|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана (национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | *Робототехника и комплексная автоматизация* |
|  |  |
| КАФЕДРА | *Системы автоматизированного проектирования (РК-6)* |

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ***

«Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | РК6-82Б | |  |  | М.Т. Идрисов |
|  | (Группа) | |  | (Подпись, дата) | (И.О.Фамилия) |
|  | |  |  |  |  |
| Руководитель ВКР | | |  |  | А.П. Соколов |
|  | |  |  | (Подпись, дата) | (И.О.Фамилия) |
|  | |  |  |  |  |
| Консультант | | |  |  | А.Ю. Першин |
|  | |  |  | (Подпись, дата) | (И.О.Фамилия) |
|  | |  |  |  |  |
| Нормоконтролер | | |  |  | С.В. Грошев |
|  | |  |  | (Подпись, дата) | (И.О.Фамилия) |

*Москва, 2020 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | УТВЕРЖДАЮ | | |
|  |  | Заведующий кафедрой | | *РК-6* |
|  |  |  | | (Индекс) |
|  |  |  | |  |
|  |  |  | *А.П. Карпенко* | |
|  |  |  | (И.О.Фамилия) | |
|  |  |  |  | |
|  |  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ *2020* г. | | |

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы**

Студент группы РК6-82Б

|  |
| --- |
| *Идрисов Марат Тимурович* |
| (Фамилия, имя, отчество) |

Тема выпускной квалификационной работы

*Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям*

Источник тематики (кафедра, предприятие, НИР): кафедра

Тема квалификационной работы утверждена распоряжением по факультету РК № 03.04.01 – 02/104\_ВКР\_Б от «25» октября *2020* г.

***Часть 1.*** *Аналитический обзор литературы*

*Требуется провести анализ существующих методов решения рассматриваемой задачи: представить ссылки на работы, описывающие методы решения аналогичных задач, представить известные программные системы, обеспечивающие решение аналогичных задач. В результате проведенного обзора литературы должна быть обоснована актуальность тематики».*

***Часть 2.*** *Постановка задачи, разработка архитектуры программной реализации, программная реализация*

*В рамках раздела должна быть представлена постановка задачи, включающая описание исходных данных и планируемых к получению требуемых результатов. Раздел должен содержать описание общих алгоритмов и принципов создания требуемого программного обеспечения (ПО), предлагаемую автором архитектуру ПО, а также подробное описание отдельных ключевых элементов созданного ПО».*

***Часть 3.*** *Проведение отладки и тестирования*

*Для демонстрации работоспособности созданных программных решений автором должно быть проведено тестирование с использованием данных, размещённых в множестве gitlab-репозиториев, доступ к которым будет предоставлен руководителем.*

***Оформление курсовой работы:***

Расчетно-пояснительная записка на 44 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

|  |
| --- |
| *4 графических листа* |
|  |
|  |
|  |
|  |

Дата выдачи задания «18» февраля 2020 г.

|  |  |  |  |
| --- | --- | --- | --- |
| **Студент** | |  | М.Т. Идрисов |
|  |  | (Подпись, дата) | (И.О.Фамилия) |
| **Руководитель выпускной квалификационной работы** | |  | А.П. Соколов |
|  |  | (Подпись, дата) | (И.О.Фамилия) |

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ФАКУЛЬТЕТ** | **РК** |  | УТВЕРЖДАЮ | | |
| **КАФЕДРА** | **РК-6** |  | Заведующий кафедрой | | *РК-6* |
| **ГРУППА** | **РК6-82Б** |  |  | | (Индекс) |
|  |  | |  |
|  |  |  |  | *А.П. Карпенко* | |
|  |  |  |  | (И.О.Фамилия) | |
|  |  |  |  |  | |
|  |  |  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ *2020* г. | | |

**КАЛЕНДАРНЫЙ ПЛАН**

**выполнения выпускной квалификационной работы**

|  |  |
| --- | --- |
| студента: | *Идрисов Марат Тимурович* |
|  | (Фамилия, имя, отчество) |

Тема выпускной квалификационной работы: «Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ п/п** | **Наименование этапов выпускной квалификационной работы** | **Сроки выполнения этапов** | | **Отметка о выполнении** | |
| **план** | **факт** | **Должность** | **ФИО, подпись** |
|  | Задание на выполнение работы. Формулирование проблемы, цели и задач работы | *18.02.2020*  *Планируемая дата* | *18.02.2020* | Руководитель ВКР | *А.П. Соколов* |
|  | 1 часть: *аналитический обзор литературы* | *18.02.2020*  *Планируемая дата* | *31.03.2020* | Руководитель ВКР | *А.П. Соколов* |
|  | Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач | *28.02.2020*  *Планируемая дата* | *28.02.2020* | Заведующий кафедрой | *А.П. Карпенко* |
|  | 2 часть: *постановка задачи, разработка архитектуру программной реализации, программная реализация* | *31.03.2020*  *Планируемая дата* | *31.03.2020* | Руководитель ВКР | *А.П. Соколов* |
|  | 3 часть: *проведение отладки и тестирования* | *30.04.2020*  *Планируемая дата* | *30.04.2020* | Руководитель ВКР | *А.П. Соколов* |
|  | 1-я редакция работы | *31.05.2020*  *Планируемая дата* | *31.05.2020* | Руководитель ВКР | *А.П. Соколов* |
|  | Подготовка доклада и презентации | *17.06.2020*  *Планируемая дата* | *17.06.2020* |  |  |
|  | Заключение руководителя | *15.06.2020*  *Планируемая дата* | *15.06.2020* | Руководитель ВКР | *А.П. Соколов* |
|  | Допуск работы к защите на ГЭК (нормоконтроль) | *15.06.2020*  *Планируемая дата* | *15.06.2020* | Нормоконтролер | *С.В. Грошев* |
|  | Внешняя рецензия | *12.06.2020*  *Планируемая дата* | *28.06.2020* |  |  |
|  | Защита работы на ГЭК | *25.06.2020*  *Планируемая дата* | *29.06.2020* |  |  |

|  |  |  |
| --- | --- | --- |
| *Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Т. Идрисов* |  | *Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.П. Соколов* |
| (подпись, дата) |  | (подпись, дата) |

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**НАПРАВЛЕНИЕ НА ЗАЩИТУ  
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

**Председателю**

**Государственной Экзаменационной Комиссии № \_\_\_\_\_**

факультета «Робототехника и комплексная автоматизация» МГТУ им. Н.Э. Баумана

Направляется студент *Идрисов Марат Тимурович* группы *РК6-82Б*

на защиту выпускной квалификационной работы «Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям»

Декан факультета *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**Справка об успеваемости**

Студент *Идрисов Марат Тимурович* за время пребывания в МГТУ имени Н.Э. Баумана

с *2017* г. по *2020* г. полностью выполнил учебный план со следующими оценками:   
отлично –  %, хорошо –  %, удовлетворительно –  %.

Инспектор деканата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отзыв руководителя выпускной квалификационной работы**

*Студент Идрисов М.Т. в процессе выполнения ВКР реализовал поставленные ему задачи самостоятельно, в полном объёме, в полном соответствии с заданием и календарным планом. Автором создано программное обеспечение, работоспособность которого подтверждена результатами его тестирования на основе данных, выгружаемых из множества gitlab-проектов. Среди планируемых направлений развития следует отметить: доработку в области поддержки дополнительных независимых шаблонов, на основе которых должны формироваться конечные документы. Считаю, что автор достоин «отличной» оценки и присвоения квалификации бакалавр по направлению «Информатика и вычислительная техника».*

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель ВКР |  | А.П. Соколов | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |
|  | (подпись) | (ФИО) | (дата) |
| Студент |  | М.Т. Идрисов | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |
|  | (подпись) | (ФИО) | (дата) |

**Заключение кафедры о выпускной квалификационной работе**

Выпускная квалификационная работа просмотрена и студент *Идрисов М.Т.* может быть

допущен к защите этой работы в Государственной Экзаменационной Комиссии.

Зав. кафедрой **\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г.

### РЕФЕРАТ

Работа посвящена разработке приложения для автоматизации процесса обработки и сбора постоянно формируемой отчетной документации с последующей её интерпретацией, систематизацией и объединением в единые документы. Предполагается, что формируемые таким образом документы, позволят наглядно, и, что самое главное, в сжатой форме, демонстрировать во времени процессы проведения исследований по разным научным и/или образовательным направлениям, развиваемым в некотором подразделении. В результате анализа был предложен и реализован в программном виде подход обработки и создания отчётной документации. Разработано web-приложение, позволяющие в реальном времени создавать итоговые отчеты.

Тип работы: выпускная квалификационная работа.

Тема работы: Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям

Объект исследований: автоматизированные методы создания отчетной документации.

### СОДЕРЖАНИЕ

[РЕФЕРАТ 6](#_Toc44252957)

[СОДЕРЖАНИЕ 7](#_Toc44252958)

[ВВЕДЕНИЕ 8](#_Toc44252959)

[1. ПОСТАНОВКА ЗАДАЧИ 16](#_Toc44252960)

[1.1. Концептуальная постановка задачи 16](#_Toc44252961)

[2. АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ 17](#_Toc44252962)

[2.1. Обзор микросервисной архитектуры 17](#_Toc44252963)

[2.2. Создание docker-образа приложения 18](#_Toc44252964)

[2.3. Разработка основного приложения 22](#_Toc44252965)

[2.4. Описание требований, предъявляемые к отчетной документации 31](#_Toc44252966)

[2.5. Конфигурация группы docker-контейнеров 33](#_Toc44252967)

[2.6. Внедрение приложения в цикл непрерывной интеграции 36](#_Toc44252968)

[3. ТЕСТИРОВАНИЕ И ОТЛАДКА 37](#_Toc44252969)

[АНАЛИЗ РЕЗУЛЬТАТОВ 39](#_Toc44252970)

[ЗАКЛЮЧЕНИЕ 40](#_Toc44252971)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41](#_Toc44252972)

[ПРИЛОЖЕНИЕ А 45](#_Toc44252973)

### ВВЕДЕНИЕ

Современный образовательный процесс неизбежно сопряжен с активным применением информационных технологий, обеспечивающих автоматизацию как самой образовательной деятельности, так и процессы формирования студенческой электронной отчетной документации тех или иных типов: расчетно-пояснительные записки, формируемые в рамках курсовых проектов, курсовых работ, домашних заданий, лабораторных работа и пр. Со всё более явным и массовым переходом к гибридным формам образовательного процесса, включающего элементы дистанционного образования, количество электронных документов указанных типов существенно возросло. В указанных условиях актуальным становится решение задач систематизации документов рассматриваемых типов и их содержания.

В ввиду большие количества генерируемых документов не представляется возможным применение ручных способов их систематизации. В результате, например, даже в случае применения современных систем контроля версий Git, сохраняемые многочисленные документы в многочисленных репозиториях большим числом студентов и преподавателей будучи размещёнными в них уже очень скоро «затеряются» среди прочих. В результате таким образом организованная научно-образовательная деятельность не позволит создать основу для последовательной генерации новых знаний, а также не позволит создать условия для адекватного принятия управленческих решений на основе результатов выполненных работ тех или иных типов.

Для решения существующей проблемы необходимо программное обеспечение для автоматизации процесса обработки и сбора постоянно формируемой отчетной документации с последующим их объединением в единые документы. Предполагается, что формируемые таким образом документы, позволят наглядно, и, что самое главное, в сжатой форме, демонстрировать во времени процессы проведения исследований по разным научным направлениям, развиваемым в некотором подразделении.

Рассматривая существующие разработки в этом направлениии следует обратить внимание на отчет консалтинговой компании DataPine [1] за 2020 год. Компания выделяет 5 наиболее прогрессивных решений для создания отчетности на основе массива данных: Tableau [2], Microsoft Power BI [3], Board [4], SAS Visual Analytics [5] и Oracle Analytics [6]. Данные системы интерактивной аналитики позволяют в кратчайшие сроки проводить глубокий и разносторонний анализ больших массивов информации, строить отчеты и не требуют обучения пользователей и дорогостоящего внедрения. Вышеуказанные системы различаются в цене за использование, а также количеством возможных функций и поддерживаемых источников данных.

Такие системы также подразделяются на два типа: облачные и клиентские.

Клиентское приложение — программное обеспечение, которое инсталлируется на рабочую станцию пользователя и запускается локально, или запускается удаленно. К таким система можно отнести: Tableau и Microsoft Power BI.

Облачные приложения — программное обеспечение, размещенное на удаленном сервере, предоставляются интернет-пользователю как онлайн-сервис. Программы запускаются и выдают результаты работы в окне web-браузера на локальном ПК. Все необходимые для работы приложения и их данные находятся на удаленном сервере и временно кэшируются на клиентской стороне. К таким системам можно отнести Oracle Analytics Сloud и SAS Visual Analytics. Облачные приложения имею ряд преимуществ по сравнению с традиционными desktop-приложениями [7]:

* возможность доступа к данным c любого компьютера, имеющего выход в интернет;
* возможность организации совместной работы с данными;
* высокая вероятность сохранения данных даже в случае аппаратных сбоев;
* все процедуры по резервированию и сохранению целостности данных предоставляются разработчиком приложения, который не вовлекает в этот процесс клиента.

Как видно выше облачные приложения имеют сильно конкурентное преимущество оп сравнению с клиентскими. Конечному пользователю нет необходимости заботиться о работе приложения. Достаточно иметь рабочую машину с выходом в интернет, чтоб воспользоваться приложением. Такое преимущество связано с правильным выбором архитектурного паттерна при разработке приложения. В данной работе так же немаловажно задуматься о выборе архитектурного паттерна. От правильно выбранной архитектуры зависит дальнейшая возможность разработки и эксплуатации приложения.

Крупнейший мировой лидер в области облачных вычислений Amazon Web Services в своем отчете за 2018 год [8] на сегодняшний момент выделяет два типа архитектур для проектирования приложений:

* монолитная архитектура;
* микросервисная архитектура.

Концепция монолитного программного обеспечения заключается в том, что различные компоненты приложения объединяются в одну программу на одной платформе. Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения [9]. Все части программного обеспечения унифицированы, и все его функции управляются в одном месте. В статье [10] автор подробно описывает применение монолитной архитектуры для разработки приложения для автоматизации сбора и анализа данных. Также в работах Таненбаума [11] и Осипова [12] приводятся сравнительный анализ микросервисной и монолитной архитектуры. Авторы выделяют следующие достоинства:

* **простота разработки**: IDE и другие инструменты разработки сосредоточены на построении единого приложения;
* **легкость внесения радикальных изменений**: авторы демонстрируют легкость изменения кода и структуры базы данных, а затем сборку и развёртывания полученного результата;
* **простота тестирования**: авторы работ написали сквозные тесты, которые запускали приложение, обращались к REST API и проверяли пользовательский интерфейс с помощью Selenium;
* **простота развертывания**: авторам достаточно было скопировать все файлы на сервер.

Но у такого подхода есть огромный недостаток. Успешные приложения имею склонность вырастать из монолитной архитектуры. С каждым разом в проект добавляются новые возможности, и кодовая база проекта увеличивается. В своей книге [13] выделил недостатки монолитной архитектуры.

* **Высокая сложность**: проект становится слишком большим, чтобы его мог понять один разработчик. Исправление ошибок и реализация новых возможностей занимает много времени. Сложность повышается экспоненциально и с каждое изменение усложняет код и делает его менее понятным.
* **Длинный и тяжелый путь от сохранения изменений до их развертывания**: «переход» от готового кода к промышленной среде предполагает существенные трудозатраты. Работа такого большого количества программистов над одной и той же кодовой базой часто приводит к тому, что сборку нельзя выпустить вовремя.
* **Длительное тестирование**: код настолько сложен, а эффект от внесенного изменения так неочевиден, что разработчикам и серверу непрерывной интеграции приходится выполнять весь набор тестов, а тестировать измененный компоненты.
* **Трудности с масштабированием**: требования к ресурсам разных программных модулей конфликтуют между собой. Например, модуль обработки изображений сильно нагружает ЦПУ и в идеале должен работать на серверах с большими вычислительными ресурсами. Но, поскольку эти модули входят в одно и то же приложение, приходится идти на компромисс при выборе серверной конфигурации.
* **Сложно добиться надежности приложения**: из-за большого размера приложения его сложно как следует протестировать. Недостаточное тестирование означает, что ошибки попадают в итоговую версию программы. Время от времени ошибка в одном модуле (например, утечка памяти) приводит к поочередному сбою всех экземпляров системы.
* **Зависимость от постепенно устаревающего стека технологий**: монолитная архитектура, заставляет использовать постепенно устаревающий стек технологий. При этом разработчикам сложно переходить на новые фреймворки и языки программирования. Переписать все монолитное приложение, применив новые и, предположительно, лучшие технологии, было бы чрезвычайно дорого и рискованно. Как следствие, приходится работать с теми инструментами, которые были выбраны при запуске проекта. Из-за этого часто приходится поддерживать код, написанный с помощью устаревших средств.

В первые микросервисную архитектуру описали Майкл Т. Фишер и Мартин Л. Эббот в своей книге [14]. Ее ключевой идеей было трехмерное представление модели масштабирования приложения в виде куба. В соответствии с ним масштабирование по оси Y обозначает разбиение приложения на сервисы. Сейчас такой подход кажется довольно очевидным.

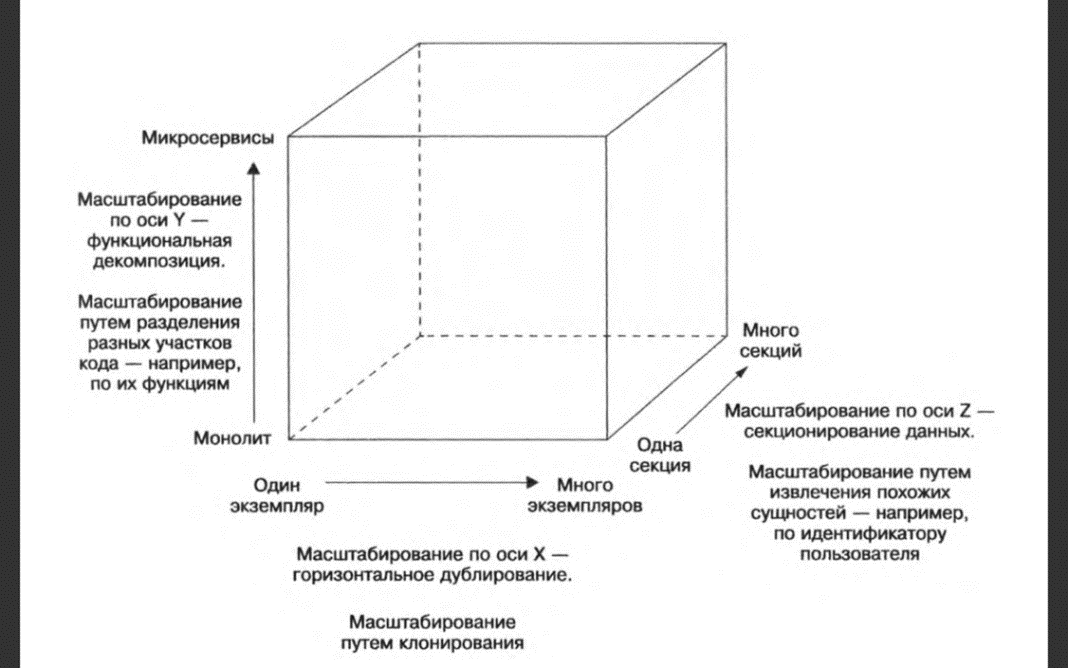


Рисунок 1 Модель определяет три направления для масштабирования приложения: масштабирование по оси X распределяет нагрузку между несколькими идентичными экземплярами, по оси Z — направляет запросы в зависимости от их атрибутов, ось Y разбивает приложение на сервисы с разными функциями

Масштабирование по оси X часто применяют в монолитных приложениях. Принцип работы этого подхода показан на рисунке 2. Балансировщик нагрузки распределяет запросы между N одинаковыми экземплярами. Это отличный способ улучшить мощность и доступность приложения.

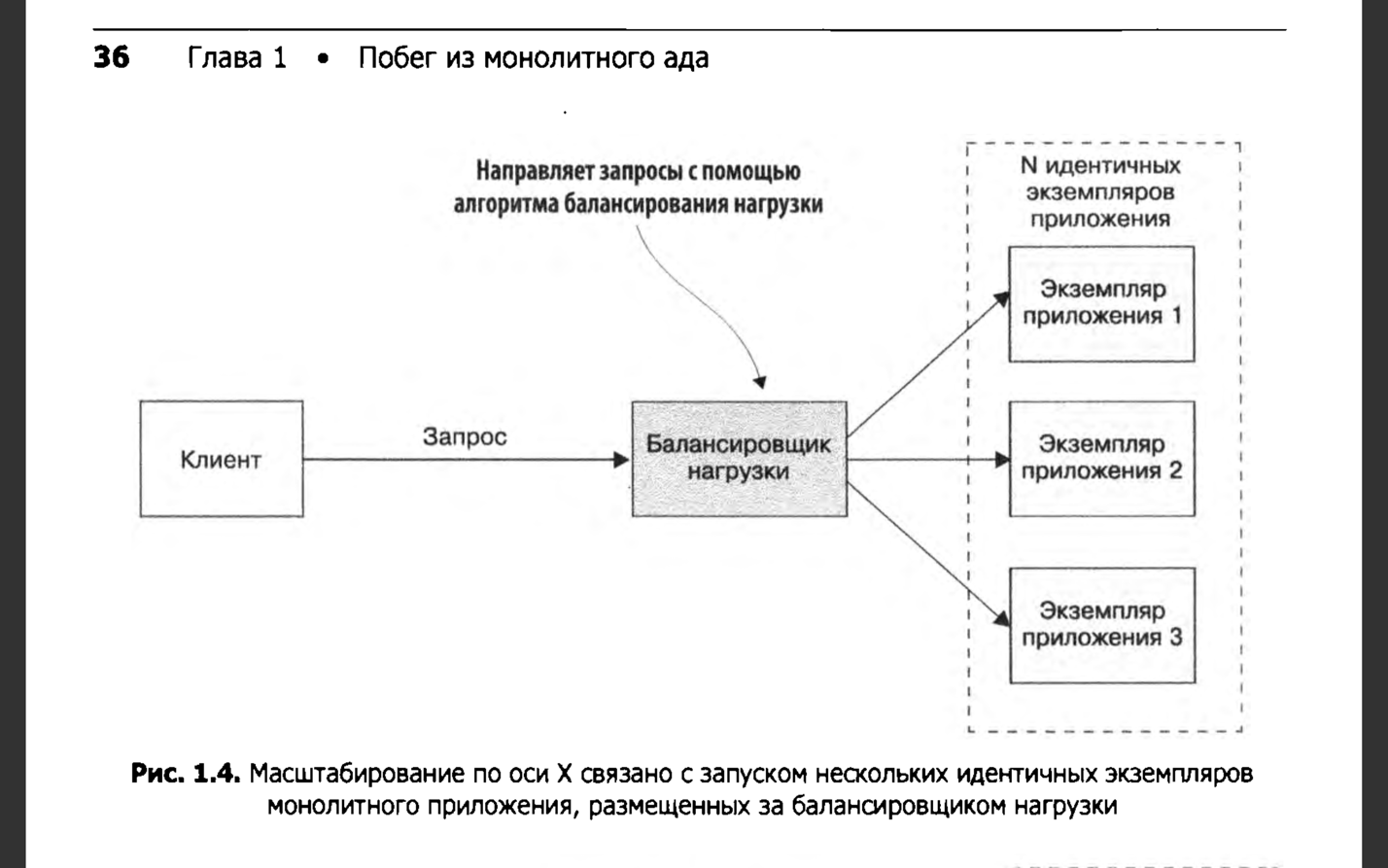


Рисунок 2 Масштабирование по оси X связано с запуском нескольких идентичных экземпляров монолитного приложения, размещенных за балансировщиком нагрузки

Масштабирование по *оси Z* тоже предусматривает запуск нескольких экземпляров монолитного приложения, но в этом случае, в отличие от масштабирования по оси X, каждый экземпляр отвечает за определенное подмножество данных (рисунок 2).



Рисунок 3 Масштабирование по оси Z связано с запуском нескольких идентичных экземпляров монолитного приложения, размещенных за маршрутизатором, который направляет запросы в зависимости от их атрибутов

Масштабирование по осям *X* и *Z* увеличивает мощность и доступность приложения. Но ни один из этих подходов не решает проблем с усложнением кода и процесса разработки. Чтобы справиться с ними, следует применить масштабирование по *оси* Y, или *функциональную декомпозицию* (разбиение). То, как это работает, показано на рисунке 4: монолитное приложение разбивается на отдельные сервисы.



Рисунок 4 Масштабирование по оси Y разбивает приложение на отдельные сервисы. Каждый из них отвечает за определенную функцию и масштабируется по оси X  
(а также, возможно, по оси Z)

В апреле 2012 года Ридчарсон описал этот метод проектирования в своем докладе под названием «Декомпозиция приложений для улучшения развертываемости и масштабируемости» [15]. На тот момент у подобной архитектуры не было общепринятого названия.

Термин «микросервис» [16] впервые использовался во время выступления Фреда Джорджа на конференции Oredev 2013 [17].

В январе 2014 года Ридчарсон создал сайт [18], чтобы описать архитектуру и шаблоны проектирования. В марте 2014 Джеймс Льюис и Мартин Фаулер опубликовали статью о микросервисах [19], которая популяризировала этот термин и сплотила сообщество вокруг новой концепции.

Так уже в 2019 году Ричардсон в своей книге [20] рассмотрел основные преимущества микросервисной архитектуры:

* она делает возможными непрерывные доставку и развертывание крупных, сложных приложений;
* сервисы получаются небольшими и простыми в обслуживании;
* сервисы развертываются независимо друг от друга;
* сервисы масштабируются независимо друг от друга;
* микросервисная архитектура обеспечивает автономность команд разработчиков;
* она позволяет экспериментировать и внедрять новые технологии.

Очевидно, что идеальных технологий не существует, поэтому микросервисная архитектура по мнению Ричардсона имеет следующие недостатки.

* Сложность в подборе подходящий набор сервисов: проблема, возникающая при использовании микросервисной архитектуры, связана с отсутствием конкретного, хорошо описанного алгоритма разбиения системы на микросервисы.
* Сложность распределенных систем затрудняет разработку, тестирование и развертывание: недостаток состоит в том, что при создании распределенных систем возникают дополнительные сложности для разработчиков. Сервисы должны использовать механизм межпроцессного взаимодействия. Это сложнее, чем вызывать обычные методы. К тому же проект должен уметь справляться с частичными сбоями и быть готовым к недоступности или высокой латентности удаленного сервиса.
* Развертывание функций, охватывающих несколько сервисов, требует тщательной координации: проблема связана с тем, что развертывание функций, охватывающих несколько сервисов, требует тщательной координации действий разных команд разработки.
* Решение о том, когда следует переходить на микросервисную архитектуру, является нетривиальным: трудность связана с решением о том, на каком этапе жизненного цикла приложения следует переходить на микросервисную архитектуру. Часто во время разработки первой версии вы еще не сталкиваетесь с проблемами, которые эта архитектура решает. Более того, применение сложного, распределенного метода проектирования замедлит разработку.

Резонно заметить, что идеальных технологий не существует и каждая имеет свои плюсы и минусы. Но несмотря на это плюсы использования микросервисной архитектуры нивелируют над ее минусами.

### ПОСТАНОВКА ЗАДАЧИ

### Концептуальная постановка задачи

Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям

Объект разработки: методы создания отчетной документации

**Цель** разработки: создать программное обеспечение для автоматизированного сбора и верстки отчетной документации

**Задачи** выпускной квалификационной работы.

1. Провести обзор литературы по теме: «Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям».
2. Предложить архитектуру разрабатываемого приложения.
3. Разработать приложение, для автоматической систематизации и верстки отчетной документации.
4. Разработать скрипт для введения приложения в цикл непрерывной интеграции.

### АРХИТЕКТУРА ПРОГРАММНОЙ РЕАЛИЗАЦИИ

### Обзор микросервисной архитектуры

Современная архитектура ПО начала отходить от крупных монолитных приложений [21]. Теперь основное внимание в вопросах архитектуры уделяется достижению высокого уровня масштабируемости. «Разбивая монолит» на компоненты, инженерные организации предпринимали усилия по децентрализации управления изменениями, предоставляя командам больше контроля над тем, как функции вводятся в эксплуатацию. Повышая изолированность между компонентами, команды создателей ПО постепенно переходят к разработке распределенных систем, фокусируясь на написании менее крупных, более специализированных сервисов с независимыми циклами выпуска.

В приложениях, оптимизированных для выполнения в облаке, используется набор принципов, позволяющих командам более свободно оперировать способами ввода функций в эксплуатацию. По мере роста степени «распределенности» приложений (в результате повышения степени изолированности, необходимой для предоставления б*о*льшего контроля над ситуацией командам, владеющим приложением) возникает серьезная проблема, связанная с повышением вероятности сбоя при обмене данными между компонентами приложения. Неизбежным результатом является превращение приложений в сложные распределенные системы.

Архитектуры приложений, оптимизированных для работы в облачной среде, придают этим приложениям преимущества исключительно высокой масштабируемости, притом гарантируя их всеобщую доступность и высокий уровень производительности.

* При использовании микросервисной архитектуры можно выделить главные преимущества ее использования [22]:
* модули можно легко заменить в любое время: акцент на простоту, независимость развёртывания и обновления каждого из микросервисов;
* модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;
* модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах: приоритет отдаётся в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;
* архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые.

Философия микросервисов фактически «копирует» философию Unix [23], согласно которой каждая программа должна «делать что-то одно, и делать это хорошо» и взаимодействовать с другими программами простыми средствами: микросервисы минимальны и предназначаются для единственной функции. Основные изменения, в связи с этим, налагаются на организационную культуру, которая должна включать автоматизацию разработки и тестирования, а также культуру проектирования, от которой требуется предусматривать «обход» прежних ошибок, исключение унаследованного кода.

Наиболее популярная среда для выполнения микросервисов — технология Docker [24]. В этом случае каждый из микросервисов изолируется в отдельный контейнер или небольшую группу контейнеров, доступную по сети другим микросервисам и внешним потребителям, и управляется средой оркестрации, обеспечивающей отказоустойчивость и балансировку нагрузки. Типовой практикой является включение в контур среды выполнения системы непрерывной интеграции [25].

### Создание docker-образа приложения

Для создания docker-образа приложения, написанного на Python, требуется построить его поверх существующего образа, которых существует огромное множество. Существуют образы ОС, такие как Ubuntu и CentOS, а также существует множество различных вариантов Python образов.

В результате анализа поставленной задачи были определены ряд критериев выбора базового образа:

1. **Стабильность.** Требуется, чтобы образ в долгосрочной перспективе содержал один и тот же базовый набор библиотек, структуру каталогов и инфраструктуру.
2. **Обновления безопасности:** требуется, чтобы базовый образ получал своевременные обновления безопасности для базовой операционной системы.
3. **Современные зависимости:** при создании сложного приложения существует зависимость от установленных в операционной системе библиотек и приложений. Требуется, чтобы нужные библиотеки имели самые новые и стабильные версии.
4. **Новая версия Python:** этот критерий не является основополагающим, т.к. требуемую версию Python можно установить самостоятельно, но заранее предустановленная версия Python экономит время и усилия на ее ручную установку.
5. **Небольшой размер образа:** при прочих равных условиях лучше иметь образ Docker меньшего размера, чем образ Docker большего размера.

Существуют четыре основные операционные системы, которые соответствуют вышеуказанным критериям.

* Ubuntu 18.04 (ubuntu:18.04 образ) был выпущен в апреле 2018 года, и, поскольку это релиз долгосрочной поддержки, он будет получать обновления безопасности до 2023 года [26];
* Ubuntu 20.04 (ubuntu:20.04 образ) был выпущен в конце апреля 2020 года, и, поскольку это релиз долгосрочной поддержки, он получит обновления безопасности до 2025 года [27];
* CentOS 8 (centos:8) был выпущен в 2019 году и будет иметь полные обновления до 2024 года и обновления до 2029 года [28];
* Debian 10 («Buster») был выпущен 6 июля 2019 года и будет поддерживаться до 2024 года.

Только Ubuntu 20.04 включает в себя последнюю версию Python.

Также существую «официальные python образы» [29], в которых уже установлены нужные версии Python (3.5, 3.6, 3.7, 3.8 бета и т.д.). Они также имею несколько вариацией:

* Debian Buster, с множеством установленных пакетов [30].
* slim вариант Debian Buster. В нем отсутствуют множество общих пакетов, поэтому сам образ намного меньше.

В результате анализа вышеуказанных docker-образов, было принято решение выбрать в качестве основы нашего приложения образ **python:3.8-slim-buster**. Он актуальнее, чем ubuntu:18.04, стабилен, не будет иметь изменений в библиотеках. Объем образа 60 МБ при загрузке и 180 МБ без сжатия [31], что говорит о его небольшом размере. Образ содержит последнюю версию Python и обладает всеми преимуществами Debian Buster.

Конечной задачей работы является получение отчета в формате PDF, для получения которого было принято решение использовать язык верстки LaTeX. Для решения этой задачи требуется дополнить базовый docker-образ соответствующими утилитами пакетами для компиляции LaTeX файлов. Такой утилитой является latexmk и дополнительные пакеты с кириллическими шрифтами: texlive-lang-cyrillic и texlive-latex-recommended. *Dockerfile* с инструкциями для дополнения базового образа представлен в листинге 1.

Листинг 1. Dockerfile с пакетами для компиляции TeX файлов

|  |
| --- |
| FROM python:3**.**8-slim-buster  COPY /tmp /tmp  RUN apt-get update \  && apt-get install с texlive-lang-cyrillic texlive-latex-recommended texlive-pictures texlive-latex-extra -y \  && **cd** "$\_" \  && latex hyphenat**.**ins \  && mkdir -p /usr/share/texlive/texmf-dist/tex/latex/hyphenat \  && mv hyphenat**.**sty /usr/share/texlive/texmf-dist/tex/latex/hyphenat |

Базовый образ **python:3.8-slim-buster** состоит из стека слоев, которые доступны только для чтения (иммутабельны) (рисунок 1), а все изменения происходят в верхнем слое стека

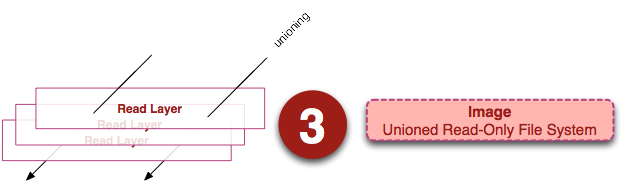


Рисунок 5 Общая структура базового Docker образа

Для дополнения базового образа требуется добавить верхний слой для записи сверху стека слоев (рисунок 2), записать изменения и преобразовать верхний слой в слой для чтения (рисунок 3).



Рисунок 6 Добавление в стек верхнего слоя для записи

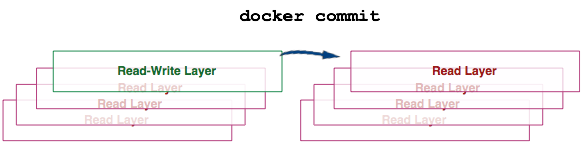


Рисунок 7 Преобразование верхнего слоя в слой для записи

Данную цепочку преобразований, выполняет команда **docker build**. Визуализация работы данной команды представлена на рисунке 4.

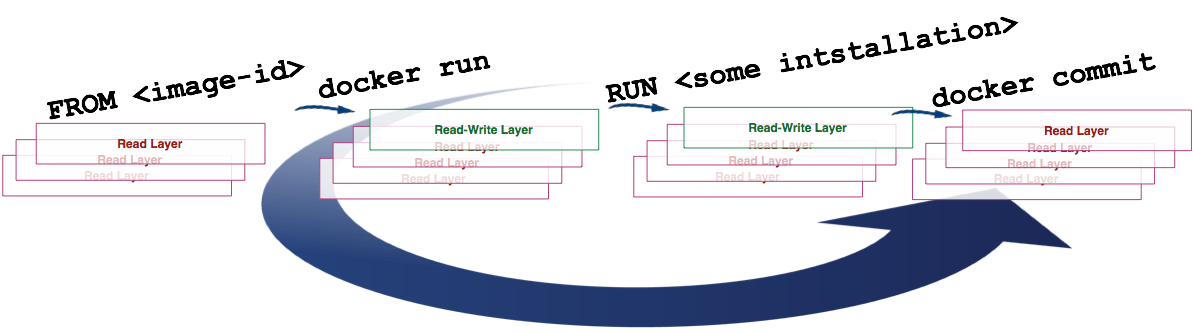


Рисунок 8 Цепочка преобразований, выполненная docker build

Команда **build** использует значение инструкции FROM из файла *Dockerfile* как базовый образ после чего: а) запускает контейнер (create и start); б) изменяет слой для записи; в) вызывает операцию commit.

Таким образом был получен базовый образ, содержащий утилиты для работы с TeX фалами, который был назван **python-latexmk.**

С помощью команды **docker pull** данный образ был размещен в Docker Hub – крупнейшую в мире библиотеку контейнерных образов [32], для возможности его использования в основном проекте.

### Разработка основного приложения

Клиентская часть приложения разработана на языке Python и фреймфорка Django. В качестве сервера используется внутренний сервер, входящий в состав Django. Django также включает в себя технологию ORM, что которая связать базу данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Это позволяет проектировать работу с данными в терминах классов, а не таблиц данных. Такие классы называются моделями.

Для генерации конечного отчетного документа в формате TeX, используется модуль Jinja —шаблонизатор для языка программирования Python. Он подобен шаблонизатору Django, но предоставляет Python-подобные выражения [33].

Отчеты располагаются в репозитории GitLab. Требуется API-интерфейс, позволяющий получать данные и оперировать ими. Gitlab предоставляет API-интерфейс вместе с подробной документацией [34]. Чтобы не работать с “сырыми” get/post запросами, был использован модуль python-gitlab [35], обеспечивающий доступ к API сервера GitLab, используя python-вызовы.

На основе схемы таблиц базы данных были сформированы соответствующие Django-модели:

Листинг 2. Django-модели базы данных.

|  |
| --- |
| **class** Solun(models.Model):  slnid = models.CharField(primary\_key=**True**, max\_length=3)  dscra = models.CharField(max\_length=50)  slnna = models.CharField(unique=**True**, max\_length=20)  cpxid = models.ForeignKey(**'Cmplx'**, models.DO\_NOTHING, db\_column=**'cpxid'**, blank=**True**, null=**True**)  dscrb = models.TextField(blank=**True**, null=**True**)  dscrc = models.TextField(blank=**True**, null=**True**)  dirna = models.CharField(max\_length=70, blank=**True**, null=**True**)  rlvnc = models.CharField(max\_length=3, blank=**True**, null=**True**)   **class** Meta:  db\_table = **'solun'** unique\_together = ((**'slnid'**, **'cpxid'**),)  **class** Tmpls(models.Model):  tmlid = models.CharField(primary\_key=**True**, max\_length=3)  catid = models.ForeignKey(**'Tmcat'**, models.DO\_NOTHING, db\_column=**'catid'**, blank=**True**, null=**True**)  activ = models.NullBooleanField()  rlpth = models.TextField(blank=**True**, null=**True**)  dscra = models.TextField(blank=**True**, null=**True**)  versi = models.CharField(max\_length=14, blank=**True**, null=**True**)  foldr = models.CharField(max\_length=35, blank=**True**, null=**True**)  rlvnc = models.CharField(max\_length=3, blank=**True**, null=**True**)   **class** Meta:  db\_table = **'tmpls'  class** Cmplx(models.Model):  cpxid = models.CharField(primary\_key=**True**, max\_length=3)  dscra = models.CharField(max\_length=50, blank=**True**, null=**True**)  cpxna = models.CharField(unique=**True**, max\_length=20)  pcpxi = models.ForeignKey(**'self'**, models.DO\_NOTHING, db\_column=**'pcpxi'**, blank=**True**, null=**True**)  dscrc = models.TextField(blank=**True**, null=**True**)  dirna = models.CharField(max\_length=70, blank=**True**, null=**True**)  rlvnc = models.CharField(max\_length=3, blank=**True**, null=**True**)   **class** Meta:  db\_table = **'cmplx'   class** Tmcat(models.Model):  catid = models.CharField(primary\_key=**True**, max\_length=3)  dscra = models.TextField(blank=**True**, null=**True**)  foldr = models.CharField(max\_length=35, blank=**True**, null=**True**)  rlvnc = models.CharField(max\_length=3, blank=**True**, null=**True**)   **class** Meta:  db\_table = **'tmcat'  class** Project(models.Model):  project\_id = models.IntegerField(verbose\_name=**'ID проекта'**, unique=**True**)  name = models.CharField(max\_length=255, verbose\_name=**'Имя проекта'**)  created\_at = models.DateTimeField(auto\_now\_add=**True**)  updated\_at = models.DateTimeField(auto\_now=**True**)  push\_commit\_to = models.CharField(max\_length=255, blank=**True**, null=**True**)  valid = models.BooleanField(default=**False**)   **class** Meta:  verbose\_name = **'Проект'** verbose\_name\_plural = **'Проекты'  class** Branch(models.Model):  branch\_id = models.CharField(max\_length=255, verbose\_name=**'Идентификатор ветки'**)  name = models.CharField(max\_length=255, verbose\_name=**'Имя'**)  project = models.ForeignKey(Project, on\_delete=models.CASCADE)  last\_commit = models.CharField(max\_length=255, blank=**True**, null=**True**, verbose\_name=**'Последний коммит'**)   **class** Meta:  verbose\_name = **'Ветка'** verbose\_name\_plural = **'Ветки'** |

Модели *Solun, Tmpls, Cmplx, Tmcat* требуются для сопоставления идентификаторов комплекса, решения и типа документа к их названию. Модели *Project* и *Branch* хранят в себе информацию о репозиториях и связанных с ними вектами.

Основой цикл приложения представлен двумя классами: *GitLabExtractor* и *ReportCreator*:

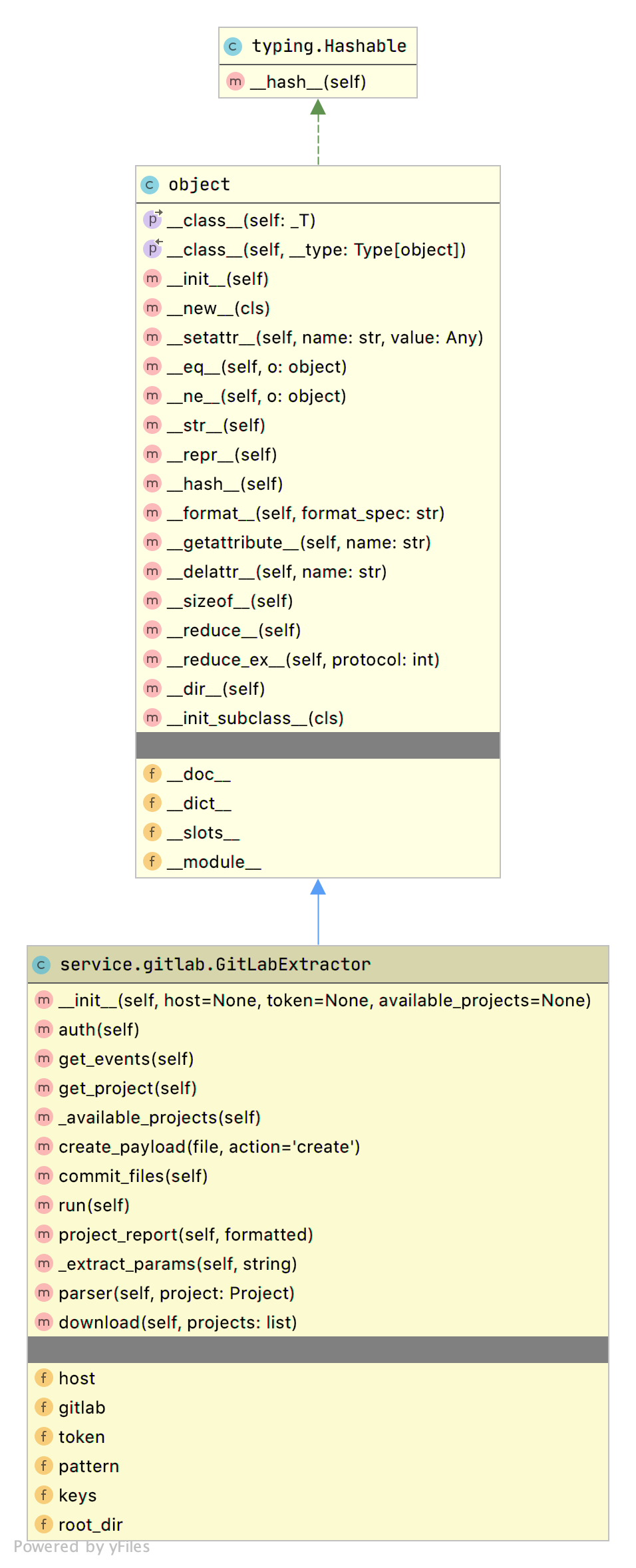


Рисунок 9 Диаграмма класса GitlabExtractor

В классе *GitLabExtractor* реализованы методы получения отчетов. При инициализации класса происходи авторизация на сервере gitlab по token, указанному в файлах настроек. В классе метод run является ключевым, который содержит в себе все требуемые вызовы. При вызове данного метода происходит получение всех доступных для чтения проектов. Каждый проект обрабатывается методом *parser*, который рекурсивно извлекает каждый документ. Возвращается список словарей, где каждый словарь содержит атрибуты документа: имя файла, уникальный идентификатор, полный путь, права доступа и тип документа. Пример представлен в листинге 3.

Листинг 3. Фрагмент результата рекурсивного извлечения файлов

|  |
| --- |
| [{'id': 'f57b055d3e2b249a388be635787337eeba9a0ffb',  'mode': '100644',  'name': 'full.tex',  'path': 'content/full.tex',  'type': 'blob'},  {'id': '829e6fe6eefeee7776f87554b3a5bf65be21bbd4',  'mode': '100644',  'name': 'rndcmp.pdf',  'path': 'content/rndcmp/rndcmp.pdf',  'type': 'blob'},  {'id': '6d9568f532f0d846ac3e7bca3277d59408820028',  'mode': '100644',  'name': 'rndcmp.tex',  'path': 'content/rndcmp/rndcmp.tex',  'type': 'blob'},  {'id': 'e45b31d5882bdf2ac62791c889dcd602421efd18',  'mode': '100644',  'name': 'rndcmp\_rem\_2020\_06\_05\_n01.tex',  'path': 'content/rndcmp/rndcmp\_rem\_2020\_06\_05\_n01.tex',  'type': 'blob'},  {'id': 'db21d6e1c17ce0d376e9e667b2c4bd47d28c65e1',  'mode': '100644',  'name': 'rndhpc.pdf',  'path': 'content/rndhpc/rndhpc.pdf',  'type': 'blob'},  {'id': '0651e841d8b01c17888ba10efb81ec7857d66fbc',  'mode': '100644',  'name': 'rndhpc.tex',  'path': 'content/rndhpc/rndhpc.tex',  'type': 'blob'},  {'id': '876264f4321725bc07764949b474dc215503b834',  'mode': '100644',  'name': 'rndhpc\_rem\_2020\_06\_07\_n01.tex',  'path': 'content/rndhpc/rndhpc\_rem\_2020\_06\_07\_n01.tex',  'type': 'blob'}] |

Каждый файл сравнивает с regex-выражением, позволяющим определить, соответствует ли имя файла заданным требованиям. Требования к именам файлов описаны в разделе «Требования, предъявляемые к отчетной документации». Из имени файла, подходящего regex-выражению, извлекаются атрибуты документа: комплекс, решение, тип документа к которым относится файл, а также дата и имя документа. Новые атрибуты добавляются к существующим. На основе списка идентификаторов файлов метод download загружает их.

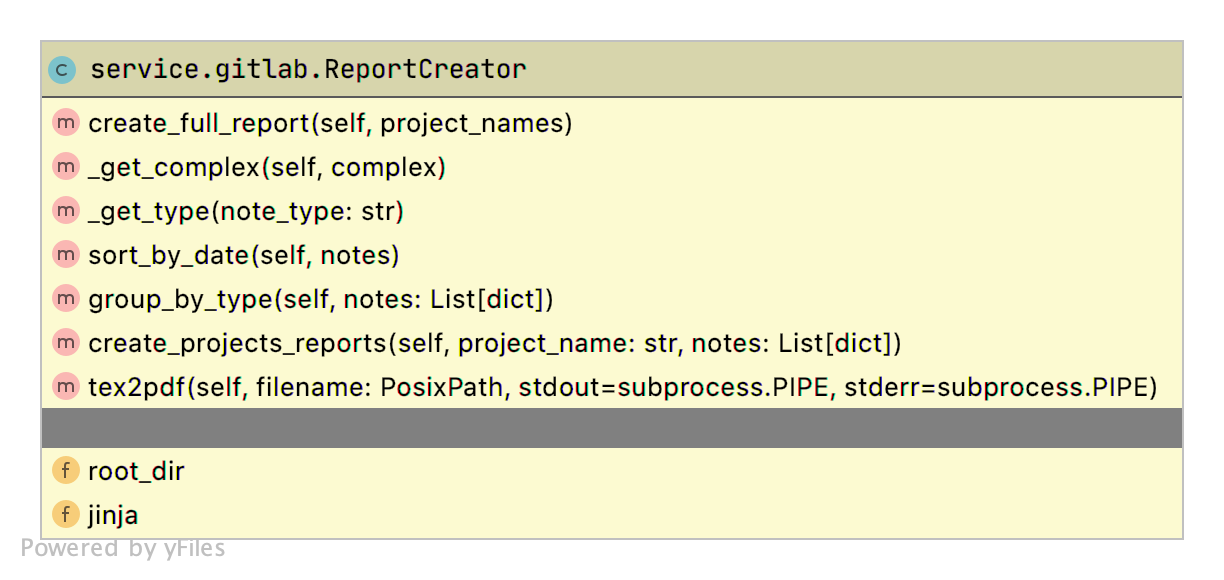


Рисунок 10 Диаграмма класса ReportCreator

Далее управление предается классу объекту класса ReportCreator. В классе реализованы методы создания конечных отчетов. Методы *\_get\_complex* и *\_get\_type* используя Django модели Tmpls и Solun делает запрос в базу данных для определения соответствия имени комплекса, решения и типа документа для каждого документа. Метод *group\_by\_type*группирует все фалы по комплексу, решению и типу файла. Метод *sort\_by\_date* сортирует файлы по дате из названия файла.

Полученная обработанная и сгруппированная информация о файлах обрабатывается поступает в качестве аргумента модулю Jinja. Шаблон содержит статические части вывода TeX, а также некоторый специальный синтаксис, описывающий, как будет вставляться динамический контент. Предварительно заготовленный шаблон представлен в листинге 4. Динамический контент обозначен ключевой командой **\VAR{}***.*

Листинг 4. Шаблон отчета

|  |
| --- |
| \documentclass{report} \usepackage[T2A]{fontenc} \usepackage[utf8]{inputenc} \input{glyphtounicode} \pdfgentounicode=1 \usepackage[russian]{babel} \usepackage{hyphenat} \usepackage{titlesec} \titleformat{\chapter}[display]  {\normalfont\huge\bfseries}{}{0pt}{\Huge} \titlespacing\*{\chapter}  {0pt}{10pt}{40pt} \makeatother \renewcommand\thesection{\arabic{section}} \title{Отчет об исследованиях \VAR{title.upper()}} \begin{document} \maketitle \tableofcontents %% for complex, groups in notes.items()  \chapter{\VAR{complex}}  %% for type\_, group in groups.items()  \section{\VAR{type\_}}  %% for note in group  \input{\VAR{note['name']}}  %% endfor  %% endfor %% endfor \end{document} |

Сгенерированное модулем jinja2 результат сохраняется в файл, а его расположение предается методу text2pdf. Метод вызывает утилиту *latexmk*, которая компилирует tex файл в pdf для каждого комплекса.

После создания отчетов формируется полный отчет, содержащий в себе отчеты по каждому комплексу. Методу *create\_full\_report* передается список расположений отчетов, который формирует итоговый полный отчет. Для этого был разработан еще один файл шаблона jinja, представленный в листинге 4.

Листинг 5. Шаблон итогового отчета

|  |
| --- |
| \\documentclass{report} \usepackage[T2A]{fontenc} \usepackage[utf8]{inputenc} \input{glyphtounicode} \pdfgentounicode=1 \usepackage[russian]{babel} \usepackage{hyphenat} \usepackage{titlesec} \usepackage{standalone} \usepackage{import} \renewcommand\thesection{\arabic{section}} \title{Сводный отчет} \titleformat{\chapter}[display]  {\normalfont\huge\bfseries}{}{0pt}{\Huge} \titlespacing\*{\chapter}  {0pt}{10pt}{40pt}  \begin{document} \maketitle \tableofcontents %% for note in notes \subimport{\VAR{note}/}{\VAR{note}.tex} %% endfor \end{document} |

После формирования отчетов управление вновь предаётся объекту класса GitLabExtractor. Метод *commit\_files* отправляет запрос на рекурсивное извлечение всех файлов. Такой же список формируется на основе локальных файлов. Затем вычисляется разница двух списков, на основе которого можно принять решение по методу изменения файла: создание, изменение или удаление.

Для каждого файла формируется определённое сообщение. Оно состоит из: содержим файла, методом изменения и именем коммита. Содержимое файла кодируется в формат base64. Каждое сообщение отправляется на сервер gitlab фиксации изменений фалов.

Для того чтобы запускать скрипт создания отчетов требуется механизм отслеживания изменений в ветках репозитория, а также предотвратить лишние вызовы, т.е. создавать отчет, когда нужная ветка в репозитории действительно изменилась. При всяком изменении в репозитории каждому событию присваивается уникальный идентификатор. Данный идентификатор является хэшем SHA-1, который вычисляется на основе информации о изменениях, дате изменениях, авторе и журнале сообщений. В приложении этот идентификатор сохраняется в поле commit\_id модели Branch. Во время изменений в репозитории это позволяет определять, в конкретно каких ветках произошли изменения путем сравнения текущего идентификатора и идентификатора, хранящегося в базе данных для этой ветки. Это значительно ускоряет работу, т.к. нет необходимости повторно просматривать все ветки в репозитории.

Механизм сохранения идентификаторов реализован в модуле create\_report.py. В модуле описан класс Worker, который сравнивает изменения и отправляет в очередь задачу на создание создания отчета. Очередь задач реализована с помощью модуля celery.

Celery — асинхронная очередь задач, основанная на распределенной передаче сообщений [36]. Она ориентирована на работу в реальном времени, но также поддерживает планирование задач. Единицы исполнения, называемые задачами, выполняются одновременно на одном или нескольких серверах. Задачи могут выполняться асинхронно (в фоновом режиме) или синхронно (в ожидании готовности).

В качестве брокера сообщений используется RabbitMQ — программный брокер сообщений на основе стандарта AMQP [37].

Для мониторинга и администрирования задач Celery используется веб-инструмент Flower [38].

При запуске метода apply\_async встроенный в celery объект Producer (поставщик) отправляет в очередь сообщение идентификатор и название задачи, которую нужно выполнить. Producer (потребитель) поочередно прочитывает сообщения из очереди и передает управление celery-worker’у, который прямо запускает нужную функцию. Панель мониторинга Flower позволяет отслеживать текущий статус задачи, просматривать сообщения об ошибках, в ручном режиме останавливать задачи и просматривать статистку. Механизм асинхронной очереди задач позволяет продолжать работу всех функций приложения даже при неудачном создании отчета.

Класса Worker также включает в себя дандер-методы *\_\_enter\_\_* *и \_\_end\_\_*, что позволяет использовать его как контекстный менеджер. Методы start и stop позволяют запустить или остановить Worker соответственно.

В качестве аргумента классу передается период в секундах, с которым будет производится проверка обновлений в репозиториях.

Общая схема работы приложения представлена на рисунке 11.

### Описание требований, предъявляемые к отчетной документации

Документы должны располагаться в одном из репозиториев GitLab сервера sa2systems.ru, входящих в список доступных в панели администратора.

Требований к расположению файла внутри репозитория нет, т.к. анализируются рекурсивно все файлы.

Пример файла, соответствующего требуемым правилам:

**rndhpc\_rem\_2020\_04\_09\_filename.tex,** гдеrnd – идентификатор комплекса, hpc – идентификатор решения, 2020 – год, 04 – месяц, 09 – день, filename – имя файла (может быть произвольным), .tex – формат файла. Допускается формат .tex и .pdf

Документы, не советующие данным правилам, а также содержащие неопределенные идентификаторы также будут проигнорированы.

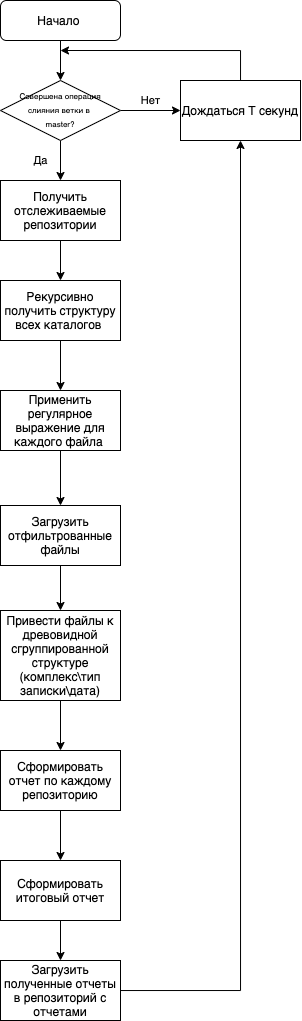


Рисунок 11. Блок-схема программы

### Конфигурация группы docker-контейнеров

Каждый из этих приложений размещён как отдельный docker-контейнер.

Для определения и запуска многоконтейнерных приложений используется инструмент compose [39]. Для работы с инструментом требуется выполнить предварительные шаги:

1. Определить базовое приложение с помощью *Dockerfile* файла.
2. Определить сервисы, из которых состоит приложение, чтобы их можно было запускать вместе в изолированной среде. Требуется написать специальный YAML-файл под названием *docker-compose.yml*, в котором указана конфигурация для каждого сервиса.
3. Выполнить команду **docker-compose up**, которая запустит все приложения, указанные в *docker-compose.yml* файле.

В рамках текущей работы был определен следующий *docker-compose.yml* файл (листинг 6):

Листинг 6. docker-compose.yml файл для одновременного запуска нескольких контейнеров

|  |
| --- |
| **version**: **'3.1' services**:  **web**:  **restart**: always  **build**:  **context**: .  **env\_file**:  - var.env  **volumes**:  - .:/opt/project  **command**: [**"web"**]  **ports**:  - **"9999:80"  mq**:  **hostname**: mq  **image**: rabbitmq:3-management  **ports**:  - **"5672:5672"** - **"15672:15672"  tasks**:  **restart**: always  **build**:  **context**: .  **links**:  - mq  **env\_file**:  - var.env  **command**: [**'tasks'**]  **flower**:  **restart**: always  **build**:  **context**: .  **links**:  - mq  **env\_file**:  - var.env  **volumes**:  - .:/opt/project  **ports**:  - **"5555:5555"  command**: [**"flower"**]  **worker**:  **restart**: always  **build**:  **context**: .  **links**:  - mq  - flower  **env\_file**:  - var.env  **command**: [**'create\_report'**] |

Структурная схема взаимодействия контейнеров представлена на рисунке 12:

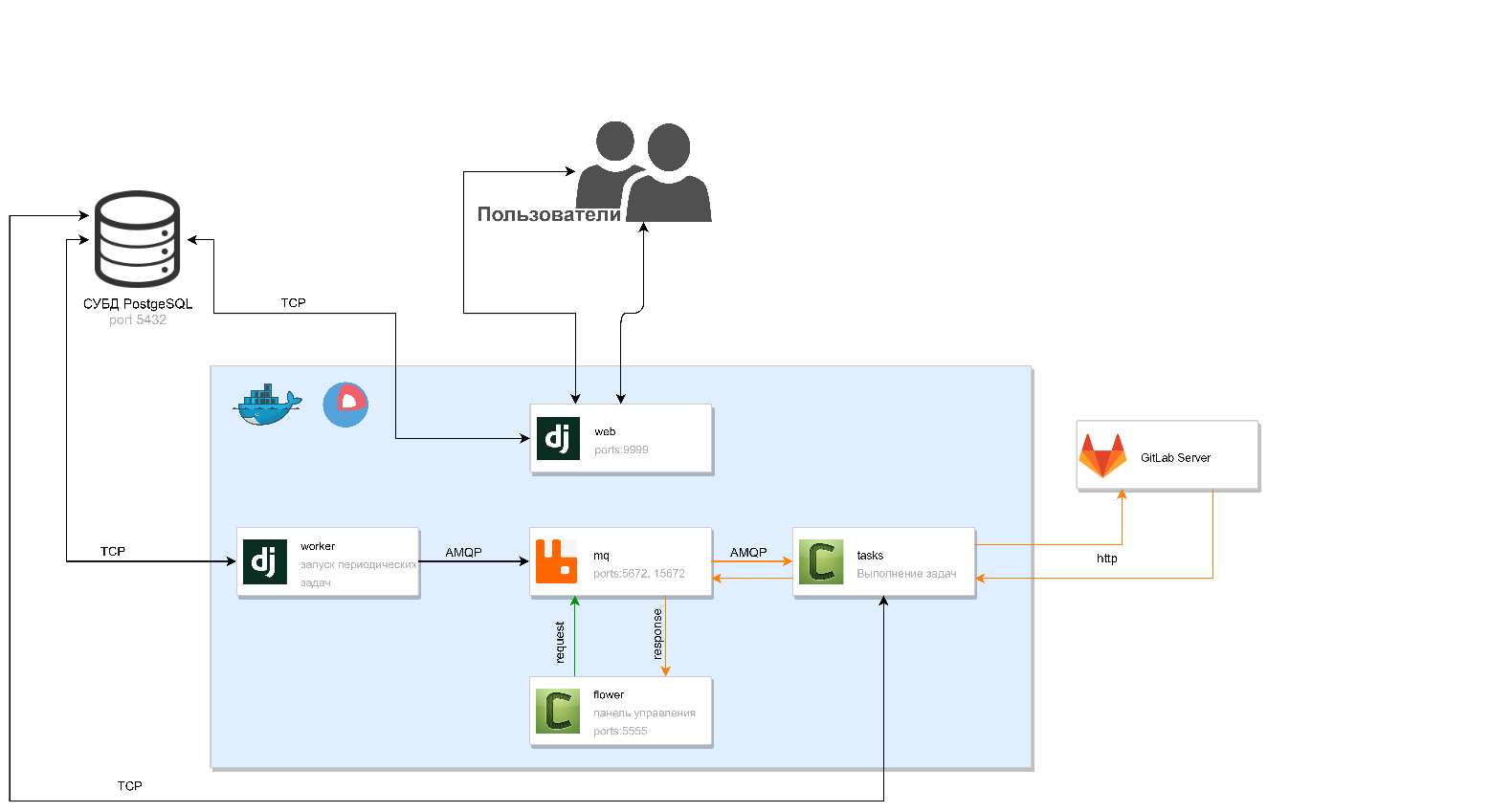


Рисунок 12 Структурная схема микросервисов и протоколы взаимодействия

Во время выполнения инструкции **command: [‘имя команды’],** инструмент compose извлекает из файла *docker-enrypoint.sh* соответствующие указанному имени команды и выполняет их в каждом контейнере соответственно. Написанный *docker-enrypoint.sh* файл представлен в листинге 3.

Листинг 7. docker-enrypoint.sh с именами и советующими им командами

|  |
| --- |
| **#!/bin/bash** set -e **case "**$1**" in** web)  python3 manage.py collectstatic --noinput  python3 manage.py runserver 0.0.0.0:80  **;;** tasks)  exec celery worker -A reporter -l info --concurrency=1 -n reporter-creator\_worker@%n  **;;** flower)  exec celery -A reporter flower --db=/flower/flower -l info  **;;** create\_report)  exec python manage.py create\_report 60  **;;** \*)  exec **"**$@**"  ;; esac** |

### Внедрение приложения в цикл непрерывной интеграции

Для успешного автоматического тестирования и развертывания приложения и его компонент на удаленном сервере требуется инструмент, реализующий непрерывные методологии: непрерывная интеграция (CI); непрерывная доставка (CD); непрерывное развертывание (CD).

Непрерывная интеграция работает путём добавления небольших фрагментов кода в кодовую базу проекта, размещённую в git-репозитории, и запуске конвейера скриптов для сборки, тестирования и проверки изменений кода перед их слиянием с основной веткой.

Непрерывная доставка и развёртывание состоят из следующего шага, разворачивая ваше приложение в промышленную эксплуатацию при каждом изменении.

Эти методологии позволяют отлавливать ошибки на ранних стадиях цикла разработки, гарантируя, что весь код, развернутый в производство, соответствует тем стандартам, которые были ранее установлены.

Требованиям для выпускной квалификационной работы является размещение отчетов в репозитории GitLab, который уже включает встроенный инструмент для разработки программного обеспечения с помощью непрерывных методологий – Gitlab CI [40].

Для автоматического тестирования и развертывания приложения был написан специальный файл-инструкция на языке YAML, содержащий выполняемые команды на каждом этапе цикла непрерывной интеграции.

Листинг 8. docker-enrypoint.sh с именами и советующими им командами

|  |
| --- |
| **image**: docker:latest  **stages**:  - up  **services**:  - docker:dind  **before\_script**:  - docker version  - docker-compose version  **deploy**:  **variables**:  **CI\_DEBUG\_TRACE**: **"true"  only**:  - master  **stage**: up  **script**:  - docker-compose down -v  - docker-compose up --build -d |

Указанный файл используется специальной утилитой GitLab Runner [41], которая используется для запуска заданий и отправки результатов в GitLab. В данном случает GitLab Runner во время изменений ветки master выполнит задание deploy. По команде **docker-compose down -v** уничтожатся все работающие контейнеры, далее по команде **docker-compose up --build -d** на основе *docker-compose.yml* файла конфигурации контейнеры переопределяться и запустятся в фоновом режиме.

### 3. ТЕСТИРОВАНИЕ И ОТЛАДКА

Для тестирования применялся метод модульного тестирования, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

За модульное тестирование отвечает модуль pytest [42]. Для тестирования приложения был добавлен дополнительный контейнер c СУБД PostgeSQL. C основной базы данных сделан дамп таблиц solun, tmpls, cmplx и tmcat. На основе дампа были сформированы фикстуры – тестовые данные для тестирования. Ниже представлен пример тестирования функционала группировки списка отчетов (Листинг 9).

Листинг 9. Пример модульного теста

|  |
| --- |
| def test\_group\_by\_type(self):  notes = [  {'id': 'f0b4b21c607290d52c35af3ca6b36270a4d92fc9', 'name': 'rndhpc\_rem\_2020\_04\_09\_n01.tex', 'type': 'rem',  'path': 'rnddoc/rndhpc\_rem\_2020\_04\_09\_n01.tex', 'mode': '100644', 'complex': 'rndhpc', 'year': '20',  'month': '04', 'day': '09', 'title': 'n01'},  {'id': '78e41117f4663ac886c01c4790ccf5524e194fc9', 'name': 'rndhpc\_rem\_2025\_12\_10\_214124.tex',  'type': 'rem', 'path': 'rnddoc/rndhpc\_rem\_2025\_12\_10\_214124.tex', 'mode': '100644', 'complex': 'rndhpc',  'year': '25', 'month': '12', 'day': '10', 'title': '214124'}]  grouped\_notes = self.worker.group\_by\_type(notes)  assert grouped\_notes == {  'Разработка ресурсоемкого ПО инж.анализа. ': {'Заметка общего назначения': [{'complex': 'rndhpc',  'day': '10',  'id': '78e41117f4663ac886c01c4790ccf5524e194fc9',  'mode': '100644',  'month': '12',  'name': 'rndhpc\_rem\_2025\_12\_10\_214124.tex',  'path': 'rnddoc/rndhpc\_rem\_2025\_12\_10\_214124.tex',  'title': '214124',  'type': 'rem',  'year': '25'},  {'complex': 'rndhpc',  'day': '09',  'id': 'f0b4b21c607290d52c35af3ca6b36270a4d92fc9',  'mode': '100644',  'month': '04',  'name': 'rndhpc\_rem\_2020\_04\_09\_n01.tex',  'path': 'rnddoc/rndhpc\_rem\_2020\_04\_09\_n01.tex',  'title': 'n01', 'type': 'rem', 'year': '20'}]}} |

Также применялся метод ручного тестирования. Предварительно в несколько репозиториев gitlab были размещены файлы, соответствующие правилам именования. В течении минуты в панели управления Flower отобразилась успешно выполненная задача (рисунок 13).

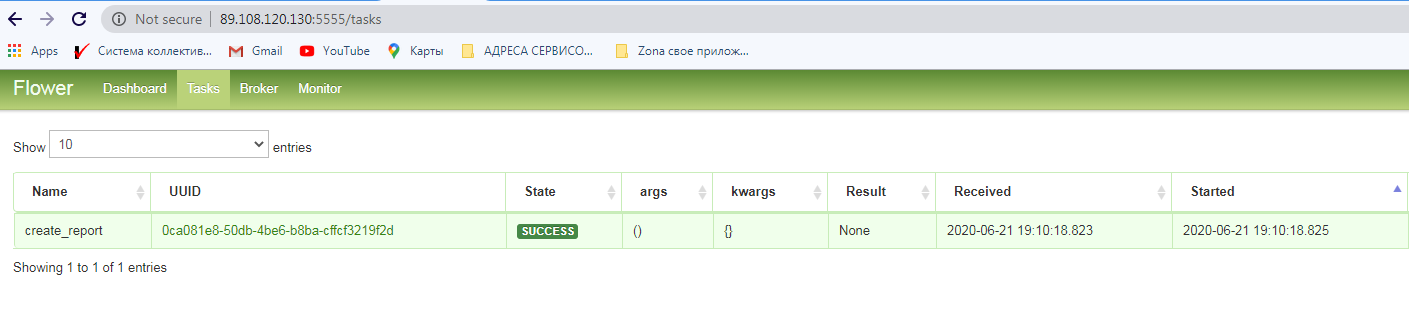


Рисунок 13 Состояние панели управления задач Flower во время тестирования

Также в репозитории rnddoc появились файлы отчетов (рисунки 14 и 15).

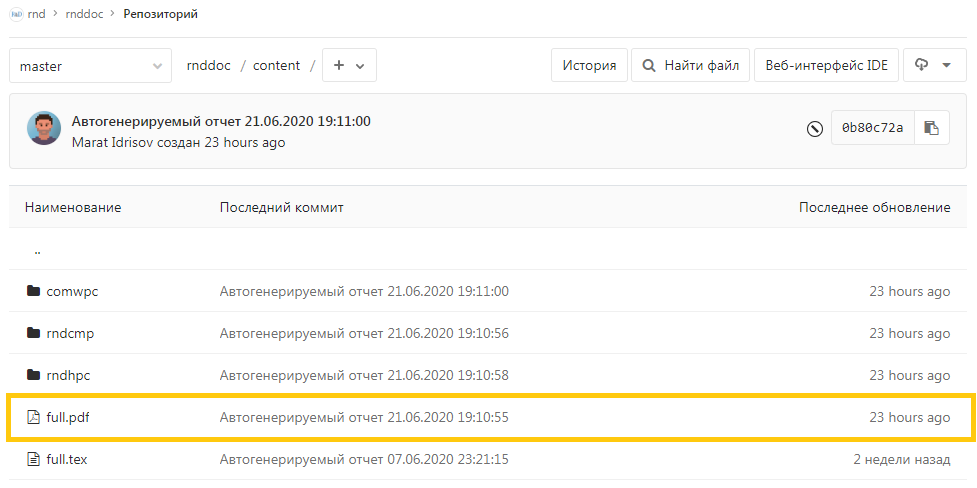


Рисунок 14 Демонстрация появления автоматически созданного отчета

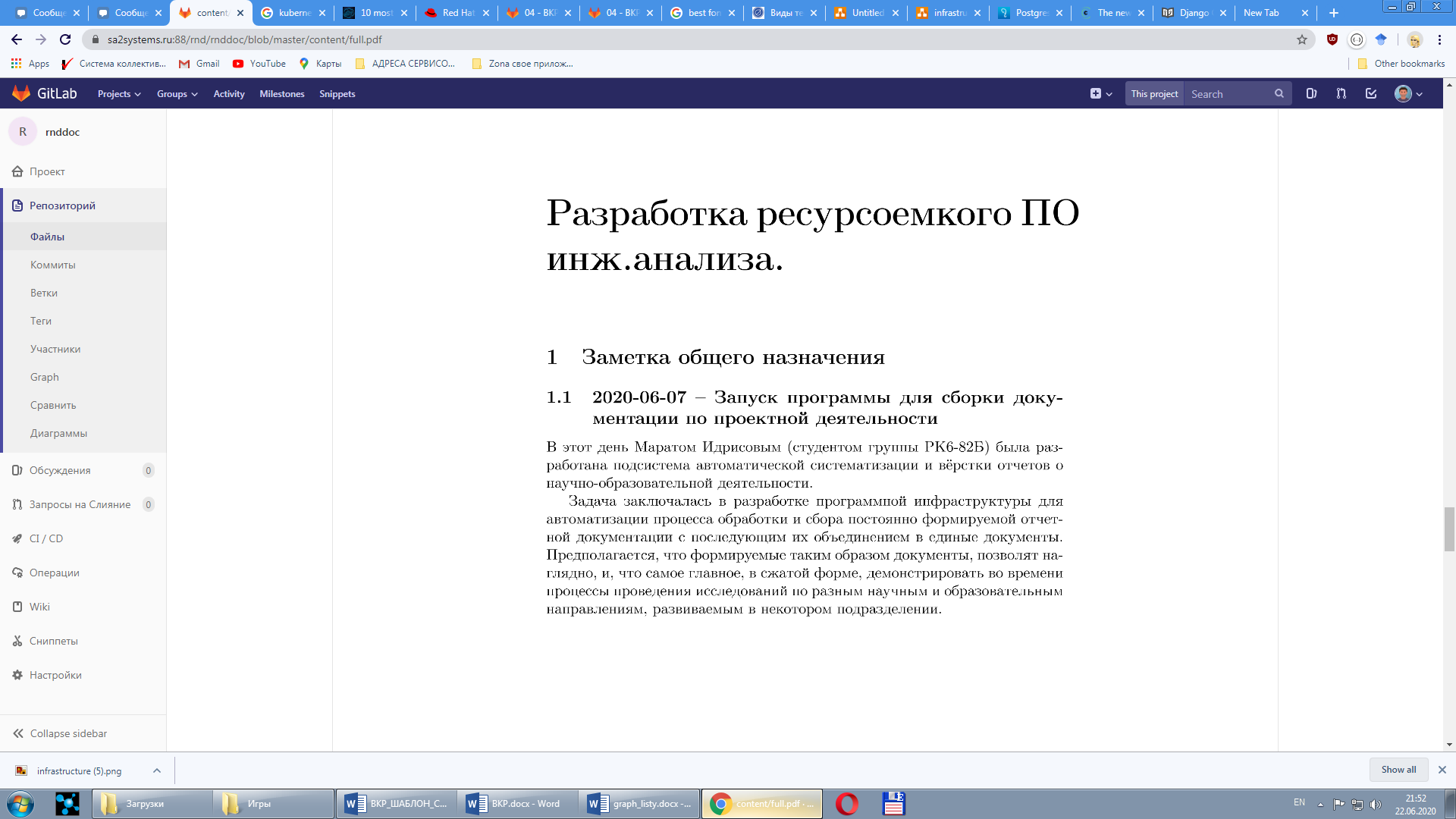


Рисунок 15 Фрагмент итогового отчета

### АНАЛИЗ РЕЗУЛЬТАТОВ

В результате проделанной работы удалось получить программную реализацию web-приложения для автоматизации процесса обработки и сбора постоянно формируемой отчетной документации с последующим их объединением в единые документы. Формируемые таким образом документы, позволят наглядно, и, что самое главное, в сжатой форме, демонстрировать во времени процессы проведения исследований по разным научным направлениям, развиваемым в некотором подразделении. Среди положительных сторон работы стоить отметить внедрение микросервисной архитектуры, которая позволила распределить нагрузку между контейнерами, а также поделить приложение на отдельные сервисы, выполняющие определенный набор функций. Также это делает возможными непрерывную доставку и развертывание приложений. Сервисы получаются небольшими и простыми в обслуживании, развертываются и масштабируются независимо друг от друга. Другой положительной стороной работы является внедрение асинхронной очереди задач. Этот механизм позволяет отслеживать ошибки и продолжать работу всех функций приложения даже при неудачном создании отчета. Недостатком работы можно выделить отсутствие возможности персонализировать итоговые отчеты для каждого конкретного пользователя.

### ЗАКЛЮЧЕНИЕ

В результате исследования по теме было разработана подсистема, автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям. Для решения задач были проанализированы программные решения для создания отчетов. Проведен сравнительный анализ двух типов архитектур, были выделены плюсы и минусы каждой из них. В результате анализа была разработана микросервисная архитектура, позволяющая разделить приложение на сервисы, каждое из которых выполняет определенный набор функций.

Были предложены и разработаны методы создания отчетов, а также отслеживания их изменений.

Разработанная подсистема была протестирована с помощью модульных тестов и в ручном режиме. Все написанные тест-кейсы прошли успешно.

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Top 10 BI Tools - The Best BI Software Review List for 2020 [Электронный ресурс] /. Электрон. текстовые дан. — Режим доступа: <https://www.datapine.com/articles/best-bi-tools-software-review-list>
  2. Tableau Desktop [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://www.tableau.com/products/desktop>
  3. Что такое Power BI? [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://powerbi.microsoft.com/ru-ru/what-is-power-bi/>
  4. Business intelligence (BI) and CPM software [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://www.board.com/en>
  5. SAS Visual Analytics [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://www.sas.com/ru_ru/software/visual-analytics.html>
  6. Oracle Analytics Cloud [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://www.oracle.com/business-analytics/analytics-cloud.html>
  7. Облачное хранилище данных [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%87%D0%BD%D0%BE%D0%B5_%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85>
  8. Amazon Annual report [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <http://www.annualreports.com/HostedData/AnnualReportArchive/a/NASDAQ_AMZN_2018.pdf>
  9. Лучшая архитектура для MVP: монолит, SOA, микросервисы или бессерверная [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://habr.com/ru/company/otus/blog/476024/>
  10. Артамонов Юрий Сергеевич, Востокин Сергей Владимирович Разработка распределенных приложений сбора и анализа данных на базе микросервисной архитектуры // Известия Самарского научного центра РАН. 2016. №4-4.
  11. Таненбаум Э. и др. Распределенные системы. Принципы и парадигмы. – Питер, 2003.
  12. Осипов Д. Б. Проектирование программного обеспечения с помощью микросервисной архитектуры //Вестник науки и образования. – 2018. – Т. 2. – №. 5 (41).
  13. Richardson C. Microservices Patterns: With Examples in Java. – Manning Publications, 2019.
  14. Abbott M. L., Fisher M. T. The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise. – Pearson Education, 2009.
  15. Decomposing Applications for Scalability and Deployability [Электронный ресурс] / Chris Richardson. — Электрон. текстовые дан. — 2012. — Режим доступа: [www.slideshare.net/chris.e.richardson/decomposing-applications-for- scalability-and-deployability-april-2012](http://www.slideshare.net/chris.e.richardson/decomposing-applications-for-%20scalability-and-deployability-april-2012)
  16. Микросервисная архитектура [Электронный ресурс] / Chris Richardson. — Электрон. текстовые дан. — 2012. — Режим доступа: [ru.wikipedia.org/wiki/Микросервисная\_архитектура](http://ru.wikipedia.org/wiki/Микросервисная_архитектура)
  17. Implementing microservice architectures [Электронный ресурс] / Fred George. — Электрон. текстовые дан. — 2013. — Режим доступа: <https://archive.oredev.org/oredev2013/2013/wed-fri-conference/implementing-micro-service-architectures.html>
  18. What are microservices? [Электронный ресурс] / Chris Richardson. — Электрон. текстовые дан. — Режим доступа: <https://microservices.io/>
  19. Microservices — a definition of this new architectural term [Электронный ресурс] / James Lewis, Martin Fowler. — Электрон. текстовые дан. — 2014. — Режим доступа: <https://martinfowler.com/articles/microservices.html>
  20. Ричардсон, К. Микросервисы. Паттерны разработки и рефакторинга: книга / К. Ричардсон. – СПб: Питер, 2019. – 544 с.
  21. Осипенко, А.А. Переход от монолита к микросервисам [Электронный ресурс] / А.А. Осипенко. — Электрон. текстовые дан. — 2016. — Режим доступа: <https://habr.com/ru/post/305826/>
  22. Microservices [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://en.wikipedia.org/wiki/Microservices>
  23. Basics of the Unix Philosophy [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа:  
       <http://www.catb.org/esr/writings/taoup/html/ch01s06.html>
  24. Docker overview [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://docs.docker.com/get-started/overview/>
  25. Continuous Integration [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <http://wiki.c2.com/?ContinuousIntegration>
  26. Ubuntu 18.04.4 LTS (Bionic Beaver) [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://releases.ubuntu.com/18.04.4/>
  27. Ubuntu 20.04 LTS (Focal Fossa) [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://releases.ubuntu.com/20.04/>
  28. CentOS-8 (1911) Release Notes [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS8.1911?action=show&redirect=Manuals%2FReleaseNotes%2FCentOSLinux8>
  29. Python - Docker Hub [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://hub.docker.com/_/python>
  30. Выпущен Debian 10 "buster" [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://www.debian.org/News/2019/20190706.ru.html>
  31. python:3.8-slim-buster [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: [https://hub.docker.com/layers/python/library/python/3.8-slim-buster](https://hub.docker.com/layers/python/library/python/3.8-slim-buster/)
  32. Docker Hub is the world's easiest way to create, manage, and deliver your teams' container applications [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://hub.docker.com/>
  33. Jinja — Jinja Documentation (2.11.x) [Электронный ресурс] /. — Электрон. журн. — Режим доступа: <https://jinja.palletsprojects.com/en/2.11.x/>
  34. Automate GitLab via a simple and powerful API [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://docs.gitlab.com/ee/api/README.html>
  35. Python wrapper for the GitLab API [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://github.com/python-gitlab/python-gitlab>
  36. Celery - Distributed Task Queue [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://docs.celeryproject.org/en/stable/index.html>
  37. RabbitMQ is the most widely deployed open source message broker [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://www.rabbitmq.com/documentation.html>
  38. Flower - Celery monitoring tool [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://flower.readthedocs.io/en/latest/>
  39. Overview of Docker Compose [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://docs.docker.com/compose/>
  40. GitLab CI/CD [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: [https://docs.gitlab.com/ee/ci](https://docs.gitlab.com/ee/ci%20)
  41. GitLab Runner Docs [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://docs.gitlab.com/runner/>
  42. Full pytest documentation [Электронный ресурс] /. — Электрон. текстовые дан. — Режим доступа: <https://docs.pytest.org/en/latest/contents.html>

### ПРИЛОЖЕНИЕ А

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Диаграмма класса GitlabExtractor* | | | | | | | | |
|  |  |  |  |  | *Выпускная квалификационная работа бакалавра* | | | |
|  |  |  |  |  |
| *Изм* | *Лист* | *№ докум.* | *Подпись* | *Дата* |
| *Разработ.* | | *Идрисов М.Т.* |  |  | *Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям* | *Литер.* | *Лист* | *Листов* |
| *Проверил* | | *Соколов А.П.* |  |  | *У* | *1* | 5 |
| *Зав. каф.* | |  |  |  | *МГТУ им. Н. Э. Баумана ф-т РК гр. РК6-82* | | |
| *Н. контр.* | |  |  |  |
| *Утвердил* | |  |  |  |

|  |  |
| --- | --- |
| *Диаграмма класса ReportCreator* | |
|  | *Лист* |
| *2* |

|  |  |
| --- | --- |
| *Блок-схема основного цикла программы* | |
|  | *Лист* | |
| *3* | |

|  |  |
| --- | --- |
| *Структурная схема микросервисов и протоколы взаимодействия*  H:\Загрузки\infrastructure (5).png | |
|  | *Лист* | |
| *4* | |

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**АКТ**

**проверки выпускной квалификационной работы**

Студент группы *РК6-82Б*

|  |
| --- |
| *Идрисов Марат Тимурович* |
| *(Фамилия, имя, отчество)* |

Тема выпускной квалификационной работы: *«Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям»*

Выпускная квалификационная работа проверена, размещена в ЭБС «Банк ВКР» в полном объеме и соответствует / не соответствует требованиям, изложенным в Положении о порядке

*ненужное зачеркнуть*

подготовки и защиты ВКР.

Объем заимствования составляет 6.4 % текста, что с учетом корректного заимствования соответствует / не соответствует требованиям к ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*ненужное зачеркнуть бакалавра, специалиста, магистра*

|  |  |  |  |
| --- | --- | --- | --- |
| **Нормоконтролёр** |  |  | С.В. Грошев |
|  |  | (подпись) | (ФИО) |
| Согласен: |  |  |  |
| **Студент** |  |  | М.Т. Идрисов |
|  |  | (подпись) | (ФИО) |

Дата: