

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

Отчет о прохождении

|  |
| --- |
| ***преддипломной практики*** |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ОБУЧАЮЩЕГОСЯ | **4** | КУРСА | БАКАЛАВРИАТА | ГРУППЫ | ИДБ-16-07 |
|  |  |  |  |  |  |

|  |
| --- |
| **Махмудов Бобурбек Нодирбекович** |
|  |

|  |  |
| --- | --- |
| Кафедра: | информационных систем |

|  |  |
| --- | --- |
| НАПРАВЛЕНИЕ ПОДГОТОВКИ: | 09.03.02 Информационные системы и технологии |
| Направленность ПОДГОТОВКИ: | Информационные системы и технологии |

|  |  |
| --- | --- |
| Место прохождения  практики: | ФГБОУ ВО «МГТУ «СТАНКИН» |

|  |  |
| --- | --- |
| сРОКИ прохождения  ПРАКТИКИ: | 27.04.2020 – 10.05.2020 |

Руководители практики:

|  |  |
| --- | --- |
| от кафедры | Бумарин Д.П.,к.т.н., доцент |
|  |  |

Москва

2020

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc39518603)

[ГЛАВА 1. ПРОЕКТИРОВАНИЕ СЕРВИСА ПО ПРЕДОСТАВЛЕНИЮ УСЛУГ СИСТЕМЫ ВЕБ-ШАБЛОНОВ 4](#_Toc39518604)

[1.1. Требования, предъявляемые к качеству разрабатываемой системы 4](#_Toc39518605)

[1.2. Моделирование работы сервиса 6](#_Toc39518606)

[1.3. Выбор инструментов 14](#_Toc39518607)

[ГЛАВА 2. РАЗРАБОТКА И ТЕСТИРОВАНИЕ 16](#_Toc39518608)

[2.1. Разработка сервиса 16](#_Toc39518609)

[2.2. Результат разработки, оценка производительности 25](#_Toc39518610)

[ЗАКЛЮЧЕНИЕ 35](#_Toc39518611)

[СПИСОК ЛИТЕРАТУРЫ 36](#_Toc39518612)

ВВЕДЕНИЕ

С момента своего изобретения, электронные вычислительные машины становились всё более сложными и одновременно более производительными, на сегодняшний день данная тенденция сохраняется. Например, согласно закону Мура, количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца, что увеличивает сложность интегральной схемы, но при этом увеличивая количество операций, которая она может совершать, то есть её производительность. Теория вычислительных алгоритмов тоже не стоит на месте, и с каждым годом на свет появляются различные публикации, описывающие способы уменьшения временных и других ресурсов, требуемых для решения популярных задач в области вычислений.

В рамках данной работы рассматривается проблема, связанная с относительно низкой производительностью системы веб-шаблонов. В ходе её написания будут проанализированы самые широко используемые решения, доступные как среди проприетарных, так и свободно распространяемых программных продуктов. На основе анализа будут выявлены наиболее распространённые причины низкой производительности. После чего будет изучен ряд мер которые можно использовать для устранения причин понижения быстродействия или уменьшения их эффекта. На основе предложенных мер будут спроектирована высокопроизводительная система шаблонизации, и разработаны требования к технологиям и инструментам для её реализации. С использованием выбранных технологий будет разработана система шаблонизации, которая должна адресовать ранее выявленные причины низкой производительности. Для оценки результатов разработки будет произведено тестирование разработанного приложения под нагрузкой, с целью оценки показателей быстродействия.

# ПРОЕКТИРОВАНИЕ СЕРВИСА ПО ПРЕДОСТАВЛЕНИЮ УСЛУГ СИСТЕМЫ ВЕБ-ШАБЛОНОВ

## Требования, предъявляемые к качеству разрабатываемой системы

Перед тем как начать проектировать систему веб-шаблонов, необходимо определить перечень требований к качеству, которым должная удовлетворять целевая система. В «ГОСТ Р ИСО/МЭК 25010 Модели качества систем и программных продуктов»[1] представлены две модели качества: модель качества при использовании и модель качества продукта. Целесообразным решением будет использовать именно эти модели для определения требований к разрабатываемой системе.

Модель качества при использовании содержит пять основных характеристик качества, структура модели изображена на рис. 3.1.1.



Рис. 1.1.1 Модель качества при использовании

Из приведённых характеристик качества, к разрабатываемой системе можно предъявить требования только по трём из них:

1. Эффективность, результативность. Точность и полнота, с которой пользователи достигают определенных целей.
2. Производительность. Связь точности и полноты достижения пользователями целей с израсходованными ресурсами.
3. Удовлетворенность. Способность продукта или системы удовлетворить требованиям пользователя в заданном контексте использования.

Модель качества продукта содержит восемь основных характеристик качества, иерархическая модель данной модели изображена на рис. 1.1.2



Рис. 1.1.2 Модель качества продукта

Из приведённых характеристик качества продукта, к разрабатываемой системе можно предъявить требования по пяти из них:

1. Уровень производительности. Производительность относительно суммы использованных при определенных условиях ресурсов.
2. Совместимость. Способность продукта, системы или компонента обмениваться информацией с другими продуктами, системами или компонентами, и/или выполнять требуемые функции при совместном использовании одних и тех же аппаратных средств или программной среды.
3. Удобство использования. Степень, в которой продукт или система могут быть использованы определенными пользователями для достижения конкретных целей с эффективностью, результативностью и удовлетворенностью в заданном контексте использования.
4. Надежность. Степень выполнения системой, продуктом или компонентом определенных функций при указанных условиях в течение установленного периода времени.
5. Сопровождаемость, модифицируемость. Результативность и эффективность, с которыми продукт или система могут быть модифицированы предполагаемыми специалистами по обслуживанию.

Все описанные требования в дальнейшем будут оказывать влияние на выбор определённых технологий для разработки проекта, и на архитектурные решения во время его проектирования

## Моделирование работы сервиса

Переда началом разработки необходимо спроектировать архитектуру веб-сервиса, и смоделировать его работу. Как говорилось ранее, веб-сервис будет иметь клиент-серверную архитектуру, то есть приложения клиенты будут отправлять HTTP запросы на сервер, сервер будет обрабатывать запрос и посылать обратно HTTP ответ. Такая архитектура позволяет удовлетворить требование совместимости с уже существующей ИТ инфраструктурой. Ещё одним преимуществом такого подхода является удобство использования, так как от приложения клиента требуется лишь поддержка HTTP протоколов обмена данными. Для повышения характеристики сопровождаемости веб-сервис необходимо разрабатывать модульным, то есть за каждый конкретный тип запроса будет отвечать отдельный модуль, и для расширения возможностей веб-сервиса необходимо будет просто разработать новый модуль с минимальными изменениями в остальных частях приложения.

Так как взаимодействовать исключительно через API может быть неудобным для специалистов не технического профиля и даже прикладным программистам, то целесообразно будет разработать веб-интерфейс, который позволит интерактивно взаимодействовать с веб-сервисом, например, для проведения отладки шаблона или создания совершенно новых шаблонов.

Использование многопоточного и асинхронного программирования в разработке сервиса позволит ему обрабатывать огромное количество запросов одновременно даже на машинах с ограниченными вычислительными ресурсами.

Далее будет проведено моделирование работы разрабатываемого веб-сервиса с использованием функциональных диаграмм в нотации IDEF0[2]. Для создания диаграмм использовалась программа Ramus Educational версии 1.1.

На самом верхнем уровне, работа веб-сервиса выглядит следующим образом (рис. 1.2.1).

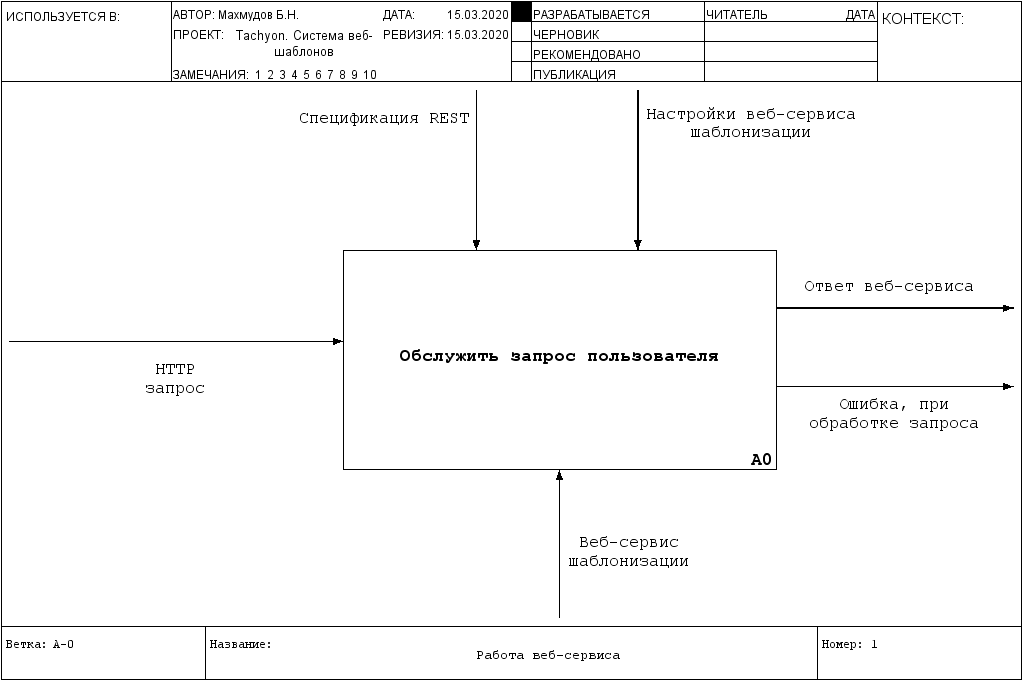


Рис. 1.2.1 Контекстная диаграмма работы веб-сервиса

На вход поступает HTTP запрос от клиента, веб-сервис шаблонизации обрабатывает его, руководствуясь спецификацией REST[3] и настройками самого сервиса, и на выходе он выдаёт ответ, который содержит запрашиваемый ресурс или ошибку, если запрос не может быть удовлетворён.

Декомпозируя этот уровень более детально, можно выделить основные блоки (модули) которые принимают участие в обработке запроса (рис. 1.2.2).

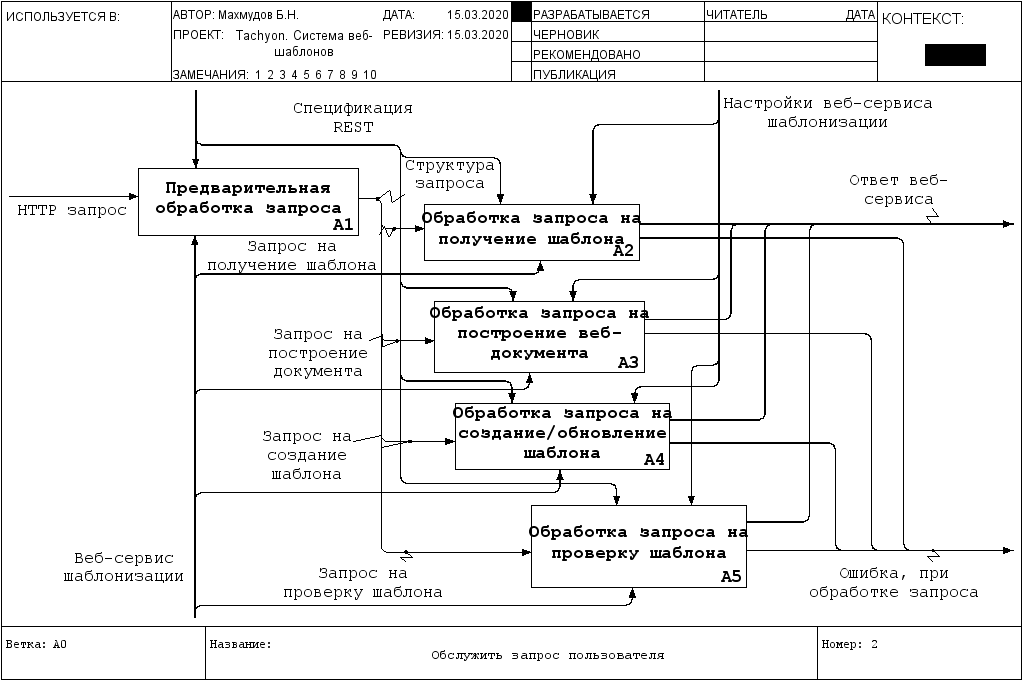


Рис. 1.2.2 Декомпозиция контекстной диаграммы

Как видно из диаграммы, запрос клиента сначала подвергается предварительной обработке для конвертации его во внутреннее представление запроса, с которым могут работать другие модули. Далее в зависимости от типа запроса он отдаётся на обработку соответствующему модулю. На диаграмме изображено четыре модуля обработки, но вообще говоря их может быть больше. Каждый модуль может быть декомпозирован на более детализированный уровень, что и будет сделано далее.

На рис. 1.2.3 приведена DFD декомпозиция блока предварительной обработки запроса. Так как HTTP запрос представляет из себя поток байт, то его сначала необходимо считать с сокета установленного подключения, после чего это множество байт разбирается в составляющие компоненты запроса, и из этих компоненты собирается внутреннее представление для дальнейшей обработки.

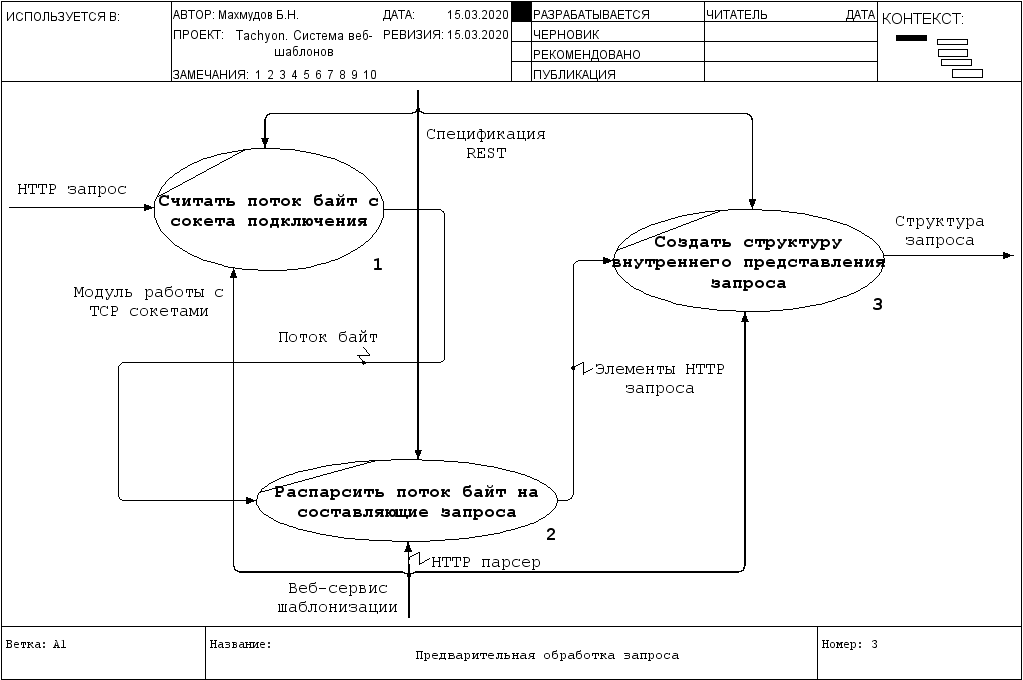


Рис. 1.2.3 DFD декомпозиция предварительного обработки запроса

Из полученной структуры данных можно определить то, как нужно обработать запрос, и эта структура отдается соответствующему модулю. На рис. 1.2.4 продемонстрирована диаграмма, которая моделирует процесс обработки запроса на получение шаблона. Для этого необходимо получить уникальный идентификатор шаблона, проверить его наличие в специальном файле индексе, который содержит всю информацию о шаблоне. В случае нахождения шаблона в индексе, его надо считать из файла шаблона и сформировать ответ в соответствии со спецификацией REST, в противном случае клиенту отправляется ошибка об отсутствии шаблона.

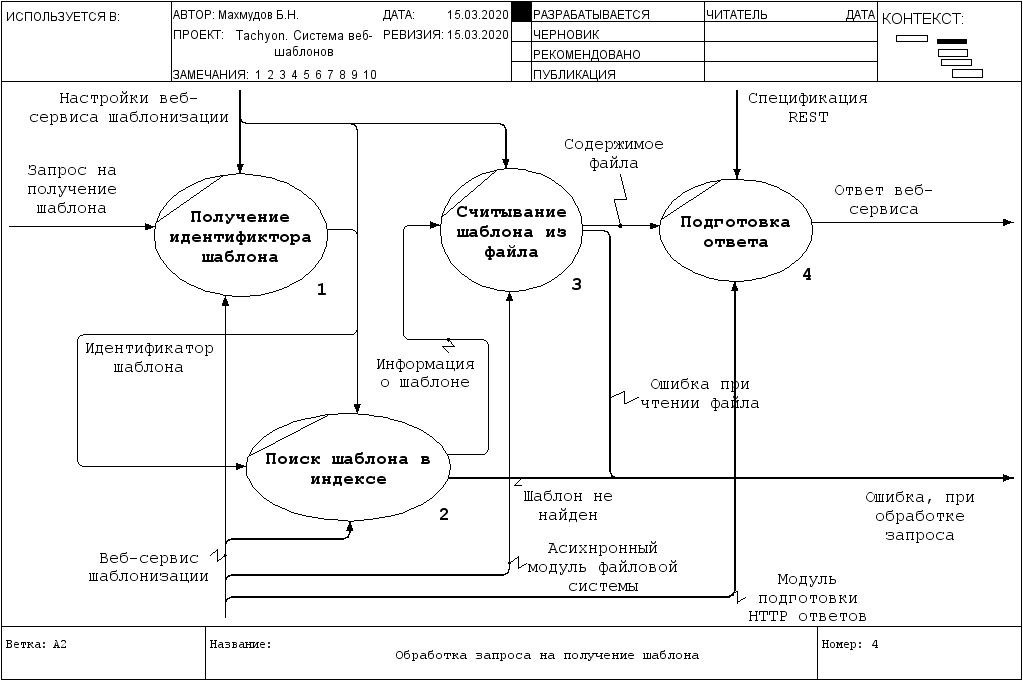


Рис. 1.2.4 DFD декомпозиция процесса получения шаблона

На рис. 1.2.5 изображена IDEF0 декомпозиция блока по построению веб-документа (основная функция проектируемого веб-сервиса). Шаблонизатору для построения конечного веб-документа необходимо наличие абстрактного синтаксического дерева и данных шаблонизации. Так как для ускорения процесса шаблонизации была введена особая структура данных для хранения уже построенных АСД – кеш, то перед тем как загружать шаблон из файла необходимо сделать проверку наличия АСД в кеше. Если шаблон отсутствует в кеше, нужно выполнять все шаги по построению АСД сначала, в противном случае можно сразу вернуть готовое АСД, данный процесс изображён на рис. 1.2.6. После получения абстрактного синтаксического дерева, необходимо сделать валидацию данных в соответствии с настройками шаблонизатора. Последним шагом является непосредственное построение конечного веб-документа и формирование ответа сервиса (рис. 1.2.7).

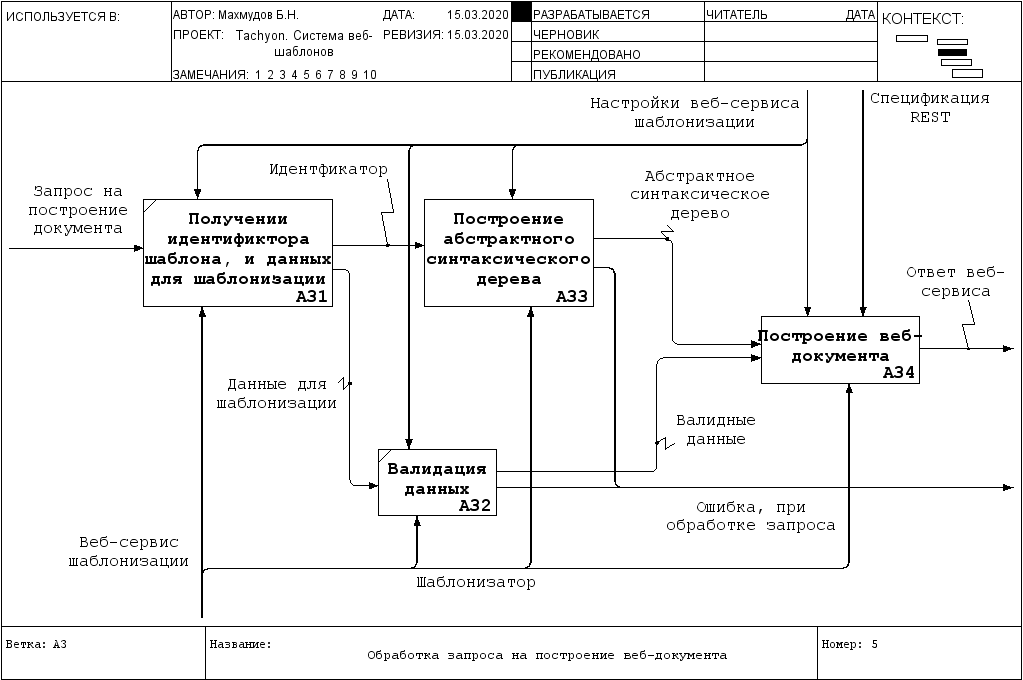


Рис. 1.2.5 IDEF0 декомпозиция процесса генерации веб-документа

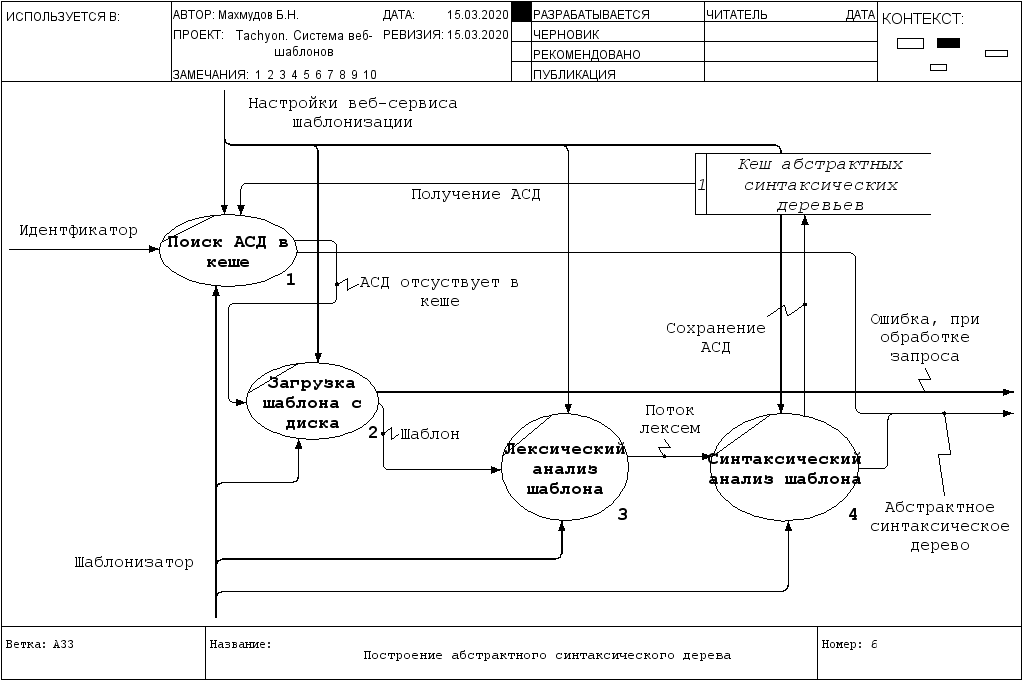


Рис. 1.2.6 DFD декомпозиция процесса получения абстрактного синтаксического дерева

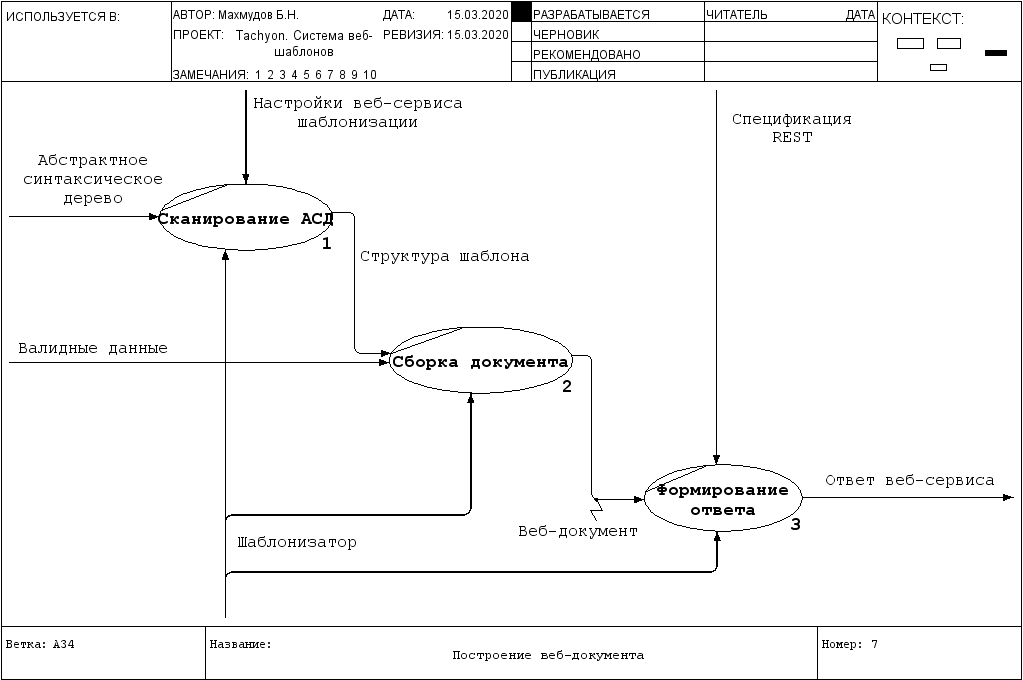


Рис. 1.2.7 DFD декомпозиция процесса построения документа

Рис. 1.2.8 демонстрирует относительную сложную диаграмму, моделирующую процесс создания или обновления шаблона. Для начала необходимо сделать проверку самого шаблона на корректность путём анализа его шаблонизатором. Если шаблон корректен, то далее нужно найти его в индекс файле, и если он там присутствует, то нужно обновить файл шаблона новыми данными, если же шаблон отсутствует в индекс файле, значит необходимо создать новый шаблон, создав новый файл шаблона. В конце необходимо сохранить изменения в индекс файле и уведомить клиента об успешном сохранении. Если в каком-то из шагов произойдёт ошибка, то никакие изменения не будут сохранены в постоянную память, тем самым не нарушая целостность системы, а клиенту будет отправлено соответствующее сообщение о возникшей ошибке обработки запроса.

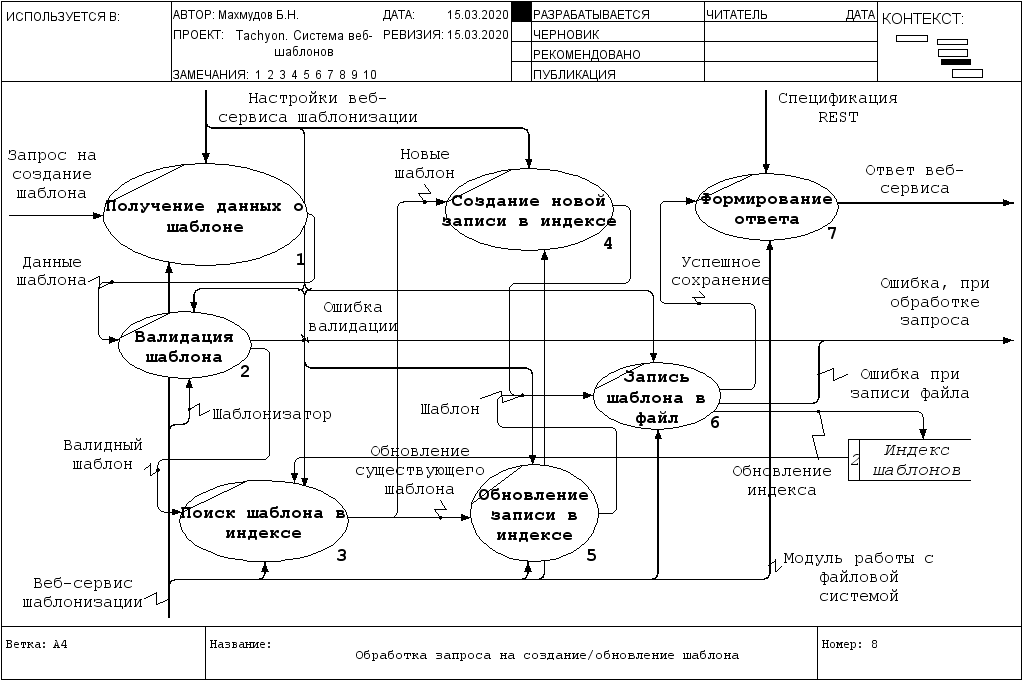


Рис. 1.2.8 DFD декомпозиция процесса создания или обновления шаблона

На рис. 1.2.9 приведена диаграмма процесса проверки шаблона на корректность в нотации DFD. Процесс аналогичен тому, что был описан при создании или обновлении шаблона.

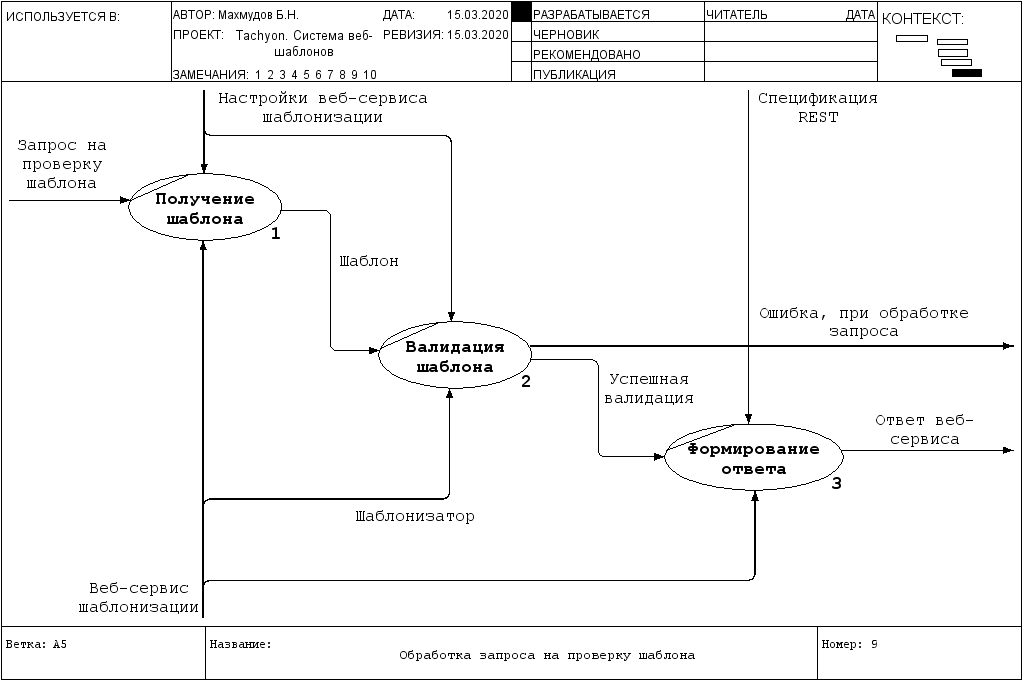


Рис. 1.2.9 DFD декомпозиция проверки веб-шаблона

## Выбор инструментов

Перед началом разработки следует уделить тщательное внимание выбору инструментов и технологий, которые будут использованы для реализации проекта. В силу того, что разрабатываемое приложение имеет клиент-серверную архитектуру, где клиент может быть любой сущностью, поддерживающая HTTP протокол для обмена данными, то разработка клиента не относиться к целям данной выпускной квалификационной работы, выбор клиента остаётся на усмотрение конечного пользователя. Но как оговаривалось ранее, для удобства использования сервиса, требуется разработать веб-интерфейс для интерактивного взаимодействия с системой шаблонизации. С учётом сказанного разработку можно разделить на два области: разработка серверной части или бэкенд составляющей и разработка графического пользовательского интерфейса или фронтенд составляющей.

Для разработки серверной части будет использован язык программирования Rust[4] с библиотеками из экосистемы языка, а именно необходимы будут следующие библиотеки:

* библиотека обработки HTTP запросов,
* библиотека, предоставляющая многопоточную и асинхронную среду выполнения для разрабатываемого приложения,
* библиотека сериализации и десериализации структур данных Rust в форматы обмена данными, таких как JSON или XML,
* библиотека шаблонизации.

В качестве обработчика HTTP запросов была выбрана библиотека httparse, в силу того, что библиотека почти полностью избегает выделений памяти и использует низкоуровневые SIMD инструкции процессора для ускорения процесса разбора запроса.

Для предоставления многопоточной и асинхронной среды выполнения будет использована достаточно зрелая библиотека tokio, так как она предоставляет множество абстракций над низкоуровневыми механизмами управления потоками исполнения и ввода/вывода данных, при этом не жертвуя производительностью.

Для сериализации и десериализации структур данных Rust будет использована библиотека serde, ввиду наличия хорошей документации, и ориентации на высокие показатели быстродействие.

Основным компонентом системы веб-шаблонов является шаблонизатора, из-за чего необходимо выбрать одновременно быструю и удобную в использовании библиотеку шаблонизации. В экосистеме Rust существует два вида шаблонизаторов, те, которые компилируют шаблоны вместе с исполняемым файлом, и те, которые анализируют шаблон во время выполнения программы. Причем в первом случае процесс построения веб-документов на порядок быстрее чем во втором, но такой подход имеет два существенных изъяна:

1. Время компиляции программы, которое и без того достаточно большое, становится ещё больше в зависимости от количества шаблонов.
2. Отсутствует возможность динамического добавления или обновления шаблонов.

Так как оба этих недостатка недопустимы в случае разрабатываемой системы, то выбирать придётся из второй группы. Среди таких библиотек, согласно нескольким тестам на производительность, лидирующую позицию занимает библиотека Tera, которая обладает синтаксисом языка шаблонизации схожую с Django. Также Tera имеет очень подробную документацию по использованию, как самой библиотеки, так и языка шаблонизации.

Для разработки графического пользовательского интерфейса будут использована совокупность технологий веб-разработки HTML/CSS/JavaScript. Для облегчения разработки также будут применены дополнительные фреймворки, которые основаны на этих технологиях, а именно:

* Element-UI, HTML/CSS фреймворк для создания веб-интерфейсов[7],
* Vue.js JavaScript фреймворк для создания динамичных и асинхронных веб-интерфейсов[6].

В следующем разделе будет описан процесс разработки веб-сервиса, с использование вышеописанных технологий.

# РАЗРАБОТКА И ТЕСТИРОВАНИЕ

## Разработка сервиса

По причине того, что архитектурно само приложение можно разделить на бэкенд и фронтенд составляющие, то и саму разработку удобно проводить в два отдельных этапа:

1. Разработка серверной части (бэкенда).
2. Разработка графического интерфейса пользователя (фронтенда).

Работу над созданием серверной части можно подразбить на отдельные подзадачи:

● разработке модуля шаблонизации, будет являться ядром сервиса,

● разработке общего интерфейса обработки ошибок, с учетом спецификации REST.

● разработке модуля работы с файловой системой, а именно реализация базовых возможностей веб-сервера,

● разработке асинхронной многопоточной среды для работы сервиса.

Модуль шаблонизации полностью базируется на библиотеке Tera. Задачей модуля является обработка запросов на построение документа с использование указанного в запросе шаблона, и предоставленных данных. Основой модуля является структура данных представляющая из себя экземпляр шаблонизатора и все ассоциированные с ним данные. Эта структура данных является глобальной и статичной, и может быть безопасно использована множеством обработчиков запросов одновременно, ввиду того, что она не изменяема и гарантированно будет сохранять одно и тоже состояние. Для безопасного изменения этой структуры данных, например, при добавлении или изменении шаблонов, необходимо будет накладывать на структуру какую-либо блокировку с целью недопущения порчи данных. Для таких целей в стандартной библиотеке Rust имеется особая разновидность блокировок, называемая RWLock, её отличие от мютексов состоит в том, что мютексом одновременно может владеть только один поток исполнения, вне зависимости от того, используется ли он для чтения или записи, а RWLock, может одновременно иметь множество потоков, которые обращаются к содержимому для чтения или только один поток, который может обращаться к содержимому для изменения. Таким образом в стандартном режиме множество потоков могут одновременно использовать структуру данных шаблонизатора, которая обёрнута в RWLock[5], для построения документов, но для изменения шаблонов, изменяющему потоку нужно будет дождаться другие читающие потоки, до завершения их работы, и только после этого получить эксклюзивную блокировку и произвести изменение. Похожее поведение имеется в системах управления базами данных без поддержки много-версионности.

Стандартизованная обработка ошибок также является важной задачей, в силу того, что ошибки в ходе работы сервиса могут возникать в самых разных ситуациях, далее приведен пример наиболее возможных ошибок:

* ошибка чтения из файла или записи в файл;
* ошибка получения запроса от клиента;
* ошибка валидации данных для шаблонизации;
* ошибка проверки шаблона на корректность.

Все эти ошибки возникают в разных программных модулях и вообще говоря имеют разную структуру. Основная задача состоит в том, чтобы привести все эти ошибки к единому формату для дальнейшего формирования ответа об ошибке для клиента. В Rust не имеется механизмов отлова ошибок, которые присутствуют в других языках, таких как try-catch в JavaScript или try-except в Python, сделано это всё опять-таки для повышения производительности. Вместо этого в Rust принято возвращать результат операции, который представляет из себя алгебраический тип данных, то есть составной тип, который может быть либо ожидаемым значением, которая должна произвести операция, если она завершилась успехом, или же ошибкой, если операция завершилась неудачно. Также, как и в других языках в Rust имеется механизм распространения ошибок вверх по стеку вызовов. То есть если ошибка произошла в какой-либо функции, то программист может решить передать ошибку вызывающей функции, а та в свою очередь вызвавшей её, и так далее до обработчика ошибок, который извлечёт необходимую информацию и сформирует ответ, уведомляющий пользователя о произошедшей ошибке.

Возможность работать с файловой системой необходима для:

1. возможность считывать и записывать файлы шаблонов;
2. возможность обслуживать запросы на получение статических файлов пользовательского интерфейса.

Второй случай использование позволяет разрабатываемому приложение выступать в качестве базового веб-сервера для обслуживания запросов на статичные файлы, например, это могут быть HTML страницы, файл таблицы стилей или файл, содержащий JavaScript код. При этом нужно учесть, что скорость чтения и записи файлов может быть значительно ниже чем возможности сервиса обработке запросов. В связи с этим, дабы не допустить простоя вычислительных ресурсов и появления очередей запросов, необходимо разработать асинхронную неблокирующую среду выполнения программы.

Для создания асинхронной среды выполнения используется библиотека tokio, которая построена вокруг библиотеки futures, предоставляющая элементарные единицы асинхронных вычислений. Future – это значение какого-либо вычисления, которое ещё не завершилось, аналогично Promise в JavaScript, но с оптимизациями, направленные на более эффективное использование ресурсов. Основной механизм работы Future можно описать следующим образом:

1. Программист описывает структуру данных, и ассоциированные с ней методы, которая представляет из себя, строго говоря, конечный автомат, который может переходить из одного состояния в другое под влиянием каких-либо событий, происходящих в системе. Состояния соответствуют этапам вычисления, через которые оно должно пройти для получения конечного результата.
2. Описание этого вычисления отдаётся на исполнение среде выполнения, называемой executor. Executor будет пытаться выполнить все этапы вычисления поочерёдно, но может столкнуться с блокировками, например связанными с вводом/выводом. В таком случае, среда выполнения сохранит состояние конечного автомата, с регистрацией запроса операционной системе, на уведомлении о наступлении требуемого события, например, завершения операции ввода/вывода. После этого среда выполнения возьмётся за исполнение другого вычисления, которое может быть продолжено. С наступлением этого события executor продолжит выполнение сохранённого ранее вычисления с того этапа, с которого оно было прервано. Executor одновременно можно использовать несколько потоков выполнения, тем самым максимально эффективно используя вычислительные ресурсы.

Веб-сервис будет компилироваться в один исполняемый файл, который не имеет сторонних зависимостей, так как все необходимые компоненты уже включены в состав бинарного файла. Этот факт делает приложение очень простым в использовании. От пользователя требуется лишь указать на каком сетевом порту нужно запуститься веб-сервису, что делается из командной строки.

Следующим этапом разработки является создание графического пользовательского интерфейса. Интерфейс пользователя будет предоставлять базовые возможности, такие как:

* построение документов на основе существующих шаблонов, и передаваемых пользователем данными;
* ознакомление со списком существующих шаблонов;
* редактирование шаблонов;
* создание новых шаблонов;
* ознакомление с документацией по синтаксису языка шаблонизации;
* ознакомление с документацией по методам, предоставляемых API веб-сервиса.

Основная часть разработки интерфейса будет производится с использованием JavaScript фреймворка Vue.js. Данный фреймворк за последние годы, хорошо зарекомендовал себя как средство создания SPA (Single Page Application), то есть веб-приложений в одной странице. Иначе говоря, при посещении страницы, все необходимые компоненты веб-приложения грузятся с сервера с самого начала, и дальнейшее взаимодействие с сервером будет производиться асинхронно без перезагрузки страницы. За каждую функциональную возможность в Vue.js отвечает отдельный компонент, и так как всего графический интерфейс предоставляет шесть различных функциональных возможностей, то целесообразно для каждой разработать отдельный компонент. Каждый компонент включает в себя три составляющих: html верстка страницы, стили страницы, логика страницы, написанная на JavaScript с использованием Vue.js.

При обращении к сервису, пользователь попадает на главную страницу, которая изображена на рис. 2.1.1. На этой странице имеется панель навигации, посредством которой можно выбирать интересующий пользователя инструмент.

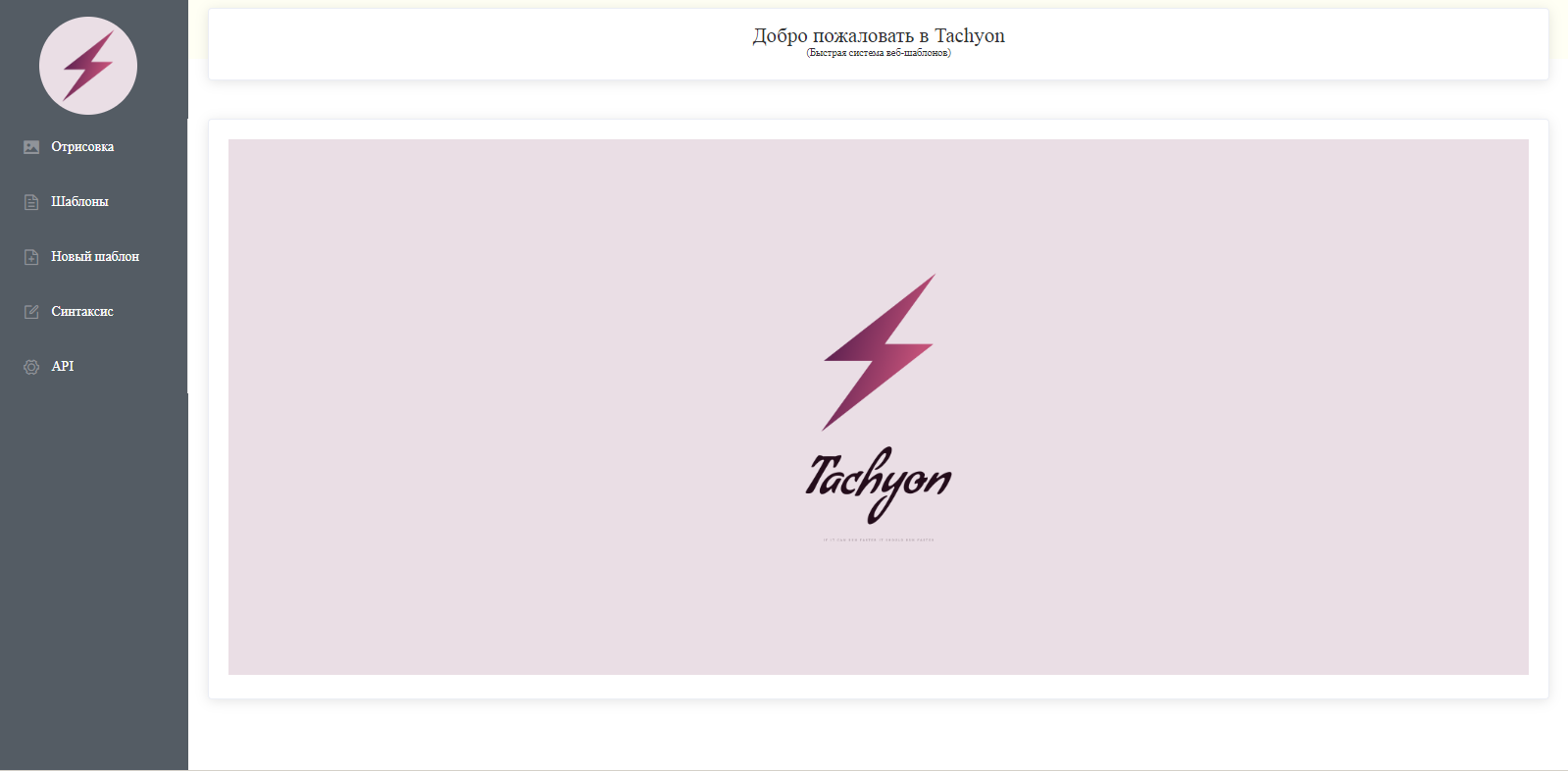


Рис. 2.1.1 Страница по умолчанию, куда попадает пользователь

Далее можно воспользоваться возможностями генерации документа на основе существующих шаблонов. На рис. 2.1.2 изображён интерфейс построение документа, основном элементом страницы является инструмент для ввода JSON, данный инструмент поддерживает несколько режимов отображения, таких как дерево, код, или форма, также в нем реализована проверка введённого JSON на корректность. В правой части страницы находятся кнопки загрузки шаблона из файла, запуска генерации документа и перегрузки шаблонов. Помимо этого, в этой же части находится выпадающий список со списком доступных шаблонов. После ввода данных в формате JSON и выбора интересующего шаблона можно запустить генерацию веб-документа.



Рис. 2.1.2 Интерфейс ввода данных для построения документа

На рис. 2.1.3 продемонстрирован сгенерированный документ, который выводится поверх интерфейса ввода данных.

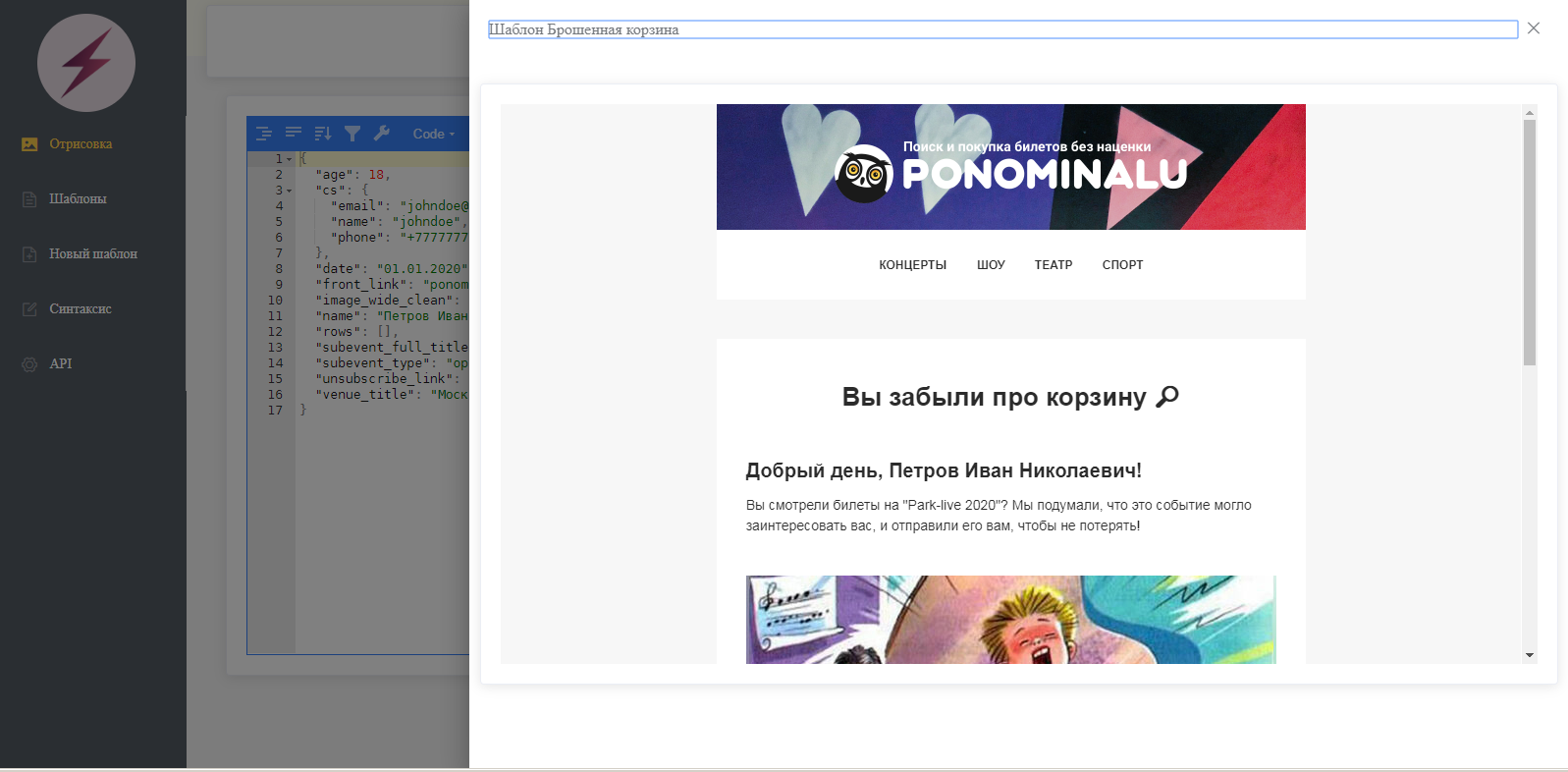


Рис. 2.1.3 Сгенерированный документ, поверх интерфейса ввода данных

Чтобы подробно ознакомится со списком доступных шаблонов необходимо перейти в раздел «Шаблоны». Тут же можно отредактировать интересующий шаблон (рис. 2.1.4).

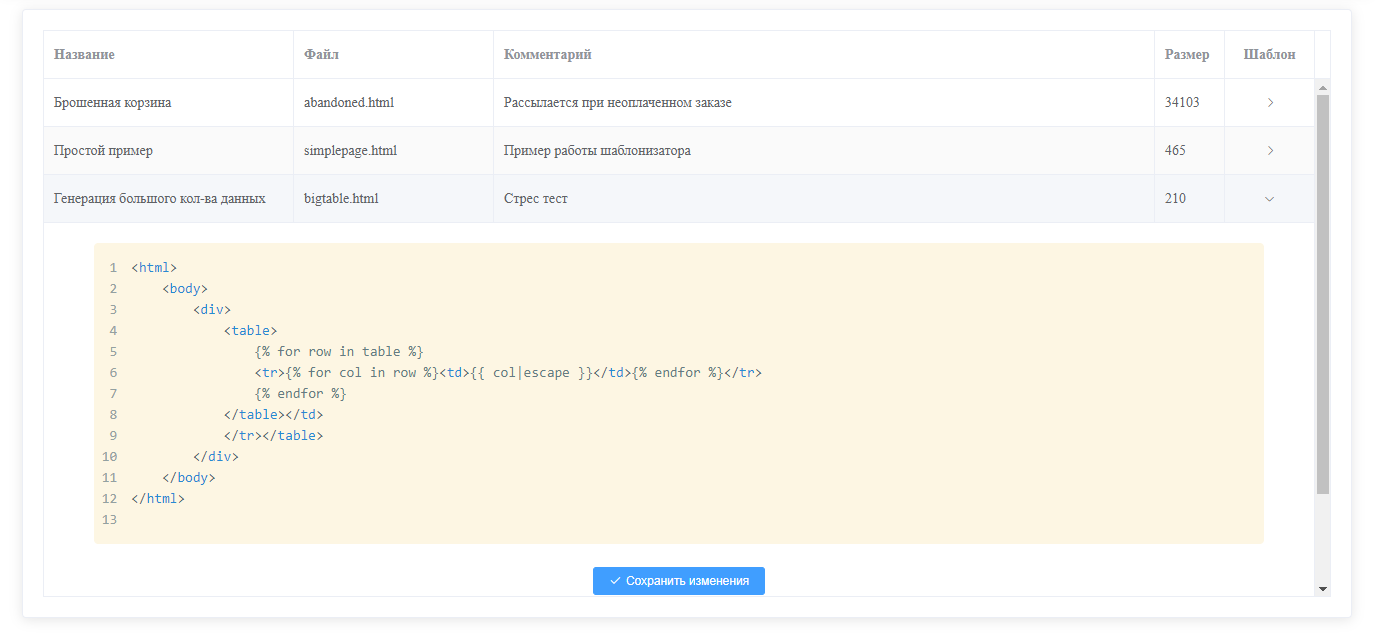


Рис. 2.1.4 Интерфейс со списком существующих шаблонов, и возможностью их редактирования

В случае если нужно добавить новый шаблон, то необходимо открыть раздел «Новый шаблон», интерфейс которого изображён на рис. 2.1.5. Здесь доступен текстовый ввода шаблона, текстовый ввод тестовых данных, название шаблона, название файла шаблона и комментарий к шаблону. Также шаблон, как тестовые данные можно выгрузить из файла.

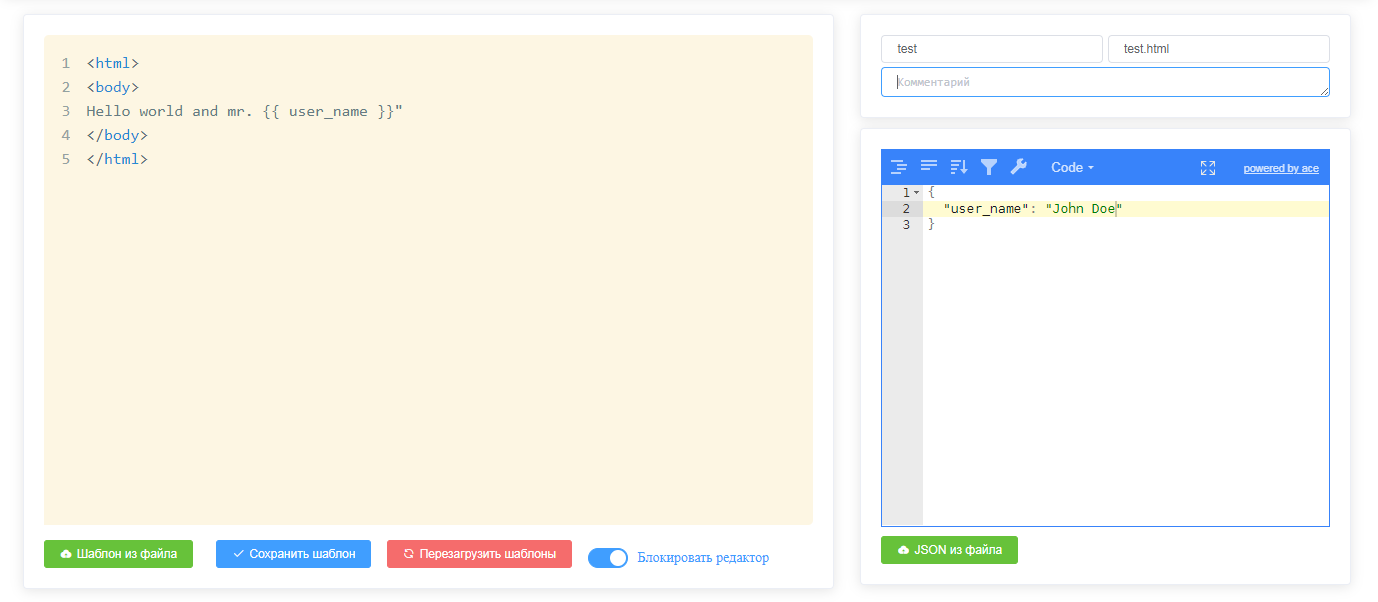


Рис. 2.1.5 Интерфейс создания нового шаблона

Для ознакомления с синтаксисом языка шаблонизации, требуется открыть раздел «Синтаксис» (рис. 2.1.6), где каждая пункт подробно расписан с примерами. Похожая документация существует и для API, который предоставляет веб-сервис (рис. 2.1.7).

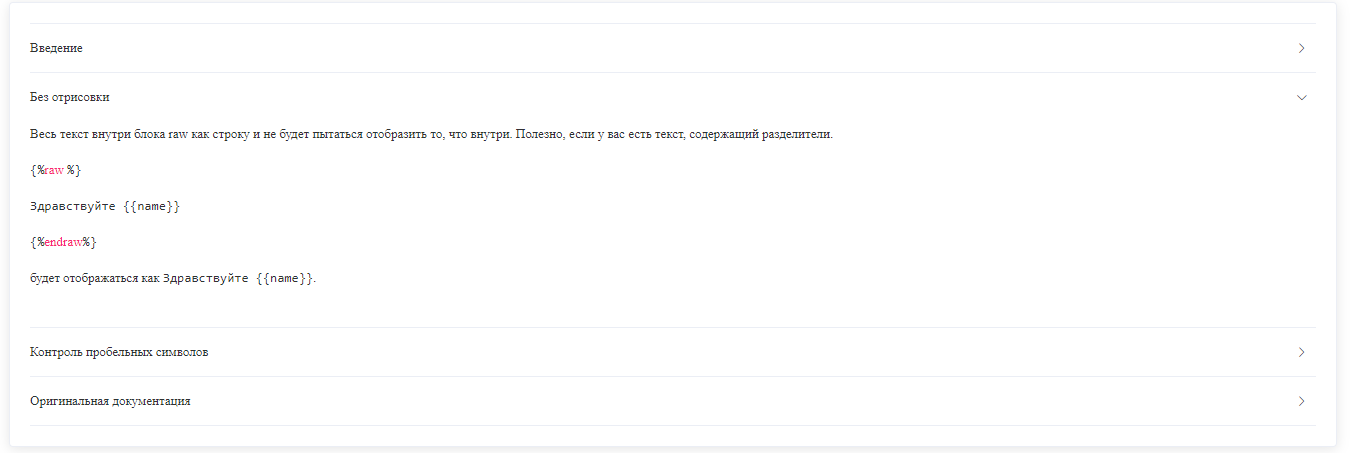


Рис. 2.1.6 Интерфейс для ознакомления с документацией

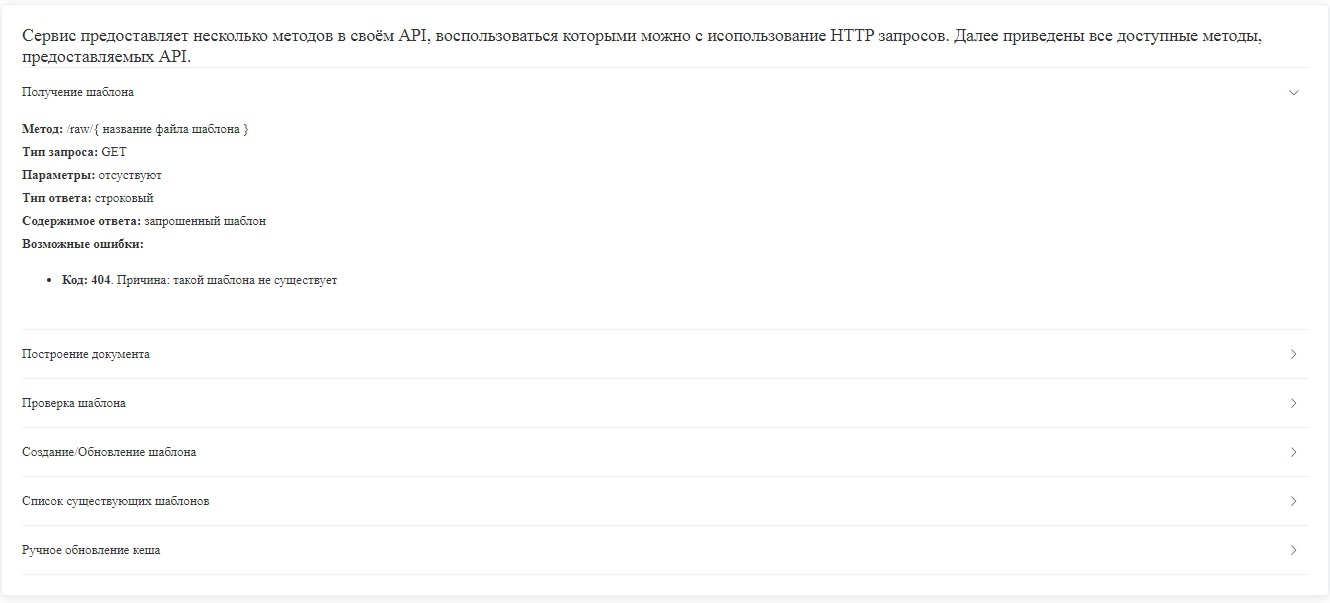


Рис. 2.1.7 Документация API

В результате был разработан асинхронный веб-сервис на языке Rust, который удобен в применении и теоретически должен обладать высокими показателями производительности. Для упрощения взаимодействия с веб-сервисом к нему был разработан веб-интерфейс, который позволяет использовать приложение людям не технической специальности. В следующем разделе будут произведены тесты, с целью оценки показателей производительности разработанной системы шаблонизации.

## Результат разработки, оценка производительности

Для оценки производительности необходимо создать нагрузку на веб-сервер, которая будет имитировать реальную работу в условиях эксплуатации. Основная суть теста заключается в инкрементном увеличении количества одновременных запросов на API веб-сервиса, и параллельный мониторинг времени ответа веб-сервиса и количества потребляемых вычислительных ресурсов на сервере. Так как сложно создать интенсивный трафик из одной точки, для проведения теста будет использованы интернет сервисы по тестированию веб-приложений[8].

В целях проведения нагрузочных тестов, разработанное приложение было установлено на виртуальную машину в облачном сервисе Google Cloud Platform. Характеристики машины, следующие:

* двухъядерный процессор от Intel, поколения Kaby Lake, с тактовой частотой два гигагерца на ядро;
* два гигабайте оперативной памяти;
* гигабитный интернет канал;
* операционная система – Debian Stretch.

Для проведения первого теста использовался сервис K6 который позволяет без взимания оплаты проводить нагрузочные тесты с использованием до пятидесяти одновременных подключений. Настройки теста выглядят как изображено на рис. 2.2.1 нагрузка начнёт постепенно увеличиваться в течении пяти минут и в пиковый момент достигнет пятидесяти одновременных подключений. Обращение к серверу, который расположен в Стокгольме, будет производится из Франкфурта. Каждый запрос, сделанный к веб-сервису представляет из себя запрос на построение веб-документа, который отображает таблицу, содержащую две тысячи ячеек. Такой объём таблицы позволяет оценить производительность самого шаблонизатора. Результат теста изображён в виду графика на рис. 2.2.2. По графику видно, что количество подключений постепенно возрастает, но при этом время ответа веб-сервиса остаётся постоянным в пределах семидесяти миллисекунд.



Рис. 2.2.1 Тест №1, настройки нагрузки



Рис. 2.2.2 Тест №1, 50 одновременных подключений

На рис. 2.2.3 и рис. 2.2.4 показано состояние вычислительной машины, на которой работает веб-сервис, по ним видно, что даже в момент максимальной нагрузки, потребление веб-сервисом процессорного времени не превышает 10%. Отсюда можно сделать вывод о том, что даже при пятидесяти одновременных подключениях, разработанный веб-сервис хорошо справляется со своей задачей.



Рис. 2.2.3 Состояние системы, тест № 1, начало



Рис. 2.2.4 Состояние системы, тест № 1, пиковая нагрузка

В связи с ограничением в пятьдесят одновременных подключений, для дальнейшего тестирования использовался другой сервис LoaderIO, который имеет ограничение по одновременным подключениям равным десяти тысячам. Соответственно второй тест производился с использованием ста новых подключений каждую секунду, что привело к ситуации, в которой одновременно веб-сервис обрабатывал двести запросов. Обращение к веб-сервису происходило из штата Вирджиния в США. Результат теста показан на рис. 2.2.5.



Рис. 2.2.5 Тест №2, 100 одновременных подключений

Время ответа сервиса колеблется в пределах двух тысяч миллисекунд, что по большей части связано с большой дистанцией между Вирджинией и Стокгольмом. Нагрузку на сервер можно увидеть на рис. 2.2.6, по которому видно что в потребление процессорного времени веб-сервисом находится в пределах 20%.



Рис. 2.2.6 Тест №2, состояние системы, пиковая нагрузка

Далее количество новых подключений в секунду было увеличено до двухсот. График времени ответа показан на рис. 2.2.7. Время ответа сервиса чуть превысило две тысячи миллисекунд, а на нагрузка на систему изображена на рис. 2.2.8. Потребление процессорного времени составило чуть более 50%.



Рис. 2.2.7 Тест № 3, 200 одновременных подключений



Рис. 2.2.8 Тест №3, состояние системы, пиковая нагрузка

Чтобы найти верхний предел возможностей веб-сервиса по обслуживанию запросов, количество одновременных подключений было увеличено до пятисот пятидесяти. График результатов показан на рис. 2.2.9, из него видно что время ответа сервиса возросло до четырёх тысяч пятисот миллисекунд.



Рис. 2.2.9 Тест № 4, 550 одновременных подключений

Нагрузка на систему также возросла, до 100% процессорного времени, и полной загрузки одного из ядер процессора (рис. 2.2.10).



Рис. 2.2.10 Тест № 4, состояние системы, одно ядро полностью нагружено

Последним тестом было увеличение одновременных подключений до семисот. При этом произошёл резкий рост времени ответа до более чем двенадцать секунда на запрос (рис. 2.2.11), и появлением ошибок в ответах, связанных с не способностью сервера обработать такое количество запросов. Загрузка процессора показана на рис. 2.2.13, по нему видно что оба ядра процессора полностью нагружены. Возникновение ошибок связано с ограничением операционной системе на количество единовременно открытых файлов на процесс, которое по умолчанию составляет тысячу файлов. Данный лимит можно увеличить, но к снижению времени ответа это не приведёт ввиду того что процессор все равно не сможет обработать большее количество запросов.

По результатам тестов можно сделать вывод что разработанный веб-сервис обладает достаточно высокими показателями производительности, и может обрабатывать большое количество запросов одновременно. Эти показатели можно улучшить за счёт установки приложения на более производительное аппаратное обеспечение и расположением клиентов в локальной сети.



Рис. 2.2.11 Тест № 5, 700 подключений, деградация производительности



Рис. 2.2.12 Тест № 5, наличие ошибок в ответе от сервиса



Рис. 2.2.13 Тест №5, состояние системы, оба ядра полностью нагружены

ЗАКЛЮЧЕНИЕ

Переда началом разработки было произведено моделирование работы системы шаблонизации с использование функциональных диаграмм в нотации IDEF0/DFD. После чего, с использованием языка программирования Rust и библиотек, существующих в экосистеме данного языка, был разработан асинхронный высокопроизводительный RESTful веб-сервис для шаблонизации, способный быстро обрабатывать большое количество одновременных запросов. Для удобства взаимодействия с данным веб-сервисом, к нему был разработан графический пользовательский интерфейс с применением JavaScript фреймворка Vue.js.

По окончанию разработки было произведено тестирование производительности веб-сервиса с направлением на него большого количества «тяжёлых» запросов одновременно. Результаты теста показали, что разработанный веб-сервис может одновременно справляться с очень высокими нагрузками, даже не на очень производительном аппаратном обеспечении.

СПИСОК ЛИТЕРАТУРЫ

1. ГОСТ Р ИСО/МЭК 25010-2015 Информационные технологии (ИТ). Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов.
2. Систематика и обобщение методологий моделирования и языков программирования [электронный ресурс]. Режим доступа: <http://www.sciteclibrary.ru/rus/catalog/pages/10764.html>, свободный (дата обращения: 30.04.2020).
3. Машнин Тимур Сергеевич. Технология Web-сервисов платформы Java. – БХВ-Петербург, 2012. – С. 115. – 560 с.
4. Блэнди Джим, Орендорф Джейсон. Программирование на языке Rust. – ДМК Пресс, 2018. – 550 с.
5. Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching Strategies to Improve Disk System Performance – Computer, 1994. – 46 с.
6. Introduction – Vue.js [электронный ресурс]. Режим доступа: <https://vuejs.org/v2/guide/>, свободный (дата обращения: 01.05.2020).
7. Guide | Element [электронный ресурс]. Режим доступа: <https://element.eleme.io/#/en-US/guide/design>, свободный (дата обращения: 01.05.2020).
8. ГОСТ Р 56920-2016/ISO/IEC/IEEE 29119-1:2013 Системная и программная инженерия. Тестирование программного обеспечения.

|  |  |  |  |
| --- | --- | --- | --- |
| Практикант | Махмудов Бобурбек Нодирбекович |  |  |
|  |  |  | *(подпись)* |

Отчет о прохождении практики сдан «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Руководители практики:

|  |  |  |  |
| --- | --- | --- | --- |
| от Университета | Бумарин Д.П., к.т.н., доц. |  |  |
|  |  |  | *(подпись)* |