повторить операцию прямо с того места, где он был прерван, что экономит время пользователей и пропускную способность.

* + 1. База данных реального времени Firebase

База данных Firebase Realtime представляет собой облачную базу данных [6]. Данные хранятся в формате JSON и синхронизируются в режиме реального времени с каждым подключенным клиентом. При создании кроссплатформенного приложения с помощью набора средств разработки SDK для iOS, Android и JavaScript, все клиенты совместно используют один экземпляр базы данных в реальном времени и автоматически получают обновления с самыми новыми данными.

База данных Firebase Realtime позволяет создавать многофункциональные приложения для совместной работы, обеспечивая безопасный доступ к базе данных непосредственно из кода на стороне клиента (Рис.1.3.).



Рис.1.3. Совместная работа устройств в реальном времени*.*

Данные сохраняются локально, и даже в автономном режиме события в реальном времени продолжают срабатывать, предоставляя конечному пользователю доступ к приложению. Когда устройство восстанавливает соединение, база данных реального времени синхронизирует локальные изменения данных с удаленными обновлениями, которые произошли, когда клиент находился в автономном режиме, автоматически объединяя любые несоответствия.

Ключевые возможности:

* Режим реального времени;
* Доступность в оффайн режиме;
* Доступно с клиентских устройств;
* Масштабирование по нескольким базам данных.

База данных реального времени является базой данных NoSQL и, как таковая, имеет различные оптимизации и функциональность по сравнению с реляционной базой данных.

* 1. Google OAuth

OAuth 2 представляет собой фреймворк для авторизации, позволяющий приложениям осуществлять ограниченный доступ к пользовательским аккаунтам на HTTP сервисах, например, на Facebook, GitHub и DigitalOcean. Он работает по принципу делегирования аутентификации пользователя сервису, на котором находится аккаунт пользователя, позволяя стороннему приложению получать доступ к аккаунту пользователя. OAuth 2 работает на web, desktop и мобильных приложениях [7].

OAuth выделяет четыре роли:

* Владелец ресурса;
* Клиент;
* Сервер ресурсов;
* Авторизационный сервер.

Владельцем ресурса является пользователь, который авторизует приложение для доступа к своему аккаунту. Доступ приложения к пользовательскому аккаунту ограничен “областью видимости” (scope) предоставленных прав авторизации (например, доступ на чтение или запись).

Сервер ресурсов непосредственно хранит защищённые данные аккаунтов пользователей, а авторизационный сервер проверяет подлинность информации, предоставленной пользователем, а затем создаёт авторизационные токены для приложения, с помощью которых приложение будет осуществлять доступ к пользовательским данным.

С точки зрения разработчика приложения API сервиса одновременно выполняет и роль сервера ресурсов и роль сервера авторизации. Далее мы будем считать эти две роли одной, и называть её Сервис или API.

Клиентом является приложение, которое хочет осуществить доступ к аккаунту пользователя. Перед осуществлением доступа приложение должно быть авторизовано пользователем, а авторизация должна быть одобрена со стороны API.

1.2.1. Абстрактное описание протокола

В данной диаграмме (Рис.1.4.) представлены роли, используемые в OAuth, и их взаимодействия друг с другом.



Рис.1.4. Диаграмма Взаимодействия ролей в OAuth.

Последовательность шагов на этой диаграмме:

1. Приложение запрашивает у пользователя авторизацию на доступ к серверу ресурсов;
2. Если пользователь авторизует запрос, приложение получает разрешение на авторизацию (authorization grant);
3. Приложение запрашивает авторизационный токен у сервера авторизации (API) путём предоставления информации о самом себе и разрешении на авторизацию от пользователя;
4. Если подлинность приложения подтверждена и разрешение на авторизацию действительно, сервер авторизации (API) создаёт токен доступа для приложения. Процесс авторизации завершён;
5. Приложение запрашивает ресурс у сервера ресурсов (API), предоставляя при этом токен доступа для аутентификации;
6. Если токен действителен, сервер ресурсов (API) предоставляет запрашиваемый ресурс приложению.

Фактический порядок шагов описанного процесса может отличаться в зависимости от используемого типа разрешения на авторизацию, но в целом процесс будет выглядеть описанным образом.

1.2.2. Регистрация приложения

Перед применением OAuth в приложении, необходимо зарегистрировать его на сервисе. Это делается путём регистрации в разделе “developer” или “API” сайта сервиса, где предоставляется следующая информация:

* Название приложения;
* Сайт приложения;
* Redirect URL или callback URL.

Redirect URL - это URL, на который сервис будет перенаправлять пользователя после авторизации (или отказа в авторизации) вашего приложения.

1.2.3. Идентификатор клиента и секрет клиента

После регистрации приложения сервис создаст учётные данные клиента - идентификатор клиента (client ID) и секрет клиента (client secret). Идентификатор клиента представляет собой публично доступную строку, которая используется API сервиса для идентификации приложения, а также используется для создания авторизационных URL для пользователей. Секрет клиента используется для аутентификации подлинности приложения для API сервиса, когда приложение запрашивает доступ к аккаунту пользователя. Секрет клиента должен быть известен только приложению и API.

* 1. Сервис Google Map

Google Maps – это комплекс приложений, созданных на базе бесплатного сервиса картографии и технологии, используемой Google. Данное приложение используется для поиска информации на карте с отметками достопримечательностей, организаций и т.д [8].

Оформление и возможности Google Maps

Сервис предлагает такие режимы отображения, как:

«Спутник» – показ спутниковых и аэрофотографий (сюда относятся «Имена объектов» – отображение сведений на схемах. До этого режим назывался «Гибрид»);

«Карта» – показ данных картографии;

«Ландшафт» – схемы, отображающие массивные элементы поверхности, например леса или холмы (ложный 3D-рельеф);

«Просмотр улиц» – режим, позволяющий ориентироваться по панорамным фотоснимкам на фоне городских улиц;

«Личные схемы» – персональный режим, благодаря которому можно нанести на карту требуемое содержимое.

Благодаря Google Maps пользователь может:

* просматривать интересные локации на карте;
* открывать файлы с расширением KML и KMZ, а также сохранять нужные метки в этих форматах;
* прокладывать маршрут;
* вносить персональный текст, картинки и видеоролики;
* отправлять созданную схему друзьям.

Некоторые режимы и функции:

Режим Google Transit дает подробные рекомендации по проезду между двумя указанными точками на доступном общественном транспорте и приблизительно подсчитывает, сколько это займет времени. Данная опция доступна для всей России.

Пользователь может отправить на почтовый ящик ссылку со своим местоположением либо разместить HTML-код карты Google на веб-ресурсе. Также в Google Maps имеется инструмент генерирования карты для последующей распечатки.

Встроены результаты поиска информации по организациям – поиск адресов компаний и их контактных данных в выбранном районе. К примеру, если написать запрос «суши в районе Тверская улица», на локальной системе координат обозначатся точки расположения ближайших заведений, их адреса, режимы работы. Также можно посмотреть дополнительные сведения, такие как формы оплаты, рейтинг, отзывы клиентов и др.

В картах доступна функция Map Maker (ранее это был отдельный сервис), позволяющая изменить содержание собственной информацией.

Значение Google Maps для SEO-продвижения

Сервис Google Maps полностью бесплатный. Но при этом существует ограничение: в локальном поиске отображается только 7 компаний, предоставляющих схожие товары/услуги.

Факторы ранжирования в «Картах Google»:

* полнота информации о компании, представленной при регистрации в «Google Мой Бизнес»;
* упоминание поисковых запросов и категории в имени компании;
* наличие веб-сайта, на котором прописан фактический адрес в тегах <adress> </adress>;
* положительные отзывы;
* интересная информация в заметках;
* подключение контекстной рекламы AdWords;
* присутствие на Google Earth.
  1. Итог по первой главе

Авторизация с помощью данного гугл сервиса дает большое количество преимуществ разработчикам, так же это простой способ аналога регистрации для пользователя, ведь им всего лишь нужно аутентифицироваться с помощью данных своего гугл аккаунта для первого и последующих входов в приложение, нет необходимости в заполнении какими-либо данными профиля, ведь все необходимое предоставляет гугл. Что касается разработчиков, такой подход авторизации снимает с них ответственность за хранимые данные в своей базе данных, таких как пароль или других данных [9]. Так же этот способ экономит много времени, которое могло быть потрачено на создание, построении защиты данных, которые должны храниться у разработчика, а также, устранять возможности обхода, обмана системы, кражи данных и так далее.

# ФРЕЙМВОРКИ JAVASCRIPT

* 1. Фреймворк VUE.js

Vue - это прогрессивный фреймворк для создания пользовательских интерфейсов Web с использованием шаблона архитектуры MVVM (Model-View-ViewModel) [12]. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами. Он работает с уровнем представления. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его совместно с современными инструментами и дополнительными библиотеками. Двумя основными конкурентами Vue являются React и Angular. Вместе с ними Vue закрывает почти 100% рынка Web-разработки.

Поскольку Vue работает только на «уровне представления» и не используется для промежуточного программного обеспечения и бэкэнда, он может легко интегрироваться с другими проектами и библиотеками, в том числе с Vuetify [13]. Vue.js содержит широкую функциональность для уровня представлений и может использоваться для создания мощных одностраничных веб-приложений.

Функции Vue.js:

* Реактивные интерфейсы;
* Декларативный рендеринг;
* Связывание данных;
* Директивы (все директивы имеют префикс «V-». В директиву передается значение состояния, а в качестве аргументов используются html атрибуты или Vue JS события);
* Логика шаблонов;
* Компоненты;
* Обработка событий;
* Свойства;
* Переходы и анимация CSS;
* Фильтры.

Основная библиотека Vue.js 2 очень маленькая (всего 17 кБ). Это гарантирует, что нагрузка на проект, реализованный с помощью Vue.js, минимальна, а сайт будет быстро загружаться.

Vue подходит для небольших проектов, которым необходимо добавить немного реактивности, представить форму с помощью AJAX, отобразить значения при вводе данных пользователем, авторизацию или другие аналогичные задачи. Vue легко масштабируется и хорошо подходит для объемных проектов, поэтому его называют прогрессивным фреймворком [14].

Vue также отлично подходит для крупных одностраничных приложений благодаря своим основным компонентам, таким как Router и Vuex. С Vue можно как использовать общедоступные API для создания приложений, так и реализовывать выполняемые сервером приложения. Но Vue лучше всего подходит для разработки решений, которые используют внешние API для обработки данных.

С помощью Vue также можно создавать frontend блога на популярных CMS. Vue.js отлично подходит и для разработки динамических интерфейсов, которые адаптируются под пользователя.

Vue.js: особенности, применение и отличия от других Javascript фреймворков

Релиз фреймворка Vue.js 2.0 состоялся еще в 2016 году, но многие владельцы технологических компаний и Javascript разработчики все еще не решаются использовать его в своих проектах. В этой статье мы рассмотрим ключевые моменты данного решения.

По Vue JS отзывы довольно положительные. Согласно опросу компании Monterail, 81% разработчиков отмечают простоту интеграции как основное преимущество фреймворка Vue, в т. ч. и интеграцию с backend фреймворками. Большинство специалистов считает, что освоить Vue намного легче чем другие популярные Javascript фреймворки. Документация — еще одна сильная сторона Vue — так считают 60% опрошенных разработчиков. Аналогичное число респондентов (56%) отмечает производительность данного решения как одну из его самых сильных сторон. Разберемся подробнее, что же такое Vue.js?

Описание фреймворка

Vue.js — это JavaScript библиотека для создания веб-интерфейсов с использованием шаблона архитектуры MVVM (Model-View-ViewModel).

Поскольку Vue работает только на «уровне представления» и не используется для промежуточного программного обеспечения и бэкэнда, он может легко интегрироваться с другими проектами и библиотеками. Vue.js содержит широкую функциональность для уровня представлений и может использоваться для создания мощных одностраничных веб-приложений.

Функции Vue.js:

Реактивные интерфейсы;

Декларативный рендеринг;

Связывание данных;

Директивы (все директивы имеют префикс «V-». В директиву передается значение состояния, а в качестве аргументов используются html атрибуты или Vue JS события);

Логика шаблонов;

Компоненты;

Обработка событий;

Свойства;

Переходы и анимация CSS;

Фильтры.

Основная библиотека Vue.js 2 очень маленькая (всего 17 кБ). Это гарантирует, что нагрузка на ваш проект, реализованный с помощью Vue.js, минимальна, а ваш сайт будет быстро загружаться. Скачать соответствующий .js файл можно по ссылке.

Где применяется Vue.js

Vue подходит для небольших проектов, которым необходимо добавить немного реактивности, представить форму с помощью AJAX, отобразить значения при вводе данных пользователем, авторизация или другие аналогичные задачи. Vue легко масштабируется и хорошо подходит для объемных проектов, поэтому его называют прогрессивным фреймворком.

Vue также отлично подходит для крупных одностраничных приложений благодаря своим основным компонентам, таким как Router и Vuex. С Vue можно как использовать общедоступные API для создания приложений, так и реализовывать выполняемые сервером приложения. Но Vue лучше всего подходит для разработки решений, которые используют внешние API для обработки данных.

С помощью Vue также можно создавать frontend блога на популярных CMS. Vue.js отлично подходит и для разработки динамических интерфейсов, которые адаптируются под пользователя.

Vue.js: примеры лучших проектов

Рассмотрим некоторые крупные проекты, основанные на фреймворке Vue. Отметим сложности, с которыми столкнулись их разработчики, и что им дал переход на Vue.js.

Chess.com

Chess.com — самый посещаемый сайт игры в шахматы в мире с более чем 19 миллионами пользователей. Ресурс содержит раздел новостей, блог, сообщества, уроки, головоломки и возможность играть в шахматы в режиме реального времени.

Его legacy-код был реализован в PHP и Angular 1, поэтому возникали определенные сложности при работе с таким кодом. Команда проекта перешла с AngularJS на Vue. Благодаря простоте и быстродействию Vue.js, основателю Chess.com стало легче сотрудничать с его дистанционной командой, а на создание нового функционала ушло совсем немного времени.

Codeship

Codeship — это платформа для непрерывной интеграции, основанная на облачной технологии. Она позволяет хранить веб-приложения в облаке. Эту платформу используют такие гиганты, как Red Bull, CNN и Product Hunt.

Прежде чем Codeship перешел с jQuery на Vue, их пользователи постоянно сталкивались с зависанием и сбоями при работе с веб-приложением. У них был длинный список пользователей которые были недовольны работой приложения. Их история является отличным примером того, как Vue может помочь создать программное обеспечение с надежным и простым в обслуживании код.

Vue.js помог команде Codeship правильно организовать свой код и улучшить пользовательский интерфейс.

Livestorm

Livestorm — это веб-приложение для создания вебинаров. Оно помогает таким компаниям, как Workable, Pipedrive и Instapage продавать свои услуги или обучать клиентов с помощью вебинаров.

Livestorm — это пример Vue JS приложения,созданного с нуля. Благодаря Vue и его повторно используемым компонентам, их процесс разработки был быстрым и довольно простым.

Чем Vue.js отличается от Angular и React

Рассмотрим основные отличия фреймворка Vue от Angular и React.

Компоненты

Все эти фреймворки основаны на компонентах. При этом необходимо отметить, что React и Vue хорошо подходят для обработки так называемых немых компонентов — небольших, не имеющих состояния функций, которые получают входные данные и возвращают элементы в качестве вывода. Vue.js компоненты не имеют особых требований к своим именам, но рекомендуется придерживаться правил W3C для кастомных компонентов — использовать буквы нижнего регистра и разделения через дефис.

Фреймворк против библиотеки

Angular — это фреймворк, а не библиотека, так как содержит четкие указания по структуре приложения, а также имеет широкую функциональность. Angular является полноценным решением для enterprise-приложений, которое не требует от вас анализа каких-либо других библиотек или применения дополнительных инструментов. React и Vue, с другой стороны, универсальны. Их библиотеки могут быть сопряжены со всеми типами пакетов, хотя у Vue их немного, так как он еще довольно молод.

Гибкость использования

Вы можете работать с React или Vue, просто добавив библиотеку Javascript в исходный код. Это невозможно в случае с Angular, так как он предназначен для более сложных задач. Когда же речь идет о микросервисах и микроприложениях, React и Vue предоставляют больше контроля над размером приложений, позволяя выбирать только те элементы, которые необходимы в конкретных случаях. Также они предлагают большую гибкость для перехода от одностраничных приложений к микросервисам, позволяя использовать части прежнего приложения. А Angular, благодаря своему широкому функционалу, лучше всего подходит для разработки самих одностраничных приложений.

Быстродействие и размер файлов

Фреймворк Angular довольно объемен. Из-за своего широкого функционала, размер архивированного файла составляет около 143k, по сравнению с более простыми Vue и React с 23K и 43k соответственно.

React и Vue имеют Virtual DOM (document object model), который создает копию объектного представления структурного документа и позволяет работать с визуальной копией, а не с самим представлением. Этот подход помогает повысить производительность фреймворков и таким образом ваше приложение будет работать быстрее. Vue, в частности, обладает отличной производительностью и глубоким распределением памяти, но все эти фреймворки в основном схожи по своим характеристикам.

В качестве вывода необходимо отметить, что основными преимуществами Vue JS является его простота и легкость изучения. Вместо того, чтобы изучать сложную терминологию и инструменты для создания простого приложения, как в случае с React, вы можете сразу приступать к разработке. Это делает Vue JS идеальным выбором для стартапов или любой команды разработчиков, которая хочет быстро создавать высокопроизводительные веб-приложения с легко читаемым кодом.

2.2. Vuetify - Framework для создания гибридных мобильных приложений

Vuetify является библиотекой #1 для Vue.js и активно разрабатывается с 2016 года [15]. Цель проекта - предоставить пользователям все, что необходимо для создания многофункциональных, красивых и интересных веб-приложений, используя спецификацию Material Design. Это возможно благодаря постоянному циклу обновлений, долгосрочной поддержке предыдущих версий, отзывчивому участию сообщества, обширной экосистеме ресурсов и приверженности качественным компонентам (Tаб.1.).

Таб.1. Сравнение UI фреимворков

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Функции | Vuetify | Bootstrap | Buefy | Element UI | Quasar Framework |
| Full Accessibility and Section 508 support | + | + | + |  |  |
| Server Side Rendering support | + | + | + | + | + |
| Long-term Support | + |  |  |  |  |
| Release cadence\*\* | Weekly | Bi-Weekly | Bi-Monthly | Bi-Weekly | Bi-Weekly |
| Treeshaking | Automatic | Manual | Manual | Manual | Automatic |
| RTL support | + | + |  | + | + |
| Premium Themes | + | + |  |  |  |
| Business and Enterprise support | + |  |  |  |  |

Библиотека Vuetifyjs получила более 7 тысяч звёзд на GitHub. Она даёт разработчикам возможности по созданию пользовательских интерфейсов с использованием принципов Google Material Design. В её релизе V 1.0 Alpha можно найти более 80 компонентов, подходящих для повторного использования и спроектированных с применением семантических принципов, основанных на простых и чётких именах свойств, которые легко запоминать и использовать [16].

* 1. Фреймворк React

React — это самая популярная в мире JavaScript-библиотека. Но эта библиотека не потому хороша, что популярна, а потому популярна, что хороша. Большинство существующих вводных руководств по React начинается с примеров того, как пользоваться этой библиотекой. Но эти руководства ничего не говорят о том, почему стоит выбрать именно React.

У такого подхода есть свои сильные стороны. Если кто-то стремится к тому, чтобы, осваивая React, тут же приступить к практике, ему достаточно заглянуть в официальную документацию и взяться за дело.

Этот материал (вот, если интересно, его видеоверсия) написан для тех, кто хочет найти ответ на следующие вопросы: «Почему React? Почему React работает именно так? С какой целью API React устроены так, как устроены?».

Почему React?

Жизнь становится проще в том случае, если компоненты не знают об обмене данными по сети, о бизнес-логике приложения или о его состоянии. Такие компоненты, получая одни и те же входные параметры, всегда формируют одни и те же визуальные элементы.

Когда появилась библиотека React — это на фундаментальном уровне изменило то, как работают JavaScript-фреймворки и библиотеки. В то время как другие подобные проекты продвигали идеи MVC, MVVM и прочие подобные, в React был выбран другой подход. А именно, тут рендеринг визуальной составляющей приложения был изолирован от представления модели. Благодаря React во фронтенд-экосистеме JavaScript появилась совершенно новая архитектура — Flux.

Почему команда разработчиков React поступила именно так? Почему такой подход лучше тех, что появились раньше него, вроде архитектуры MVC и спагетти-кода, который пишут на jQuery? Если вы из тех, кого интересуют эти вопросы, можете посмотреть это выступление 2013 года, посвящённое разработке JavaScript-приложений в Facebook.

В 2013 году компания Facebook только что завершила серьёзную работу по интеграции в свою платформу чата. Эта новая возможность была встроена практически в каждую страницу проекта, чат влиял на обычные сценарии работы с платформой. Это было сложное приложение, встроенное в другое приложение, которое и до этого нельзя было назвать простым. Команде Facebook пришлось столкнуться с решением нетривиальных задач, справляясь с неконтролируемой мутацией DOM и с необходимостью обеспечить параллельную асинхронную работу пользователей в новой среде.

Например, как заранее узнать о том, что будет выведено на экране в ситуации, когда что угодно, в любое время и по любой причине, может обратиться к DOM и внести туда изменения? Как обеспечить правильность построения того, что увидит пользователь?

Используя популярные фронтенд-инструменты, существовавшие до React, ничего такого гарантированно обеспечить было нельзя. В ранних веб-приложениях «состояние гонок» в DOM было одной из самых распространённых проблем.

Отсутствие детерминизма = параллельные вычисления + мутабельное состояние.

Мартин Одерски

Главной задачей команды разработки React было решение этой проблемы. Они с ней справились, применив два основных инновационных подхода:

Однонаправленная привязка данных с использованием архитектуры Flux.

Иммутабельность состояния компонента. После того, как состояние компонента установлено, оно уже не может быть изменено. Изменения состояния не затрагивают визуализированные компоненты. Вместо этого подобные изменения приводят к выводу нового представления, обладающего новым состоянием.

Самый простой обнаруженный нами способ структурирования и рендеринга компонентов, с концептуальной точки зрения, заключался в том, чтобы просто стремиться к полному отсутствию мутаций.

Том Оччино, JSConfUS 2013

Библиотека React смогла серьёзно снизить остроту проблемы неконтролируемых мутаций благодаря использованию архитектуры Flux. Вместо того чтобы присоединять к произвольному количеству произвольных объектов (моделей) обработчики событий, вызывающие обновления DOM, библиотека React дала разработчикам единственный способ управления состоянием компонента. Это — диспетчеризация действий, влияющих на хранилище данных. Когда меняется состояние хранилища, система предлагает компоненту перерендериться.

Архитектура Flux

Когда мне задают вопрос о том, почему стоит обратить внимание на React, я даю простой ответ: «Дело в том, что нам нужен детерминированный рендеринг представлений, а React значительно упрощает решение этой задачи».

Обратите внимание на то, что чтение данных из DOM ради реализации некоей логики — это анти-паттерн. Тот, кто так поступает, идёт вразрез с целью использования React. Вместо этого данные нужно читать из хранилища, а решения, основанные на этих данных, нужно принимать до того, как будут отрендерены соответствующие компоненты.

Если бы детерминированный рендеринг компонентов был бы единственной фишкой React, то одно это уже было бы замечательной инновацией. Но команда разработчиков React на этом не остановилась. Эта команда представила миру библиотеку, обладающую и другими интереснейшими, уникальными возможностями. А по мере развития проекта в React появилось ещё больше всего полезного.

JSX

JSX — это расширение JavaScript, позволяющее декларативно создавать компоненты пользовательского интерфейса. JSX обладает следующими заметными возможностями:

Применение простой декларативной разметки.

Код разметки расположен там же, где и код компонента.

Реализация принципа разделения ответственностей (например — отделение описания интерфейса от логики состояния и от побочных эффектов). Причём, реализация, основанная не на использовании различных технологий (например — HTML, CSS, JavaScript).

Абстрагирование управления изменениями DOM.

Абстрагирование от особенностей различных платформ, для которых создают React-приложения. Дело в том, что благодаря использованию React можно создавать приложения, предназначенные для множества платформ (речь идёт, например, о разработке для мобильных устройств с использованием React Native, о приложениях для систем виртуальной реальности, о разработке для Netflix Gibbon, о создании Canvas/WebGL-интерфейсов, о проекте react-html-email).

Если, до появления JSX, нужно было декларативно описывать интерфейсы, то нельзя было обойтись без использования HTML-шаблонов. В те времена не было общепризнанного стандарта по созданию таких шаблонов. Каждый фреймворк использовал собственный синтаксис. Этот синтаксис приходилось изучать тому, кому, например, нужно было пройтись в цикле по неким данным, встроить в текстовый шаблон значения из переменных или принять решение о том, какой компонент интерфейса выводить, а какой — нет.

В наши дни, если взглянуть на разные фронтенд-инструменты, окажется, что без специального синтаксиса, вроде директивы \*ngFor из Angular, тоже не обойтись. Но, так как JSX можно назвать надмножеством JavaScript, создавая JSX-разметку можно пользоваться существующими возможностями JS.

Например, перебрать некий набор элементов можно, воспользовавшись методом Array.prototype.map. Можно использовать логические операторы, организовывать условный рендеринг с помощью тернарного оператора. Можно пользоваться чистыми функциями, можно конструировать строки с использованием шаблонных литералов. В общем-то, тому, кто описывает интерфейсы средствами JSX, доступны все возможности JavaScript. Полагаю, что в этом заключается огромное преимущество React перед другими фреймворками и библиотеками.

Вот пример JSX-кода:

const ItemList = ({ items }) => (

<ul>

{items.map((item) => (

<li key={item.id}>

<div>{item.name}</div>

</li>

))}

</ul>

);

Правда, при работе с JSX нужно учитывать некоторые особенности, которые, поначалу, могут показаться непривычными.

Тут используется подход к именованию атрибутов элементов, отличающийся от того, который принят в HTML. Например, class превращается в className. Речь идёт о применении стиля именования camelCase.

У каждого элемента списка, который нужно вывести, должен быть постоянный уникальный идентификатор, предназначенный для использования в JSX-атрибуте key. Значение идентификатора должно оставаться неизменным в ходе различных манипуляций с элементами списка. На практике большинство элементов списков в моделях данных имеют уникальные id, эти идентификаторы обычно отлично показывают себя в роли значений для key.

React не навязывает разработчику единственно правильный способ работы с CSS. Например, компоненту можно передать JavaScript-объект со стилями, записав его в свойство style. При таком подходе большинство привычных имён стилей будет заменено на их эквиваленты, записанные по правилам camelCase. Но этим возможности по работе со стилями не ограничиваются. На практике я одновременно пользуюсь разными подходами к стилизации React-приложений. Выбор конкретного подхода зависит от того, что именно нужно стилизовать. Например, глобальные стили я применяю для оформления тем приложений и макетов страниц, а локальные стили — для настройки внешнего вида конкретного компонента.

Вот мои любимые возможности React, касающиеся работы со стилями:

CSS-файлы, которые можно загружать в заголовочной части страницы. Они могут использоваться для настройки макетов страниц, шрифтов и прочих подобных элементов. Это — надёжный, работоспособный механизм стилизации.

CSS-модули — это CSS-файлы область применения которых ограничена локальной областью видимости. Их можно импортировать непосредственно в JavaScript-файлы. Для того чтобы применять CSS-модули, нужно воспользоваться правильно настроенным загрузчиком модулей. В Next.js, например, этот механизм активирован по умолчанию.

Пакет styled-jsx, который позволяет объявлять стили прямо в коде React-компонентов. Это напоминает использование тега <style> в HTML. Область видимости таких стилей можно назвать «гиперлокальной». Речь идёт о том, что стили воздействуют только на элементы, к которым они применяются, и на их дочерние элементы. При применении Next.js пакетом styled-jsx можно пользоваться без необходимости самостоятельно что-то подключать и настраивать.

Синтетические события

React даёт в наше распоряжение кроссбраузерную обёртку SyntheticEvents, представляющую синтетические события и предназначенную для унификации работы с событиями DOM. Синтетические события весьма полезны по нескольким причинам:

Они позволяет унифицировать особенности различных платформ, связанные с обработкой событий. Это упрощает разработку кроссбраузерных приложений.

Они автоматически решают задачи по управлению памятью. Если вы, например, собираетесь создать некий список с бесконечной прокруткой, пользуясь лишь чистыми JavaScript и HTML, то вам придётся делегировать события или подключать и отключать обработчики событий по мере появления и скрытия элементов списка. Всё это нужно будет делать для того чтобы избежать утечек памяти. Синтетические события автоматически делегируются корневому узлу, что приводит к тому, что React-разработчикам не приходится решать задачи по управлению памятью.

В их работе используются пулы объектов. Механизмы поддержки синтетических событий способны генерировать тысячи объектов в секунду и организовывать высокопроизводительную работу с такими объектами. Если решать подобные задачи, каждый раз создавая новые объекты, это приведёт к частой потребности в вызове сборщика мусора. А это, в свою очередь, может привести к замедлению программы, к видимым задержкам в работе пользовательского интерфейса и анимаций. Объекты синтетических событий создаются заранее и помещаются в пул объектов. Когда надобности в событии нет, оно возвращается обратно в пул. В результате разработчик может не беспокоиться о том, что сборщик мусора заблокирует главный поток JavaScript, очищая память от ставших ненужными объектов.

Обратите внимание на то, что из-за использования пула событий к свойствам синтетического события нельзя обратиться из асинхронной функции. Для реализации такой схемы работы нужно взять данные из объекта события и записать их в переменную, доступную асинхронной функции.

Жизненный цикл компонента

Концепция жизненного цикла React-компонентов ориентирована на защиту состояния компонента. Состояние компонента не должно меняться в процессе его вывода на экран. Это достигается благодаря следующей схеме работы: компонент оказывается в некоем состоянии и рендерится. Затем, благодаря событиям жизненного цикла, оказывается возможным применение к нему эффектов, можно воздействовать на его состояние, работать с событиями.

Понимание особенностей жизненного цикла компонентов React крайне важно для того чтобы разрабатывать интерфейсы и при этом не сражаться с React, а пользоваться этой библиотекой так, как задумано её разработчиками. «Сражение» с React, вроде неправильного изменения состояния компонентов или чтения данных из DOM, сводит на нет сильные стороны этой библиотеки.

В React, начиная с версии 0.14, появился синтаксис описаний компонентов, основанных на классах, позволяющий обрабатывать события жизненного цикла компонентов. В жизненном цикле React-компонентов можно выделить три важнейших этапа: Mount (монтирование), Update (обновление) и Unmount (размонтирование).

Жизненный цикл компонента

Этап Update можно разделить на три части: Render (рендеринг), Precommit (подготовка к внесению изменений в дерево DOM), Commit (внесение изменений в дерево DOM).

Структура этапа Update

Остановимся на этих этапах жизненного цикла компонента подробнее:

Render — на этом этапе жизненного цикла компонента производится его рендеринг. Метод компонента render() должен представлять собой детерминированную функцию, не имеющую побочных эффектов. Эту функцию стоит рассматривать как чистую функцию, получающую данные из входных параметров компонента и возвращающую JSX.

Precommit — на этом этапе можно прочитать данные из DOM, пользуясь методом жизненного цикла компонента getSnapShotBeforeUpdate. Это может оказаться очень кстати, например, если перед повторным рендерингом компонента нужно узнать нечто вроде позиции скроллинга или размеров визуализированного элемента.

Commit — на этой фазе жизненного цикла компонента React обновляет DOM и рефы. Здесь можно воспользоваться методом componentDidUpdate или хуком useEffect. Именно здесь можно выполнять эффекты, планировать обновления, использовать DOM и решать другие подобные задачи.

Дэн Абрамов подготовил отличную схему, которая иллюстрирует особенности работы механизмов жизненного цикла компонентов.

Жизненный цикл React-компонентов

Я полагаю, что представление компонентов в виде долгоживущих классов — это не самая лучшая ментальная модель React. Помните о том, что состояние React-компонентов не должно мутировать. Устаревшее состояние должно заменяться на новое. Каждая такая замена вызывает повторный рендеринг компонента. Это даёт React его, пожалуй, самую главную и самую ценную возможность: поддержку детерминированного подхода к созданию визуальных представлений компонентов.

Подобное поведение лучше всего представить себе так: при каждом рендеринге компонента библиотека вызывает детерминированную функцию, возвращающую JSX. Эта функция не должна самостоятельно вызывать собственные побочные эффекты. Но она, если ей это нужно, может передавать React запросы на выполнение подобных эффектов.

Другими словами, большинство React-компонентов имеет смысл представлять себе в виде чистых функций, получающих входные параметры и возвращающих JSX. Чистые функции обладают следующими особенностями:

Получая одни и те же входные данные, они всегда возвращают одни и те же выходные данные (они являются детерминированными).

У них нет побочных эффектов (то есть — они не работают с сетевыми ресурсами, не выводят что-либо в консоль, не записывают ничего в localStorage и так далее).

Обратите внимание на то, что если для работы некоего компонента нужны побочные эффекты, выполнять их можно, пользуясь useEffect или обращаясь к создателю действия, переданному компоненту через входные параметры и позволяющему организовать обработку побочных эффектов за пределами компонента.

Хуки React

В React 16.8 появилась новая концепция — хуки React. Это — функции, которые позволяют подключаться к событиям жизненного цикла компонентов, не пользуясь при этом синтаксисом классов и не обращаясь к методам жизненного цикла компонентов. Компоненты, в результате, стало возможным создавать не в виде классов, а в виде функций.

Вызов хука, в целом, означает появление побочного эффекта — такого, который позволяет компоненту работать со своим состоянием и с подсистемой ввода-вывода. Побочный эффект — это любое изменение состояния, видимое за пределами функции, за исключением изменения значения, возвращаемого функцией.

Хук useEffect позволяет ставить побочные эффекты в очередь для их последующего выполнения. Они будут вызываться в подходящее время жизненного цикла компонента. Это время может настать сразу после монтирования компонента (например — при вызове метода жизненного цикла componentDidMount), во время фазы Commit (метод componentDidUpdate), непосредственно перед размонтированием компонента (componentWillUnmount).

Обратили внимание на то, что с одним хуком связано целых три метода жизненного цикла компонента? Дело тут в том, что хуки позволяют объединять связанную логику, а не «раскладывать» её, как было до них, по разным методам жизненного цикла компонента.

Многим компонентам нужно выполнять какие-то действия во время их монтирования, нужно что-то обновлять при каждой перерисовке компонента, нужно освобождать ресурсы сразу перед размонтированием компонента для предотвращения утечек памяти. Благодаря использованию useEffect все эти задачи можно решить в одной функции, не разделяя их решение на 3 разных метода, не смешивая их код с кодом других задач, не связанных с ними, но тоже нуждающихся в этих методах.

Вот что дают нам хуки React:

Они позволяют создавать компоненты, представленные в виде функций, а не в виде классов.

Они помогают лучше организовывать код.

Благодаря им упрощается совместное использование одной и той же логики в разных компонентах.

Новые хуки можно создавать, выполняя композицию существующих хуков (вызывая их из других хуков).

В целом, рекомендуется пользоваться функциональными компонентам и хуками, а не компонентами, основанными на классах. Функциональные компоненты обычно компактнее компонентов, основанных на классах. Их код лучше организован, отличается лучшей читабельностью, лучше подходит для многократного использования, его легче тестировать.

Компоненты-контейнеры и презентационные компоненты

Я, стремясь улучшить модульность компонентов и их пригодность для многократного использования, ориентируюсь на разработку компонентов двух видов:

Компоненты-контейнеры — это компоненты, которые подключены к источникам данных и могут иметь побочные эффекты.

Презентационные компоненты — это, по большей части, чистые компоненты, которые, получая на вход одни и те же входные параметры и контекст, всегда выдают один и тот же JSX.

Чистые компоненты не следует отождествлять с базовым классом React.PureComponent, который назван именно так из-за того, что его небезопасно использовать для создания компонентов, не являющихся чистыми.

▍Презентационные компоненты

Рассмотрим особенности презентационных компонентов:

Они не взаимодействуют с сетевыми ресурсами.

Они не сохраняют данные в localStorage и не загружают их оттуда.

Они не выдают неких непредсказуемых данных.

Они не обращаются напрямую к текущему системному времени (например, путём вызова метода Date.now()).

Они не взаимодействуют напрямую с хранилищем состояния приложения.

Они могут использовать локальное состояние компонента для хранения чего-то наподобие данных, введённых в формы, но при этом они должны поддерживать возможность приёма таких данных и задания на их основе своего исходного состояния, что облегчает их тестирование.

Именно из-за последнего пункта этого списка я, говоря о презентационных компонентах, упомянул о том, что это, по большей части, чистые компоненты. Эти компоненты считывают своё состояние из глобального состояния React. Поэтому хуки вроде useState и useReducer дают в их распоряжение неявным образом определённые данные (то есть — данные, не описанные в сигнатуре функции), что, с технической точки зрения, не позволяет назвать такие компоненты «чистыми». Если нужно, чтобы они были бы по-настоящему чистыми, можно делегировать все задачи по управлению состоянием компоненту-контейнеру, но я полагаю, что делать этого не стоит, по крайней мере, до тех пор, пока правильность работы компонента можно проверить с помощью модульных тестов.

Лучшее — враг хорошего.

Вольтер

▍Компоненты-контейнеры

Компоненты-контейнеры — это такие компоненты, которые отвечают за управление состоянием, за выполнение операций ввода-вывода и за решение любых других задач, которые можно отнести к побочным эффектам. Они не должны самостоятельно рендерить некую разметку. Вместо этого они делегируют задачу рендеринга презентационным компонентам, а сами служат обёрткой для таких компонентов. Обычно компонент-контейнер в React+Redux-приложении просто вызывает mapStateToProps() и mapDispatchToProps(), после чего передаёт соответствующие данные презентационным компонентам. Контейнеры, кроме того, могут использоваться для решения некоторых задач общего характера, о которых мы поговорим ниже.

Компоненты высшего порядка

Компонент высшего порядка (Higher Order Component, HOC) — это компонент, который принимает другие компоненты и возвращает новый компонент, реализующий новый функционал, основанный на исходных компонентах.

Компоненты высшего порядка функционируют, оборачивая одни компоненты другими. Компонент-обёртка может реализовывать некую логику и создавать элементы DOM. Он может передавать оборачиваемому компоненту дополнительные входные параметры, а может этого и не делать.

В отличие от хуков React и от механизма render props, компоненты высшего порядка поддаются композиции с использованием стандартного подхода к композиции функций. Это позволяет декларативно описывать результаты композиции возможностей, предназначенных для использования в разных местах приложения. При этом готовые компоненты не должны знать о существовании тех или иных возможностей. Вот пример HOC с EricElliottJS.com:

import { compose } from 'lodash/fp';

import withFeatures from './with-features';

import withEnv from './with-env';

import withLoader from './with-loader';

import withCoupon from './with-coupon';

import withLayout from './with-layout';

import withAuth from './with-auth';

import { withRouter } from 'next/router';

import withMagicLink from '../features/ethereum-authentication/with-magic-link';

export default compose(

withEnv,

withAuth,

withLoader,

withLayout({ showFooter: true }),

withFeatures,

withRouter,

withCoupon,

withMagicLink,

);

Тут показана смесь множества возможностей, совместно используемых всеми страницами сайта. А именно, withEnv читает настройки из переменных окружения, withAuth реализует механизм GitHub-аутентификации, withLoader показывает анимацию во время загрузки данных пользователя, withLayout({ showFooter: true }) выводит стандартный макет с «подвалом», withFeature показывает настройки, withRouter загружает маршрутизатор, withCoupon отвечает за работу с купонами, а withMagicLing поддерживает аутентификацию пользователей без пароля с использованием Magic.

Кстати, учитывая то, что аутентификация пользователей с помощью пароля устарела, и то, что это — опасная практика, в наши дни стоит использовать другие методы аутентификации пользователей.

Почти все страницы вышеупомянутого сайта используют все эти возможности. Учитывая то, что их композиция выполнена средствами компонента высшего порядка, можно включить их все в компонент-контейнер, написав всего одну строку кода. Вот, например, как это будет выглядеть для страницы с уроками:

import LessonPage from '../features/lesson-pages/lesson-page.js';

import pageHOC from '../hocs/page-hoc.js';

export default pageHOC(LessonPage);

У подобных компонентов высшего порядка есть альтернатива, но она представляет собой сомнительную конструкцию, называемую «pyramid of doom» («пирамида погибели») и ей лучше не пользоваться. Вот как это выглядит:

import FeatureProvider from '../providers/feature-provider';

import EnvProvider from '../providers/env-provider';

import LoaderProvider from '../providers/loader-provider';

import CouponProvider from '../providers/coupon-provider';

import LayoutProvider from '../providers/layout-provider';

import AuthProvider from '../providers/auth-provider';

import RouterProvider from '../providers/RouterProvider';

import MagicLinkProvider from '../providers/magic-link-provider';

import PageComponent from './page-container';

const WrappedComponent = (...props) => (

<EnvProvider { ...props }>

<AuthProvider>

<LoaderProvider>

<LayoutProvider showFooter={ true }>

<FeatureProvider>

<RouterProvider>

<CouponProvider>

<MagicLinkProvider>

<YourPageComponent />

</MagicLinkProvider>

</CouponProvider>

</RouterProvider>

</FeatureProvider>

</LayoutProvider>

</LoaderProvider>

</AuthProvider>

</EnvProvider>

);

И это придётся повторить на каждой странице. А если надо будет что-то в этой конструкции изменить, то изменения в неё придётся вносить везде, где она присутствует. Полагаю, недостатки такого подхода совершенно очевидны.

Использование композиции для решения задач общего характера — это один из лучших способов уменьшения сложности кода приложений. Композиция — это настолько важно, что я даже написал об этом книгу.

Итоги

Почему React? React даёт нам детерминированный рендеринг визуальных представлений компонентов, в основе которого лежит однонаправленная привязка данных и иммутабельное состояние компонентов.

JSX даёт нам возможность простого декларативного описания интерфейсов в JavaScript-коде.

Синтетические события сглаживают кросс-платформенные различия систем обработки событий и облегчают управление памятью.

Концепция жизненного цикла компонентов направлена на защиту состояния компонентов. Жизненный цикл компонента состоит из фаз монтирования, обновления и размонтирования. Фаза обновления состоит из фазы рендеринга, фазы подготовки к внесению изменений в DOM и фазы внесения изменений в DOM.

Хуки React позволяют подключаться к методам жизненного цикла компонентов без использования синтаксиса, основанного на классах. Применение хуков, кроме того, облегчает совместное использование одного и того же кода в разных компонентах.

Компоненты-контейнеры и презентационные компоненты позволяют отделить задачи формирования визуального представления интерфейсов от задач по управлению состоянием приложения и от побочных эффектов. Это улучшает возможности по многократному использованию и тестированию компонентов и бизнес-логики приложения.

Компоненты высшего порядка упрощают совместное использование возможностей, представляющих собой композицию других возможностей. При этом компонентам не нужно знать об этих возможностях (и не нужно, чтобы компоненты были бы тесно связаны с ними).

Что дальше?

В этом материале о React мы затронули множество концепций функционального программирования. Если вы стремитесь к глубокому пониманию принципов разработки React-приложений, вам полезно будет освежить свои знания о чистых функциях, об иммутабельности, о каррировании и частичном применении функций, о композиции функций. Соответствующие материалы вы можете найти на EricElliottJS.com.

Я рекомендую использовать React совместно с Redux, Redux-Saga и RITEway. Redux рекомендуется использовать совместно с Autodux и Immer. Для организации сложных схем работы с состоянием можно попробовать воспользоваться Redux-DSM.

Когда вы разберётесь с основами и будете готовы к созданию реальных React-приложений, обратите внимание на Next.js и Vercel. Эти инструменты помогут автоматизировать настройку системы сборки проекта и CI/CD-конвейера, с их помощью можно подготовить проект к оптимизированному развёртыванию на сервере. Они дают тот же эффект, что и целая команда DevOps-специалистов, но пользоваться ими можно совершенно бесплатно.

* 1. Фреймворк Angular

# 3. ОПИСАНИЕ ПРИЛОЖЕНИЯ( СТАРАЯ ГЛАВА -ВНЕДРЕНИЕ ГУГЛ-СЕРВИСОВ В СИСТЕМЫ, ИСПОЛЬЗУЮЩИЕ VUEJS)

3.1. **Описание структуры приложения**

В ходе данной работы были рассмотрены преимущества гибридных web технологий, были изучены их основные элементы, а также, как работают ключевые технологии, входящие в данный набор, такие как Firebase, Google OAuth, Cordova, Vue, автономная работа приложений и другие.Приложение имеет следующие функциональные возможности:

* Авторизация пользователя - имея логин и пароль, пользователи, зарегистрированные в системе, имеют возможность войти в систему;
* Список растрат, сделанных членами семьи - пользователи имеют доступ к списку месячных растрат, сделанным ими, и / или тем, которые находятся на их уровне доступа;
* Модуль отчетов, в котором есть список отчетов, доступ к которым так же определяется в зависимости от уровня доступа.

Авторизация пользователей - это первый шаг, который должны сделать пользователи. Это делается путем ввода имени пользователя и пароля, полученных от системного администратора. Если логин и / или пароль были введены неправильно, пользователю показываются соответствующие сообщения, так что он понимает, какие данные были введены неправильно.

После авторизации, пользователь перенаправляется на свою главную страницу. Он может содержать различную информацию, такую как: карточка, содержащая информацию о совершенной покупке, ее цене и, по возможности, картинке; имени пользователя, изображении профиля, почты.

Пользователь, в зависимости от уровня доступа, имеет возможность просматривать, оценивать и добавлять собственные записи.

* 1. Черты гибридных технологий в приложении Swallow
  2. Чем обосновано предпочтение гибридных технологий нативным

3.2. Установка и настройка среды разработки

3.2.1. Node Package Manager (NPM)

Рекомендуется использовать NPM при создании больших приложений на Vue. Эта опция прекрасно работает с такими инструментами сборки, как Webpack и Browserify. Vue также имеет совместимые инструменты для использования однофайловых компонентов. Команда установки: npm install vue устанавливает последнюю стабильную версию.

3.2.2. Инструменты командной строки (CLI)

Vue.js предоставляет инструкции (CLI) для быстрого создания инфраструктуры для амбициозных одностраничных приложений. Всего за несколько минут вы получите рабочую конфигурацию с перезагрузкой модулей, анализом ошибок при настройке производственной сборки.

CLI - это инструмент для тех, кто знаком с Node.js и соответствующими инструментами сборки. Команда установки: npm install vue-cli.

3.2.3. Использование Cordova для определения гибридной трансляции

Средство командной строки Cordova распространяется как пакет npm в готовом к использованию формате. Нет необходимости для его компиляции из исходного кода.

Чтобы установить инструмент командной строки cordova, необходимо выполнить следующие действия:

* Выполнить предыдущие 2 пункта;
* Установить модуль cordova с помощью инструмента npm Node.js.

Модуль cordova будет автоматически загружен npm при вводе команды npm install –g cordova.

3.2.3. Создание приложения

Разработка приложений в контексте рабочего пространства VueJs. Рабочая область содержит файлы для одного или нескольких проектов. Чтобы создать новое рабочее пространство и начальный проект для приложения, нужно выполнить команду CLI: vue create vue-app, vue-app в данном случае это название создаваемого проекта, оно может быть произвольным.

Следуя этим командам, все файлы, необходимые для начала создания приложения, а также основные компоненты App.Vue и main.js динамически создаются в выбранной папке (Рис.2.2). Далее нужно установить Vuetify. Для этого нужно ввести команду *vue add vuetify.* После установки Vuetify нужно установить cordova через команду *vue add cordova*.

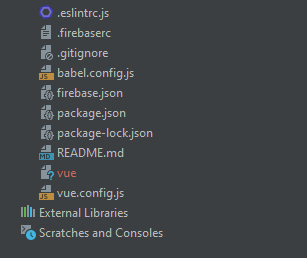
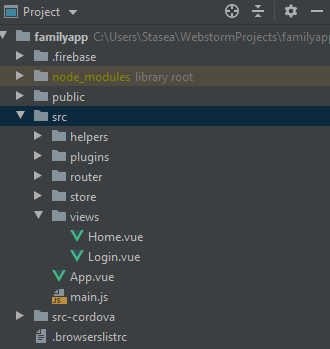


Рис.2.2. Стуктура vue vli проекта.

3.3. Создание проекта в Firebase

Если учетная запись Google уже существует, можно сразу войти в систему на странице https://firebase.google.com/. В противном случае необходимо зарегистрироваться по адресу http:// accounts.google.com. После входа можно создать свой проект. После нажатия кнопки Add project (Добавить проект) нужно ввести название проекта и выбрать страну. Нажать кнопку Create Project (Создать проект), чтобы перейти в консоль Firebase. Здесь конфигурируется база данных и аутентификация, а также генерируются ключи, которые нужны для начала работы. В разделе Database (База данных) есть два варианта: Realtime Database и Cloud Firestore. В проекте используется Realtime Database.

3.4. Интеграция Firebase

После настройки Firebase, следует его интеграция в приложение. Для этого файлы из библиотеки Firebase импортируются в файл проекта main.js.

import \* as firebase from "firebase/app";

import "firebase/analytics";

import "firebase/auth";

import "firebase/storage";

import "firebase/firestore";

Далее необходимо авторизовать приложение в Firebase. Для этого приложение инициализируется следующих свойств:

firebase.initializeApp({

apiKey: "AIzaSyDSwwHp51-LlwzIak0WXF8cScPMSS9VBwU",

authDomain: "family-d2146.firebaseapp.com",

databaseURL: "https://family-d2146.firebaseio.com",

projectId: "family-d2146",

storageBucket: "family-d2146.appspot.com",

messagingSenderId: "117478996669",

appId: "1:117478996669:web:e16a09e0f1bcf69ccfd5d4"

});

Для того, чтобы firebase был доступен глобально для всех компонентов приложения, необходимо написать следующее:

Vue.prototype.$firebase = firebase;

Далее происходит создание файла Login.vue, служащего для авторизации пользователя в приложении. Следом происходит создание интерфейса авторизации при загрузке страницы.

this.ui = new this.auth.AuthUI(this.$firebase.auth());

Далее был отрисован интерфейс в таг с id #firebaseui-auth-container.

this.ui.start("#firebaseui-auth-container")

<v-container class="auth-container" fluid>

<div id="firebaseui-auth-container"></div>

</v-container>

Вслед за этим новый объект интерфейса(инстанс) интерфейса передает callback, который вызывается при успешной регистрации.

signInSuccessWithAuthResult: authResult => {

const { user } = authResult;

this.$store.dispatch("setUserInfo", {

displayName: user.displayName,

email: user.email,

photoURL: user.photoURL

});

this.$router.push({ name: "Home" });

Код описанный выше запишет в глобальное хранилище store все данные пользователя для дальнейшего из использования в меню и при сохранении покупок приложения и перенаправит его на главную страницу.

Далее были описаны методы авторизации в приложении

signInOptions: [

this.$firebase.auth.GoogleAuthProvider.PROVIDER\_ID,

this.$firebase.auth.EmailAuthProvider.PROVIDER\_ID

]

Далее был написан метод для проверки авторизации при загрузке страницы и обновления данных в store, если они актуальны, в ином случае переадресации на страницу авторизации

this.$firebase.auth().onAuthStateChanged(user => {

if (user) {

this.$store.dispatch("setUserInfo", {

displayName: user.displayName,

email: user.email,

photoURL: user.photoURL

});

} else {

this.$router.push({ name: "Login" });

}

});

После проверки авторизации пользователь должен подписаться на обновления всех покупок и получить текущий список.

this.$firebase

.firestore()

collection("goods")

.orderBy("date", "desc")

onSnapshot(querySnapshot => {

this.items = [];

querySnapshot.forEach(doc => {

this.items.push(doc.data());

});

});

Затем из базы данных загружается firestore отсортированная коллекция goods и записывается в массив items для отрисовки на странице

<v-card class="mx-auto" :key="index" v-for="(item, index) of items">

<v-list-item>

<v-list-item-avatar color="grey">

<v-img :src="item.photoURL"></v-img>

</v-list-item-avatar>

<v-list-item-content>

<v-list-item-title class="headline">{{

item.displayName

}}</v-list-item-title>

<v-list-item-subtitle>{{ item.email }}</v-list-item-subtitle>

</v-list-item-content>

</v-list-item>

<v-img v-if="item.image" :src="item.image"></v-img>

<v-card-text>

{{ item.text }}

</v-card-text>

<v-card-actions>

<v-btn text color="deep-purple accent-4"> {{ item.price }} $ </v-btn>

<v-spacer></v-spacer>

<v-btn text color="deep-purple accent-4">

{{ getFormattedDate(item.date) }}

</v-btn>

</v-card-actions>

</v-card>

После отрисовки данных пользователь может просмотреть, оценить или добавить новые записи. Ниже приведен пример кода для отображения кнопки добавления

<v-btn @click="dialog = true" color="primary" bottom right>

<v-icon>mdi-plus</v-icon>

</v-btn>

При нажатии на кнопку пользователю будет предложено создать новую запись, написать название, цену и прикрепить картинку. После сохранения покупки данные будут отравлены на сервер.

if (this.image) {

const snapshot = await this.$firebase

.storage()

.ref()

.child(`images/${this.generateUuid()}`)

.put(this.image);

image = await snapshot.ref.getDownloadURL();

}

Код выше проверяет наличие картинки, и загружает ее на сервер. Для того что бы они не повторялись для каждой был сгенерирован уникальный номер. После загрузки картинки на сервер, в переменную записывается ее ссылка для сохранения базе данных. После получения ссылки, все данные сохраняются в базу данных, в коллекцию под названием goods. Ниже приведен пример кода:

try {

await this.$firebase

.firestore()

.collection("goods")

.add({

text: this.text,

price: this.price,

date: new Date(),

...this.user,

image

})

После добавления данных в базу диалоговое окно будет закрыто, а данные полей сброшены.

# Заключение

В данной работе было произведено исследование гибридных web приложений. Был произведен анализ создания web приложений применяя соответствующий набор технологий.

В процессе выполнения работы были:

* Изучены основы гибридных web приложений;
* Исследованы возможности использования гибридных web технологий в качестве замены нативных технологий;
* Анализ работы современных облачных технологий для интеграции в приложения;
* Разработано приложение с использованием данного набора технологий.

В ходе данной работы были рассмотрены преимущества гибридных web технологий, были изучены их основные элементы, а также, как работают ключевые технологии, входящие в данный набор, такие как Firebase, Google OAuth, Cordova, Vue, автономная работа приложений и другие.

На основе всего вышесказанного, можно сформулировать вывод, что гибридные web технологии являются крайне полезными и инновационными инструментами для создания мобильных приложений. Гибридные web приложения включают в себя большинство преимуществ нативных приложений, при этом технически оставаясь - web приложением.

В заключении можно добавить, что в перспективе гибридные web технологии позволят убрать проблему кроссплатформенности, так как они позволяют вести разработку одновременно под все платформы сразу. Это приведет к более прогрессивному, прямолинейному и рентабельному пути разработки приложений, ведь теперь компаниям не нужно будет беспокоиться о разработке приложения сразу под несколько разных платформ, что выражается не только в количестве времени, потраченного на разработку, но и количества и специализации самих разработчиков нужных для разработки.

Данная работа предназначена для изучения гибридных web приложений, а также для ознакомления с методами и архитектурой web приложений на основе гибридных web технологии, также может быть полезна разработчикам, студентам, учителям и всем желающим ознакомиться с гибридными web приложениями.

# Библиография

1. *Поверхностный обзор javascript-enabled BaaS платформ*. Доступен: <https://habr.com/ru/post/150729/> [on-line].[цитирован 15.03.20];
2. *Firebase secures its real-time back-end service.* Доступен: <https://gigaom.com/2012/12/18/firebase-secures-its-real-time-back-end-service/>[on-line].[цитирован 20.03.20];
3. *Gerardus Blokdyk. Firebase The Ultimate Step-By-Step Guide. Emereo Pty Ltd*. ISBN 9780655321477;
4. *Используйте Firebase, единый кроссплатформенный SDK от Google, чтобы улучшить приложения.* Доступен: <https://developer.android.com/distribute/best-practices/develop/build-with-firebase?hl=ru> [on-line].[цитирован 27.03.20];
5. *Sourav BHOI. Internet of Things: an Application Based Approach Using Arduino Platform and Firebase. Independently Published, 2018* ISBN 1983041378, 9781983041372;
6. *Обзор облачных сервисов для разработки бэкенда мобильных приложений*. Доступен: <https://habr.com/ru/company/surfstudio/blog/463435/> [on-line].[цитирован 07.04.20];
7. *OAuth 2.0 простым и понятным языком*. Доступен: <https://habr.com/ru/company/mailru/blog/115163/> [on-line].[цитирован 10.04.20];
8. Описание сервиса Google Maps <https://wiki.rookee.ru/google-maps/> [on-line].[цитирован 10.04.21];
9. *Использование сервисов Firebase при разработке мобильных приложений*. Доступен: <https://cmsmagazine.ru/journal/cases-4264/> [on-line].[цитирован 16.04.20];
10. *Нативные, веб и гибридные приложения: что выбрать?* Доступен: <https://smartum.pro/ru/blog-ru/native-web-and-hybrid-apps/> [on-line].[цитирован 26.04.20];
11. *Нативные vs. гибридные приложения*. Доступен: <https://umbrellait.com/ru/blog/native-vs-hybrid-app/> [on-line].[цитирован 04.05.20];
12. *Разница между нативным и гибридным мобильным приложением*. Доступен: <https://wezom.com.ua/blog/chem-otlichajutsja-nativnoe-i-gibridnoe-mobilnye-prilozhenija> [on-line].[цитирован 10.05.20];
13. *Vue.js*. Доступен: <https://ru.vuejs.org/v2/guide/index.html> [on-line].[цитирован 10.05.20];
14. *Callum Macrae. Vue.js: Up and Running., O’Reilly, 2017., 219 с.,* [ISBN 9781491997246](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9781491997246);
15. *Alex Kyriakidis, Kostas Maniatis.*[*The Majesty of Vue.js*](https://books.google.ru/books?id=Xp7cDgAAQBAJ)*., Packt Publishing Ltd, 2016., 230 с.,* [ISBN 9781787125209](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9781787125209);
16. *VuetifyJS — строим свое приложение в стиле Material Design*. Доступен: <https://weatherless.ru/javascript/vuejs/vuetifyjs-stroim-svoe-prilozhenie-v-stile-material-design/> [on-line].[цитирован 20.05.20];
17. *Хэнчеп Эрик, Листуон Бенджамин Х88 Vue.js в действии., СПб.: Питер, 2019., 304 с.: ил., (Серия «Библиотека программиста»). Стр. 257*. ISBN 9785446110988

Declaralie privind asumarea raspunderii

Subsemnata \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, absolventa al Universitdtii de Stat “Alecu Russo” din Balti, Facultatea de \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ program de studii \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, declar pe propria raspundere ca teza de licenta/teza de master cu titlul \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, elaborata sub indrumarea dlui/dnei \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ este rezultatul propriilor cercetdri.

Declar ca nu am plagiat altd lucrare de licenta/master, monografii, articole etc., publicate sau postate pe Internet, toate sursele utilizate in tezi fiind mentionate in cuprinsul acesteia.

De asemenea, declar ca sunt de acord ca teza de licenta/teza de master sa fie verificata pentru confirmarea originalitilii.

Data Semnatura