Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА ГИБРИДНОГО КРОССПЛАТФОРМЕННОГО ПРИЛОЖЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Шевченко Артема Александровича

Научный руководитель
к. ф.-м. н. 10.06.17 С. В. Миронов

Заведующий кафедрой
к.ф.-м.н. 10.06.17 С. В. Миронов

Саратов 2017

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

MVC (англ. Model View Controller) — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер.

JS (англ. JavaScript) — язык сценариев для обработки событий браузера;

SQL (англ. Structured Query Language) — язык для управления реляционными базами данных;

HTTP (англ. HyperText Transfer Protocol) — протокол передачи гипертекста;

URL (англ. Uniform Resource Locator) — единообразный локатор (определитель местонахождения) ресурса;

CORS (англ. Cross-origin resource sharing) — технология совместного использования ресурсов между разными источниками;

API (англ. application programming interface) — программный интерфейс приложения;

RIA (англ. Rich Internet Applications) — насыщенное интернет приложение;

СУБД — система управления базами данных.

ВВЕДЕНИЕ

Сегодня все больше приложений создается сразу для нескольких платформ, а приложения, созданные изначально для одной платформы, активно портируются на другие, но как правило для портирования требуется знания языка программирования для реализации на данной операционной системе, что делает разработку более затруднительной, увеличивает её сроки, а так же увеличивает стоимость в зависимости от условий реализации.

В большинстве случаев понятие кроссплатформенности перетекает в многоплатформенность. При этом не существует качественного решения, которое базировалось бы на одном языке и могло бы запуститься сразу на нескольких платформах без установки пользователем дополнительного специализированного программного обеспечения для запуска таких приложений.

Идея создания приложений, способных работать на нескольких платформах является далеко не новой. Компания Microsoft пыталась решить проблему разработки под Windows, Xbox и Windows Mobile 7 при помощи .NET и XNA. Но данный подход не является универсальным, т.к. количество платформ ограничено и чаще всего относится к одному семейству операционных систем, базирующихся на одном ядре.

Приложения реализованные на языке Java запускаются практически на любой платформе, но требуют от пользователей установки виртуальной машины. Такого рода программы не являются рабочими «из коробки», что доставляет неудобство тем, кто их использует.

Идея настоящей работы возникла в процессе реализации кроссплатформенного приложения, предназначенного для оповещения о ближайших мероприятиях. Целью данной работы является исследование возможностей платформы разработки мобильных и десктопных приложении на основе гибридной архитектуры. Для достижения этои цели необходимо:

— реализовать решение для развертки серверного приложения;
— реализовать серверное приложение для обеспечения работы клиентской части;
— для демонстрации кроссплатформенности реализовать версии приложения для операционных систем Android и iOS.

1 Теоретические сведения

1.1 Гибридное приложение

Для многих отраслей деятельности человека мобильные устройства стали критически важным. Когда предприятие управляет своим брендом через веб-приложение, оно сталкивается с проблемой доставки разнообразной информации как с сервера на мобильные устройства, так и с мобильных устройств на сервер. Базовая концепция гибридного приложения заключается в совместной работе серверного веб-приложения и клиентского мобильного (нативного — native application shell) приложения. Основное преимущество их совместной работы - возможность веб-браузера получить доступ к таким возможностям клиентских мобильных устройств, как камера, акселерометр, контакты, файловая система или геопозиционирование. Шейн Черч — директор отдела эффективных технологий в Microsoft приводит в источнике [1] пример внедренного гибридного приложения. Приложение предназначено для мобильных сотрудников, обрабатывающих заказы на техническое обслуживание и ремонт различного муниципального имущества, такого как рекламные щиты, скамейки и пожарные гидранты. Для определения текущего местонахождения сотрудника оно использует преимущества функций, поддерживаемых браузером, а для получения снимков муниципального имущества и их последующей загрузки на сервер — доступ к аппаратным средствам конкретного мобильного устройства.

Гибридные приложения относятся к классу RIA-приложений. Таким образом, основное отличие работы гибридных приложений от мобильных вебсайтов состоит в уходе от клиент-серверной архитектуры, при которой браузер являлся тонким клиентом. При этом запускается полноценное мобильное приложение, для которого взаимодействие с сервером носит не основной характер. По сути, это приложения, работающие через браузер, но обладающие функциональностью полноценных десктопных приложений. Для обмена данными между веб-приложениями и нативным кодом мобильных устройств существуют фреймворки, которые выступают в роли моста и обеспечивают общий интерфейс, позволяющий получить доступ непосредственно к модулям операционной системы данного устройства. К примеру к фотокамере, push-уведомлениям или геолокации. Примерами популярных технологий для разработки мобильных RIA служат Microsoft Silverlight, Flash/Flex от Adobe,

JavaFX от Sun и Apache Cordova (старое название до поглощения Apache — Phone Gap), которая базируется на связке веб-технологий.

Согласно источнику [2], гибридное приложение для бизнеса может обеспечить существенную экономию расходов по сравнению с уникальными нативными приложениями — как в краткосрочной перспективе, так и в долгосрочной. В случае мобильных устройств, число которых быстро растет, у разработчика есть несколько вариантов создания такого рода информационных систем.

Создание нативного приложения для каждой поддерживаемой платформы, безусловно, обеспечивает максимальное удобство для пользователя и производительность на каждой платформе, в то же время, открывая доступ ко всем встроенным аппаратным средствам. Однако недостаток этого подхода в том, что он может потребовать гораздо больших затрат на создание и сопровождение, так как понадобятся раздельные кодовые базы для каждой платформы, которую необходимо поддерживать.

Создание мобильного веб-приложения — самый простой и дешевый вариант для разработки, выпуска и обновления на всех платформах, а также доступа к базе данных сервера, но удобство в использовании может пострадать из-за отсутствия доступа к встроенным аппаратным средствам.

Гибридный подход представляет собой хороший компромисс между затратами на разработку уникального для каждой платформы родного приложения, возможностям доступа к встроенным аппаратным средствам и доступу к серверной базе данных. Важно отметить, что ни один из этих подходов не является принципиально лучшим других — все они имеют свои сильные и слабые стороны. Глубокий анализ соотношения выгоды по каждому варианту поможет определить, какой из них будет лучше для пользователей конкретного бизнеса. При принятии решения важно учитывать удобство в использовании, затраты на разработку и последующие расходы на сопровождение, а также более тонкие факторы вроде маркетинга и признания среди пользователей.

Для многих бизнес сценариев предпочтительнее вариант с гибридным приложением, так как дополнительные усилия и расходы на создание уникальных нативных приложений для каждой мобильной платформы могут перевесить все выгоды для бизнеса.

1.2 История развития Cordova

Фреймворк Cordova возник как решение проблем работы мобильных приложений, создаваемых на основе HTML5 [3]. Например, можно разработать мобильное веб-приложение с JavaScript [4] и HTML5 контентом, направляя пользователей на URL хост-машины. Первой проблемой станет продажа этого приложения через онлайн магазины. Нельзя отправить в такой магазин URL, по которому размещено ваше веб-приложение, вы не получите за него деньги. Вторая проблема заключается в том, как осуществляется доступ к аппаратному обеспечению мобильного устройства и его графическим возможностям. По части графики в веб-приложениях можно делать только то, что позволяют CSS/HTML/JavaScript. Некоторые функции управления четкостью изображения и пользовательским интерфейсом, доступные в среде операционной системы мобильного устройства, в пространстве браузера недоступны. У браузеров нет широко поддерживаемых API для доступа к телефонным контактам, уведомлениям, камерам, датчикам и прочему. Apache Cordova — бесплатная инфраструктура с открытым исходным кодом — решает обе эти проблемы.

Apache Cordova начинался как PhoneGap, который был разработан Nitobi. В октябре 2011 года Nitobi была приобретена компанией Adobe Systems вместе с открытым исходным кодом инфраструктуры PhoneGap по лицензии Apache Software Foundation, и название PhoneGap было сменено на Cordova. Переходный период не закончился до сих пор. Cordova предоставляет среду для хостинга разрабатываемого контента HTML5/JavaScript внутри тонкой «родной» оболочки. Для ОС на каждой платформе мобильных устройств она использует родной элемент управления «браузер» для рендеринга контента вашего клиентского приложения, причем ресурсы приложения упаковываются в дистрибутив. В случае Windows Phone ваши HTML5-ресурсы упаковываются в `.xap` - файл и загружаются в изолированное хранилище, когда запускается ваше Cordova-приложение. Фактически, Cordova-приложение представляет собой мобильный виджет. В период выполнения элемент управления WebBrowser визуализирует контент виджета и исполняет его JavaScript-код.

Cordova также предоставляет набор стандартных API для доступа к функциям, общим для всех мобильных устройств. К некоторым из этих функ-

ций относятся хранилище (локальное хранилище и базы данных HTML5), контакты, камера, геопозиционирование, акселерометр. Каждая из этих функций предоставляется как JavaScript API, который используется из необходимого кода на JavaScript. Фреймворк берет на себя всю черновую работу, связанную с предоставлением необходимой родной реализации, и тем самым гарантирует, что вы будете иметь дело с одинаковыми JavaScript API независимо от ОС мобильного устройства, на котором выполняется ваш код. Тем самым, она дает возможность выполнять одно и то же HTML5-приложение на разных операционных системах. Пример архитектуры приложения можно увидеть на рис. 1.

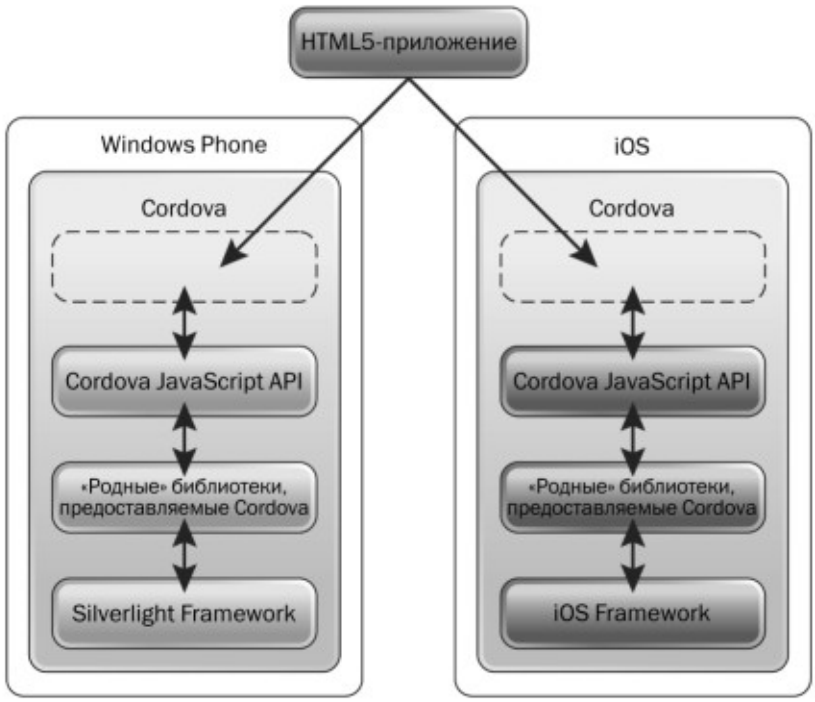

Рисунок 1 – Архитектура приложения Apache Cordova на Windows Phone и iOS

Этот фреймвок уже стал стандартом для платформ WindowsPhone, iOS, Android и многих других. Его следует воспринимать как надстройку над SDK мобильной платформы, ориентированную на разработку виджетов (клиентской части) гибридных приложений, хотя в ней с успехом можно разрабатывать и офлайновые мобильные приложения. В 2012 году авторы свободного JavaScript - фреймворка Apache Cordova сообщили о том, что их проект «повысили» до уровня ведущего (top-level projects) в некоммерческой организации Apache Software Foundation.

2 Практическая реализация

Было решено не только реализовать серверную и клиентскую части, но так же создать полную развертку практического решения, что позволяет быстро воспроизвести работу и проверить работу на любом персональном компьютере.

Для реализации развертки были использованы следующие технологии: Docker [5], Docker Compose [6], Alpine Linux [7], готовый Docker-образ для СУБД PostgreSQL.

Для быстрой и комфортной реализации серверной части, было принято использовать язык Python. Он не имеет строгой типизации, что позволяет не задумываться о типах данных, которые будут использоваться. Так же следующие технологии: микрофреймворк Flask, реляционная СУБД PostgreSQL [8] и CORS-модуль [9] для Flask [10].

В реализации клиентской части, а именно самого кроссплатформенного приложения, были использованны: фреймворк Angular 2 [11] для создания клиентских приложений, фреймворк Ionic [12] для реализации пользовательского интерфейса приложений основанных на гибридной архитектуре, язык TypeScript [13] интерпретируемый в язык JavaScript для сжатия исходного кода и упрощения реализации, технология Apache Cordova [14] для создания кроссплатформенных версий полученного приложения.

2.1 Реализация серверной части

Само приложение представляет собой сервис по оповещению о мероприятиях, которые будут организованы в ближайшее время. За посещение каждого из них, пользователю предлагается определенное количество очков, которое в дальнейшем он сможет потратить на услуги или вещи.

Для реализации серверной части приложения было решено использовать микрофреймворк Flask. Так как он является микрофреймворком, то он требует минимально конфигурации, а именно определения адреса по которому можно будет обратиться. Это делает реализованную платформу легковесной и легкой для понимания. Данный фреймворк использует MVC-модель, но для реализации решения описанного в данной работе будут не будет использована визуальная часть. Она будет предоставлять клиентское гибридное приложение.

Для организации архитектуры запросов используется предложенный в [15] стиль Rest. Он предназначен для построения распределенных масштабируемых веб-сервисов. Стандартные http методы представляются в таком виде, при котором каждый из них имеет свой смысл и свою реализацию. Этот стиль позволяет использовать методы реализованные посредствам сервера для всех клиентских приложений. Любой клиент в таком случае будет являться простым воспроизводчиком данных и не иметь никакой их логической обработки на своей стороне.

Для создания упорядоченной архитектуры приложения используется микросервисное представление описанное в книге [16]. Её смысл в делегировании обязанностей между сервисами, что позволяет контролировать рост проекта. При данной архитектуре, если серверное приложение является масштабным, то его кодом удобно управлять, так как чаще всего сервисы состоят из небольших фрагментов и всегда можно определить, к чему относится рассматриваемый сервис.

В данной работе было решено распределить сервисы по следующим категориям:

— repositories (рус. репозитории) — данный сервис содержит в себе только работу с базой данных и возвращения из неё информации;
— models (рус. модели) — описывает сущности и методы работы с ними;
— controllers (рус. контроллеры) — отвечают за получение информации посредством протокола http и передачу информации непосредственно сервисам для получения результата, а затем отправки его обратно;
— services (рус. сервисы) — содержит в себе логическую часть, т.е. является обработчиком получаемой информации из репозиториев, а так же из контроллеров.

Так же, разбиение сервисов было произведено по следующим сущностям:

— Модель, репозиторий, сервис и контроллер для работы с пользователями;
— Модель, репозиторий, сервис и контроллер для работы с мероприятиями;
— Модель, репозиторий и сервис для работы с токенами.

2.2 Реализация развертки

Данный раздел описывает реализацию системы, которая позволяет быстро запустить приложение без предварительной установки дополнительных пакетов в операционной системе, которые потребуется для поднятия серверного решения. Для запуска в системе должен быть установлен только пакет Docker. Данное решение называется «разверткой».

В целях упрощения развертки всего решения была выбрана технология контейниризации Docker. Она позволяет изолировать отдельные составляющие всего проекта и избежать не запуска сервера при переносе на другие компьютеры.

Контейнер в Docker подразумевает под собой персональное окружение для каждого процесса. Это позволяет не только упростить развертку, но и содержать всю структуру отдельной компоненты приложения в одном месте, а так же избегать засорения ненужными файлами и конфигурациями в основной системе, из которой происходит запуск данных контейнеров. Благодаря такой организации можно запускать одновременно несколько экземпляров сервера на одной машине с одними и теми же или разными конфигурациями. Ни один экземпляр не будет конфликтовать с другим.

В целях автоматизации запуска контейнеров, было решено описать конфигурацию каждого из них при помощи официального конфигуратора запуска Docker-Compose, который является отдельным решением написанным на языке Python. Это позволит избежать повторной настройки и описания при каждом запуске экземпляра серверной части, так как все команды будут описаны в одном месте и, если нужно будет поменять конкретный параметр, то достаточно будет открыть файл `docker-compose.yml` и произвести замену конкретного параметра. Не менее важно, что для запуска или выключения всего решения достаточно будет написать одну команду в терминале операционной системы.

Контейнеры были построены для следующих частей приложения:

— База данных основанная на системе управления базами данных PostgreSQL;

— Серверная часть основанная на Python и его модуле написанном на фреймворке Flask.

2.3 Реализация клиентской части

Реализация клиентской части будет базироваться на технологии Apache Cordova, как было описано ранее, данная технология позволяет использовать средства для создания web-приложений, как полноценные инструменты в реализации нативного решения для любой платформы.

Во избежание траты времени на создание и отрисовку пользовательского интерфейса используется фреймворк ionic предназначенный, как раз для такой цели в реализации гибридного приложения. Данный фреймворк имеет готовые компоненты для отрисовки интерфейса. Как пример можно привести кнопки, иконки, списки, меню, поля ввода...

Помимо элементов пользовательского интерфейса ionic предоставляет в своей комплектации фреймворк Angular 2, предназначенный для создания клиентски веб-приложений. Для написания исполняемого кода под данный фреймворк, требуется знание языка TypeScript.

После установки описанных выше средств, нужно подготовить рабочую директорию. Для этого требуется выполнить следующую команду из терминала системы:

```
1 ionic start myApp tabs
```

Данная команда создаст базовую конфигурацию для работы с Apache Cordova, а так же добавит готовые графические элементы. В данной команде содержатся следующие аргументы:

— `start` — данный аргумент сообщает ionic cli о том, что будет создаваться новый проект;

— `myApp` — имя проекта, который будет создан. Имя «myApp» является примером и может быть заменено на любое другое название;

— `tabs` — тип отображения графического меню, которое будет использовать в приложении. В данном случае выбраны вкладки (англ. Tabs).

Перейдем непосредственно к коду, который находится в директории `src/`. Так же, было решено использовать микросервисную архитектуру, так как она является удобной при дальнейшем масштабировании приложения, что позволит быстро править и добавлять новые фрагменты кода.

Приложение делится на три составляющие:

— components (рус. компоненты) — данная составляющая позволяет отрисовывать ту или иную часть приложения;

— repositories (рус. репозитории) — эти сервисы отвечают за хранение информации. В основном будет использоваться хранение в `LocalStorage` (рус. локальном хранилище);
— services (рус. сервис) — в данных файлах содержится логическая обработка информации, а так же получение информации с серверной части приложения.

ЗАКЛЮЧЕНИЕ

Apache Cordova является удобным решением для быстрого создания полноценных кроссплатформенных приложений. Более того, это решение является оптимальным, так как имеется возможность портации на такие операционные системы, как macOS, Windows, iOS, Android, Ubuntu и многие другие. Данная технология позволяет сделать быстрое и удобное клиентское решение для пользователей. Используя встроенные возможности языка JavaScript, можно организовать асинхронную работу каждой из частей приложения, что в нативных языках программирования невозможно без установки дополнительных библиотек.

В ходе настоящей работы:

— разработано приложение построенное на основе гибридной архитектуры — приложение для оповещений о предстоящих мероприятиях;
— описаны инструменты, используемые для реализации подобных решений;
— реализована серверная часть приложения;
— реализовано кроссплатформенное клиентское приложение на базе Apache Cordova;
— было проведено тестирование разработанного решения.

Разработанное приложение уже можно использовать в реализованном минимальном функционале, но возможно расширение приложения за счет добавления новых модулей (например, полное описание мероприятия, включение онлайн-магазин, ленты фотографий с мероприятий).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Разработка гибридных веб-приложений, способных использовать аппаратные средства мобильных устройств [Электронный ресурс]. — URL: https://msdn.microsoft.com/ru-ru/magazine/hh852592.aspx (Дата обращения 09.06.2017). Загл. с экр. Яз. рус.

2 Гибридные приложения как ответ на растущий спрос мобильной разработки [Электронный ресурс]. — URL: https://goo.gl/71ycfj (Дата обращения 10.06.2017). Загл. с экр. Яз. рус.

3 HTML5 — платформа, предназначенная для создания веб-приложений использующих аудио, видео, графику, анимацию и многое другое.[Электронный ресурс]. — URL: http://htmlbook.ru/html5 (Дата обращения 10.05.2017). Загл. с экр. Яз. рус.

4 JavaScript is the programming language of HTML and the Web[Электронный ресурс]. — URL: http://www.javascript.com/learn/javascript/ (Дата обращения 10.05.2017). Загл. с экр. Яз. англ.

5 Docker — isolated environment technology [Электронный ресурс]. — URL: https://www.docker.com (Дата обращения 12.05.2017). Загл. с экр. Яз. англ.

6 Docker compose — tool for defining and running multi-container Docker applications [Электронный ресурс]. — URL: https://docs.docker.com/compose/overview/ (Дата обращения 12.05.2017). Загл. с экр. Яз. англ.

7 Alpine — security-oriented, lightweight Linux distribution based on musl libc and busybox. [Электронный ресурс]. — URL: https://alpinelinux.org/about/ (Дата обращения 15.05.2017). Загл. с экр. Яз. англ.

8 PostgreSQL — open source object-relational database system [Электронный ресурс]. — URL: https://www.postgresql.org/about/ (Дата обращения 20.05.2017). Загл. с экр. Яз. англ.

9 Flask-CORS — A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible. [Электронный ресурс]. — URL: https://flask-cors.readthedocs.io/en/latest/ (Дата обращения 08.06.2017). Загл. с экр. Яз. англ.

10 Flask — open source microframework for web applications [Электронный ресурс]. — URL: http://flask.pocoo.org/docs/0.12/foreword/#what-does-micro-mean (Дата обращения 20.05.2017). Загл. с экр. Яз. англ.

11 Angular — framework for client side creation [Электронный ресурс]. — URL: https://angular.io/docs/ts/latest/ (Дата обращения 8.05.2017). Загл. с экр. Яз. англ.

12 Ionic — open source framework for building mobile apps [Электронный ресурс]. — URL: https://ionicframework.com (Дата обращения 15.05.2017). Загл. с экр. Яз. англ.

13 TypeScript — superset of JavaScript compiles to plain JavaScript [Электронный ресурс]. — URL: https://www.typescriptlang.org (Дата обращения 20.05.2017). Загл. с экр. Яз. англ.

14 Введение в Apache Cordova [Электронный ресурс]. — URL: https://cordova.apache.org/docs/ru/latest/guide/overview/ (Дата обращения 7.05.2017). Загл. с экр. Яз. рус.

15 *Richardson, L.* RESTful Web Services / L. Richardson, S. Ruby. — Sebastopol: O'Reilly Media, Inc, 2008.

16 *Newman, S.* Building Microservices / S. Newman. — Sebastopol: O'Reilly Media, Inc, 2015.

10.06.2017 г.