МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий

(наименование института полностью)

Кафедра «Прикладная математика и информатика»

(наименование кафедры)

09.04.03 Прикладная информатика

(код и наименование направления подготовки, специальности)

Информационные системы и технологии корпоративного управления

(направленность (профиль)/специализация

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Технологии создания корпоративных веб-приложений»

Студент Д.В. Клочков _____

(И.О. Фамилия) (личная подпись)

Научный руководитель О.М. Гущина _____

(И.О. Фамилия) (личная подпись)

Руководитель программы д.т.н., доцент, С.В. Мкртычев _____

(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____20_____г.

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____

«_____» _____20_____г. (личная подпись)

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире корпоративные приложения уже перестали быть прерогативой больших организаций или фирм. Хотя некоторые специалисты будут утверждать, что корпоративное приложение должны поддерживать одновременную работу большого количества пользователей, иметь высокую загрузку данных, включать большие объемы базы данных, быть масштабируемым или иметь бизнес и персистентный слой, как большинство java приложений. Это все верно, но сегодня парадигма корпоративных приложений сместилась.

Использование Интернета как платформы для создания программных приложений, как средства коммуникации, развлечений, образования и бизнеса резко изменил ландшафт разработки программного обеспечения. Обычные приложения разрабатывались, следуя четкому жизненному циклу, в котором обслуживание и эволюция измерялись годами, конечные пользователи были хорошо известны, а проблемы интерфейса, эргономики и взаимодействия не были решающим фактором. Но в веб-приложениях появился новый набор проблем. Требования к веб-приложениям часто меняются очень быстрыми темпами, приходится иметь дело с тысячами неизвестных пользователей, которые получают доступ к миллионам информационных элементов и которые ожидают персонализированного контента и функциональности. Эти пользователи редко лояльны, и они ожидают простых и эффективных в использовании приложений. Что еще хуже, новое поколение пользовательских устройств, т. е. смартфонов и планшетов, еще больше создало проблем. Как следствие, команды разработчиков веб-приложений находятся под постоянным давлением, так как им необходимо в короткие сроки предоставлять рабочие приложения с часто нестабильными требованиями. Кроме того приложения должны работать с большим количеством разнообразных устройств. Разработка таких приложений, отличается от разработки обычных приложений. Веб-приложения стали применятся повсеместно. Это случилось благодаря огромному росту различных стартапов и электронной коммерции. Для

организаций, стратапов или небольших команд, веб-приложения обычно включают в себя рабочие процессы, специфичные для их компании или фирмы, и обычно они интегрированы с рядом внутренних систем, источников данных и процессов. Такие приложения можно назвать корпоративные веб-приложения.

Согласно различным источникам, а также описаниям процессов разработки при разработке веб-приложений, разработчики используют классические методы разработки, такие как водопадная модель, итеративные модели или более современные «гибкие» методы разработки. А иногда не используют вообще никакую модель. В настоящее время очень глубоко проработаны все этапы разработки информационных систем, сбор требований, проектирование, реализация, тестирование, внедрение, поддержка. Существуют уже устоявшиеся методики разработки веб-приложений. В основном прорабатываются вопросы, связанные с решением таких проблем, как навигация, дизайн, функциональность, ориентированная на пользователя, созданием модели данных необходимых для отображения и не уделяется такой важной детали как управлением состоянием приложения. Управлением состоянием приложения это то, что нужно планировать заранее, это не то, что нужно добавить в конце проекта, планирование этой техники имеет решающее значение для успеха в создании приложений.

Из вышесказанного следует, что проблема управлением состоянием приложения является очень важной, и необходимо такой проблеме уделять внимание. Поэтому изучение данного вопроса считаю актуальной темой для написания этой работы.

Целью работы будет являться разработка технологии, которая позволит на этапе проектирования добавить технику управления состоянием веб-приложения.

Для достижения поставленной цели, предполагается решить ряд следующих **задач**:

- анализ научно-технической литературы в области разработки веб-приложений;

- исследование и анализ веб-приложений как объектов проектирования, определение основных характеристик веб-приложения;

- разработка технологии, которая позволит внедрить в разработку веб-приложения управление состоянием приложения;

- экспериментальная проверка технологии и формулирование рекомендаций по ее использованию.

Объектом данной работы являются модели и методы разработки веб-приложений.

Предметом данной работы является применение техники управления состоянием приложения.

Теоретической основой и основными источниками информации для написания данной работы служили, публикации и работы различных авторов, специалистов, экспертов и ученых, как отечественных, так и зарубежных. В том числе использовались различные тематические электронные ресурсы сети интернет.

Для того чтобы разобраться, что такое корпоративное веб-приложение использовались книги Е.Г. Сысолетин [16], А.Ф. Тузовский [54], работа зарубежного автора P.ZELENKA [34].

Для изучения трудностей и проблем внедрения современных информационных технологий в виде информационных систем по управлению персоналом на предприятии, использовались такие труды, как В.П. Баранчеев [19], К.Я. Бакис [15].

Методологической основой исследования и анализа послужили научные и методические материалы, которые являются основой в таких областях как программная инженерия, маркетинговые исследования, проектирование систем управления, разработка и проектирование информационных систем на предприятии. В процессе исследования применялись общенаучные методы: сравнение, методы экспертных оценок, классификация, анализ эффективности инженерные решения и др. Данные методы позволили обеспечить наглядность, достоверность и обоснованность выводов и практических рекомендаций.

Эмпирическую базу исследования составили электронные ресурсы (веб – сайты) блогов производителей и разработчиков обычных информационных систем и программного обеспечения так и веб-приложений. Также были проанализированы данные периодических изданий и электронных журналов, статей и презентаций. Электронные площадки разработчиков. Основной базой для получения информации по методике проектирования современных веб-приложений являлись книги Е.Г. Сысолетин [16], А.Ф. Тузовский [54], и научного журнал Web Engineering [40].

Работа включает в себя четыре главы. В первой главе веб-приложение рассматривается как объект проектирования. Разбираются классические модели и методологии проектирования с целью изучения как с помощью анализируемых методик решается проблема управления состоянием приложения. Вторая глава данной работы составляет описание какие характеристики присущи современному веб-приложению и отдельно идет описание того, что такое управление состоянием приложения и как оно призвано помощь в разработке. В третьей главе представляется разработанная для этой диссертации технология «Визуальное проектирование хранилища параметра сеансов». В четвертой главе этой работы, уже проводится эксперимент с целью выявить эффективность применяемой технологии, какие веские преимущества дает сама технология и использование управление состоянием приложения. В заключении работы делается общий вывод по всей работе.

Диссертационная работа состоит из введения, 4 глав, заключения, списка использованных источников из 43 наименования. Текст работы изложен на 72 страницах, содержит 38 рисунков, 1 таблицу.

Глава 1 ВЕБ-ПРИЛОЖЕНИЕ КАК ОБЪЕКТ ПРОЕКТИРОВАНИЯ

1.1 Понятие корпоративного веб-приложения

Веб-приложения – это приложения, развернутые в среде Интернет. Такие приложения используют единую клиент-серверную модель. Веб-приложения запускаются в веб-браузере на компьютере пользователя (клиента). Одной из самых отличительных характеристик веб-приложений является немедленное развертывание среди всех пользователей. Это достигается путем размещения новой версии веб-приложения на веб-сервере (сервере). После этого новая версия сразу доступна всем пользователям (клиентам). На сегодня существуют различные вариации определения веб-приложений. Их также могут называть и интернет-приложения, и веб-сайты. Термины интернет-приложения и веб-приложения не являются синонимами. Интернет-приложение не обязано базироваться именно на веб-службе и использовать для межкомпонентного обмена протокол HTTP. Однако, поскольку большую часть интернет-приложений составляют именно такие приложения, разница между терминами практически сходит на нет [16]. Также термин веб-приложение описано в учебнике Проектирование и разработка web-приложений. Под web-приложением понимается прикладная программа, разработанная по архитектуре «клиент-сервер», использующая в качестве клиента web-браузер и работающая на стороне web-сервера. При этом взаимодействие между клиентом и сервером выполняется с использованием протокола HTTP [17]

Поскольку основным преимуществом веб-приложений является их доступность широкой аудитории. Веб-приложения стали применятся повсеместно. Это случилось благодаря огромному росту различных стартапов и электронной коммерции. Для организаций, стратапов или небольших команд, веб-приложения обычно включают в себя рабочие процессы, специфичные для их компании или фирмы, и обычно они интегрированы с рядом внутренних систем, источников данных или процессов. Такие приложения можно назвать корпоративные веб-приложения. Именно такие приложения делают бизнес более

доступным, адаптивным, многофункциональным, современным. А поскольку такие веб-приложения являются доступными широкой аудитории, то к ним предъявляются более высокие требования в плане дизайна и эргономики пользовательского интерфейса.

В связи с этим для создания таких веб-приложений необходимы инструменты, методики, набор информационных технологий, технические средства.

Рассмотрим какие сложности и вызовы существуют при разработке веб-приложений.

1.2 Подходы, используемые при разработке веб-приложений

Веб-приложения - в основном, это динамичные приложения, они постоянно меняются и развиваются. Это все результат развития Интернета и рыночной среды, частью которой являются эти приложения. По мере развития или совершенствования веб-приложений они также могут изменять категорию, к которой они принадлежат. Сама эволюция веб-приложений — это не только изменение категории веб приложения, это вопрос восприятия и мотивации веб-разработки, а также общего изменения в использовании веб-технологий.

Сама по себе веб-разработка, требует специальных навыков. К таким навыкам относятся знания в области взаимодействия человека и компьютера, а также навыки в проектировании информационной архитектуры.

Мотивация, сильно зависит от первоначальной цели применения и использования веб-приложения, ожиданий клиентов, конкурентной среды и развитию самого Интернета.

Изменение Интернета, изменяет и технологии, которые используются в этой среде. Что в свою очередь отражается и на разрабатываемых веб-приложениях.

Такое положение вещей, привело к тому, что существующие методики разработки настольных приложений не подходят для разработки веб-приложений. Основное отличие, это высокая скорость развертывания и

моментальная доступность. Такая «гонка за рынком», ставит высокие требования к разработке веб-приложений, постепенно сокращая время, отведенное на разработку. Еще одной важной особенностью, является огромный рынок, ведь каждый веб-браузер, установленный на персональном компьютере это потенциальный пользователь создаваемого веб-приложения. Также влияют как было сказано выше, развитие технологий, вспомогательных инструментов, программных библиотек и каркасов. Появилось большое разнообразие различных инструментов. Но все же в этом большом наборе можно выделить так называемую архитектуру или модели современных веб-приложений.

Так, например, в настоящее время разработка веб-приложений может быть сгруппирована в два вида разработки. Первое это традиционное веб-приложение или как их еще называют (многостраничные приложения), а второе это так называемые SPA (одностраничное приложение). Основное отличие состоит в том, что, когда пользователь переходит или как говорят осуществляет навигацию на другой пункт меню, страницу, браузеру требуется время для получения нового HTML-документа с сервера. Внутренняя обработка сервера также может занять некоторое время. В наше время, устройства конечного потребителя будь то компьютер, планшет или мобильный телефон постоянно развиваются. Современные устройства обладают большей мощностью обработки информации и более большим объемом памяти. Благодаря таким улучшенным характеристикам большую долю логики и обработки приложения можно передать конечному устройству потребителя. Это освободит сервер от использования большого количества ресурсов для каждого клиента и большого количества запросов и обращений к серверу. Поэтому отталкиваясь от этой концепции, к ней лучше всего подходит так называемая одностраничная модель приложения (SPA – single page application). В так называемых одностраничных приложениях все содержимое приложения загружается сразу и поэтому начальная загрузка страницы обычно длиннее, но последние изменения страницы происходят мгновенно. Это достигается за счет манипуляций моделью HTML документа или по-другому манипуляцией на DOM деревом.

DOM означает объектную модель документа и является абстракцией структурированного текста. Для веб-разработчиков этот текст является HTML кодом, а DOM просто называется HTML DOM. Элементы HTML становятся узлами в DOM. Поэтому одностраничные приложения динамически получают доступ и обновляют содержимое, структуру и стиль документа. С помощью объектной модели документа программисты могут создавать документы, перемещаться по их структуре и добавлять, изменять или удалять элементы и содержимое. DOM представляет документ как узлы и объекты. Таким образом, языки программирования могут подключаться к странице. DOM – это объектно-ориентированное представление веб-страницы, которое может быть изменено с помощью языка сценариев, такого как JavaScript. Все, что найдено в HTML документе, можно получить, изменить, удалить или добавить с помощью объектной модели документа.

И по мере развития одностраничных приложений, все больше и больше обработки стало сосредоточено на стороне клиента. Поэтому такие языки программирования как JavaScript тоже стали развиваться. Все это породило появление большого количества инструментов, программных каркасов, методик и т.д. Это все увеличивает сложность разработки и понимание того какой результат необходимо достичь.

Проблема как кажется автору данной работы лежит в следующем. Например, если взять и проанализировать существующие так называемые классические методы разработки веб-приложений [11].

1. Каскадная модель Каскадная модель или «водопад». Это одна из самых старых методик разработки программного обеспечения. Эта методика подразумевает прохождение всех этапов. Каждый из которых в свою очередь должен завершиться полностью до начала, следующего этапа (Рисунок 1).

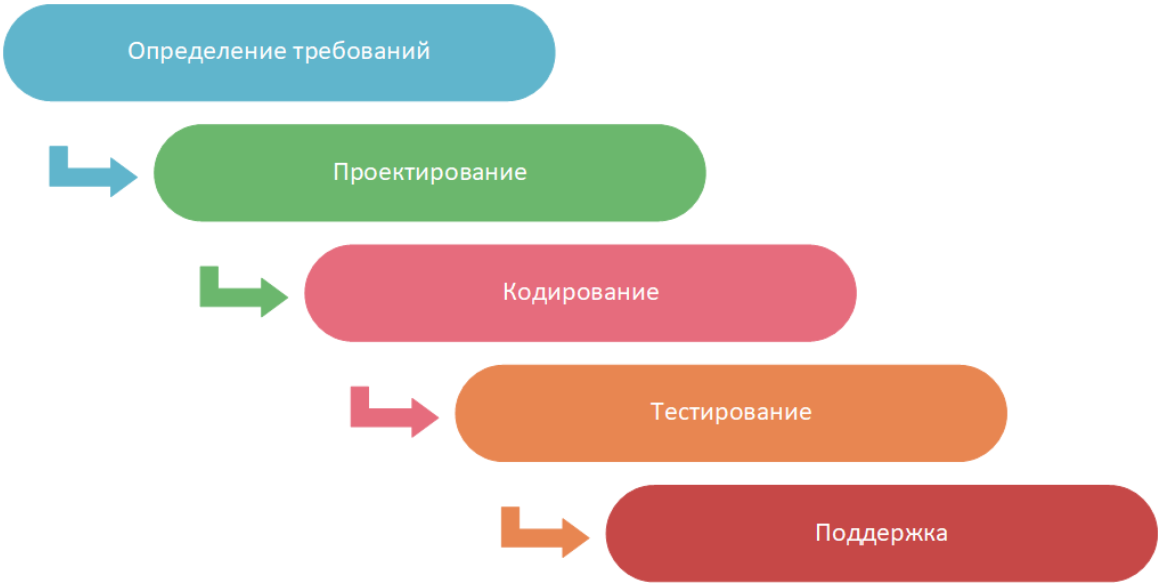

Рисунок 1 - Каскадная модель разработки информационных систем

2. Итеративная модель.

При использовании итеративной модели в начале жизненного цикла нет нужды использовать какие-либо требования, вместо этого создается начальная версия продукта. Которая в свою очередь и является точкой отсчета, т.е служит исходным материалом для определения требований (Рисунок 2).

Рисунок 2 – Итеративная модель

3. Инкрементная модель.

Модель используется для поэтапной сборки программного обеспечения. Она имеет несколько циклов разработки, где они и составляют общий жизненный цикл информационной системы. Сам цикл разделен на мелкие модули. В свою очередь каждый модуль проходит всю цепочку, анализ, проектирование, реализация, тестирование. Сам процесс продолжается пока не будет создана вся система (Рисунок 3).

Рисунок 3 – Инкрементная модель

Если с моделями разработки все понятно, то методологии разработки еще не были рассмотрены здесь. Самыми современными и известными методологиями проектирования являются WebML и WSDM [Тузовский].

WebML – это одновременно метод разработки и язык Web Modeling Language с помощью которого производиться проектирование и разработка;

WSDM – это один из первых методов разработки веб-приложений Web Site Design Method.

WebML – это визуальный язык для определения структуры контента веб-приложения, а также организации и представления такого контента в гипертексте [Ceri et al., 2000, 2002] (Рисунок 4).

Рисунок 4 – Этапы модели разработки WebML

В основе проектирования с помощью этой методологии лежит концептуальное моделирование, которое состоит из определения концептуальных схем. Эти схемы выражают организацию приложения на высоком уровне абстракции, независимо от деталей реализации. Согласно подходу WebML, концептуальное моделирование состоит из проектирования данных и гипертекстового дизайна (проектирование гипертекстов).

Структура данных соответствует организации основных информационных объектов, ранее выявленных в ходе анализа требований, в комплексную и согласованную схему данных, возможно обогащенную за счет производных объектов. Основными элементами модели данных WebML являются сущности, определенные как контейнеры элементов данных, и отношения, определенные как семантические связи между сущностями. Сущности имеют именованные свойства, называемые атрибутами, со связанным типом. Сущности могут быть организованы в иерархии обобщения, а отношения могут быть ограничены с помощью ограничений мощности.

Затем идет проектирование гипертекстов, которые создают гипертекстовый дизайн. Это проектирование схемы представления сайта поверх ранее определенной схемы данных. Гипертекстовая модель позволяет

определить интерфейс пользователя, который отображается пользователю в браузере. Этот этап позволяет определять страницы и их внутреннюю организацию с точки зрения компонентов (называемых единицами контента) для отображения контента. Он также поддерживает определение связей между страницами и блоками контента, которые поддерживают расположение и просмотр информации. Компоненты также могут указывать операции, такие как управление контентом или процедуры ввода-вывода информации пользователя. Представления сайта выражают состав контента и сервисов внутри гипертекстовых страниц, а также навигацию и взаимосвязь компонентов. Для приложений, в которых различные группы пользователей выполняют несколько действий, или для многоканальных приложений, в которых пользователи могут использовать различные устройства доступа, гипертекстовый дизайн требует определения нескольких представлений сайта, обращения к соответствующим группам пользователей и их требованиям доступа.

WSDM – эта методология предоставляет примитивы моделирования, которые позволяют веб-разработчику создавать модели, описывающие веб-сайт или веб-приложение с разных точек зрения и на разных уровнях абстракции, но и обеспечивает систематический способ разработки веб-приложения. Разработка с помощью WSDM начинается с формулировки так называемого заявления о миссии и следует четко определенной философии дизайна, которая предлагает дизайнеру необходимую поддержку для структурирования веб-приложения. Метод состоит из последовательности этапов. Каждая фаза имеет четко определенный выход. Для каждой фазы предоставляется метод, описывающий, как получить что-то на выходе имея что-то на входе. То есть свое рода применяется определенная технология. Полученные результаты на выходе одной фазы будут являться материалам для входа следующей фазы. Также важно отметить, что WSDM позволяет разрабатывать семантически аннотированные веб-системы, что позволяет эффективно использовать семантический веб. Могут быть созданы аннотации, связанные с содержанием (семантические), а также структурные аннотации (Рисунок 5).

Рисунок 5 – Этапы модели разработки WSDM

WSDM следует подходу что главное это аудитория (пользователи) приложения или сайта. Философия дизайна, ориентированная на аудиторию, означает, что различные целевые аудитории (пользователи) и их требования рассматриваются в качестве отправной точки для проектирования и что основная структура веб-приложения является производной от этого. Конкретно, это приводит к различным путям навигации (называемым треками аудитории), предлагаемым с домашней страницы, по одному для каждого отдельного посетителя.

Само проектирование происходит на стадии концептуальное проектирование. Этот этап используется для определения информации, функциональности и структуры веб-системы на концептуальном уровне. Концептуальный дизайн помогает создать абстракцию независящей от любой технологии реализации или целевой платформы. Информация и функциональность задаются во время под фазы задачи и информационного

моделирования. Общая концептуальная структура навигации, включает в себя навигационные возможности для каждого класса аудитории.

На этапе моделирования задач ставиться цель смоделировать различные задачи, которые члены каждого класса аудитории должны иметь возможность выполнять и формально описывать данные и функции, необходимые для этих задач. Задачи, которые должен уметь выполнять член класса аудитории, основаны на требованиях, сформулированных для класса аудитории при классификации аудитории, т. е. для каждого информационного и функционального требования, сформулированного для класса аудитории, определяется задача, которая должна позволить удовлетворить это требование. Результатом этой деятельности является модель задачи.

Далее создается информационная модель. Для этого для каждой элементарной задачи в этой модели создается фрагмент объекта. Основная цель фрагмента объекта, это формально описать информацию и функциональность, необходимые пользователю при выполнении связанной задачи. Если требование, связанное с задачей, является чисто информационным требованием (т. е. пользователь только ищет информацию, ей не нужно выполнять действия), то фрагмент объекта можно рассматривать как концептуальное описание информации, которая будет отображаться на (части) экрана.

После идет этап проектирование навигации в веб-приложении. Целью такого проектирования является определение концептуальной структуры веб-приложения и моделирование того, как члены различных классов аудитории могут перемещаться по веб-приложению и выполнять свои задачи. Результатом этапа навигационного проектирования является навигационная модель.

Далее начинается этап проектирования визуальной составляющей веб-приложения. Цель этого этапа дополнить концептуальный проект необходимыми деталями для реализации. Это нужно для того, чтобы если веб-приложение будет создано непосредственно из концептуального дизайна, можно получить только стандартные и довольно упрощенные презентации. Поэтому необходим графический дизайн. Результатом этого этапа является модель

структуры веб-приложения. Структура веб-приложения графически представлена путем макетов страниц (экранов), которые в последствии должны быть сгруппированы. Потом идет реализация проекта.

Если посмотреть на вышеопределённые методологии, то можно увидеть, что весь цикл разбит на определенные этапы. Между этапами существуют контрольные точки. Контрольные точки — это некая абстрактная модель, у которой есть входящие, вводные данные и исходящие, это результат деятельности. Например, согласно теме работы, нас интересует этап проектирование. На входе должны быть какие-то требования, а на выходе результат проектирования. Этот процесс и будет называться технологией разработки веб-приложений. Потому-что технология это – последовательность материальных процессов и операций, реализация которых приводит к появлению продукта (потребительной стоимости) с необходимыми и полезными для дальнейшего использования человеком свойствами. [13]. Как было выявлено раньше сам процесс разработки веб-приложений отличается от разработки классических приложений. Поскольку сами веб-приложения используют, клиент-серверную архитектуру, то разработку имеет смысл разделить на две разные методики. Разработка клиентской части и разработка серверной части. Такое разделение не ново. Если посмотреть на рынок труда, то существуют вакансии разработчиков веб-приложений как front-end которые отвечают за клиентскую часть и back-end которые отвечают за разработку серверной части. В данной работе будет исследована разработка клиентской части. Поскольку даже во всех исследуемых методиках разработки веб приложений, под веб приложением подразумевается клиентская часть.

1.3 Анализ основных направлений исследований в области построения корпоративных веб-приложений

Самой проблемой разработки занимались многие исследователи. В рамках этой работы были рассмотрены ряд схожей и близкой по тематике работы.

Работа Solomon Antony под названием A REVIEW AND ANALYSIS OF TECHNOLOGIES FOR DEVELOPING WEB APPLICATIONS (обзор и анализ технологий используемых для разработки веб-приложений). В этой работе автор приводит обзор технологий как для серверной разработки, так и для клиентской. Автор предлагает использовать для разработки на клиентской стороне такой набор технологий как html, css, javascript, что по своей сути логично, поскольку клиентская часть выполняется в браузере. Также автор делает акцент на такой понятии как богатство графики в интерфейсе пользователя при создании веб-приложений, чтобы они не уступали по качеству интерфейса настольным. Автор предлагает использовать для этого Flash от Adobe, Silverlight от Microsoft, Active X. Это уже, кажется немного странным и не подходит для создания современных веб-приложений, которые бы работали как на настольных, так и на мобильных платформах. Поскольку такие технологии как Flash, Silverlight, не поддерживаются в мобильных браузерах. Да и на сегодняшний день, эти технологии являются устаревшими. На смену им пришли такие новые технологии как CSS3 для оформления и форматирования HTML элементов. Что позволяет создавать эргономичные интерфейсы, которые не уступают настольным системам.

Статья MODERN METHODS OF WEB APPLICATIONS ANALYSIS AND DESIGN (современные методы проектирования веб-приложений) автор P. ZELENKA. Автор данной публикации поднимает очень важную тему, проектирования веб-приложений. Автор говорит о том, что хоть и прошло очень много времени с того, как появились данный вид приложений, до сих пор не существует устойчивой методологии проектирования и создания веб-приложений. Такой методики или процесса, которая бы позволила использовать ее массово. Она предлагает разделить все веб-приложения на классы, с целью удобной их классификации и идентификации. С последующим их анализом. Она предупреждает, что эта классификация не общепринятая, а это ее субъективная оценка. В своей классификации она приводит три класса, Web presentation – это как правило статические сайты или страницы, с простой навигацией и

взаимодействием которые используют технологию, PHP, ASP, ASP.NET, JSP или PYTHON. Web application - в широком смысле является нетривиальным приложением, которое имеет некоторые аспекты веб-презентации (Web presentation). Прежде всего, это приложение, предназначенное для широкой общественности. Основной пользователь этих приложений не является конкретным человеком или компанией, эти приложения доступны для всех, кто знает правильный url (хостинг адрес). Эти приложения реализуются исключительно как динамические веб-страницы и содержат нетривиальную логику приложения. Типичным примером таких приложений является интернет-магазин, рекламные серверы или дискуссионные форумы и чаты [29]. Http based information systems - информационные систем на основе протокола http. Это как правило системы, разрабатываются для конкретного заказчика. Эти системы отличаются от обычных информационных систем только архитектурой приложения, где клиентская часть опирается на использование http, HTML и браузера.

Эта работа скорее носит попытку начала создания методологи. И носит исключительно ознакомительный характер, а не как руководство или почву для размышлений.

Работа SOFTWARE ENGINEERING PROCESS IN WEB APPLICATION DEVELOPMENT (Программная инженерия в разработке веб-приложений), авторы Manju K Mathai Rakhi Venugopal Dr. John T Abraham. В данной работе предпринята попытка, применить принципы, заложенные в программной инженерии для разработки веб-приложений. Авторы сразу предупреждают, что разработка веб-приложений существенно отличается от разработки обычных систем. Они описывают процесс разработки веб-приложения как процесс поэтапной разработки. Это происходит потому, что требования изменяются с течением времени проекта и эти изменения происходят часто. Поэтапная разработка веб-приложения позволяет управлять этими изменениями. Используя доступные требования, разработчики разрабатывают веб-приложение и выпускают первую версию. Дополнительные требования включены в первый

релиз и дальше происходит переход на следующий шаг и так далее. Такой процесс может пройти неопределенное количество кругов из-за частых изменений в требованиях.

Поэтому разработка веб-приложения носит цикличный характер и включает в себя такие этапы анализ, проектирование, внедрение и обслуживание. В этом они видят проблему, когда разработка идет для крупных приложений. Получается, когда разработка как процесс находится на этапе реализации то любые изменения, внесенные на этапе проектирования, делают реализацию дорогостоящей. В таком случае нужно начинать с самого начала, чтобы учесть новые изменения.

Основные отличия в разработке авторы видят в том, что большое количество пользователей могут получить доступ к веб-приложению одновременно, поэтому может возникнуть необходимость в функции параллелизма. Для защиты конфиденциального контента и обеспечения безопасных способов передачи данных необходимо применять строгие меры безопасности во всей инфраструктуре, поддерживающей веб-приложение, а также в самом приложении [24]. Еще одним отличием они называют, это эргономику интерфейса и его эстетические качества.

Дальше авторы на основе вышеизложенного предлагают, модель разработки веб-приложений, которая бы учитывала все эти отличительные факторы. Такая модель включает в себя определенные этапы и дополнительные тесты. В целом статья носит практичных характер и предложенную модель даже можно опробовать на проекте.

Работа COMPARISON BETWEEN WEB ENGINEERING METHODS TO DEVELOP MULTI WEB APPLICATIONS (сравнение методов разработки в веб-инженерии для разработки мульти веб-приложений), авторы Karzan Wakil и Dayang Norhayati Abang Jawawi. В этой работе в качестве вступления показана история и эволюция развития веб-приложений каким оны были и какие они есть. Авторы описывают, что современные веб-приложения должны быть наделены определенными особенностями:

- богатство графики и анимации, для создания эстетического и эргономичного интерфейса;

- рендер страниц как на клиенте, так и на сервере;

- создание приложений которые могли бы работать офлайн на мобильных устройствах;

- доступность, в любом месте, в любое время, где угодно, на любых устройствах, адаптивные, контекстно – зависимые;

- семантические гиперссылки, продуманность;

- основанные на персонализации, адаптации, семантике с поддержкой интеллектуальных агентов [22].

Это доказывает, насколько высоки требования к современным веб-приложениям. В основной части публикации, авторы приводят методы, которые предложены для разработки современных веб-приложений. Суть этой статьи сравнение существующих методов веб-инжиниринга, используемых для разработки веб-приложений. В этом сравнении они используют функции современного веб-приложения, которые были названы выше. Как итог авторы пришли к выводу, о том, что методы веб-инженерии не могут полностью поддерживать все современные типы веб-приложений, потому что каждый метод предназначен для специальной задачи и другой проблемы, поэтому необходимо определить новый метод для поддержки всех типов веб-приложений.

Работа ENGINEERING WEB APPLICATIONS USING REAL-TIME COLLABORATIVE MODELING (Разработка веб-приложений с использованием совместного моделирования в реальном времени) авторы Peter de Lange, Ralf Klamma, Petru Nicolaescu. В данной работе предлагается новый подход к совместному моделированию и созданию веб-приложений в режиме реального времени на основе web-технологий путем связывания компонентов интерфейса и микрослужб в качестве ключевых элементов. Это приводит к четко определенным сервисным интерфейсам, которые облегчают взаимодействие между службами и интерфейсами. Авторы говорят о повышении

производительности за счет более эффективной поддержки совместной деятельности в веб-инженерии на основе моделей [19].

В работе приводится процесс разработки веб-приложений. Первым этапом предложенного процесса является определение потребности в новом приложении и начало процесса совместного моделирования. Первое решение, которое должно принять команда, это хотят ли они повторно использовать существующие компоненты для создания нового приложения или существует необходимость в разработке или редактировании компонентов интерфейса и / или микрослужб. Если было принято решение не использовать повторно существующие компоненты, начинается моделирование отдельных компонентов. Здесь все члены команды могут привнести свои идеи, совместно моделируя компоненты приложения. После определения микрослужб и компонентов интерфейса начинается этап моделирования гибридного приложения. Если текущее состояние приложения еще не соответствует всем требованиям, приложение и его компоненты должны пройти дополнительные этапы моделирования / редактирования, пока они не будут соответствовать требованиям. Как только требования выполнены приложение может быть развернуто. После этого начинается этап использования приложения, который заканчивается, как только изменяются первоначальные требования приложения. Затем команда должна принять решение о том, будет ли существующее приложение переработано для восстановления этих новых требований. Если это должно быть сделано, то цикл повторяется, иначе цикл заканчивается в этой точке.

Эта статья описывает более взаимодействие между членами команды разработчиков, нежели сам процесс проектирования и разработки веб-приложения.

Публикация COMBINING WEB ENGINEERING METHODS TO COVER LIFECYCLE (Комбинирование методов разработки веб-инженерии который бы покрывал весь жизненный цикл разработки), автор Karzan Wakil. Как понятно из названия, автор комбинирует различные методы разработки веб-приложения для

разных стадий жизненного цикла приложения. В этой статье автор предлагает основу для нового метода веб-инжиниринга путем сочетания трех методов, включающих:

- метод навигационных методов разработки (NDT) для фазы требований;

- веб-инжиниринг на основе UML (UWE) для фазы анализа и проектирования;

-язык моделирования потоков взаимодействия (IFML) для фазы реализации [33].

Авторы говорят, что их новый метод может поддерживать весь жизненный цикл разработки. Более того, этот метод более удобен для разработчиков.

Технический документ NATIVE, WEB OR HYBRID MOBILE-APP DEVELOPMENT (разработка нативных, веб или гибридных мобильных приложений), авторство принадлежит компании IBM.

Как было упомянуто выше, с ростом мобильного трафика в сети Интернет и внедрением мобильных устройств в нашу повседневную жизнь, мобильные приложения играют центральную роль в бизнесе. В данном документе рассматриваются процессы подхода к разработке мобильного приложения, а именно нативного, т.е. разработанного с помощью инструментов разработки которые предоставляет конкретная платформа, веб-приложения с использованием технологий адаптивного дизайна и гибридное приложение, это когда веб-приложение помещают в нативный контейнер мобильной платформы. Цель этого документа заключается не в определении наилучшего подхода к разработке, поскольку такового не существует, а скорее в перечислении плюсов и минусов каждого из них и описании различных сценариев или требований предприятия, которые наилучшим образом соответствуют тем или иным требованиям.

Последний обзор, это учебное пособие ПРОЕКТИРОВАНИЕ ИНТЕРНЕТ-ПРИЛОЖЕНИЙ, авторов Е. Г. Сысолетин и С. Д. Ростунцев. В этом учебном пособии раскрывается тема основ проектирования интернет-приложения. В первой части книги содержится теоретический материал, вводная информация

об интернете и способах взаимодействия с ней. Технологии построения интернет-приложений. Есть раздел, посвященный особенности создания клиентской и серверной части приложения. Отдельно выделены особенности проектирования интернет-приложений. Во второй части содержаться практические задания. Данное учебное пособие содержит вводный материал, который будет полезен начинающим разработчикам, которые хотят научиться разрабатывать веб-приложения. В книге сделан акцент на применение базовых технологий, на которых основано веб-приложение. Книга описывает простые подходы, без каких-либо ухищрений или приемов проектирования, или программирования. Все это позволяет новичку понять, как разрабатывать и проектировать веб-приложения.

Выводу по главе

Из-за стремительного развития информационных технологий, в частности, разработки веб-приложений информации по проблеме проектирования и внедрения управления состоянием в веб-приложениях либо отсутствует, либо упоминается косвенно, как будто этой проблемы не существует или она решается автоматически. Все авторы в основном заняты решением других проблем. Хотя в силу особенностей архитектуры реализации веб-приложений проблема существует, поскольку существуют решения, которые помогают реализовать управление состоянием. Но нигде не упоминается о том, как его спроектировать под конкретный проект.

Глава 2 ВЕБ-ПРИЛОЖЕНИЕ КАК СИСТЕМА

2.1 Свойства и элементы веб-приложения

Само понятие термина веб-приложение описано в учебнике Проектирование и разработка web-приложений автор Тузовский А.Ф. Под web-приложением понимается прикладная программа, разработанная по архитектуре «клиент-сервер» используя в качестве клиента web-браузер и работающая на стороне web-сервера. При этом взаимодействие между клиентом и сервером выполняется с использованием протокола HTTP [15]. Поскольку в данной работе рассматривается такой тип корпоративных веб-приложений, которые интегрируются в бизнес-процессы организации. Например, приложение Microsoft Office Online по своей сути не является корпоративным приложением, а вот уже интеграция этого сервиса в бизнес-процессы компании, например, как поиск по рабочим документам компании, совместное создание документов будет являться как часть корпоративного приложения. В такой комбинации, компания пользуется услугами этого сервиса выполняя свою работу. Также, например, популярный сервис maps.yandex.ru, сам по себе не является корпоративным приложением, а вот его интеграция в бизнес среду компании или фирмы, например, для расчета маршрутов доставки грузов или заказов, геокодирования, будет являться частью корпоративного приложения. Поэтому простое использование в коммерции какого-либо, онлайн сервиса, как например почты или выше упомянутый сервис maps.yandex.ru, не делает его корпоративным приложением, а вот интеграция его в бизнес процессы своей организации, сделает его таковым. Даже онлайн-игра, подключенная к платежной системе, является корпоративным приложением. Поэтому при разработки таких приложений уже предполагается наличие серверной части у которого есть свой application programming interface (API). Соответственно вся разработка таких приложений сосредоточена на клиентской части. Для этого даже придуман термин front-end developer [32]. В России Министерством труда и социальной защиты утвержден профессиональный стандарт «Разработчик Web и мультимедийных приложений». Сам стандарт описывает что основной целю

данного вида деятельности является: создание, модификация и сопровождение web-сайтов, корпоративных порталов организаций, мультимедиа и интерактивных приложений, информационных ресурсов [2]. В трудовые функции которого входит создание интерактивных веб-страниц, их верстка, сбор требований и т.д. Для всего этого используется набор веб-технологий таких как HTML, CSS, JavaScript и все принципы которые используются при разработке клиентской части веб-приложений одинаковы с принципами, которые используются при разработке интернет-сайтов. Такая аналогия не случайна, потому как в Федеральном законе №149-ФЗ «Об информации, информационных технологиях и защите информации», обозначено что, интернет-сайт сам является частью информационной системы. Поэтому и его также можно рассматривать как информационную систему [1]. Также термин веб-приложение нашел сове отражение в ГОСТ 9241-151-2014 «Эргономика взаимодействия человек-система» Веб-приложение это – (англ. Web application, World Wide Web application): Приложение, предоставляющее функциональные возможности пользователю через браузер или другой тип агента пользователя, использующего веб-форматы и протоколы [8]. Поэтому между интернет-сайтом и клиентской частью веб-приложения практически не существует разницы и все т.е принципы, которые существуют для создания интернет-сайтов можно с успехом применять и для создания веб-приложений. Прежде чем начать рассматривать веб-приложение как систему, необходимо в начале выяснить что подразумевается вообще под системой.

СИСТЕМА (греч. systema – целое) – объединение некоторого разнообразия в единое и четко разделенное целое, элементы которого по отношению к целому и другим частям занимают соответствующие им места. Система представляет собой совокупность элементов и связей между ними. [13].

Изучив различные ГОСТы, которые касаются качества и требований, которыми должны обладать программные средства, можно ими также наделить и исследуемые веб-приложения, поскольку они тоже являются программным

обеспечением (средством). Эти качества будут рассматриваться как совокупность свойств системы:

- целостность системы, т.е. веб-приложение состоит из множества, структурных компонентов, которые имеют между собой связи, и в совокупности используются по целевому назначению. Целостность веб-приложения это, состояние программного обеспечения и данных, характеризующееся отсутствием изменений преднамеренного или случайного характера [10];

- функциональность веб-приложения, это когда оно должно иметь определенные функции для взаимодействия с внешней средой, которую могут составлять пользователи, другие информационные системы, сервисы. В документе ГОСТ 28806-90 «Качество программных средств» функциональность это, совокупность свойств программного средства в котором существует такой набор функций, который способен удовлетворять заданные или подразумеваемые потребности [4]. Эта характеристика также включает в себя подпункты, которыми должно обладать программное средство, их также можно применить и к веб-приложениям. Это будут удобство использования программного средства, мобильность программного средства, надежность программного средства;

- структурность и организованность, подразумевает под собой, что веб-приложение имеет структуру, которая функционирует как единое целое. Также веб-приложение имеет определенный набор и расположение графических элементов, которые между собой связаны, что представляет собой информационную архитектуру. По внутреннему содержанию веб-приложение состоит из модулей, структурных элементов, функциональных элементов, что представляет собой архитектуру приложения. Так в ГОСТ Р ИСО/МЭК 12207-2010 дается описание архитектуры программных средств. Архитектура должна определять структуру программного обеспечения и определять, как элементы этой структуры относятся и взаимодействуют с друг другом. Также архитектура гарантирует что все требования распределены между этими составными частями. Составные части конфигурации технических средств, программных

средств и ручных операций должны последовательно идентифицироваться этими составными частями [5];

- наличием выполнения определенных сценариев использования, эта характеристика подразумевает под собой что веб-приложение должно правильно реагировать на внешние намерения, как правило такие сценарии описываются в техническом задании;

- устойчивость функционирования, это способность веб-приложения противостоять внешним негативным действиям без прекращения работы самого приложения. Согласно документу «Оценка качества программных средств» под устойчивостью функционирования подразумевается, способность обеспечивать продолжение работы программы после возникновения отклонений, вызванных сбоями технических средств, ошибками во входных данных и ошибками обслуживания [5];

- адаптируемость, подразумевает под собой наличие у веб-приложения такого поведение или структуры которое должно или может меняться с целью сохранить, улучшить или приобрести новые качества или свойства в условиях изменения внешней среды. Адаптируемость, согласно стандарту «Качество программных средств», – это совокупность свойств программного средства, характеризующая приспособленность для переноса из одной среды функционирования в другие;

- непрерывность – это новое свойство особенно актуально для современных веб-систем. Например, можно начав свою работу на одном устройстве, потом ее продолжить на другом с того места, где пользователь прервался. Таким образом веб-приложение обладает свойствами системы, а сама структура системы уже будет являться архитектурой веб-приложения.

Архитектура вычислительной системы – это структура или структуры системы, включающие программные компоненты, внешние видимые свойства этих компонентов и отношения между ними [16]. Архитектура программного обеспечения определяет структуру и организацию взаимодействующих компонентов программного обеспечения и подсистем, образующих систему [16].

Такую систему можно представить в виде трех компонентов:

1. Элемент системы – это минимальная неделимая часть системы;
2. Отношение между элементами – это то как элементы взаимодействуют между собой и зависят друг от друга;
3. Цель создания такой системы.

Каждый компонент можно рассмотреть более подробно.

Элемент – это минимальная неделима часть системы. В веб-приложении элементом может быть, элемент графического интерфейса пользователя, например, кнопка, программный модуль, программный класс, структура, все это является элементами системы.

Отношения между элементами – с помощью этого можно найти связи между элементами с целью их группировки в структуры. Например, элементы графического интерфейса можно связать в структуру интерфейса пользователя, что будет являться информационной архитектурой интерфейса пользователя. Классы программы также можно связать между собой в структуру, под названием программная архитектура. Такие действия помогают сделать декомпозицию всей системы. Декомпозиция помогает преодолеть общую сложность системы. Декомпозиция делает систему проще для понимания, а в случае разработки веб-приложения, становиться проще управлять разработкой и сопровождением.

Цель создания системы. Под целью создания веб-приложения подразумевается ответ на вопрос «Для чего это делается?». Цель системы можно представить в виде двух стратегий. Первая стратегия относится к экономической стратегии, это когда система создается для экономической выгоды или какого-либо экономического эффекта. Например, применимо к организации, это увеличение прибыли организации. Вторая стратегия это, социальная стратегия, которая должна оказать социальный эффект от создания системы. Например, для Министерства Российской Федерации по делам гражданской обороны, чрезвычайным ситуациям и ликвидации последствий стихийных бедствий (МЧС России), можно создать систему оповещения о надвигающихся природных и

стихийных бедствий в виде веб-приложения или мобильного приложения. Такая система оповещала бы гражданское население о надвигающихся катаклизмах и другой полезной информации. В таком случае от такой системы будет неправильным ожидать большой экономической эффективности если она будет вообще, а вот организационная или социальная эффективность будет достигнута.

Цель создания системы определяет функциональность системы в данном случае веб-приложения. Как было сказано выше функциональность это, совокупность свойств программного средства в котором существует такой набор функций, который способен удовлетворять заданные или подразумеваемые потребности [3]. Поэтому создание системы, нужно всегда начинать с определения цели ее создания.

Дальше необходимо рассмотреть, как веб-приложение взаимодействует с внешней средой (Рисунок 6).

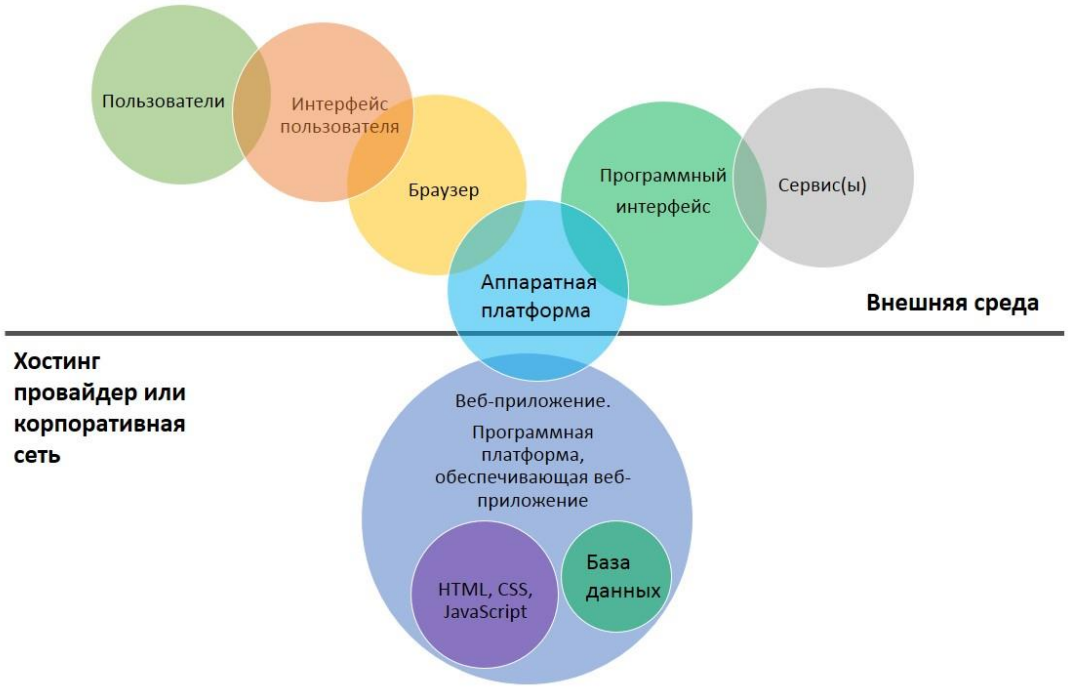

Рисунок 6 – Взаимодействие веб-приложения с внешней средой

Само веб-приложение как правило располагается у хостинг-провайдера либо в собственной корпоративной сети организации. Веб-приложение состоит из программной платформы (framework), которая обеспечивает веб-приложение и которое облегчает разработку веб-приложения. Именно программная

платформа определяет структуру приложения и позволяет объединять разные компоненты в одну большую систему. Поэтому роль программной платформы в жизненном цикле веб-приложения очень велика. Взаимодействие с внешней средой происходит через программную платформу и аппаратную платформу, которая отвечает за передачу данных по сети, обработку и получение запросов из внешней среды, а также за хранение и операциями с данными.

За обмен информацией между сервисами и системой, отвечает программный интерфейс. Он связывает веб-приложение и сервис между собой. Согласно ГОСТ 52872-2012, Интерфейс это – совокупность правил взаимодействия устройств и программ между собой или с пользователем и средств, реализующих это взаимодействие. Понятие интерфейс включает в себя как сами аппаратные и программные средства, связывающие различные устройства или программы между собой или с пользователем, так и правила, и алгоритмы, на основе которых эти средства созданы [6].

Для взаимодействия между системой и пользователями, существует интерфейс пользователя. Интерфейс пользователя это – компоненты интерактивной системы (программное обеспечение и аппаратные средства), которые предоставляют пользователю информацию и инструменты управления для выполнения производственных заданий [6]. Поэтому интерфейс пользователя является своего рода краеугольным камнем, вокруг которого строится веб-приложение. Интерфейс пользователя - это то, с чем в первую очередь сталкивается пользователь. Есть такое понятие пригодность использования (англ. Usability). Это термин нашел свое отражения во множества стандартов. В одном из них, ГОСТ 9241-210 – 2012. Свойство системы, продукции или услуги, при наличии которого установленный пользователь может применить продукцию в определенных условиях использования для достижения установленных целей с необходимой результативностью, эффективностью и удовлетворенностью. Получается, что интерфейс пользователя очень важная составляющая веб-приложения, а такое качество, как пригодность использования очень важно для всей системы в целом.

Также необходимо отметить, что помимо технической составляющей системы, существует и эстетическая составляющая. Сюда относятся те принципы, которые заложены при создании веб-сайта. Это шрифты, цветовая палитра, формы, эстетика, наборы значков и указателей, звуковое сопровождение, анимация.

На основе проведенного исследования и последующего анализа, можно представить разработанную модель современного корпоративного веб-приложения в виде системы. Описание модели приведено в таблице 1.

Таблица 1 – Модель корпоративного веб-приложения как системы

Элемент системы	Описание
Цель создания	
Экономическая	Получение большей рыночной доли, усовершенствование продукта или услуги, вывод на рынок нового продукта или услуги, оптимизация бизнес-процессов, реклама, информирование потребителей, получение обратной связи от потребителей.
Социальная	Объединение и группировка пользователей по интересам, обмен информацией между участниками проекта, получение различного вида информации от различных институтов, помощь пользователям.
Функции	
Функциональность или пригодность использования	Обеспечение таким набором функций, которые позволяют достигать, поставленных целей, с необходимой результативностью.

Элемент системы	Описание
Архитектура	
Слой взаимодействия с внешней средой	Сюда относятся: - программный интерфейс; - графический интерфейс пользователя.
Программный слой	Сюда относятся: - программная платформа; - HTML, CSS, JavaScript код; - набор ресурсов (шрифты, значки, указатели, звуки, файлы настроек); - базы данных.
Аппаратный слой	Хостинг сервер, аппаратная платформа, телекоммуникационная сеть, сервер базы данных.

На основе вышеизложенного материала и исследований, можно графически отобразить архитектурную модель веб-приложения (Рисунок 7).

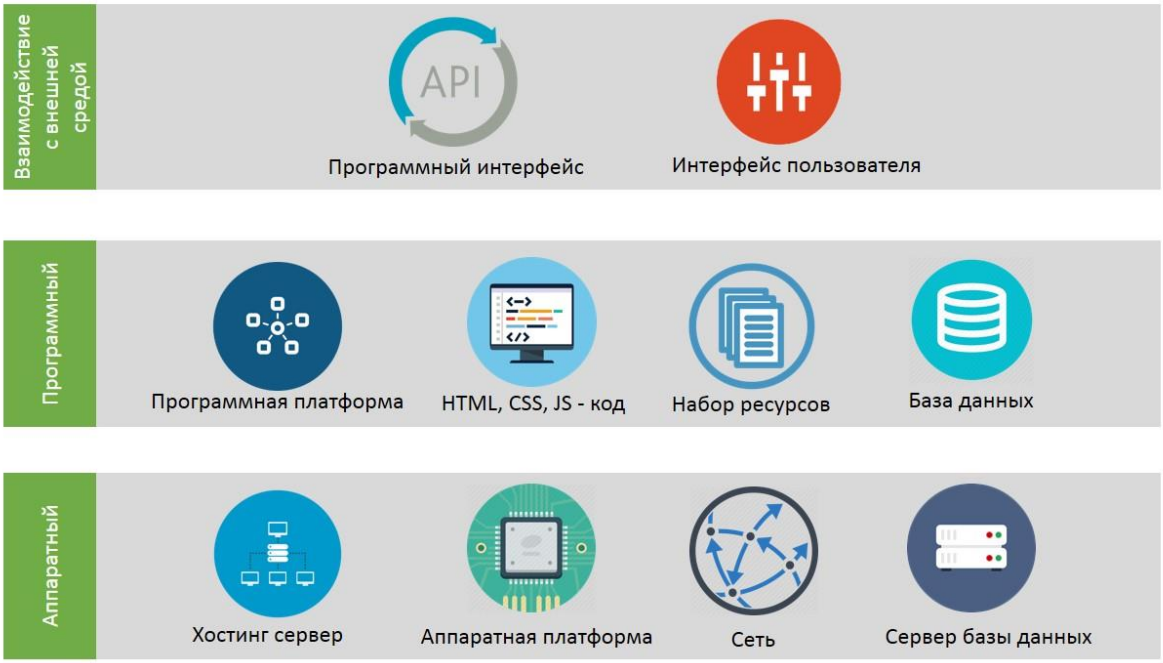

Рисунок 7 – Архитектурная модель веб-приложения

Таким образом, основными элементами системы корпоративного веб-приложения являются цели, функции, архитектура.

2.2 Управление состоянием веб-приложения

Ознакомившись в первой главе с классическими и современными методиками разработки веб-приложений таких как WebML и WSDM, можно заметить, что она сильно отличается от классических Win32 приложений. Как одной из наиболее важных задач современных корпоративных веб-приложений в разработке является сохранение состояния в приложении. Прежде чем сохранить состояние в приложении, необходимо вообще понять, что такое состояние. Состояние это, то, что приложение знает о пользователе, их текущем взаимодействии с приложением и других частях глобальной информации. Какая информация имеется в виду? Это сведения, о том, кто такой пользователь, где он находится в приложении, какую информацию он ввел, и другие сведения о конфигурации приложения. Обычно первое что приходит на ум при разработке веб-приложений это хранить такую информацию о текущем пользователе, его предпочтениях или текущей конфигурации приложения глобально в переменных или как свойства объекта. Эти глобальные переменные остаются в памяти приложения до тех пор, пока приложение не будет завершено. Но, к сожалению, веб-приложения – это совсем другая история.

Веб-приложения не имеют состояния. Пользователь просто получает запрос на отображение веб-страницы, и веб-сервер обязуется отправить ответ в браузер пользователя для отображения. На деле кажется, все достаточно просто. Но тут важно понимать, что на этом этапе общение заканчивается, пока пользователь не запросит другую страницу или информацию. Когда следующий запрос пользователя поступает на веб-сервер, он понятия не имеет, кто этот пользователь. Веб-сервер не знает, какие страницы он отправлял раньше этому пользователю или отправлял ли они их вообще. После того, как веб-сервер обрабатывает каждый запрос пользователя, он очищает свой кэш и готовится к следующему входящему запросу, от этого пользователя или от другого. Вопрос

в том, если веб-сервер постоянно стирает свою память после каждого запроса, как он может знать, кто этот текущий пользователь и как сервер общался с этим пользователем в прошлом. Этот ответ на этот вопрос включает в себя использование методов хранения информации, чтобы помочь ему вновь вспомнить пользователя.

Поскольку требования к одностраничным приложениям JavaScript становятся все более сложными, код веб-приложения должен управлять большим состоянием всего приложения, чем это было раньше. Это состояние может включать ответы сервера и кэшированные данные, а также локально созданные данные, которые еще не были сохранены на сервере. Состояние пользовательского интерфейса также с каждым годом все усложняется, поскольку нужно управлять маршрутизацией и навигацией, выбранными вкладками, элементами управления разбиением на страницы и т. д. Сама по себе проблема управление состоянием приложения возникла не на пустом месте. Все современные веб приложения сегодня используют такой шаблон проектирования как MVC Model View Controller (Рисунок 8)

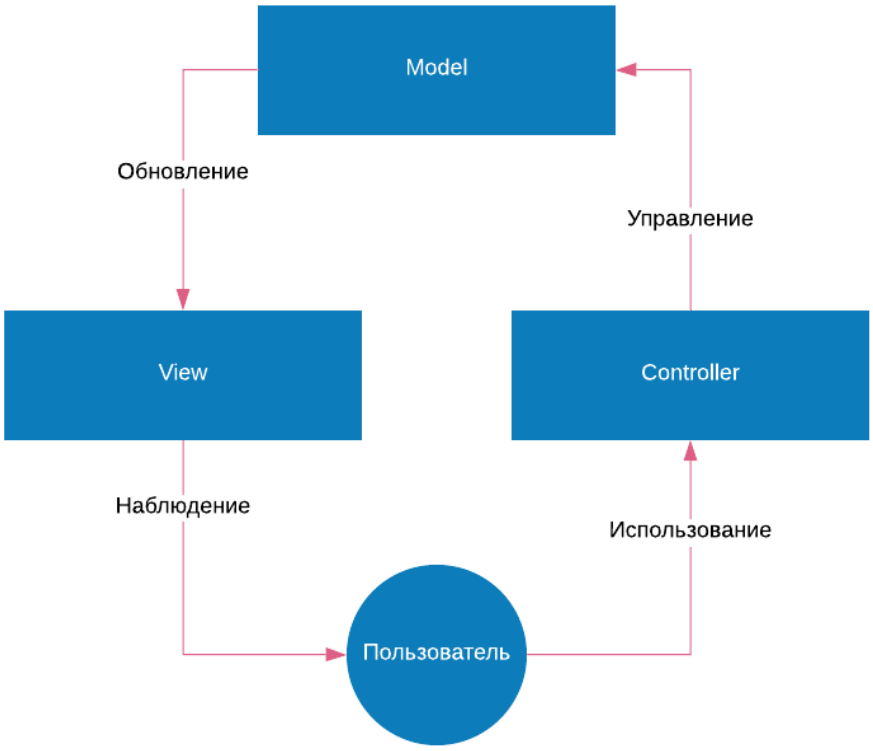

Рисунок 8 – Шаблон проектирования веб-приложений MVC

Шаблон Model-View-Controller (MVC) – это архитектурный шаблон, в котором модель данных, вид (интерфейс пользователя) и функциональный контроллер разделены между собой. Модель данных представляет из себя все данные приложения, вид указывает на представление информации, которое видит пользователь у себя в браузере, а контроллер обрабатывает обработку данных и перемещает ее между моделью и представлением. Пользователь создает различные действия и события, контроллер управляет моделью, и таким образом также обновляется. Модель MVC определяет стандартный способ разбиения приложения на части, и поэтому понимание функциональности приложения становится проще в отношении используемой технологии. С помощью этого шаблона можно догадаться что должна быть система, которая должна отслеживать состояния компонентов и в случае их изменения как-то реагировать на это.

В современных веб-приложениях управление состоянием между множеством компонентов может стать сложной задачей. Идеальным способом было бы, чтобы данные в компоненте находились только в одном месте или компоненте. Размышляя об обмене данными между смежными компонентами: состояние приложения должно было бы находиться в родительском компоненте, который также может обновить это состояние. В свою очередь родительский компонент передает состояние вниз по дереву в качестве реквизита (Рисунок 9).

На рисунке оранжевым цветом показано, как параметры передаются вниз по дереву компонентов. С самого корня до нужного компонента вниз по дереву. В начале это не вызывает проблем и затруднений. Проблема начинается после того, когда состояние должно быть разделено между компонентами, которые находятся далеко друг от друга в дереве компонентов, состояние должно передаваться от одного компонента к другому, пока оно не достигнет места, где оно необходимо (Рисунок 10).

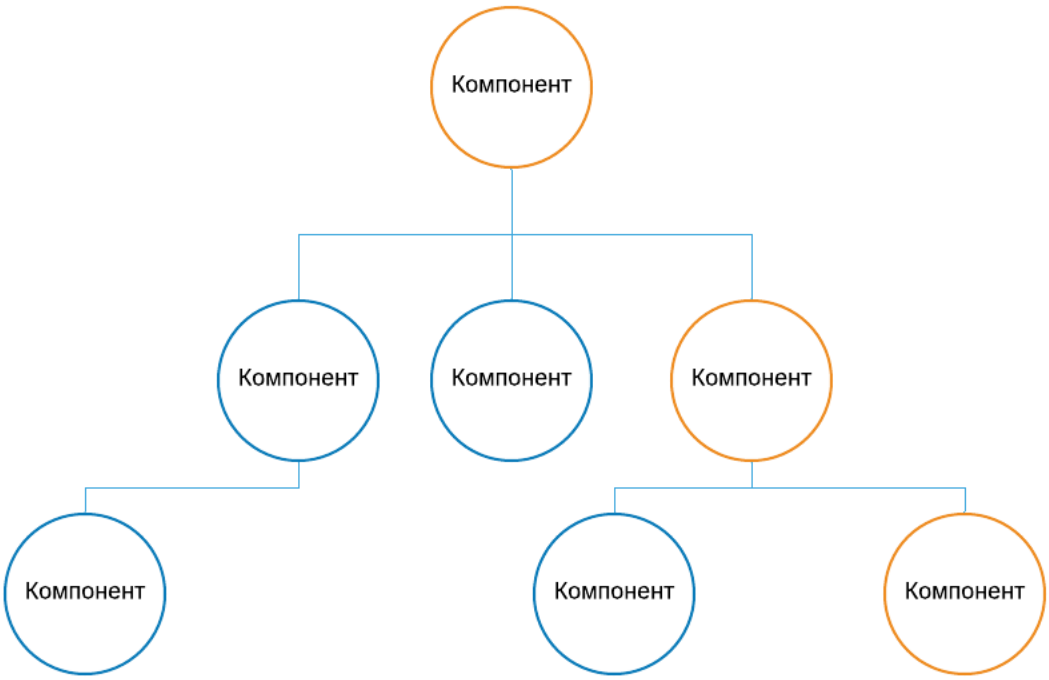

Рисунок 9 – Передача параметров состояния дочерним компонентам по дереву (оранжевый цвет)

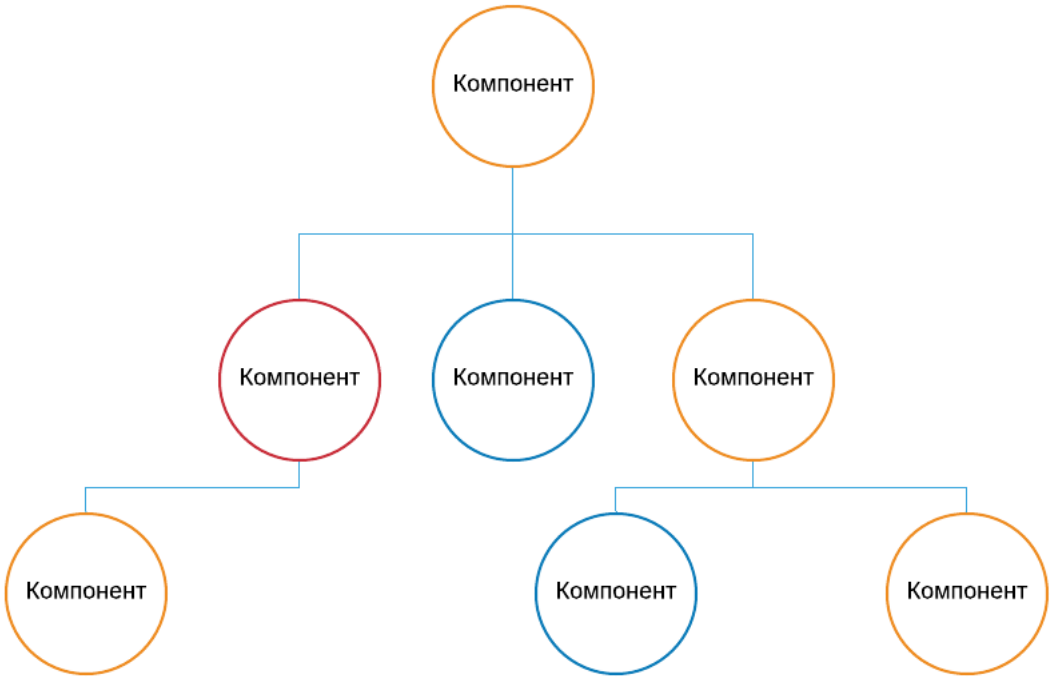

Рисунок 10 – Передача параметров (оранжевый цвет) компоненту, которому это не нужно (красный цвет), чтобы добраться до компонента, которому это нужно

Рисунок иллюстрирует, как и в предыдущем примере, что нужные параметры, выделенные оранжевым цветом, передаются вниз по дереву, но с небольшим отличием. Параметры также необходимо передать смежным компонентам внизу дерева, оранжевый круг в нижней правой ветке дерева. Для этого используется транзитный компонент, выделенный на рисунке красным цветом. Получается транзитный компонент принимает на себя ненужные и не свойственные ему параметры для того, чтобы их просто передать дальше.

Такая практика не обязательно плоха и может считаться хорошей, однако это может быть проблемой, особенно в процессе рефакторинга.

Также это обусловлено еще архитектурой и самой концепцией HTML представляющей из себя дерево или по-другому DOM.

DOM означает «объектная модель документа». DOM – это платформа и нейтральный к языку интерфейс, который позволяет программам и скриптам динамически получать доступ и обновлять содержимое, структуру и стиль документа. С помощью объектной модели документа программисты могут создавать документы, перемещаться по их структуре и добавлять, изменять или удалять элементы и содержимое. DOM представляет документ как узлы и объекты. Таким образом, языки программирования могут подключаться к странице. DOM – это объектно-ориентированное представление веб-страницы, которое может быть изменено с помощью языка сценариев, такого как JavaScript. Все, что найдено в HTML документе, можно получить, изменить, удалить или добавить с помощью объектной модели документа.

А поскольку сегодня все современные веб-приложения являются одностраничными. Одностраничное приложение – это приложение, которое работает внутри браузера и не требует перезагрузки страницы во время использования. Одностраничное приложения запрашивает разметку и данные независимо и отображает страницы прямо в браузере. Одностраничные сайты помогают держать пользователя в одном, удобном веб-пространстве, где контент представлен пользователю простым, легким и работоспособным способом.

Это все накладывает ограничения на поток данных, по дереву. Но в современной разработке веб-приложений эта проблема решается с помощью библиотек, которые управляют состоянием веб-приложений. Самый популярные эти библиотеки Flux, MobX, Redux. В основе этих библиотек лежит идея в том, чтобы управлять состоянием для приложений JavaScript. Они основаны на следующих принципах: состояние и логика приложения хранится в хранилище, и любое состояние, необходимое каждому компоненту из этого хранилища, может получить доступ. Для обновления хранилища необходимо отправить действие, описывающее изменения состояния. Это действие заменит состояние, что означает, что состояние приложения является неизменяемым.

Выводы по главе

Управление состоянием веб-приложения является запутанная тема, в которой очень сложно сразу разобраться. Особенно это становиться запутанным и важным когда нужно, создать интерактивность для пользователей, что подразумевает асинхронную логику. Нужно постоянно отслеживать состояние, реагировать на изменения пользователя и полученных данных с сервера. А также быть уверенным, что состояние храниться в единственном источнике, что упрощает разработку и отладку приложения.

Глава 3 ОПИСАНИЕ ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ДЛЯ РАЗРАБОТКИ СОВРЕМЕННЫХ ВЕБ-ПРИЛОЖЕНИЙ

3.1 Технология проектирования «Визуальное проектирование хранилища параметров сеанса»

Эта глава будет посвящена проектированию современных веб-приложений, с помощью технологии «Визуальное проектирование хранилища параметров сеанса». Данная технология нужна для того, чтобы задействовать в разработке такую технику как управление состоянием веб-приложения. Эта технология может применяться для любой модели жизненного цикла разработки веб-приложений.

В основе любого проектирования веб-приложения необходимо решать следующие проблемы:

1) проблемы, связанные с отображением (представлением) данных и информации;
2) проблемы, связанные с навигацией, касающимися структуры и поведения навигации;
3) проблемы, связанные с функциональностью приложения, которые выходят за рамки навигации
4) проблемы, связанные с вопросами интерфейса и дизайна в целом.

Поэтому в общем виде любая модель или технология проектирования, должна постараться решить большую часть этих проблем, если не все.

Но как можно заметить здесь не раскрыт вопрос, управления состоянием приложения. На самом деле это очень большая проблема в разработке веб-приложения и решение этой задачи является одной из наиболее сложных. Потому что, состояние – это то, что приложение знает о пользователе, его текущем взаимодействии с приложением и других частях глобальной информации. Такими сведениями могут быть, как модель пользователя, информация о сессии, где он в текущий момент находится в приложении с точки зрения навигации, какую информацию он ввел до, какое состояние сейчас у интерфейса, информация о доменных моделях и другие важные сведения о

конфигурации приложения. Эта информация остаются в памяти приложения до тех пор, пока приложение не будет завершено или пока они не будет удалено из памяти.

Управлять этим постоянно меняющимся состоянием очень трудно. Если модель может обновить другую модель, то представление может обновить модель, которая обновляет другую модель, и это, в свою очередь, может привести к обновлению другого представления. В какой-то момент Вы перестали понимать, что происходит в вашем приложении, как вы потеряли контроль над тем, когда, почему, и как его состояние. Когда система непрозрачна и недетерминирована, трудно воспроизвести ошибки или добавить новые функции.

Вот почему поддержание состояния в веб-приложении — это актуальная задача.

Технология «Визуальное проектирование хранилища параметров сеанса», в итоге помогает создать модель, а далее реализовать состояние веб-приложения с помощью общего хранилища параметров сеанса. Эта модель будет недостающим звеном в общем цикле (модели) разработки веб-приложения. Технология представляет из себя процесс и включает в себя 3 этапа (Рисунок 11).

Далее каждый этап будет рассмотрен более детально, для того чтобы лучше понять процесс.

Этап 1. Создание модели состояний. Для того чтобы построить эту модель нам нужно уже иметь на руках макеты экранов или их модели, представление о модели данных, а также модель предметной области. Все эти модели можно получить с помощью любого метода проектирования веб-приложений. Это будет необходимый входной набор данных для рассматриваемой технологии. Макеты экранов нужны для того, чтобы можно понять какие графические элементы должны, изменятся т.е. иметь различное состояние. Модель данных и модель предметной области нужна для того, чтобы выявить общие элементы, которые нужны во всем приложении.

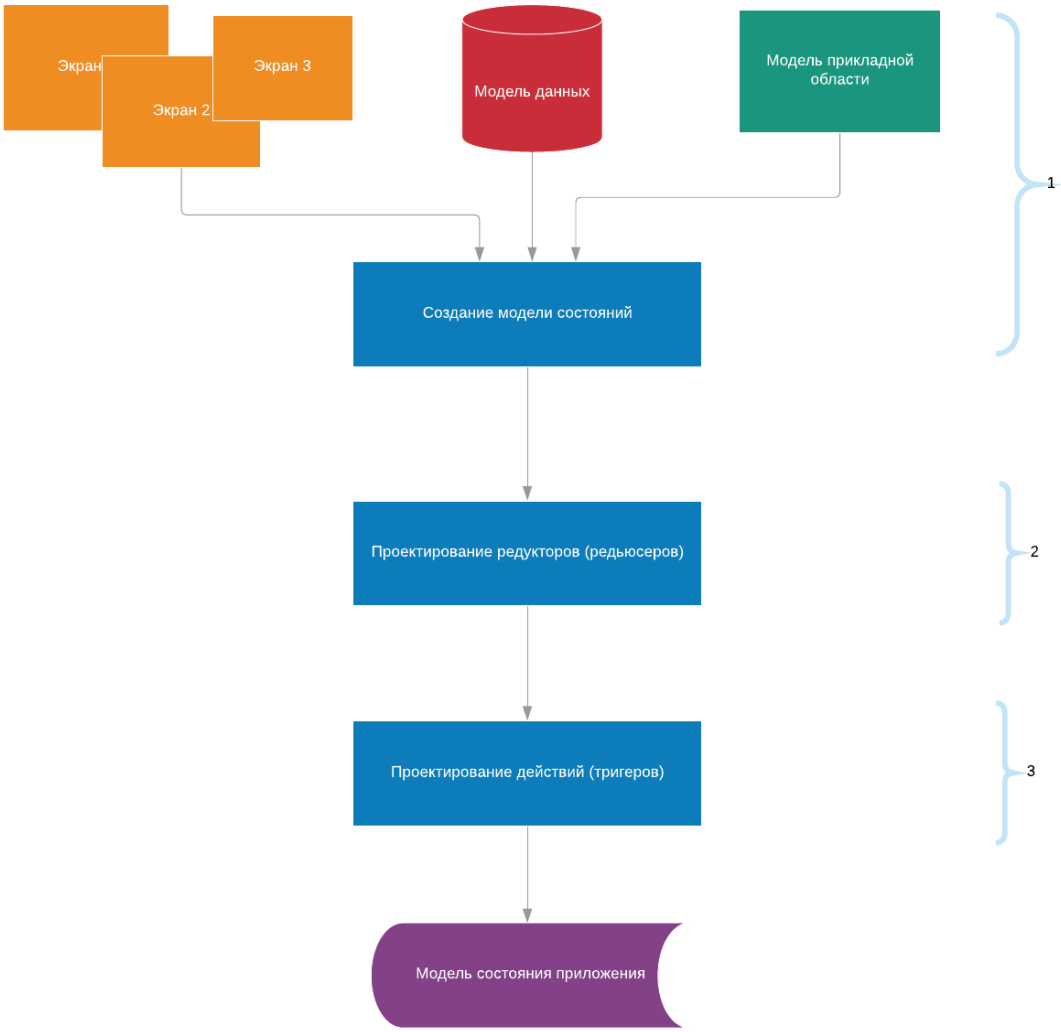

Рисунок 11 – Технология создания состояния приложения (общего хранилища параметров сеанса)

Например, в качестве примера, мы разрабатываем веб-приложение электронной коммерции, которое будет интегрировано в общую корпоративную сеть. Где у нас есть уже некоторые макеты экранов (прототипы низкой точности или wireframes), такие как экран входа, каталог товаров, информация о товаре и экран оплаты заказа (Рисунки 12-15).

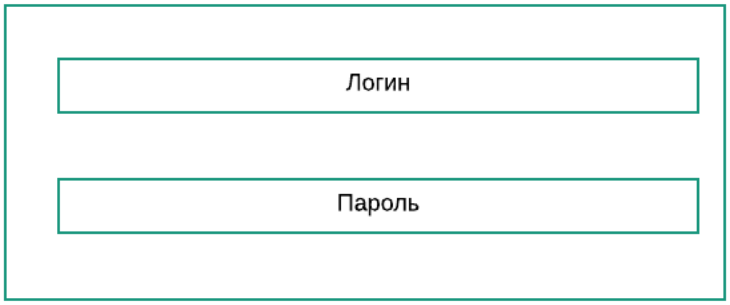

Рисунок 12 – Экран авторизации

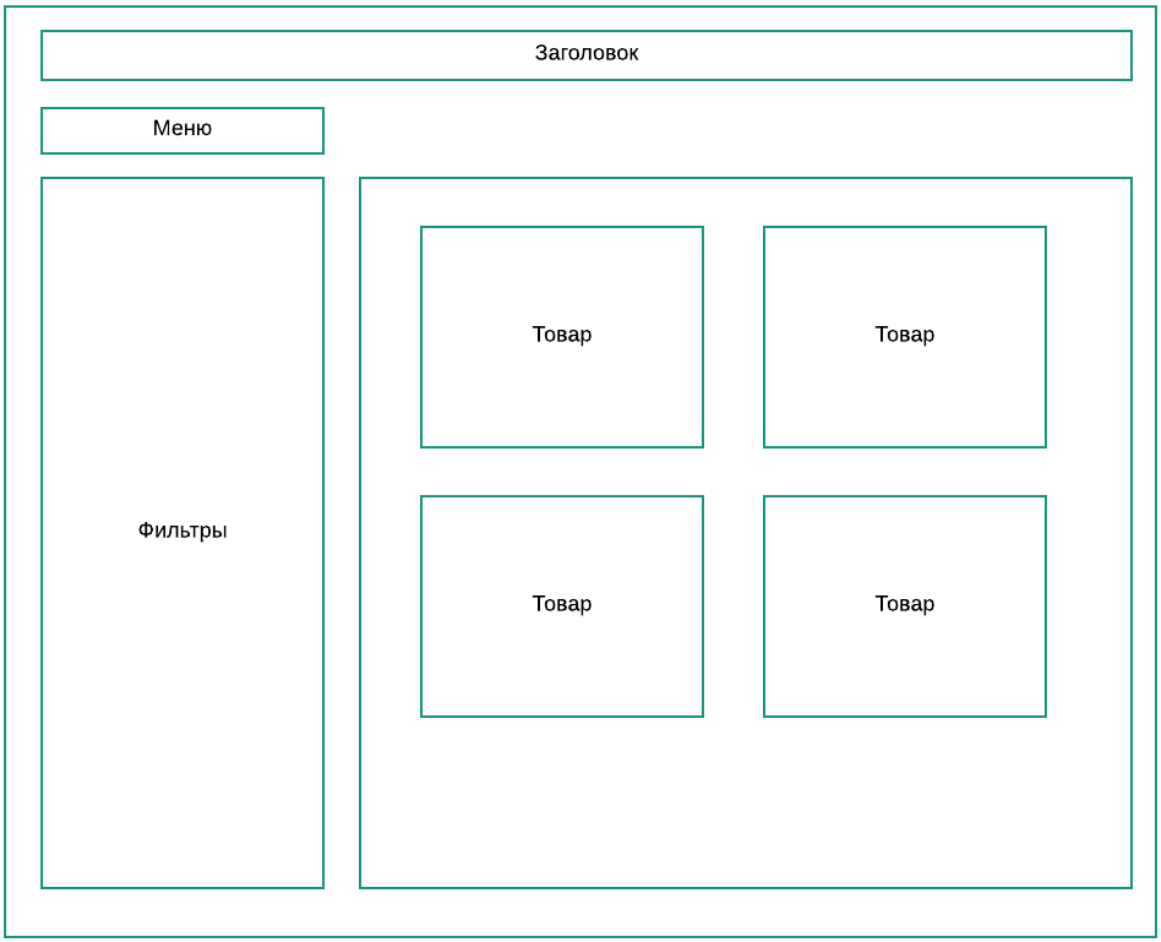

Рисунок 13 – Экран список товара

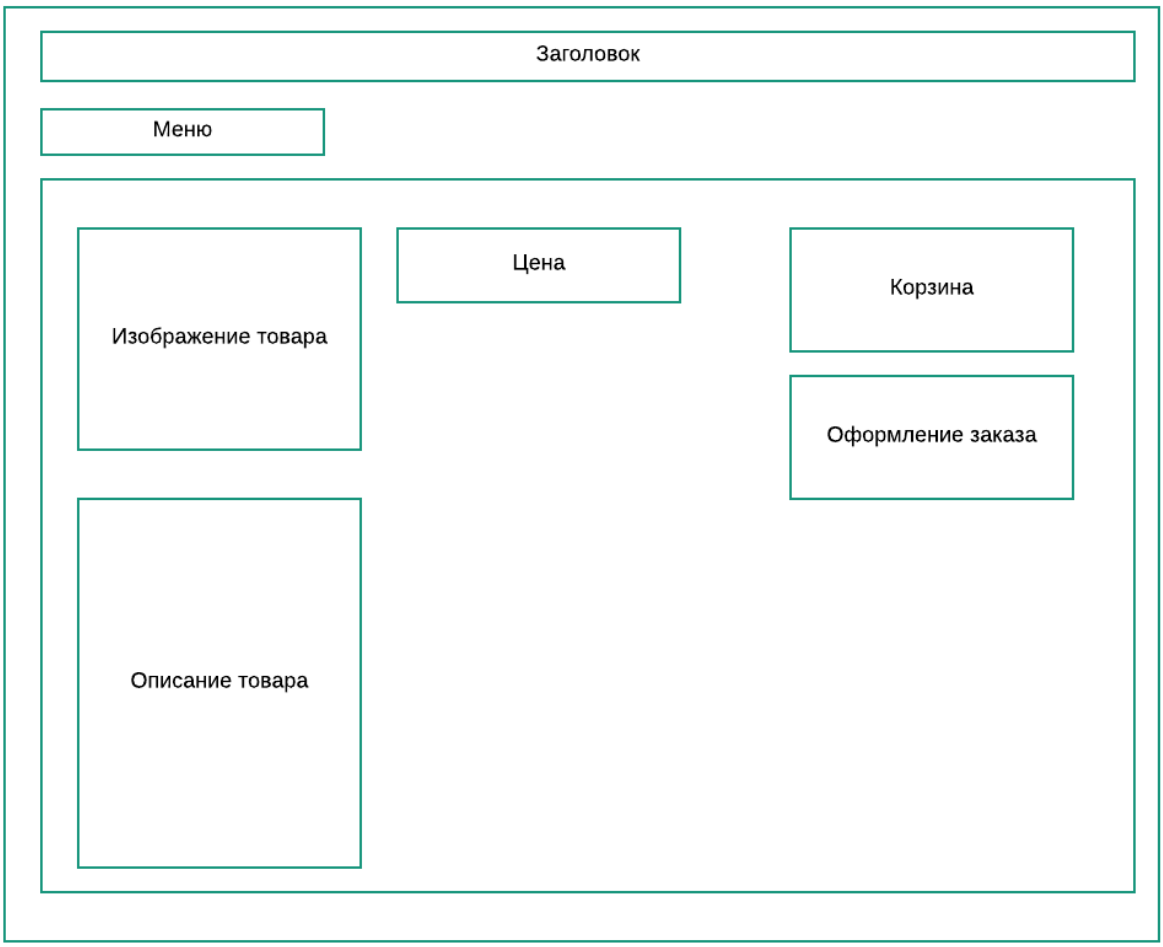

Рисунок 14 – Информация о товаре

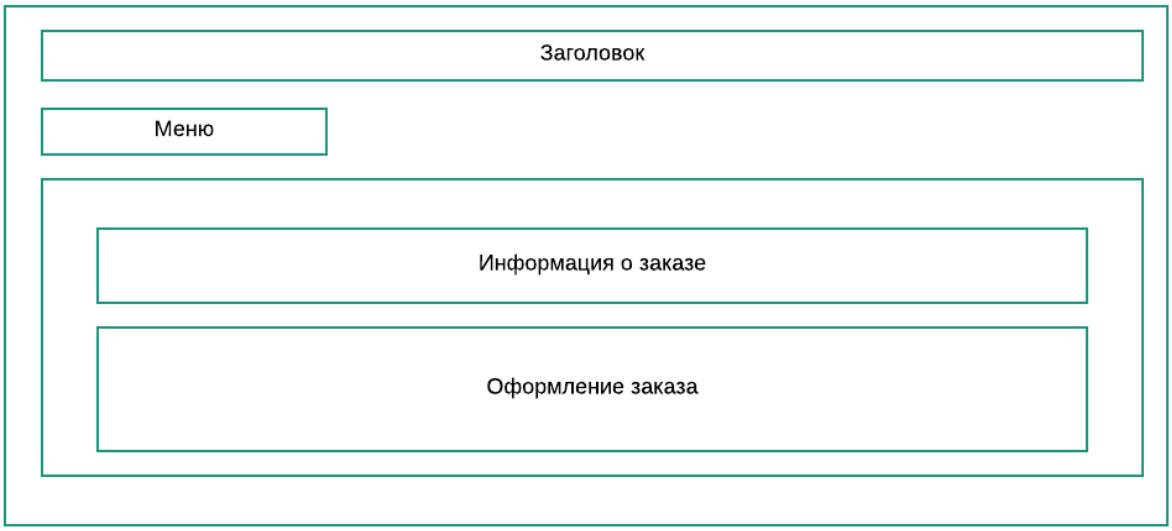

Рисунок 15 – Оформление заказа

Также у нас есть модель данных пользователя и модель предметной области список товаров.

Исходя из этой информации можно составить модель состояний. Модель состояний представлена в виде дерева. В корне дерева находится само веб-приложения, а ветви это уже сущности, которые означают каждый экран или модель предметной области или модель данных (Рисунок 16).

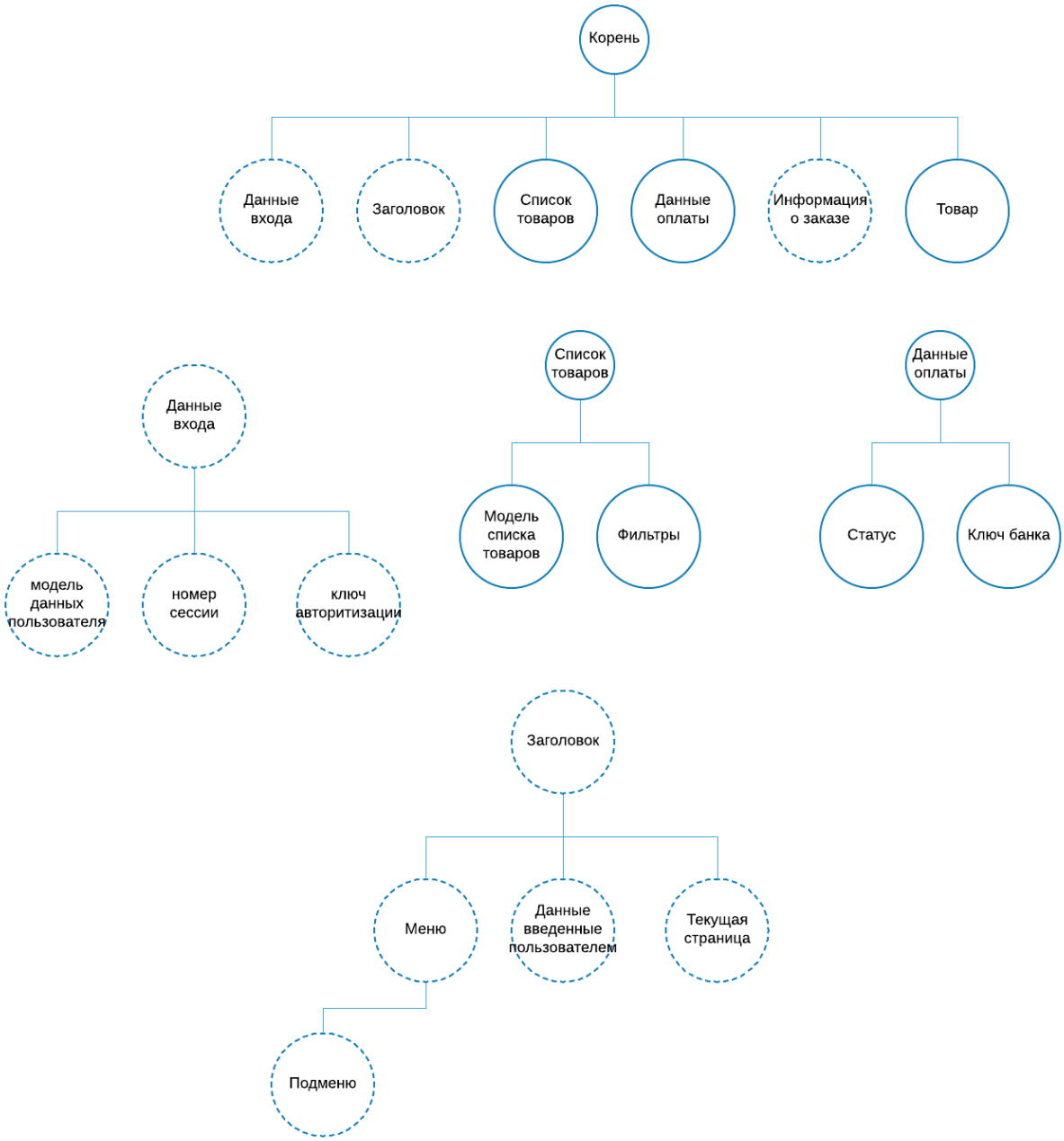

Рисунок 16. Модель состояний в виде дерева

Как видно из рисунка у нас есть визуализированное дерево состояний веб-приложения. Это очень важная часть, как видно из рисунка некоторые сущности имеют прерывистую линию, а некоторые сплошные. Прерывистой линией обозначены сущности, к которым необходимо иметь доступ из любой части веб-приложения, а сплошную линию имеют сущности, которые специфичных для конкретного экрана и являются уникальным только для него. Как нетрудно догадаться, что сущности с прерывистой линией лягут в основу объекта хранилища состояния всего веб-приложения.

Этап 2. Проектирование редукторов. Редукторы обозначает как состояние приложения должно измениться в ответ на действия, которые пытаются изменить объект хранилища. Действия описывают только то, что произошло, они не описывают, как изменяется состояние приложения.

Каждое из состояний может иметь свои собственные редукторы. Позже мы можем объединить их в один корневой редуктор, который в конечном итоге определит хранилище (единственный источник истины приложения). Таким образом мы будем полностью контролировать свои состояния и их поведение. В качестве примера рассмотрим проектирования редуктора с помощью дерева состояний приложения, которое было приведено выше (Рисунок 17).

```
// корневой редуктор
const rootReducer = combineReducer({
  header: headerReducer,
  loginData: loginReducer,
  orderInfo: orderInfoReducer,
  common: commonReducer,
  product: productReducer,
  productList: productListReducer,
  payment: paymentReducer
});
```

Рисунок 17 – Корневой редуктор (листинг на языке JavaScript)

Корневой редуктор содержит все, что хранилище должно знать о приложении. Теперь давайте посмотрим, как выглядит редуктор данных входа в веб-приложение. Само состояние выглядит так (Рисунок 18)

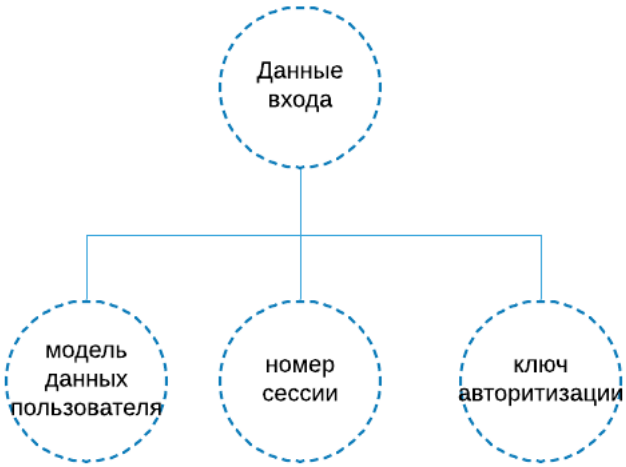

Рисунок 18 – Модель состояния данных входа (авторизации) веб-приложения

Можно обратить внимание, как один редуктор содержит в себе другие редукторы. Поэтому нам не нужно создавать один огромный редуктор. Его можно легко разбить на более мелкие редукторы, поскольку каждый из них имеет свою особенность поведения и свои собственные специфические операции. Это поможет создать разделение логики, что очень важно для разработки и поддержки больших веб-приложений приложений.

Дальше, будет показана модель как должен выглядеть типичный редуктор, например userReducer (Рисунок 19).

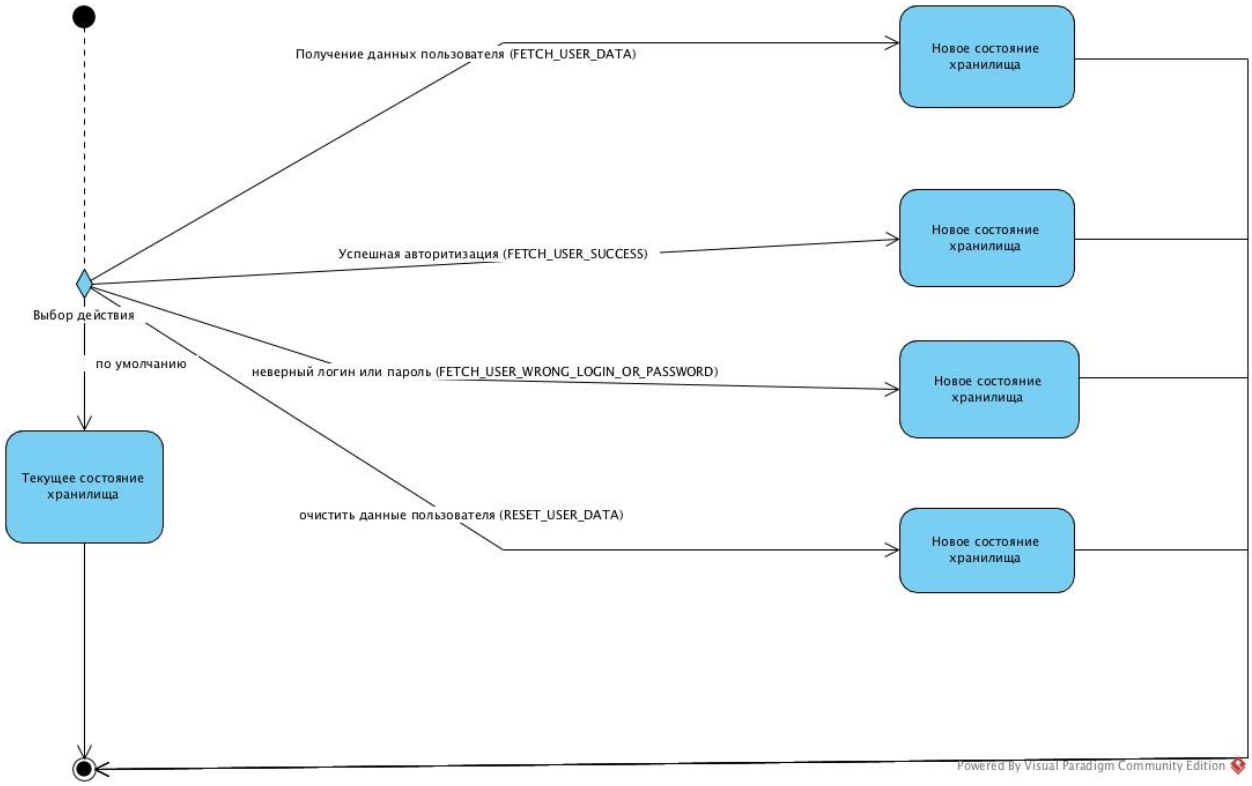

Рисунок 19. - Редуктор получения модели пользователя

Показанный шаблон редуктора определяет, возможные изменения, через изменения состояния модели пользователя при вызове функций авторизации. Они определяют какие возможные действия могут быть выполнены для данного редуктора. (Рисунок 20).

```
FETCH_USER_DATA, FETCH_USER_SUCCESS, FETCH_USER_WRONG_LOGIN_OR_PASSWORD, RESET_USER_DATA
```

Рисунок 20 – Триггеры действий, которые изменяют состояние хранилища

Этап 3. Проектирование действий триггеров. Продолжая проектирование на примере авторизации, можно понять, что действия могут быть сгруппированы в одну функцию, в данном случае авторизацию пользователя (Рисунок 21).

```
function logonUser(authData) {
  return async (dispatch) => {
    // инициализируем начальное состояние
    dispatch({
      type: FETCH_USER_DATA
    });

    try {
      // обращение к серверу
      const result = await authUser(authData);

      // обновить состояние если пользователь
      // прошел авториизацию
      dispatch({
        type: FETCH_USER_SUCCESS,
        userData: result.userData,
      });
    } catch (err) {
      // если произошла ошибка во время авторитизации
      dispatch({
        type: FETCH_SEARCH_FAILURE,
        error: err
      });
    }
  };
}
```

Рисунок 21 – Пример кода аутентификации (входа в приложение) пользователя

Можно увидеть, как поток данных отслеживается хранилищем с помощью действий. Это помогает каждое изменение в приложении делать подотчетным и отслеживаемым. При этом подобные действия записываются для каждого изменения в редукторах различных состояний. Одним из самых больших преимуществ Redux является абстракция каждого действия (Рисунок 22).

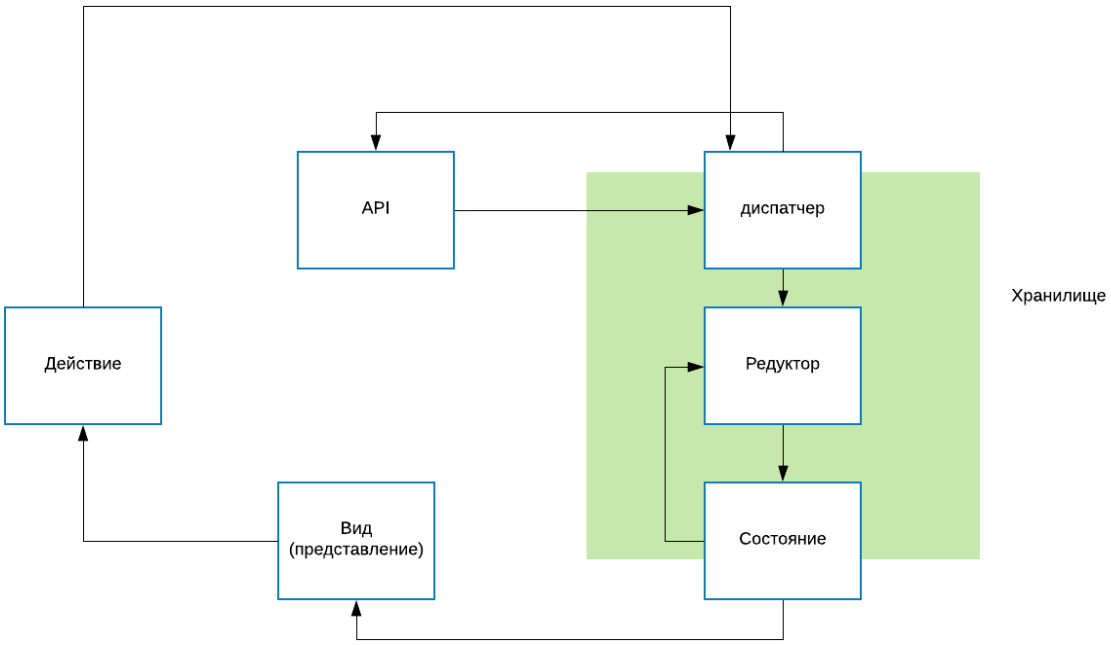

Рисунок 22 – Поток данных в веб-приложении с использованием Redux

В финальной стадии у нас на руках имеются все необходимые данные, шаблоны, модели, чтобы начать реализацию модели состояния приложения в разрабатываемом веб-приложении. Таким образом, с помощью визуального проектирования хранилища и технологии Redux, удается спроектировать недостающее звено во всех моделях разработки. Это недостающее звено (управление состоянием приложения), позволит создавать большие и легко масштабируемые веб-приложения.

3.2 Описание применения технологии «Визуальное проектирование хранилища параметров сеанса»

Данная технология может применяться для любой методологии работки веб-приложений. Ее можно использовать как инъекцию, чтобы внедрить управление состоянием веб-приложения. Но здесь также важно соблюдать последовательность. Для того чтобы начать использовать эту технологию, для нее на входе нужны первоначальные данные. Это будут макеты экранов приложения, модель данных приложения и модель предметной области. Без этих начальных данных применение этой технологии невозможно.

Например, если используется методология WSDM, то согласно этой методологии на этапе проектирования сначала создается модель задач. После создается информационная модель, затем проектируется навигация по приложению, что в свою очередь создаст навигационную модель веб-приложения. Потом разрабатывается визуальная структура веб-приложения и ее дизайн. И вот только тогда, когда на руках уже есть все данные, можно начать использовать технологию «Визуальное проектирование хранилища сеансов». Вообще строго говоря, эту технологию, можно начинать применять перед этапом реализации веб-приложения. Потому как на перед эти этапом у разработчиков точно должны быть уже на руках все необходимые данные и модели. Придерживаясь такого подхода, не должно возникнуть трудностей с применением этой технологии.

Выводы по главе

Созданная технология «Визуальное проектирование хранилища параметров сеанса», позволяет создавать современные веб-приложения, в которых задействована такая технология как Redux. Применяя эту, технология для добавления хранилища состояния приложения мы получаем принцип единой ответственности, который, обеспечивает простоту кода разрабатываемого веб-приложения, поскольку каждая функция может делать только одно. Единственной обязанностью использования состояния приложения является управление этим состоянием. Важно также отметить, что данная технология, может применяться вместе с любой моделью разработки, принятой в компании.

Глава 4 ЭКСПЕРИМЕНТАЛЬНАЯ ПРОВЕРКА РАЗРАБОТАННОЙ ТЕХНОЛОГИИ

4.1 Экспериментальная проверка разработанной технологии с помощью реализации веб-приложений на языке JavaScript

Был проведен экспиримент, чтобы выявить какие положительные или отрицательные результаты даст применение разработанной технологии в разработке веб-приложений, и вообще как сказывается использование такой техники управление состоянием приложения с помощью общего хранилища параметров сеанса. Для проведения эксперимента было спроектировано и реализовано два простых веб-приложения в одном уделяется внимание проектированию хранилища сеансов, а в другом нет. Реализация была осуществлена с помощью известного Java Script framework ReactJS.

JavaScript framework – это большой набор функций и инструментов для, со своим собственным потоком управления. В качестве абстракции framework предоставляет комплексную платформу разработки, один из стандартных способов построения приложений. Он может обеспечить управление зависимостями, структуру файловой системы и возможностями маршрутизации.

Сами framework состоят из компонентов. Компоненты являются основными строительными блоками framework. Способ инициализации и обработки компонентов варьируется между framework, но основные характеристики одинаковы: компоненты могут принимать информацию в качестве параметров, выполнять действия и, возможно, возвращать некоторый результат. Компоненты взаимодействуют друг с другом, могут использовать свойства друг друга и могут иметь родительские или дочерние компоненты. Это очень важно понимать того, как работает веб-приложения в плане передачи информации между компонентами. Применение framework в эксперименте, призвано чтобы усилить появление экспериментальных результатов. Именно использование framework выявить все проблемы передачи потока данных между компонентами.

Результаты эксперимента должны показать, насколько оправданно применение разработанной технологии при разработке веб-приложений. По результатам эксперимента будет проведен сравнительный анализ, который и должен показать оправдано или неоправданно применение данной технологии.

Экспериментальное приложение, представляет из себя список сотрудников с возможностью поиска по списку и возможностью добавления нового сотрудника в список.

После того, как были выяснены требования к приложению, можно выявить сценарии использования, их будет всего три (Рисунок 23).

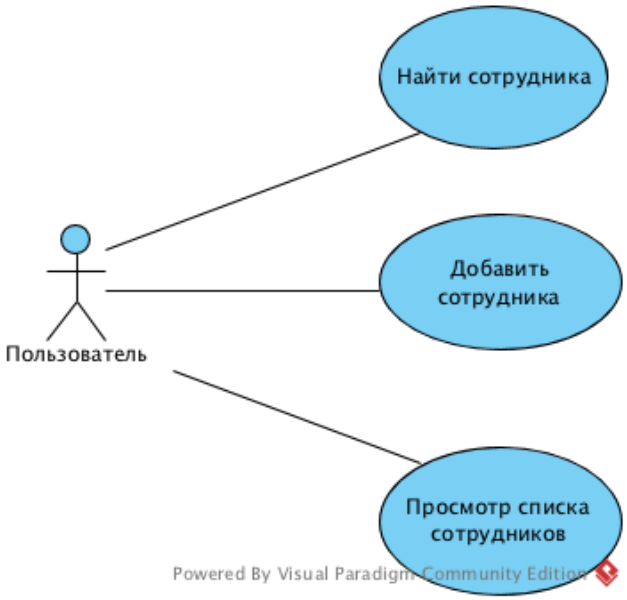

Рисунок 23 – Диаграмма вариантов использования

Далее спроектируем модель данных. Она будет очень простая. Это модель самого сотрудника и модель списка всех сотрудников (Рисунок 24).

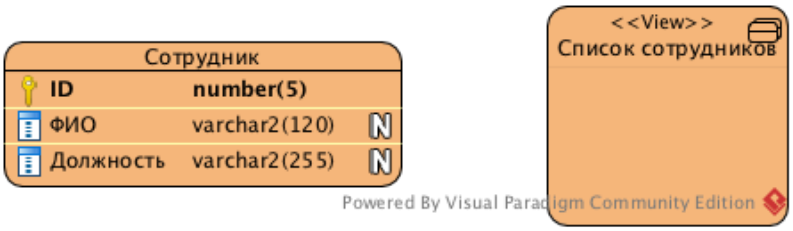

Рисунок 24 – Модель данных

После этого можно создать макеты экранов, их всего будет один. На основе предыдущих моделей экран приложения, который включает в себя выполнение всех сценариев использования может выглядеть так (Рисунок 25).

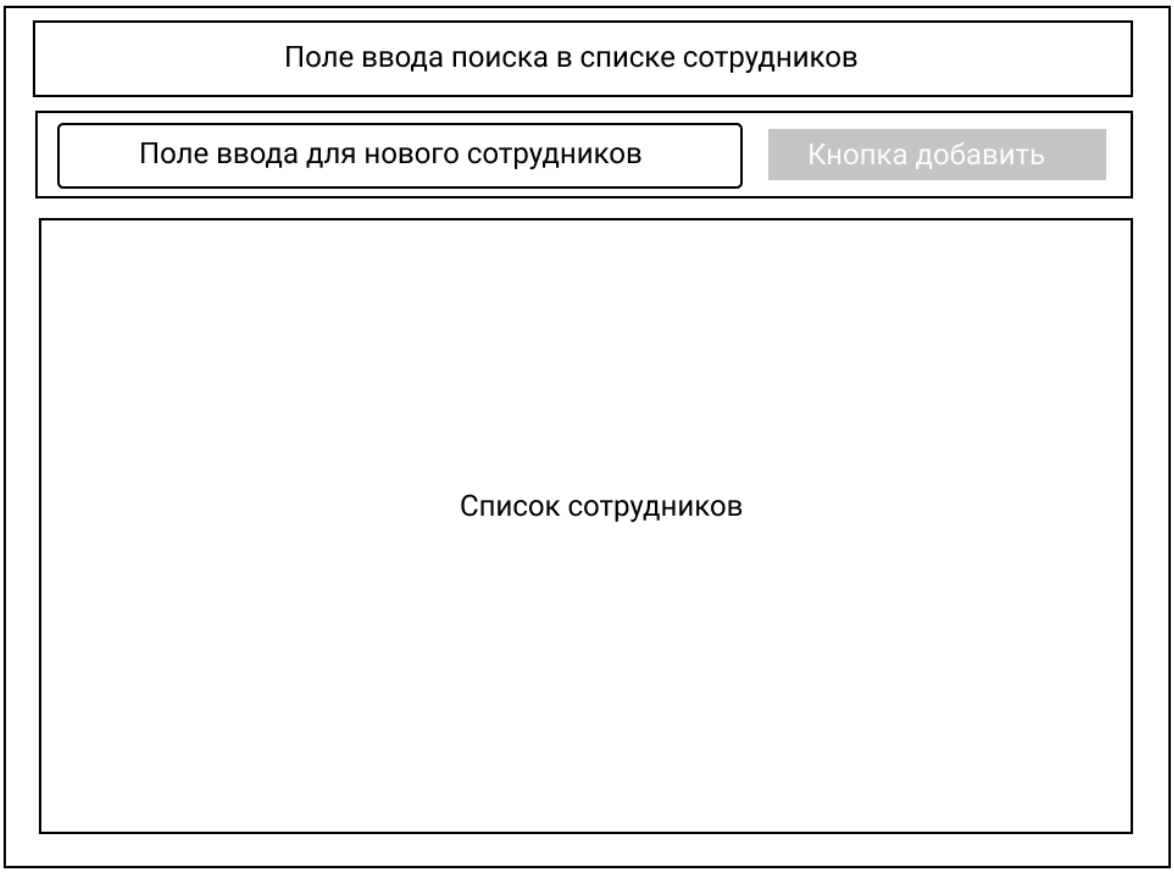

Рисунок 25 – Эскиз экрана приложения

Дальше уже можно приступить к реализации веб-приложения без учета применения рассматриваемой технологии. Но поскольку нас интересует эксперимент, то дальше необходимо создать модель состояний, с помощью технологии «Визуальное проектирование хранилища параметров сеанса», для этого у нас на руках уже есть весь необходимый материал.

Первое что необходимо сделать создать (визуализировать) сущности, которые есть на экране в виде дерева (Рисунок 26).

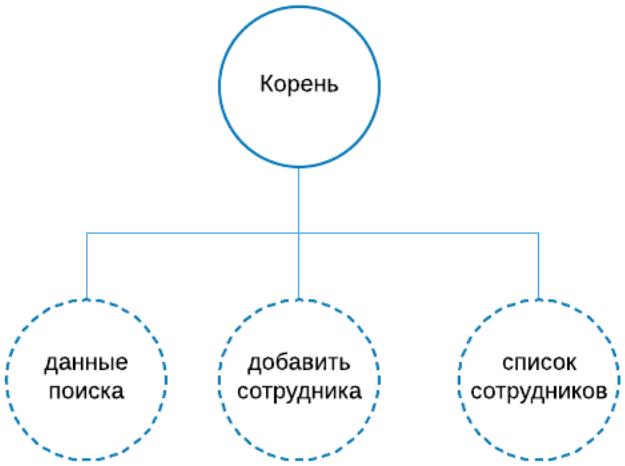

Рисунок 26 – Модель состояний в виде дерева

Как видно из рисунка корень дерева это приложение, дальше идут элементы, обозначенные прерывистой линией, а это значит, что доступ к этим сущностям нужен из любого места приложения. Здесь видно, что присутствуют такие элементы как, элемент интерфейса пользователя, функциональный элемент и модель предметной области.

Теперь можно приступить к проектированию редукторов. Редукторы изменения данных поиска и изменения списка сотрудников выглядят следующим образом (Рисунок 27,28).

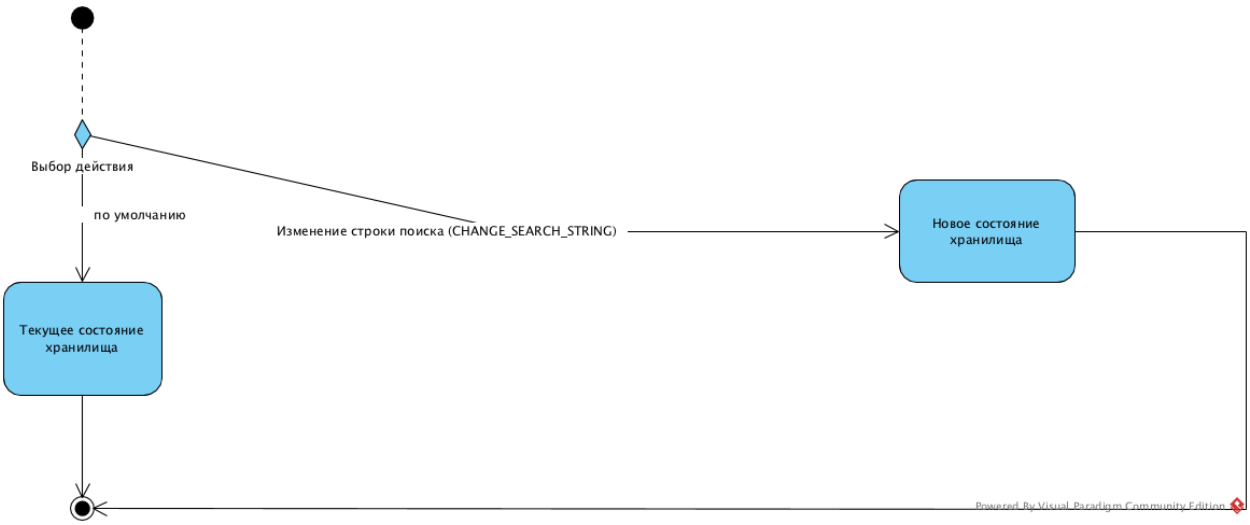

Рисунок 27 – Редуктор изменение данных поиска

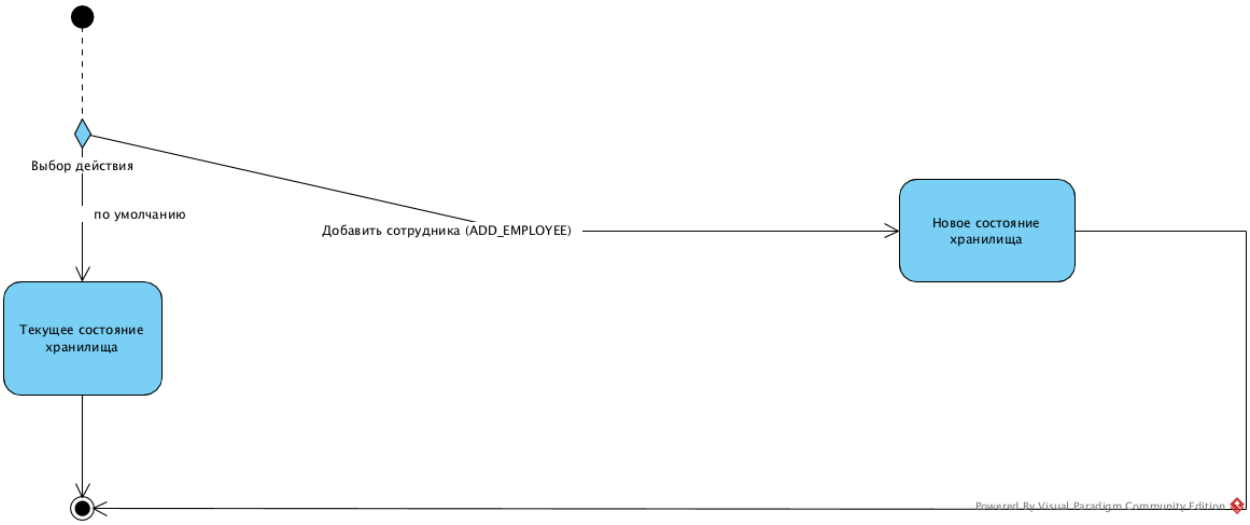

Рисунок 28 – Редуктор изменение списка сотрудников

Дальше выявляем экшены (триггеры), это будут CHANGE_SEARCH_STRING и ADD_EMPLOYEE. Соответственно далее их и спроектируем (Рисунок 29,30).

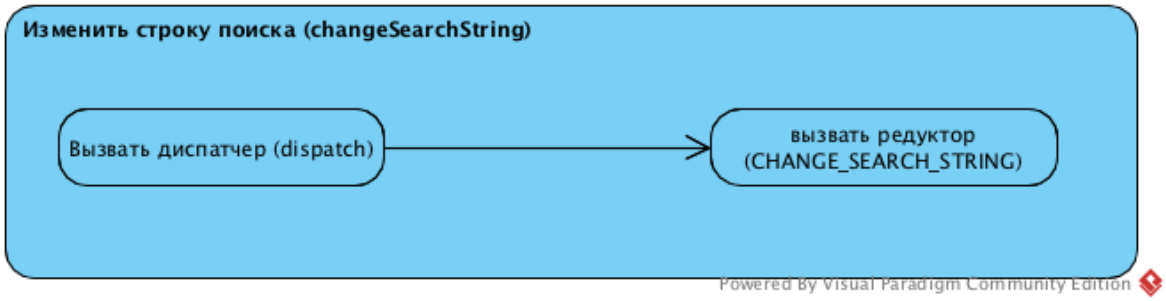

Рисунок 29 – Экшн (тригер) изменение строки поиска

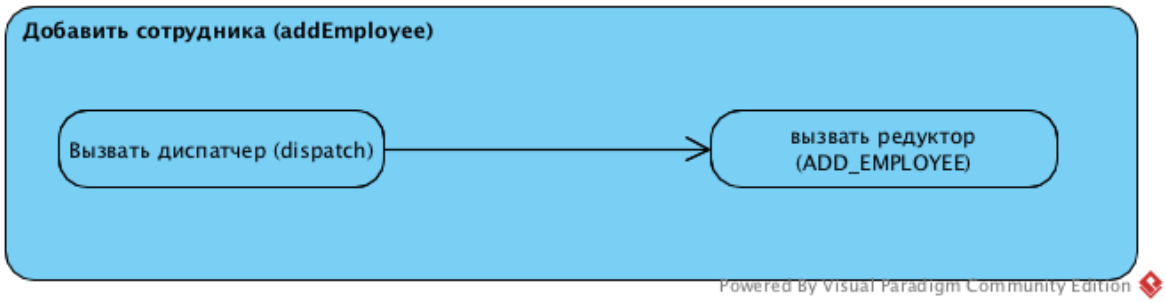

Рисунок 30 – Экшн (тригер) добавление нового сотрудника

Теперь, имея на руках все необходимое, можно приступить к реализации самого веб-приложения. Как было сказано выше, реализация приложения будет осуществляться с помощью библиотеки ReactJS, а реализация хранилища будет использовано с помощью библиотеки Redux. Визуально приложение выглядит так (Рисунок 31)

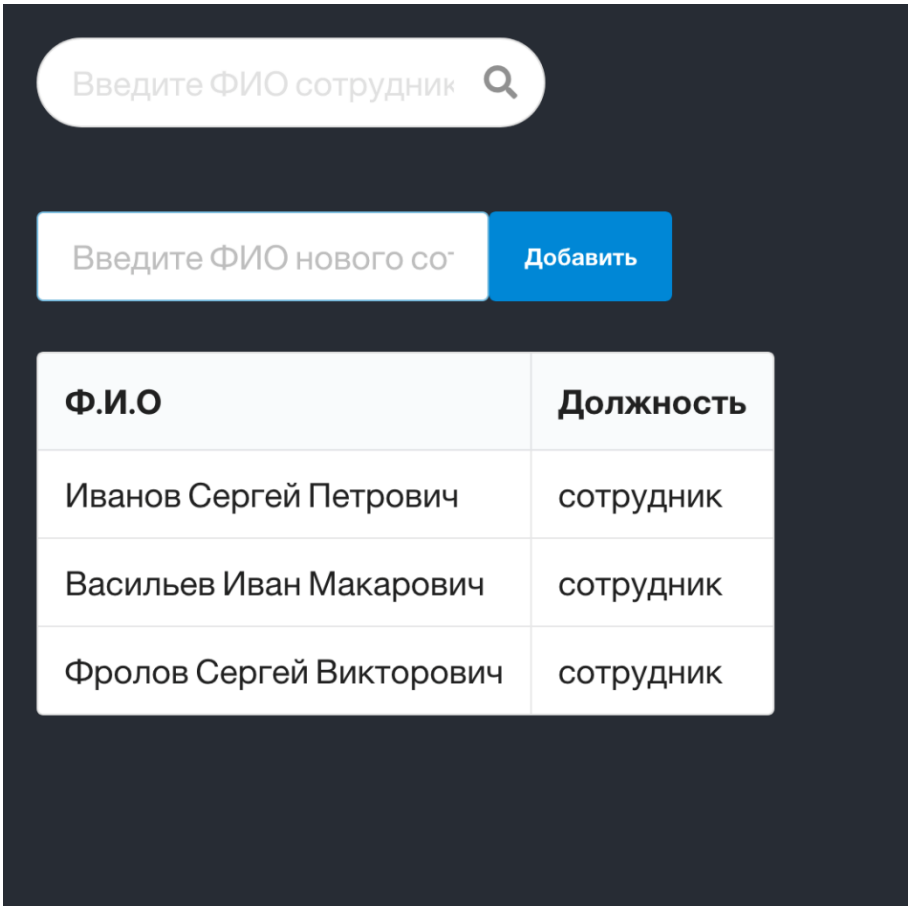

Рисунок 31 – Экспериментальное приложение

С точки зрения визуальной части эти приложения идентичны. Хоть они и спроектированный одно по классической методике, а другое с добавлением разработанной технологии. После того как приложения созданы будет произведен анализ этих двух приложений по следующим критериям:

- Сравнить разницу в управлении состоянием приложения;
- Сравнить сложность реализации масштабирования приложения;
- Сравнить возможность добавления такой концепции как «путешествие во времени».

4.2 Анализ и сравнение полученных результатов эксперимента

В результате эксперимента с помощью сравнения полученных результатов удалось выявить следующие результаты.

Сравнение разницы в управлении состоянием приложения и реализацию масштабирования.

В самом веб-приложении у нас есть различные компоненты, состояние которых предоставляется им в качестве реквизита. Это состояние может исходить из родительских компонентов, в которые они вложены, согласно общей практике разработки веб-приложений, а может исходить из любой точки при использовании хранилища состояний. Это простые функции, которые принимают на себя параметры и создают новое состояние без каких-либо побочных эффектов и поэтому они отделены от управления состоянием непосредственно. Поэтому при реализации веб-приложения на языке JavaScript в силу его архитектуры и архитектуры языка HTML, можно констатировать тот факт, что данные в приложении передаются в одном направлении, от корня элемента к его потомкам (Рисунок 32).

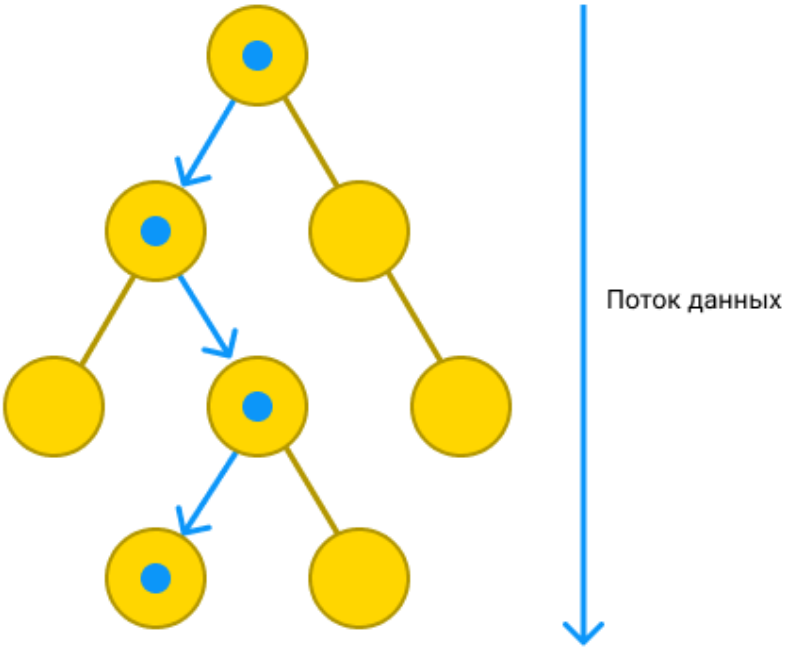

Рисунок 32 – Односторонний поток данных в приложении

Из-за такого ограничения, приходится проектировать структуру веб-приложения, его компоненты и элементы особым образом. В корне всего приложения, должны располагаться такие компоненты, которые отдают свои данные дочерним компонентам. Это очень сильно ограничивает и усложняет масштабируемость веб-приложения.

Если при проектировании применить технологию «Визуальное проектирование хранилища параметров сеанса» которая учитывает такое ограничение и позволяет задействовать такую технику как управление состоянием приложения через общее хранилище сеансов. Которую можно реализовать через такую библиотеку как Redux, то проблема с односторонним потоком данных решается очень хорошо (Рисунок 33).

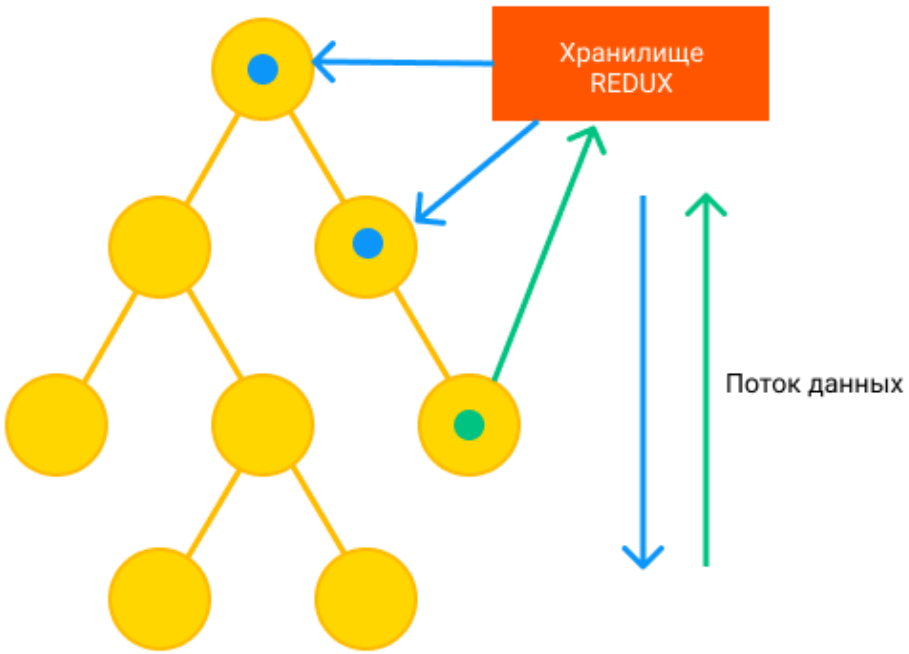

Рисунок 33 – Двухсторонний поток данных в веб-приложении

Из этого следует, что если веб-приложение проектируется без использования управления состоянием приложения, то появляется избыточность данных, передаваемых компонентам потомкам. Это приводит к тому, что усложняется код, потому как если приложение становиться большим и сложным, что всегда свойственно корпоративным приложениям. То может возникнуть

ситуация, что компоненту потомку нужно будет принять к себе скажем более десяти ненужных параметров только для того, чтобы их передать далее по дереву, нужному адресату. При использовании общего хранилища сеансов такой проблемы не будет. Для сравнения можно привести два рисунка. Каждый из них отображает как параметры передаются внутри приложения (Рисунок 34, 35).

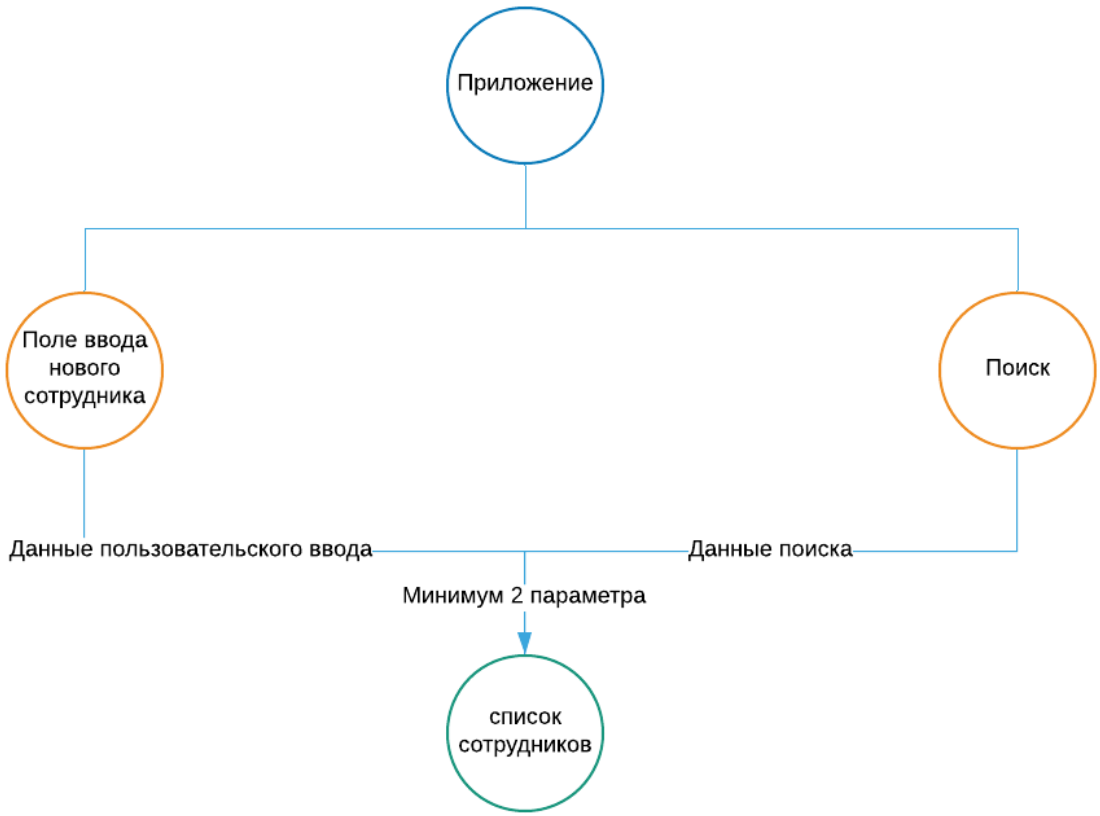

Рисунок 35 – Передача параметров компонентам в приложении без использования общего хранилища.

Рисунок иллюстрирует передачу параметров, выделенных оранжевым кругом, компоненту (Список сотрудников) выделенных зеленым цветом. И в случае масштабирования добавления нового компонента скажем который будет отвечать за сортировку и фильтры в таблице и в тоже время он должен быть корнем для таблицы, чтобы передавать ей свои параметры. Он будет вынужден принять на себя параметры от родительских компонентов, о которых он знать ничего не должен (Рисунок 36).

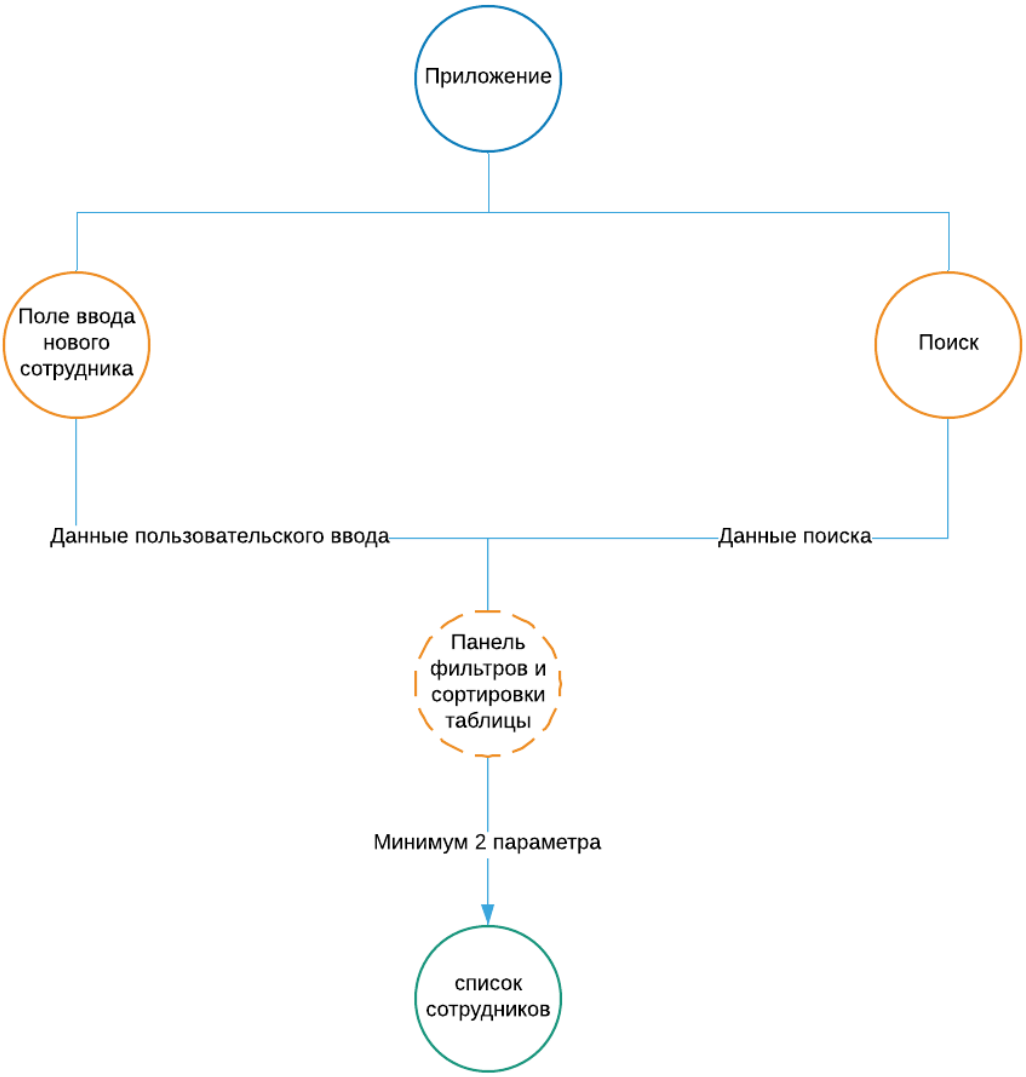

Рисунок 36 – Масштабированное приложение с добавлением одного компонента (оранжевый пунктир) без использования общего хранилища

Рисунок иллюстрирует как вновь добавленный компонент (оранжевый пунктир) становиться транзитным для компонента (список сотрудников) и должен передать вниз по дереву ему не свойственные параметры.

В случае использования общего хранилища параметров сеанса, схема реализации приложения выглядит уже по-другому (Рисунок 37).

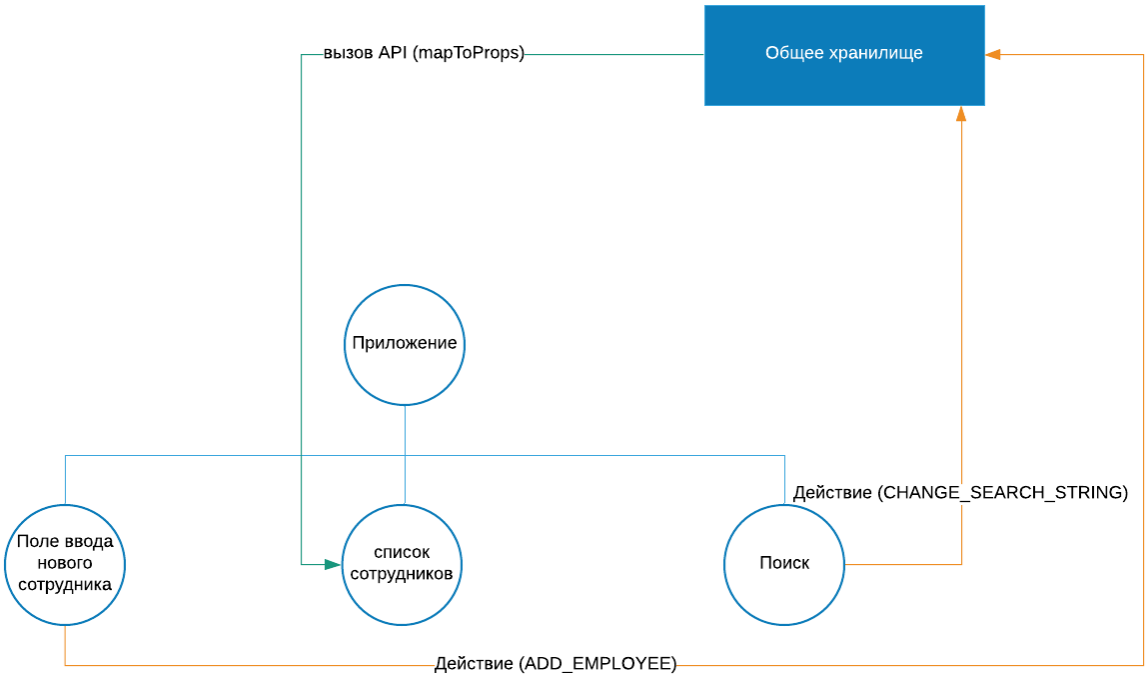

Рисунок 37 – Схема реализации веб-приложения с использованием общего хранилища

Рисунок показывает, как уже компоненты с помощью действий (триггеров) выделенных оранжевым цветом, изменяют общее хранилище, а хранилище делает вызов API в рассматриваемом эксперименте используется ReactJS и вызов mapToProps, который из хранилища передает нужные параметры компоненту (список сотрудников).

Если есть необходимость также добавить новый компонент, по управлению фильтрами и сортировкой таблицы. То схема реализации с помощью общего хранилища будет выглядеть так (Рисунок 38).

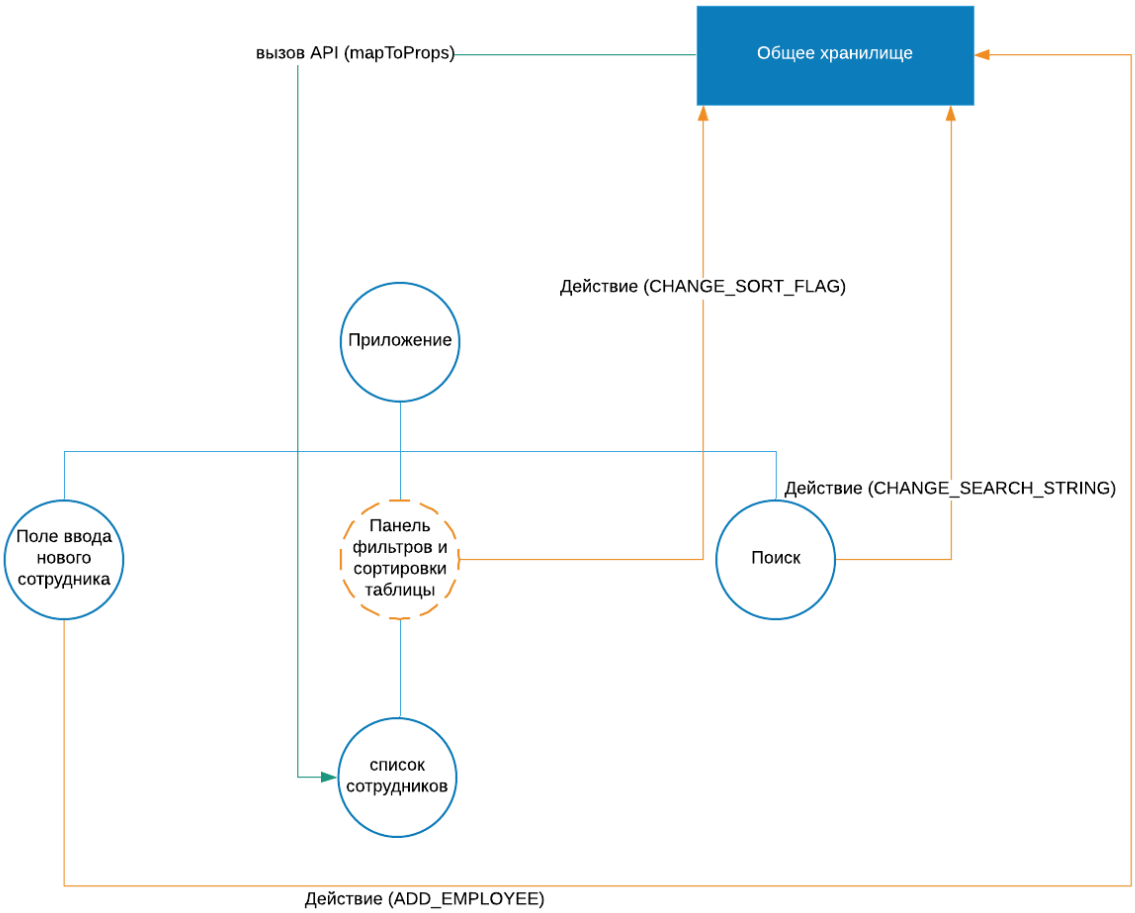

Рисунок 38 – Масштабированное приложение с добавлением одного компонента (оранжевый пунктир) с использованием общего хранилища

Здесь также, как и в примере без использования общего хранилища, добавлен новый компонент (панель фильтров и сортировки таблицы) оранжевый пунктир. Это компонент теперь просто вызывает функцию на изменение флага сортировки и передает это в общее хранилище, в свою очередь список сотрудников просто принимает на себя новый флаг и сортирует таблицу в соответствии с заданными настройками.

Этот пример наглядно показывает разную реализацию с использованием общего хранилища и без использования общего хранилища. Применение общего хранилища в разработке веб-приложений позволяет легко создавать очень большие и простые в поддержке корпоративные веб-приложения. Состояние приложения полученные, например с сервера, могут быть сериализованы на клиенте без дополнительных усилий. Используя одно и единственное хранилище

также упрощает отладку или проверку приложения; оно позволяет сохранять состояние приложения в процессе разработки для более быстрого цикла разработки. Даже можно реализовать некоторые функции, которые традиционно трудно сделать.

Таким образом приложение полностью является масштабируемым и гибким, без каких-либо ограничений на поток данных и контроля состояний. Также с помощью поставленного эксперимента можно увидеть, как проектирование повлияло на реализацию и дельнейший жизненный цикл разрабатываемого веб-приложения. Поэтому все-таки при разработке веб-приложений не стоит пренебрегать проектированием в каком-либо виде.

Выводы по главе

Эксперимент показал, что применение технологии «Визуальное проектирование хранилища параметров сеанса» в качестве инъекции в методологию использования классических методов проектирования, позволяет создавать современные веб-приложения, которые в будущем очень легко масштабировать и осуществлять их поддержку. Данная технология учитывает такое важно и недостающее звено в классических моделях проектирования, как управление состоянием приложения. Результаты эксперимента выявили различия в разработанных веб-приложениях для эксперимента. Эти различия показали, что применение общего хранилища параметров сеанса для управления состоянием веб-приложения дает положительный результат. Что сказывается на эффективности самой разработки и в дальнейшем поддержки веб-приложения.

Поэтому не стоит пренебрегать данной технологией, когда необходимо начать разработку нового веб-приложения. Также может быть и стоит подумать над тем, чтобы текущее веб-приложение, которое уже запущенно в эксплуатацию, перевести на использование данной технологии, поэтапно. Сам перевод текущего работоспособного веб-приложения, выходит за рамки данного исследования и здесь не рассматривается.

ЗАКЛЮЧЕНИЕ

В ходе исследовательской работы проведены исследования научно-технической литературы, различные статьи и публикация, а также научные журналы, по методологии разработки веб-приложений любых типов.

На основании общей теории различных методик, методов, моделей разработки современных веб-приложений. Был проведен анализ веб-приложения как объекта проектирования. Были даны критерии того, что представляет из себя корпоративное веб-приложение.

В ходе проведенного анализа методологий разработки современных веб-приложений было выявлено, что большинство веб-приложений эволюционируют по своей природе, требуя (частых) изменений контента, функциональности, структуры, навигации, презентации или реализации. Они особенно развиваются с точки зрения их требований и функциональности (нестабильность требований), особенно после того, как система введена в эксплуатацию. В большинстве веб-приложений частота и степень изменений намного выше, чем в традиционных программных приложениях, и во многих приложениях невозможно полностью указать все их требования в начале. Поэтому сам процесс проектирования веб-приложений является важной технической, организационной и управленческой задачей. И является более сложной, чем традиционная разработка программного обеспечения.

Чтобы разобраться в вопросе эффективности проектирования веб-приложений был проведен анализ известных и современных методологий разработки веб-приложений WebML и WSDM. Было выявлено что эти методики, очень сильно сконцентрированы на решении определенных задач при проектировании веб-приложений. В основном прорабатываются вопросы, связанные с решением как спроектировать, навигацию в приложении, дизайн приложения, функциональность, ориентированная на пользователя, созданием модели данных необходимых для отображения и не уделяется такой важной детали как управлением состоянием приложения. Также было доказано что данная проблема возникла не на пустом месте, а из-за использования

определенного архитектурного шаблона MVC и архитектуры построения визуальной части веб-приложения с помощью DOM и HTML. Поэтому на основании такого заключения была разработана технология «Визуальное проектирование хранилища параметров сеанса», которая бы помогла внедрить в разрабатываемое веб-приложения управление состоянием приложения.

Было показано как с помощью разработанной технологии, учитывая современные тенденции разработки веб-приложений, можно на этапе проектирования легко добавить управление состоянием приложения. Технология не призвана полностью отказаться от классических методов разработки, она лишь их дополняет таким важным элементом как проектирование управления состоянием приложения. Также стоит учесть, что если разрабатываемое веб-приложение является небольшим или несложным, то такое внедрение этой технологии может оказаться чересчур сложным и запутанным.

Одной из целей исследования заключалась в проверке целесообразности и работоспособности разработанной технологии «Визуальное проектирование хранилища параметров сеанса». Для этой цели было разработано тестовое приложение с использованием таких библиотек как ReactJS и Redux. Использование этих библиотек было выбрано с целью усилить полученный эффект от эксперимента. Тестовое приложение, которое было реализовано в ходе проведение эксперимента, реализовало некоторые основные функции, характерные для всех веб-приложений. Это были такие функции как поиск в данных, добавление данных и отображение этих данных. Как упоминалось в начале, не существует точных требований к применению выбранной структуры, но, тем не менее, структура должна быть проверена. Поэтому и были отобраны эти функции, которые считаются наиболее подходящими, с тем чтобы можно было подтвердить осуществимость эксперимента и получить правдоподобные данные. Экспериментальная проверка и анализ показал, что даже такие простые приложения нуждаются в такой технике как управление состоянием приложения. Потому как реализация некоторых простых функций может

показаться совсем непростой задачей, не говоря уже о дальнейшей поддержке и масштабировании приложения. Также в ходе анализа было выявлены серьезные ограничения и дополнительные расходы при реализации, если не использовать управление состоянием приложения. Поэтому при разработке нового веб-приложения всегда стоит подумать, чтобы использовать такую вещь, а не отказываться от нее.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Федеральный закон «Об информации, информационных технологиях и защите информации» [Текст] // Федер. закон от 27.07.2006 № 149-ФЗ // Российская газета – 2006 – 29 июля.
2. Приказ Министерства труда и социальной защиты РФ от 18 января 2017 г. № 44н «Об утверждении профессионального стандарта «Разработчик Web и мультимедийных приложений»» [Текст] // Приказ от 18.01.2017 // Российская газета – 2017 – 08 февраля.
3. ГОСТ 28195-89. Оценка качества программных средств. Общие положения. [Текст] – М.: Издательство стандартов, 1989. Переиздано: ИПК Издательство стандартов , 2001 – 31 с.
4. ГОСТ Р ГОСТ 28806-90 Качество программных средств. Термины и определения [Текст] – М.: Стандартинформ, 2005., 36с.
5. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств [Текст] – М.: Стандартинформ, 2011., 76с.
6. ГОСТ Р 52872-2012 Интернет-ресурсы. Требования доступности для инвалидов по зрению [Текст] – М: Стандартинформ, 2014., 28с.
7. ГОСТ Р ИСО 9241-210 – 2012 Человеко-ориентированное проектирование интерактивных систем [Текст] – М.: Стандартинформ, 2013., 36с.
8. ГОСТ Р ИСО 9241-151 – 2014 Руководство по проектированию пользовательских интерфейсов сети Интернет [Текст] – М.: Стандартинформ, 2015., 50с.
9. ГОСТ Р 55241.50 – 2014 Методы обеспечения пригодности использования в человеко-ориентированном проектировании [Текст] – М.: Стандартинформ, 2015., 42с.
10. ГОСТ Р 8.654-2015 Государственная система обеспечения единства измерений (ГСИ). Требования к программному обеспечению средств измерений [Текст] – М.: Стандартинформ, 2015., 12с.

11. Азбука клиента Модель заказной веб-разработки CMS – Magazine [Электронные ресурс]// Статья – Режим доступа: http://www.cmsmagazine.ru/library/items/management/modeli-zakaznoj-veb-razrabotki/

12. Итан Маркотт. Отзывчивый веб-дизайн – Responsive Web Design. [Текст] – М.: Манн, Иванов и Фербер, 2012 – 159 с.

13. Лебедев, С. А. Философия науки. Словарь основных терминов [Текст] / Лебедев С.А – М.: Академический проект, 2004 – 320 С.

14. Мацяшек, Лешек А., Анализ и проектирование информационных систем с помощью UML 2.0, 3-е изд. [Текст]/ Пер. с англ – М.: ООО «И.Д. Вильямс», 2016 – 816 с.

15. Садовский, В.Н. Исследования по общей теории систем [Текст] / Садовский В.Н., Юдин Э.Г – М.: Прогресс, 1969 – 400с.

16. Сысолетин, Е. Г. Проектирование интернет-приложений: учеб.-метод. пособие [Текст]/ Е. Г. Сысолетин, С. Д. Ростунцев. — Екатеринбург : Изд-во Урал.ун-та, 2015 – 92 с.

17. Тузовский, А.Ф. Проектирование и разработка web-приложений : учеб. пособие для академического бакалавриата [Текст]/ А. Ф. Тузовский – М. : Издательство Юрайт, 2016 – 218 с.

18. Bass, L., Clements, P., Kazman, R.: Software Architecture in Preactice, 2md edn. Addison-Wsesley, Reading (2003)

19. [Beta] How do you design ? [Электронный ресурс] //Статья – Режим доступа: http://www.dubberly.com/wp-content/uploads/2008/06/ddo_designprocess.pdf

20. Brambilla, M., Fraternali, P.: Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML. Morgan Kaufmann (2014).

21. Cooper, A., - About Face: The Essentials of Interaction Design 4th Edition. 2014

22. Daniel, F., Matera, M.: Mashups - Concepts, Models and Architectures. Springer (2014) DOI:10.1007/978-3-642-55049-2.

23. de Lange P., Nicolaescu P., Klamma R., Jarke M. (2017) Engineering Web Applications Using Real-Time Collaborative Modeling. In: Gutwin C., Ochoa S., Vassileva J., Inoue T. (eds) Collaboration and Technology. CRIWG 2017. Lecture Notes in Computer Science, vol 10391. Springer, Cham

24. Firmenich, D. et al.: CrowdMock: an approach for defining and evolving web augmentation requirements. Requir. Eng. 1–29 (2016) DOI:10.1007/s00766-016-0257-3.

25. Griffin, A., Design thinking: new product development essentials from the PDMA / edited by Michael G. Luchs, K. Scott Swan, Abbie Griffin.

26. Karzan Wakil, Dayang N. A. Jawawi, Comparison between Web Engineering Methods to Develop Multi Web Applications, JSW 2017 Vol.12(10): 783-793 ISSN: 1796-217X

27. Kienle, H.M., Distante, D.: Evolution of Web Systems. In: Mens, T. et al. (eds.) Evolving Software Systems. pp. 201–228 Springer Berlin Heidelberg, Berlin, Heidelberg (2014) DOI:10.1007/978-3-642-45398-4_7.

28. Keaveney, S., Keaveney, S., COST ESTIMATION IN AGILE DEVELOPMENT PROJECTS., Proceedings of the Fourteenth European Conference on Information Systems, ECIS 2006, Göteborg, Sweden, 2006

29. Manju K Mathai, Rakhi Venugopal, Dr. John T Abraham, Software Engineering Process in Web Application Development – IOSR Journal of Computer Engineering (IOSR-JCE), Volume 17, Issue 1, Ver. V (Jan – Feb. 2015)

30. Martínez, Y. et al.: Empirical study on the maintainability of Web applications: Model-driven Engineering vs Code-centric. Empir. Softw. Eng. 19, 6, 1887–1920 (2013) DOI:10.1007/s10664-013-9269-5.

31. Milicevic, A., Jackson, D., Gligoric, M., Marinov, D.: Model-based, event-driven programming paradigm for interactive web applications. In: Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms and Reflections on Programming and Software, pp. 17–36 (2013)

32. Native, web or hybrid mobile-app development [Электронный ресурс] //Технический документ – Режим доступа: ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf

33. Norrie, M.C. et al.: The Forgotten Many? A Survey of Modern Web Development Practices. In: 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings. pp. 290–307 Springer International Publishing (2014) DOI:10.1007/978-3-319-08245-5_17.

34. P. ZELENKA, Modern methods of web applications analysis and design – AGRIC. ECON – CZECH, 52, 2006 (4): 152–154

35. Wirfs-Brock, R. 1993. «Designed Scenarios: Making the Case for a Use Case Frameworks.» Smalltalk Report, November-December

36. Solomon Antony, A review and analysis of technologies for developing web applications. Conference: ITERA 2012

37. T. J. Shelford and G. A. Remillard, «Real web project management: Case studies and best practices from the trenches,» Boston, Addison-Wesley Professional, 2002.

38. The Guardian: What is front-end development? [Электронный ресурс]. URL:https://www.theguardian.com/help/insideguardian/2009/sep/28/blogpst

39. Wakil K, Jawawi D N A, Combining web-engineering methods to cover lifecycle – COMPUTER MODELLING & NEW TECHNOLOGIES 2017 21(1) 20-27

40. Web Engineering: Introduction and Perspectives, Part of the Lecture Notes in Computer Science book series (LNCS, volume 2016)

41. WebModels, 2006. WebRatio Tool Suite. [Электронный ресурс]. //Технический документ – Режим доступа: http://www.webratio.com.

42. What are the exact demerits of two-way data binding? [Электронный ресурс]. Hashnode; 2015. //Технический документ – Режим доступа: https://hashnode.com/post/what-are-the- exact-demerits-of-two-way-data-binding-ciibz8fnq01f8j3xthmjjs6di

43. Wohlin, C., Host, M., and Henningsson, K., 2005, Empirical research methods in Web and software Engineering. In Web Engineering, E. Mendes and N. Mosley, eds., Springer, New York, pp. 409–430.