Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

СРАВНИТЕЛЬНЫЙ АНАЛИЗ CSS-МЕТОДОЛОГИЙ

Автор_ Бабахин Антон Валерьевич _____ _____

Направление подготовки (специальность) 09.04.02
Информационные системы и технологии

Квалификация магистр

Руководитель ВКР_ Государев И.Б., к.педагог.н., доцент _ _____

К защите допустить

Руководитель ОП_ Шуклин Д.А., к.педагог.н., доцент _ _____

«_____»_____ 2019 г.

Санкт-Петербург, 2019 г.

Студент **Бабахин А.В.** Группа Р4263 Факультет ПИиКТ

Направленность (профиль), специализация 09.04.02 Информационные системы и технологии, Веб-технологии

ВКР принята «27» мая 2019 г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты «4» июня 2019 г.

Секретарь ГЭК Калимуллина А.Ю. _____

Листов хранения 78

Демонстрационных материалов/Чертежей хранения 16

Оглавление

Введение

Активное развитие сети интернет и интернет-технологий стало основным драйвером роста многих отраслей экономики. Интернет-сервисы позволяют решать, не выходя из дома, огромное количество задач, которые раньше требовали личное присутствие человека.

Веб-ресурсы активно развиваются, модернизируются, появляется огромное количество новых функций. Сложность взаимодействия с подобными ресурсами также увеличивается. Пользовательский интерфейс играет большую роль при взаимодействии пользователя с сайтом, в значительной степени влияет на конверсию и удовлетворённость посетителя интернет-ресурсом. В конечном итоге, современный и удобный веб-интерфейс приносит компании дополнительный доход, увеличивая размер лояльной аудитории.

Сегодня один из основных универсальных подходов к разработке современных веб-интерфейсов – использование CSS-методологий. Методологии содержат правила по организации всего проекта, позволяют разработчикам эффективно взаимодействовать между собой.

Актуальность темы исследования. На сегодняшний день существует большое количество разнообразных CSS-методологий вёрстки, обладающих своими положительными и отрицательными сторонами. Самыми популярными представителями являются методологии OOCSS, БЭМ, SMACSS и Atomic CSS.

Правильный выбор методологии в каждом конкретном проекте значительно влияет на сроки его реализации, трудовые затраты и стоимость будущей поддержки. Таким образом, существует необходимость понимания, какую технологию использовать в определённых условиях.

Проблему применения CSS-методологий в проектах изучали российские (Е. А. Селиванова, И. В. Левина, В. А. Захаров, А. А. Охрименко) и зарубежные (L. Watts, P. Catanzariti, P. Sikora, M. Leino) исследователи. Однако в данных научных работах были рассмотрены лишь отдельные вопросы использования

методологий вёрстки, не были определены критерии сравнения, отсутствовал подробный сравнительный анализ популярных методологий.

На основе обзора и сравнения существующих CSS-методологий имеется возможность выявить их основные положительные и отрицательные стороны, определить рекомендации по выбору методологий, а также синтезировать новую методологию вёрстки, обладающую конкурентными преимуществами.

Таким образом, учитывая сказанное ранее, можно сделать вывод об актуальности темы данной научной работы. Результаты её выполнения дают возможность аргументированного выбора CSS-методологий при реализации реальных проектов.

Степень теоретической разработанности темы. В научных работах [7; 11; 14; 17; 19; 22; 24] рассмотрены отдельные вопросы использования методологий вёрстки, сравниваются не более двух методологий между собой, не определены критерии их сравнения, а также отсутствует подробный сравнительный анализ популярных CSS-методологий.

Цель выпускной квалификационной работы – выполнить сравнительный анализ популярных CSS-методологий вёрстки, на основе которого разработать новую методологию, обладающую конкурентными преимуществами.

Задачи выпускной квалификационной работы:

- определить основные проблемы разработки веб-интерфейсов;
- проанализировать научные работы по применению CSS-методологий;
- провести обзор популярных CSS-методологий, существующих на сегодняшний день;
- на основе закономерностей и общих черт определить критерии сравнения методологий;
- провести сравнение CSS-методологий по критериям сравнения;
- определить рекомендации по выбору CSS-методологий в конкретных условиях;
- определить положительные и отрицательные стороны каждой методологии;
- на основе выявленных особенностей и положительных сторон синтезировать новую CSS-методологию;
- определить технологии реализации посадочной страницы согласно новой методологии;
- разработать необходимые файлы проекта и осуществить сборку;

- разместить полученную страницу в сети интернет;
- определить соответствие полученных результатов теоретическим ожиданиям.

Объект исследования – CSS-методологии вёрстки.

Предмет исследования – правила организации кода и файловой структуры CSS-методологий вёрстки.

Область исследования – проектирование и разработка веб-ресурсов.

Теоретической основой проводимого исследования выступили работы российских (А. С. Дорогина, О. Н. Сафонова, С. А. Белоусова, Ю. И. Рогозов, В. А. Захаров, А. А. Охрименко, Е. А. Селиванова, И. В. Левина, Ю. К. Шакирова, А. Е. Маденова, Н. К. Савченко, Г. Б. Абилдаева, В. Э. Резанович) и зарубежных (L. Watts, P. Sikora, M. Leino, P. Catanzariti, M. A. Perna, J. Snook, N. Gallagher, N. Sullivan, L. Lazaris, T. Koblentz) исследователей, посвящённые анализу CSS-методологий вёрстки.

Основной **информационной базой исследования** послужили следующие источники: elibrary.ru, cyberleninka.ru, scholar.google.com, github.com, xakep.ru, habr.com, medium.com, googleblog.com, itchief.ru.

Для решения задач исследования использовался комплекс теоретических и эмпирических взаимодополняющих **методов исследования**, среди которых ведущими были следующие методы: анализ, сравнительный анализ, синтез, обобщение.

Теоретическая и прикладная значимость. Теоретическая значимость работы обусловлена тем, что на сегодняшний день существует большое количество различных CSS-методологий, но при этом не представлены научные работы с рекомендациями по выбору методологий в проектах, не определены критерии их сравнения, а также отсутствует сравнительный анализ. В рамках данной выпускной квалификационной работы был проведён обзор популярных CSS-методологий вёрстки, были определены критерии сравнения и выполнен сравнительный анализ. Результаты данного сравнительного анализа были

опубликованы, что может стать основой к аргументированному выбору методологий в реальных проектах.

На основе проведённого исследования были определены требования к новой CSS-методологии, образованной в результате синтеза подходов БЭМ, OOCSS и SMACSS. Данная методология поддерживает динамическую смену тем оформления страницы, адаптивную вёрстку, оптимизацию под различные браузеры и многократное использование компонентов. С использованием данной методологии вёрстки была создана и размещена в сети интернет специальная посадочная страница, что стало практическим результатом выполнения данной научной работы.

Научная новизна данного исследования заключается в определении критериев сравнения CSS-методологий, а также формировании таблицы сравнения популярных методологий вёрстки.

Положения, выносимые на защиту:

- не существует универсальной методологии вёрстки, для каждого проекта необходимо выбирать методологию индивидуально;
- сформулированные рекомендации по выбору методологий позволяют повысить эффективность разработки типичного веб-проекта;
- предложенная новая CSS-методология объединяет положительные стороны БЭМ, OOCSS и SMACSS.

Апробация результатов исследования. Результаты данного научного исследования обсуждались на следующих конференциях:

- XLVII научная и учебно-методическая конференция Университета ИТМО;
- XLVIII научная и учебно-методическая конференция Университета ИТМО.

Были опубликованы следующие статьи:

- Бабахин А.В. Сравнительный анализ подходов к разработке веб-интерфейсов // Альманах научных работ молодых ученых Университета ИТМО – 2018. – Т. 7. – С. 54-56;
- Бабахин А.В., Государев И.Б. Сравнительный анализ CSS-методологий // Альманах научных работ молодых ученых Университета ИТМО (отдана в печать) – 2019.

ГЛАВА 1. ПОДХОДЫ К РЕШЕНИЮ РАСПРОСТРАНЁННЫХ ПРОБЛЕМ ВЕБ-РАЗРАБОТКИ

В данной главе рассматриваются основные проблемы, возникающие при разработке веб-интерфейсов, и способы их решения. Анализируются научные работы по применению CSS-методологий и выполняется их подробный обзор. На основе общих закономерностей определяются критерии сравнения разных методологий и выполняется сравнительный анализ.

1.1 Особенности разработки пользовательских веб-интерфейсов

В настоящее время происходит стремительное развитие интернет-технологий. С каждым днём количество пользователей глобальной сети увеличивается. Задачи, решения которых также связаны с использованием сети интернет, становятся всё сложнее, они проникают в новые сферы деятельности человека: науку, производство, сферу услуг, развлечения и многое другое. На сегодняшний день интернет стал неотъемлемой частью жизни современного человека [1].

Во время взаимодействия пользователя с интернет-ресурсом существенную роль играет веб-интерфейс. Именно интерфейс позволяет пользоваться всеми возможностями веб-ориентированной информационной системы (ИС) [2]. Но ИС бывают различного масштаба и сложности: от статических сайтов, предоставляющих информацию о какой-либо компании или персоне, до крупных международных веб-сервисов, хранящих и обрабатывающих петабайты данных. Также отличаются и способы взаимодействия пользователя с интернет-ресурсом. Например, одни сайты лишь отображают информацию, не позволяя с ней активно взаимодействовать. Другие сайты позволяют выводить информацию на основе пользовательских запросов, имеют нетривиальную логику и множество функций.

Таким образом, в ближайшем будущем сложность работы веб-интерфейсов будет только увеличиваться, количество доступных пользователю функций будет расти. Благодаря технологии отправки запросов от браузера пользователя к серверу – AJAX, появляется возможность получения всей необходимой информации на одной странице. Ввиду этого, объём кода, используемого на клиентской стороне, становится значительным, что затрудняет дальнейшую разработку.

На сегодняшний день существует общепринятый набор технологий для разработки современных пользовательских интерфейсов: HTML – язык разметки, позволяющий располагать элементы на веб-странице для последующего вывода; CSS – язык описания стилей; JavaScript – язык программирования, позволяющий динамически взаимодействовать с содержимым страницы.

Данные технологии стандартизированы и имеют определённые варианты использования. Однако существуют проблемы, связанные с их реализацией в конкретных условиях [3].

Одна из проблем связана с невозможностью знать условия исполнения программного кода заранее, на стороне пользователя. Для отображения контента веб-страницы используются специальные программы – браузеры. Сегодня существует огромное количество различных браузеров, и, как в других областях, есть наиболее популярные продукты. Одним из самых часто используемых браузеров является Google Chrome от компании Google. Конкурентами данной программы являются Safari и Firefox от компаний Apple и Mozilla соответственно. Для нашей страны также следует упомянуть такие браузеры как Opera, Яндекс.Браузер, Edge и Internet Explorer.

Основная проблема заключается в том, что каждый браузер имеет свои особенности. Стандартные элементы языка HTML, такие как input, select, button отображаются по-разному. Они имеют отличия в размерах, границах, цветах и отступах. Поэтому, приходится применять специальные методики, позволяющие добиться одинакового отображения элементов в различных средах. В некоторых

случаях целесообразно отказываться от использования стандартных элементов в пользу аналогов, написанных самими разработчиками сайта.

Также браузеры разных производителей отличаются друг от друга с функциональной точки зрения. Современный стандарт HTML5 содержит описание большого количества дополнительных тегов, которые отсутствуют в стандарте HTML четвёртой версии. Несмотря на то, что HTML5 был опубликован в 2014 году, на сегодняшний день многие браузеры не поддерживают определённые функциональные возможности, предусмотренные стандартом. Например, тег input с типом color не могут отобразить браузеры Safari для операционной системы iOS и Microsoft Internet Explorer.

Похожая ситуация наблюдается и с современным стандартом описания стилей веб-страниц – CSS3. Данный стандарт привнёс значительное количество возможностей по улучшению внешнего вида страниц сайта, а также созданию анимации и повышению производительности. На сегодняшний момент есть ряд возможностей, не реализованных в современных браузерах. Также ряд свойств поддерживается только с указанием префикса конкретного браузера [4].

Другой проблемой, связанной с исполнением кода на клиентской стороне, является большое количество форм-факторов устройств, используемых для отображения. Сегодня пользователи просматривают сайты сети интернет как с мобильных телефонов, имеющих небольшие размеры дисплея, так и со Smart-телевизоров, имеющих большую диагональ. Это создаёт определённый набор требований как к оформлению веб-страниц, так и к их содержанию. Существует несколько способов решения данной проблемы. Наиболее распространёнными способами являются создание отдельных страниц для использования на каждом конкретном виде устройств, а также разработка страниц, автоматически адаптирующихся под размер рабочей области – адаптивная вёрстка.

Вторая проблема использования технологий веб-разработки заключается в их применении конкретными разработчиками в конкретных проектах.

Многие задачи не имеют единого общепринятого решения, а могут быть реализованы двумя или большим количеством различных способов. Выбор конкретного варианта реализации зависит непосредственно от разработчика.

Это не является критичным до тех пор, пока с проектом не начинает работать большая команда. Ввиду сильного роста сектора информационных технологий, включая область веб-разработки, на сегодняшний день штат крупных интернет-компаний может насчитывать десятки тысяч человек. Над одним проектом могут одновременно работать сотни разработчиков из разных стран мира.

В данном случае проблема заключается в том, что у каждого разработчика свои способы решения типичных задач, свой стиль написания кода. Сотрудник компании может использовать библиотеки и фреймворки, которые другие разработчики не применяют. Для успешного создания продукта программисты должны разбираться в коде своих коллег, иметь возможность проводить аудит кода и его рефакторинг.

Другой важной частью эффективного процесса разработки веб-приложений является скорость поиска необходимых компонентов программы во всём объёме кода. Каждый разработчик должен иметь возможность оперативно найти требуемую часть программы. Для этого требуется иметь строго определённую архитектуру проекта. Помимо архитектуры, в проекте должны использоваться только выбранные необходимые инструменты и технологии. Недопустимо использование нескольких аналогичных программных средств в рамках работы над одним проектом ввиду разных предпочтений разработчиков.

Так как прогресс не стоит на месте, программы для просмотра страниц постоянно совершенствуются. В связи с этим, определённые браузеры и их версии утрачивают популярность. Как было определено ранее, все браузеры имеют свои особенности. Таким образом, написание качественного кода напрямую зависит от списка поддерживаемых программ. Поэтому, для эффективной работы нескольких сотрудников над одним проектом необходима единая политика поддержки старых браузеров.

Одна из главных проблем при разработке большого проекта в крупной компании – сложность внедрения новых функций и компонентов.

Чем больше в проекте используется различных компонентов и библиотек, тем сложнее вносить какие-либо изменения и осуществлять поддержку. Также увеличивается время, требуемое для внесения соответствующих изменений.

Когда увеличивается число компонентов системы, также увеличивается вероятность конфликтов между ними. Например, разные элементы HTML-страниц могут иметь одинаковые имена классов, описывающих их стили. В данном случае все конфликтующие элементы будут выглядеть с искажениями, не предусмотренными разработчиками. Это не является большой проблемой, если сотрудники имеют возможность быстро обнаружить и исправить данную ошибку. Но во многих случаях это невозможно. Часто разработчики вносят изменения, которые затрагивают несколько страниц сразу. Проблема может проявиться на странице, которая не была протестирована после изменений. Также сама необходимость тестировать множество страниц перед публикацией является отрицательной стороной подобного метода. В случае работы большой команды это делать практически невозможно.

Таким образом, для создания эффективного процесса разработки интерфейсов веб-сервисов, избегания появления распространённых проблем при работе над проектом нескольких сотрудников, должны быть решены следующие задачи:

- возможность быстрого внесения изменений и поиска участков кода;
- получение самодокументированного кода;
- уменьшение взаимного влияния компонентов друг на друга;
- размещение нескольких сущностей в одном HTML-документе без необходимости копирования частей кода;
- обеспечение одинакового отображения страниц во всех браузерах;
- возможность реализации проекта любой сложности;
- обеспечение единого стилевого оформления однотипных элементов;
- разделение кода на модули;
- многократное использование кода в разных проектах;
- организация единого набора инструментов и технологий при работе над одним проектом нескольких разработчиков;

- быстрое подключение новых сотрудников к проекту;
- возможность замены инструментов и технологий в будущем на более современные;
- обеспечение корректного отображения страниц на разных устройствах с разными размерами дисплея.

Один из самых эффективных и популярных подходов к решению данных задач на сегодняшний день является использование CSS-методологий вёрстки при разработке проекта.

Сегодня существует большое количество различных CSS-методологий, созданных разными компаниями и независимыми разработчиками. Поэтому, необходимо провести их сравнительный анализ для выявления наиболее эффективных в различных условиях использования, основываясь на положительных и отрицательных сторонах этих методологий.

1.2 Обзор популярных CSS-методологий вёрстки

CSS-методологиями являются наборы правил, предъявляемых к структуре веб-проектов, принципам именования классов и использования тегов, инструментам реализации и сборки, а также другим важным аспектам разработки.

Самыми популярными CSS-методологиями вёрстки сегодня являются:

- БЭМ (BEM);
- OOCSS;
- SMACSS;
- Atomic CSS.

1.2.1 БЭМ

Методология БЭМ (Блок, Элемент, Модификатор) была разработана и поддерживается компанией Яндекс. Основой методологии является принцип

модульности, заключающийся в многократном использовании одних и тех же компонентов. В данном случае, компонентами являются независимые блоки интерфейса. Данный подход позволяет значительно сократить количество кода, который должен быть написан разработчиком [5].

Один из ключевых элементов методологии BEM – блок. Блок представляет собой компонент страницы, независимый от других компонентов. Именно блок может быть многократно использован, поэтому определяется при помощи атрибута class. Использовать id или селекторы по тегам для блоков крайне не рекомендуется. Название класса определяется его смысловой нагрузкой. Блоки могут быть вложены друг в друга, что позволяет создавать сложную структуру веб-страницы.

Элемент – одна из внутренних частей блока. Внутри одного блока может быть неограниченное количество элементов, которые непосредственно с ним связаны и не могут существовать сами по себе. Например, внутри блока header могут находиться такие элементы, как logo, title или contacts. Элемент title запрещено использовать в других частях веб-страницы вне блока header. Согласно правилам именования методологии БЭМ, класс элемента определяется по формуле: «название-блока__название-элемента». Таким образом, имя элемента отделяется двумя знаками подчёркивания. Аналогично блокам, элементы могут быть вложены друг в друга, но имя блока у всех этих элементов будет одинаковым, что создаёт пространство имён.

Для определения специфического внешнего вида, поведения, состояния блока или элемента применяются модификаторы. Они не могут существовать сами по себе, а лишь модифицируют существующий компонент. Имя модификатора, согласно методологии БЭМ, определяется по формуле: «название-блока_название-модификатора» или «название-блока__название-элемента_название-модификатора». То есть, название модификатора отделяется одним знаком подчёркивания. Например, если блок button (кнопка) имеет больший, по сравнению с другими кнопками на странице, размер, то название

класса будет иметь вид button_size_xl. Если кнопка не является активной, целесообразно дать такому блоку класс button_disabled.

Принятая файловая структура проектов, основанных на методологии БЭМ, значительно отличается от стандартной, используемой в большинстве веб-проектов. Вместо отдельных папок под файлы скриптов и стилей, папки создаются под конкретный блок, элемент и модификатор, подчиняясь иерархии, аналогичной использованию на странице [6]. Названия директорий элементов начинаются с двух знаков подчёркивания. Названия директорий модификаторов начинаются с одного знака подчёркивания. Данный подход позволяет подключать к интернет-странице только необходимые компоненты.

Важно отметить, что данная файловая структура не является обязательной. Имеется возможность использовать любую альтернативную структуру проекта.

Методология BEM не ограничивает в выборе используемых CSS-препроцессоров. По умолчанию поддерживаются Stylus и Sass. Использование препроцессоров позволяет сократить исходный код стилей и уменьшить вероятность совершения ошибки. В методологии БЭМ активно используются ссылки на родительский селектор. Пример данной возможности представлен на рисунке 1.

```
.header {
        height: 50px;
        &__title {
                font-size: 18px;
        }
}
```

Рисунок 1 – Ссылка на родителя

После компиляции в CSS появится элемент header__title. В случае переименования блока header автоматически будет осуществлено переименование соответствующих элементов.

В каждом БЭМ-проекте есть специальные директории – уровни переопределения. Данные директории содержат наборы блоков, специфичных для каждого уровня. Например, блоки, элементы и модификаторы, единые для всех версий сайта, содержатся в директории common. Другие уровни переопределения

могут содержать компоненты, описывающие специальные версии страниц сайта, менять общие блоки сайта, добавлять темы оформления и многое другое.

Анализ многих методологий вёрстки, в том числе BEM, был произведён на конференции 4front в качестве доклада «Методологии верстки: БЭМ, AMCSS, OOCSS, Atomic CSS, OPOR, MCSS, SMACSS, FUN, DoCSSa» [7].

Было выявлено, что методологии позволяют решить такие проблемы, как разный стиль кода у разработчиков, разные подходы к организации вёрстки, большое количество повторных реализаций, трудности рефакторинга кода.

Определено, что методология БЭМ на сегодняшний день является наиболее распространённой и популярной. В значительной степени это связано с тем, что она разработана известной российской интернет-компанией Яндекс [6] и широко применяется в собственных продуктах.

В качестве альтернативы существующим CSS-методологиям предлагается методология, объединяющая БЭМ и SMACSS. Также возможно использование модификации БЭМ с собственными разделителями для блоков, элементов и модификаторов.

Однако авторами не были определены критерии для сравнения различных методологий между собой. Без этого невозможно проведение всеобъемлющего исследования разных методологий вёрстки и определение наиболее подходящих в конкретных условиях.

1.2.2 SMACSS

В основе методологии SMACSS, в отличие от BEM, лежит разделение стилей на категории. Всего предусмотрено пять категорий:

- базовые стили (Base);
- стили макета (Layout);
- стили модулей (Module);
- стили состояния (State);

- стили темы (Theme).

Каждая категория предназначена для определённых вариантов использования, но в процессе разработки допускается частичное их смешение. Главная цель подхода – сделать код лёгким для анализа и поддержки [8]. Применяются особые техники для понимания, к какой категории относится каждое CSS-правило.

Базовые правила отвечают за исходное состояние каждой страницы на сайте, то, как она должна выглядеть в любом браузере. Поэтому, в данной категории используются селекторы элементов, селекторы дочерних элементов и псевдо-классов. Наиболее часто встречаются определения внешних и внутренних отступов тегов body, form, а также задание цветов и способов отображения ссылок. Для решения подобной задачи могут использоваться библиотеки normalize.css [9] или Reset CSS [10]. Учитывая данные особенности, базовые стили не должны использовать селекторы по классам или ID, а также содержать ключевое слово «!important».

Стили макета определяют внешний вид больших элементов страницы, являющихся контейнерами для других элементов. В качестве данных контейнеров часто выступают шапка (header), подвал (footer), колонки и меню сайта. Их отличительной чертой является то, что они встречаются на каждой странице лишь один раз. Поэтому, для таких элементов целесообразно использовать селекторы по ID. Зачастую требуется создать нестандартный вид страницы, ввиду определённых факторов. Для подобных случаев селекторы по ID выступают в качестве дочерних элементов, а у родительских классов в названии присутствует префикс с указанием на принадлежность к стилям макета. Например, если на странице требуется фиксированное меню, CSS-селектор может иметь вид, представленный на рисунке 2.

```
.l-fixed #sidebar
```

Рисунок 2 – Селектор для зафиксированного меню

В данном случае, в качестве префикса выступает «l-».

Стили модулей, аналогично блокам методологии БЭМ, применяются для описания повторяющихся компонентов веб-страницы. Эти компоненты находятся в контейнерах макета, рассмотренных ранее. Так как они многократно повторяются на одной странице, то описание стилей происходит при помощи классов. Для модулей, находящихся внутри других модулей, применяются селекторы потомков и другие современные возможности CSS. Подобно методологии БЭМ, внутренние элементы модулей могут иметь префикс с названием модуля, что создаёт пространство имён и уменьшает вероятность совершения ошибки.

Стили состояния, аналогично модификаторам БЭМ, меняют вид существующих элементов в зависимости от внешних факторов. Элементами могут быть как модули, так и макеты. Согласно методологии SMACSS, названия классов, описывающих состояние, начинаются с префикса «is-» [11]. Например, наиболее встречающимися состояниями являются: is-active, is-collapsed, is-hidden. Так как стили состояния должны перевешивать стандартные стили модулей и макетов, то допускается применение ключевого слова «!important».

Стили темы отвечают за внешний облик элементов страниц сайта. Наиболее часто в данной категории стилей задают цвета, фоновые изображения, радиусы скругления, тени и другие косметические настройки. Благодаря данному подходу имеется возможность создавать несколько различных тем оформления на выбор посетителя сайта, менять дизайн в зависимости от страны или языка. Стили темы могут применяться к любым другим категориям стилей, например, к базовым или стилям модулей. Для наглядного отличия данных стилей от других, методология SMACSS определяет специальный префикс «theme-» в названиях.

Методология SMACSS позволяет использовать препроцессоры для увеличения скорости написания стилей и объединения их в единый файл. Но, ввиду особой структуры проектов и разделения на категории, полноценное использование вложенных селекторов и ссылок на родителя не представляется возможным, что активно применяется в проектах по методологии БЭМ.

Автор P. Catanzariti в статье «BEM and SMACSS: Advice From Developers Who’ve Been There» [12] сделал сравнение двух методологий – BEM и SMACSS, основанное на опросе разработчиков, работавших как с первой, так и со второй технологиями.

Определено, что БЭМ-методология подходит к использованию как в крупных, так и в небольших проектах. Таким образом, код страницы получается чистым и понятным, не конфликтует с кодом на других страницах. Это является положительной стороной методологии. Отрицательной стороной можно назвать слишком длинные названия классов.

Другой специалист в качестве положительной стороны БЭМ назвал отсутствие необходимости использовать вложенные селекторы. Названия классов получаются понятными для других разработчиков. Методология совместима с препроцессором Sass.

Ещё одним важным достоинством BEM является возможность эффективного взаимодействия нескольких разработчиков, работающих над одним проектом. Сайты, на которых использовалась данная методология, могут поддерживаться много лет без ущерба качеству кода. БЭМ снижает вероятность появления ошибки, связанной с конфликтами классов на разных страницах.

Разработчик J. Hunt отмечает, что методология хорошо подходит для инкапсулирующих компонентов, таких как веб-компоненты. Angular, React, Polymer-фреймворки применять значительно проще в паре с BEM.

При сравнении методологий SMACSS и BEM, разработчик C. Wright отмечает, что SMACSS – более читаемая и понятная, чем остальные методологии. Добавление префиксов позволяет сообщить, на каком уровне находится нужный HTML-класс. Обучение разработчиков правилам использования SMACSS занимает меньше времени по сравнению с другими методологиями.

Также имеется возможность использовать одновременно обе эти методологии. Например, использовать три уровня SMACSS – base, layout и modules, а также принцип именования элементов BEM.

Таким образом, автор подводит итог, что самый правильный подход к организации разработки интерфейса – гибрид методологий БЭМ и SMACSS. Необходимо совместить положительные стороны методологий. В таком случае организация процесса разработки вырастет, не будет необходимости выбирать одну из методологий.

1.2.3 Object Oriented CSS

Методология Object Oriented CSS (OOCSS) базируется на том же принципе, что и методология БЭМ – максимальное повторное использование компонентов. Об этом говорит название, ведь объектно-ориентированный подход заключается в создании независимых элементов, упрощающих добавление и поддержку кода [13].

В основе OOCSS лежат два главных принципа – отделение структуры от оболочек, а также отделение контейнеров от содержимого.

Первый принцип заключается в определении повторяющихся визуальных элементов (оболочек), таких как фоновый цвет и стиль границ, которые можно применять к различным объектам для достижения визуального разнообразия без написания большого количества кода. Этими элементами могут быть фирменные цвета сайта, радиус закругления границ, тени или внешние отступы [14]. Под «структурами» понимаются компоненты сайта, многократно используемые на странице. Таким образом, оболочки определяют внешний облик структур.

Данные особенности свидетельствуют, что для названия объектов и их компонентов используются классы, а не уникальные идентификаторы. Это позволяет делать код гибким и модульным.

Например, если задать оболочку, определяющую оформление элементов сайта, как показано на рисунке 3, то её можно применять к различным компонентам.

```
.skin {
        border: 1px solid #999;
        border-radius: 5px;
        background-color: #f6f6f6;
}
```

Рисунок 3 – Оболочка методологии OOCSS

Данный подход значительно сокращает кодовую базу стилей и увеличивает скорость загрузки.

Второй принцип – отделение контейнеров от содержимого – определяет, что не рекомендуется использование селекторов дочерних элементов и селекторов-потомков без крайней необходимости. Данные возможности языка CSS создают зависимые от расположения на странице компоненты, что увеличивает связанность кода [15]. Связанный код не позволяет добиться модульности и усложняет рефакторинг.

В случае отказа от использования селекторов-потомков и селекторов дочерних элементов, имеется возможность помещать объекты в любую часть страницы с одинаковым отображением. Также это позволяет избавиться от лишнего дублирования кода и переопределения существующих свойств.

Учитывая данные факторы, определено, что, по аналогии с первым принципом методологии OOCSS, элементы должны задаваться при помощи классов, без применения селекторов по тегам. Использование ключевого слова «!important» также ограничивается.

Например, в случае создания заголовка на странице при помощи кода, представленного на рисунке 4, возникает проблема, что использовать его вне контейнера main невозможно.

```
.main h1 {
        font-family: Arial;
        font-size: 24px;
        font-weight: bold;
        margin: 20px 0px;
        color: #333;
}
```

Рисунок 4 – Стили заголовка

Таким образом, если необходимо создать на странице второй заголовок, придётся либо дублировать, либо дополнять существующий CSS-код.

Поэтому, для заголовков целесообразно использовать класс title, как показано на рисунке 5.

```
.title {
        font-family: Arial;
        font-size: 24px;
        font-weight: bold;
        margin: 20px 0px;
        color: #333;
}
```

Рисунок 5 – Класс title

Данный подход имеет большие сходства с методологией БЭМ, но также существуют и различия. Например, ввиду того, что методология OOCSS не предусматривает наличия пространства имён, имеется вероятность создать класс с названием уже существующего класса. В данном случае возникнет непредвиденный вариант отображения элемента на странице, но, впрочем, подобная ошибка быстро выявляется.

На сегодняшний день существует большое количество готовых компонентов страницы, основанных на методологии OOCSS [16]. Имеется возможность их кастомизации под существующий проект. Также не ограничивается применение CSS-препроцессоров для увеличения скорости разработки.

В работе В. А. Захарова «Современные подходы к верстке веб-проектов» [17] рассматривается несколько подходов к вёрстке веб-страниц.

Среди них, например, представлен бессистемный подход – метод написания кода без осмысления, либо с понятным только автору смыслом. Отмечается, что разрабатывать шаблоны страниц с таким подходом можно достаточно быстро, для этого не требуется высококвалифицированный специалист. Но дальнейшая поддержка кода становится практически невозможной как автору этого кода, так и сторонним разработчикам.

Делается вывод о необходимости структуризации кода и созданию набора правил. Таким образом, делается вывод о необходимости использования CSS-методологий.

В работе автором рассмотрены методологии OOCSS [13] и БЭМ [6]. Отмечается, что OOCSS позволяет максимально оптимизировать код для браузеров. Также, над проектом может работать большая команда разработчиков. В качестве недостатков метода отмечается, что при слишком большом количестве объектов в проекте можно забыть, какие объекты уже были созданы, и сделать их дубликаты. Помимо этого, в методологии OOCSS отсутствуют какие-либо правила по именованию блоков, что усложняет работу с исходным кодом.

БЭМ методология подразумевает простую работу с кодом. При поддержке проекта часто возникают задачи по перемещению или модификации блоков. БЭМ позволяет выполнить подобные изменения за очень короткий срок.

Таким образом, автор делает вывод о том, что методология БЭМ является самой актуальной методологией вёрстки, существующей на сегодняшний день. Она позволяет поддерживать проекты на протяжении всего жизненного цикла, снижать к минимуму лишнюю работу разработчиков.

В книге «Professional CSS3» автора P. Sikora [18] рассматриваются распространённые CSS-методологии:

- SMACSS;
- OOCSS;
- BEM.

Отмечается, что SMACSS – хорошая методология, так как имеет правила для определения базовых HTML-элементов, позволяет описывать состояния элементов, разрешено использование ID для базовых элементов.

Методология OOCSS позволяет повторно использовать классы в своём HTML-коде, так как они не привязаны к конкретному модулю. Положительной стороной является зрелость методологии, так как она доказала свою эффективность в огромном количестве проектов.

БЭМ-методология имеет особые правила по именованию, поэтому позволяет эффективно использовать CSS-препроцессоры при работе с ней [19].

1.2.4 Atomic CSS

Методология Atomic CSS предоставляет совершенно иной подход к созданию оформления, чем методологии БЭМ, SMACSS и OOCSS. В данном случае, все стили элементов собираются из отдельных небольших частей – так называемых «атомов». Атомы – это специальные классы, выполняющие лишь одну функцию. Такой процесс похож на написание inline-стилей, но вместо них используются классы, предоставляющие доступ к медиа-запросам, псевдо-классам и псевдо-элементам [20]. Короткие названия классов позволяют значительно сократить исходных код, а также обеспечить максимальное повторное его использование.

Например, чтобы создать красный прямоугольник с белым текстом внутри и внутренними отступами по 20 пикселей, нужно написать код, представленный на рисунке 6.

```
<div class="Bgc(#f00) C(#fff) P(20px)">
        Lorem ipsum
</div>
```

Рисунок 6 – Код с использованием Atomic CSS

Таким образом, цвет фона задаётся классом «Bgc», цвет текста указывается классом с названием «C», а внутренние отступы – классом «P».

Псевдо-классы и псевдо-элементы создаются аналогично стандартному CSS – при помощи знака двоеточия.

Для использования медиа-запросов в конфигурационном файле определяются ключевые точки, описывающие различные значения ширины области показа. К именам классов добавляются названия ключевых точек, разделённые двумя знаками переноса.

Например, чтобы ширина блока была равна 50 % при маленькой ширине экрана, 33 % – при средней и 25 % – при большой, необходимо написать код, представленный на рисунке 7.

```
<div class="W(50%)--sm W(33%)--md W(25%)--lg">
        Lorem ipsum
</div>
```

Рисунок 7 – Адаптивный дизайн с использованием Atomic CSS

Таким образом, оформление элемента является совокупностью незначительных изменений каждого отдельного класса. Чтобы использовать все возможности методологии Atomic CSS, необходимо знать принцип именования большинства классов. Это создаёт дополнительные требования к профессиональным навыкам разработчиков данных проектов, а компании необходимо проводить обучение сотрудников.

Для генерации CSS-кода применяются специальные плагины. Официальным плагином, используемым в методологии Atomic CSS, является Atomizer. Atomizer анализирует исходный HTML-код, определяет используемые для описания оформления классы и собирает их в единый файл стилей [21]. Таким образом, в общий CSS-файл не попадают неиспользуемые стили.

Можно заметить, что подход Atomic CSS заключается в фактическом переносе написания стилей из таблиц стилей в сам HTML-код. Таким образом, размер файлов формата CSS значительно сокращается, но, в то же время, размер HTML-страниц увеличивается. Это может сыграть отрицательную роль, если на странице используется большое количество однотипных элементов со сложным оформлением. В данном случае, методологии БЭМ или SMACSS проявили бы свои положительные стороны.

Ещё одной отрицательной стороной методологии Atomic CSS является трудный для понимания код. Чтобы осознать итоговое оформление элемента, требуется потратить большое количество времени на прочтение и анализ его классов.

Таким образом, подход, предложенный Atomic CSS, применим в основном для разработки крупных проектов, в которых участвуют разработчики высокой квалификации и используются специальные программные средства для реализации.

Автор L. Watts в своей книге «Mastering Sass» [22] считает, что комплексным подходом к проектированию интерфейсов является применение специальных правил, входящих в методологии. Первой методологией, рассмотренной автором в качестве примера соглашения по именованию, является BEM. Главным плюсом БЭМ является возможность многократного использования. Этот принцип также применяется в другой популярной CSS-методологии – OOCSS.

OOCSS базируется на нескольких ключевых концепциях: разделение структуры от внешнего вида, а также разделение контейнера от контента. OOCSS также позволяет эффективно использовать препроцессоры.

Также автор рассматривает Atomic Design [23]. Основа подхода заключается в разделении стилей на части – «атомы». Этот способ организации кода позволяет описывать стили небольших элементов, а затем описывать большие элементы при помощи стилей маленьких.

Таким образом, для того, чтобы эффективно вести работу над большим проектом, необходимо знать принципы разных методологий, даже если непосредственно эти методологии не применяются в рамках проекта. Правильно выбранная методология позволяет ускорить процесс разработки, упростить внесение изменений в интернет-ресурс, увеличивать количество сотрудников, работающих над проектом. Методологии позволяют создавать элегантный CSS-код, понятный для всех разработчиков.

Автор А. А. Охрименко отмечает [24], что даже крупные порталы не всегда используют методологию БЭМ. Например, на сайте Youtube используется OOCSS, на Facebook – собственная методология, на сайте Yahoo используется Atomic CSS, на Amazon – аналог Atomic CSS собственной разработки.

Таким образом, CSS-методологию БЭМ нельзя назвать универсальным инструментом для любого проекта.

1.3 Сравнение CSS-методологий

В результате проведённого обзора популярных CSS-методологий вёрстки, существующих на сегодняшний день, были выявлены их ключевые отличия, схожие черты и варианты использования. На основе этих закономерностей, для проведения полноценного сравнительного анализа, были определены критерии сравнения.

Таким образом, для сравнения CSS-методологий БЭМ, SMACSS, Atomic CSS и OOCSS были выявлены следующие основные критерии:

- общее количество HTML-кода для реализации однотипных проектов;
- общее количество CSS-кода для реализации однотипных проектов;
- наличие специальных инструментов для эффективной разработки мобильной версии сайта;
- наличие специальных инструментов для эффективного создания адаптивной вёрстки;
- поддержка создания самодокументированного кода;
- возможность оптимизации страниц сайта под различные браузеры;
- необходимость написания конфигурационных файлов и сборки проекта;
- наличие модульной архитектуры;
- поддержка специальных инструментов для разработки (CSS-препроцессоров, шаблонизаторов);
- количество документации и книг по методологии, поддержка сообщества;
- количество готовых плагинов, модулей, SDK и т.д.;
- возможность применения методологии для проектов различного масштаба (крупные, небольшие проекты).

Для сравнения методологий вёрстки по общему количеству HTML-кода создаются четыре веб-страницы, имеющие идентичное отображение в браузере, но использующие различные подходы. Оценка данного критерия происходит по следующей системе: 2 балла получает CSS-методология с наименьшим объёмом получившейся HTML-страницы; каждая последующая методология, объём страницы которой больше предыдущей, получает на 0,5 балла меньше. Таким образом, CSS-методология, имеющая наибольший объём HTML-страницы, получает 0,5 балла.

Для оценки критерия объёма CSS-кода используется следующая система: 2 балла получает CSS-методология с наименьшим объёмом получившейся страницы каскадных стилей; каждая последующая методология, объём страницы которой больше предыдущей, получает на 0,5 балла меньше. CSS-методология, имеющая наибольший объём CSS-страницы, получает 0,5 балла.

Оценка критерия эффективности разработки мобильной версии происходит по следующей системе: 1 балл получает CSS-методология, имеющая специальные инструменты для создания мобильной версии сайта; 0,5 балла – отсутствуют специальные инструменты разработки, создание мобильной версии не требует копирования значительного объёма кода; 0 баллов – создание мобильной версии затруднено, требуется копирование или написание большого объёма кода.

Для оценки критерия эффективности создания адаптивного дизайна применяется следующая система: 1 балл – имеются специальные инструменты для адаптивной вёрстки; 0,5 балла – отсутствуют какие-либо специальные инструменты создания адаптивной вёрстки, поддерживается простое использование медиа-запросов; 0 баллов – создание адаптивной вёрстки и использования медиа-запросов усложнено.

Оценка критерия поддержки самодокументированного кода осуществляется по следующей системе: 1 балл – структура проекта упрощает поиск и анализ кода, поддерживаются генераторы документации; 0,5 балла – поддерживаются генераторы документации; 0 баллов – не поддерживаются генераторы документации.

Для оценки критерия возможности оптимизации под браузеры используется следующая система оценки: 1 балл – поддерживается нормализация стилей, поддерживается добавление префиксов браузеров; 0,5 баллов – не поддерживается нормализация стилей, поддерживаются префиксы браузеров; 0 баллов – не поддерживается нормализация стилей, не поддерживается добавление префиксов браузеров.

Для оценки простоты сборки проектов используется следующая система: 1 балл – сборка проекта не требуется; 0,5 балла – применима любая система

сборки проекта; 0 баллов – сборка проекта усложнена, подходит ограниченное число сборщиков.

Критерий наличия модульной архитектуры оценивается по следующему принципу: 1 балл – имеется возможность копирования кода между проектами без внесения каких-либо изменений; 0,5 балла – копирование кода из одного проекта в другой возможно с небольшими внесениями изменений; 0 баллов – копирование кода между проектами требует внесения большого количества изменений.

Для оценки специальных инструментов разработки используется следующая система оценки: 1 балл – поддерживаются любые препроцессоры, имеется специальный шаблонизатор; 0,5 балла – поддерживаются любые препроцессоры, специальные шаблонизаторы не предусмотрены; 0 баллов – не поддерживаются шаблонизаторы и препроцессоры.

Оценка критерия поддержки сообщества и наличия документации происходит по следующей системе: для каждой методологии рассчитывается показатель, основанный на сумме количества результатов поиска по запросам, включающим названия соответствующих методологий, в сервисах google.com, github.com и stackoverflow.com; CSS-методологии сортируются по уменьшению данного показателя; методология, находящаяся на первом месте, получает 2 балла, на втором месте – 1,5 балла, на третьем месте – 1 балл, на четвёртом месте – 0,5 балла.

Для оценки количества готовых плагинов и модулей анализируются результаты поиска в сервисе npmjs.com по следующей системе: 2 балла получает методология, имеющая наибольшее количество результатов поиска; каждая последующая CSS-методология, имеющая меньшее количество результатов поиска, получает на 0,5 балла меньше. Таким образом, методология с минимальным количеством результатов поиска получает 0,5 балла.

Критерий эффективности применения методологий для проектов различного масштаба оценивается по следующей системе: 1 балл получает методология, применение которой эффективно в крупных, средних и небольших проектах; 0,5 балла получает CSS-методология, применение которой эффективно

в проектах двух видов масштаба; 0 баллов – только в крупных, средних или небольших проектах.

На основе представленных критериев был проведён сравнительный анализ следующих CSS-методологий: БЭМ, SMACSS, Atomic CSS и OOCSS. Результат сравнения представлен в таблице 1.

Таблица 1 – Сравнение CSS-методологий

Критерий сравнения	БЭМ	SMACSS	Atomic CSS	OOCSS
1	2	3	4	5
Общее количество HTML-кода для реализации однотипных проектов	1 (особые принципы именования классов подразумевают использование пространств имён, что увеличивает HTML-код)	2 (применение селекторов по тегам и id, а также классов позволяет сделать HTML-код сбалансированным, без лишних элементов)	0,5 (из-за написания классов аналогично inline-стилям, код HTML получается значительно больше, чем в других методологиях)	1,5 (отсутствие пространства имён позволяет избавиться от лишней информации в названиях классов)
Общее количество CSS-кода для реализации однотипных проектов	0,5 (код блоков может быть многократно использован, но невозможность повторного использования кода элементов увеличивает общий код стилей)	1 (методология подразумевает использование библиотек наподобие normalize.css или Reset CSS, что увеличивает код стилей)	2 (ввиду особого применения классов, код CSS получается максимально сжатым и многократно используемым)	1,5 (отделение оболочек от структуры позволяет сократить код стилей)
Наличие специальных инструментов для эффективной разработки мобильной версии	1 (при помощи уровней переопределения упрощается разработка мобильной версии и версии для планшета)	0,5 (создание мобильной версии подразумевает значительное изменение стилей, но HTML-код может быть частично использован в других проектах)	0 (при создании мобильной версии сайта требуется значительная переработка исходного HTML-кода, что увеличивает время разработки и требуемые усилия)	0,5 (создание мобильной версии подразумевает значительное изменение стилей, но HTML-код может быть частично использован в других проектах)

Продолжение таблицы 1

1	2	3	4	5
Наличие специальных инструментов для эффективного создания адаптивной вёрстки	0,5 (жёсткие правила именования классов не подразумевают полноценное использование состояний)	0,5 (отсутствуют какие-либо дополнительные ограничения на применение медиа-запросов и состояний)	1 (использование ключевых точек позволяет динамически менять оформление в зависимости от размера экрана, изменения вносятся непосредственно в код HTML)	0,5 (отсутствуют какие-либо дополнительные ограничения на применение медиа-запросов и состояний)
Поддержка создания самодокументированного кода	1 (строгая архитектура проекта ускоряет поиск кода, поддерживаются генераторы документации)	1 (понятная архитектура проектов, поддерживаются генераторы документации)	0 (не поддерживаются генераторы документации)	0,5 (отсутствует строгая архитектура проекта, поддерживаются генераторы документации)
Возможность оптимизации страниц сайта под различные браузеры	0,5 (уровни переопределения позволяют эффективно проводить оптимизации под различные платформы, методологией не рекомендуется нормализация стилей)	1 (стили темы и селекторы потомков позволяют настраивать отображение в зависимости от внешних факторов)	0 (основные принципы методологии делают оптимизацию под браузеры трудно выполнимой)	0,5 (имеется возможность добавления префиксов браузеров)
Необходимость написания конфигурационных файлов и сборки проекта	0 (методология не ограничивает используемые сборщики проектов. По умолчанию поддерживаются ENB и Gulp. Сборка требует определённых настроек)	0,5 (для объединения всех файлов стилей в один и компиляции с языков препроцессоров может использоваться любая система сборки)	0,5 (есть готовый плагин Atomizer для создания файла стилей)	1 (методология не требует применения сборки. Для компиляции с языков препроцессоров может использоваться любая система сборки)

Продолжение таблицы 1

1	2	3	4	5
Наличие модульной архитектуры	1 (пространства имён и модульный подход позволяют обеспечить максимальное повторное использование кода как на одной странице, так и в разных проектах)	0,5 (разделение стилей по категориям позволяет использовать многократно базовые стили и стили макета, но усложняет использование модулей)	0 (отсутствие модульной архитектуры затрудняет повторное использование кода)	0,5 (объектно-ориентированный подход позволяет добиться наличия независимых компонентов, но, ввиду отсутствия пространств имён, возможны конфликты)
Поддержка специальных инструментов для разработки (CSS-препроцессоров, шаблонизаторов)	1 (поддерживаются любые препроцессоры, процесс разработки существенно ускоряется, используется шаблонизатор bem-xjst)	0,5 (поддерживаются любые CSS-препроцессоры, скорость разработки незначительно увеличивается, специальных шаблонизаторов нет)	0,5 (использование препроцессоров на процесс разработки практически не влияет)	0,5 (поддерживаются любые CSS-препроцессоры, скорость разработки незначительно увеличивается, специальных шаблонизаторов нет)
Количество документации и книг по методологии, поддержка сообщества	2 (официальный ресурс методологии, документация на github, широкая поддержка сообществом)	1 (официальная книга методологии, множество документации, широкая поддержка сообщества)	0,5 (официальный ресурс методологии, множество документации, меньшая, по сравнению с БЭМ и SMACSS, поддержка сообщества)	1,5 (малосодержательный официальный сайт, множество документации, широкая поддержка сообщества)
Количество готовых плагинов, модулей, SDK	2 (множество готовых плагинов Node.js, стартовых проектов, SDK, компонентов)	0,5 (готовые стартовые проекты, готовые библиотеки для стилей базовой категории)	1 (официальный плагин Atomizer для автоматизированного создания стилей, готовые стартовые проекты)	1,5 (библиотеки готовых компонентов, стартовые проекты)

Продолжение таблицы 1

1	2	3	4	5
Возможность применения методологии для проектов различного масштаба	1 (методология применима как для создания небольших статических и динамических сайтов, так и в крупных проектах. Основные положительные стороны проявляются при создании крупных проектов)	2 (методология применима при создании крупных и небольших сайтов)	0,5 (методология в основном применима при создании крупных проектов, при создании небольших сайтов увеличивается время реализации)	1,5 (положительные стороны методологии проявляются как в крупных, так и в небольших проектах. Требуется дополнительное время для определения оболочек)
ИТОГО	11,5	11	6,5	11,5

В результате проведённого сравнения максимальную сумму баллов показали методологии БЭМ и OOCSS – 11,5. Методология SMACSS набрала на 0,5 балла меньше – 11. Наименьшая сумма баллов у Atomic CSS – 6,5.

Выводы по главе

Основными проблемами, возникающими при разработке пользовательских веб-интерфейсов, являются необходимость поддержки большого количества браузеров разных производителей и различных форм-факторов устройств для просмотра страниц, а также сложности взаимодействия нескольких разработчиков в рамках одной команды.

Определено, что наиболее универсальным способом решения данных проблем является применение методологий вёрстки. Большинство научных исследований посвящено следующим популярным CSS-методологиям: SMACSS, OOCSS, BEM и Atomic CSS.

На основе особенностей и общих характеристик данных методологий были определены критерии их сравнения и проведён обширный сравнительный анализ. Таким образом, наибольшим количеством конкурентных преимуществ обладают CSS-методологии БЭМ и OOCSS.

ГЛАВА 2. АНАЛИЗ CSS-МЕТОДОЛОГИЙ И ИХ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

В данной главе определяются наиболее эффективные CSS-методологии для создания проектов различной направленности. На основе положительных и отрицательных сторон методологий разрабатывается новая методология, обладающая конкурентными преимуществами.

2.1 Рекомендации по выбору методологий в различных проектах

На сегодняшний день не существует какой-либо методологии вёрстки, являющейся универсальным способом создания пользовательских интерфейсов для сайтов. Каждый проект обладает своими особенностями, которые влияют на требования к используемой методологии. Таким образом, необходимо определить сферы применения различных подходов.

2.1.1 Разработка мобильной версии сайта

В последнее время создание удобной для просмотра на мобильных устройствах версии сайта является обязательным элементом веб-разработки. Согласно статистике, в 2017 году доля трафика на сайты с мобильных устройств превысила 50 % [25]. К четвёртому кварталу 2018 года эта доля составляет 52.2 % и продолжает расти. К тому же, поисковая система Google, лидирующая по количеству обрабатываемых запросов, в 2018 году ввела приоритетное индексирование мобильных версий сайтов [26].

На сегодняшний день существует два различных подхода к разработке удобных для просмотра на мобильных устройствах сайтов – создание отдельной мобильной версии и использование адаптивной вёрстки [27]. Мобильная версия имеет ряд преимуществ, но также и несколько недостатков. К преимуществам разработки мобильной версии можно отнести: возможность оптимизации контента и его расположения, так как мобильная версия не зависит от десктопной; отсутствие необходимости менять существующую кодовую базу, в случае наличия готовой полной версии; возможность оптимизации объёма веб-страниц. Недостатками данного подхода являются: большое количество времени на разработку, в случае создания вместе с десктопной версией; необходимость перенаправления мобильных пользователей на специальную версию сайта, а также сложность SEO-оптимизации.

Таким образом, главными требованиями к CSS-методологиям вёрстки при реализации мобильной версии являются:

- минимальное количество HTML-кода;
- минимальное количество CSS-кода;
- наличие модульной архитектуры;
- специальная архитектура для мобильной версии;
- поддержка самодокументированного кода;
- возможность оптимизации страниц под различные браузеры;
- большое количество готовых компонентов и плагинов.

Согласно таблице сравнения CSS-методологий, представленной в Главе 1, наиболее подходящими методологиями для разработки мобильной версии сайта являются:

- БЭМ (7 баллов);
- OOCSS (6,5 балла);
- SMACSS (6,5 балла);
- Atomic CSS (3,5 балла).

Таким образом, для разработки мобильной версии сайта целесообразно использовать методологию БЭМ, так как она опережает методологии SMACSS и OOCSS на 0,5 балла.

Главная особенность методологии БЭМ – наличие уровней переопределения. Эти уровни помогают разделять стили на основные, для мобильного устройства и планшета, поддерживать разные темы оформления. Таким образом, после сборки проекта получается несколько CSS-файлов, предназначенных для различных задач. Данный подход позволяет без проблем разрабатывать мобильную версию при наличии десктопной.

Так как БЭМ – одна из самых популярных методологий на сегодняшний день, существует огромное количество готовых модулей, компонентов и плагинов, предназначенных для данной методологии. Также в открытом доступе присутствует большое количество документации на разных языках с описанием основных положений методологии.

Однако методология БЭМ обладает также отрицательными сторонами. К таким недостаткам можно отнести сложные и длинные названия классов. По этой причине, количество HTML-кода оказывается больше, чем у конкурентов.

2.1.2 Создание адаптивной вёрстки

Создание адаптивной вёрстки, наравне с разработкой мобильной версии, является одним из основных способов создания удобных для просмотра на мобильных устройствах сайтов. Данный подход является оптимальным в случаях, когда требуется одновременно создать десктопную и мобильную версии.

Суть адаптивной вёрстки заключается в том, что при разработке применяются особые техники, позволяющие менять контент веб-страниц в зависимости от размеров экрана устройства, на котором этот контент просматривается. Элементы страницы могут автоматически скрываться или появляться, может меняться их расположение или размер.

Ключевой технологией, применяемой при создании адаптивной вёрстки, являются медиа-запросы (media queries) [28]. Они позволяют корректировать стили элементов в зависимости от размеров области просмотра. При помощи

метатега viewport имеется возможность настроить соответствие физического пикселя на экране устройства с размеров виртуального пикселя, используемого при отображении информации. Также, существует возможность запретить пользователю масштабировать страницу на мобильном устройстве.

Преимуществами создания адаптивной вёрстки являются: меньшие временные затраты при разработке десктопной и мобильной версий одновременно; отсутствие необходимости перенаправлять пользователя на мобильную версию; простая SEO-оптимизация; возможность менять расположение элементов «на лету».

Недостатками данного подхода являются: увеличение объёма работ при наличии готовой десктопной версии; сложность тестирования интерфейса при изменении размеров области просмотра; невозможность применения на сайтах с большим количеством элементов; сложность оптимизации объёма кода.

Таким образом, к CSS-методологиям для создания адаптивного дизайна должны предъявляться следующие требования:

- минимальное количество CSS-кода;
- простота создания адаптивной вёрстки;
- высокая скорость разработки;
- возможность повторного использования кода в других проектах;
- широкая поддержка сообщества.

В соответствии с таблицей сравнения CSS-методологий, наиболее удобными методологиями для создания адаптивного дизайна являются:

- OOCSS (4 балла);
- БЭМ (4 балла);
- Atomic CSS (3,5 балла);
- SMACSS (3 балла).

Таким образом, методологии OOCSS и БЭМ набрали одинаковое количество баллов (4 балла), но приоритетной методологией является OOCSS.

OOCSS не имеет каких-либо жёстких правил именования классов элементов, но содержит два главных принципа – отделение структуры от

оболочек, и отделение контейнеров от содержимого. Это позволяет значительно сократить объём CSS-кода, ввиду обширного повторного использования стилей. Данные стили могут быть использованы при создании других проектов. Также, данная методология определяет границы разумного использования селекторов дочерних элементов, что не позволяет делать CSS-код чрезмерно сложным. Благодаря этому, имеется возможность использовать селекторы дочерних элементов в медиа-запросах для управления элементами при изменении области просмотра.

Методология вёрстки OOCSS не содержит сложных правил написания каскадных таблиц стилей. Поэтому, обучение сотрудников занимает не слишком много времени. Также, это увеличивает скорость разработки страниц, что необходимо при создании адаптивной вёрстки. К тому же, CSS-код, написанный по методологии OOCSS, получается наглядным и лёгким к анализу. Увеличивается скорость проведения рефакторинга кода, а также не возникает необходимость менять HTML-код под код стилей.

2.1.3 Создание одностраничных сайтов и лендингов

В последнее время область веб-разработки по созданию одностраничных сайтов и лендингов активно развивается [29]. Этому способствует широкое использование онлайн каналов привлечения посетителей, в основном – контекстной рекламы. На одной посадочной странице должна присутствовать вся информация о товаре или услуге: изображения, описание, характеристики, преимущества. Также должны быть контакты продавца и инструменты для связи, такие как форма обратной связи или онлайн-чат.

В результате анализа 18639 посадочных страниц, проведённого Unbounce Landing Page Analyzer, средняя конверсия страницы может достигать 13.5 % [30].

Ключевыми параметрами, влияющими на конверсию полученного одностраничного сайта, являются:

- время загрузки страницы;
- удобство просмотра на мобильных устройствах;
- общее количество слов.

Преимуществами создания одностраничного сайта, по сравнению с многостраничным, являются: высокая скорость разработки; оптимизация страницы под конкретный товар или услугу для повышения конверсии; простота тестирования.

Недостатками создания одностраничного сайта или лендинга являются: сложность SEO-оптимизации; необходимость сокращения информации для расположения на одной странице; возможность описания лишь одного товара или услуги.

Таким образом, при разработке одностраничного сайта или лендинга к CSS-методологиям вёрстки предъявляются следующие требования:

- минимальное количество HTML-кода;
- минимальное количество CSS-кода;
- простота создания мобильной версии или адаптивной вёрстки;
- высокая скорость разработки;
- возможность оптимизации страницы под браузеры.

Таким образом, основываясь на таблице сравнения CSS-методологий, были получены результаты эффективности применения методологий для разработки одностраничных сайтов:

- SMACSS (5 баллов);
- OOCSS (4,5 балла);
- БЭМ (3,5 балла);
- Atomic CSS (3,5 балла).

Таким образом, методология SMACSS позволяет решить задачу по созданию лендингов более эффективно, так как имеет на 0,5 балла больше, чем CSS-методология OOCSS.

В основе CSS-методологии вёрстки SMACSS лежит разделение всех стилей на пять категорий: базовые стили, стили макета, стили модулей, стили состояния и стили темы. Благодаря этому получается очень простая структура проекта, что облегчает анализ и обучение сотрудников. Также, имеется возможность копировать готовые стили в другие проекты.

Ввиду представленных положительных сторон методологии, итоговая скорость разработки одностраничных сайтов возрастает. К тому же, существует набор готовых библиотек для различных категорий. Стили темы и селекторы потомков позволяют настраивать отображение в зависимости от внешних факторов, таких как браузер пользователя. В SMACSS отсутствуют какие-либо дополнительные ограничения на применение медиа-запросов и состояний, что позволяет без лишних проблем создавать адаптивную вёрстку.

2.1.4 Создание крупных высоконагруженных сайтов

Определить величину проекта можно по нескольким параметрам, главные из которых: количество запросов в секунду (посетителей в день), количество строк кода, размер команды, работающей над проектом, требуемое время для реализации [31].

Таким образом, к крупному высоконагруженному проекту можно отнести проект со следующими минимальными характеристиками:

- 1 миллион посетителей в день;
- 2 миллиона строк кода;
- 10 сотрудников, постоянно работающих над проектом;
- срок реализации – 6 месяцев.

Особенность разработки сайтов для крупных проектов заключается в повышенных требованиях к организации и качеству кода. Также необходима возможность быстрого обучения новых сотрудников, подключившихся к проекту.

Код должен быть модульным с возможностью использования в нескольких частях сайта или других проектах.

Во время разработки крупного высоконагруженного сайта могут возникнуть следующие проблемы: сложность поиска нужного участка кода, отсутствие документации к какому-либо плагину или модулю, ошибки, связанные со сборкой проекта.

Таким образом, к CSS-методологиям вёрстки при разработке высоконагруженного сайта предъявляются следующие требования:

- возможность повторного использования кода в других проектах;
- большое количество документации и широкая поддержка сообщества;
- возможность применения разных инструментов для разработки;
- большое количество готовых модулей и плагинов;
- возможность быстрого обучения нового сотрудника методологии;
- наглядность полученного кода, строгая структура проекта.

В соответствии с таблицей сравнения CSS-методологий, представленной в Главе 1, наиболее подходящими методологиями для создания крупного высоконагруженного сайта являются:

- БЭМ (6 баллов);
- OOCSS (4 балла);
- SMACSS (2,5 балла);
- Atomic CSS (2 балла).

На основе полученных результатов видно, что методология БЭМ значительно опережает другие методологии вёрстки при разработке крупных проектов. Отрыв от второго и третьего мест (OOCSS и SMACSS) составляет 2 и 3,5 балла соответственно.

Главным преимуществом БЭМ при разработке высоконагруженного сайта является особая модульная структура. Благодаря данной структуре и правилам именования классов гарантируется многократное корректное использование кода в разных частях веб-страницы и всего сайта. Это ускоряет разработку и тестирование вёрстки. Все компоненты разделены по папкам, что упрощает поиск

необходимых участков кода. Код, в свою очередь, получается наглядным, лёгким к анализу.

Как было определено ранее, CSS-методология БЭМ является одной из самых популярных. Таким образом, имеется большое количество инструментов для разработки, таких как сборщики, шаблонизаторы и CSS-препроцессоры.

Также, на сегодняшний день существует множество специалистов, использующих данную методологию при создании сайтов. Это говорит о большом рынке разработчиков, которые могут участвовать в проекте. Обучение новых сотрудников не занимает много времени, так как представлено огромное количество литературы на разных языках, посвящённой данной технологии реализации.

2.1.5 Создание плагинов и библиотек

Согласно статистике одной из самых популярных систем контроля версий GitHub, количество проектов с открытым исходным кодом постоянно увеличивается [32]. Значительную часть всех проектов составляют плагины и библиотеки, связанные с пользовательскими веб-интерфейсами. Разработка данных пользовательских интерфейсов имеет определённые особенности. К этим особенностям относится необходимость инкапсуляции кода, так как библиотеки не должны влиять на существующие элементы страниц.

Также, одним из важных требований к плагинам и библиотекам является их минимальный объём. Дело в том, что объём библиотеки влияет на время загрузки страниц, их отрисовку. В случае, если интерфейс загружается с задержкой, у пользователя возникает негативный пользовательский опыт.

Таким образом, CSS-методологии для разработки плагинов и библиотек должны соответствовать следующим требованиям:

- простота создания адаптивной вёрстки;

- возможность оптимизации под различные браузеры;
- возможность повторного использования кода;
- количество готовых модулей и плагинов.

Основываясь на таблице сравнения CSS-методологий, наиболее эффективными методологиями являются:

- БЭМ (4 балла);
- OOCSS (3 балла);
- SMACSS (2,5 балла);
- Atomic CSS (2 балл).

Таким образом, методология BEM набирает максимальное количество баллов и подходят для реализации библиотек и плагинов пользовательского веб-интерфейса.

Основным преимуществом данной методологии является возможность изоляции кода от внешнего окружения, что обеспечивает возможность к повторному использованию. Также CSS-методология БЭМ позволяет создавать адаптивную вёрстку эффективнее, чем другие методологии.

Недостатком данной CSS-методологии вёрстки является не самый маленький объём кода, получаемый в результате создания плагинов и библиотек.

2.2 Положительные и отрицательные стороны методологий вёрстки

Как было определено ранее, методология БЭМ имеет наиболее широкие сферы применения, такие как разработка мобильной версии сайта, создание крупных высоконагруженных сайтов, реализация плагинов и библиотек. На это влияют главные аспекты каждой конкретной CSS-методологии.

Таким образом, основной особенностью БЭМ является его модульная архитектура, заключающаяся в определённом принципе именования классов. Благодаря данной архитектуре упрощается повторное использование кода, что

важно при создании проектов с открытым исходным кодом. Это также ускоряет разработку мобильной версии сайта.

Данные положительные стороны привели к доминированию BEM среди методологий вёрстки. В связи с этим, на сегодняшний день существует большое число готовых инструментов для разработки.

Недостаток методологии одновременно связан с его плюсом – особенностью именования классов. В результате, имена классов оказываются очень длинными, из-за чего увеличивается объём HTML и CSS-кода.

Методология SMACSS наиболее подходит для создания небольших сайтов, таких как лендинги. Это достигается за счёт простой структуры проекта, завязанной на разделении стилей по категориям. Таким образом, ключевые особенности методологии вёрстки SMACSS – быстрое обучение сотрудников правилам и простота сборки проектов. Благодаря этому увеличивается скорость разработки и появляется возможность использования для создания одностраничных сайтов.

SMACSS не обладает явно выраженными отрицательными сторонами, поэтому может применяться в качестве универсального способа создания пользовательских веб-интерфейсов.

CSS-методология Atomic CSS значительно отстаёт от своих аналогов в вариантах её использования. Возможными сферами применения данной методологии являются разработка крупных сайтов и создание адаптивной вёрстки. Это связано с тем, что названия классов формируются в виде сокращений от выполняемых ими функций. Таким образом, Atomic CSS подходит для использования высококвалифицированными специалистами. При работе большой команды высококвалифицированных специалистов над крупным проектом появляется возможность сократить необходимость написания стилей до минимума.

Недостатком методологии является фактический перенос написания стилей в HTML-код. В результате, объём кода разметки получается существенным, что сказывается на быстродействии.

Подход OOCSS позволяет сократить общее количество кода веб-ресурса, благодаря чему уменьшается время начальной загрузки интерфейса. Небольшое количество правил методологии делает оптимальным процесс обучения сотрудников, что увеличивает поддержку технологии сообществом. На основе данной методологии был разработан ряд подходов, улучшающих определённые её аспекты для увеличения вариантов использования.

Недостатком CSS-методологии OOCSS является отсутствие определённой архитектуры всего проекта. Из-за этого могут возникать ситуации с дублированием или некорректным использованием классов. В результате, возникают конфликты переопределения стилей.

2.3 Синтез новой CSS-методологии

Как было определено ранее, существующие методологии вёрстки имеют существенные недостатки. Причём, некоторые отрицательные стороны одной методологии могут быть компенсированы подходом его конкурента. Таким образом, имеется возможность синтезировать новую CSS-методологию на основе существующих.

Главные положительные стороны представленных методологий вёрстки:

- особая архитектура проектов, созданных по SMACSS, упрощающая поиск и анализ исходного кода;
- принципы именования классов БЭМ, благодаря которым достигается модульность кода;
- принцип отделения оболочек от структур в OOCSS, снижающий объём кода каскадных стилей;
- отделение контейнеров от содержимого;
- генерация CSS-кода при помощи плагинов в Atomic CSS.

Основные недостатки представленных CSS-методологий:

- увеличение объёма HTML-кода ввиду особенностей именования классов BEM;

- высокая вероятность совершения ошибок при использовании классов методологии OOCSS, так как отсутствуют строгая архитектура проекта и правила именования классов;
- ограничение на использование селекторов по тегам и id элементов в методологии вёрстки БЭМ;
- увеличение объёма HTML-кода в Atomic CSS в результате фактического переноса написания стилей в код страниц;
- слишком большое количество категорий в методологии SMACSS;
- сложные названия классов в Atomic CSS.

Таким образом, главными элементами новой методологии должны быть архитектура (разделение стилей по папкам) и правила именования классов.

Архитектура проектов, созданных по методологии БЭМ, усложняет процесс разработки. При этом, архитектура, предложенная SMACSS, наоборот, увеличивает скорость разработки. Таким образом, в общей архитектуре требуются категории стилей, имеющие описания тегов и id элементов.

Стили модулей SMACSS имеют схожие функции с блоками BEM. Но, ввиду большей модульности, целесообразно использовать именно подход, созданный в российской компании Яндекс. К тому же, все блоки должны быть распределены по отдельным папкам, что облегчит поиск необходимых участков кода.

Блоки БЭМ имеют недостаток, связанный с вынужденным копированием стилей из одного блока в другой для обеспечения модульности. Эта проблема решена в методологии OOCSS путём выделения оболочек. Оболочки не имеют своей категории согласно архитектуре SMACSS, поэтому её необходимо добавить.

Стили состояний, согласно SMACSS, описывают стили при изменении каких-либо динамических параметров элементов. Этими параметрами могут быть: изменение или добавление классов, добавление псевдо-классов, изменение размеров видимой области страницы. Контролировать изменение видимой области можно при помощи медиа-запросов. Реакция на изменение данных параметров описывается непосредственно в блоках БЭМ. Поэтому, пропадает необходимость в папке стилей состояний.

Также, стили темы SMACSS дублируют функции оболочек OOCSS. Изменение тем осуществляется при помощи использования селекторов дочерних элементов и селекторов-потомков. Таким образом, чтобы изменить стиль всей страницы, достаточно добавить класс к корневому контейнеру страницы, например, body. Имеется аналогичная возможность менять оформление частей макета. Учитывая данное обстоятельство, из общей архитектуры проекта необходимо убрать стили темы.

В итоге, категории стилей проекта, основанного на синтезе принципов БЭМ, SMACSS и OOCSS, выглядят следующим образом:

- Base;
- Layout;
- Blocks;
- Skins.

В базовых стилях (base) используются только селекторы тегов. Это необходимо для первоначального оформления основы сайта. На этом уровне возможно использование таких библиотек, как normalize.css или reset.css. Также, категория base описывает фирменный стиль сайта: используемый шрифт, цвета ссылок, цвет текста, подчёркивание элементов, стиль списков. На данном уровне крайне не рекомендуется использовать модификатор «!important».

Стили макета (layout) описывают оформление элементов, которые используются на странице всегда лишь один раз: шапка сайта, подвал, меню, колонки, всплывающее окно и т. д. Для задания стилей таким элементам используются селекторы по уникальному идентификатору (id). Также имеется возможность использовать id в селекторе в качестве дочернего элемента или элемента-потомка от класса. Это позволяет динамически менять оформление сайта.

Стили блоков (blocks) используются для описания элементов, многократно используемых на странице: кнопки, контейнеры с диалогами, слайдеры, виджеты, элементы меню и многое другое. Стили таким элементам присваиваются при помощи классов (class). Это позволяет достигнуть модульности кода. Причём, для

обеспечения быстрого поиска нужных участков кода, все стили блоков разделены по отдельным файлам. Данный подход аналогичен методологии BEM, где блоки тоже структурно разделены. В стилях блоков нужно избегать использования селекторов по тегам. Это необходимо для максимальной гибкости кода и возможности проведения быстрого рефакторинга.

Стили оболочек (skins) предназначены для выделения характерных правил оформления элементов страницы в отдельные классы. Данный подход аналогичен правилу методологии OOCSS. Он позволяет сократить код каскадных таблиц стилей за счёт выделения повторяющихся свойств в отдельные классы. К таким свойствам относятся: фоновый цвет, цвет текста, закругление углов, способ отображения, внутренние и внешние отступы, стиль текста, тени и другие многократно встречающиеся свойства. На данном уровне приветствуется использование медиа-запросов. Это позволяет менять оформление страницы, ввиду изменения видимой области. Также классы оболочек целесообразно использовать в качестве дочерних элементов и элементов-потомков в селекторах.

Названия классов, созданных по представленной методологии, строятся по следующей формуле:

[Skin_class_1 [Skin_class_2 […]]] [block_class_1 [block_class_2]] [element_class] [modifier_class_1 [modifier_class_2 […]]].

Таким образом, чтобы отличать классы оболочек от классов блоков, необходимо названия классов оболочек писать с заглавной буквы.

Выводы по главе

На основе проведённого сравнительного анализа CSS-методологий были выявлены особенности реализации различных видов пользовательских веб-интерфейсов. Таким образом, для разработки мобильной версии сайта, библиотек и плагинов, а также в крупных проектах целесообразно использовать методологию БЭМ. Для создания одностраничных сайтов и лендингов подходит

методология SMACSS. При использовании в проекте адаптивного дизайна целесообразно применение методологии OOCSS.

Преимуществами представленных методологий являются: архитектура проектов, созданных по SMACSS; правила именования классов БЭМ; объём кода каскадных таблиц стилей у методологии OOCSS.

Основываясь на положительных сторонах исследуемых методологий вёрстки, была синтезирована новая методология, имеющая следующие категории стилей: base, layout, blocks, skins.

ГЛАВА 3. РАЗРАБОТКА ВЕБ-СТРАНИЦЫ С ПРИМЕНЕНИЕМ CSS-МЕТОДОЛОГИИ, ОБЪЕДИНЯЮЩЕЙ ПРИНЦИПЫ BEM, OOCSS И SMACSS

3.1 Определение технологий реализации

Для создания данной посадочной страницы была выбрана современная платформа Node.js, поддерживающая использование пре- и постпроцессоров, систем сборки и тестирования, а также запуск локальных веб-серверов для разработки.

3.1.1 Выбор языка описания стилей

На сегодняшний день для описания стилей веб-страниц применяется стандартизированный язык CSS. Благодаря данному языку имеется возможность задать внешний вид страниц согласно исходному макету.

Помимо языка стилей CSS сегодня существует несколько языков, компилируемых в CSS. С подобными языками работает специальное программное обеспечение – CSS-препроцессоры. Наиболее популярными представителями данной категории являются Sass, Less и Stylus [33]. Эти языки добавляют некоторые синтаксические приёмы, которые не предусмотрены в самом CSS.

Язык Sass поддерживает использование переменных. Благодаря переменным имеется возможность быстрой замены повторяющихся значений или свойств. Также это позволяет уменьшить вероятность совершения ошибки. Благодаря поддержке вложенности сокращается объём исходного кода стилей.

Для увеличения модульности кода в Sass применяется фрагментация. В рамках этого, стили разделяются на файлы с последующим импортом. Имена фрагментов начинаются со знака нижнего подчёркивания.

Sass поддерживает примеси (миксины). Благодаря им имеется возможность упростить добавление префиксов браузеров и многократно использовать однотипный код. Также поддерживается специальный оператор «&», обозначающий ссылку на родительский селектор [34].

Для сокращения объёма кода используются классы-шаблоны и наследование. Таким образом, один и тот же код стилей может применяться в различных селекторах.

Помимо этого, язык Sass позволяет выполнять математические операции, такие как сложение, вычитание, умножение и деление. Также поддерживаются различные встроенные функции для преобразований цветов.

Одна из возможностей Sass – условные операторы и циклы, аналогичные функциональным языкам программирования.

Sass имеет два различных синтаксиса: SCSS и Sass. SCSS является новым синтаксисом, требующим использования фигурных скобок и разделителя – точки с запятой. Поэтому, любой CSS-документ является совместимым с SCSS. Синтаксис Sass не требует фигурных скобок и разделителя.

Язык стилей Less поддерживает большинство возможностей, представленных в Sass. Среди них: переменные, миксины, вложенность, математические операции, ссылки на родительский селектор, встроенные функции и фрагментация [35].

Язык Less, аналогично синтаксису CSS, требует использования фигурных скобок и разделителя, что может увеличить объём исходного кода.

Использование условного оператора отличается от Sass. Оператор «if» содержит три параметра, соответствующих тернарному оператору в функциональных языках программирования. Также поддерживаются циклы «each», взаимодействующие со списками.

Язык стилей Stylus имеет схожие черты с языками, представленными выше. Поддерживаются переменные, вложенность, ссылки на родительский селектор, примеси, математические операции, пользовательские функции, импорт и встроенные функции.

Аналогично синтаксису Sass, Stylus не требует использования фигурных скобок и разделителей. В отличие от языков Sass и Less, Stylus тесно связан с языком JavaScript [36].

Современный стандарт CSS3 также поддерживает некоторые из представленных возможностей. Среди таких технологий: переменные, импорт, выполнение математических операций внутри функции calc.

Сравнение языков описания стилей представлено в таблице 2.

Таблица 2 – Сравнение языков стилей

Критерий сравнения	Sass	Less	Stylus	CSS3
Переменные	+	+	+	+
Миксины	+	+	+	-
Импорт	+	+	+	+
Вложенность	+	+	+	+
Ссылки на родительский селектор	+	+	+	-
Встроенные функции	+	+	+	-
Циклы и условные операторы	+	+/- (оператор if аналогичен тернарному оператору, отсутствует цикл for)	+	-
Упрощённый синтаксис	+	-	+	-
Поддержка сообщества	+ (84 коммита за год)	+ (более 100 коммитов за год)	+/- (5 коммитов за год)	+

Таким образом, для эффективной разработки проектов, соответствующих методологии, представленной в Главе 2, к языку стилей предъявляются следующие требования:

- поддержка фрагментации (импорт файлов);
- поддержка ссылок на родителя;
- упрощённый синтаксис;
- обширная поддержка сообщества и большое количество документации.

Основываясь на сравнении языков описания стилей, наибольшим количеством положительных сторон для предложенной методологии обладает Sass.

Объединение стилей четырёх категорий в один файл осуществляется при помощи импорта.

Благодаря ссылкам на родительский селектор в виде оператора «&» упрощается написание стилей блоков, снижается вероятность совершения ошибки в наименованиях классов.

3.1.2 Оптимизация под различные браузеры

Для оптимизации стилей под различные браузеры существует несколько подходов. Один из самых эффективных и быстрых способов оптимизации – применение CSS-постпроцессоров.

CSS-постпроцессор – специальное программное обеспечение, принимающее на вход код стилей в формате CSS и преобразующее его при помощи плагинов [37]. Благодаря этому автоматически выполняются некоторые рутинные операции, такие как добавление префиксов вендоров и добавление полифиллов. Помимо этого, для увеличения скорости передачи часто выполняется минификация кода. К тому же, постпроцессор сигнализирует об обнаружении синтаксических ошибок в коде, что играет важную роль при разработке крупных проектов.

На сегодняшний день самым популярным CSS-постпроцессором является PostCSS, к которому доступно более 2000 плагинов для преобразований [38].

Для преобразования современных возможностей CSS, не поддерживаемых некоторыми старыми браузерами, в более универсальный код применяется PostCSS-плагин postcss-preset-env. В него встроен плагин для автоматического добавления префиксов вендоров – autoprefixer. Для определения версий браузеров, для которых необходимо оптимизировать код, применяется специальный инструмент – Browserslist. Данный инструмент составляет список браузеров, соответствующих заданным разработчиком критериям.

Поддерживается следующий список критериев:

- последние несколько версий браузеров (например, последние три версии);

- браузеры, занимающие определённую долю рынка (например, более 1 %);
- браузеры, поддерживаемые Node.js определённой версии (например, node 10.4);
- только «живые» браузеры.

В случае, если выбрать браузеры с долей мирового рынка более 1 %, то плагин autoprefixer преобразует CSS-код, представленный на рисунке 8, к виду, представленному на рисунке 9.

```
.form::placeholder {
  color: #43bfe1;
  font-size: 14px;
}
```

Рисунок 8 – CSS-код без использования autoprefixer

```
.form:-ms-input-placeholder {
  color: #43bfe1;
  font-size: 14px;
}
.form::placeholder {
  color: #43bfe1;
  font-size: 14px;
}
```

Рисунок 9 – CSS-код после обработки autoprefixer

Плагин postcss-preset-env часто применяется для поддержки технологии CSS Grid различными браузерами, в частности, Internet Explorer [39]. Дело в том, что некоторые старые браузеры не поддерживают свойства grid-area, grid-template-columns, grid-template-rows и grid-template-areas.

Например, postcss-preset-env преобразовывает исходный код, представленный на рисунке 10, к целевому виду, представленному на рисунке 11.

```
#container {
  display: grid;
  grid-template-columns:
    auto minmax(700px, 800px)  minmax(300px, 500px) auto;
  grid-template-rows:
    200px 90px minmax(600px, auto) 200px;
  grid-template-areas:
    "header header header header"
    " . content sidebar . ";
}

#header {
  grid-area: header;
}

#content {
  grid-area: content;
}

#sidebar {
  grid-area: sidebar;
}
```

Рисунок 10 – CSS-код с использованием Grid Layout

```
#container {
  display: -ms-grid;
  display: grid;
  -ms-grid-columns:
    auto minmax(700px, 800px)  minmax(300px, 500px) auto;
  grid-template-columns:
    auto minmax(700px, 800px)  minmax(300px, 500px) auto;
  -ms-grid-rows:
    200px 90px minmax(600px, auto) 200px;
  grid-template-rows:
    200px 90px minmax(600px, auto) 200px;
  grid-template-areas:
    "header header header header"
    " . content sidebar . ";
}

#header {
  -ms-grid-row: 1;
  -ms-grid-column: 1;
  -ms-grid-column-span: 4;
  grid-area: header;
}

#content {
  -ms-grid-row: 2;
  -ms-grid-column: 2;
  grid-area: content;
}

#sidebar {
  -ms-grid-row: 2;
  -ms-grid-column: 3;
  grid-area: sidebar;
}
```

Рисунок 11 – CSS-код после использования postcss-preset-env

Самым популярным и часто используемым плагином для минификации CSS-кода на сегодняшний день является cssnano [40]. Благодаря тому, что из исходного кода удаляются комментарии, пробелы, кавычки, а также применяются некоторые другие подходы, сокращается объём файла стилей до 60 %.

3.1.3 Сборка проекта

Чтобы скомпилировать код на языке препроцессора в код CSS, преобразовать его при помощи постпроцессора с последующим помещением в папку назначения, применяются специальные программы – сборщики проектов.

На сегодняшний день основными игроками этого рынка являются Gulp, Webpack и Grunt. Представленные сборщики работают на платформе Node.js.

Grunt – один из самых ранних представителей сборщиков проектов на Node.js [41]. Поэтому, сегодня существует большое количество документации и статей по использованию данного инструмента.

Конфигурационный файл Gruntfile, отвечающий за правила сборки проекта, носит декларативный характер. В нём подключаются дополнительные плагины, которые отвечают за обработку файлов. Названия файлов могут содержать переменные шаблона, благодаря чему они могут формироваться динамически.

Webpack является самым популярным сборщиком проектов на сегодняшний день. Новая версия данной программы выходит не реже одного раза в год, обладает большим количеством нововведений. Одной из важных функций является встроенное «встряхивание дерева» (tree shaking), благодаря которому уменьшается итоговый файл формата «js» [42].

Конфигурацию сборки можно разделить на два файла – версии для разработки и для публикации. Таким образом, ускоряется процесс разработки и тестирования. Конфигурационные файлы Webpack используют декларативный стиль. Для преобразований файлов используются специальные модули – loaders.

Gulp – современный сборщик проектов, значительно отличающийся от рассмотренных ранее. В основе преобразований данного инструмента лежат трансформации потоков [43]. Благодаря распараллеливанию некоторых операций скорость сборки в большинстве случаев выше, чем у прямых конкурентов.

Файл конфигураций имеет название gulpfile.js. Код файла носит императивный характер, что отличается от Webpack и Grunt. Таким образом, чтобы настроить корректную сборку проекта, разработчику требуются хорошие знания языка JavaScript.

На основе проведённого обзора сборщиков проектов было выполнено их сравнение. Результаты сравнения представлены в таблице 3.

Таблица 3 – Сравнение сборщиков проектов

Критерий сравнения	Grunt	Webpack	Gulp
Количество доступных плагинов для преобразований	Более 2000	Более 8000	Более 5000
Стиль написания конфигурационных файлов	Декларативный	Декларативный	Императивный
Объём пакета сборщика с включёнными зависимостями	Более 4 MB	Более 20 MB	Более 5 MB
Поддержка сообщества	+/- (10 коммитов за год)	+ (более 500 коммитов за год)	+ (64 коммита за год)
Наличие плагинов для компиляции Sass	+	+	+
Наличие плагинов для компиляции Less	+	+	+
Наличие плагинов для компиляции Stylus	+	+	+
Возможность подключения постпроцессора PostCSS	+	+	+

Таким образом, для сборки проектов, соответствующих методологии, представленной в Главе 2, лучше всего подходит сборщик проектов Webpack, ввиду активной поддержки инструмента сообществом и наличия большого количества плагинов для преобразований.

3.2 Программная реализация

Для разделения стилей по категориям были созданы четыре директории:

- 1-base;
- 2-layout;
- 3-blocks;
- 4-skins.

В папке «1-base» хранятся файлы со стилями базовой категории, в папке «2-layout» – стили макета, в директории «3-blocks» – стили многократно используемых блоков, в папке «4-skins» хранятся стили оболочек.

Для хранения всех переменных языка Sass создан отдельный файл _settings.sass. В него были добавлены пользовательские шрифты, основные цвета, используемые на сайте, размер максимальной ширины блока-обёртки, контрольные точки, используемые в медиа-запросах при адаптивной вёрстке.

В категорию «1-base» добавлены стили библиотеки normalize.css, благодаря которым задаётся одинаковое исходное оформление тегов во всех браузерах.

Также создан файл _base.sass с определением фирменного стиля и оформления всего сайта. Пример данного файла представлен на рисунке 12.

```
body
  font-family: $font
  background-color: $body-background
  color: $body-font-color
  transition: all ease .4s

a
  color: $link-color

a:hover
  color: $link-color-hover

a:visited
  color: $link-color-visited

body.Dark-mode
  background-color: #333
```

Рисунок 12 – Стили базовой категории

В качестве основы оформления блоков макета выбрана современная технология – CSS Grid. При использовании адаптивной вёрстки макет подстраивается под различные размеры видимой области.

Главными блоками в данном макете являются: header, description, content, sidebar, footer. Также используется отдельный фиксированный блок для размещения кнопки смены темы оформления – popup.

Стили многократно используемых блоков разделены на отдельные файлы. Каждый файл содержит один блок, соответствующий методологии БЭМ. Благодаря данному подходу исключается возможность создания двух блоков с одинаковыми названиями. Все блоки находятся в директории «3-blocks».

При создании посадочной страницы на основе представленной CSS-методологии были определены следующие блоки: button, description, footer, header, main-text, mobile-menu, sidebar-menu, subheading, theme-mode-button и title.

Для сокращения кода стилей оформления были определены специальные классы-оболочки. Одной из оболочек является «выделенный» текст, представленный на рисунке 13.

```
.Highlighted
  color: $black
  background-color: rgba($black, 0.05)
  font-style: italic
  border-radius: 4px
```

Рисунок 13 – Класс-оболочка для выделенного текста

При смене темы оформления на тёмную меняется цвет текста и цвет фона. Соответствующий CSS-код приведён на рисунке 14.

```
.Dark-mode
  .Highlighted
    color: $white
    background-color: rgba($white, 0.05)
```

Рисунок 14 – CSS-код для смены темы оформления

Для создания адаптивной вёрстки определена специальная оболочка Adaptive, пример которой представлен на рисунке 15.

```
@media #{$media} and (max-width: $width-M)
  .Adaptive
    display: block
    text-align: center
```

Рисунок 15 – Класс-оболочка для адаптивной вёрстки

Одной из возможностей языка Sass является расширение одних классов другими классами-шаблонами. Для этого создан отдельный файл _placeholder-classes.sass, в котором хранятся такие классы. При создании данной посадочной страницы был определён подстановочный класс, приведённый на рисунке 16.

```
%background-img
  display: inline-block
  background-position: center center
  background-repeat: no-repeat
  background-size: cover
```

Рисунок 16 – Подстановочный класс для фонового изображения

Входной точкой, объединяющей все стили проекта, является файл style.sass. Данный файл импортирует все остальные компоненты. Код файла style.sass представлен на рисунке 17.

```
/* General settings */
@import settings
@import placeholder-classes

/* Base styles */
@import 1-base/normalize
@import 1-base/base

/* Layout */
@import 2-layout/content
@import 2-layout/desc
@import 2-layout/footer
@import 2-layout/header
@import 2-layout/sidebar
@import 2-layout/grid-layout
@import 2-layout/popup

/* Blocks */
@import 3-blocks/header
@import 3-blocks/description
@import 3-blocks/title
@import 3-blocks/subheading
@import 3-blocks/main-text
@import 3-blocks/button
@import 3-blocks/sidebar-menu
@import 3-blocks/mobile-menu
@import 3-blocks/footer
@import 3-blocks/theme-mode-button

/* Skins */
@import 4-skins/skins
```

Рисунок 17 – Содержание файла style.sass

3.3 Анализ полученных результатов

В рамках разработки посадочной страницы на основе представленной CSS-методологии были созданы файлы стилей и директории проекта. Помимо этого, были созданы все необходимые файлы для платформы Node.js, установлены зависимости, составлены файлы конфигурации Webpack и разработан файл разметки index.html.

Проект был успешно собран, получены файлы main.css и main.js. После этого, проект был загружен на хостинг статических файлов по адресу https://abvcss.github.io/abvcss-website/. Интерфейсы данного проекта представлены на рисунках 18-23.

Первый экран проекта, включающий «шапку» и описание, представлен на рисунке 18.

Рисунок 18 – Первый экран посадочной страницы

На странице используются блоки для оформления исходного кода. Примеры данных блоков представлены на рисунке 19.

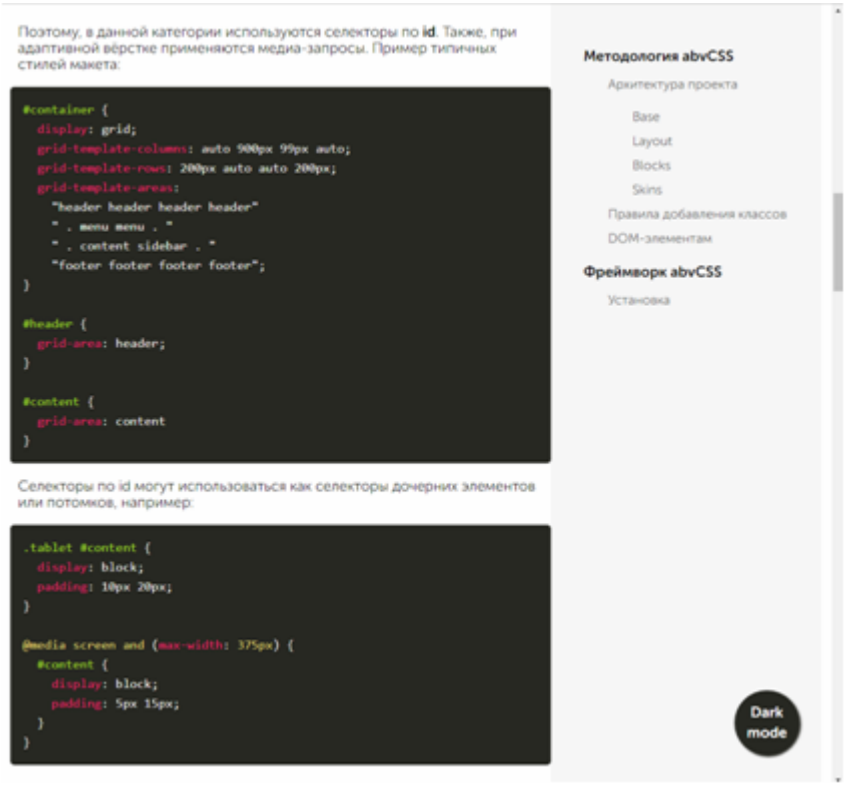

Рисунок 19 – Оформление блоков кода

На рисунке 20 изображён «подвал» посадочной страницы.

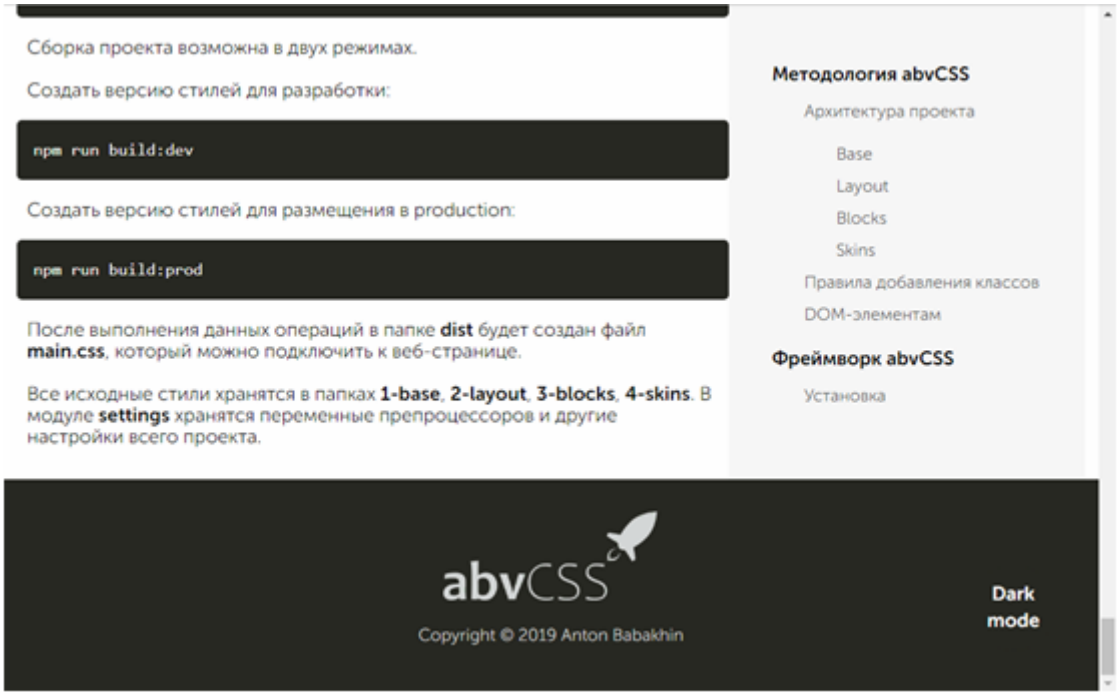

Рисунок 20 – «Подвал» страницы

Для удобства просмотра страницы на мобильных устройствах применяется адаптивная вёрстка. Пример первого экрана при просмотре на мобильном устройстве представлен на рисунке 21.

Рисунок 21 – Адаптивная вёрстка страницы

На рисунке 22 изображён пример адаптации изображений на странице.

Рисунок 22 – Адаптация изображений под размеры экрана

Отличительной чертой используемой в проекте методологии является динамическая смена темы оформления страницы. На рисунке 23 представлена тёмная тема оформления, включающаяся при нажатии кнопки «Dark mode».

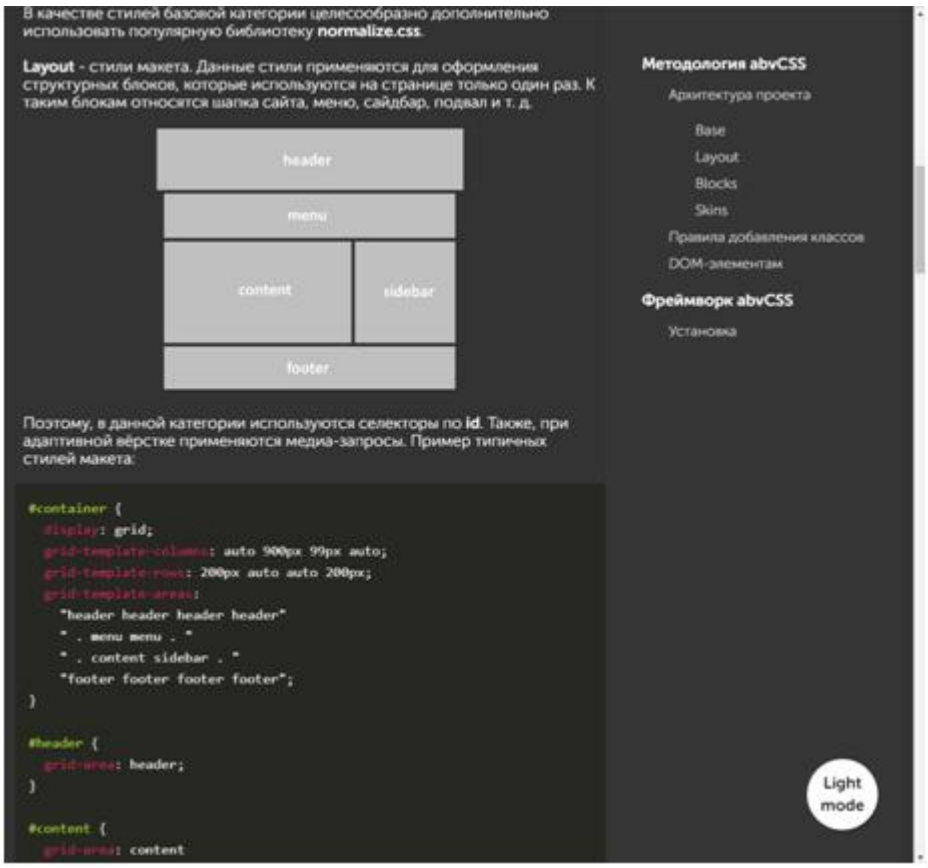

Рисунок 23 – Включённая тёмная тема оформления

Таким образом, в результате создания посадочной страницы, соответствующей исследуемой CSS-методологии, были получены следующие результаты:

- методология позволяет без существенных временных затрат создавать адаптивную вёрстку;
- подтверждена возможность динамической смены темы оформления страницы;
- благодаря разделению стилей макета на отдельные файлы уменьшается вероятность совершения ошибки при использовании селекторов id;
- подтверждена возможность оптимизации страниц сайта под различные браузеры.

Выводы по главе

Были проанализированы основные технологии реализации проекта с применением исследуемой CSS-методологии: препроцессоры, постпроцессоры и сборщики проектов. Таким образом, среди языков описания стилей Less, Sass и Stylus наибольшим количеством преимуществ для реализации данного проекта обладает язык Sass благодаря упрощённому синтаксису и поддержке сообщества. Обработку стилей выполняет постпроцессор PostCSS с плагинами postcss-preset-env и cssnano. Для компиляции стилей Sass и запуска постпроцессора PostCSS используется популярный сборщик проектов Webpack, имеющий более 8000 плагинов.

Все файлы стилей, необходимых для реализации веб-страницы, были распределены по директориям base, layout, blocks и skins.

Были подтверждены теоретические ожидания применения методологии вёрстки, объединяющей BEM, SMACSS и OOCSS, такие как возможность динамической смены тем оформления, применение адаптивной вёрстки, корректное отображение страниц сайта в большинстве современных браузеров.

Заключение

В процессе выполнения данной ВКР были определены проблемы, связанные с разработкой веб-интерфейсов, определены основные подходы к их решению. Был произведён обзор популярных CSS-методологий вёрстки и выполнен анализ научных работ по их применению. На основе закономерностей и общих черт методологий были определены критерии их сравнения, а также выполнен сравнительный анализ. Были даны рекомендации по выбору CSS-методологий в различных проектах, определены преимущества и недостатки каждой методологии. На основе этого была предложена новая методология вёрстки и разработана соответствующая посадочная страница.

Таким образом, в результате выполнения данной научной работы были получены следующие теоретические результаты:

- определены критерии сравнения различных CSS-методологий вёрстки между собой;
- выполнен сравнительный анализ популярных CSS-методологий.

Также были получены следующие практические результаты:

- даны рекомендации по выбору методологий в различных условиях;
- разработана новая CSS-методология, объединяющая принципы БЭМ, OOCSS и SMACSS;
- разработана посадочная страница с применением этой новой методологии.

Анализ полученной посадочной страницы подтвердил такие особенности предложенной методологии, как динамическая смена тем оформления, эффективное применение адаптивной вёрстки и возможность оптимизации под различные браузеры.

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БЭМ – Блок, Элемент, Модификатор.

ВКР – выпускная квалификационная работа.

ИС – информационная система.

AJAX – Asynchronous Javascript and XML.

BEM – Block, Element, Modifier.

CSS – Cascading Style Sheets.

HTML – HyperText Markup Language.

ID – identifier.

OOCSS – Object-Oriented CSS.

SCSS – Sassy CSS.

SDK – Software Development Kit.

SEO – Search Engine Optimization.

SMACSS – Scalable and Modular Architecture for CSS.

СПИСОК ТЕРМИНОВ

Библиотека: набор подпрограмм, используемый для разработки программного обеспечения.

Браузер: программное обеспечение, предназначенное для просмотра ресурсов в сети интернет.

Веб-ресурс: сайт в сети интернет, содержащий и предоставляющий информацию.

Вендор: компания-производитель браузера.

Инкапсуляция: специальная техника для скрытия деталей реализации от пользователя.

Кастомизация: внесение изменений под конкретные условия использования или требования заказчика.

Коммит: фиксация текущего состояния исходного кода в системе контроля версий после внесения изменений.

Компиляция: трансляция программы из исходного языка программирования к низкоуровневому целевому языку.

Конверсия: отношение целевых посетителей сайта к общему числу посетителей, выраженное в процентах.

Лендинг: посадочная страница, ориентированная на решение конкретной задачи, чаще всего продажу товара или услуги.

Минификация: процесс сокращения исходного кода без изменения функциональности, заключающийся в удалении необязательных символов.

Плагин: внешний модуль, подключаемый к программе.

Полифилл: специальный код, добавляющий поддержку устаревших технологий.

Препроцессор: инструмент, выполняющий манипуляции с исходным кодом программы перед компиляцией.

Сборщик: программное обеспечение, обрабатывающее и объединяющее разные однотипные файлы в один файл.

Скрипт: программа или сценарий, запускаемый сервером для автоматического выполнения определённой задачи.

Тег: главный синтаксический элемент языка HTML.

Форм-фактор: требования к размерам устройства, а также другим важным техническим параметрам, в том числе форме и ориентации.

Фреймворк: шаблон для программной платформы, который определяет структуру будущего проекта.

Шаблонизатор: программное обеспечение, преобразующее код на специальном языке разметки в HTML-код.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дорогина, А.С. Front-end разработка / А.С. Дорогина, О.Н. Сафонова, Н.В. Тутова // Телекоммуникации и информационные технологии. – 2015. − № 2. – С. 69-72.
2. Белоусова, С.А. Анализ подходов к созданию пользовательского интерфейса / С.А. Белоусова, Ю.И. Рогозов // Известия ЮФУ. Технические науки. – 2014. − № 6. – С. 142-148.
3. Чернов, В.В. К проблеме разработки веб-интерфейсов / В.В. Чернов // Фундаментальные исследования. – 2012. − № 11. – С. 463-465.
4. Биктимиров, Р.Р. Современные инструменты front-end разработки / Р.Р. Биктимиров, А.Б. Джемалетдинов // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. – 2016. − № 3. – С. 63-69.
5. A mostly reasonable approach to CSS and Sass. [Электронный ресурс]. – Режим доступа: https://github.com/airbnb/css (дата обращения 23.04.2019).
6. Методология / БЭМ. [Электронный ресурс]. – Режим доступа: https://ru.bem.info/methodology/ (дата обращения 23.04.2019).
7. Селиванова, Е.А. Методологии верстки: БЭМ, AMCSS, OOCSS, Atomic CSS, OPOR, MCSS, SMACSS, FUN, DoCSSa [Электронный ресурс] / Е.А. Селиванова, И.В. Левина // 4front. − 2015. − Режим доступа: http://html5.by/blog/bem-amcss-oocss-atomiccss-opor-mcss-smacss-fun-docssa-video/ (дата обращения 23.04.2019).
8. Snook J. Scalable and Modular Architecture for CSS: A Flexible Guide to Developing Sites Small and Large. – Snook Ca Web Development, 2012. – 4 p.

9. Normalize.css: Make browsers render all elements more consistently. [Электронный ресурс]. – Режим доступа: https://necolas.github.io/normalize.css/ (дата обращения 23.04.2019).
10. CSS Tools: Reset CSS. [Электронный ресурс]. – Режим доступа: https://meyerweb.com/eric/tools/css/reset/ (дата обращения 23.04.2019).
11. Способы организации CSS-кода. [Электронный ресурс]. – Режим доступа: https://habrahabr.ru/post/256109/ (дата обращения 23.04.2019).
12. Catanzariti, P. BEM and SMACSS: Advice From Developers Who've Been There [Электронный ресурс] / P. Catanzariti // Sitepoint. − 2015. − Режим доступа: https://www.sitepoint.com/bem-smacss-advice-from-developers/ (дата обращения 23.04.2019).
13. Object Oriented CSS. [Электронный ресурс]. – Режим доступа: https://github.com/stubbornella/oocss/wiki (дата обращения 23.04.2019).
14. Правильный CSS: OOCSS, SMACSS, BEM и SASS. [Электронный ресурс]. – Режим доступа: https://medium.com/@stepanovv.ru/правильный-css-oocss-smacss-bem-и-sass-49351a119283 (дата обращения 23.04.2019).
15. Введение в объектно-ориентированный CSS (OOCSS). [Электронный ресурс]. – Режим доступа: https://webformyself.com/vvedenie-v-obektno-orientirovannyj-css-oocss/ (дата обращения 23.04.2019).
16. Modules – Container Objects. [Электронный ресурс]. – Режим доступа: https://github.com/stubbornella/oocss/wiki/Module (дата обращения 23.04.2019).
17. Захаров, В.А. Современные подходы к верстке веб-проектов / В.А. Захаров // Ползуновский альманах. – 2012. − № 2. – С. 183-184.
18. Sikora, P. Professional Css3 / P. Sikora. – Packt Publishing Ltd, 2016. – 362 p.
19. Leino, M. Coding Standards in Web Development [Электронный ресурс] / M. Leino // Metropolia. − 2016. − Режим доступа:

https://www.theseus.fi/bitstream/handle/10024/112473/leino_martti.pdf?sequence=1 (дата обращения 23.04.2019).

20. Atomic CSS. [Электронный ресурс]. – Режим доступа: https://acss.io (дата обращения 23.04.2019).
21. Acss-io / atomizer. [Электронный ресурс]. – Режим доступа: https://github.com/acss-io/atomizer (дата обращения 23.04.2019).
22. Watts, L. Mastering Sass / L. Watts. – Packt Publishing Ltd, 2016. – 318 p.
23. Atomic Design by Brad Frost. [Электронный ресурс]. – Режим доступа: http://atomicdesign.bradfrost.com (дата обращения 23.04.2019).
24. Охрименко, А.А. CSS методологии от О до Б [Электронный ресурс] / А.А. Охрименко // Web Standards Days. − 2016. − Режим доступа: https://wsd.events/2016/10/01/pres/css-methodologies.pdf (дата обращения 23.04.2019).
25. Mobile share of website visits worldwide 2018. [Электронный ресурс]. – Режим доступа: https://www.statista.com/statistics/241462/global-mobile-phone-website-traffic-share/ (дата обращения 23.04.2019).
26. Official Google Webmaster Central Blog: Rolling out mobile-first indexing. [Электронный ресурс]. – Режим доступа: https://webmasters.googleblog.com/2018/03/rolling-out-mobile-first-indexing.html (дата обращения 23.04.2019).
27. Шакирова, Ю.К. Первостепенность дизайна для мобильных устройств / Ю.К. Шакирова, А.Е. Маденова, Н.К. Савченко, Г.Б. Абилдаева // Дистанционное и виртуальное обучение. – 2015. − № 4. – С. 103-110.
28. CSS - Медиа запросы (media queries). [Электронный ресурс]. – Режим доступа: https://itchief.ru/lessons/html-and-css/css-media-queries (дата обращения 23.04.2019).
29. Резанович, В.Э. Лендинг: эффективное продвижение бренда / В.Э. Резанович // Всероссийская научно-практическая конференция «Череповецкие научные чтения – 2017». – 2018. – С. 114-116.

30. The Data Behind Landing Page Trends in 2018. [Электронный ресурс]. – Режим доступа: https://medium.com/unbounce-marketing/infographic-the-data-behind-landing-page-trends-in-2018-b4b6787a40dd (дата обращения 23.04.2019).
31. What makes a project big. [Электронный ресурс]. – Режим доступа: https://softwareengineering.stackexchange.com/questions/17177/what-makes-a-project-big (дата обращения 23.04.2019).
32. The State of the Octoverse. [Электронный ресурс]. – Режим доступа: https://octoverse.github.com/projects.html (дата обращения 23.04.2019).
33. Разбираемся в отличиях препроцессоров CSS [Электронный ресурс]. – Режим доступа: https://xakep.ru/2014/05/18/razbiraemsya-v-otlichiyah-preprotsessorov-css/ (дата обращения 23.04.2019).
34. Символ «&» в SassScript | Документация по Sass на русском языке [Электронный ресурс]. – Режим доступа: https://sass-scss.ru/documentation/sassscript/&_v_sassscript.html (дата обращения 23.04.2019).
35. Getting started | Less.js [Электронный ресурс]. – Режим доступа: http://lesscss.org/ (дата обращения 23.04.2019).
36. Expressive, dynamic, robust CSS – expressive, robust, feature-rich CSS preprocessor [Электронный ресурс]. – Режим доступа: http://stylus-lang.com/ (дата обращения 23.04.2019).
37. Post & Pre Processing CSS / Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/434098/ (дата обращения 15.03.2019).
38. Варганов Д.А. Обзор постпроцессоров css и их задач // Альманах научных работ молодых ученых Университета ИТМО – 2018. – Т. 7. – С. 82-83
39. Convert modern CSS into something browsers understand [Электронный ресурс]. – Режим доступа: https://github.com/csstools/postcss-preset-env (дата обращения 23.04.2019).

40. A modular minifier, built on top of the PostCSS ecosystem [Электронный ресурс]. – Режим доступа: https://github.com/cssnano/cssnano (дата обращения 23.04.2019).
41. Разбираемся с популярными системами сборки фронтенда [Электронный ресурс]. – Режим доступа: https://xakep.ru/2014/05/18/popular-front-end-systems/ (дата обращения 23.04.2019).
42. Tree Shaking | webpack [Электронный ресурс]. – Режим доступа: https://webpack.js.org/guides/tree-shaking/ (дата обращения 23.04.2019).
43. JavaScript and Gulpfiles [Электронный ресурс]. – Режим доступа: https://gulpjs.com/docs/en/getting-started/javascript-and-gulpfiles (дата обращения 23.04.2019).