МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика

(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование

(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка алгоритма управления внешним видом и языком интерфейса личного кабинета пользователя»

Обучающийся Д.А. Цветков _____
(И.О. Фамилия) (личная подпись)

Руководитель М.А. Тренина
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент Т.С. Якушева

Тольятти 2023

# Аннотация

Выпускная квалификационная работа посвящена следующей теме: «Разработка алгоритма управления внешним видом и языком интерфейса личного кабинета пользователя». Задачи, указанные в названии темы, подразумевают под собой реализацию и внедрение динамической персонализации в веб-приложение, что, в свою очередь, является актуальным направлением в сфере веб-разработки и электронной коммерции. Целью данной работы является создание алгоритма, позволяющего реализовывать и внедрять компоненты динамической персонализации на веб-сайт.

Для достижения заданной цели были выделены следующие задачи:

1. провести анализ существующих инструментов редактирования дизайна веб-сайта;
2. произвести декомпозицию общей задачи на подзадачи и выделить отдельные компоненты на каждую из них;
3. описать алгоритмы реализации для каждого из полученных компонентов;
4. выбрать подходящие инструменты разработки;
5. привести примеры реализации каждого описанного алгоритма.

В первом разделе данной выпускной квалификационной работы представлены общие теоретические сведения о каждом из выделенных компонентов, даны их общие представления, а также сформулированы алгоритмы реализаций. Второй раздел посвящен описанию наиболее эффективных инструментов разработки и выбору из них наиболее подходящих для выполнения поставленных задач. В третьем разделе описываются примеры реализаций компонентов с использованием алгоритмов, сформулированных ранее.

## Abstract

This graduation work is devoted to the topic: “Development of an algorithm for managing the appearance and language of the personal account interface”. The tasks that are stated in the title of the work imply the implementation of dynamic personalization in a web application. As of today, this area of study is considered to be one of the most important and demanded in the field of current web development and e-commerce. The aim of the work is to create an algorithm that would allow to implement dynamic personalization components on a website.

To achieve the main goal, the following tasks were identified:

1. Analyze existing website design editing tools;
2. Decompose the main task into subtasks and single out individual components for each of them;
3. Create implementation algorithms for each of the obtained components;
4. Choose suitable development tools;
5. Give implementation examples of each described algorithm.

The first chapter of the work gives general representation and theoretical information about each of the selected components. Algorithms for each component are also given in this chapter. The second chapter is devoted to the mentioning of the most effective development tools to date. It also suggests the most effective of them. The third chapter is aimed to give implementation examples for each of the obtained components using development tools that have been chosen in the previous chapter.

Содержание

# Введение

В современном быстро развивающемся мире всё большее количество пользователей открывают для себя глобальную сеть интернет. В связи с этим, наличие удобного и интуитивно понятного дизайна для удовлетворения нужд большинства стало неотъемлемой частью каждого современного веб-сайта. То, как мы взаимодействуем с интерфейсом веб-сайта, может оказать существенное влияние на наш опыт работы, и, следовательно, важно убедиться, что эти интерфейсы спроектированы интуитивно понятными и удобными для пользователя. Одними из ключевых факторов, которые могут повлиять на работу пользователя, являются внешний вид и язык интерфейса.

Основным направлением в данной выпускной квалификационной работе является изучение методологий по разработке динамической персонализации веб-сайта. Изучение и анализ основных практик и методов разработки динамической персонализации позволят разработать наиболее гибкий и адаптируемый алгоритм. Алгоритм будет разработан таким образом, чтобы динамически адаптироваться к потребностям пользователя, гарантируя, что интерфейс будет простым в использовании и адаптирован к их требованиям, позволяя выполнять свои задачи эффективно и с минимальными усилиями.

Целью данной выпускной квалификационной работы является разработка алгоритма, позволяющего создать завершенное и оптимизированное с точки зрения быстродействия решение по внедрению динамической персонализации на веб-сайт.

Основной задачей данной работы является достижение всех необходимых критериев, связанных с определением динамической персонализации.

Для достижения поставленной цели, выделим следующие задачи:

– анализ существующих инструментов редактирования дизайна веб-сайта как пользователем, так и администратором. Такими

инструментами могут являться компоненты выбора темы, дизайна и языка для пользователя, а также поле выбора тематического дизайна для администратора;

- реализация компонента, считывающего текущий домен. Данный компонент будет динамически изменять дизайн и наполнение веб-сайта в зависимости от полученного домена;
- внедрение функционала, анализирующего каждого отдельного пользователя, его поведение, предпочтения и изменяющего или добавляющего определенные компоненты, основываясь на этих данных.

В целом ожидается, что разработка данного алгоритма позволит реализовывать наиболее эффективные, практичные и корректные с точки зрения оптимизации и соответствия нормам решения.

Первый раздел данной работы посвящен созданию алгоритмов, описывающих реализацию компонентов динамической персонализации. В нем даны основные теоретические сведения и общие представления о каждом из описываемых компонентов. Каждый из алгоритмов выполнен в виде пошагового описания выполнения задачи.

Во втором разделе были изучены существующие инструменты разработки. Из них были выбраны наиболее актуальные и подходящие под нужды текущего веб-приложения.

В третьем разделе были представлены реализации всех компонентов, описанных в первом разделе. Реализация представляет из себя пошаговое выполнение каждого из алгоритмов, сформулированных в первом разделе, с демонстрацией полученных результатов в виде текстового описания и скриншотов.

Результатом работы являются сформулированные алгоритмы реализации поставленных задач, а также примеры разработок компонентов по этим алгоритмам с использованием наиболее подходящих инструментов разработки.

# 1 Анализ динамической персонализации, теоретические основы. Разработка алгоритма реализации

## 1.1 Введение в динамическую персонализацию

Динамическая персонализация представляет собой подход к управлению взаимодействием между системами и пользователями на основе анализа и адаптации к их предпочтениям, интересам, поведению и контексту. В последние годы динамическая персонализация стала широко применяться в различных областях, таких как интернет-реклама, электронная коммерция, рекомендательные системы, социальные сети и т.д., с целью повышения эффективности взаимодействия между системами и пользователями. Также к динамической персонализации относится реализация и внедрение на веб-сайт динамического дизайна.

В контексте веб-дизайна и разработки, термин "динамический дизайн" относится к созданию интерактивных и анимированных элементов на веб-сайте, которые меняются в режиме реального времени и реагируют на действия пользователя или другие события. Динамический дизайн может включать различные техники, такие как анимации, переходы, изменение цветов, форм, размеров и т. д., которые создают эффект движения, изменения и взаимодействия на веб-сайте [9].

Понятие динамической персонализации представляет из себя довольно обширный спектр задач и идей, каждые из которых имеют свои особенности в плане реализации и представления. Для более конкретного и точного описания алгоритма реализации данной задачи было решено прибегнуть к декомпозиции обобщенного определения динамической персонализации на более конкретные и узконаправленные подзадачи. Таким образом, в данной квалификационной работе основными направлениями по изучению и исследованию области динамической персонализации являются следующие темы:

– возможность редактирования темы веб-сайта пользователем;
– возможность изменения языка на веб-сайте;
– тематические, сезонные и праздничные темы, настраиваемые администратором;
– динамическое изменение наполнения и дизайна веб-сайта в зависимости от текущего домена;
– анализ цифрового следа пользователя для отображения динамической персонализации.

В данном разделе описаны основные теоретические сведения по каждому из перечисленных пунктов, а также подробно сформулированы структуры алгоритмов, позволяющих реализовывать каждую из перечисленных задач.

Формулировка каждого алгоритма включает в себя обобщенное представление текущего компонента, а также этапы реализации данного компонента. Алгоритмы будут рассмотрены в контексте реализации каждого компонента отдельно, а также взаимосвязи и взаимодействия этих компонентов в рамках всей системы динамической персонализации.

## 1.2 Алгоритм разработки компонента, изменяющего темы веб-сайта

Возможность редактирования темы веб-сайта пользователем предполагает наличие функционала, который позволит пользователям настраивать внешний вид веб-сайта в соответствии с их предпочтениями и представлениями о дизайне. Это может включать следующие аспекты:

– цветовая схема: Пользователи могут иметь возможность выбирать цветовые схемы для веб-сайта из предварительно заданных вариантов или настраивать собственные цвета для различных элементов дизайна, таких как фон, заголовки, текст и т. д.;

– шрифты: Пользователи могут иметь возможность выбирать шрифты для текстовых элементов на веб-сайте. Это может включать выбор из заранее заданных вариантов или настройку собственных шрифтов;
– макеты: Пользователи могут иметь возможность выбирать различные макеты или компоновки страницы, такие как одна колонка, две колонки, сайдбары (меню, расположенные по сторонам от основного блока информации) и т. д., в зависимости от их предпочтений;
– графика и изображения: Пользователи могут иметь возможность загружать и настраивать собственные графические элементы или изображения на веб-сайте, такие как логотипы, фоновые изображения, иллюстрации и т. д.;
– стили и эффекты: Пользователи могут иметь возможность выбирать или настраивать стили и эффекты для элементов дизайна, такие как кнопки, ссылки, формы и т. д., чтобы придать им определенный вид или взаимодействие.

Реализация возможности редактирования темы веб-сайта пользователем обычно основывается на концепции "визуального редактора" или "тулкита для настройки внешнего вида". Это инструмент позволяет пользователям изменять внешний вид веб-сайта в режиме реального времени, без необходимости внесения изменений в исходный код или использования сложных технических навыков [5].

Реализация данного функционала включает в себя следующие основные этапы.

1. Создание диаграммы: Разработка диаграммы компонента, описывающая структуру компонента, принцип работы или его взаимодействие с другими компонентами в рамках всего приложения.

2. Создание интерфейса редактора: Разработка пользовательского интерфейса (UI) для редактора, который будет интуитивно понятен и удобен в использовании. Это может включать размещение на странице различных

инструментов и элементов управления, таких как кнопки, слайдеры, окна выбора цветов и т.д. [4]

3. Настройка опций редактирования: «Определение доступных опций редактирования, таких как изменение цветов, шрифтов, фонового изображения, расположения элементов и т. д. Эти опции могут быть представлены в виде настраиваемых параметров, которые пользователь может легко изменять в интерфейсе редактора» [19].

4. Тестирование и оптимизация: Проведение тестирования редактора на различных устройствах и браузерах, чтобы удостовериться в его корректной работе и удобстве использования. Оптимизация процесса редактирования для максимальной производительности и быстрой загрузки веб-сайта.

Остановимся на данных этапах и разберем их более подробно.

Диаграмма компонента необходима для более упрощенного и понятного представления данного компонента, что, в свою очередь, позволит упростить и ускорить процесс разработки.

Создание интерфейса редактора веб-сайта может быть реализовано различными способами, в зависимости от выбранных технологий и инструментов.

Для начала необходимо определить функциональные требования данного редактора – какие конкретно функции должен предоставлять интерфейс. Такими функциями могут являться редактирование и удаление контента (добавление и удаление темы), управление настройками темы и т.д.

Затем следует проектирование макета и дизайна пользовательского интерфейса (UI). При разработке стоит учитывать удобство и интуитивность, при этом обращая внимание на доступность и соответствие общему стилю.

Следующим шагом является выбор технологий и инструментов разработки. Это может включать в себя выбор подхода разработки, выбор фреймворка, библиотеки, языка программирования и т.д.

Следующий этап – начало разработки пользовательского интерфейса. Сперва необходимо выделить и создать компоненты пользовательского интерфейса, такие как формы, кнопки, модальные окна, деревья контента и т.д., с использованием выбранных технологий и инструментов. После чего идет реализация логики взаимодействия между компонентами, включая обработку событий, валидацию данных, взаимодействие с бэкендом и обновление интерфейса в режиме реального времени.

По завершении разработки важным этапом является проведение тестирования и отладки полученного интерфейса, чтобы убедиться в его работоспособности, функциональности и безопасности. Данное тестирование позволит выявить и исправить возможные ошибки, улучшить пользовательский опыт и внести необходимые корректировки.

### 1.3 Алгоритм разработки компонента, изменяющего язык веб-сайта

Функция изменения языка на веб-сайте предоставляет возможность пользователям выбрать предпочтительный язык интерфейса или контента на веб-сайте. Это может быть важной функцией, особенно для многоязычных сайтов или сайтов, которые предоставляют контент для глобальной аудитории [17].

Теоретически, функция изменения языка на веб-сайте может быть реализована разными способами. Один из распространенных подходов – это предоставление списка доступных языков в виде выпадающего меню или другого элемента интерфейса, который пользователь может выбрать. При выборе языка, сайт переводится на выбранный язык, если доступен соответствующий перевод контента.

Компонент, отвечающий за изменение языка на веб-сайте, зачастую состоит из программного кода и интерфейсных элементов, которые позволяют пользователям выбирать предпочитаемый язык. Компонент следует располагать в верхних углах экрана версий для персональных компьютеров и

в верхней части страницы на мобильных устройствах. Он должен обладать достаточной детализацией и гибкостью, чтобы отвечать всем потребностям пользователей.

Данный компонент может включать в себя следующие элементы:

- интерфейс выбора языка: это может быть выпадающий список, переключатель, кнопки или другие элементы интерфейса, которые позволяют пользователям выбирать язык, на котором они хотят видеть контент сайта;
- обработчик событий: это программа или функция, которая отвечает за обработку выбора пользователя и изменение языка на сайте в соответствии с выбранным значением;
- языковые файлы или ресурсы: это наборы текстовых файлов или ресурсов, которые содержат локализованный контент на разных языках. Компонент может использовать эти файлы или ресурсы для загрузки и отображения контента на выбранном языке;
- автоматическое определение языка: в некоторых случаях компонент может иметь функциональность автоматического определения языка пользователя на основе его местоположения, настроек браузера или других параметров. Это может помочь упростить процесс выбора языка для пользователей;
- логика обновления контента: компонент также может быть ответственным за динамическое обновление контента на сайте при изменении языка. Это может включать перевод текстов, изменение форматирования дат, времени и других языковых особенностей контента.

Далее перейдем к алгоритму разработки данного компонента, который включает в себя следующие шаги.

Шаг 1: Планирование и анализ:

- определение требований и функциональности компонента, связанного с изменением языка веб-сайта. Например, решение о том,

какие языки поддерживать, какие элементы веб-сайта должны быть переведены, какие языковые флаги или другие элементы интерфейса использовать для выбора языка и т. д.;

- создание диаграммы компонента для более упрощенного и понятного представления;
- проектирование пользовательского интерфейса компонента, включая взаимодействие с пользователем при выборе языка;
- определение технологий и инструментов, которые будут использоваться для реализации компонента, таких как JavaScript-библиотеки или фреймворки для разработки фронтенда, бэкенд-серверы для обработки языковых настроек, базы данных для хранения переведенных строк и т. д.

Шаг 2: Разработка фронтенда:

- создание интерфейса компонента на фронтенде с использованием выбранных технологий и инструментов. Это может включать создание языковых флагов, выпадающего списка или других элементов интерфейса для выбора языка, а также механизмов для отображения контента на выбранном языке;
- разработка логики компонента, включая обработку событий, связанных с выбором языка, обновление интерфейса и взаимодействие с бэкендом для обработки языковых настроек.

Шаг 3: Разработка бэкенда:

- создание серверной логики для обработки выбора языка от клиента и установки языковых настроек на сервере. Это может включать реализацию API-методов или других механизмов взаимодействия между фронтендом и бэкендом;
- разработка механизмов для хранения и управления переводами, таких как база данных или файлы переводов. Это может включать создание системы управления переводами, а также механизмов для добавления, редактирования и удаления переведенных строк.

Шаг 4: Тестирование:

- проведение тестирования компонента на различных языках и устройствах, проверка корректности работы и соответствия требованиям;
- проведение тестирования взаимодействия между фронтендом и бэкендом, чтобы убедиться в правильной передаче данных о выбранном языке и обработке на сервере;
- тестирование компонента на различных сценариях использования, таких как выбор языка на разных страницах сайта, смена языка в процессе просмотра контента, проверка отображения переведенных строк и т. д.

Таким образом, был сформулирован алгоритм разработки компонента, позволяющего изменить язык веб-сайта, который может быть адаптирован в соответствии с конкретными требованиями и техническими деталями проекта.

## 1.4 Алгоритм разработки компонента, позволяющего администратору задавать тематические, сезонные и праздничные темы

Тематические, сезонные и праздничные темы на веб-сайтах – это дизайн, оформление и контент, связанные с определенной тематикой, временным сезоном или праздником, которые меняются на сайте в соответствии с конкретным событием, праздником или временным периодом. Эти темы могут быть использованы для создания атмосферы, вовлечения пользователей, повышения интереса и улучшения пользовательского опыта на веб-сайте [6].

Тематические темы на веб-сайтах могут быть связаны с определенным событием, тематикой или интересами аудитории. Например, это могут быть темы, связанные с определенным праздником, сезоном года, культурой, спортом, музыкой, фильмами, наукой, искусством и т. д. Такие темы могут использоваться для создания уникального дизайна сайта, размещения

соответствующего контента, привлечения внимания и пропаганды определенной тематики.

Сезонные темы на веб-сайтах меняются в соответствии с временными сезонами года, такими как весна, лето, осень и зима. Это может включать изменение цветовой палитры, изображений, графики, а также контента, связанного с текущим сезоном. Сезонные темы могут помочь создать атмосферу, соответствующую времени года, и создать более актуальное и привлекательное визуальное оформление сайта [11].

Праздничные темы на веб-сайтах связаны с определенными праздниками, такими как Рождество, Новый год, День святого Валентина, Хэллоуин, Пасха и другие. Праздничные темы могут включать соответствующие изображения, графику, декорации, акции или специальные предложения, связанные с праздником. Они могут помочь создать праздничную атмосферу на веб-сайте, привлечь внимание посетителей и стимулировать их к дополнительной активности на сайте, такой как совершение покупок или участие в акциях.

Доверение в настройке и установке тематических тем на веб-сайтах обычно остается в руках администраторов. Администраторы веб-сайтов обладают правами доступа, которые позволяют им изменять дизайн и функциональность сайта, включая выбор и установку тематических тем.

Решение об ограничении прав на установку может быть обосновано несколькими факторами. Во-первых, тематические темы могут содержать специфические дизайнерские элементы, функциональные модули или плагины, которые требуют экспертизы в области веб-разработки или дизайна. Администраторы, как правило, имеют необходимые знания и опыт настройки и установки определенных тем на веб-сайте.

Во-вторых, ограничение прав доступа к установке может быть мерой безопасности. Неправильная установка тем или использование тем из не доверенных источников может представлять угрозу безопасности веб-сайта, так как темы могут содержать вредоносный код или иметь уязвимости,

которые могут быть использованы злоумышленниками. Ограничение прав доступа помогает предотвратить возможные угрозы безопасности и обеспечить надлежащий уровень защиты веб-сайта.

Компонент, предоставляющий возможность задавать тематические, сезонные и праздничные темы, зачастую представляет собой набор настроек, которые позволяют администратору веб-сайта изменять внешний вид и оформление в зависимости от текущего сезона, праздника или других событий. Реализацией компонента может служить создание определенного, доступного лишь для администратора интерфейса, предоставляющего возможность выбора цветовых схем, изображений фона, элементов дизайна, а также настройку видимости и расположения различных элементов пользовательского интерфейса [12].

Сформулируем алгоритм разработки данного компонента.

1. Определить четкие требования к компоненту и его функционал. Прежде всего необходимо выделить определенные типы тем, которые будут поддерживаться. Таковыми могут быть сезонные, праздничные, корпоративные и т.д. Затем стоит заострить внимание на определении конкретного списка тем, которые должны быть доступны на сайте. Также необходимо уточнить, какие именно параметры каждой из тем можно задавать, например, цветовые схемы, шрифты, фоновые изображения и т.д.

2. Разработать диаграмму, четко описывающую весь функционал компонента, а также принцип его взаимодействия с другими компонентами и системой в целом.

3. Разработать дизайн интерфейса для выбора темы. В зависимости от требований к компоненту, необходимо разработать дизайн интерфейса для выбора темы. Это может быть, например, выпадающий список, переключатель или любой другой элемент управления.

4. Написать код для компонента. Необходимо написать код компонента, позволяющего загрузить и применить выбранную тему на веб-сайт. Это может быть реализовано с помощью таких инструментов разработки

как CSS, JavaScript и других технологий. Стоит также отметить, что доступ к данному функционалу должен быть закреплен непосредственно за администратором, данная особенность реализуется именно на этом этапе.

5. Протестировать компонент. После разработки необходимо провести тестирование и убедиться в корректной работе полученного решения, а также убедиться в соответствии требованиям и техническим деталям проекта.

6. Документировать компонент. После полного завершения разработки необходимо составить подробную и понятную документацию по работе с разработанным компонентом, которая будет предназначена для работы исключительно со стороны администратора.

7. Интегрировать компонент на веб-сайт. Наконец, необходимо интегрировать компонент на веб-сайт. Это может быть сделано путем добавления компонента на страницу с помощью HTML и JavaScript. После интегрирования стоит провести дополнительное тестирование, за счет которого можно будет убедиться в корректном проведении интегрирования.

## 1.5 Алгоритм разработки компонента, динамически изменяющего наполнение и дизайна веб-сайта в зависимости от текущего домена

Динамическое изменение наполнения и дизайна веб-сайта в зависимости от текущего домена – это подход, который позволяет менять содержимое и внешний вид веб-сайта на основе домена, с которого пользователь обращается на сайт. Данная функция может быть полезна в различных ситуациях, таких как создание мультиязычных сайтов, адаптация содержимого под разные регионы или целевые аудитории, или предоставление различных версий сайта для разных брендов или поддоменов [23].

Поддомен – это часть доменного имени, расположенная перед основным доменным именем и разделенная от него точкой. Он добавляется перед

основным доменом и может быть использован для создания отдельных разделов, подразделов или подсайтов в рамках основного домена.

Например, в домене "example.com" "www" является поддоменом. Также могут быть созданы другие поддомены, такие как "blog.example.com", "shop.example.com" или "support.example.com", каждый из которых может иметь отдельное содержание, функциональность или дизайн.

Поддомены могут использоваться для разделения различных типов контента, предоставления доступа к разным сервисам или приложениям, организации многоязычных сайтов или для других целей, в зависимости от потребностей веб-сайта или организации. Они могут быть созданы и настроены на уровне DNS (системы доменных имен) и веб-сервера для правильной маршрутизации запросов на соответствующий контент или приложение.

Данная технология требует тщательного планирования и реализации. Необходимо убедиться, что контент и дизайн веб-сайта корректно адаптируются под разные домены и не нарушают пользовательский опыт. Также для создания оптимального решения необходимо определить правила и логику работы динамического дизайна на основе домена, что, в свою очередь, может потребовать тесного взаимодействия между разработчиками, дизайнерами и бизнес-заказчиками.

Реализация компонента представляет собой программный модуль, позволяющий динамически изменять наполнение и дизайн веб-сайта в зависимости от текущего домена. Это достигается путем определения правил, согласно которым компонент будет производить замену определенных элементов на странице веб-сайта, в том числе текстов, изображений, ссылок, цветовых схем и других атрибутов.

Также, распространенной практикой является закрепление за веб-сайтом сразу нескольких доменов или версий веб-сайта, которые могут коренным образом отличаться друг от друга своим наполнением. Данный компонент позволит упростить и оптимизировать переход между этими версиями с точки

зрения быстродействия и удобства, а также даст возможность автоматизировать процесс поддержки нескольких версий веб-сайта.

Далее рассмотрим алгоритм реализации данного компонента по шагам.

1. Получение текущего домена веб-сайта. Прежде всего необходимо разработать функционал, который будет осуществлять переход между доменами веб-сайта. Для этого можно разработать отдельное меню с ссылками, либо кнопками, при нажатии на которые будет происходить изменение домена. Затем необходимо реализовать получение текущего URL-адреса (домена) веб-сайта при его изменении. Для этого используется соответствующий функционал языка программирования, который возвращает строку, содержащую URL-адрес текущей страницы. Эту строку можно разбить на отдельные части, включая доменное имя, с помощью специальных функций парсинга URL.

2. Определение соответствующего контента и дизайна для домена. На этом шаге мы определяем, какой контент и дизайн должны быть отображены для данного домена. Для этого нужно предварительно определить контент и дизайн для каждого домена, на который может быть направлен пользователь. Эти данные можно хранить в базе данных или в конфигурационных файлах.

3. Загрузка контента и дизайна для текущего домена. На этом шаге мы загружаем контент и дизайн для текущего домена. Если данные хранятся в базе данных, то мы должны выполнить запрос на получение данных из базы данных. Если данные хранятся в конфигурационных файлах, то нужно загрузить соответствующий файл.

4. Обновление веб-сайта с новым контентом и дизайном. После загрузки контента и дизайна для текущего домена, необходимо обновить веб-страницу, чтобы отобразить новый контент и дизайн. Для этого нужно изменить соответствующие элементы HTML-разметки и стили, используя язык программирования, который используется на серверной стороне (например, PHP или Python).

5. Создание диаграммы компонента и реализация программного кода.

6. Тестирование и отладка. Важным шагом является тестирование и отладка разработанного компонента. Необходимо убедиться, что компонент работает должным образом на всех поддерживаемых доменах и что изменения в контенте и дизайне происходят корректно. При обнаружении каких-либо ошибок или недочетов, необходимо исправить их и повторно протестировать компонент.

7. Развертывание компонента на сервере. После успешного тестирования и отладки компонента, его можно развернуть на сервере, где будет запущен веб-сайт. Необходимо убедиться, что компонент работает должным образом на реальном сервере и не вызывает проблем с производительностью веб-сайта.

## 1.6 Алгоритм разработки компонента, анализирующего цифровой след пользователя для отображения динамической персонализации

Динамическая персонализация – это процесс настройки контента, предложений или функциональности в реальном времени в зависимости от данных о поведении, предпочтениях и характеристиках конкретного пользователя или группы пользователей. Она основана на сборе и анализе данных о взаимодействии пользователя с веб-сайтом, приложением или другим цифровым продуктом, для отображения более актуальной информации [22].

Динамическая персонализация использует различные типы данных, такие как история посещений, демографические данные, данные о покупках, геолокационные данные и другие, для определения наилучшего способа представления контента или предложения конкретному пользователю. Примеры динамической персонализации включают:

– рекомендации товаров на основе предыдущих покупок: интернет-магазин может использовать данные о предыдущих покупках пользователя, чтобы предлагать персонализированные рекомендации товаров на основе его предпочтений и интересов. Если пользователь ранее покупал книги определенного автора, система динамической персонализации может предложить ему другие книги того же автора или подобные книги в том же жанре;
– персонализированные приветствия на основе геолокации: веб-сайт может использовать данные о геолокации пользователя, чтобы приветствовать его на сайте на его родном языке или предлагать информацию о ближайших достопримечательностях или событиях в его регионе [10];
– персонализированные предложения на основе поведения: система электронной коммерции может анализировать поведение пользователя на сайте, такое как переход на страницы товаров, время, проведенное за просмотром товаров, нажатие на ключевые кнопки, добавление в избранное или корзину, и предлагать персонализированные скидки или специальные предложения для стимулирования завершения покупки;
– динамические рекламные баннеры: рекламный баннер на сайте или в приложении может быть динамически адаптирован в зависимости от интересов и предпочтений пользователя. Если пользователь ранее интересовался спортивными товарами, система динамической персонализации может показать ему рекламу спортивного снаряжения или актуальных спортивных мероприятий.

Компонент анализа цифрового следа представляет из себя программный код, анализирующий цифровой след пользователя на веб-сайте и использующий полученную информацию для отображения динамической персонализации контента в реальном времени.

Компонент использует различные технологии, такие как сбор данных, машинное обучение и аналитику данных, для обработки информации, полученной от пользователя, и предоставляет ему наиболее релевантный и подходящий контент.

Распишем подробнее алгоритм реализации данного компонента по пунктам.

1. Определение цифрового следа пользователя. Прежде всего необходимо определить список метрик, которые необходимо собирать. Метрики могут включать данные о поведении пользователя на сайте, такие как время пребывания на странице, действия на сайте, источник трафика и т.д. Затем необходимо реализовать непосредственно сбор данных, используя различные методы сбора информации о поведении пользователя на веб-сайте, такие как запись действий пользователя в лог-файлы, использование cookies и т.д. Важно убедиться, что процесс сбора информации не нарушает права пользователя и соответствует законодательству.

2. Анализ цифрового следа пользователя. Для этого используются методы машинного обучения и анализа данных, такие как кластеризация, классификация, ассоциативные правила и т.д. Эти методы позволяют определить предпочтения пользователя и его потребности.

3. Определение параметров динамической персонализации. На основе результатов анализа цифрового следа пользователя необходимо определить, какие параметры будут использоваться для динамической персонализации. Это могут быть, например, персонализированные рекомендации, отображение контента в зависимости от геолокации пользователя, изменение дизайна веб-сайта и т.д.

4. Разработка алгоритмов динамической персонализации. На основе определенных параметров необходимо разработать алгоритмы, которые будут использоваться для динамической персонализации. Эти алгоритмы могут быть различными, в зависимости от конкретных параметров и задач, которые необходимо решить.

5. Создание диаграммы и реализация компонента. На этом этапе необходимо создать диаграмму компонента, а также реализовать разработанные алгоритмы и параметры динамической персонализации в компоненте, который будет встроен на веб-сайт. Компонент должен иметь возможность быстро и эффективно обрабатывать цифровой след пользователя и вносить необходимые изменения на веб-сайт.

6. Тестирование и оптимизация компонента. После реализации необходимо провести тестирование компонента на реальных пользователях и оптимизировать его для повышения эффективности и качества пользовательского опыта. Необходимо также убедиться, что компонент соответствует законодательным требованиям и не нарушает права пользователей.

7. Разработка системы аналитики для отслеживания эффективности системы динамической персонализации и для выявления областей для улучшения. Для того, чтобы убедиться в том, что система динамической персонализации работает эффективно, необходимо разработать систему аналитики, которая позволит отслеживать различные параметры поведения пользователей на сайте, такие как: время, проведенное на сайте, количество посещений, конверсионные показатели и другие метрики. Эти данные могут быть использованы для анализа эффективности динамической персонализации и выявления областей для улучшения.

8. Организация поддержки и обновления системы для обеспечения ее работоспособности и развития. После запуска системы динамической персонализации на рабочем сервере необходимо организовать ее поддержку и обновление. Это включает в себя регулярное обновление системы, устранение возможных ошибок и неполадок, а также добавление новых функций и возможностей для развития системы и повышения ее эффективности.

Подводя итоги по сформулированным ранее алгоритмам, стоит также отметить, что для корректного функционирования каждого из перечисленных

компонентов необходимо убедится, что взаимодействия компонентов друг с другом протекают лаконично и безошибочно.

Выводы по 1 разделу.

В первом разделе был сформулирован термин динамической персонализации, были даны необходимые определения и рассказаны основные направления изучения данной выпускной квалификационной работы, а также был разработан обобщенный алгоритм реализации динамической персонализации путем декомпозиции общей задачи на множество подзадач и рассмотрения каждой из них в отдельности.

Были сформулированы алгоритмы реализаций компонентов для каждой подзадачи, а также были даны обобщенные представления о каждом из компонентов.

# 2 Создание общей структуры разработки веб-приложения, выбор инструментов разработки

## 2.1 Введение в структуру проекта, этапы разработки

Прежде чем приступать к полноценной разработке и внедрению динамической персонализации на веб-сайт необходимо сформировать четкий план разработки проекта и поэтапно расписать конкретные действия, необходимые для достижения поставленной цели. Распишем подробнее каждый из этапов.

Этап 1 – изучение требований и анализ проекта. В этом этапе определяются основные функциональные и нефункциональные требования к проекту, анализируются риски и возможности. Определяются инструменты, технологии и фреймворки, которые будут использоваться для создания проекта.

Этап 2 – проектирование архитектуры проекта. Определяются компоненты и модули, которые будут использоваться в проекте, их взаимосвязи и взаимодействие. Создаются прототипы интерфейса и макеты страниц.

Этап 3 – разработка. На этом этапе создаются компоненты и модули проекта, их взаимодействие и интеграция. Создаются стили и разметка страниц. В процессе разработки проекта используется компонентный подход, что позволяет создавать многоразовые компоненты с различными параметрами.

Этап 4 – тестирование и отладка проекта. В этом этапе проверяется работоспособность всех компонентов и модулей, а также функциональность проекта в целом. Исправляются ошибки и дорабатываются компоненты, если необходимо.

Этап 5 – запуск и внедрение проекта на рабочем сервере. На этом этапе настраивается серверная часть, создаются необходимые базы данных,

происходит настройка инфраструктуры. После запуска проекта происходит его мониторинг и поддержка.

## 2.2 Выбор направления разработки

Для начала необходимо определить основное направление разрабатываемого проекта, т.е. к какой именно части веб-приложения относится данный проект. Для определения рассмотрим все имеющиеся направления:

Серверная часть веб-приложения (Backend) – это часть приложения, которая запускается на сервере и обрабатывает запросы от клиентской части. Она отвечает за доступ к данным, бизнес-логику и взаимодействие с базами данных. В состав серверной части веб-приложения могут входить различные компоненты и технологии, такие как веб-сервер, серверное API, базы данных и другие. Веб-сервер отвечает за обработку запросов от клиента и отправку ответов. Он может использоваться для различных задач, таких как обслуживание статического контента, обработка динамического контента, обработка запросов API и других. Серверное API представляет собой интерфейс программирования приложений, который используется для взаимодействия между клиентской и серверной частями приложения. API может включать в себя функции, которые обрабатывают запросы от клиента и возвращают ответы.

Клиентская часть веб-приложения (Frontend) – это часть приложения, которая представляет собой набор технологий и инструментов, которые используются для создания пользовательского интерфейса (UI) и взаимодействия с пользователем через браузер. Она обеспечивает работу с данными и взаимодействие пользователя с приложением [24].

Данный проект – исключительно frontend разработка, основным требованием которого является реализация всех алгоритмов динамической персонализации, подробно рассмотренных в предыдущем разделе.

## 2.3 Выбор архитектуры приложения

Архитектура приложения – это общая структура и организация программного обеспечения, которая определяет его компоненты, их взаимодействие и принципы построения системы. Архитектура программы описывает, как компоненты системы взаимодействуют друг с другом, как они обрабатывают данные, как они обеспечивают безопасность и масштабируемость системы [21].

Архитектура является важным аспектом разработки программного обеспечения, поскольку она влияет на качество, надежность и поддерживаемость системы. Хорошо спроектированная архитектура может облегчить разработку, снизить затраты на тестирование и поддержку системы, а также сократить время разработки [15].

Существует множество различных архитектур в программировании, перечислим наиболее известные:

- монолитная архитектура – это традиционный подход к разработке, где все приложение находится в одном монолите. Эта архитектура может быть простой для начала работы, но со временем может стать сложной для поддержки и масштабирования;
- клиент-серверная архитектура – это архитектура, где приложение разделено на клиентскую и серверную части. Клиентская часть обычно запускается на устройстве пользователя, а серверная часть на удаленном сервере;
- распределенная архитектура – это архитектура, где приложение разделено на независимые компоненты, которые могут работать на разных устройствах или на разных серверах в сети;
- MVC архитектура – это архитектура, где приложение разделено на три части: модель, представление и контроллер. Модель отвечает за хранение и обработку данных, представление отвечает за

отображение данных, а контроллер отвечает за управление взаимодействием между моделью и представлением [14];

- MVP архитектура – это архитектура, где приложение также разделено на три части: модель, представление и презентер. Презентер отвечает за управление взаимодействием между моделью и представлением, в отличие от контроллера в архитектуре MVC [2];
- MVVM архитектура – это архитектура, где приложение разделено на три части: модель, представление и модель представления. Модель представления отвечает за управление взаимодействием между моделью и представлением, аналогично презентеру в MVP, но также обеспечивает двустороннюю привязку данных между моделью и представлением.

Стоит отметить, что выбор той или иной архитектуры зависит от многих факторов, таких как размер и сложность проекта, его цели, бюджет, доступные ресурсы и опыт команды разработчиков. В вопросе выбора наилучшей архитектуры нельзя дать точного и универсального ответа, т.к. одна архитектура может уступать другой по определенным критериям, но при этом доминировать по другим. Важно помнить, что не стоит выбирать архитектуру только на основе ее популярности. Необходимо внимательно изучить специфику проекта, его цели, бюджет, доступные ресурсы и опыт команды разработчиков, а затем выбрать подходящую архитектуру. Также стоит учитывать, что архитектура приложения может эволюционировать в процессе его разработки и подвергаться изменениям в зависимости от потребностей и новых требований к приложению.

Для данного приложения была выбрана компонентно-ориентированная архитектура, т.к. в данном случае она является наиболее подходящей. Для аргументации данного высказывания сперва сформулируем определение данной архитектуры.

Компонентно-ориентированная архитектура (КОА) – это подход к проектированию программного обеспечения, в котором приложение

разбивается на независимые компоненты, которые могут быть использованы повторно в разных частях системы. Каждый компонент имеет четко определенный интерфейс, который определяет, как он может быть использован другими компонентами. КОА является примером модульной архитектуры. Основным преимуществом данной архитектуры является возможность повторного использования компонентов в разных проектах и частях проекта. Это может существенно ускорить процесс разработки, снизить затраты на проект и улучшить качество кода, поскольку компоненты являются тестируемыми, самодостаточными и независимыми. Компоненты могут быть написаны на различных языках программирования и иметь различные реализации, например, как классы, библиотеки или модули. Компоненты обычно представляют собой логически законченные блоки функциональности, которые можно использовать в различных контекстах.

Таким образом, выбор компонентно-ориентированной архитектуры для данного приложения обусловлен простой горизонтальной масштабируемостью, высокой гибкостью при разработке отдельных модулей, возможностью переиспользования компонентов, а также динамику в плане использования различных языков программирования в одном проекте.

## 2.4 Выбор основных инструментов разработки

Существует множество различных инструментов разработки, позволяющих реализовать клиентскую часть веб-приложения. Зачастую используются такие инструменты как HTML, CSS и JavaScript. HTML используется для определения структуры страницы, CSS — для оформления и стилей, а JavaScript — для добавления динамики и логики [20]. Рассмотрим подробнее каждый из них.

HTML (HyperText Markup Language) – это язык разметки, который используется для определения структуры веб-страницы. Он состоит из набора тегов, которые описывают содержимое страницы и определяют его тип. Теги

могут использоваться для создания заголовков, параграфов, изображений, ссылок и других элементов на странице. HTML также используется для создания форм и таблиц [16].

CSS (Cascading Style Sheets) – это язык стилей, который используется для определения внешнего вида веб-страницы. Он определяет цвета, шрифты, размеры, положение элементов и другие свойства визуального оформления страницы. С помощью CSS можно создавать анимацию и применять эффекты переходов между страницами [7].

JavaScript – это язык программирования, который используется для добавления интерактивности на веб-страницы. Он может использоваться для создания сложных приложений, обработки форм, валидации данных, обработки событий и многого другого. JavaScript также может взаимодействовать с элементами на странице, изменять их свойства и содержимое, а также делать запросы к серверу и получать данные в реальном времени [18].

Каждый из этих языков имеет свои особенности и специфические возможности. Выбор именно этих языков обусловлен простотой использования в связке, поскольку данные языки обладают высокой степенью интеграции друг с другом и при этом вместе они покрывают весь спектр возможных задач.

### 2.5 Выбор фреймворка

Для наиболее комфортной и эффективной разработки с точки зрения скорости и оптимизации необходимо выбрать и интегрировать в проект наиболее подходящий фреймворк. Рассмотрим наиболее популярные на данный момент фреймворки:

React – это библиотека JavaScript, созданная Facebook для разработки пользовательских интерфейсов. React позволяет создавать компоненты, которые можно повторно использовать в приложении и динамически

обновлять при изменении данных. React работает с виртуальным DOM, что делает его более эффективным и быстрым в работе с большими объемами данных. Он также имеет множество расширений и общественных проектов, что делает его популярным выбором для разработчиков.

Angular – это фреймворк JavaScript, созданный Google. Он используется для разработки сложных веб-приложений и включает в себя множество функций, таких как директивы, шаблоны, компоненты и инструменты для работы с HTTP-запросами и формами. Angular также предоставляет широкий набор инструментов для тестирования, а также множество расширений и общественных проектов. Однако, Angular обладает достаточно большим порогом входа, поскольку он имеет более сложную архитектуру, чем React и Vue.

Vue – это фреймворк JavaScript, созданный для разработки пользовательских интерфейсов и однопоточных приложений. Vue имеет более легковесную структуру, чем Angular и более простую синтаксическую структуру, чем React. Vue также имеет возможность переиспользования компонентов и директив, что делает его подходящим выбором для разработки масштабируемых приложений. Кроме того, Vue имеет достаточно низкий порог входа и легко интегрируется с другими библиотеками и фреймворками.

Выбор между React, Angular и Vue зависит от ряда факторов, таких как опыт разработки, тип проекта, требования к производительности, доступность дополнительных библиотек и т.д. Каждый из них имеет свои преимущества и недостатки [3].

Для данного проекта был выбран фреймворк React, потому как данный фреймворк имеет ряд преимуществ по сравнению с Angular и Vue, вот основные из них.

Простота использования: React является библиотекой, что означает, что ее можно использовать как дополнение к другим фреймворкам. Это делает ее более легкой в освоении и быстрее внедряемой в проект.

Высокая производительность: React использует виртуальную DOM (Document Object Model), которая позволяет эффективно изменять только те элементы, которые действительно нуждаются в обновлении. Это улучшает производительность и позволяет создавать более быстрые и отзывчивые пользовательские интерфейсы.

Масштабируемость: React хорошо подходит для создания крупных проектов с большим количеством компонентов. Его модульная архитектура позволяет разбивать приложение на отдельные компоненты и легко переиспользовать их в других проектах.

### 2.6 Выбор методологии программирования

При разработке любого программного обеспечения или веб-приложения возникает необходимость организации самого процесса разработки. Достичь этого можно при выборе подходящей методологии.

Методология программирования – это совокупность принципов, правил, процедур и инструментов, которые используются при разработке программного обеспечения. Она определяет способы организации и управления процессом разработки, а также детали технических решений, которые необходимы для достижения поставленных целей [8].

Методология программирования может включать в себя шаблоны проектирования, принципы объектно-ориентированного программирования, методы тестирования и многие другие аспекты, которые позволяют разработчикам эффективно и качественно создавать программное обеспечение. Кроме того, методология программирования может устанавливать стандарты для документирования проекта и коммуникации между членами команды разработчиков.

Перечислим наиболее эффективные методологии.

Agile – методология разработки, которая основана на инкрементальном и итеративном подходе, с акцентом на быстрое реагирование на изменения

требований заказчика. Она подразумевает непрерывную коммуникацию и сотрудничество между разработчиками и заказчиком, быстрое выявление проблем и решение их в кратчайшие сроки.

Scrum – одна из Agile-методологий, которая представляет собой фреймворк для организации коллективной работы по разработке ПО. Scrum разбивает проект на небольшие итерации, называемые спринтами, и обеспечивает более эффективное управление проектом благодаря регулярным собраниям и постоянному общению между участниками команды.

Kanban – методология, основанная на визуализации процессов разработки и контроле за потоком задач. Она предназначена для управления работой, представляющей собой поток задач, которые могут постоянно меняться.

Waterfall – методология, которая предполагает последовательное выполнение этапов проекта, начиная со сбора требований и заканчивая тестированием и внедрением. Она предназначена для проектов с жестко определенными требованиями и четким планом работ.

Lean – методология, которая стремится устранить все ненужные этапы и процессы в разработке ПО, увеличивая эффективность работы команды и улучшая качество продукта.

Выбор между этими фреймворками зависит от конкретных требований и задач проекта, а также от опыта разработчиков.

При использовании связки фреймворка React и компонентно-ориентированной архитектуры наиболее подходящими методологиями являются Flux и Redux, так как они были разработаны все той же компанией Facebook для работы непосредственно с фреймворком React.

Flux – это архитектурный шаблон, разработанный Facebook для управления состоянием веб-приложений. Он основан на однонаправленном потоке данных и предполагает, что данные всегда движутся в одном направлении - от действия пользователя к хранилищу данных, а затем к

представлению. Flux разделяет приложение на четыре основных компонента: действия, диспетчеры, хранилища и представления.

Redux – это библиотека, основанная на концепции Flux, но с упрощенным и более предсказуемым подходом к управлению состоянием. Он также использует однонаправленный поток данных и разделяет приложение на три компонента: действия, редукторы и хранилища. Одной из основных концепций в Redux является использование неизменяемых объектов для представления состояния приложения.

Обе библиотеки предоставляют разработчикам инструменты для управления сложными состояниями приложений и помогают улучшить масштабируемость, тестируемость и общую структуру кода. Использование того или иного подхода зависит от требований конкретного проекта и предпочтений команды разработчиков.

Выводы по 2 разделу.

Во втором разделе были рассмотрены ключевые этапы разработки веб-приложения. Были сделаны акценты и даны определения на основополагающие сущности в программировании, такие как архитектуры и методологии, позволяющие при их наличии в проекте добиться наиболее эффективной стратегии разработки.

Были даны примеры существующих инструментов разработки. Для текущего проекта были выбраны наиболее подходящие инструменты реализации, начиная от выбора архитектуры приложения, до выбора основных инструментов разработки, фреймворка и методологии программирования. Были описаны основные преимущества и недостатки рассмотренных инструментов и подходов, что позволяет сделать обоснованный выбор при разработке конкретного проекта.

# 3 Реализация компонентов по алгоритмам динамической персонализации

Реализация каждого из компонентов будет выполнена по пунктам, каждый из которых будет взят из алгоритмов реализации, подробно описанных в 1 разделе.

При реализации для каждого пункта будет дан пример. После описания всех пунктов будет сформирован готовый пример компонента.

## 3.1 Реализация компонента, изменяющего тему веб-сайта

1. Определение функциональных требований. Определим, каким функционалом должен обладать данный компонент. Таким функционалом может являться изменение цветов текста и различных частей веб-приложения, изменение положения и форм блоков и т.д. Пусть данный компонент будет иметь возможность изменять цвета текста и определенных блоков на веб-сайте.

2. Создание диаграммы. После определения функциональных требований наиболее целесообразным действием является составление диаграммы компонента. Наличие диаграммы к компоненту позволяет существенно упростить понимание логики работы, что является весомым аргументом при работе в команде, а также позволяет сократить время разработки. В случае с предложенным функционалом диаграмма приняла вид, изображенный на рисунке 1.

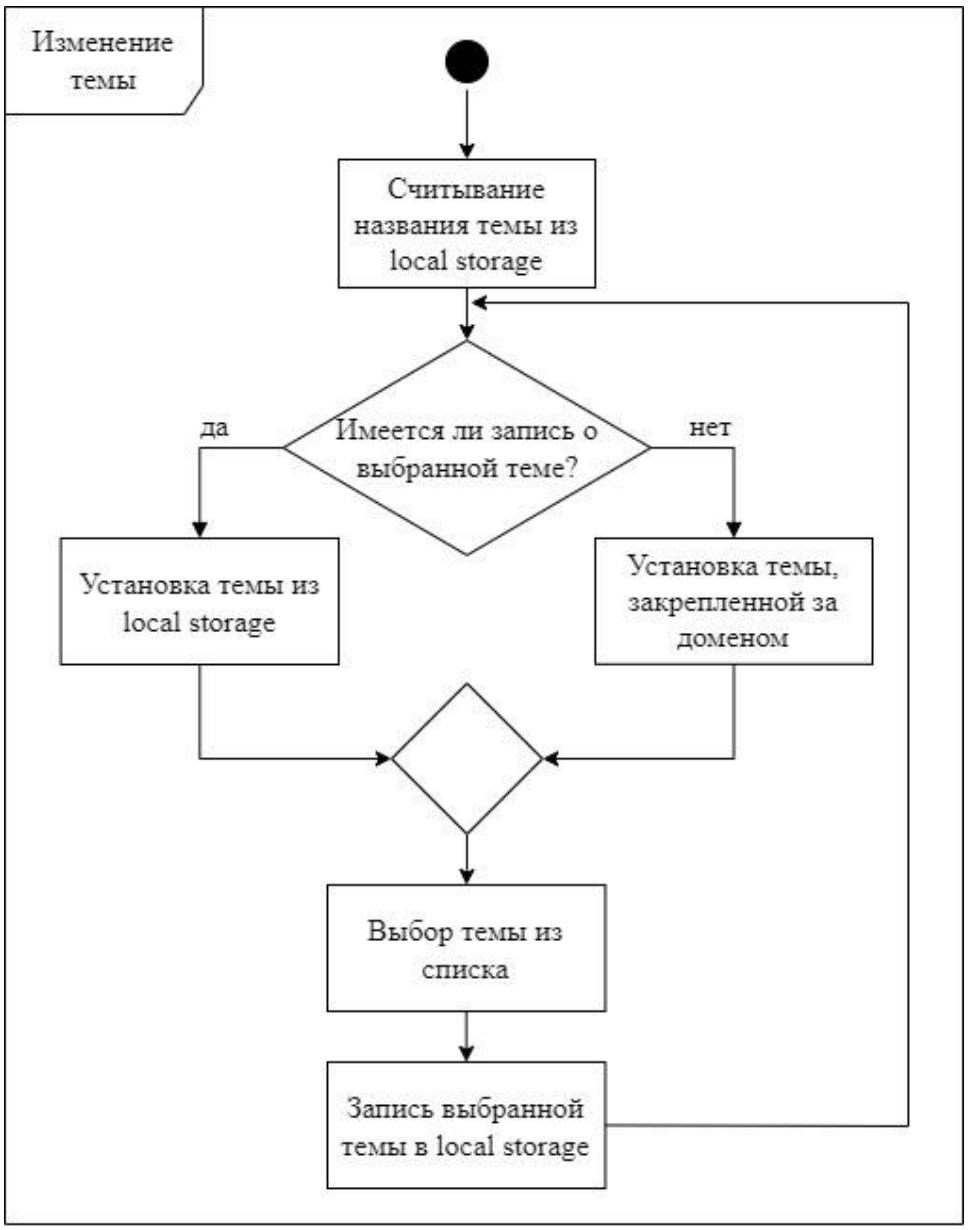


Рисунок 1 – Диаграмма компонента, изменяющего тему веб-сайта

3. Создание интерфейса редактора. Далее определим возможный дизайн данного компонента, т.е. создадим его макет. «При создании макета можно воспользоваться графическими редакторами, такими как Adobe Photoshop или Adobe Illustrator, а также инструментами для создания макетов, такими как Figma, Sketch или Adobe XD» [1]. Допустим, данный компонент будет представлять из себя некое меню, в котором находятся цветные карточки тем. При клике левой кнопкой мыши по любой из карточек тема будет изменена на соответствующую.

4. Написание программного кода. Сперва необходимо определиться каким именно образом данный компонент будет изменять стили сразу во всех файлах CSS. Есть множество различных способов реализации данной особенности. Одним из них является использование переменных CSS. «Данные переменные могут быть объявлены в основном файле CSS, который должен быть подключен к главному js файлу (зачастую index.js), и затем стили из этого файла могут быть использованы в различных местах документа, включая правила из других CSS файлов. Основным CSS файлом можно считать файл, содержащий тему веб-сайта. При смене главного CSS файла на другой, все правила из других файлов также будут изменены» [13].

Далее необходимо реализовать функционал, который будет отвечать за замену главного CSS файла на другой и делать это при нажатии на карточку темы. При использовании React, наиболее уместным решением является создание кастомного хука, использующего базовые хуки React, такие как useState и useLayoutEffect.

Хук useState отвечает за создание и управление состоянием компонента в React. Он позволяет определить переменную состояния и обновлять ее значение внутри компонента при изменении состояния. При обновлении состояния компонент будет перерисован.

Хук useLayoutEffect запускается синхронно после того, как React обновит DOM, но перед тем, как браузер отрисует изменения на экране. Это позволяет выполнять дополнительные операции, связанные с изменением компонента и/или его контента, которые должны быть завершены до того, как пользователь увидит изменения на экране.

При обновлении или перезапуске веб-сайта, выбранная тема сбрасывается и возвращается тема по умолчанию. Для предотвращения подобного поведения необходимо убедиться в том, что выбранная пользователем тема сохраняется в хранилище и достается именно оттуда при повторном посещении или обновлении веб-сайта. Хранилищем темы может служить база данных или localStorage.

Пример готового компонента представлен на рисунке 2.

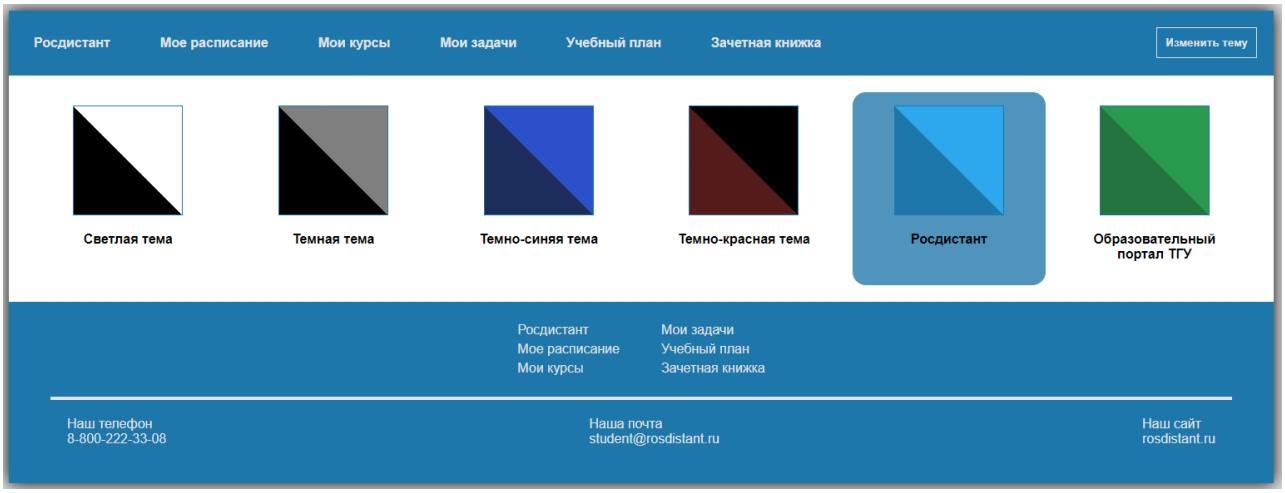


Рисунок 2 – Пример реализации компонента, изменяющего тему веб-сайта

По итогам разработки был разработан компонент, изменяющий тему веб-сайта за счёт использования базовых хуков useState и useLayoutEffect, с применением переменных CSS для задания стилей. Тема в данном случае записывается в localStorage в виде строки, содержащей имя CSS файла, закрепленного за темой.

### 3.2 Реализация компонента, изменяющего язык веб-сайта

1. Определение функциональных требований. Требования к данному компоненту могут заключаться в возможности смены основного языка веб-приложения, в поддержке определенных языков, а также в переводе отдельных блоков или всего приложения целиком. Реализуем функционал данного компонента таким образом, чтобы он переводил текст во всем веб-приложении с русского на английский и наоборот. При этом необходимо, чтобы выбранный язык сохранялся при обновлении страницы. Также в компонент необходимо внедрить поддержку мультидоменности.

2. Создание диаграммы компонента. Диаграмма для функционала, определенного в предыдущем пункте, изображена на рисунке 3.

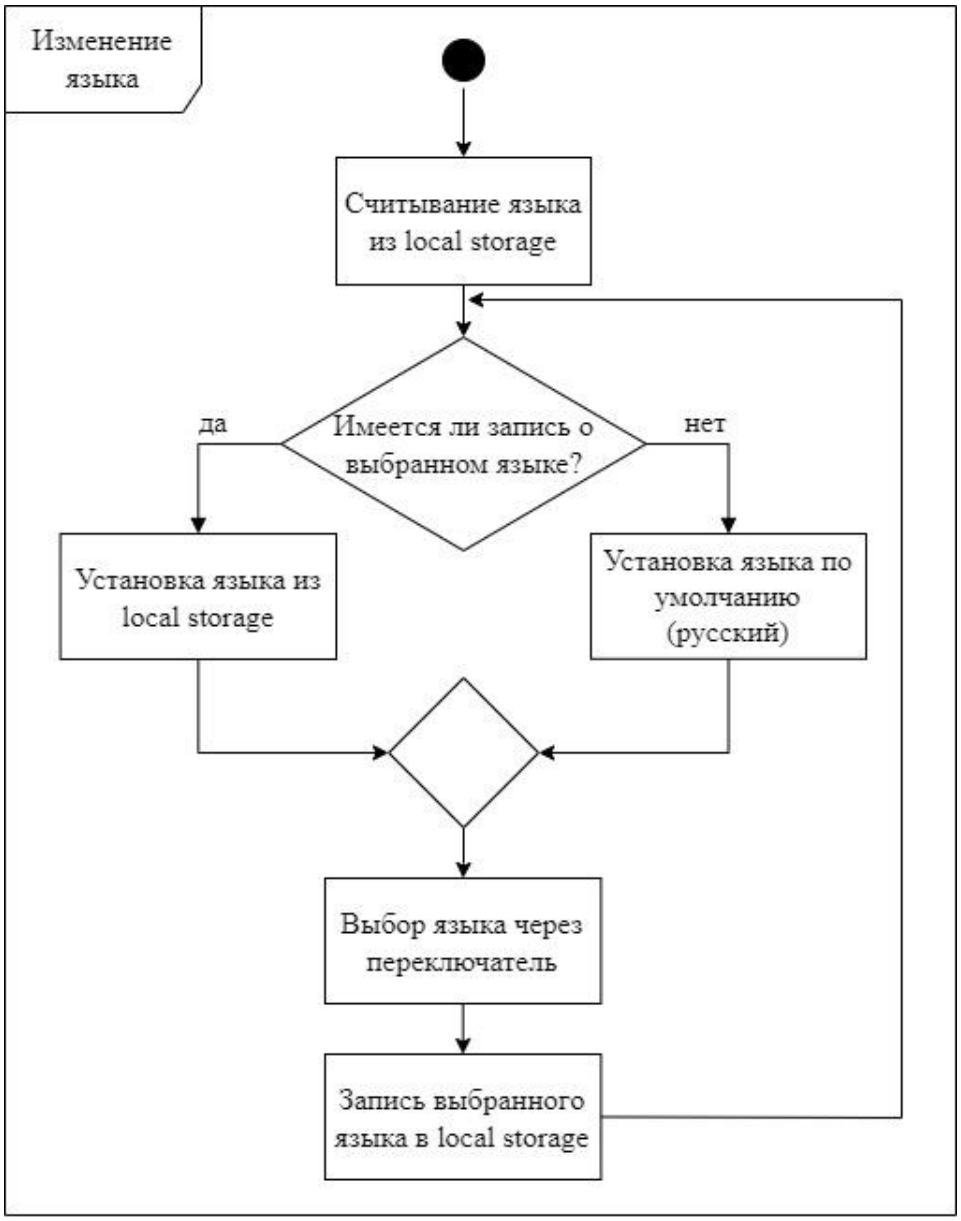


Рисунок 3 – Диаграмма компонента, изменяющего язык веб-сайта

3. Создание макета компонента. «Данный компонент зачастую определяют в виде селектора или переключателя, изменяющего язык с одного на другой. В данном случае, компонент будет реализован в виде переключателя языка с русского на английский» [25].

4. Реализация программного кода. При реализации компонента, основной функционал перевода текста был описан с помощью библиотеки i18next, разработанной специально для решения задач локализации в фреймворке React. При переводе текста использовался хук useTranslation.

Для реализации корректной архитектуры, основной объект i18next был вынесен в отдельный файл. При изменении домена данный объект пересоздается с необходимыми параметрами, предназначенными для перевода конкретного домена.

Файлы локализации в данном случае были вынесены в файлы json, каждый из которых хранит в себе локализацию под определенный язык.

Для сохранения выбора пользователем того или иного языка, информация о выбранной локализации записывается в localStorage.

Пример результата работы данного компонента представлен на рисунке под номером 4.

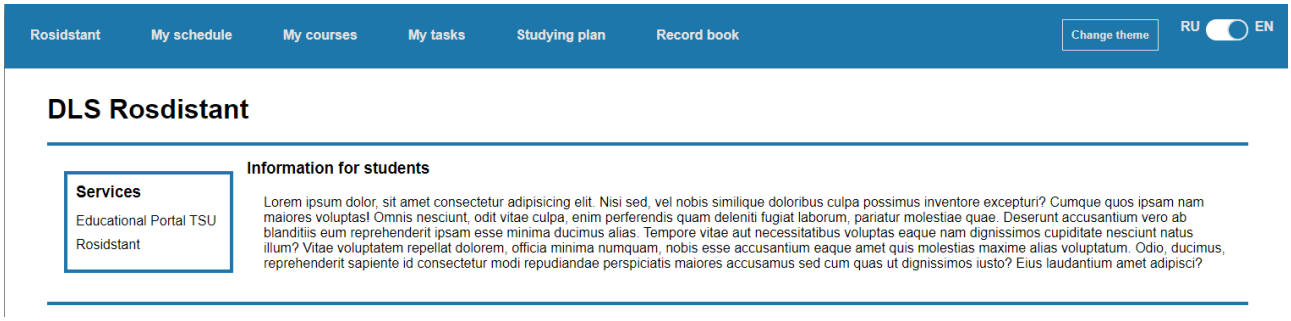


Рисунок 4 – Пример работы компонента локализации

По итогам разработки был реализован компонент, позволяющий пользователю изменять язык локализации веб-приложения на русский и английский языки. При разработке для перевода текста использовалась библиотека i18next и хук useTranslation.

### 3.3 Реализация компонента, позволяющего администратору задавать тематические, сезонные и праздничные темы

1. Определение функциональных требований. Для компонента, позволяющего администратору задавать тематические, сезонные и праздничные темы, функциональные требования могут быть следующими:

- возможность добавлять новые темы и удалять существующие;
- возможность задавать даты начала и окончания действия темы;
- возможность выбирать цветовую схему для каждой темы;
- возможность выбирать фоновое изображение для каждой темы.

2. Создание диаграммы компонента. Диаграмма данного компонента представлена на рисунке 5.

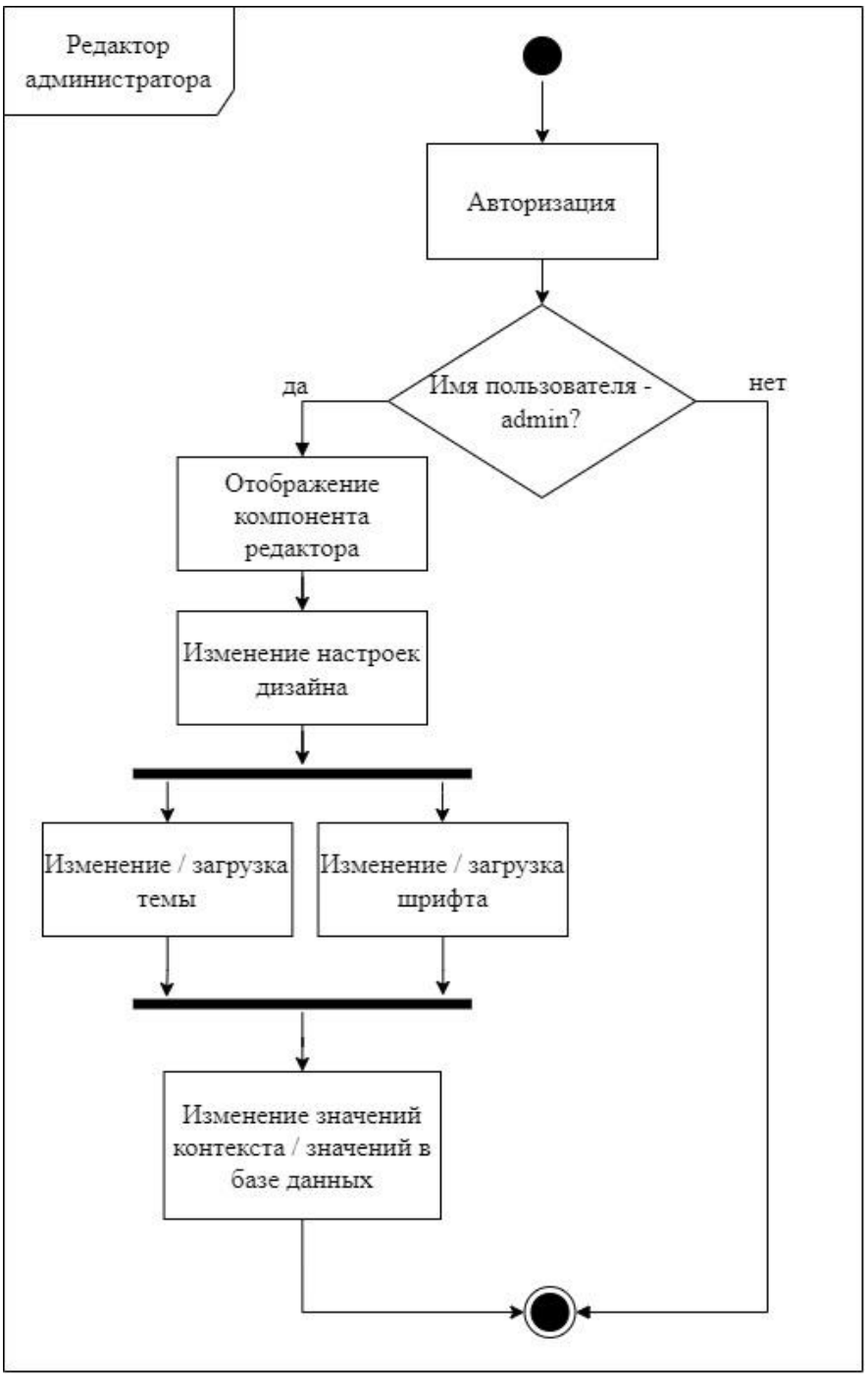


Рисунок 5 – Диаграмма компонента редактора администратора

3. Создание дизайна интерфейса. Интерфейс данного компонента может содержать поля ввода для дат начала и окончания действия темы, выбор цветовой схемы и фонового изображения, а также кнопки удаления и добавления тем. Наличие того или иного функционала на макете напрямую

зависит от требований к компоненту. На рисунке 6 приведен пример реализации интерфейса данного компонента.

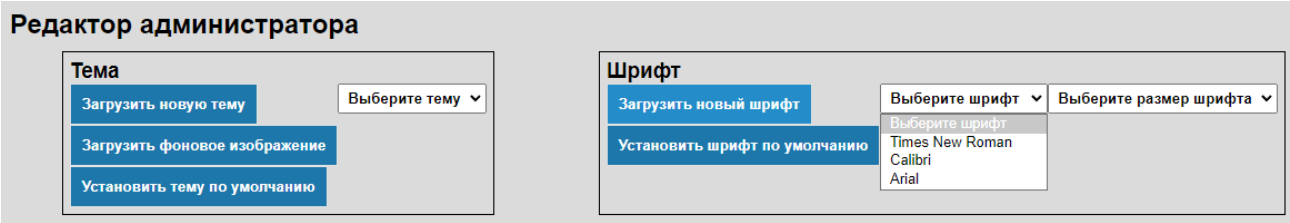


Рисунок 6 – Пример реализации интерфейса редактора администратора

4. Программная реализация. При разработке данного компонента необходимо создать множество побочных компонентов, таких как компоненты формы, загрузки фоновых изображений и тем, а также кнопки удаления определенных тем. Необходимо также создать обработчик событий для отправки данных на сервер.

Необходимо настроить функционал отображения компонента при авторизации конкретного пользователя – администратора веб-сайта.

Для функционала, изменяющего тему веб-сайта, можно использовать уже готовый компонент изменения тем, рассмотренный ранее. При интеграции одного компонента в другой, необходимо убедится в корректном взаимодействии и отображении компонентов друг с другом.

Для реализации функционала, изменяющего дизайн во всем приложении для всех пользователей, сперва необходимо определить, где именно будет хранится информация о дизайне. В данном случае данную информацию можно хранить как базе данных, так и в контексте.

Хранение информации в базе данных позволит увеличить масштабируемость всего приложения, а также значительно упростит синхронизацию изменений дизайна между разными устройствами и сессиями пользователей. Однако, реализация подобного функционала может потребовать дополнительных ресурсов и сложностей. Потребуется настроить

базу данных, создать соответствующую схему для хранения данных о дизайне и реализовать механизмы для обновления и чтения этих данных из базы данных. Данный функционал является наиболее подходящим для более крупных проектов, поскольку он подразумевает под собой лучшую масштабируемость. Однако, для небольших проектов реализация настолько усложненного решения может оказаться избыточным.

Использование контекста является более упрощенным вариантом, не требующим использования базы данных. Контекст – это механизм в фреймворке React, позволяющий передавать данные через дерево компонентов без явной передачи параметров от компонента к компоненту. Контекст предоставляет удобный способ взаимодействия между компонентами, которые находятся на разных уровнях иерархии. Таким образом, обращение к информации из контекста будет происходить гораздо быстрее и проще. Однако, при данном подходе значительно усложняется масштабируемость. Следовательно, использование контекста является наиболее актуальным при внедрении в небольшие приложения, не подразумевающие наличие авторизации и использование баз данных для хранения информации.

5. Документация компонента. Данный пункт включает в себя создание подробной инструкции по использованию компонента для администратора веб-сайта. Документация должна включать информацию о том, как использовать компонент, какие параметры доступны для настройки и какие ограничения присутствуют при использовании компонента.

Также в документации должны быть описаны возможные ошибки при использовании компонента и способы их устранения. Рекомендуется добавить скриншоты и примеры использования компонента для лучшего понимания его работы. Документация компонента должна быть понятна для всех пользователей, которые будут работать с ним.

По итогам разработки был реализован компонент, позволяющий администратору веб-сайта загружать, изменять, и удалять темы веб-сайта.

Также был реализован компонент формы, дающий администратору возможность устанавливать время действия той или иной темы.

Была описана подробная документация, содержащее инструкцию для работы с компонентом, а также возможные ошибки, способы их устранения и скриншоты.

### 3.4 Реализация компонента, динамически изменяющего наполнение и дизайна веб-сайта в зависимости от текущего домена

1. Создание диаграммы компонента. Диаграмма представлена на рисунке 7.

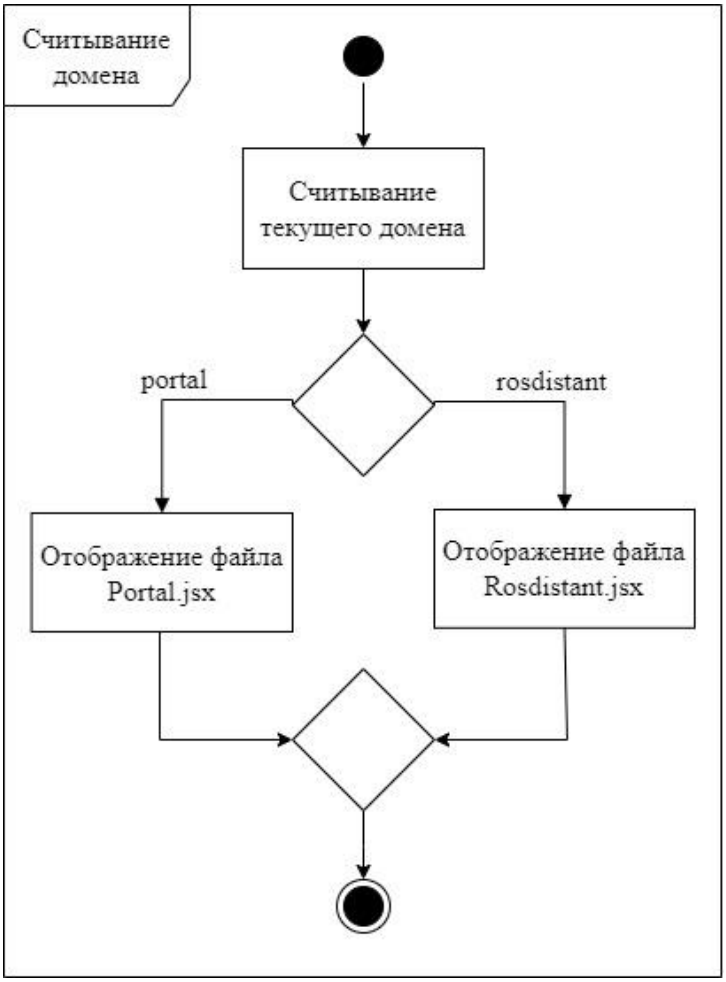


Рисунок 7 – Диаграмма компонента, считывающего домен веб-сайта

2. Получение текущего домена веб-сайта. При реализации данного компонента необходимо реализовать функционал, который будет считывать домен веб-сайта при первом запуске или изменении домена во время работы. Для данного компонента было решено считывать название домена с помощью глобального объекта window с последующей проверкой полученного домена.

3. Определение соответствующего контента и дизайна для домена. Для выполнения этого пункта были реализованы две основные страницы, представляющие собой две разные версии веб-сайта. Данные страницы были закреплены за соответствующими доменами.

4. Загрузка контента и дизайна для текущего домена. В данном случае загружаемый контент хранится в конфигурационных файлах в самом проекте. При загрузке и отображении необходимых компонентов при смене домена достаточно обратиться к соответствующему файлу.

5. Обновление веб-сайта с новым контентом и дизайном. Для более упрощенного перехода между доменами был реализован компонент, содержащий ссылки на имеющиеся домены. При написании ссылок использовался тег Link, позволяющий изменять текущий домен. При переходе по ссылке происходит изменение домена, затем компонент считывает новый домен и обновляет страницу для загрузки нового контента.

Пример работы компонента представлен на рисунке 8.

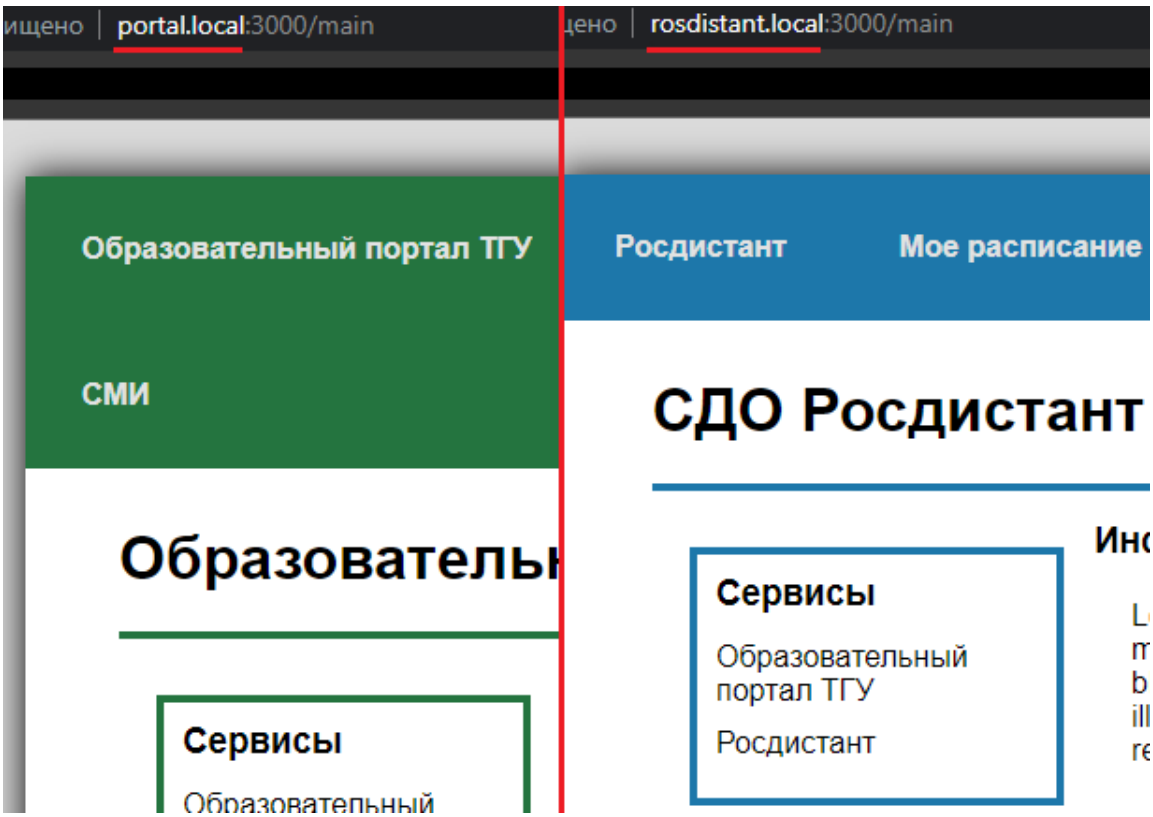


Рисунок 8 – Пример работы компонента динамического домена

По итогам разработки был реализован компонент, считывающий текущий домен при первом запуске и при изменении домена во время работы. Для двух доменов были определены две основные страницы, представляющие собой две разные версии веб-сайта.

Был реализован вспомогательный компонент, отвечающий за навигацию между двумя доменами и содержащий ссылки на эти домены.

### 3.5 Реализация компонента, анализирующего цифровой след пользователя для отображения динамической персонализации

1. Определение цифрового следа пользователя. На данном этапе необходимо определить, какую конкретно информацию о пользователе необходимо собирать. Такими данными могут быть геолокация пользователя, его местное время, время пребывания на странице и другие поведенческие

данные. В данном случае будет производится считывание геолокации пользователя для отображения погоды на веб-сайте. При сборе данных были использованы API Geolocation для сбора данных о геолокации и API OpenWeatherMap для получения данных о погоде в текущей геолокации.

2. Анализ цифрового следа пользователя. Данный шаг предполагает обработку и анализ данных, полученных на первом этапе, для выявления особенностей поведения пользователя, таких как интересы, предпочтения, привычки, демографические данные и др. Для этого могут быть использованы различные методы машинного обучения, статистические методы, анализ социальных сетей и другие.

3. Определение параметров динамической персонализации. На основе результатов анализа цифрового следа пользователя необходимо определить, какие параметры персонализации следует использовать для каждого конкретного пользователя. Это могут быть, например, персональные рекомендации, настройки интерфейса, цветовые схемы и др.

4. Разработка алгоритмов динамической персонализации. На этом этапе необходимо разработать алгоритмы, которые будут определять, какие параметры персонализации необходимо использовать для каждого пользователя на основе его цифрового следа.

5. Создание диаграммы компонента. После определения функционала компонента целесообразным действием является построение диаграммы компонента для более четкого понимания структуры. Диаграмма представлена на рисунке 9.

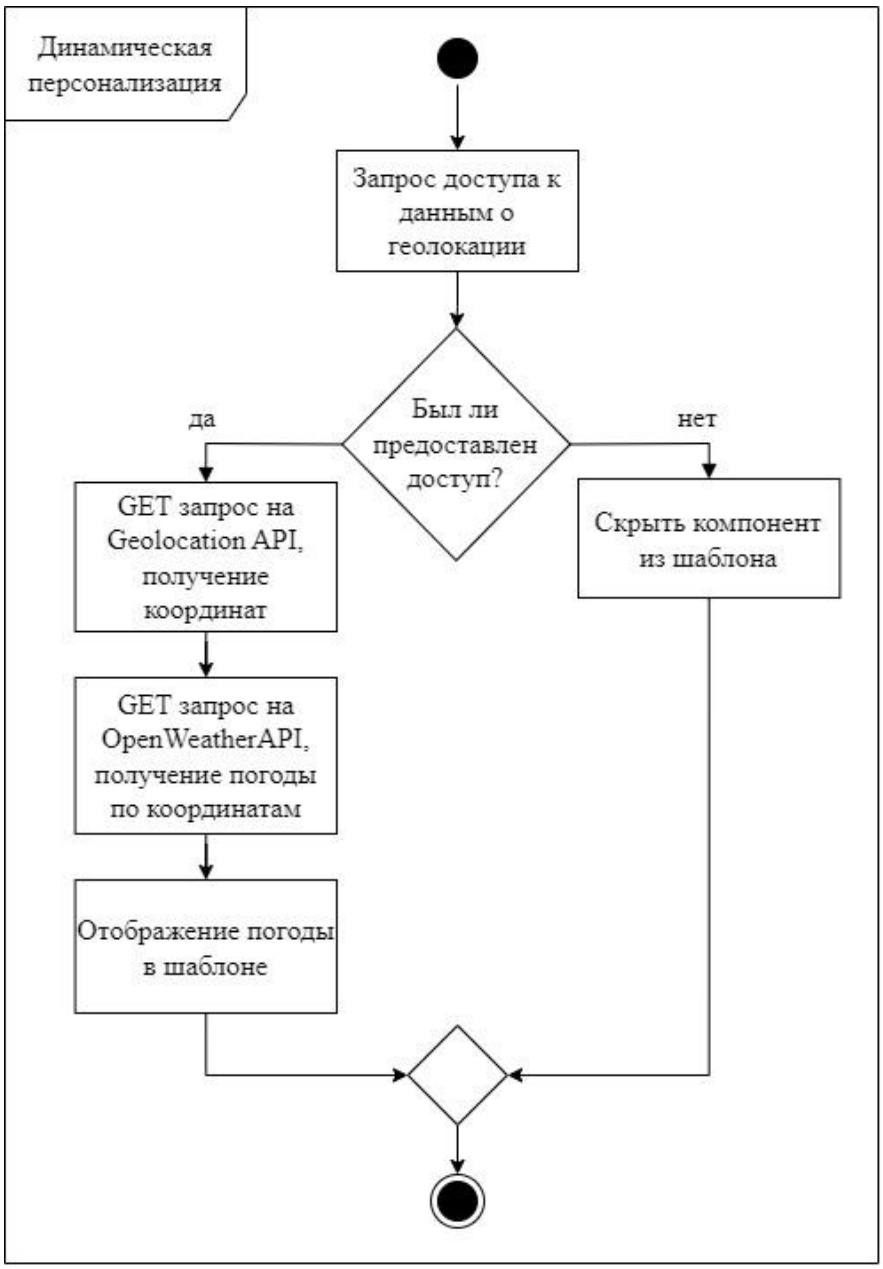


Рисунок 9 – Диаграмма компонента, считывающего геолокацию пользователя и отображающего динамическую персонализацию

6. Реализация компонента. На последнем этапе следует реализовать компонент, который будет использовать разработанные алгоритмы динамической персонализации для отображения персонализированного контента. Это может быть реализовано через создание отдельного модуля веб-приложения, который будет взаимодействовать с базой данных и другими компонентами приложения для предоставления персонализированного

контента пользователю. Пример работы компонента представлен на рисунке 10.

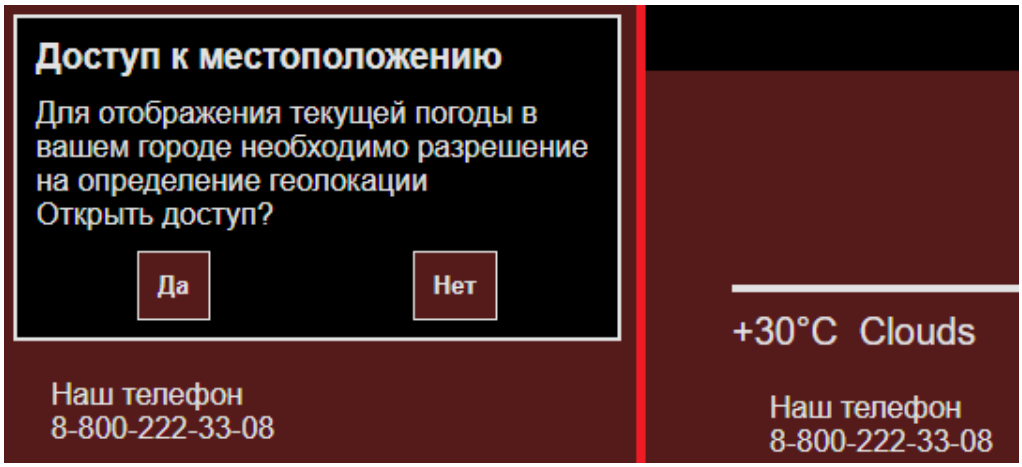


Рисунок 10 – Пример работы компонента

По итогам разработки был реализован компонент, считывающий информацию о геолокации конкретного пользователя и отображающий данные о погоде в текущей геолокации на веб-сайте.

При разработке были задействованы API Geolocation для сбора данных о геолокации и API OpenWeatherMap для получения данных о погоде в текущей геолокации.

Выводы по 3 разделу.

В третьем разделе были пошагово реализованы все алгоритмы, сформулированные в первом разделе. Были приведены общие представления о выполнении каждого из пунктов, а также были даны примеры реализаций каждого из алгоритмов в виде текстового описания и скриншотов полученных компонентов.

При реализации были использованы инструменты разработки, выбранные во втором разделе.

# Заключение

По итогам данной выпускной квалификационной работы были разработаны алгоритмы реализации динамической персонализации.

В рамках проделанной работы был произведен анализ предметной области и теоретических сведений динамической персонализации. Была произведена декомпозиция общей задачи динамической персонализации на отдельные подзадачи, которые рассматривались по отдельности. Для каждой из поставленных задач были сформулированы пошаговые алгоритмы реализации.

В рамках процесса реализации поставленных задач по сформулированным алгоритмам был произведен анализ существующих инструментов разработки. Для реализации текущего веб-приложения были выбраны наиболее актуальные и подходящие инструменты. При выборе архитектуры приложения, выбор был сделан в пользу компонентно-ориентированной архитектуры. Такой выбор обусловлен наличием у данной архитектуры простой горизонтальной масштабируемости, высокой гибкости при разработке отдельных модулей и возможности переиспользования компонентов. Основным инструментом разработки послужил фреймворк React и его сторонние библиотеки. Данный фреймворк был выбран за счет простоты использования, высокой производительности и масштабируемости.

Были сформулированы общие представления реализуемых компонентов. На основе разработанных алгоритмов были описаны пошаговые примеры реализаций компонентов в текстовом формате и с демонстрацией скриншотов.

Итогом работы являются разработанные алгоритмы реализации динамической персонализации.

## Список используемой литературы


1. Figma – простое решение для дизайнера, сложное решение для верстальщика / [Электронный ресурс] URL: https://habr.com/ru/post/463181
2. Архитектура микросервисов / [Электронный ресурс] URL: https://habr.com/ru/companies/vk/articles/320962/
3. Байдыбеков А.А., Гильванов Р.Г., Молодкин И.А. Современные фреймворки для разработки web-приложений // Интеллектуальные технологии на транспорте. 2020. №4 (24). URL: https://cyberleninka.ru/article/n/sovremennye-freymvorki-dlya-razrabotki-web-prilozheniy (дата обращения: 16.05.2023).
4. Бондаренко С.О. Современные интерактивные веб-приложения - построение пользовательского интерфейса с React // Вестник науки и образования. 2018. №5 (41). URL: https://cyberleninka.ru/article/n/sovremennye-interaktivnye-veb-prilozheniya-postroenie-polzovatelskogo-interfeysa-s-react (дата обращения: 16.05.2023).
5. Вяткина Б.М. Разработка графического дизайна веб-сайта фирмы на основе бизнес-плана и фирменного стиля // Известия вузов. Инвестиции. Строительство. Недвижимость. 2016. №1 (16). URL: https://cyberleninka.ru/article/n/razrabotka-graficheskogo-dizayna-veb-sayta-firmy-na-osnove-biznes-plana-i-firmennogo-stilya (дата обращения: 16.05.2023).
6. Гайд по контент-маркетингу (сезонному) / [Электронный ресурс] URL: https://seo2you.ru/blog/seo/gajd-po-sezonnomu-kontent-marketingu/
7. Грошкова А.А., Махова А.И. Преимущества использования CSS // Актуальные проблемы авиации и космонавтики. 2017. №13. URL: https://cyberleninka.ru/article/n/preimuschestva-ispolzovaniya-css (дата обращения: 16.05.2023).
8. Ильясова Ф.С., Клеблеев Ш.А. Современные методологии объектно-ориентированного программирования // Символ науки. 2015. №11-

2. URL: https://cyberleninka.ru/article/n/sovremennye-metodologii-obektno-orintirovannogo-programmirovaniya (дата обращения: 16.05.2023).

9. Исмойилов Х.Б. Стили сайтов в веб-дизайне // Современные материалы, техника и технологии. 2018. №2 (17). URL: https://cyberleninka.ru/article/n/stili-saytov-v-veb-dizayne (дата обращения: 16.05.2023).

10. Кондакова А.А., Михайлов А.С. Интерактивный веб-дизайн V.2.0 // Актуальные проблемы авиации и космонавтики. 2019. №. URL: https://cyberleninka.ru/article/n/interaktivnyy-veb-dizayn-v-2-0 (дата обращения: 16.05.2023).

11. Луковский М.А. Современные технологии развития и разработки веб-сайтов на основе принципов эмоционального дизайна // Наука и современность. 2014. №27. URL: https://cyberleninka.ru/article/n/sovremennye-tehnologii-razvitiya-i-razrabotki-veb-saytov-na-osnove-printsipov-emotsionalnogo-dizayna (дата обращения: 16.05.2023).

12. Правильный редизайн сайта – пошаговый алгоритм, основные вопросы и нюансы / [Электронный ресурс] URL: https://habr.com/ru/articles/472172/

13. Радченко А.Ю. Сравнительный анализ CSS-препроцессоров // Молодой исследователь Дона. 2021. №3 (30). URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-css-preprotsessorov (дата обращения: 16.05.2023).

14. Сабиров Д.А. Микросервисная архитектура на Frontend // Научный журнал. 2021. №7 (62). URL: https://cyberleninka.ru/article/n/mikroservisnaya-arhitektura-na-frontend (дата обращения: 16.05.2023).

15. Сервис-ориентированная архитектура (SOA) / [Электронный ресурс] URL: https://habr.com/ru/companies/vk/articles/342526/

16. Степанов А.В. Языки разметки. Часть 2: основные средства форматирования // КИО. 2008. №2. URL: https://cyberleninka.ru/article/n/yazyki-

razmetki-chast-2-osnovnye-sredstva-formatirovaniya (дата обращения: 16.05.2023).

17. Тараскина Я.В., Цыремпилон А.О., Платицина Т.В. Локализация веб-сайта университета: переводческий аспект // Филология: научные исследования. 2020. №3. URL: https://cyberleninka.ru/article/n/lokalizatsiya-veb-sayta-universiteta-perevodcheskiy-aspekt (дата обращения: 16.05.2023).

18. Удовицкий И.А. Электронное учебное пособие «Программирование на JavaScript» // Гаудеамус. 2011. №18. URL: https://cyberleninka.ru/article/n/elektronnoe-uchebnoe-posobie-programmirovanie-na-javascript (дата обращения: 16.05.2023).

19. Якутова О.М., Петрова О.А. Шрифт как элемент дизайна веб-сайта // Актуальные проблемы авиации и космонавтики. 2013. №9. URL: https://cyberleninka.ru/article/n/shrift-kak-element-dizayna-veb-sayta (дата обращения: 16.05.2023).

20. Яровая Е.В. Современные инструменты для написания веб приложений // Столыпинский вестник. 2022. №5. URL: https://cyberleninka.ru/article/n/sovremennye-instrumenty-dlya-napisaniya-veb-prilozheniy (дата обращения: 16.05.2023).

21. Grinkrug E.M., Shakurov A.R. Component Architecture with Runtime Type Definition // Радиоэлектроника и информатика. 2010. №4. URL: https://cyberleninka.ru/article/n/component-architecture-with-runtime-type-definition (дата обращения: 16.05.2023).

22. Gogua M.M., Smirnova M.M. Revisiting Personalization through Customer Experience Journey // Вестник Санкт-Петербургского университета. Менеджмент. 2020. №4. URL: https://cyberleninka.ru/article/n/revisiting-personalization-through-customer-experience-journey (дата обращения: 16.05.2023).

23. Manage dynamic and custom subdomains in React / [Электронный ресурс] URL: https://dev.to/parth2412/manage-dynamic-and-custom-subdomains-in-react-3071

24. Programming Methodologies / [Электронный ресурс] URL: https://social.technet.microsoft.com/wiki/contents/articles/28335.programming-methodologies.aspx

25. Volkova I.D. Localization of Website Verbal Content into English as a Marketing Strategy // СИСП. 2019. №1-1. URL: https://cyberleninka.ru/article/n/localization-of-website-verbal-content-into-english-as-a-marketing-strategy (дата обращения: 16.05.2023).