Исследование и сравнение современных реализаций Flux-архитектур разработки веб-приложений

Скороходов И. С.[1], Тихомирова А. Н.[2]

*[1]Скороходов Иван Сергеевич / Skorokhodov Ivan Sergeevich – бакалавр менеджмента, магистрант,
кафедра экономики и менеджмента в промышленности,
факультет управления и экономики высоких технологий;
[2]Тихомирова Анна Николаевна / Tikhomirova Anna Nikolaevna – кандидат технических наук, доцент
кафедра кибернетики,
факультет кибернетики,
Федеральное государственное автономное образовательное учреждение высшего образования,
Национальный исследовательский ядерный университет, Московский инженерно-физический институт,
г. Москва*

Аннотация: *в данной работе исследуются современные реализации Flux-архитектур разработки веб-приложений, а также проводится их анализ и сравнение. Flux-архитектуры зародились совсем недавно: буквально через год после релиза реактивного шаблонизатора React от компании Facebook, использующей виртуальное DOM-дерево. Сегодня существует большое количество архитектурных решений, позволяющих работать с ним, поэтому выбор между ними значительно затруднен.*

Ключевые слова: *Flux-архитектура, React, веб-приложения, фронтенд-разработка.*

Введение

Научно-технический прогресс неустанно привносит в нашу жизнь множество полезных вещей. Достижения в робототехники, электронике, информатике, связи и других отраслях навсегда изменили способ взаимодействия человека с окружающим миром. Но самой влиятельной и быстроразвивающейся отраслью является Интернет.

Интернет — глобальная компьютерная сеть, охватывающая весь мир. По разным данным доступ в Интернет имеют от 15 до 30 миллионов людей в более чем 150 странах мира. Ежемесячно размер сети увеличивается на 7 – 10 процентов. Интернет образует как бы ядро, обеспечивающее связь различных информационных сетей, принадлежащих различным учреждениям во всем мире, одна с другой. Если ранее сеть использовалась исключительно в качестве среды передачи файлов и сообщений электронной почты, то сегодня решаются более сложные задачи распределения доступа к ресурсам [1].

Развитие персональных компьютеров значительно расширило использование Интернета. Изначально он существовал на них в виде обособленных приложений — так называемых «настольных», каждое из которых взаимодействовало напрямую с операционной системой. Сейчас подавляющее большинство пользований сетью происходит через «окно в Интернет» — интернет-браузер, который сам по себе является настольной программой и позволяет пользователю запрашивать ресурсы с удаленных серверов и выполнять их. Мало того, появились даже специальные компьютеры, операционная система которых — это фактически интернет-браузер.

Такая тенденция произошла по двум основным причинам:

1. Это удобнее для пользователей — у них появляется возможность иметь только одну установленную программу, чтобы получить сразу мириады возможностей [3].

2. Это удобнее для разработчиков — они могут «создавать приложения только один раз» — то есть, им не приходится тратить дополнительные ресурсы на поддержку других платформ: средой выполнения программ является интернет-браузер, который поддерживает (по крайней мере, должен поддерживать) единые стандарты и единый программный интерфейс [3].

Существует множество типов архитектур создания веб-приложений, традиционной для веб-разработки является так называемая архитектура MVC («model-view-controller», «модель-представление-контроллер») — 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. Для реализации данной архитектуры написано огромное количество библиотек и «архитектурных каркасов», значительно упрощающих и ускоряющих разработку [4].

Но в последнее время начал приобретать подход реактивного программирования — парадигма программирования, ориентированная на потоки данных и распространение изменений. Ее зарождение обязано разработке компанией Facebook, одной из крупнейших технологических компаний в истории, крайне быстрого шаблонизатора (программного обеспечения, позволяющее использовать html-шаблоны для генерации конечных html-страниц) React, который является реактивным — то есть обновление данных, на которые завязаны шаблоны, автоматически обновляет их html-представление, и разработчику не приходится тратить время на создание логики обновления самому. Будучи реактивным

шаблонизатором, React особенно повышает производительность разработчика, если архитектура всего приложения также является реактивной. Поэтому компания Facebook разработала новый «архитектурный каркас», называемый Flux [5].

Во многом Flux является больше концепцией, чем библиотекой функций, что позволило множеству программистов по всему миру разработать на его основе модификации этой архитектуры и библиотеки, их реализующих.

Целью данной работы является исследование и сравнение существующих Flux-архитектур, а также библиотек, упрощающих их реализацию. Для достижения данной цели были поставлены следующие задачи:

- рассмотрение современных тенденций разработки веб-приложений, а именно SPA — одностраничных приложений;
- сравнение Flux и MVC архитектур разработки веб-приложений;
- сравнение различных Flux-архитектур и их реализаций.

Научная новизна и актуальность данного исследования обуславливается тем, что начав зарождаться буквально около года назад, Flux-технологии сегодня являются беспрецендентно передовыми, находясь на пике прогресса современной теории разработки клиентской части веб-приложений.

Одностраничные веб-приложения

На заре развития Интернета все сайты были статическими. Статический сайт — это сайт, состоящий из статичных html (htm, dhtml, xhtml) страниц составляющих единое целое. Содержит в себе текст, изображения, мультимедиа содержимое (аудио, видео) и HTML-теги. Теги бывают как служебные, предназначенные для обозревателя, так и предназначенные для размещения, формирования внешнего вида и отображения информации. Все изменения на сайт вносятся в исходный код документов (страниц) сайта, для чего необходимо иметь доступ к файлам на веб сервере. Статические сайты имеют следующие недостатки:

- Невозможность динамической генерации содержимого.
- Невозможность полноценной поддержки посетителей (выбор внешнего вида, поддержка браузеров, cookie).
- Для наполнения сайта информацией необходимо получать доступ к файлам-страницам посредством FTP, или сторонним веб-скриптам, позволяющим редактировать страницы.
- При большом количестве страниц (файлов), если возникает необходимость внести однотипные изменения (дизайн, оформление, добавление новых разделов) необходимо использовать стороннее ПО (утилиты).

В связи с этими недостатками, особенно отсутствием возможности динамической генерации содержимого, начали появляться веб-приложения, чтобы дать пользователю больше возможностей. Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами. Веб-приложения стали широко популярными в конце 1990-х — начале 2000-х годов [3].

Существенное преимущество построения веб-приложений для поддержки стандартных функций браузера заключается в том, что функции должны выполняться независимо от операционной системы данного клиента [6].

В связи с архитектурным сходством с традиционными клиент-серверными приложениями, в некотором роде «толстыми» клиентами, существуют споры относительно корректности отнесения подобных систем к веб-приложениям; альтернативный термин «Богатое Интернет приложение».

Веб-приложение состоит из клиентской и серверной частей, тем самым реализуя технологию «клиент-сервер».

Клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него.

Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует веб-страницу и отправляет её клиенту.

В настоящее время широко используется технология AJAX. При ее использовании страницы веб-приложения не перезагружаются целиком, а лишь догружают необходимые данные с сервера, что делает их более интерактивными и производительными.

Также в последнее время набирает большую популярность технология WebSocket, которая не требует постоянных запросов от клиента к серверу, а создает двунаправленное соединение, при котором сервер может отправлять данные клиенту, без запроса от последнего.

Один из недостатков веб-приложений является необходимость клиенту для каждого сеанса использования выкачивать с сервера исходный код программ. Таким образом, для того, чтобы повысить

удобство использования своих приложений, разработчик старается повышать скорость загрузки исходного кода и его выполнение.

Существует множество техник, способствующих этому, одна из которых является минимизация случаев полной перезагрузки страницы. Веб-приложения, вся работа которых осуществляется вообще без перезагрузки страниц, называются одностраничными (SPA — “single page application”), то есть они используют единственный HTML-документ как оболочку для всех веб-страниц и организуют взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX. SPA напоминают настольные приложения намного больше, чем традиционные веб-сайты [3].

Но вместе с этим у разработчика появились дополнительные трудности, которые были вызваны совокупностью совокупностью факторов: асинхронностью загрузки данных, слабыми возможностями языка разработки, поддерживаемого платформой, медленностью DOM и желанием создавать насыщенные веб-приложения со множеством различных представлений и широким функционалом. [8]

На помощь пришли «архитектурные каркасы», чаще называемые фреймворками — программные платформы, определяющие структуру программной системы, облегчающие разработку и объединение разных компонентов большого программного проекта. Наиболее популярной подобной архитектурой в веб-разработке является архитектура MVC.

Обзор архитектур MVC и Flux

Как уже говорилось, наиболее популярной архитектурным решением при разработки веб-систем является 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 [7].

Основная цель применения этой концепции состоит в отделении бизнес-логики от её визуализации. За счет такого разделения повышается возможность повторного использования. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

К одной модели можно присоединить несколько представлений, при этом не затрагивая реализацию модели.

Не затрагивая реализацию представлений, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных), для этого достаточно использовать другой контроллер.

Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики, вообще не будут осведомлены о том, какое представление будет использоваться.

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

1. 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.

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

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

Их взаимодействие представлено на рисунке 1.

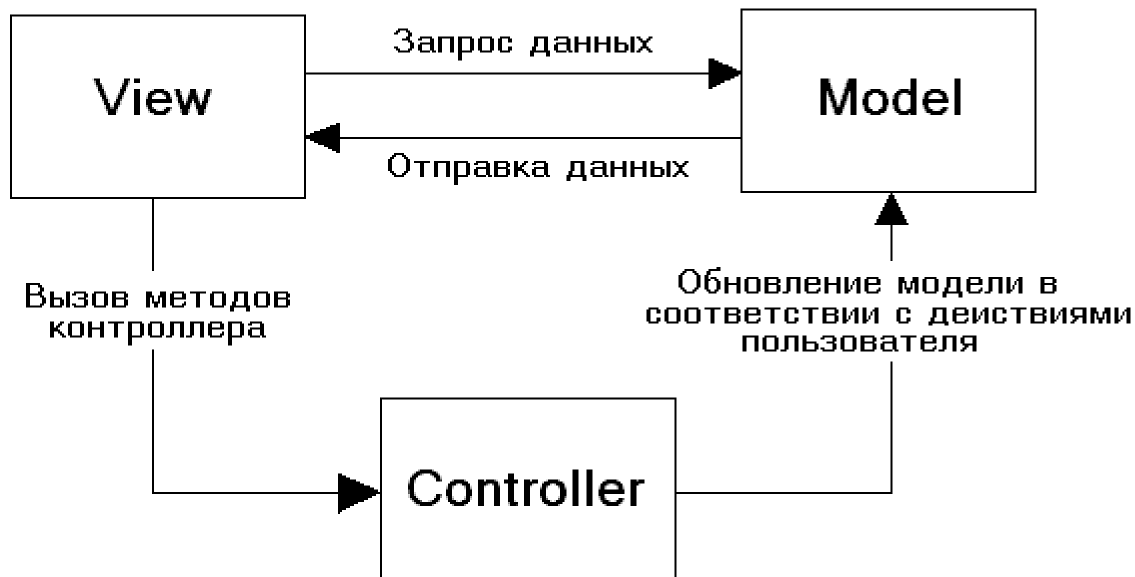

Рис. 1. Взаимодействие модели, представления и контроллера в парадигме MVC

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 [8].

Реализовать архитектуру MVC помогают множество фреймворков, наиболее популярными из которых являются Backbone, Ember и Angular. Но данные фреймворки не смогут исправить основного недостатка парадигмы MVC — неоднозначности в потоке данных: в больших приложениях крайне сложно понять, к какому результату приведут какие действия. Это приводит к тому, что допустить ошибку становится крайне просто и делает сам код менее понятным и поддерживаемым, особенно при разработке больших систем. Таким образом, все большую популярность начинает приобретать архитектура Flux.

Flux — это архитектура, которую команда Facebook использует при работе с React. Это не фреймворк или библиотека, а новый архитектурный подход, который дополняет React и принцип однонаправленного потока данных.

Тем не менее, Facebook предоставляет репозиторий, который содержит реализацию Dispatcher. Диспетчер играет роль глобального посредника в шаблоне «Издатель-подписчик» и рассылает полезную нагрузку зарегистрированным обработчикам.

Типичная реализация архитектуры Flux может использовать эту библиотеку вместе с классом EventEmitter из NodeJS, чтобы построить событийно-ориентированную систему, которая поможет управлять состоянием приложения.

Flux содержит следующие компоненты:

- Actions / Действия — хелперы, упрощающие передачу данных Диспетчеру
- Dispatcher / Диспетчер — принимает Действия и рассылает нагрузку зарегистрированным обработчикам.
- Stores / Хранилища — контейнеры для состояния приложения и бизнес-логики в обработчиках, зарегистрированных в Диспетчере.
- Controller Views / Представления — React-компоненты, которые собирают состояние хранилищ и передают его дочерним компонентам через свойства.

Диспетчер — это менеджер всего процесса, центральный узел приложения. Он является классом-синглтоном (классом-одиночкой). Диспетчер получает на вход действия и рассылает эти действия (и связанные с ними данные) зарегистрированным обработчикам. Его отличие от традиционной событийной модели состоит в том, что он рассылает действия всем обработчикам, и уже сами обработчики решают, должны ли они реагировать на определенное действие.

Одним из преимуществ Диспетчера является возможность описать зависимости и управлять порядком выполнения обработчиков в Хранилищах.

Хранилища в Flux управляют состоянием определенных частей предметной области вашего приложения. На более высоком уровне это означает, что Хранилища хранят данные, методы получения этих данных и зарегистрированные в Диспетчере обработчики Действий.

Хранилища уже могут работать по событийной модели, это позволяет им слушать и рассылать события, что, в свою очередь, позволяет компонентам представления обновляться, отталкиваясь от этих событий, то есть узнавать о том, что состояние приложения изменилось, и пора получить (и отобразить) актуальное состояние.

Фабрика Действий — это набор методов, которые вызываются из Представлений (или из любых других мест), чтобы отправить Действия Диспетчеру. Действия и являются той полезной нагрузкой, которую Диспетчер рассылает подписчикам.

В реализации Facebook Действия различаются по типу — константе, которая посылается вместе с данными действия. В зависимости от типа, Действия могут быть соответствующим образом обработаны в зарегистрированных обработчиках, при этом данные из этих Действий используются как аргументы внутренних методов.

Представления — это собственно React-компоненты, которые подписаны на изменения Хранилищ и получают из них состояние приложения. Далее они передают эти данные дочерним компонентам через свои свойства.

Таким образом, схема работы приложения Flux крайне прозрачна, и его состояние намного проще отследить и запрограммировать. Помимо этого, архитектура Flux легко расширяется, что обеспечивает такое большое количество существующих ее модификаций.

Сравнение основных модификаций Flux-архитектуры

Flux стремительно развивается с самого момента своего зарождения, и сегодня существует более 30 различных его модификаций, используемых в проектах различного уровня. Наиболее популярные сегодня решения — это Reflux, Alt и Redux. По своему существу они являются не только архитектурными подходами, но и конкретными фреймворками, помогающими данные подходы реализовать, поэтому их оценка должна базироваться не только на идеологии, которую они в себе несут, но и программных средствах, предоставляемых их сторонниками.

Согласно обсуждению разработчиков на официальном форуме React, наиболее важными для них критериями при выборе того или иного программного фреймворка являются следующие его показатели:

1. Величина сообщества — как много сторонников и потребителей того или иного программного средства? Проект должен продолжать развиваться, даже если его основатель покинет его.

2. Простота: насколько легко понять функционал библиотеки и начать производить код, не тратя много времени на чтение документации? Простой код намного проще поддерживается, а значит на разработку расходуется меньше ресурсов.

3. Функциональность: как много возможностей предоставляет библиотека?

4. Качество документации: даже если фреймворк будет идеальный и обладать широчайшим функционалом, им будет невозможно воспользоваться без наличия у него хорошей документации.

Таким образом, необходимо оценить каждую из основных Flux-реализаций по вышеперечисленным критериям.

Reflux — это одна из старейших Flux-имплементаций, зародившаяся в июле 2014 года. Ее возраст является и минусом — на ее ошибках выросли новейшиие Flux-реализации [10].

По сути, она является улучшенной версией Flux, но более динамичной и подходящей для функционального реактивного программирования, обладая следующими основными отличиями:

1. В Reflux нет Диспетчера. Его роль берут на себя Действия, каждое из которых является своим собственным диспетчером. Это дает возможность Хранилищам в логике подписок на события не писать проверки на то, какие Действия в настоящий момент отрабатывают, что уменьшает объем кодовой базы.

2. Хранилища могут подписываться на другие Хранилища. Таким образом, появляется возможность создавать хранилища, которые агрегируют и обрабатывают данные в стиле MapReduce.

3. Действия могут обрабатываться параллельно.

4. Для обработки действий, зависящих от других действий, они сливаются в единую сущность.

5. Специальные Фабрики Действий не нужны, потому что Действия являются функциями, передающими нужные данные всем подписчикам.

У Reflux огромное сообщество: являясь древнейшей Flux-реализацией, он собрал вокруг себя много разработчиков. Сейчас у него на популярном сообществе программистов github.com более 4000 поклонников, что делает его одной из популярнейших библиотек в принципе.

Reflux по своей идее значительно проще Flux, слив воедино Диспетчер и Действия, то есть устранив одну сущностью, получив при этом преимущество. Это делает код более простым и читаемым. Хотя при этом, Reflux добавляет две новые концепции для подписки на события.

Хотя Действия Reflux могут казаться проще, чем Фабрики Действий, они дают намного больше возможностей. Он позволяет создавать более сложные цепочки действий и зависимостей между ними. Но такая возможность дается не даром — код становится больше похожим на асинхронный, а также сложнее логгировать различные Действия.

Благодаря развитому сообществу, Reflux без труда имеет качественную документацию, доступную на нескольких языках, включая различные обучающие материалы на русском языке.

Alt, так же, как и Reflux, делает акцент на минималистичности, но, в отличие от второго, он старается не отходить от традиционной архитектуры. По сути, он является скорее модификацией библиотеки Flux, нежели архитектуры [11].

Он имеет меньшее сообщество, в отличие от Reflux, но оно в разы активнее, и гораздо быстрее развивается: об этом можно судить по количеству присылаемых в библиотеку патчей — их практически в два раза больше.

Alt крайне прост в освоении. Он поддерживает ES2015 — последний стандарт JavaScript от ECMA и декораторы ES7, которые значительно упрощают работу с кодовой базой. Аналогично Reflux, Alt сливает понятие Фабрик Действий и Действий в единую сущность — Действия-Функции.

Alt предоставляет все преимущества Flux, наделяя функционал более приятным синтаксисом.

Документация Alt является превосходной, в ней описаны не только весь его функционал, но и концепции Flux, а также идеи, стоящие за ними.

Разработка Redux началась в середине 2015 года. Изначально Redux не задумывался как серьезный проект, но сейчас является популярнее самого Flux [12].

Redux намного проще остальных фреймворков засчет того, что структуры в Redux — иммутабельны, то есть неизменяемы. Это делает Redux более похожим на функциональный фреймворк и позволяет легче управлять потоками, накладывая изменения на данные вместо их фактического изменения.

Redux имеет три основных принципа в своей основе:

1. Состояние всего приложения хранится в едином Хранилище.

2. Состояние приложения доступно только для чтения, то есть неизменяемо. Для того, чтобы получить измененное состояние, нужно взять исходное и наложить на него мутации.

3. Изменения осуществляются чистыми функциями, называемыми Редукторами, то есть такими, результат работы которых зависит только от переданных в них аргументов.

Таким образом, на основе рассмотренных параметров, можно обобщить приведенные суждения в виде оценок, которые приведены в таблице 1.

Таблица 1. Сравнительные оценки основных модификаций Flux-архитектуры

	Reflux	**Alt**	**Redux**
Величина сообщества	5	4	5
Простота	4	5	5
Функциональность	4	3	5
Качество документации	5	5	5

Из данной таблицы преимущество Redux становится совершенным очевидным. Такие выводы подтверждают и факты — в настоящее время количество проектов, которые используют Redux значительно превосходит остальные. Об этом можно судить по статистике скачиваний данной библиотеки с www.npmjs.org — наиболее популярного менеджера пакетов для языка JavaScript.

Таким образом, после проведенного исследования, можно оценить экономическую эффективность использования того или иного подхода с учетом того, что имеется опыт разработки с рассматриваемой библиотекой. Для этого необходимо рассмотреть процесс создания базового веб-приложения с использованием каждого из приведенных фреймворков.

В веб-разработке традиционным приложением, на создании которого оценивают тот или иной фреймворк, является базовый менеджер задач. Он выглядит всегда одинаково для всех реализаций, которые лишь определяют его «двигатель».

Данные приложения были реализованы с использованием каждого из программных фреймворков. Основная статистика результата их реализации приведена в таблице 2.

Таблица 2. Статистика результатов реализации базового приложения с использованием основных Flux-модификаций

	Reflux	Alt	Redux
Количество файлов, шт	16	3	24
Количество строк кода, шт	486	25	551

Размер библиотеки, KB	18	9	37

Кодовая база, затрачиваемая на разработку у всех фреймворков сопоставима. При этом количество файлов является важным показателем, потому что говорит о том, сколько много кода разработчик пишет самостоятельно, а сколько — в рамках фреймворка. Если файлов много, значит проект разложен по архитектуре и трудозатраты на каждый файл невелики. В противном случае разработчик самостоятельно реализовал большие объемы логики в больших файлах.

Компания StackOverFlow, являющаяся крупнейшей платформой вопросов и ответов для программистов, провела исследования касательного того, как много строк кода производит профессиональный разработчик за рабочий день. Для начальной стадии проекта этот показатель составляет 200 - 500 строк в зависимости от того, насколько хорошо программист знаком с используемыми инструментами. Таким образом, можно сказать, что на основе каждого из рассмотренных фреймворков профессиональный программист способен создать базовое приложение за 1 - 2 дня, что является хорошим показателем. При среднем заработке разработчика, равному 500 рублей в час (согласно статистике рекрутингового портала www.hh.ru), можно сказать, что его стоимость будет равна 4 - 8 тысячам рублей. При этом Alt является аутсайдером по данному критерию, а Reflux и Redux практически идентичны. Но с учетом того, что в проекте, реализованном средствами Redux, больше файлов, это говорит о том, что он лучше масштабируется, потому что он лучше поддерживается, так как структура проекта становится нагляднее, а каждый отдельно взятый файл проще для понимания [13].

Таким образом, Redux является оптимальным решением для создания одностраничного приложения на основе Flux-архитектуры.

Заключение

Неустанно развиваясь, веб-браузеры постепенно вытесняют настольные приложения, которые тяжелее распространять и поддерживать. Веб уже сегодня является достойной им альтернативой, а современные сайты имеют возможность эксплуатации практически всех ресурсов компьютера пользователя.

При этом не остается незамеченным очевидное усложнение веб-приложений. На клиентскую часть приходится все больше бизнес-логики, которую невозможно поддерживать без четких архитектурных паттернов. Такие паттерны дает использование популярных фреймворков, исповедующих какую-либо концепцию.

Основными концепциями разработки веб-приложения сегодня является MVC и Flux. Последняя из них является справедливым современным трендом в мире фронтенд-разработки и ее популярность сегодня растет буквально с каждым днем.

Flux имеет множество модификаций, наиболее популярными из которых сегодня являются Reflux, Alt и Redux. На основе проделанного исследования можно сделать следующие выводы:

1. MVC является удобной концепцией разработки веб-приложений, но Flux по достоинству перенимает звание наиболее популярной архитектуры в мире веб-разработки.
2. Оценка программных фреймворков разработки веб-приложений является нетривиальной задачей, и в различных ситуациях могут справедливо подойти различные решения.
3. Redux в подавляющем большинстве случаев является наиболее оптимальным решением для проектов различного уровня, являясь более качественным и производительным решением для разработки программного обеспечения.

Литература

1. Кибернетическая революция и шестой технологический уклад — Историческая психология и социология истории. Том 8, номер 1 / 2015. стр. 172 - 197.
2. Характеристики и описание ноутбука Chromebook — https://www.google.com/chromebook/about/.
3. Миковски Майкл, Пауэлл Джош. Разработка одностраничных веб-приложений = Single Page Web Applications: JavaScript End-to-end. — ДМК Пресс, 2014. 512 с. ISBN 978-5-457-83457-6.
4. *Чедвик Джесс.* ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC = Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC. М.: «Вильямс», 2013. 432 с. ISBN 978-5-8459-1841-3.
5. Официальный сайт Flux — https://facebook.github.io/flux/.
6. *Фримен Адам.* ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов, 4-е издание = Pro ASP.NET MVC 4, 4th edition. М.: «Вильямс», 2013. — 688 с. — ISBN 978-5-8459-1867-3.
7. *Олищук Андрей Владимирович.* Разработка Web-приложений на PHP 5. Профессиональная работа. М.: «Вильямс», 2006. С. 352. ISBN 5-8459-0944-9.

8. *Беллиньясо Марко.* Разработка Web-приложений в среде ASP.NET 2.0: задача — проект — решение = ASP.NET 2.0 Website Programming: Problem - Design - Solution. М.: «Диалектика», 2007. С. 640. ISBN 0-7645-8464-2.
9. Официальный сайт Reflux — https://github.com/reflux/refluxjs.
10. Официальный сайт Alt.js — http://alt.js.org/.
11. Официальный сайт Redux — http://redux.js.org/.
12. Официальный сайт StackOverFlow — http://stackoverflow.com/.