ВВЕДЕНИЕ

В связи с бурным развитием информационных технологий и непрерывным увеличением объемов информации, доступной в глобальной сети Интернет, всё большую актуальность приобретают вопросы эффективного построения пользовательских интерфейсов. Развиваются не только компьютерв, но и сети. Еще несколько десятков лет Интернет представлял собой небольшую частную сеть, но теперь это миллиарды устройств и система, занимающая все большую часть в современной жизни.

Web-технологии изменили представление о работе с информацией. Оказалось, что традиционные параметры развития вычислительной техники — такие как производительность, емкость запоминающих устройств — не учитывали основного узкого места — интерфейса для взаимодействия с человеком.

Web-приложения представляют собой особый тип программ, построенных по архитектуре «клиент-сервер». Основные вычисления происходят на сервере, а клиент отвечает за взаимодействие с пользователем. Однако, технологии развиваются и на смену статичным страницам пришли интерактивные приложения с использованием JavaScript и jQuery. В настоящее время все популярнее становятся фреймворки, реализующие паттерн MVC. С их помощью логика клиентского приложения отделяется от представления, происходит стандартизация кодирования и увеличивает эффективность разработки.

Одним из основных инструментов для создания современных web-приложений является проект Angular.JS, разработанный и сопровождаемый компанией Google. Этот фреймворк имеет достаточно низкий порог вхождений, использует декларативный подход, отделяет логику от представления и позволяет создавать одностраничные приложния, в которых представления меняются без перезагрузки страницы. При этом взаимодействие с сервером происходит в асинхронном режиме.

Также нельзя обойти стороной мобильные устойства. По состоянию на начало 2015 года количество активных мобильных устройств уже давно привысило количество людей на планете[1]. В связи с этим повышается необходимость разработки мобильного кроссплатформенного приложения. Самым известным и используемым инструментом является Apache Cordova[2]. Дан-

ный фреймворк позволяет создавать универсальные мобильные приложения, работающие на различных мобильных платформах, с использованием стандартных web-технологий.

Целью данной работы является реализация кроссплатформенного приложения программной системы вероятностного моделирования ad-hoc сетей в рамках web-технологий. Для достижения данной цели были поставлены следующие задачи:

а) Изучить проблематику и примеры web-инструментов в реализованных web-приложениях.

б) Произвести обзор web-технологий, позволяющих создавать эффективные и удобные web-приложения с использованием шаблонов проектирования.

в) Реализовать свое web-приложение, взаимодействующее с web-сервером по протоколу SOAP и позволяющее производить вероятностное моделирование ad-hoc сетей.

г) Реализовать приложение с использованием web-технологий, функционирующие на широком спектре мобильных платформ.

Развитие современной компьютерной техники и внедрение новейших технологий вызвало появление новых программных продуктов. Развиваются не только компьютеры, но и сети. Если еще несколько десятков лет назад Интернет представлял собой небольшую частную сеть, то теперь это гигантская система взаимосвязанных компьютеров, без которой, возможно, мы не сможем представить себе жизнь. Web-технологии полностью перевернули представление о работе с информацией. Оказалось, что традиционные параметры развития вычислительной техники – производительность, пропускная способность, емкость запоминающих устройств – не учитывали главного «узкого места» системы – интерфейса с человеком. И только когда интерфейс между человеком и компьютером был упрощен до естественности восприятия обычным человеком, последовал беспрецедентный взрыв интереса к возможностям вычислительной техники[1].

Web-приложения представляют собой особый тип программ, построенных по архитектуре «клиент-сервер». Особенность их заключается в том, что само Web-приложение находится и выполняется на сервере – клиент при этом получает только результаты работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата. Передача запросов и результатов их обработки происходит через Интернет как представлено на рисунке 1

Рисунок 1 – Архитектура Web-приложения

За счет наличия исполняемой части, Web-приложения способны выполнять практически те же операции, что и обычные Windows-приложения, с тем лишь ограничением, что код исполняется на сервере, в качестве интерфейса системы выступает браузер, а в качестве среды, посредством которой происходит обмен данными, – Интернет. К наиболее типичным операциям, выполняемым Web-приложениями, относятся:

а) прием данных от пользователя и сохранение их на сервере;

б) выполнение различных действий по запросу пользователя: извлечение данных из базы данных (БД), добавление, удаление, изменение данных в БД, проведение сложных вычислений;

в) аутентифицирование пользователя и отображение интерфейса системы, соответствующего данному пользователю;

г) отображение постоянно изменяющейся оперативной информации и т.д.

Современные web-приложения – это порталы, предоставляющие услуги. Более четкую иерархию web-приложений можно посмотреть на рисунке 2 В настоящее время с точки зрения назначения различают три основных типа порталов:

а) Публичные, или горизонтальные, порталы (называемые иногда мега-порталами), такие как Rambler. Такие порталы нередко являются результатом развития поисковых систем. Предназначены они для самой широкой аудитории, что отражается на содержании предоставляемой ими информации и услуг. Как правило, эта информация носит общий характер, равно как и предоставляемые услуги (электронная почта, новостные рассылки и так далее).

б) Вертикальные порталы. Этот вид порталов предназначен для специфических видов рынка. Вертикальные обслуживает аудиторию, пользующуюся услугами этого рынка или работающую на нем. Примерами таких порталов могут служить туристические агентства, предоставляющие услуги по бронированию мест в гостиницах, заказу и доставке билетов, доступу к картам и сведениям об автомобильных маршрутах. Или порталы типа business-to-business, позволяющие своим клиентам

реализовывать совместные бизнес-операции (например, выбирать поставщиков и осуществлять закупку товаров, проводить аукционы).

в) Корпоративные порталы предназначены для сотрудников, клиентов и партнеров одного предприятия. Пользователи такого портала получают доступ к предназначенным им сервисам и приложениям в зависимости от их роли и персонального профиля[3].

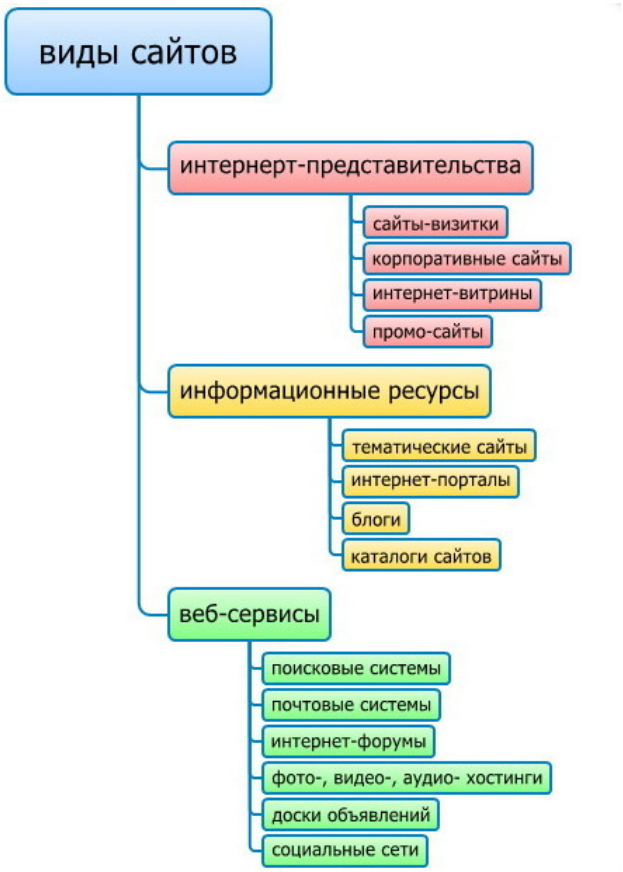

Рисунок 2 – Иерархия Web-приложений

Другие наиболее распространённые web-приложения:

а) Региональные Интернет-порталы, универсальные по своему направлению, но ограниченные географией заинтересованных посетителей (e1.ru).

б) Поисковые системы – это Интернет-порталы, которые предназначены для того, чтобы предоставить их посетителю возможность найти сайты, на которых встречаются заданные слова или целые фразы (yandex.ru, google.ru).

в) Каталог – это коллекция ссылок на сайты. Зачем же нужны каталоги, если есть поиск? Очень часто мы не знаем точно, что нам нужно, и не можем это сформулировать парой слов (mail.ru).

г) Электронные доски объявлений являются местом в интернете, где практически любой желающий может оставить информацию ознакомительного, пригласительного или рекламного характера.

д) Форумы – это специальные сайты или разделы на сайтах, предназначенные для того, чтобы посетители, оставляя свои сообщения, обменивались мнениями, задавали вопросы в поисках ответов.

е) Чаты – являются еще одним местом для общения в Интернет, только его назначение не обмен мнениями на какую-то тему, а просто времяпрепровождение.

ж) Интернет-магазины и аукционы[3].

1.2 Серверная часть

На стороне сервера Web-приложение выполняется специальным программным обеспечением (Web-сервером), который и принимает запросы клиентов, обрабатывает их, формирует ответ в виде страницы, описанной на языке HTML, и передает его клиенту.

В процессе обработки запроса пользователя Web-приложение компонует ответ на основе исполнения программного кода, работающего на стороне сервера, Web-формы, страницы HTML, другого содержимого, включая графические файлы. В результате, как уже было сказано, формируется HTML-страница, которая и отправляется клиенту. Получается, что результат работы Web-приложения идентичен результату запроса к традиционному Web-сайту, однако, в отличие от него, Web-приложение генерирует HTML-код в зависимости от запроса пользователя, а не просто передает его клиенту в том виде, в котором этот код хранится в файле на стороне сервера. То есть Web-приложение динамически формирует ответ с помощью исполняемого кода – так называемой исполняемой части.

PHP – скриптовый язык. В первую очередь PHP используется для создания скриптов, работающих на стороне сервера, для этого его, собственно, и придумали. PHP способен решать те же задачи, что и любые другие CGI-скрипты, в том числе обрабатывать данные html-форм, динамически генери-

ровать html страницы и тому подобное. Но есть и другие области, где может использоваться PHP. Вторая область – это создание скриптов, выполняющихся в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые будут исполняться, вне зависимости от web-сервера и браузера, на конкретной машине. И последняя область – это создание GUI-приложений (графических интерфейсов), выполняющихся на стороне клиента[**?**].

VBScript – язык создания сценариев VBScript разработан фирмой Microsoft, является подмножеством достаточно распространенного в среде программистов языка Visual Basic разработки прикладных программ Windows-приложений. Как и его родитель, язык VBScript достаточно прост и лёгок в изучении. Преимуществом его применения для создания сценариев является возможность использования, с небольшими корректировками, ранее написанных процедур на языках Visual Basic и Visual Basic for Application. Функциональные возможности сценариев, написанных на VBScript, ничем не отличаются от возможностей сценариев JavaScript: динамические создание документа или его частей, перехват и обработка событий и так далее. VBScript используется для написания сценариев клиента (в этом случае браузер должен иметь встроенный интерпретатор этого языка), а также для написания сценариев на сервере (в этом случае сервер должен поддерживать язык VBScript). Для создания сценариев клиента используется набор объектов, аналогичный набору JavaScript. Объекты клиента и сервера отличаются друг от друга, но существует общая часть (ядро) объектов, используемых при разработке как сценариев клиент, так и сценариев сервера[**?**].

Perl – скриптовый язык. Наиболее широко Perl используется для разработки инструментов системного администрирования, однако в последнее время он получил огромную популярность в области разработки Интернет-приложений: CGI-сценариев, систем автоматической обработки электронной почты и поддержки узлов Web[**?**]. Вот некоторые примеры задач, которые можно решать с помощью Perl:

а) проверка пользователей Windows NT на несоответствие их статуса и возможностей;

б) управление NT-сервисами из командной строки и дистанционно с локальной машины получение статистических данных на отдельной машине;

в) может работать и с протоколом FTP;

г) системная поддержка UNIX и Windows.

1.2 Клиентская часть

Отображением результатов запросов, а также приемом данных от клиента и их передачей на сервер обычно занимается специальное приложение – браузер. Как известно, одной из функций браузера является отображение данных, полученных из Интернета, в виде страницы, описанной на языке HTML, следовательно, результат, передаваемый сервером клиенту, должен быть представлен на этом языке. Рассмотрим основные инструменты, с помощью которых можно создать любой web-сайт.

HTML – язык разметки гипертекста (Hypertext Markup Language) – это компьютерный язык, лежащий в основе World Wide Web (Всемирной Паутины). Благодаря языку HTML любой текст можно разметить, преобразовав его в гипертекст с последующей публикацией в Web. Язык HTML имеет собственный набор символов, с помощью которых Web-браузеры отображают страницу. Эти символы, называемые дескрипторами, включают в себя элементы, необходимые для создания гиперссылок. Одной из отличительных особенностей HTML-документов является то, что сам документ содержит только текст, а все остальные объекты встраиваются в документ в момент его отображения Браузером с помощью специальных тэгов и хранятся отдельно. При сохранении HTML-файла в месте размещения документа создается папка, в которую помещаются сопутствующие ему графические элементы оформления[**?**].

Web-сайт должен быть не только функциональным, но и привлекательным. Для того чтобы приукрасить обычную HTML-страницу, наполненную текстом, понадобиться инструмент *Cascading Style Sheets* – формальный язык описания внешнего вида документа, написанного с помощью языка разметки. CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формаль-

ного языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода[**?**].

Но язык разметки не обладает логикой, не умеет обрабатывать данные – только отображать. Для того чтобы наделить клиентскую часть функционалом и создать интерактивный HTML-документ, используют язык программирования *JavaScript* . Это прототипно-ориентированный сценарный язык разработки встраиваемых приложений, выполняющихся как на стороне клиента, так и на стороне сервера[**?**]. Основные области применения JavaScript делятся на следующие категории:

а) динамическое создание документа с помощью сценария;

б) оперативная проверка достоверности заполняемых пользователем полей форм HTML до передачи их на сервер;

в) создание динамических HTML-страниц совместно с каскадными таблицами стилей и объектной моделью документа;

г) взаимодействие с пользователем при решении «локальных» задач, решаемых приложением JavaScript, встроенном в HTML-страницу.

Частая перезагрузка страницы снижает производительность и удобство работы с Web-сайтом. Для того чтобы на каждый запрос сервер не выдавал новую страницу, а отсылал лишь те данные, которые нужны клиенту, в HTML из них прямо в браузере формирует движок Ajax. *Ajax* расшифровывается как Asynchronous Javascript And XML и технологией в строгом смысле слова не является. Он определяет, какие запросы можно обработать «на месте», а за какими необходимо обращаться на сервер. Асинхронность проявляется в том, что далеко не каждый клик пользователя доходит до сервера, причем обратное тоже справедливо – далеко не каждая реакция сервера обусловлена запросом пользователя. Большую часть запросов формирует движок Ajax, причем его можно написать так, что он будет загружать информацию, предугадывая действия пользователя[**?**]. Где стоит использовать Ajax:

а) Формы. Если асинхронно передавать данные, страница не перезагружается, что заметное ускоряет работу.

б) Навигация в виде «дерева». Такая навигация не является удобной и лучше использовать простую топологию, но если уж до этого дошло, лучше использовать Ajax.

в) Голосования. Пользователю будет приятней оставить свой голос за несколько секунд, чем за 30-40.

г) Фильтры. Часто на сайтах делают сортировку по дате, по имени. Ajax это будет значительно удобнее.

jQuery– библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с Ajax. jQuery обладает широким спектром возможностей, главными из которых являются визуальные эффекты, Ajax-дополнения и JavaScript-плагины[**?**].

2 Выбор инструментов для реализации web-приложения

2.1 Single Page Application

Основной задачей данного проекта является реализация web-приложения, взаимодействующее с web-сервером по протоколу SOAP. Результатом проекта должно получится приложение, которое сможет быстро и мгновенно реализовывать функциональнные операции. В настоящее время в сфере web разработки наряду с многостраничными сайтами (Multi-page Application, MPA) широкое распространение получили и одностраничные приложения (Single-page Application, SPA), формирование контента которых происходит динамически на стороне клиента. В связи с этим было решено реализовывать одностраничное приложение. На рисунках 3 и 4 представлена схема работы многостраничного и одностраничного сайта соответственно.

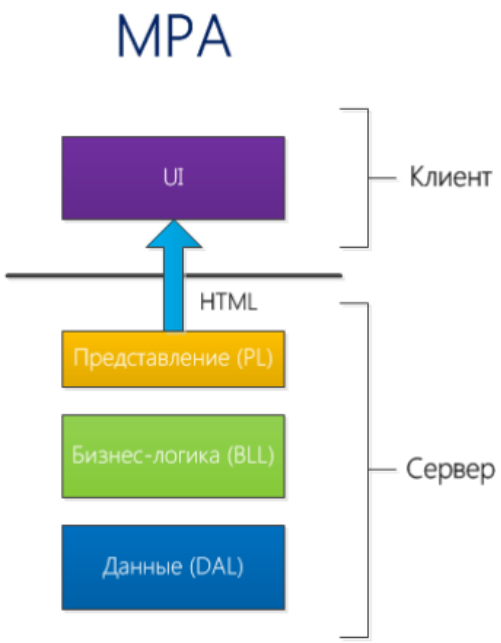

Рисунок 3 – Архитектура Multi-page Application

Single Page Application – это web-приложение, размещенное на одной странице. Применяя такую технологию, можно создать сайт, который будет представлять собой одну html-страницу, интерактивность которой обеспечивается скриптами. Работа такого web-сайт максимально полностью перенесена на сторону клиента. Сайт «общается» с сервером только чистыми данными, без загрузки html-контента.

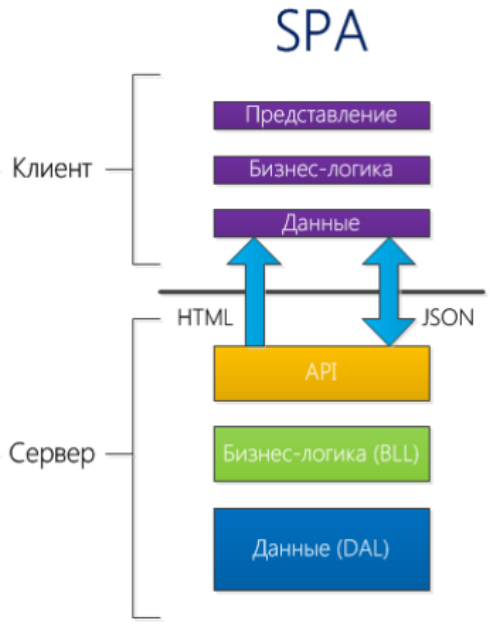

Рисунок 4 – Архитектура Single Page Application

Для того чтобы написать одностраничное приложение необходимо придерживаться определенных правил:

а) Все сущности приложения основаны на моделях и объектах. Внутри объектов инкапсулирована работа с DOM-элементами страницы.

б) Насколько позволяет структура хранит HTML шаблоны в скриптах

в) При любые изменения на странице динамически изменяют url.

г) Прямая загрузка любого url должна отобразить соответствующую страницу с данными.

д) Обработчик события History back, что соответствует кнопки назад в браузере, должен выполняться корректно и возвращать страницу в предыдущее состояние.

е) Кеширование моделей данных на стороне клиента.

Если учитывать эти основные правила, в результате получится эффективное, быстрое и полнофункциональное одностраничное web-приложение. Но как и любой продукт, приложение обладает рядом преимуществ и недостатков. Рассмотрим плюсы и минусы данного подхода и почему SPA все равно остается популярным.

К преимуществам SPA можно отнести следующее:

а) Работа на большом количестве устройств. Приложения на SPA отлично работают на устройствах как стационарных, так и мобильных. Пер-

сональные компьютеры, планшеты, смартфоны могут беспрепятственно работать с сайтами построенных по принципу SPA. Создав одно приложение, мы получим гораздо большую аудиторию пользователей нежели при использовании стандартного подхода.

б) Богатый пользовательский интерфейс. Так как web-страница одна, построить функцинальный и приятный пользовательский интерфейс гораздо проще. Не так затруднительно хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией.

в) Отсутствие загрузки одного и того же контента снова и снова. Если сайт использует шаблон, то вместе с основным содержанием какой-либо страницы посетитель сайта обязательно загружает разметку шаблона. Конечно, кэширование данных на данном этапе развития в web-программировании достигло высоких результатов, но если нечего кэшировать, то время и ресурсы на это не тратятся.

Самым главным неудобством при разработке SPA – это работа с языком программирования JavaScript. JavaScript изначально позиционировался как простой язык программирования с Java-подобным синтасисом. Но JavaScript не обладает статической типизацией, его объектная модель не является привычной для многих разработчиков. Отладка кода представляет собой трудный процесс. Кроме того, различные интернет-обозреватели могут по-разному интерпретировать JavaScript-код. Поэтому разработка требуемого приложения с использованием исключительно языка JavaScript является довольно трудоёмким процессом. Чтобы как-то исправить несовместимость с некоторым браузерами, приходится писать отдельный код для различных клиентов. Таким образом, размер кода возрастает, а функциональность нет. В итоге приходится основную часть времени тратить на обработку особенностей выполнения кода различными движками, а не на реализацию продукта. Частично последнюю проблему можно решить использованием специальных библиотек, примеру, библиотека jQuery.

2.2 JavaScript-фреймворки

На смену библиотекам вроде jQuery в мир JavaScript приходят фреймворки, реализующие функциональную схему Model-view-controller.

Model-view-controller (MVC) – схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Архитектура работы MVC представлена на рисунке 5

Рисунок 5 – Схема Model-view-controller

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

а) Модель (англ. Model). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

б) Представление, вид (англ. View). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.

в) Контроллер (англ. Controller). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как *представление*, так и *контроллер* зависят от *модели*. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Преимущества фреймворков видны невооруженным глазом. Один из самых существенных является избавление от рутинного кода, который тянется

от проекта к проекту. Фреймворк предоставляет разработчикам каркас будущего приложения и решение задач, встречающихся в большинстве проектов. Например, программисту не нужно думать, как принять данные от клиента и передать их на сервер, т.к. все необходимое скорей всего реализовано авторами фреймворка. Вместо этого разработчику предлагается сосредоточиться функционалом собственного приложения.

Другим немаловажным плюсом всех фреймворков является стандартизация кодирования. Если разработчик решается применять готовый каркас в своем проекте, то он должен быть готовым следовать его заповедям. Это значит, что ему нужно не полениться - один раз ознакомиться с правилами и быть спокойным, что в последствие код без проблем может дорабатываться другими разработчиками.

Очень часто многие разработчики задают вопрос - Чем один Javascript фреймворк лучше другого? На этот вопрос трудно ответить, ведь каждый фреймворк обладает определенным набором инструментов и имеет свой круг задач, с которыми он успешно справляется. Выбор JavaScript MVC фреймворка — тяжёлая работа. Нужно учесть много факторов, и число вариантов выбора может быть огромно.

Для создание приложения на основе SPA необходимо отобрать несколько фреймворков, чей функционал справится с поставленной задачей, рассмотреть сильные и слабые стороны каждого и выбрать подходящий вариант.

2.2.1 Angular

2.2.2 BackBone

Приложения на Backbone не придерживаются строгой архитектуры. Основная идея, которую несёт документация: используйте инструменты этого фреймворка так, как вам хочется. Благодаря такому подходу Backbone хорош для абсолютно разных задач, и на нем очень просто начать писать приложения. Однако с другой стороны, это приводит к тому, что новички совершают ошибки в самом начале работы с ним.

Работая с Backbone, данные представляются как Модели (Models), которые могут быть созданы, провалидированы, удалены, и сохранены на сервере. Всякий раз, когда в интерфейсе изменяется атрибуты модели, модель вызы-

вает событие "change"; все Представления (Views), которые отображают состояние модели, могут быть уведомлены об изменении атрибутов модели, с тем чтобы они могли отреагировать соответствующим образом — например, перерисовать себя с учетом новых данных. При изменении модели представление просто обновит себя самостоятельно.

Рассмотрим поподробней сущности Backbone, их предназначение и минимальную реализацию:

а) Backbone.Model; Model – это единица данных. Отвечает за получение, отправку, хранение, валидацию и прочие манипуляции с данными какой-то сущности. Простая реализация такой единицы данных представлена на рисунке 6.

б) Backbone.Collection; Collection – это упорядоченный набор моделей. Отвечает за получение и отправку данных какой-то сущности, а так же манипуляции с моделями (создание, обновление, удаление). Работает только с моделями определенного типа. Простая реализация такого списка представлена на рисунке 7.

в) Backbone.View; View – это представление модели или коллекций, отвечает за реализацию интерфейса. Отвечает за рендеринг модели или коллекции, работу с шаблонами, обработку событий и другое. Простая реализация представлена на рисунке 8.

г) Backbone.Router; Router предоставляет методы для маршрутизации на стороне клиента, а также связывания этих действий с событиями. Простая реализация таких методов представлена на рисунке 9.

2.2.3 Сравнение Angular и BackBone

Перед тем как реализовывать приложение, были тчательно изучены и опробованы оба фреймворка. Были выбраны характеристики, которые являются важными при реализации приложения, и на основе полученных данных произвели сравнение фреймворков. характеристики для сравнения:

а) Порог входа и документация

б) Продуктивность разработки

в) Набор функций

г) Размер

```
var sampleModel = Backbone.Model.extend({
    url: '/path/to/data',
    defaults: {
        //Значения атрибутов по-умолчанию
    },
    initialize: function () {
        //Конструктор модели
    },
    someMethod: function () {
        //Тело метода
    }
});
```

Рисунок 6 – Backbone.Model

```
var sampleCollection = Backbone.Collection.extend({
    url: '/path/to/data',
    model: sampleModel,
    initialize: function () {
        //Конструктор коллекции
    },
    filteredByName: function (name) {
        return this.filter(function (model) {
            return model.get('name') === name;
        });
    },
});
```

Рисунок 7 – Backbone.Collection

д) Защита от утечки памяти

Порог входа и документация

Angular легко позволяет делать сложные вещи такие, как двунаправленная синхронизация. Но после освоения базовых знаний, порог обучения становится выше: открывается сложная структура с большим количеством особенностей. Чтение документации затруднено из-за специфического жаргона и малого количества примеров.

Backbone — довольно прост в освоении. Но после длительного использования вы можете обнаружить, что не хватает понимания того, как лучше структурировать код. Чтобы исправить сложившуюся ситуацию необходимо просмотрет или прочитать несколько учебников.

```
var sampleView = Backbone.View.extend({
    tagName: 'div',
    className: 'someDiv',
    initialize: function () {
        //Конструктор представления
        this.model.on('change', this.render, this);
    },
    render: function () {
        var data = this.model.toJSON();
        this.$el.html(_.template(this.template, data));
        return this;
    }
});
```

Рисунок 8 – Backbone.View

```
var sampleRouter = Backbone.Router.extend({
    routes: { //Словарь роутов и экшенов
        'index': 'index',
        'search/:id': 'search'
    },
    initialize: function () {
        //Конструктор роута
    },
    index: function () {
        //Тело метода
    },
    search: function (id) {
        //Тело метода
    }
});
```

Рисунок 9 – Backbone.Router

Продуктивность разработки

Разработка с помощью Angular будет достаточно быстрой и эффективной, после того, как изучить его основные функции и принцип работы.

Backbone требует написания очень большого объёма шаблонного кода. Что оказывается прямой угрозой производительности труда и является неэффективным.

Набор функций

Среди основного набора функций, которым оладает схема MVC является:

а) Реализация паттерна «Наблюдатель»: объекты, изменения которых отслеживаются.

б) Наличие автоматически изменяемых представлений

в) Представления (визуализация шаблонов), включающие другие представления.

г) Показ представлений по некоторым условиям.

д) Использование автоматически изменяемых представлений, когда наблюдаемый объект изменяется.

Angular полный набор перечисленных функций.

Backbone, в свою очередь, иметь только реализацю паттерна «Наблюдатель»

Размер и зависимости

Является важным фактором для мобильной разработки.

Angular имеет размер в 80 килобайт, однако это едиственный фреймворк, не требующий дополнительных библиотек.

Backbone считается самым маленький фреймворк, но требует как минимум две библиотеки, что увеличивает размер фреймворка до 61 килобайта.

Защита от утечки памяти

Так же является важным фактором для длительно открытых одностраничных приложений.

С фреймворком Angular можно эффективно решать проблему с утечкой памяти, даже не обладая огромным опытом в разработке.

Что нельзя сказать о фреймворке Backbone. При недостаточных знаний эта проблема окажется глобальной.