Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по преддипломной практике

Выполнил студент гр. 852001
Ярошевич Ю. А.

Руководитель практики от
предприятия:
начальник отдела
Ковалёв А. М.

Руководитель практики
от университета:
доцент кафедры информатики
Волосевич А. А.

Минск 2013

СОДЕРЖАНИЕ

1 ХАРАКТЕРИСТИКА МЕСТА ПРАКТИКИ

Частная компания ООО «Техартгруп» занимается предоставлением услуг по разработке программного обеспечения, консалтингом и внедрением корпоративных решений для многих европейских и североамериканских компаний. Компания была основана в 2003 году. Штаб-квартира компании расположена в городе Изелин (Iselin), штат Нью-Джерси, Соединенные Штаты Америки. Центры разработки компании расположены в Минске, Беларусь и Киеве, Украина. Среди партнеров компании можно выделить такие общеизвестные компании как Microsoft, Oracle, IBM, Adobe и другие.

Компания предоставляет следующие услуги своим заказчикам [**?**]:

– разработка программного обеспечения;
– интеграция решений в существующую инфраструктуру;
– разработка мобильных приложений;
– миграция больших объемов данных;
– проведение тестирования и контроля качества;
– поддержка существующих решений;
– консалтинг в сфере информационных технологий;
– администрирование баз данных;
– управление инфраструктурой;
– управление проектами.

Рабочая модель взаимодействия с заказчиками — оффшорный центр разработки. Данная модель представляет собой виртуальную команду разработчиков программного обеспечения. Команда создается в соответствии с требованиями клиента относительно проекта и специфики его бизнеса и выступает в качестве удаленного расширения внутреннего штата компании клиента.

В соответствии с требованиями клиентов компания предлагает различные модели взаимодействия. Разработка на стороне клиента — данный подход находит свое применение для наиболее сложных и больших проектов, когда нужно тесное взаимодействие заказчика и команд исполнителя, или когда разработка не может быть передана в другое место, например по причине законодательных ограничений, принятых в стране клиента. Оффшорная организация — вся работа по удовлетворению потребностей заказчика выполняется на стороне компании-исполнителя, данная модель является наиболее экономичной для заказчика. При смешанной организации основная работа выполняется на стороне «Техартгруп», но управление проектом и выработка бизнес требований выполняется представителем «Техартгруп» на сто-

роне клиента или представителем клиента на стороне компании-исполнителя. Данная модель часто используется для больших проектов, когда нужно соблюдать баланс между стоимостью проекта и эффективностью взаимодействия с клиентом [**?**]. В качестве примера компаний-клиентов «Техартгруп» можно привести следующие компании: Coca-Cola, Disney, FedEx, Gain Capital, 10gen и другие общеизвестные компании.

Для более удобного управления структурная организация компании представляет из себя множество отделов. В компании присутствуют отделы занимающиеся разработкой мобильных приложений для iOS и Android, разработкой приложений для Microsoft .NET, разработкой приложений для платформы Java, также есть отделы разработки, специализирующиеся на других технологиях. Управление компанией осуществляет административный отдел. Также присутствуют отделы материально-технического обеспечения, тестирования и контроля качества. Каждый отдел имеет своего руководителя с которым решаются многие вопросы, возникающие у сотрудников отдела.

Кроме деятельности направленной на зарабатывание денег, компания занимается обучением студентов. В компании с недавнего времени проходят тренинги по веб-разработке и разработке на платформе Microsoft .NET. По результатам курсов многим студентам предлагают работать в компании. Компания также сотрудничает и помогает различным университетам в Беларуси. Благодаря помощи компании был модернизирован студенческий читальный зал №1 в БГУИР.

В компании работает много молодых и зрелых специалистов. Компания хорошо относится к своим сотрудникам, созданы условия для отдыха и развлечения сотрудников. В офисе компании есть комната для отдыха и развлечений, созданная специально для сотрудников. В теплое время года компания часто организует активный отдых за городом для своих работников.

Прохождение преддипломной практики было в команде занимающейся поддержкой и развитием существующей инфраструктуры для компании GAIN Capital Holdings, Inc. Данная компания является пионером в области онлайн торговли иностранными валютами и является владельцем бренда FOREX.com Результаты, полученные в ходе выполнения индивидуального задания, в данный момент используются компанией GAIN Capital для рассылки уведомлений для своего приложения FOREXTrader for iPhone™. Разработанное решение было гармонично вписано в существующую сервисно-ориентированную архитектуру инфраструктуры GAIN Capital.

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

2.1 Синхронизация данных

При разработке видеоплеера было необходимо определиться с теми технологиями и методами синхронизации и обмена данными, которые доступны для реализации в веб-приложении. Были рассмотрены следующие варианты: стриминг видео, обмен синхронизирующих сообщений между приложениями, обмен синхронизирующих сообщений через сервер.

2.1.1 Стриминг

Разберёмся с понятием стриминга. В данном случае речь идёт об HTML5-стриминге, также существует понятие HTML5-видео. Различия заключаются в том, то для HTML5-видео используется готовый видео файл, а для HTML5-стриминга постоянный видеопоток (видео на YouTube — это HTML5-видео, а трансляция на Twitch — HTML5-стриминг).

Недостатком данного метода является наличие большого количества кодеков, транспортных и видео протоколов, которые имеют свои нюансы, ограничения и проблемы совместимости. Также данный метод не подходит для данного проекта, так как данная система накладывает ограничения на видеофайлы, которые пользователь сможет использовать, потому что сначала их необходимо загрузить на сервер, откуда они будут передаваться остальным пользователям. Это также ограничивает возможности синхронизации, так как при стриминге для обработки запросов плеера (пауза, перемотка) для множества сессий потребуются большие серверные мощности. Также это накладывает ограничения на скорость исходящего интернет соединения пользователя, необходимое для передачи видеопотока на сервер.

2.1.2 Обмен сообщений между приложениями

Второй вариант заключается в том, что каждое приложение является независимым и для синхронизации оно обменивается сообщениями о состоянии видео с другими приложениями. Данный метод позволяет снизить нагрузку на сервер, так как основные действия будут происходить на устройстве пользователя. Остаётся вопрос обмена синхронизирующими сообщениями. Для этих целей подходит технология Web Real-Time Communications. Web Real-Time Communications (WebRTC) - это технология, которая позволяет веб-приложениям и сайтам осуществлять захват и передачу аудио и/или видео потоков. Также имеется возможность обмениваться данными произвольного типа между браузерами, без необходимости посредника в этом

процессе. На данный момент реализация WebRTC в современных браузерах неполная, что выражается в различной степени реализованности как функций, так и кодеков WebRTC. WebRTC предлагает ряд интерфейсов для передачи медиа потоков и произвольных данных, контроля соединения, управление идентификацией и многое другое.

В основе WebRTC лежит ряд протоколов, которые обеспечивают возможность P2P соединения:

- Interactive Connectivity Establishment (ICE);
- Session Traversal Utilities for NAT (STUN);
- Session Traversal Utilities for NAT (STUN);
- Network Address Translation (NAT);
- Traversal Using Relays around NAT (TURN);
- Session Description Protocol (SDP).

Network Address Translation (NAT) - механизм, позволяющий заменить внутренний IP адрес и порт в пакетах, отправляемых с локального устройства, на внешний адрес маршрутизатора для доступа во внешнюю сеть. Существует несколько способов трансляции адреса: статический, при котором адреса соотносятся один к одному (это необходимо когда данное устройство должно быть доступно из внешней сети), динамически, когда внутреннему адресу ставиться один из доступных адресов, и перегруженный, при котором адреса нескольких устройств заменяются на один и тот же адрес, но с различными номерами портов. Существует несколько типов NAT, которые способны оказать трудности в процессе установки соединения между устройствами. Из них нам интересен симметричный NAT.

Симметричный NAT - трансляция, при которой каждый запрос с определённого внутреннего адреса и порта на конкретный внешний адрес и порт преобразуется в уникальный внешний адрес и порт. Если устройства с одинаковым адресом и портом отправят запрос на разные адреса и порты, им будут присвоены разные адреса. При данном типе NAT компьютер способен получать пакеты только от тех источников, которым он уже отправлял запросы, иначе говоря, запрос из внешней сети от неизвестного отправителя будет проигнорирован.

Interactive Connectivity Establishment (ICE) - это технология, используемая для нахождения кратчайшего пути коммуникации между двумя компьютерами. Данный протокол необходим для тех приложений, в которых обработка сообщений через центральный сервер является неэффективным как в плане скорости передачи данных, так и из-за дополнительных финансовых затрат.

Для установки соединения необходимо знать свой IP адрес и ограничения NAT. В этом помогает вспомогательный сервер, который реализует Session Traversal Utilities for NAT (STUN) протокол. Если роутер реализует симметричный NAT, то организация соединения по публичным IP адресам становиться невозможной. В таком случае необходимо использовать дополнительный сервер, реализующий Traversal Using Relays around NAT (TURN). Работа данного сервера заключается в пересылке данных в обход симметричного NAT. Этот подход имеет свои издержки и его принято использовать при отсутствии иных способов установить соединение.

Установки соединения между браузерами, используя WebRTC, также является нежелательным методом для данного проекта, потому что для того, чтобы все пользователи могли использовать видеоплеер, необходимо много дополнительной ресурсов на разработку и поддержание системы установки соединения. Минусом данного способа является его ненадёжность и высокие затраты на реализацию механизма обмена сообщений. Также данный метод является ненадёжным в ситуации потери соединения из-за независимости приложений, что является нежелательным для данного дипломного проекта.

2.1.3 Обмен сообщений с сервером

Для реализации поставленной цели было решено использовать метод, при котором приложения будут обмениваться сообщениями через серве. Это позволит избежать потери данных при сохранении приемлемой скорости. Приложения будут точно зантъ необходимое состояние видео, так как верная информация об этом будет всегда находиться на сервере. Также данный метод позволяет при рассинхронизации принудительно синхронизировать клиенты, так как имеется единый источник информации. Описание способа обмена сообщений через червер представлено в разделе 2.2.

2.2 Технологии для сервера

Для того, чтобы упростить разработку приложения, было решено использовать сервис Firebase от компании Google. Данный сервис предлагает множество модулей, которые очень полезны для веб-разработки, создания мобильных или настольных приложений. Данный сервис избавляет от нужды в собственном сервере, так как Firebase реализует практически все возможности, которые необходимы от стандартного сервера. Главным инструментом в составе Firebase является их база данных. На данный момент Firebase предлагает несколько различных вариантов баз данных: Realtime Database и

Firestore. Их главное отличие заключается в форме хранения данных и доступа к ним.

Realtime Database использует JSON файлы для хранения информации. Это является крайне неэффективным, так как для получения определённых данных сначала нужно получить все данные из файла, а только потом можно произвести поиск. Это увеличивает потребность приложения в ресурсах системы, что недопустимо для данного проекта.

Firestore — облачная NoSQL база данных реального времени. Это означает то, что, при использовании данной базы данных, пользователь может "подписаться"на определённые документы. В таком случае при изменении содержимого документа пользователь сразу получит новые данные. Благодаря данному механизму можно быть уверенным, что конечный пользователь получит самую свежую информацию, что является важным для данного проекта. Firestore поддерживает следующие типы данных:

- текстовая строка;
- числа;
- булевые значения;
- массивы;
- даты;
- словари.

Так как Firestore является NoSQL базой данных, то для организации данных в ней используются не таблицы, а документы. Отличие заключается в том, что документ не имеет жёсткой структуры, как таблицы, что позволяет хранить в документах данные произвольных типов и менять их значение и структуру по мере необходимости. Данный способ позволяет более гибко взаимодействовать с данными, но лишает дополнительной надёжности, свойственной SQL базам данных.

Для организации документов используются коллекции. Коллекция — это просто набор документов. Они не обязаны быть одинаковыми, но данный случай крайне нежелателен. Каждый документы в коллекции имеет идентификатор, уникальный для данной коллекции. В качестве идентификатора может выступать любая подходящая текстовая строка. Документы также способны содержать внутри себя вложенные коллекции, однако данная возможность не особо полезна. Также имеется возможность создания пользовательских функций и триггеров для взаимодействия с базой данных, что позволяет выполнять определённые операции при изменении, создании или удалении документов и коллекций.

Firebase также предлагает встроенную систему аутентификации поль-

зователей, которая поддерживает различные источники для идентификации: почта, профиль Google и др.

Также плюсом Firebase является возможность его интеграции с другими сервисами компании Google и дополнительная защита от прекращения работы собственного сервера.

2.3 Недостатки Firebase

Основные возможности, которые предоставляет Firebase связанные с хранением, изменением и получением информации из встроенной базы данных. Firebase не имеет возможности для создания API, поэтому для решения вспомогательных задач, не связанных с базой данных необходим дополнительный API-сервер, который будет заниматься обработкой запросов и обращаться к Firestore по мере необходимости. Это позволит избежать проблем, связанных с невозможность выполнения некоторых операций на компьютере конечного пользователя. Так как самые ресурсозатратные операции выполняет Firebase, данный сервер может выполнять небольшой список операций, что очень полезно для данного проекта.

2.4 Выбор языка программирования

Так как было решено решено разработать веб-приложение, разработку необходимо вести на языке, который имеет поддержку для выполнения внутри браузера. К таковым можно отнести: JavaScript, TypeScript, Kotlin. По своей сути все варианты языков это JavaScript, так как браузер работает именно с ним, но выбор того или иного языка может сказаться на процессе разработки и дальнейшей поддержке.

Kotlin является наиболее молодым из тройки и из-за данного обстоятельства может вызвать трудности у разработчиков не знакомых с проектом, так как на текущий момент основное применение kotlin находит в android разработке. Помимо этого механизм работы kotlina для веб-разработки создаёт лишние трудности, ввиду необходимости трансляции кода на Kotlin в рабочий JavaScript код.

TypeScript - язык программирования от компании Microsoft, как альтернатива JavaScript, расширяющая его возможности. Основной особенностью TypeScript можно считать систему типизации, так как в отличии от обычного JavaScript, она статическая. Однако реализация типизации в TypeScript не решает проблемы динамической типизации JavaScript полностью, так как всё равно возможны случаи, когда переменная может иметь тип отличный от

объявленного. Также стоит отметить, что многие возможности TypeScript в том или ином виде приходят и в JavaScript, что делает TypeScript не лучшим выбором в долгосрочной перспективе.

2.5 JavaScript

Таким образом был сделан выбор в сторону языка JavaScript. Основной целью JavaScript является придание статическим HTML страницам динамичности, например отправка форм без перезагрузки страницы, изменение разметки страницы и стилей элементов на лету и многое другое. Программы написанные на JavaScript называются скриптами, они представляют из себя простой текстовый файл. Для работы JavaScript скриптов не нужна дополнительная подготовка или компиляция для работы. За выполнение JavaScript кода отвечает отдельная программа, которую обычно называют движком или «виртуальной машиной JavaScript». За счёт это JavaScript скрипты можно выполнять не только в браузере, но и на сервере при наличии JavaScript движка, который займется выполнением кода. Движок берёт код из скрипта, преобразует его в машинный код и затем выполняет его.

Современные браузеры в своём составе имеют собственные встроенные JavaScript движки для выполнения скриптов, например V8 в Chrome или SpiderMonkey в FireFox[1]. Хоть в основном реализации различных JavaScript движком похожи, но в них всё равно присутствуют различия, которые способны сказаться на работе JS кода в разных браузерах. Также могут быть различия в разных версиях одного и того же движка. Подобные нюансы необходимо учитывать при разработке веб-приложений. Для решения подобных проблем в JavaScript присутствуем механизм полифилов, позволяющий использовать новые и недоступные функции, на разных или старых браузерах. У JavaScript имеется собственный открытый стандарт ECMAScript, который должен быть реализован движками, но данный процесс происходить не моментально.

JavaScript отличается от многих современных языков программирования. Эти отличия многими воспринимаются неоднозначно, так как они способны предложить богатую функциональность, но также эти отличия могут стать причиной непредвиденных неполадок в работе приложения. К таким отличиям можно отнести систему наследования. В отличии многих других языков, в которых для наследования используется концепция классов, в JavaScript наследование опирается на понятие прототипа. Прототип — это объект, который имеет набор свойств и методов. Это значит, что объекты одного типа

могут иметь разный набор свойств. Даже встроенные типы, структуры данных и функции представляют из себя обычные объекты с определенным набором свойств и функций. Для получения возможностей массива объекту достаточно указать объект, реализующий методы массива как прототип. Также при помощи данной особенность можно расширить возможности базовых типов, расширяя их прототипы. В стандарте ECMAScript 6 ввели синтаксис для классического описания классов, но данное нововведение является синтаксическим сахаром, так как никак не меняет механизмы наследования, используемые в JavaScript.

Другой особенностью JavaScript является контекст выполнения кода (значение переменной this). Данная переменная ссылается на текущий объект при выполнении кода (окно браузера или конкретный объект).То, какой объект является текущим, может вызвать трудности. Если в других языках при выполнении вложенных методов классов переменная, ссылающаяся на основной объект, остаётся прежней (self в Python), то в подобной ситуации в JavaScript произойдёт смена контекста выполнения на глобальный. В результате этого произойдёт потеря текущего объекта при выполнении функции, что может привести к непредвиденным ошибкам. На данный момент для решения данной проблемы в стандарте ECMAScript 6 ввели стрелочные функции. Их основной особенностью является то, что данные функции сохраняю контекст, в котором их вызывают.

Также среди возможностей JavaScript можно выделить:

- объекты с возможностью интроспекции;
- функции как объекты первого класса;
- автоматическое приведение типов;
- автоматическая сборка мусора;
- анонимные функции.

2.6 Построение пользовательского интерфейса

Далее встаёт вопрос визуализации пользовательского интерфейса. Любой сайт представляет из себя набор HTML документов, CSS стилей и JS скриптов. HTML, CSS отвечают за построение, структуру и визуальное оформление содержимого сайта. Хоть для создания динамического веб-приложения можно использовать только HTML, CSS и JavaSript, это подход вызовет дополнительные трудности при разработке большого количества динамический элементов и их дальнейшей поддержке. Поэтому для создания пользовательского интерфейса было решено использовать специализированные для этого

библиотеки и фреймворки. Среди них можно выделить:

– AngularJS
– VueJS
– ReactJS

От AngularJS было решено отказаться по некоторым причинам: слишком большие размеры фреймворка, вызванные тем, что Angular предлагает много функциональных возможностей, однако эти возможности излишни для данного дипломного проекта. Также, в сравнении с React и Vue, Angular обладает проблемами с производительностью, сложен в тестировании и отладке кода из-за своих архитектурных решений.

Vue и React разделяют многие общие идеи и функции. В данном случае выбор библиотеки обоснован личными предпочтениями разработчика. React обладает большей поддержкой из-зв своей популярности и предлагает больше возможностей чем Vue, оставаясь небольшим по размеру. Также использование React может облегчить дальнейшую разработку мобильной версии приложения, благодаря React Native. Эти две библиотеки, хоть и предназначены для разных вещей (веб и мобильная разработка соответственно), обладают схожей архитектурой и принципами работы. Этот факт является крайне полезным при портировании существующего веб-приложения на мобильные платформы IOS и Android.

2.7 ReactJS

ReactJS - JavaScript библиотека для создания пользовательских интерфейсов, разработанная компанией Facebook. React обладает хорошей производительностью и архитектурой для создания динамичных веб-приложений. Среди основных преимуществ стоит выделить: виртуальный DOM и JSX.

2.7.1 Виртуальный DOM

Основной концепцией и архитектурной особенностью React является виртуальный DOM[2]. Для взаимодействия с HTML документом используется Document Object Model — платформо независимый интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML, XML документов, модифицировать их содержимое и оформление. Элементы документа представляются в качестве узлов со связями "родитель-потомок". Проблема DOM заключается в его неспособности эффективно работать с современными динамическими пользовательскими интерфейсами,

так как обработка большого количества элементов может занимать слишком много времени. Данную проблему можно решить, используя различные ухищрения, но глобально это не решает проблемы разработки современных динамических интерфейсов. Решить эту проблему призвана концепция виртуального DOM. Виртуальный DOM не является общепринятой технологией или стандартом, это подход, который заключается в том, что взаимодействия происходят с DOM не напрямую. Код оперирует не DOM, а его легковесной копией. После обновления копии происходит процесс согласования реального DOM и виртуального. Во время данного процесса происходит сравнение текущего дерева элементов и нового, затем для изменённых элементов происходит перестроение и изменение структуры поддерева DOM. React берет на себя процесс сравнения деревьев, стараясь обеспечить наилучшую производительность и скорость. Однако React не решает всех проблем автоматически. Разработчику необходимо контролировать поток данных и проектировать интерфейс так, чтобы изменяемые элементы не имели больших поддеревьев DOM. Это позволит избежать излишних перестроек реального DOM, что значительно увеличивает производительность приложения.

2.7.2 JSX

Для описание пользовательского интерфейса React используют JSX — расширение языка JavaScript. Он напоминает стандартный язык шаблонов, но обладает всей силой JavaScript. React не разделяет разные компоненты приложения, такие как представления и логика. Вместо этого React использует абстрактную структуру "компонент которая объединяет в себе логику пользовательского интерфейса и сам интерфейс. Компонент является представлением DOM элементов в его виртуальной копии. При работе приложения JSX преобразуется в специальную функцию для создания DOM элемента, которая затем транслируется при помощи Babel в обычный JavaScript.

2.8 Технологии для серверной части

Ни одно веб-приложение невозможно без серверной части, которая отвечает за работу с базой данных, обработку данных и выполнение роли API, и данный проект не исключение. При выборе средств разработки основными критериями были: знакомство разработчика с данными инструментами, возможность быстрого создания минимально необходимой функциональности, без необходимости поддержки большой кодовой базы, легковесность, возможность расширения в дальнейшем при наличии необходимости. Наилучшим выбором оказался язык программирования Python и библиотека Flask.

2.8.1 Python

Python - высокоуровневый, многоплатформенный язык программирования общего назначения, является одним из наиболее востребованных языков, используемых при создании приложений, системных средств, в различных научных целях и веб-разработке[3]. Данный язык ориентирован на повышение скорости разработки и читаемость кода. Среди достоинств и особенностей языка можно выделить:

- скорость разработки;
- динамическая типизация;
- множество сторонних модулей и библиотек;
- высокая масштабируемость;
- многочисленное сообщество;
- встраиваемость.

Python поддерживает различные парадигмы программирования, среди них: структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Python является динамически типизированным языком программирования, это означает что тип переменной определяется в момент присваивания значения этой переменной. Данная особенность позволяет ускорить процесс разработки при работе с данными переменных типов и изменяющемся окружении[4]. Другим важным плюсом Python является собственный пакетный менеджер, при помощи которого можно добавить сторонние модули для расширения возможностей языка, например модуль numpy добавляет множество классов и функций для выполнения математических исследований. Одним из главных преимуществ Python для веб-разработки является наличие большого количества популярных и функциональных библиотек. В сравнении со многими другими языками, например Java или C#, Python сочетает в себе как продвинутые возможности для разработки веб-приложений, так и простоту их создания, не нуждаясь в громоздких и сложных фреймворках.

При выборе были рассмотрены и недостатки языка, важнейшим из которых был недостаток в скорости. Python является языком с полной динамической типизацией, автоматическим управлением памятью. Если на первый взгляд это может казаться преимуществом, то при разработке программ с повышенным требованием к эффективности, Python может значительно проигрывать по скорости своим статическим братьям (C/C++, Java, Go). Что касается динамических собратьев (PHP, Ruby, JavaScript), то здесь дела обстоят намного лучше, Python в большинстве случаев выполняет код быстрее за счет предварительной компиляции в байт-код и значительной части стандартной

библиотеки, написанной на C.

2.8.2 Flask

В качестве библиотеки для реализации серверной части был выбран Flask. Flask позиционирует себя как микро-фреймворк с возможностью к расширению. Это выражается в том, что сама библиотека содержит очень небольшой набор базовых возможностей, а при необходимости тех или иных дополнительных функций нужно использовать сторонние совместимые инструменты. В противовес данному подходу можно выделить другую крайне популярную Python библиотеку — Django. Она предлагает разработчику практически полный набор всего необходимо для написания веб-приложения, при этом накладывая ряд архитектурных ограничений. Однако для того, чтобы получить необходимую для данного проекта функциональность (API-сервер) в Django, необходимо использовать дополнительную библиотеку — Django REST. Она добавляет полноценный возможности REST API-сервера. В совокупности Django и Django REST требуют крайне много ресурсов на разработку и поддержание, особенно при требовании небольшой функциональности. Поэтому Flask оказался предпочтительнее из-за своего размера и расширяемости.

3 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Индивидуальное задание состоит из нескольких пунктов:

– Ознакомиться с условиями работы на предприятии, внутренним распорядком, техникой безопасности.

– Изучить новейшие возможности .NET Framework 4.5, углубить познания ASP.NET MVC 4.0, ознакомиться с новыми возможностями C# 5 и F# 3.

– Разработать сервис отправки уведомлений на мобильные устройства под управлением iOS. Разработать систему эмуляции.

– Подготовить отчет по преддипломной практике и подготоить главу пояснительной записки дипломного проекта.

4 ЭТАПЫ ВЫПОЛНЕНИЯ ЗАДАНИЯ

а) Подготовительный этап

1) разработка формальных требований;

2) разработка архитектуры приложения;

3) изучение документации по Apple Push Notification Service.

б) Этап реализации

1) разработка библиотеки отправки уведомлений, язык C#;

2) разработка сервиса, предоставляющего функции отсылки уведомлений, язык C#;

3) разработка эмулятора Apple Push Notification Service, язык F#.

в) Этап тестирования

1) тестирования корректности работы сервиса;

2) тестирование производительности сервиса при работе с эмулятором и с реальным APNS-сервисом;

3) тестирование клиентов, работающих с разработанным сервисом;

4) интеграционное тестирование цепочки «серверное приложение» — «разработанный сервис» — «Apple Push Notification Service» — «FOREXTrader for iPhone™».

г) Этап развертывания

1) развертывание сервиса в среде заказчика.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Современный учебник JavaScript [Электронный ресурс]. — Электронные данные. — Режим доступа: http://learn.javascript.ru/. — Дата доступа: 08.05.2013.

2. Документация React [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.reactjs.org/docs/getting-started.html. — Дата доступа: 08.05.2013.

3. Micha Gorelick, Ian Ozsvald. High Performance Python / Ian Ozsvald Micha Gorelick. — O'Reilly Media, 2014. — 370 с.

4. David Beazley, Brian K. Jones. Python Cookbook / Brian K. Jones David Beazley. — O'Reilly Media, 2013. — 706 с.

ПРИЛОЖЕНИЕ А
Исходный код системы

```
import React, { Component } from 'react';
import { MdPlayArrow, MdPause, MdFullscreen, MdQueue } from 'react-icons/
md';
import './player.scss';
import TimeIndicator from './TimeIndicator';
import RangeBar from './Slider';
import Hotkeys from 'react-hot-keys';
import VolumeSlider from './VolumeSlider';
import { PlayerBackend } from '../PlayerBackend/default';
import PlayList from './PlayList';
import Notifications from './Notifications';
import Chat from './Chat';

const pauseIcon = <MdPause />;
const playIcon = <MdPlayArrow />;

class Player extends Component {
    constructor(props) {
        super(props);
        this.processor = new PlayerBackend();
        this.playerElemetn = React.createRef();
        this.state = {
            isFullscreen: false,
            playbackIcon: playIcon,
            playbackState: false,
            playlist: false,
            showVideoChooser: false,
            showChat: false,
        }
    }

    componentDidMount() {
        this.vid_title = document.getElementById('video-title');
        this._connectToPlayerBackend();
        document.onfullscreenchange = () => this.setState({ isFullscreen: !
this.state.isFullscreen });
        if (this.props.externalSetUp) {
            for (const func of this.props.externalSetUp) {
                func(this.processor);
            }
        }
    }

    _connectToPlayerBackend = () => {
        this.processor.video = document.getElementById('video-player');
        this.setState({ time: this.processor.time, volume: this.processor.
volume })
```

```
        this.processor.video.addEventListener('timeupdate', () => this.
setState({ time: this.processor.time }));
        this.processor.video.addEventListener('volumechange', () => this.
setState({ volume: this.processor.volume }));

        for (const event in this.props.events) {
            this.processor.addPreAction(this.props.events[event], event);
        }
    }

    playback = () => {
        this.processor.dispatch('changePlayback', this.processor.paused);
        this.processor.dispatch('setTime', this.processor.time);
        let newIcon = this.processor.paused ? pauseIcon : playIcon;
        this.setState({ playbackIcon: newIcon, playbackState: this.
processor.paused });
    }

    fullscreen = () => {
        if (this.state.isFullscreen) {
            this.leaveFullscreen();
        } else {
            this.enterFullscreen();
        }
    }

    enterFullscreen = () => {
        const elem = this.playerElemetn.current;
        if (elem.requestFullscreen) {
            elem.requestFullscreen();
        } else if (elem.mozRequestFullScreen) {
            elem.mozRequestFullScreen();
        } else if (elem.webkitRequestFullscreen) {
            elem.webkitRequestFullscreen();
        } else if (elem.msRequestFullscreen) {
            elem.msRequestFullscreen();
        }
    }

    leaveFullscreen = () => {
        if (document.exitFullscreen) {
            document.exitFullscreen();
        } else if (document.mozCancelFullScreen) {
            document.mozCancelFullScreen();
        } else if (document.webkitExitFullscreen) {
            document.webkitExitFullscreen();
        } else if (document.msExitFullscreen) {
            document.msExitFullscreen();
        }
    }

    setTime = (timeValue) => {
        if (!timeValue.isNaN) {
```

```
            this.processor.dispatch('setTime', timeValue * this.processor.
duration);
        }
    }

    setVolume = (volumeValue) => {
        this.processor.dispatch('setVolume', volumeValue);
    }

    changeMute = () => {
        this.processor.dispatch('changeMute');
    }


    togglePlaylist = () => {
        this.setState({ playlist: !this.state.playlist });
    }

    toogleChat = (e) => {
        this.setState({ showChat: !this.state.showChat });
    }

    render() {
        return (
            <div className='player-wrapper' id='player' ref={this.
playerElemetn}>
                <Hotkeys keyName='space' onKeyUp={this.playback} />
                <Hotkeys keyName='c' onKeyUp={this.toogleChat} />
                <div className='player-layer'>
                    <video id='video-player' />
                </div>

                <div className='player-layer mouse-show'>
                    <div id='player-controls' className='player-controls'>
                        <div className='player-controls-row'>
                            <RangeBar value={this.processor.timeProgress}
handle_change={this.setTime} style={{ 'margin-top': '5px', 'margin-bottom'
: '5px' }} />
                        </div>
                        <div className='player-controls-row'>
                            <div onClick={this.playback} className="player-
button player-button-play">
                                {this.state.playbackIcon}
                            </div>
                            <VolumeSlider volume={this.state.volume}
volumeHandler={this.setVolume} toggleMute={this.changeMute} />
                            <TimeIndicator time={this.processor.time}
duration={this.processor.duration} />
                            <div className='player-button' onClick={this.
togglePlaylist}>
                                <MdQueue />
                            </div>
                            <div id='fullscreen-btn' onClick={this.
```

```
    fullscreen} className="player-button player-button-fullscreen">
                                        <MdFullscreen />
                                    </div>
                                </div>
                            </div>
                        </div>
                        <Notifications />
                        <PlayList processor={this.processor} visibility={this.state
    .playlist} playlistLogic={this.props.playlistLogic} libraryLogic={this.
    props.libraryLogic} toogleVideoChooser={this.toogleVideoChooser} />
                        {this.state.showChat &&
                            <div className='player-layer'>
                                <Chat hide={this.toogleChat}/>
                            </div>
                        }
                    </div>
                );
            }
        }

        export default Player;

        class PlayerBackend {
    constructor(video) {
        this._video = video;
        this._preActions = {};
        this._actions = {
            'changePlayback': this.changePlayback,
            'setTime': this.setTime,
            'setVolume': this.setVolume,
            'changeMute': this.changeMute,
            'setSrc': this.setSource
        }
    }

    set video(value) {
        this._video = value;
        this._video.addEventListener('error', (e) => { console.error('error', e)
      });
        this._video.addEventListener('waiting', (e) => { console.error('waiting',
        e) });
        this._video.addEventListener('stalled', (e) => { console.error('stalled',
        e) });
    }

    get video() {
        return this._video;
    }

    addPreAction = (value, action) => {
        if (this._preActions[action] === undefined) {
            this._preActions[action] = []
        }
```

```
        this._preActions[action].push(value);
    }

    resetPreActions() {
        this._preActions = {}
    }

    resetPostActions() {
        this._preActions = {}
    }

    // Playback
    _play = () => {
        this._video.play().catch((e) => console.error('playback', e));
    }

    _pause = () => {
        this._video.pause();
    }

    get paused() {
        return this._video.paused;
    }

    changePlayback = (paused) => {
        if (paused !== undefined) {
            paused ? this._play() : this._pause();
        } else {
            this.paused ?  this._play() : this._pause();
        }
    }

    // Volume
    set volume(value) {
        this._video.volume = value;
    }

    setVolume = (value) => this._video.volume = value;

    get volume() {
        return this._video.volume;
    }

    get muted() {
        return this._video.muted;
    }

    changeMute = () => {
        this._video.muted = !this._video.muted;
    }

    // Time
    setTime = (value) => this._video.currentTime = value;
```

```
  set time(value) {
      this._video.currentTime = value;
  }

  setTime = (value) => this._video.currentTime = value;

  get time() {
      return this._video ? this._video.currentTime || 0 : 0;
  }

  get timeProgress() {
      return this._video ? this._video.currentTime / this._video.duration || 0
    : 0;
  }

  get duration() {
      return this._video ? this._video.duration || 0 : 0;
  }

  //  Src
  set source(value) {
      this._video.src = value;
  }

  setSource = (value) => this._video.src = value;

  dispatch = (action, value) => {
      let _preActions = this._preActions[action] || [];
      const _actionWrapper = promiseWrapper(this._actions[action], value);

      let promise = new Promise(resolve => resolve());
      for (const _preAction of _preActions) {
          promise = promise.then(promiseWrapper(_preAction, value));
      }

      promise.then(promiseWrapper(this._actions[action], value));
  }
}

changePlayback = (roomId, paused) => {
  let room_ref = this.db.collection('rooms').doc(roomId);
  room_ref.update({ paused })
  return room_ref.get().then(() => 'skip');
}

listenPlayback = (roomId, dispatcher) => {
  return this.db.collection('rooms').doc(roomId).onSnapshot(snap => {
    if (snap.data()) {
      dispatcher.firebaseData.paused = snap.data().paused;
      dispatcher.changePlayback(!snap.data().paused);

    }
  });
```

```
}

getPlayback = (roomId, dispatcher) => {
 let roomRef = this.db.collection('rooms').doc(roomId);
 return roomRef.get().then(value => dispatcher.changePlayback(!value.data().
    paused));
}

setTime = (roomId, time) => {
 let room_ref = this.db.collection('rooms').doc(roomId);
 room_ref.update({ time })
 return room_ref.get().then(() => 'skip');
}

listenTime = (roomId, dispatcher) => {
 return this.db.collection('rooms').doc(roomId).onSnapshot(snap => {
   if (snap.data()) {
     if (dispatcher.firebaseData.time !== snap.data().time || dispatcher.
    firebaseData.src != snap.data().src) {
       dispatcher.firebaseData.time = snap.data().time;
       dispatcher.setTime(snap.data().time)
     }

   }
 });
}

getTime = (roomId, dispatcher) => {
 let roomRef = this.db.collection('rooms').doc(roomId);
 return roomRef.get().then(value => dispatcher.setTime(value.data().time || 0)
    );
}

listenSrc = (roomId, dispatcher) => {
 return this.db.collection('rooms').doc(roomId).onSnapshot(snap => {
   if (snap.data()) {
     if (dispatcher.firebaseData.src !== snap.data().src) {
       dispatcher.firebaseData.src = snap.data().src;
       dispatcher.setSource(snap.data().src)
     }
   }
 });
}

setSrc = (roomId, src) => {
 let room_ref = this.db.collection('rooms').doc(roomId);
 let roomInfoRef = this.db.collection('roominfo').doc(roomId);
 room_ref.update({ src })
 roomInfoRef.update({ src });
 return room_ref.get().then(() => 'skip');
}

listenPlaylist = (roomId, controller) => {
```

```
 let playlistRef = this.db.collection('playlists').doc(roomId);
 return playlistRef.onSnapshot(snap => {
   if (snap.data()) {
     controller(snap.data().videos)
   }
 })
}

addItemsToPlaylist = (playlistId, items) => {
 let playlistRef = this.db.collection('playlists').doc(playlistId);
 return playlistRef.update({ videos: firebase.firestore.FieldValue.arrayUnion
   (...items) });
}

removeItemsFromPlaylist = (playlistId, items) => {
 let playlistRef = this.db.collection('playlists').doc(playlistId);
 playlistRef.update({
   videos: firebase.firestore.FieldValue.arrayRemove(...items)
 })
}

setPlaylist = (playlistId, playlist) => {
 let playlistRef = this.db.collection('playlists').doc(playlistId);
 return playlistRef.update({ videos: playlist });
}

addUserToRoom = (roomId) => {
 let user = this.auth.currentUser;
 console.log('user', user)
 if (user) {
   let statsRef = this.db.collection('stats').doc(roomId);
   statsRef.update({
     users: firebase.firestore.FieldValue.arrayUnion({ id: user.uid, status:
   null })
   });
 }
}

removeUserFromRoom = (roomId) => {
 let user = this.auth.currentUser;
 if (user) {
   let statsRef = this.db.collection('stats').doc(roomId);
   statsRef.update({
     users: firebase.firestore.FieldValue.arrayRemove({ id: user.uid, status:
   null })
   });
 }
}

createRoom = (name, usePassword, password) => {
 if (!this.auth.currentUser.isAnonymous) {
   return this.db.collection('roominfo').where('owner', '==', this.auth.
   currentUser.uid).get().then((querySnap) => {
```

```
      if (querySnap.size < 1) {
        return this.db.collection('roominfo').add({
          name,
          usePassword,
          password,
          users: [],
          hidden: false,
          src: '',
          owner: this.auth.currentUser.uid,
        }).then(docRef => {
          this.db.collection('rooms').doc(docRef.id).set({ paused: true, src: '
    ', time: 0, })
          this.db.collection('messages').doc(docRef.id).set({ messages: [] });
          this.db.collection('videos').doc(docRef.id).set({ videos: [] });
        });
      } else {
        throw Error("You can't create more rooms")
      }
    })

  }
}

listenRooms = (controller) => {
  return this.db.collection('roominfo').onSnapshot(snap => {
    let rooms = [];
    snap.forEach(doc => {
      rooms.push({ ...doc.data(), id: doc.id });
    })
    controller(rooms);
  });
}

listenRoom = (roomId, controller) => {
  return this.db.collection('roominfo').doc(roomId).onSnapshot(snap => {
    if (snap.data()) {
      controller(snap.data());
    }
  })
}

getRoom = (roomId) => {
  return this.db.collection('roominfo').doc(roomId).get().then(doc => {
    return doc.data();
  });
}

getUserRooms = (userId) => {
  return this.db.collection('roominfo').where('owner', '==', userId || '').get
    ().then((querySnap) => {
    const rooms = []
    querySnap.forEach(doc => {
      rooms.push(doc.data());
```

```
    })
    return rooms;
  })
}

enterRoom = (roomId, password) => {
  let docRef = this.db.collection('roominfo').doc(roomId)
  return docRef.get().then(doc => {
    let result = true;
    if (doc.data().usePassword && password !== doc.data().password) {
      result = false;
    }
    return result;
  })
}

findRoom = (query) => {
  return this.db.collection('roominfo').get().then(snap => {
    let result = [];

    snap.forEach(docRef => {
      if (docRef.data().name.includes(query)) {
        result.push(docRef.data());
      }
    });

    return result;
  })
}

listenChat = (roomId, controller) => {
  return this.db.collection('messages').doc(roomId).onSnapshot(snap => {
    controller(snap.data().messages);
  });
}

class FirebasePlayer extends PureComponent {
    constructor(props) {
        super(props);
        if (props.roomId) {
            this.events = {
                'changePlayback': this.firebasePlayback,
                'setTime': this.firebaseTime,
                'setSrc': this.firebaseSource,
            }
            this.setUp = [this.setUpFirebaseData, this.firebaseListenSource,
    this.firebaseListenPlayback, this.firebaseListenTime]
            this.libraryLogic = {
                'getLibraryItem': this.firebaseGetLibraryItem
            }
            this.playlistLogic = {
                'listenPlaylist': this.firebaseListenPlaylist,
                'addItems': this.firebaseAddItemsToPlaylist,
```

```
                'removeItems': this.firebaseRemoveItemsFromPlaylist,
                'setPlaylist': this.firebaseSetPlaylist,
            }
        }
    }

    componentDidMount() {
        if (this.props.roomId) {
            this.props.firebase.sign_in_anon().then(() => {
                this.props.firebase.addUserToRoom(this.props.roomId);
            })
            window.addEventListener('beforeunload', () => this.props.firebase.
removeUserFromRoom(this.props.roomId));

        }
    }

    componentWillMount() {
        if (this.props.roomId) {
            window.removeEventListener('beforeunload', () => this.props.
firebase.removeUserFromRoom(this.props.roomId));
            this.props.firebase.removeUserFromRoom(this.props.roomId);

        }
    }

    firebaseSetPlaylist = (playlist) => {
        return this.props.firebase.setPlaylist(this.props.roomId, playlist);
    }

    firebaseRemoveItemsFromPlaylist = (items) => {
        return this.props.firebase.removeItemsFromPlaylist(this.props.roomId,
items);
    }

    firebasePlayback = (value) => {
        return this.props.firebase.changePlayback(this.props.roomId, !value)
    }

    firebaseListenPlayback = (dispatcher) => {
        return this.props.firebase.listenPlayback(this.props.roomId,
dispatcher);
    }

    fetchPlayback = (dispatcher) => {
        return this.props.firebase.getPlayback(this.props.roomId, dispatcher);
    }

    firebaseTime = (value) => {
        return this.props.firebase.setTime(this.props.roomId, value)
    }

    firebaseListenTime = (dispatcher) => {
```

```
        return this.props.firebase.listenTime(this.props.roomId, dispatcher);
    }

    setUpFirebaseData = (dispatcher) => {
        dispatcher.firebaseData = {}
    }

    firebaseSource = (value) => {
        return this.props.firebase.setSrc(this.props.roomId, value);
    }

    firebaseListenSource = (dispatcher) => {
        return this.props.firebase.listenSrc(this.props.roomId, dispatcher);
    }

    firebaseGetLibraryItem = (id) => {
        return this.props.firebase.getLibraryItem(id).then(value => value)
    }

    firebaseListenPlaylist = (controller) => {
        return this.props.firebase.listenPlaylist(this.props.roomId,
    controller)
    }

    firebaseAddItemsToPlaylist = (items) => {
        return this.props.firebase.addItemsToPlaylist(this.props.roomId, items
    )
    }

    render() {
        return (
            <div className='f-player'>
                <Player events={this.events} externalSetUp={this.setUp}
    libraryLogic={this.libraryLogic} playlistLogic={this.playlistLogic} />
            </div>
        )
    }
}
```