Дипломная работа

Дроздов Олег Михайлович

Группа АС561

Научный руководитель: Касьяник В.В. ст. преп.

Тема: программное средство (web-приложение) управления продажами (клиентская часть)

Реферат дипломной работы Дроздова Олега Михайловича

Создание клиентской части приложения по управлению продажами, визуализация статистики, обеспечение работы в последних версиях современных браузеров.

Для достижения цели дипломного проекта была спроектирована архитектура SPA ( Single page application), позволяющая горизонтально масштабировать каждый элемент приложения независимо друг от друга. Разработано web-приложение, предоставляющее JSON RESTful API для клиентской части приложения и обеспечивающее обновление данных в режиме реального времени.

**План**:

1. *Анализ существующих технических решений и выбор варианта, наиболее полно удовлетворяющего требованиям пользователей на создание автоматизированных систем обработки данных (подсистем, отдельных видов обеспечения). Выбор производится на основе анализа современных достижений в данной области.*
2. *Проектирование, реализация и испытание АСОИ (подсистем, отдельных задач или видов обеспечения) с теоретическим обоснованием выбранных решений.*
3. *Технико-экономическое обоснование создаваемой АСОИ (подсистемы или отдельного вида обеспечения).*

**Титульный лист — 1.**

**Лист задания — 1.**

**Содержание — 1.**

**Введение — 2.**

**1. Системный анализ и постановка задачи на создание АСОИ — 10.**

**2. Проектирование АСОИ — 30.**

**3. Реализация и испытание АСОИ — 10.**

**4. Технико-экономическое обоснование — 7.**

**Заключение — 1.**

**Список сокращений - 1**

**Список литературы - 1 +**

**Приложения — (по необходимости).**

ВВЕДЕНИЕ

Анализ методов и средств разработки масштабируемых WEB-приложений

Обзор возможных вариантов архитектуры WEB-приложений

**Одностраничные приложения**

С развитием веб-технологий меняются и подходы к разработке веб-приложений. Поддержка браузерами новых веб-стандартов позволила создавать приложения с богатой клиентской логикой. Одним из популярных подходов для построяния такого рода приложений являются одностраничные приложения (Single Page Application).

Single page application (SPA) — это веб-приложение, которое выполняется непосредственно на стороне клиента в Web-браузере, обычно написанное на комбинации из HTML, JavaScript и CSS. - википедия.

Сначало немного истории о том, как развивались веб-приложения.

**Multiple Page Application (MPA) ??????????????????? УБРАТЬ СЛЕНГ**

Большиство веб-приложений сейчас явлется многостраничными.

Вся их суть заключается в том, что при запросе страницы сервер отправляет клиенту кусок HTML кода, который в свою очередь как-то рендерится клиентом (браузером). Пользователь переходит по ссылке на другую страницу и цикл повторяется вновь.

Почему так?

Раньше браузер рассматривался как инструмент для отображения разметки. Их возможности были скудными и все что могли сделать разработчики - это написать код, который передаст клиенту кусок разметки в определенном формате (HTML), с целью ее отображения.

Браузеры со временем развивались, в них появилась возможность выполнения сценариев, более богатое стилевое оформление и т.д. И с этого момента для повышения отзывчивости пользовательского интерфейса, отправляемые куски HTML кода стали "приправлять" скриптами, чтобы страница стала более динамичной и "живой".

Возникло множество инструментов, которые облегчали разработку таких многостраничных веб-приложений. Они позволяют легко формировать динамическое содержимое страницы (например Asp.net MVC), основываясь на какой-либо модели данных.

И все вроде бы хорошо, но основная проблема тут кроется в том, что представление страницы в виде HTML формируется на стороне сервера, а динамически управлять им приходится на стороне клиента. Множество клиентских плагинов и сценариев на странице ждут от сервера определенно сформированной разметки, чтобы в дальнейшем как-то ей управлять. Получается, что логика работы приложения распылена между клиентом и сервером.

Также при переходе пользователя на новую страницу - ваше клиентское состояние "убивается" и инициируется заново.

Все это конечно не может радовать и в связи с развитием браузеров появился новый подход в разработке веб-приложений.

**Single Page Application (SPA)**

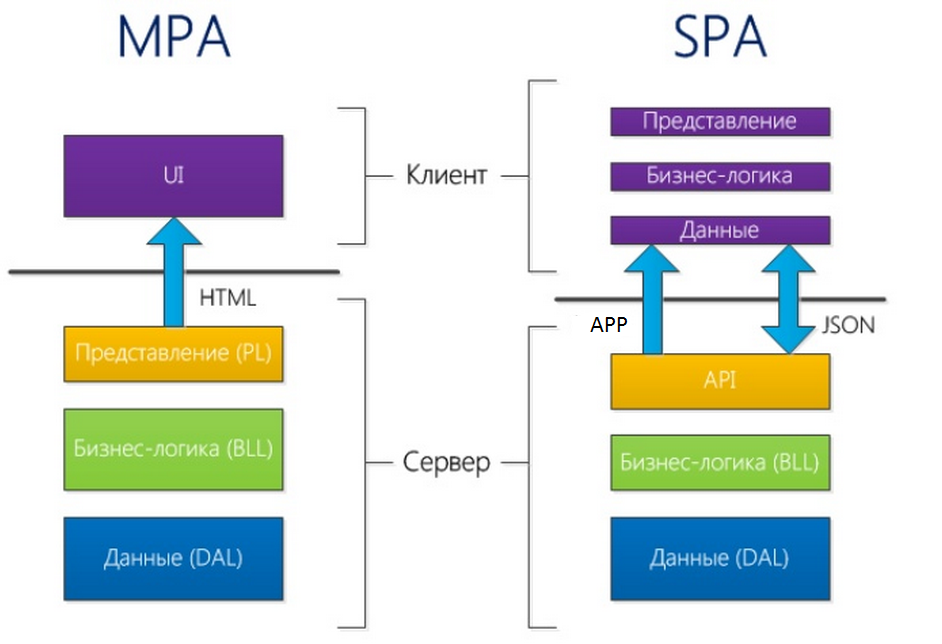
Это относительно новый подход. В чем его принципиальное отличие?

Представьте, что вы разрабатываете приложение под какую-либо операционную систему, будь то Window, Linux или iOS. Какой здесь подход? Операционная система - это хостинг площадка, которая предоставляет некоторый API для всех приложений, которые на базе нее работают. Вы разрабатываете приложение с использованием этого API. И это стандартный подход в разработке приложений.

SPA реализует этот подход. Вот основные компоненты:

* Браузер. здесь он выступает как хостинг площадка для вашего веб-приложения. У него есть определенный набор API (HTML5), который вы можете использовать.
* Удаленный сервис. Скорее всего вашему приложению API одного браузера будет недостаточно. Например, документ должен быть как-то передан и сохранен у вас на сервере и т.д. Поэтому от вас потребуется еще и сервис, с которым будет работать веб-приложение.
* Веб-приложение, которое будет работать на базе API браузера и использовать ваш удаленный сервис для взаимодействия.

Вот изображение для бОльшего понимания различий между SPA и MPA:



Как видно в случае SPA основная задача сервера - это предоставить браузеру само веб-приложение (клиентское) и API для взаимодействия. В веб-приложение входит:

* Стили (CSS);
* Шаблоны представлений;
* Клиентский код (бизнес логика работы приложения).

Загруженное в браузер веб-приложение получает данные от сервера и использует клиентские шаблоны для построения интерфейса. Когда пользователь как-то взаимодействует с ним (например нажимает на кнопку, ссылку или другой пользовательский элемент), приложение загружает новые данные, перерисовывая часть своего интерфейса. Страница браузера при работе приложения **не**перезагружается никогда (отсюда и название Single Page Application - одностраничное приложение).

Очень **важно отметить**, что сервер (в отличии от MPA) **не** передает куски готовой разметки клиенту. Клиент и сервер обмениваются только **чистыми данными** в формате JSON (как правило).

**Преимущества SPA**

Вот некоторые преимущества использования подхода SPA:

**Более грамотная архитектура**

Возможно это может показаться не очивидным, но на самом деле это так. С учетом масшатбного развития браузеров как платформы для хостинга приложений, подход MPA теряет свою актуальность.

SPA жестко делит клиентскую и серверную логику и это значительно упрощает проектирование и развитие каждого из этих уровней. Пользовательский интерфейс и логика его работы могут без проблем быть изменены, не касаясь логики прикладного уровня.

Как следствие этого вы также можете заменить серверную часть и, если вы не поменяли API, - работа клиентской логики не пострадает.

Также такой подход существенно упрощает процедуру разработки. Фронт-енд разработчики договариваются с бек-енд разработчиками о формате взаимодействия, новом API, а дальше каждый "пилит" свою часть. Фронт-енд разработчикам теперь не нужно знать специфику серверной части (Razor, Jade и т.д), чтобы делать свою работу. Эти 2 "мира" теперь могут существовать и развиваться независимо и паралельно.

**100% функциональности доступно через API**

Это преимущество вытекает из предыдущего. Ваше веб-приложение - это теперь частный случай клиента, работающего с API вашего сервиса. Решили реализовать нативный мобильный клиент? Пожалуйста, он может работать на базе того же сервиса, что и веб-приложение.

**Отзывчивый пользовательский интерфейс**

Очень важным является то, что SPA - это более отзывчивые и адаптивные приложения, свободные от рваного эффекта перезагрузки страницы и ее рендеринга заново.

Если ваше веб-приложение подразумевает сложные формы и сценарии взаимодействия с пользователем, с помощью SPA вы сможете реализовывать их значительно проще и быстрее.

### Недостатки

**Различность клиентов**

Увы, но пользователи используют разные браузеры, и так как клиентского кода в случае с SPA значительно больше, то и вероятность исключительных ситуаций в разных версиях браузеров выше.

Также, если вам требуется поддержка старый версий браузеров, то SPA - не ваш выбор, т.к. для ее реализации требуется поддержка современных веб-стандатров. Для полноценного SPA необходимо хотя бы IE9+. Но стоит иметь ввиду, что доля старых браузеров не так велика и с каждым днем сокращается. Поэтому делать ставку на поддержку таких браузеров не всегда целесообразно.

В защиту SPA можно отметить, что со временем браузеры начинают работать все быстрее и стабильнее, API стандартизуется и сейчас, в современных версиях, эта проблема ощущается не так остро. Также ее можно решить с помощью создания системных тестов и запуске их на списке поддерживаемых бразеров.

**Много клиентского кода**

... который дольше загружается в браузер, чем готовая HTML разметка (в случае с MPA).

Решить эту проблему можно разбиением вашего большого приложения на логические модули, которые будут подгружаться асинхронно в случае необходимости, и кешироваться в дальшейшем у клиента.

**Новые подходы**

Конечно для реализации этого подхода требуютcя другие знания, инструменты и подходы, отличные о тех, с которыми все привыкли работать. Требуется их тщательно изучить, чтобы в последствии грамотно использовать. Об инструментах мы сейчас и поговорим.

### Инструменты

**Сервер**

Координальным отличие здесь является то, что сервер, в случае с SPA, не является ответственным звеном по построению разметки. Нет знакомых механизмов представлений и данные для них - также не нужно подготавливать. Поэтому вам не потребуется большинство веб-фреймворков, которые предоставляет серверную шаблонизацию (например ASP.NET MVC/web forms, Ruby on rails и т д).

Если говорить про .NET, то для организации такого взаимодействия подходит новый [OWIN](http://owin.org/) хостинг от Microsoft - [Katana](https://katanaproject.codeplex.com/). В нем нет ничего лишнего и вы вольны сами выстраивать нужный вам pipeline обработки запроса. Поключите WebApi для реализации REST API сервиса. Нужно организовать PUSH уведомления - используйте SignalR и т.д.

**Клиент**

Самое важная часть на клиенте - это веб-приложение. Нужно продумать его архитектуру, чтобы оно не начало "разваливаться" при наборе "мышечной массы".

Сейчас в этом помогают множество клиентских MV\* фреймворков. Я могу порекомендовать AngularJS от Google. Он предоставляет все необходимо для построения SPA "из коробки", активно развивается, и навязывает грамотные подходы в реализации. Наша команда использует его и вполне довольна.

### Заключение

Сейчас все новые технологии и веб-стандарты направлены на упрощение и повышение эффективности разработки одностраничных приложений (html5, owin,[web components](http://www.w3.org/TR/components-intro/)) и этот тренд продолжится. Лично я не вижу в будущем веба место для MPA

Поэтому, если вы начинаете новый проект, в котором есть повышенные требования к гибкости и отзывчивости пользовалького интерфейса, планы на реализацию других клиентов (например мобильных) - советую обратить внимание на одностраничное приложение

С чего начать разработку? Куда двигаться? Что и как использовать? Об этом поговорим в следующих статьях.

**http://skyliver.ru/blog/tag/JavaScript**

СПИСОК УПОТРЕБЛЯЕМЫХ СОКРАЩЕНИЙ И ТЕРМИНОВ

API (Application Programming Interface, «Интерфейс программирования приложений» либо «Интерфейс прикладного программирования») — Набор готовых констант, структур и функций, используемых при программировании пользовательских приложений и обеспечивающих правильное взаимодействие между пользовательским приложением и системой.

REST (Representation State Transfer, «Передача состояния представления» либо «Передача репрезентативного состояния») — Стиль построения архитектуры распределённого приложения.

JSON (JavaScript Object Notation) — Текстовый формат обмена данными, основанный на JavaScript.

HTTP (HyperText Transfer Protocol, «Протокол передачи гипертекста») — Протокол прикладного уровня передачи данных.