**МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН**

**НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛЬ-ХОРЕЗМИЙ**

**ФАКУЛЬТЕТ КОМПЬЮТЕРНОГО ИНЖИНИРИНГА**
**КАФЕДРА ПРОГРАММНОГО ИНЖИНИРИНГА**

**Направления программный инжиниринг (Программный инжиниринг)**

**Допустить к защите**
**Заведующий кафедрой**
**Утеулиев Н.У**
**2019 г. «___»_____**

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**на тему: «Система размещения заказов в кафе и ресторанах через мобильное приложение»**

**Выпускник:** **Бекмуратова С.**

**Научный руководитель:**

**НУКУС, 2019**

**Оглавление**

# ВВЕДЕНИЕ

Интернет может быть отличным ресурсом для покупателей, желающих расширить свой выбор продуктов для покупки, и является бесценным способом сэкономить деньги. Интернет-магазины очень конкурентоспособны, не только с другими интернет-магазинами, но и с конкурентами. Сайты сравнения цен облегчают поиск сделок, а также помогают покупателям обращаться в интернет-магазины с лучшей репутацией, публикуя отзывы других покупателей.

Преимущества в точках продаж включают магазины, предлагающие бесплатную доставку и бесплатные варианты доставки в магазин. Многие интернет-магазины не передают налог с продаж (если этого не требует государство) покупателям, что может дать существенную экономию тем покупателям, которые покупают в основном через Интернет.

Другие преимущества онлайн заказов включают в себя:

- Круглосуточная доступность
- Редко приходится иметь дело с агрессивными продавцами.
- Нет раздражающих очередей.

В современном электронная торговля занимает одно из ведущих мест в ведении бизнеса: она обеспечивает необходимыми товарами и услугами, объединяет континенты и страны. Благодаря ей, товар может быть доставлен в отдаленные уголки мира. В настоящее время термин «электронная коммерция» трактуется по-разному. Может, это связано с тем, что данный вид торговли появился относительно недавно, а с другой стороны, ученые-экономисты с разных аспектов рассматривают сущность данного понятия. Например, А. В. Юрасов определяет электронную коммерцию как «сферу экономики, которая включает в себя все финансовые и торговые транзакции, осуществляемые при помощи компьютерных сетей, и бизнес-процессы,

связанные с проведением таких транзакций» [2, с. 6]. Большое число людей, особенно молодежь, очень часто предпочитает покупку товаров в Интернете, нежели в традиционных магазинах. Прежде всего, это объясняется низкими ценами, а также более широким ассортиментом товара. Помимо этого затрата времени и нервов на совершение покупки посредством интернет-технологий гораздо меньше.

**Целью** данной работы является разработка системы выполнения заказов в кафе в режиме онлайн, с помощью мобильного устройства.

**Актуальность** работы проистекает из повсеместного распространения сети интернет, и популярностью автоматизированных кафе и ресторанов.

В связи с вышеизложенной целью, были сформулированы следующие **задачи**:

1. Анализ существующих технологий и систем, с помощью которых возможно построение систем онлайн заказов.

2. Исследование современных фреймворков для веб и мобильной разработки.

3. Построение архитектуры базы данных.

4. Разработка мобильного приложения.

В **первой главе** рассматриваются понятия REST API, даются основные рекоммендации по разработке RESTful приложений, рассматриваются современные веб-фреймворки, одностраничные приложения и мобильные фреймворки.

Во **второй главе** описывается работа по проектированию системы для онлайн заказов, проектируется база данных и архитектура приложения, а также описывается собственно сама разработка приложения для Android на основе фреймворка Cordova.

В **заключении** даются выводы по проделанной работе.

# 1. ОБЗОР ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ КЛИЕНТ-СЕРВЕР

## 1.1. REST API и RESTful приложения

REST - это сокращение от **RE**presentational **S**tate **T**ransfer. Это архитектурный стиль для распределенных гипермедиа систем, и он был впервые представлен Роем Филдингом в 2000 году в своей знаменитой диссертации.

Как и любой другой архитектурный стиль, REST также имеет свои 6 направляющих ограничений, которые должны быть выполнены, если интерфейс должен называться RESTful. Эти принципы перечислены ниже.

### 1.1.1. Руководящие принципы REST

1. **Клиент-сервер** - отделяя задачи пользовательского интерфейса от задач хранения данных, мы улучшаем переносимость пользовательского интерфейса на несколько платформ и улучшаем масштабируемость за счет упрощения серверных компонентов.

2. **Без состояния** - каждый запрос от клиента к серверу должен содержать всю информацию, необходимую для понимания запроса, и не может использовать какой-либо сохраненный контекст на сервере. Поэтому состояние сеанса полностью сохраняется на клиенте.

3. **Кэшируемость**- ограничения кэша требуют, чтобы данные в ответе на запрос были неявно или явно помечены как кешируемые или не кешируемые. Если ответ кешируется, то клиентскому кешу предоставляется право повторно использовать эти данные ответа для последующих эквивалентных запросов.

4. **Унифицированный интерфейс** - благодаря применению принципа общности разработки программного обеспечения к интерфейсу компонента упрощается общая архитектура системы и улучшается видимость взаимодействий. Чтобы получить унифицированный интерфейс, необходимо

несколько архитектурных ограничений для управления поведением компонентов. REST определяется четырьмя интерфейсными ограничениями: идентификация ресурсов; манипулирование ресурсами через представления; информативные сообщения; и гипермедиа как двигатель состояния приложения.

5. **Слоистая система** - стиль многоуровневой системы позволяет архитектуре состоять из иерархических уровней, ограничивая поведение компонента таким образом, что каждый компонент не может «видеть» за пределами непосредственного уровня, с которым они взаимодействуют.

6. **Код по требованию** (необязательно) - REST позволяет расширять функциональность клиента путем загрузки и выполнения кода в форме апплетов или скриптов. Это упрощает работу клиентов за счет уменьшения количества функций, необходимых для предварительной реализации.

### 1.1.2. Ресурс

Ключевая абстракция информации в REST - это ресурс. Любая информация, которая может быть названа, может быть ресурсом: документ или изображение, временная служба, набор других ресурсов, не виртуальный объект (например, человек) и так далее. REST использует идентификатор ресурса для идентификации конкретного ресурса, участвующего во взаимодействии между компонентами.

Состояние ресурса в любой конкретной временной отметке называется представлением ресурса. Представление состоит из данных, метаданных, описывающих данные, и гиперссылок, которые могут помочь клиентам в переходе в следующее желаемое состояние.

Формат данных представления известен как тип носителя. Тип мультимедиа определяет спецификацию, которая определяет способ обработки представления. Действительно RESTful API выглядит как

гипертекст. Каждая адресуемая единица информации несет адрес, либо явно (например, атрибуты link и id), либо неявно (например, полученный из определения типа носителя и структуры представления).

По словам Роя Филдинга:

*Гипертекст (или гипермедиа) означает одновременное представление информации и контроль того, что информация становится доступной, благодаря чему пользователь (или автомат) получает выбор и выбирает действия. Помните, что гипертекст не должен быть HTML (или XML или JSON) в браузере. Машины могут переходить по ссылкам, когда они понимают формат данных и типы отношений.*

Кроме того, представления ресурсов должны быть самоописательными: клиенту не нужно знать, является ли ресурс сотрудником или устройством. Он должен действовать на основе медиа-типа, связанного с ресурсом. Таким образом, на практике вы в конечном итоге создадите множество пользовательских типов медиа - обычно один тип медиа, связанный с одним ресурсом.

Каждый тип носителя определяет модель обработки по умолчанию. Например, HTML определяет процесс рендеринга для гипертекста и поведение браузера вокруг каждого элемента. Он не имеет никакого отношения к методам ресурсов GET / PUT / POST / DELETE /…, за исключением того факта, что некоторые элементы медиа-типа будут определять модель процесса, которая выглядит как «якорные элементы с атрибутом href, создают гипертекстовую ссылку, которая при выборе вызывает запрос поиска (GET) по URI, соответствующему атрибуту href в кодировке CDATA ».

### 1.1.3. Методы ресурсов

Другая важная вещь, связанная с REST, - это методы ресурсов, которые будут использоваться для выполнения желаемого перехода. Большое количество людей ошибочно связывают методы ресурсов с методами HTTP GET / PUT / POST / DELETE.

Рой Филдинг никогда не упоминал никаких рекомендаций относительно того, какой метод использовать в каких условиях. Все, что он подчеркивает, это то, что это должен быть единый интерфейс. Если вы решите, что HTTP POST будет использоваться для обновления ресурса - вместо того, чтобы большинство людей рекомендовало HTTP PUT - все в порядке, и интерфейс приложения будет RESTful.

В идеале все, что необходимо для изменения состояния ресурса, должно быть частью ответа API для этого ресурса, включая методы и в каком состоянии они будут выходить из представления.

API REST следует вводить без каких-либо предварительных знаний, кроме начального URI (закладки) и набора стандартизированных типов мультимедиа, подходящих для предполагаемой аудитории (то есть ожидается, что это поймет любой клиент, который может использовать API). С этого момента все переходы состояния приложения должны определяться выбором клиентом предоставленных сервером вариантов, которые присутствуют в полученных представлениях или подразумеваются пользовательскими манипуляциями с этими представлениями. Переходы могут быть определены (или ограничены) знаниями клиента о типах мультимедиа и механизмах обмена ресурсами, которые могут быть улучшены на лету (например, код по запросу).

Еще одна вещь, которая поможет вам при создании RESTful API, заключается в том, что результаты API на основе запросов должны быть представлены списком ссылок со сводной информацией, а не массивами

исходных представлений ресурсов, поскольку запрос не заменяет идентификацию ресурсов.

**REST и HTTP не одно и то же !!**

Многие люди предпочитают сравнивать HTTP с REST. REST и HTTP не совпадают.

REST != HTTP

Хотя REST также намеревается сделать сеть (Интернет) более упорядоченной и стандартной, он выступает за более строгое использование принципов REST. И именно здесь люди пытаются начать сравнивать REST с сетью (HTTP). Рой Филдинг в своей диссертации нигде не упомянул ни одну директиву реализации - включая любые предпочтения протокола и HTTP. Пока вы соблюдаете 6 руководящих принципов REST, вы можете назвать свой интерфейс RESTful.

Проще говоря, в архитектурном стиле REST данные и функциональность считаются ресурсами и доступны с использованием унифицированных идентификаторов ресурсов (URI). На ресурсы воздействуют с помощью набора простых, четко определенных операций. Клиенты и серверы обмениваются представлениями ресурсов с использованием стандартизированного интерфейса и протокола - обычно HTTP.

Ресурсы отделены от их представления, чтобы к их содержимому можно было обращаться в различных форматах, таких как HTML, XML, простой текст, PDF, JPEG, JSON и другие. Метаданные о ресурсе доступны и используются, например, для управления кэшированием, обнаружения ошибок передачи, согласования соответствующего формата представления и выполнения проверки подлинности или контроля доступа. И что наиболее важно, каждое взаимодействие с ресурсом не имеет состояния.

Все эти принципы помогают приложениям RESTful быть простыми, легкими и быстрыми.

### 1.1.4. Используйте вразумительные имена для ресурсов

Создание отличного API - это 80% искусства и 20% науки. Создание иерархии URL, представляющей разумные ресурсы, является художественной частью. Наличие разумных имен ресурсов (которые являются просто путями URL, такими как /customer/12345 /orders) улучшает ясность того, что делает данный запрос.

Соответствующие имена ресурсов обеспечивают контекст для запроса на обслуживание, повышая понятность API. Ресурсы рассматриваются иерархически через их имена URI, предлагая потребителям дружественную, понятную иерархию ресурсов для использования в своих приложениях.

Вот несколько быстрых правил для дизайна пути URL (имя ресурса):

- Используйте идентификаторы в ваших URL, а не в строке запроса. Использование параметров строки запроса URL отлично подходит для фильтрации, но не для имен ресурсов.

**Хорошо**: /users/12345

**Плохо**: /api?type=user&id=23

- Используйте иерархическую природу URL, чтобы подразумевать структуру.

- Дизайн для ваших клиентов, а не для ваших данных.

- Имена ресурсов должны быть существительными. Избегайте глаголов в качестве имен ресурсов, чтобы улучшить ясность. Используйте методы HTTP для указания глагольной части запроса.

- Используйте множественное число в сегментах URL, чтобы поддерживать URI API-интерфейса согласованными во всех методах HTTP с использованием метафоры коллекции.

**Рекомендуем:** /customers/33245/orders/8769/lineitems/1

**Не рекомендуем:** /customer/33245 /order/8769/lineitem/1

- Избегайте использования словосочетания в URL. Например, 'customer_list' в качестве ресурса. Используйте множественное число, чтобы указать метафору коллекции (например, customer против customer_list).

- Используйте строчные буквы в сегментах URL, разделяя слова подчеркиванием ('_') или дефисом ('-'). Некоторые серверы игнорируют регистр, поэтому лучше быть точным.

- Сохраняйте URL как можно короче, используя как можно меньше сегментов.

### 1.1.5. Используйте HTTP-коды ответов для указания статуса

Коды состояния ответа являются частью спецификации HTTP. Их достаточно для решения самых распространенных ситуаций. В духе того, чтобы наши службы RESTful включали спецификацию HTTP, наши веб-API должны возвращать соответствующие коды состояния HTTP. Например, когда ресурс успешно создан (например, из запроса POST), API должен вернуть код состояния HTTP 201. Здесь доступен список допустимых кодов состояния HTTP, в котором перечислены подробные описания каждого из них.

Предлагаемые варианты использования «10 лучших кодов состояния ответа HTTP» следующие:

**200 ОК**

Общий код статуса успеха. Это самый распространенный код. Используется для обозначения успеха.

**201 СОЗДАН**

Произошло успешное создание (через POST или PUT). Установите заголовок Location, который будет содержать ссылку на вновь созданный ресурс (на POST). Содержание тела ответа может присутствовать или не присутствовать.

**204 НЕТ СОДЕРЖАНИЯ**

Указывает на успех, но в теле ответа ничего нет, часто используется для операций DELETE и PUT.

**ОШИБКА 400, НЕВЕРНЫЙ ЗАПРОС**

Общая ошибка при выполнении запроса может привести к неверному состоянию. Ошибки валидации домена, пропущенные данные и т. Д. - вот несколько примеров.

**401 РАЗРЕШЕНО**

Ответ кода ошибки для отсутствующего или неверного токена аутентификации.

**403 ЗАПРЕЩЕНО**

Код ошибки, когда пользователь не авторизован для выполнения операции или ресурс недоступен по какой-либо причине (например, ограничения по времени и т. Д.).

**404 НЕ НАЙДЕНО**

Используется, когда запрошенный ресурс не найден, существует ли он или существует 401 или 403, который по соображениям безопасности служба хочет замаскировать.

**405 МЕТОД НЕ РАЗРЕШЕН**

Используется для указания того, что запрошенный URL-адрес существует, но запрошенный метод HTTP неприменим. Например, POST / users/12345, где API не поддерживает создание ресурсов таким способом (с предоставленным идентификатором). Заголовок Allow HTTP должен быть установлен при возврате 405, чтобы указать поддерживаемые методы HTTP. В предыдущем случае заголовок выглядел бы как «Разрешить: GET, PUT, DELETE»

**409 КОНФЛИКТ**

Всякий раз, когда конфликт ресурсов будет вызван выполнением запроса. Дублирующие записи, такие как попытка создать двух клиентов с одинаковой информацией и удаление корневых объектов, когда каскадное удаление не поддерживается, являются парой примеров.

**500 - ВНУТРЕННЯЯ ОШИБКА СЕРВЕРА**

Никогда не возвращайте это намеренно. Общая ошибка всеохватывающего типа, когда серверная сторона выдает исключение. Используйте это только для ошибок, которые потребитель не может устранить на своей стороне.

### 1.2. Современные веб-фреймворки: Laravel

Laravel - это веб-фреймворк с открытым исходным кодом PHP, разработанный и поддерживаемый Тейлором Отвеллом, который пытается предоставить более продвинутые возможности CodeIgniter Framework. Его архитектурные образцы основаны прежде всего на Symphony.

Эта платформа приобрела большую популярность среди ведущих разработчиков PHP после выпуска версии 3, которая включала такие функции, как интерфейс командной строки, поддержка систем баз данных и миграция. Он также представил упаковочную систему под названием Bundle.

Первая стабильная версия Laravel была выпущена в июне 2011 года. Текущая стабильная версия - 5.6 была выпущена в феврале 2018 года и версия 5.7, которая была недавно выпущена в сентябре 2018 года.

Laravel предлагает богатый набор функций, который включает в себя базовые функции PHP-фреймворков, таких как CodeIgniter, Yii и других языков программирования, таких как Ruby on Rails. Laravel обладает очень богатым набором функций, которые повысят скорость веб-разработки.

Если вы знакомы с Core PHP и Advanced PHP, Laravel облегчит вашу задачу. Это экономит много времени, если вы планируете разработать сайт с нуля. Кроме того, веб-сайт, созданный в Laravel, защищен и предотвращает несколько веб-атак.

Далее мы расскажем о некоторых интересных особенностях фреймворка Laravel, которые объяснят, почему он достигает такой популярности.

### 1.2.1. Поддержка MVC и объектно-ориентированный подход

Первое и главное преимущество использования Laravel Framework заключается в том, что он выглядит следующим образом - архитектурные шаблоны на основе моделей, визуалов и контроллеров, а также имеет выразительный синтаксис, который делает его объектно-ориентированным.

### 1.2.2. Встроенная аутентификация и авторизация

Laravel предоставляет готовую конфигурацию для систем аутентификации и авторизации. То есть, в порядке нескольких мастеров, ваше приложение будет оснащено безопасной аутентификацией и авторизацией.

### 1.2.3. Упаковочная система

Система упаковки связана со многими вспомогательными программами или библиотеками, которые помогают веб-приложениям автоматизировать процесс. Laravel использует композитора в качестве менеджера композиторов,

который управляет всей информацией, необходимой для управления пакетами. Пакет - отличный способ ускорить разработку, мы должны предоставить функциональность вне коробки. Image, Laravel Debug Bar и Laravel IDE Helper - одни из лучших пакетов Laravel.

### 1.2.4. Оригинальная файловая система

Laravel также имеет встроенную поддержку для облачной системы хранения, такой как Amazon S3 и Rack Space, и, конечно, для локальной системы хранения. Переключение между этими вариантами хранения удивительно просто, потому что API остается одинаковым для каждой системы. Вы можете использовать все три системы в приложении для обслуживания файлов из разных мест в распределенной среде.

### 1.2.5. Консоль Artisan

Laravel имеет собственный интерфейс командной строки под названием Artisan. Общие применения Artisan включают публикацию активов пакета, управление миграцией базы данных, создание нового контроллера, модели и стандартного кода для миграции. Эта функция освобождает разработчика от создания правильных скелетов кода. Применение новой пользовательской команды расширяет функциональные возможности и возможности мастера.

### 1.2.6. Отличный ORM

Eloquent ORM - это базовая OSM-реализация Laravel. Laravel имеет лучший объектно-реляционный маппер, по сравнению с другими конфигурациями. Это объектно-реляционное отображение позволяет вам взаимодействовать с объектами вашей базы данных и связями с базой данных, используя синтаксис выражения.

### 1.2.7. Плавающий движок

Laravel поставляется со встроенным механизмом шаблонов, известным как механизм шаблонов лезвий. Блейд-шаблонизатор объединяет один или несколько шаблонов с моделью данных для генерации результирующих идей, которые путем переноса шаблона в кэшированный код PHP для повышения производительности. Blade также предоставляет набор своих собственных управляющих структур, таких как условные операторы и циклы, которые внутренне сопоставлены с их аналогами PHP.

### 1.2.8. Планирование

Планировщик, представленный в Laravel 5.0, является дополнением к утилите Artisan Command-Line, которая позволяет программно планировать задачи, выполняемые время от времени. Внутри планировщик полагается на демон Crown для выполнения одного задания ремесленника, которое, в свою очередь, выполняет настроенные задачи.

### 1.2.9. События и трансляции

Laravel имеет концепцию, называемую широковещательным именем, которая полезна в современном веб-приложении для реализации данных в реальном времени, включая прямые трансляции и т. Д. Широковещательная передача позволяет вам совместно использовать одно и то же имя события на стороне сервера и на стороне клиента, чтобы вы могли получать данные в режиме реального времени из приложения.

### 1.2.10. Тестирование

Когда дело доходит до тестирования приложения, Laravel по умолчанию предоставляет модульное тестирование приложения, в котором есть самотестирование, которое обнаруживает и останавливает регрессию в структуре. Интеграция модуля PHP в приложение Laravel, такое как тестовый

фреймворк, очень проста. Помимо этого, модульное тестирование может быть запущено с помощью утилиты командной строки artisan.

### 1.3. Приложения одностраничники (Single-Page Applications)

Сегодня существует два основных подхода к созданию веб-приложений: традиционные веб-приложения, которые выполняют большую часть логики приложения на сервере, и одностраничные приложения (SPA), которые выполняют большую часть логики пользовательского интерфейса в веб-браузере, взаимодействуя с веб-сервером. в основном с использованием веб-API. Возможен также гибридный подход, простейшим из которых является размещение одного или нескольких богатых SPA-подобных субприложений в более крупном традиционном веб-приложении.

Вы должны использовать традиционные веб-приложения, когда:

 Требования вашего приложения к клиенту просты или доступны только для чтения.

Ваше приложение должно работать в браузерах без поддержки JavaScript.

 Ваша команда не знакома с методами разработки JavaScript или TypeScript.

Вы должны использовать SPA, когда:

Ваше приложение должно предоставлять богатый пользовательский интерфейс со многими функциями.

Ваша команда знакома с разработкой JavaScript и / или TypeScript.

Ваше приложение уже должно предоставлять API для других (внутренних или общедоступных) клиентов.

Кроме того, структуры SPA требуют большего опыта в области архитектуры и безопасности. Они испытывают больший отток из-за частых обновлений и новых фреймворков, чем традиционные веб-приложения.

Настройка процессов автоматической сборки и развертывания и использование таких параметров развертывания, как контейнеры, в приложениях SPA сложнее, чем в традиционных веб-приложениях.

Улучшения в пользовательском опыте, ставшие возможными благодаря модели SPA, должны быть сопоставлены с этими соображениями.

Ниже приведено более подробное объяснение того, когда выбирать стиль разработки одностраничных приложений для вашего веб-приложения.

Ваше приложение должно предоставлять богатый пользовательский интерфейс со многими функциями

SPA могут поддерживать богатые функциональные возможности на стороне клиента, которые не требуют перезагрузки страницы, когда пользователи выполняют действия или перемещаются между областями приложения. SPA могут загружаться быстрее, извлекая данные в фоновом режиме, а индивидуальные действия пользователя более отзывчивы, поскольку полная перезагрузка страницы встречается редко. Соглашения SPA могут поддерживать пошаговые обновления, сохраняя частично заполненные формы или документы, и пользователю не нужно нажимать кнопку для отправки формы. SPA могут поддерживать более богатое поведение на стороне клиента, такое как перетаскивание, гораздо легче, чем традиционные приложения. SPA могут быть спроектированы для работы в автономном режиме, делая обновления для модели на стороне клиента, которые в конечном итоге синхронизируются обратно на сервер после восстановления соединения. Вам следует выбрать приложение в стиле SPA, если требования вашего приложения включают в себя богатую функциональность, выходящую за рамки того, что предлагают типичные HTML-формы.

Обратите внимание, что часто SPA должны реализовывать функции, встроенные в традиционные веб-приложения, такие как отображение

значимого URL-адреса в адресной строке, отражающего текущую операцию (и позволяющего пользователям добавлять закладки или глубокие ссылки на этот URL-адрес, чтобы вернуться к нему). Соглашения SPA также должны позволять пользователям использовать кнопки браузера «назад» и «вперед» с результатами, которые их не удивят.

Ваша команда знакома с разработкой JavaScript и / или TypeScript

Для написания SPA требуется знание JavaScript и / или TypeScript, а также методов и библиотек на стороне клиента. Ваша команда должна быть компетентной в написании современного JavaScript с использованием SPA-инфраструктуры, такой как Angular.

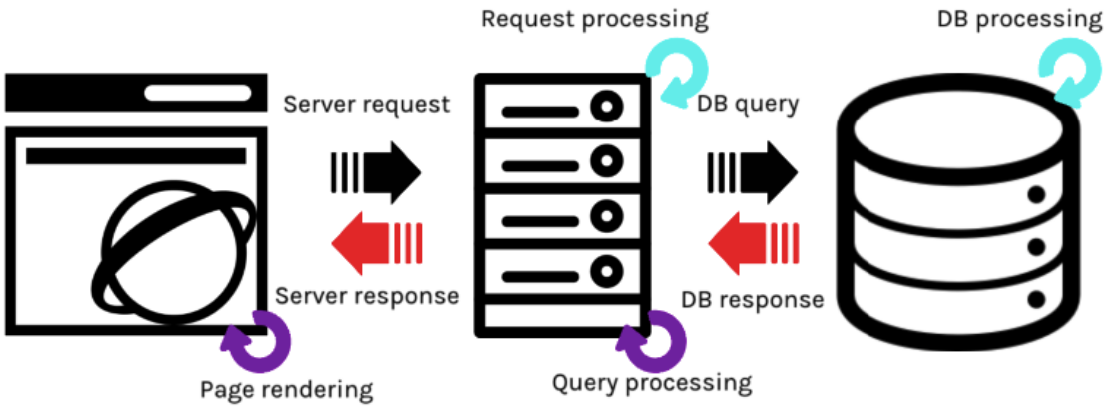
В следующей таблице решений приведены некоторые основные факторы, которые следует учитывать при выборе между традиционным веб-приложением и SPA.

| **Фактор** | **Традиционные веб-приложения** | **Приложения-одностраничники** |
|---|---|---|
| Требуется знакомство команды с JavaScript / TypeScript | **Минимум** | **Обязательно** |
| Поддержка браузеров без сценариев | **Поддерживается** | **Не поддерживается** |
| Минимум поведения на стороне клиента | **Подходит** | **Вне рамок** |
| Богатые, сложные требования пользовательского интерфейса | **Ограниченно** | **Подходит** |

Традиционный способ создания веб-приложения - использовать знакомую архитектуру MPA. Название такого подхода говорит само за себя, и все мы, кто знает хотя бы некоторые основы работы веб-сайтов, прекрасно знакомы с идеей, лежащей в основе: приложения состоят из нескольких разных страниц, которые обновляются по запросу.

Это означает, что ваш браузер перезагружает всю страницу с нуля каждый раз, когда вы пытаетесь получить доступ к новым данным или перейти на другую часть веб-сайта, на котором вы находитесь.

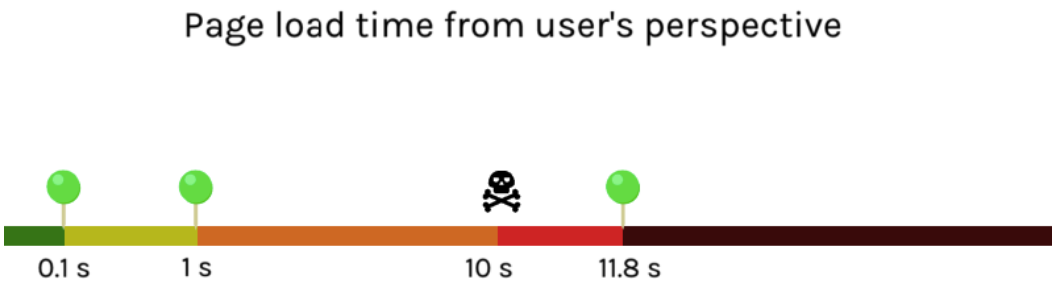
Это оставляет нас с тем фактом, что, за исключением дизайна, знакомого каждому интернет-пользователю, мы в то же время ограничены тем, что работа с многостраничным веб-приложением идет назад и вперед между браузером нашего конечного пользователя и сервером с HTML-запросы и ответы.Это традиционная технология, используемая с первых дней всемирной паутины до наших дней.




Как вы видите, когда мы говорим о консервативном подходе к созданию веб-приложений, мы должны понимать, что нам нужно иметь дело с дорожной картой, которая имеет несколько узких мест.

Почему это важно? Потому что еще в 2009 году универсальный стандарт для времени загрузки страницы считался равным 2 секундам. Должен ли я упомянуть, что с тех пор ожидания пользователей выросли еще выше?

А вот как выглядит восприятие времени загрузки:




Другими словами, вот как пользователи видят производительность веб-приложения:

0,1-0,2 секунды - такой немедленный ответ позволяет пользователю считать приложение продолжением себя.

1 секунда - пользователь начинает замечать задержанный ответ и отделяется от опыта

10 секунд - они отвлекутся и будут уже думать о чем-то другом и, скорее всего, уйдут и забудут о вашем приложении

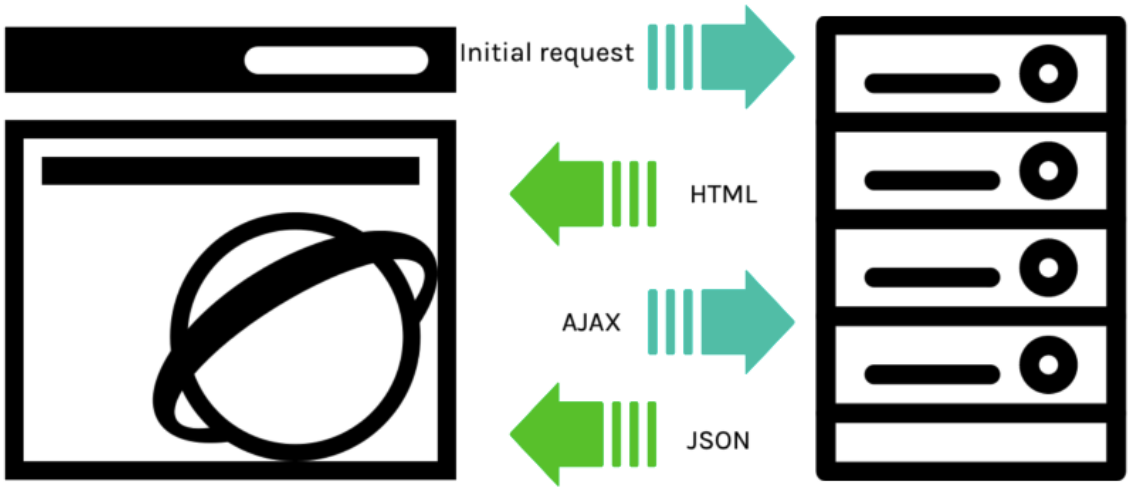
Но что это за 11,8 секунд на нашей временной шкале? Это среднее время загрузки страницы для веб-сайтов в Интернете. Излишне говорить, что у нас есть возможности для совершенствования.

Теперь, если мы посмотрим на традиционный рабочий процесс многостраничных приложений, мы заметим, что существует слишком много узких мест, которые могут ослабить нашу производительность в любой точке процесса, начиная с задержки в сети, влияющей на скорость передачи данных, и вплоть до лаги в обработке сервера или базы данных.

Естественно, у нас возникла потребность в лучшем решении, которое позволило бы устранить хотя бы некоторые слабые места. Простым решением было перенести как можно больше действий в браузеры пользователей.

Это момент, когда в игру вступает одностраничная архитектура приложения. Основная идея SPA заключается в том, чтобы избежать

перезагрузки всей страницы, а вместо этого переписать определенное место на текущей странице, используя запросы AJAX.




Рендеринг только необходимых обновлений, а не перезагрузка всей страницы - вот что действительно выделяет производительность одностраничных приложений.

Конечно, у каждого решения есть свои плюсы и минусы. SPA не являются исключением из этого правила.Хотя мы признаем критику, которая окружает эту концепцию, в этой статье мы хотели бы сосредоточиться на преимуществах одностраничного дизайна приложений и указать пути преодоления ограничений, которые нельзя игнорировать.

**Пользовательский опыт**

Наиболее отличительной особенностью SPA с точки зрения UX является сходство опыта с нативными настольными приложениями, к которым привыкли пользователи. Целостность и, казалось бы, безупречная реакция делают это чувство еще сильнее.

Благодаря принципиально новому подходу к архитектуре таких приложений, просмотр приложения больше не требует перезагрузки

нескольких страниц, что обеспечивает нам минимальные сбои при работе с приложением.

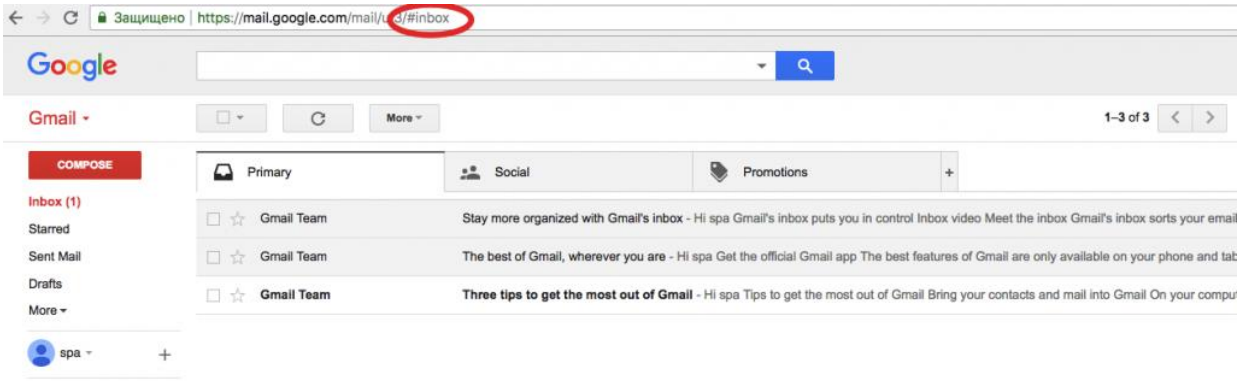
В первые дни приложений SPA была одна большая проблема с точки зрения пользовательского интерфейса, который, по сути, берет свое начало от многостраничных веб-приложений: кнопка «Назад».

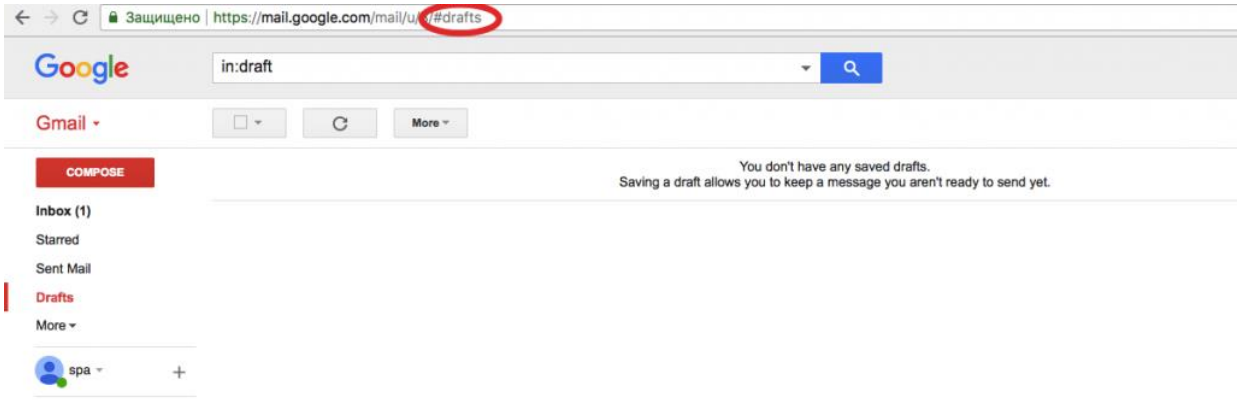
Причина, по которой возникает эта проблема, заключается в том, что кнопка «Назад» браузера управляет историей URL-адресов, а не состоянием используемого вами приложения. Соответственно, вместо того, чтобы показывать предыдущий экран одностраничного приложения, которое обычно находится полностью в одном URL, ваш браузер перенаправит вас на предыдущий URL, который вы посетили.

**Решение**

Решение этой проблемы удивительно просто: манипулировать URL-адресами с помощью API истории.Есть несколько методов, которые вы можете выбрать, но идея, лежащая в их основе, почти одинакова - вызывать события, основанные на действиях пользователя или местоположении на странице, и вносить изменения в URL, добавляя определенный хэш. Поскольку URL-адреса теперь постоянно меняются, теперь мы можем вернуться, переслать или поделиться точным местом, где они находятся, с другими.

Вот пример события hashchange, используемого в Gmail, которое вы можете даже проверить самостоятельно:



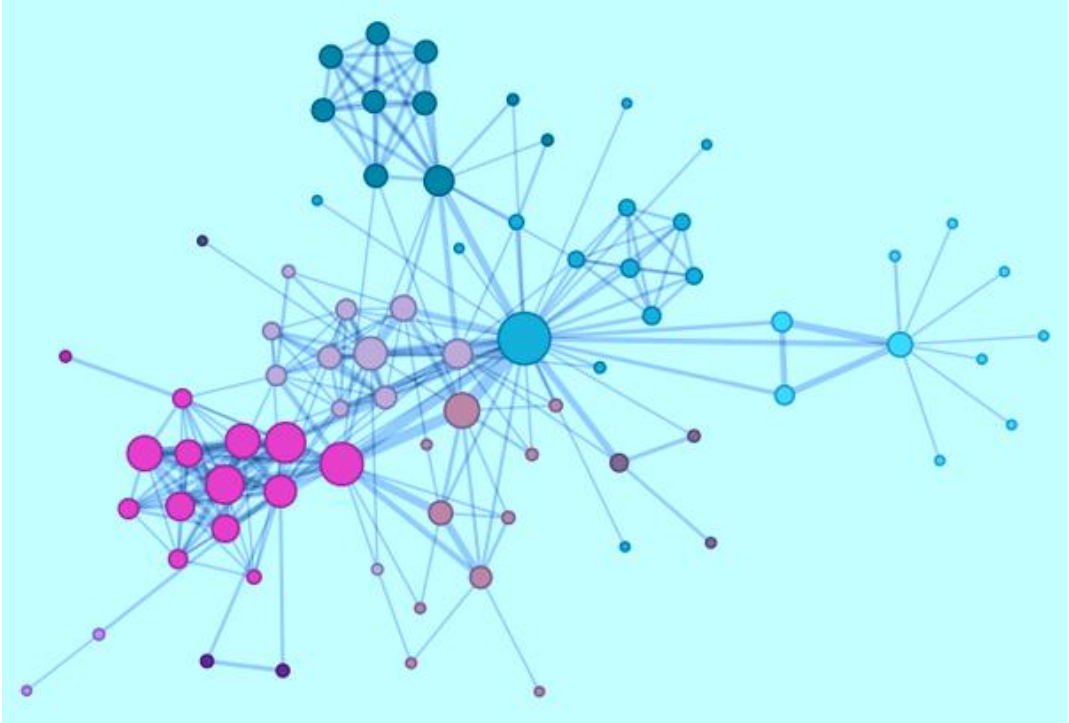

Будучи одностраничным приложением, Gmail не отправляет вас на новую страницу каждый раз, когда вы выполняете действие. Вместо этого они реализуют маршрут хеширования, который включает всю навигацию, к которой вы привыкли в своем приложении.

**Сложности с SEO**

Мы не собираемся отрицать, что трудности с SEO-оптимизацией были огромной проблемой для каждого интернет-предпринимателя, который рассматривал возможность создания одностраничного приложения. Причина этого связана с алгоритмами веб-сканеров, которые использовались до того, как был представлен этот новый подход к разработке.

Проще говоря, в поисковых системах есть процессы, называемые веб-сканерами, которые посещают URL-адреса в Интернете. Как только они появятся на вашем веб-сайте, они полны решимости посетить каждый URL-

адрес, который у вас есть, проиндексировать страницы, а затем перейти на другие ссылки, найденные на этих страницах:



Проблема заключалась в том, что веб-сканеры не были оптимизированы для SPA. Они не могли выполнить javascript, и поэтому при попытке посетить ваши URL-адреса с хэш-тегами сканер никуда не денется, а страницы не обновятся, ваш сайт не будет проиндексирован. Так как же сделать одностраничное приложение SEO-дружественным?

**Решение**

Эту проблему в основном решали сами поисковые системы. То, что они теперь делают, когда посещают URL с хэштегами, это переотображение и запрос новой страницы с ваших серверов:

От

www.singlepageapp.com #! key1 = value1

к

www.singlepageapp.com ? _escaped_fragment_ = key1 = value1

Единственное, что вам нужно сделать, это ответить на такой запрос веб-сканера и предоставить им страницу с SEO-оптимизированным контентом.

Следование такой схеме может не только решить проблему, но и дает вам значительное преимущество в том, что вам не нужно находить золотую середину между сильно улучшенным SEO текстом и дружественными для человека копиями, потому что страницы, которые вы будете передавать сканеру, не будут то же самое видят ваши посетители.

**Разработка**

Большинство обученных разработчиков на вашем пути получат больше знаний и опыта работы с традиционными шаблонами MPA. По понятным причинам. С другой стороны, если вы решите создать одностраничное веб-приложение, оно может оказаться более простым, быстрым и менее сложным способом достижения ваших конечных целей из-за подходов к проектированию, которые используются в качестве передового опыта при разработке приложений SPA ,

Еще одно преимущество SPA над MPA - это многократно используемый бэкэнд-код . Это самая важная вещь, которую вы должны знать при выборе между двумя, имея в виду проект, который требует как веб-приложений, так и мобильных приложений. Повторно используемый код означает меньше работы, меньше работы - меньше затрат на разработку.

В долгосрочных сценариях еще одной вещью, которую вы хотите сохранить, является ваше видение продукта. SPA лучше всего подходят для определенных и отточенных идей и проектов с узкой направленностью. Если вы не уверены, что одной страницы достаточно для вашего бизнеса, стоит ли пытаться уместить ее в слишком тесную коробку? МОРы более гибки в плане добавления новых страниц без нарушения какой-либо архитектуры. Только не

забывайте: возможность добавить новую страницу не всегда означает, что вы должны.

## 1.4. Фреймворки для разработки мобильных приложений

### 1.4.1. Phonegap

PhoneGap - это широко распространенная технология для разработки кроссплатформенных мобильных приложений. Эта среда разработки мобильных приложений с открытым исходным кодом была разработана Nitobi Software (сейчас Adobe). Это позволяет разработчикам создавать приложения с использованием HTML, JavaScript и CSS.

PhoneGap был впервые назван « **Apache Cordova** », который был представлен Nitobi. Adobe купила Nitobi и переименовала Apache Cordova в «PhoneGap».

Зачем использовать PhoneGap?

Разработчикам важно знать причину, по которой они должны выбрать PhoneGap, когда существует так много других платформ. Что ж, этот раздел статьи прояснит ваши сомнения.

• Кроссплатформенная совместимость

Разработка мобильных приложений - действительно сложный процесс, и для разработчиков очень важно выбрать надежную платформу.PhoneGap является надежной кроссплатформенной платформой и совместима с различными платформами разработки мобильных приложений, включая iOS, Blackberry, Android, WebOS, Symbian, Tizen и Bada. Команда разработчиков приложений должна написать только один код, сокращая усилия по написанию различных программ для разных платформ.

• экономически эффективным

Если у вас ограниченный бюджет и вы хотите получить оптимальное кредитное плечо с небольшими инвестициями, то PhoneGap - подходящий инструмент для вас. Вы можете написать одну программу для приложения, которое будет запускаться на разных платформах.Это сэкономит вам много денег и усилий.

• Лучший доступ к собственным API

PhoneGap предоставляет разработчикам мобильных приложений доступ к собственным API-интерфейсам, чтобы ваше приложение могло использовать камеру, геолокацию, акселерометр, контакты и все другие встроенные функции. Это помогает пользователям получить опыт использования нативных приложений, который намного лучше по сравнению с веб-приложениями.

• Путь в большую базу сообщества

Если вы пробуете что-то новое, вам необходимо иметь адекватную поддержку. Сообщество разработчиков Cordova - одно из крупнейших сообществ, которое постоянно растет. Люди этого сообщества дружелюбны, и вы получите ответ на большинство проблем. Есть множество примеров и очень много опытных членов, которые окажут вам большую поддержку.

• Гибкость с использованием веб-технологий

Приложения Cordova создаются с использованием некоторых простых языков, таких как CSS, HTML и JavaScript. Вам не нужно изучать новые языки, когда вы планируете начать разработку мобильного приложения с PhoneGap. Вы можете использовать существующие языки и писать программы с легкостью.

PhoneGap уже давно доступен, и есть множество библиотек и фреймворков, которые могут вам помочь. Таким образом, вы можете использовать различные типы опций для лучшего программирования.

• Надежная поддержка бэкэнда

PhoneGap имеет мощную поддержку бэкэнда, которая ускоряет процесс разработки и уменьшает усилия разработчика. Они также предоставляют новичкам руководство для начинающих разработчиков, чтобы легко понять весь проект.

• гибкость в развитии

Разработчики обладают большой гибкостью и гибкостью, поскольку весь процесс разработки приложений прост. Нет необходимости прилагать серьезные усилия для осуществления процесса развития.Базовые знания вышеупомянутых языков достаточно для разработки приложения.

• UI библиотеки, улучшающие пользовательский интерфейс

Для большинства мобильных пользователей распространенной проблемой является то, что просмотр веб-страницы, совместимой только с компьютерами и планшетами с большими экранами, действительно обременителен. Прокрутка вниз до страницы и проверка с ней и некоторыми другими задачами потребуют много усилий, и вы не сможете легко получить доступ к страницам.

Но благодаря наличию в PhoneGap библиотек пользовательского интерфейса пользовательский интерфейс приложений и веб-страниц значительно улучшен. Это помогает обеспечить лучший опыт просмотра для различных целевых зрителей.

Вот некоторые из примечательных особенностей PhoneGap, которые делают его лучшим выбором для разработчиков и легко объясняют их клиентуре. Эти функции сделали PhoneGap одной из лучших платформ Android для разработки приложений в высококонкурентном технологическом пространстве.

### 1.4.2. Ionic framework

В основном программирование на Android выполняется на Java, а программирование на iOS - в Objective C.

Мы, веб-разработчики, привыкли к менее строгим языкам, уделяем больше внимания творческой стороне и любим меньше сложности. Следовательно, мы изучаем HTML / CSS / Javascript - который не требует компиляции перед запуском кода.

Итак, мы кодируем в javascript - поэтому хорошие боги индустрии придумали фреймворки, с помощью которых вы можете разрабатывать приложения для iOS и Android, используя веб-языки, такие как html / css / javascript. Они называют это Гибридными приложениями - Non-Native - Работает на Android и iOS в одном кадре.

Ionic - это такой гибридный фреймворк, построенный на AngularJS (супергероический фреймворк javascript, созданный Google)

Мы видели наше ионное приложение, где вы видите, что переходы немного паршивые - так что Facebook тоже был таким - не используя ионные, а что-то свое ... Поэтому они придумали свои собственные алгоритмы и способ работы, а затем выпустили его как новый фреймворк в javascript под названием ReactJS. Также что-то специально для нативных приложений, с которыми также создано приложение Facebook, называется ReactNative.

Таким образом, сильная конкуренция Angular JS - React обладает хорошей логикой загрузки DOM и начала быстро набирать обороты…. Таким образом, Angular изучил все ошибки и теперь разработал совершенно новый Angular2.

Таким образом, для веб-разработчиков возможны варианты разработки нативных приложений: Ionic Framework и React Native.

### 1.4.3. JQuery Mobile

jQuery долгое время была популярной библиотекой JavaScript для создания многофункциональных интерактивных веб-сайтов и веб-приложений. Однако, поскольку он был разработан в основном для настольных браузеров, он не имеет многих функций, специально предназначенных для создания мобильных веб-приложений.

jQuery Mobile - это новый проект, который решает эту проблему. Это основа, построенная на основе jQuery, которая предоставляет ряд элементов и функций пользовательского интерфейса для использования в мобильных приложениях.

На момент написания фреймворк был довольно передовым - фактически, первая альфа-версия была выпущена только в прошлом месяце - но с ней уже можно сделать некоторые замечательные вещи.

В этой статье я расскажу о некоторых ключевых функциях и преимуществах jQuery Mobile и покажу несколько примеров того, как эта новая платформа может помочь вам быстро и качественно создавать высококачественные мобильные приложения.

Для достижения наилучших результатов вы, вероятно, захотите просмотреть примеры в этой статье на мобильном устройстве, таком как iPhone или Android. В качестве альтернативы, Safari на рабочем столе (с узкой шириной окна) является хорошей заменой.

**Что делает jQuery Mobile?**

jQuery Mobile упрощает разработку пользовательских интерфейсов для мобильных веб-приложений.

Конфигурация интерфейса основана на разметке, что означает, что вы можете в значительной степени создать весь базовый интерфейс приложения в HTML без необходимости написания одной строки JavaScript. (Конечно, вам

все равно придется писать JavaScript, если ваше приложение должно делать что-то полезное!)

Он предоставляет ряд новых пользовательских событий, позволяющих обнаруживать мобильные и сенсорные определенные действия, такие как касание, нажатие и удержание, пролистывание и изменение ориентации (то есть вращение устройства).

Он использует прогрессивное улучшение, чтобы гарантировать, что интерфейс вашего приложения работает практически на любом веб-устройстве.

Он использует темы, чтобы легко настроить внешний вид вашего приложения.

Для действительно детального ознакомления с jQuery Mobile - и того, как вы можете использовать его для создания великолепных мобильных приложений - ознакомьтесь с моей новой книгой: Master Mobile Web Apps с jQuery Mobile.

# 2. РАЗРАБОТКА СИСТЕМЫ ОНЛАЙН ЗАКАЗОВ

## 2.1. Проектирование взаимодействия по API

Наше приложение предназначено для выполнения заказов клиентами кафе. Клиенты, приходя в кафе, и используя его фирменное приложение, будут выбирать блюда из меню, в результате заказ будет отображаться на экране повара/менеджера/официанта. В приложении будет выводиться общая сумма заказа, так что будет упрощена и оптимизирована вся процедура выполнения заказа.

Для этого нам необходимо разработать клиент-серверное приложение, клиентом в котором будет выступать мобильный телефон, который по API (Application Programming Interface) будет отправлять данные о заказе на сервер. Сервер, в свою очередь, может по API отдавать список категорий (салаты, первые блюда, напитки, вторые блюда и так далее), а также блюда из этих категорий. Выполненные заказы будут отмечаться в интерфейсе.

Далее мы рассмотрим некоторые основные принципы, которые мы используем при разработке нашего приложения и сервера.

Мы должны проектировать и разрабатывать программные решения с учетом удобства обслуживания. Принципы, изложенные в этом разделе, могут помочь вам принять архитектурные решения, которые приведут к чистым, поддерживаемым приложениям. Как правило, эти принципы помогут вам создавать приложения из отдельных компонентов, которые не тесно связаны с другими частями вашего приложения, а скорее взаимодействуют через явные интерфейсы или системы обмена сообщениями.

### Общие принципы дизайна. Разделение задач

Руководящим принципом при разработке является разделение интересов . Этот принцип утверждает, что программное обеспечение должно

быть отделено на основе видов работы, которые оно выполняет. Например, рассмотрим приложение, которое включает логику для идентификации заслуживающих внимания элементов, отображаемых для пользователя, и которое форматирует такие элементы определенным образом, чтобы сделать их более заметными. Поведение, отвечающее за выбор элементов для форматирования, должно быть отделено от поведения, отвечающего за форматирование элементов, поскольку это отдельные проблемы, которые только случайно связаны друг с другом.

Архитектурно приложения могут быть логически построены в соответствии с этим принципом путем отделения основного бизнес-поведения от инфраструктуры и логики пользовательского интерфейса. В идеале бизнес-правила и логика должны находиться в отдельном проекте, который не должен зависеть от других проектов в приложении. Это помогает гарантировать, что бизнес-модель легко тестируется и может развиваться, не будучи тесно связанным с деталями реализации низкого уровня. Разделение задач является ключевым аспектом использования уровней в архитектурах приложений.

**Инкапсуляция**

Различные части приложения должны использовать инкапсуляцию, чтобы изолировать их от других частей приложения. Компоненты и уровни приложения должны иметь возможность корректировать свою внутреннюю реализацию, не нарушая своих коллабораторов, если внешние контракты не нарушаются. Правильное использование инкапсуляции помогает добиться слабой связи и модульности в проектах приложений, поскольку объекты и пакеты могут быть заменены альтернативными реализациями, если поддерживается тот же интерфейс.

В классах инкапсуляция достигается путем ограничения внешнего доступа к внутреннему состоянию класса. Если внешний субъект хочет манипулировать состоянием объекта, он должен делать это через четко

определенную функцию (или установщик свойства), а не иметь прямой доступ к закрытому состоянию объекта. Аналогично, компоненты приложения и сами приложения должны предоставлять четко определенные интерфейсы для использования их соавторами, а не позволять напрямую изменять их состояние. Это позволяет внутреннему дизайну приложения развиваться со временем, не беспокоясь о том, что это нарушит коллаборационистов, пока поддерживаются публичные контракты.

**Инверсия зависимостей**

Направление зависимости в приложении должно быть направлено на абстракцию, а не на детали реализации. Большинство приложений написаны так, что зависимость времени компиляции течет в направлении выполнения во время выполнения. Это создает граф прямой зависимости. То есть, если модуль A вызывает функцию в модуле B, которая вызывает функцию в модуле C, то во время компиляции A будет зависеть от B, который будет зависеть от C, как показано на рисунке 4-1.

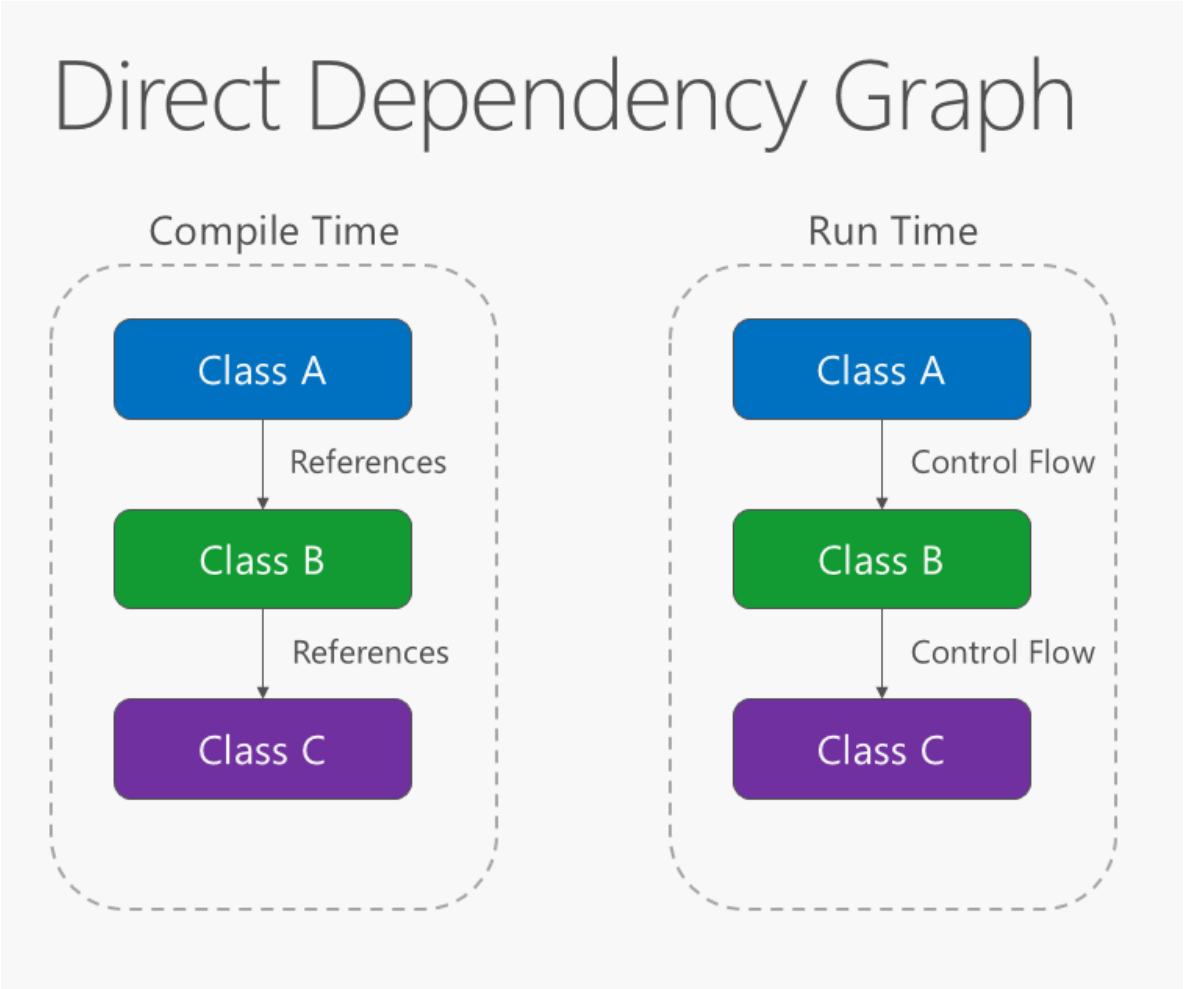



Рисунок 2.1. Граф прямой зависимости.

Применение принципа инверсии зависимостей позволяет A вызывать методы для абстракции, которую реализует B, позволяя A вызывать B во время выполнения, но для B зависеть от интерфейса, управляемого A во время компиляции (таким образом, инвертируятипичную компиляцию зависимость от времени). Во время выполнения поток выполнения программы остается неизменным, но введение интерфейсов означает, что различные реализации этих интерфейсов могут быть легко подключены.

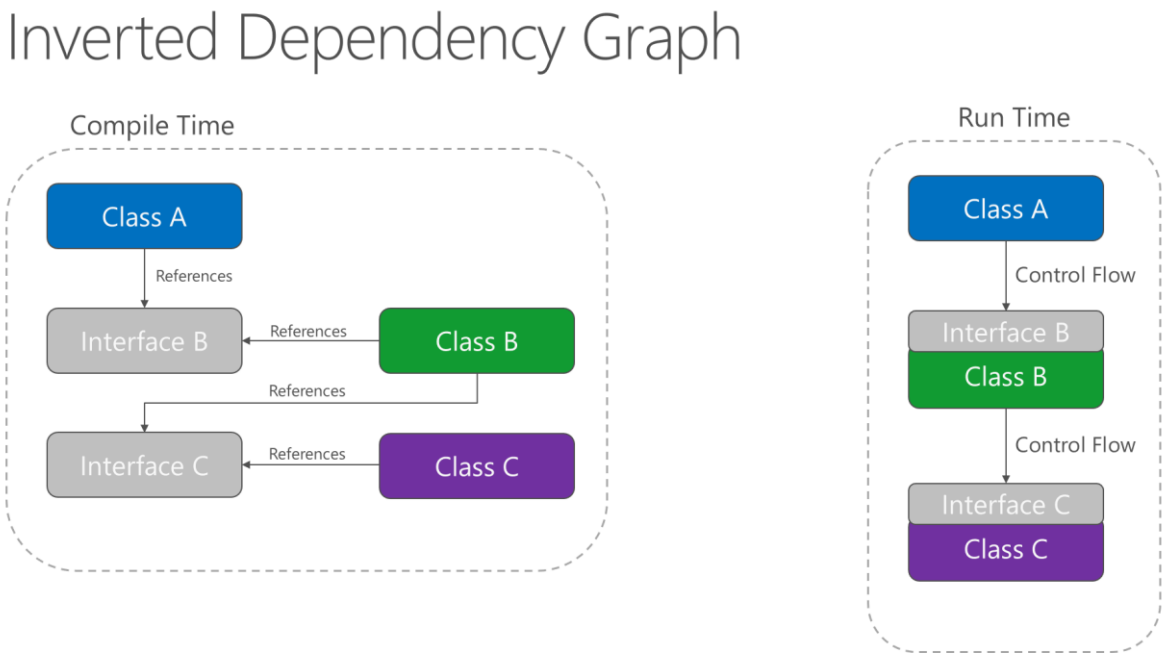



Рисунок 2.2. Перевернутый граф зависимостей.

Инверсия зависимостей является ключевой частью построения слабосвязанных приложений, поскольку детали реализации могут быть написаны так, чтобы зависеть от абстракций более высокого уровня и реализовывать их, а не наоборот. Получающиеся приложения являются более тестируемыми, модульными и в результате обслуживаемыми.Практика внедрения зависимостей стала возможной благодаря следованию принципу инверсии зависимостей.

**Явные зависимости**

Методы и классы должны явно требовать любые взаимодействующие объекты, которые им нужны для правильного функционирования. Конструкторы классов предоставляют классам возможность идентифицировать вещи, которые им необходимы, чтобы быть в допустимом состоянии и функционировать должным образом. Если вы определяете классы, которые можно создавать и вызывать, но которые будут функционировать должным образом только при наличии определенных

глобальных или инфраструктурных компонентов, эти классы будут нечестными по отношению к своим клиентам. Контракт конструктора сообщает клиенту, что ему нужны только указанные вещи (возможно, ничего, если класс просто использует конструктор по умолчанию), но затем во время выполнения оказывается, что объекту действительно нужно что-то еще.

Следуя принципу явных зависимостей, ваши классы и методы будут честны со своими клиентами относительно того, что им нужно для функционирования. Это делает ваш код более самодокументируемым, а контракты на кодирование - более удобными для пользователя, поскольку пользователи будут верить, что, пока они предоставляют то, что требуется в форме параметров метода или конструктора, объекты, с которыми они работают, будут работать правильно во время выполнения.

**Единственная ответственность**

Принцип единой ответственности применяется к объектно-ориентированному проектированию, но его также можно рассматривать как архитектурный принцип, аналогичный разделению интересов. В нем говорится, что объекты должны иметь только одну ответственность, и что у них должна быть только одна причина для изменения. В частности, единственная ситуация, в которой объект должен измениться, - это то, что способ, которым он выполняет свою единственную ответственность, должен быть обновлен.Следование этому принципу помогает создавать более слабосвязанные и модульные системы, поскольку многие виды нового поведения могут быть реализованы как новые классы, а не путем добавления дополнительной ответственности к существующим классам.Добавление новых классов всегда безопаснее, чем изменение существующих классов, так как от новых классов еще не зависит ни один код.

В монолитном приложении мы можем применять принцип единой ответственности на высоком уровне к уровням приложения. Ответственность

за представление должна оставаться в проекте пользовательского интерфейса, тогда как ответственность за доступ к данным должна оставаться в рамках проекта инфраструктуры. Бизнес-логика должна храниться в основном проекте приложения, где она может быть легко протестирована и может развиваться независимо от других обязанностей.

Когда этот принцип применяется к архитектуре приложения и доводится до логической конечной точки, вы получаете микросервисы. Данный микросервис должен нести единоличную ответственность. Если вам необходимо расширить поведение системы, обычно лучше сделать это, добавив дополнительные микросервисы, а не добавляя ответственность к существующей.

**Не повторяйся (DRY)**

Приложение должно избегать указания поведения, связанного с определенной концепцией, в нескольких местах, так как это частый источник ошибок. В какой-то момент изменение требований потребует изменения этого поведения, и вероятность того, что по крайней мере один экземпляр поведения не будет обновлен, приведет к непоследовательному поведению системы.

Вместо того, чтобы дублировать логику, заключите ее в программную конструкцию. Сделайте эту конструкцию единой властью над этим поведением, и любая другая часть приложения, которая требует этого поведения, использует новую конструкцию.

Избегайте связывания воедино поведения, которое только случайно повторяется.Например, только потому, что две разные константы имеют одинаковое значение, это не означает, что у вас должна быть только одна константа, если концептуально они ссылаются на разные вещи.

**Игнорирование постоянства**

Persistance ignorance (PI) относится к типам, которые необходимо сохранить, но на код которых не влияет выбор технологии постоянства. Такие типы в .NET иногда называют обычными старыми объектами CLR (POCO), потому что им не нужно наследовать от определенного базового класса или реализовывать определенный интерфейс. Постоянное невежество является ценным, поскольку позволяет сохранять одну и ту же бизнес-модель несколькими способами, обеспечивая дополнительную гибкость приложения. Варианты персистентности могут со временем меняться от одной технологии базы данных к другой, или могут потребоваться дополнительные формы персистентности в дополнение к тому, с чего запускается приложение (например, с использованием кэша Redis или Azure DocumentDb в дополнение к реляционной базе данных).

- Некоторые примеры нарушений этого принципа включают в себя:
- Обязательный базовый класс.
- Требуемая реализация интерфейса.
- Классы, ответственные за сохранение себя (например, шаблон Active Record).
- Обязательный конструктор по умолчанию.
- Свойства, требующие виртуального ключевого слова.
- Персистентные обязательные атрибуты.

Требование, чтобы классы имели какие-либо из перечисленных выше функций или поведений, добавляет связь между типами, которые должны быть сохранены, и выбором технологии персистентности, затрудняя принятие новых стратегий доступа к данным в будущем.

**Ограниченные контексты**

Ограниченные контексты являются центральным паттерном в доменно-управляемом дизайне. Они предоставляют способ решения сложных задач в больших приложениях или организациях, разбивая их на отдельные концептуальные модули. Каждый концептуальный модуль затем представляет контекст, который отделен от других контекстов (следовательно, ограничен) и может развиваться независимо. Каждый ограниченный контекст в идеале должен быть свободен в выборе собственных имен для концепций внутри него и должен иметь эксклюзивный доступ к своему собственному хранилищу постоянства.

Как минимум, отдельные веб-приложения должны стремиться быть в своем собственном ограниченном контексте со своим собственным хранилищем постоянства для своей бизнес-модели, а не делиться базой данных с другими приложениями. Связь между ограниченными контекстами происходит через программные интерфейсы, а не через общую базу данных, что позволяет бизнес-логике и событиям происходить в ответ на происходящие изменения.Ограниченные контексты тесно связаны с микросервисами, которые также идеально реализуются как их собственные отдельные ограниченные контексты.

### 2.2. Разработка архитектуры и схемы базы данных

Для разработки архитектуры, нам понадобятся несколько таблиц, во первых - таблица для пользователей, на случай использования веб-сайта для заказа.

Во-вторых, нам необходима таблица для структурирования блюд - categories, в которой мы будем хранить категории блюд - первые, вторые блюда, салаты, холодные напитки, закуски и так далее. Данная таблица содержит только имя категории.

Вторая таблица - menus, которая собственно содержит названия блюд, их цены, category_id для указания категории блюда, price для указания цены.

Следующая таблица - orders, которая содержит собственно заказы пользователей. Она содержит внешний ключ menu_id, который указывает на таблицу menus, и соответствует заказанному блюду, user_id для идентификации пользователя, qty - будет использоваться в будущем для определения количества заказов. На данный момент поле qty будет по умолчанию содержать значение 1. Наконец, последнее поле - finished - указывает на выполнение заказа (0 - не выполнен, 1 - выполнен).

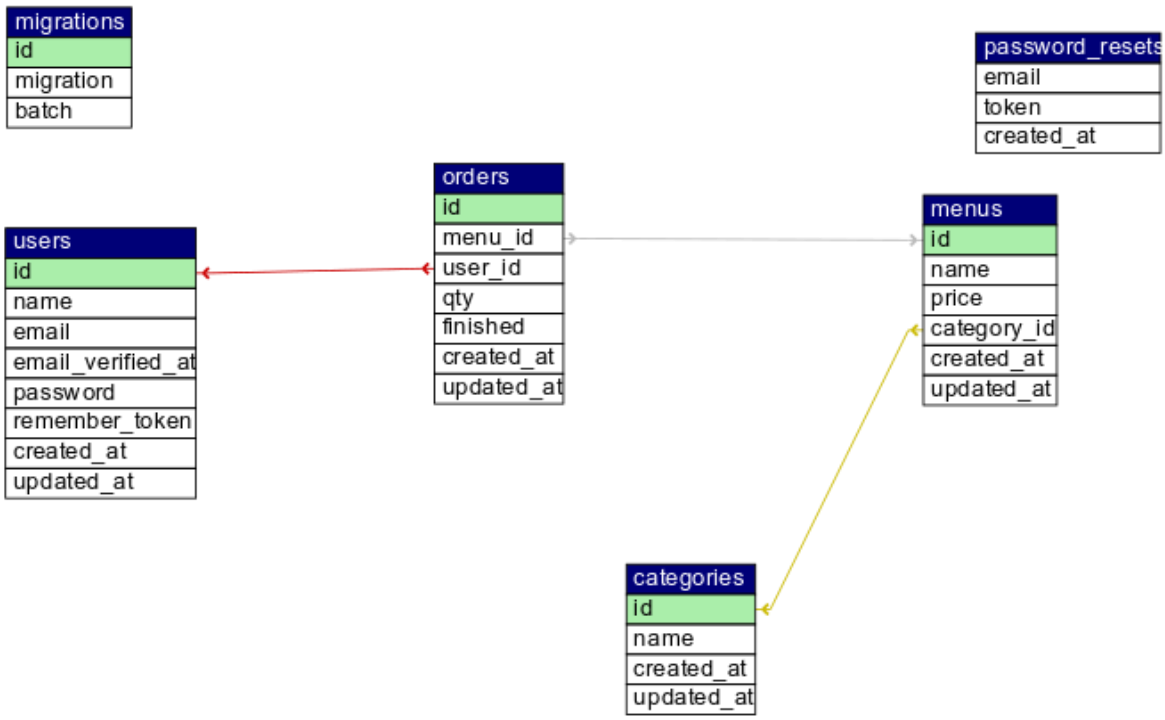



Рис.2.3. Схема базы данных

Почти все таблицы содержат поля created_at и updated_at, которые автоматически создаются инструментом миграции фреймворка Laravel. Данные поля могут понадобиться для отслеживания даты добавления записей в базу данных, для сортировки и поиска.

Вот так выглядит типичная миграция во фреймворке Laravel:

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateOrdersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('orders', function (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->integer('menu_id')->unsigned()-
>nullable();
            $table->integer('user_id')->unsigned()-
>nullable();
```

```
                $table->integer('qty')->unsigned()-
>nullable()->default(1);
                $table->tinyInteger('finished');
                $table->timestamps();
            });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
            Schema::dropIfExists('orders');
        }
    }
```

### 2.3. Создание мобильного приложения

### Установка Cordova CLI

Средство командной строки Cordova распространяется в виде пакета npm.

Чтобы установить cordova командной строки cordova, выполните следующие действия.

1. Загрузите и установите Node.js. При установке вы должны иметь возможность вызывать node и npm в командной строке.

2. (Необязательно) Загрузите и установите клиент git , если у вас его еще нет. После установки вы сможете вызывать git в командной строке. CLI использует его для загрузки ресурсов, когда на них ссылаются с помощью URL-адреса git-репо.

3. Установите модуль cordova с помощью утилиты npm из Node.js. Модуль cordova будет автоматически загружен утилитой npm .

sudo npm install -g cordova

В OS X и Linux для установки этой утилиты разработки в других ограниченных каталогах, таких как /usr/local/share может потребоваться префикс команды npm с помощью sudo . Если вы используете дополнительный инструмент nvm / nave или имеете доступ для записи в каталог установки, вы можете пропустить префикс sudo . Есть и другие советы по использованию npmбез sudo , если вы захотите это сделать.

в Windows:

C:\>npm install -g cordova

Приведенный выше флаг -g указывает npm устанавливать глобально. В противном случае он будет установлен в подкаталоге node_modules текущего рабочего каталога.

После установки вы сможете запустить cordova в командной строке без аргументов, и он должен напечатать текст справки.

**Создание приложения**

Перейдите в каталог, где вы храните свой исходный код, и создайте проект Cordova:

$ cordova create hello com.example.hello HelloWorld

Это создает необходимую структуру каталогов для вашего приложения cordova. По умолчанию cordova create создает скелетное веб-приложение, домашней страницей которого является файл проекта www/index.html .

**Добавление платформы**

Все последующие команды должны выполняться в каталоге проекта или в любых его подкаталогах:

$ cd hello

Добавьте платформы, которые вы хотите настроить для своего приложения. Мы добавим платформы «ios» и «android» и убедимся, что они сохраняются в config.xml и package.json :

$ cordova platform add ios $ cordova platform add android

Чтобы проверить свой текущий набор платформ:

$ cordova platform ls

Запуск команд для добавления или удаления платформ влияет на содержимое каталога платформпроекта, где каждая указанная платформа отображается в виде подкаталога.

Примечание. При использовании интерфейса командной строки для создания приложения не следует редактировать файлы в каталоге /platforms/ . Файлы в этом каталоге обычно перезаписываются при подготовке приложений к сборке или при переустановке плагинов.

**Установка пакетов необходимых для сборки**

Чтобы создавать и запускать приложения, вам нужно установить SDK для каждой платформы, на которую вы хотите настроить таргетинг. В качестве альтернативы, если вы используете браузер для разработки, вы можете

использовать платформу browser которая не требует каких-либо платформ SDK.

Чтобы проверить, удовлетворяете ли вы требованиям для построения платформы:

$ cordova requirements Requirements check results for android: Java JDK: installed . Android SDK: installed Android target: installed android-19,android-21,android-22,android-23,Google Inc.:Google APIs:19,Google Inc.:Google APIs (x86 System Image):19,Google Inc.:Google APIs:23 Gradle: installed Requirements check results for ios: Apple OS X: not installed Cordova tooling for iOS requires Apple OS X Error: Some of requirements check failed

**Создание приложения**

По умолчанию сценарий cordova create создает скелетное веб-приложение, стартовой страницей которого является файл проекта www/index.html . Любая инициализация должна быть указана как часть обработчика события deviceready, определенного в www/js/index.js .

Запустите следующую команду, чтобы построить проект для всех платформ:

$ cordova build

При желании вы можете ограничить область действия каждой сборки конкретными платформами - в этом случае 'ios':

$ cordova build ios

Смотрите также

Cordova build справочная документация по сборке

**Протестируем приложение**

SDK для мобильных платформ часто поставляются в комплекте с эмуляторами, которые выполняют образ устройства, так что вы можете запустить приложение с домашнего экрана и посмотреть, как оно взаимодействует со многими функциями платформы. Запустите команду, например, следующую, чтобы перестроить приложение и просмотреть его в эмуляторе конкретной платформы:

$ cordova emulate android

После выполнения команды cordova emulate обновляет изображение эмулятора, чтобы отобразить последнее приложение, которое теперь доступно для запуска с домашнего экрана:

Кроме того, вы можете подключить телефон к компьютеру и напрямую протестировать приложение:

$ cordova run android

Перед выполнением этой команды необходимо настроить устройство для тестирования, следуя процедурам, которые различаются для каждой платформы.

**2.4. Исходный код мобильного приложения**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```
        <title>Document</title>
        <script src="js/jquery-3.4.1.min.js"></script>
        <script src="js/bootstrap.min.js"></script>
        <link                              rel="stylesheet"
href="css/bootstrap.min.css">
    </head>
    <body>
        <div class="container">
            <div class="row">
                <div class="col-12">
                    <div id="top_menu">


                    </div>
                    <div class="" id="content">


                    </div>
                    <h2>Ваш заказ:</h2>
                    <div id="selected"></div>
                    <div       class="alert       alert-
success">Общая цена: <span id="total_price" style="font-
size:120%">0</span> сум.</div>
                    <button    class="btn    btn-warning
clear_selected">Очистить</button>
```

```
                </div>
            </div>
        </div>
        <script>
            $(document).ready(function(){
                $("#top_menu").on('click',  '.category',
function(){
                    $cat_id = $(this).attr('data-id');

                    $.get(
                        '/menu/get/'+$cat_id,
                        function(data){

$("#content").html("<ul></ul>");
                            data.forEach(element => {
                                $("#content
ul").append("<li><a    href='#'    class='food'    data-
price='"+element.price+"'                              data-
id='"+element.id+"'>"+element.name+"
("+element.price+")</a></li>");
                            });
                        },
                        'json'
                    );
```

```
                });
                $(".clear_selected").click(function(){
                  $.post(
                      '/order/clear',
                      {user_id : 1},
                      function(data){
                          $("#selected").text("");
                      }
                  );
                });

                $("#content").on('click',        '.food',
function(){
                    $food_id = $(this).attr('data-id');
                    $price                                =
parseInt($("#total_price").text());
                    $price                               +=
parseInt($(this).attr('data-price'));
                    $("#total_price").text($price);
                    var data = {
                        'menu_id' : $food_id,
                        'qty' : 1,
                    };
```

```
                    $.post('/order/add', data);

$("#selected").append("<li>"+$(this).text()+"</li>");
                    return false;
                });

                $.get(
                    '/menu/index', //url
                    function(data){
                        data.forEach(element => {

$("#top_menu").append("<button type='button' data-
id='"+element.id+"' class='btn btn-info
category'>"+element.name+"</button>");
                        });
                    },
                    'json' //content type
                );
            });
        </script>
    </body>
    </html>
```

## Заключение

На путь электронного ведения бизнеса встают в основном предприниматели, занимающиеся традиционными видами торговли. Их дело становится многоканальным, то есть «мультиритейлом». Однако помимо уже зарекомендовавших себя бизнесменов, открывают бизнес в электронной сфере лица, порой даже не представляющие, как выглядит на практике данный вид торговли.

В последние годы ни что другое так не распространяло бы свое массовое воздействие на все сферы человеческой жизни, как Интернет. Интернет уже давно перестал быть просто средством массовой информации и ее источником для большинства населения, это и средство общения, и способ заработка, и эффективный метод ведения полноценного бизнеса. Сегодня каждая уважающая себя компания имеет свой собственный сайт в Интернете, а производимая ею продукция или товары вовсю продаются посредством него и даже имеют приоритет в этой отрасли продаж. Через Интернет продаются всевозможные товары народного потребления, бытовая техника, косметические и фармацевтические препараты, бытовая и компьютерная техника, сотовая связь, детские игрушки, продовольственные товары и многое другое. Разве что электростанции пока еще не строятся через Интернет. Однако услуги их и других компаний широко предлагаются через персональные страницы и сайты в Глобальной сети.

С чем связано такое широкое применение Интернета в бизнесе? Во-первых, это доступно. Доступно как с точки зрения предприятия, поскольку в большинстве случаев не вызывает серьезных затрат на разработку, так и для потребителей, так как выход в Мировую паутину сегодня доступен даже школьнику. Во-вторых, это позволяет продавать товары по более низким ценам, что, конечно же, не может не радовать конечных покупателей, а также способствует развитию конкуренции на рынке. В-третьих, это удобно опять

же для каждой стороны. Предприятиям не нужно арендовать склады или выставочные залы для показа своей продукции, не нужно платить за аренду, охрану, сотрудников, консультантов и вообще намного меньше осуществляется тех или иных энергозатрат.

В данной работе мы рассмотрели понятия REST API, основные рекоммендации по разработке RESTful приложений, современные веб-фреймворки, одностраничные приложения и мобильные фреймворки.

Также мы провели работу по проектированию системы для онлайн заказов, проектированию базы данных и архитектуры приложения, а также описали собственно саму разработку приложения для Android на основе фреймворка Cordova.

Мы думаем, что данное приложение будет полезно для внедрения в точках общего питания, кафе, ресторанах и столовых, так как позволит воспользоваться всеми преимуществами современных информационно-коммуникационных технологий.

## Список литературы

1. Node-RED http://nodered.org/

2. SmartVisu http://www.smartvisu.de/

3. Netping http://www.netping.ru/

4. OpenRemote http://www.openremote.org/display/HOME/Home

5. Гершкович В.Ф. Энергосберегающие системы жилых зданий: пособие по проектированию // С.О.К., 2008. № 8

6. Спицын В.С., Спицын В.В. Серия «Компьютерные технологии, управление, радиоэлектроника» выпуск 17 // Алгоритмы управления температурой в помещениях, Вестник ЮУрГУ, 2012г. № 35. С. 79-84

7. Нимич Г.В. Общие положения автоматического управления системами кондиционирования и вентиляции / Г.В. Нимич // С.О.К. – 2005. – № 7.

8. Гершкович В.Ф. Энергосберегающие системы жилых зданий: пособие по проектированию // С.О.К., 2008. № 8.