**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_____

# ОТЧЕТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ

Студент_____Идрисов Марат Тимурович_____
*фамилия, имя, отчество*

Группа_____РК6-82Б_____

Тип практики __Преддипломная_____

Название предприятия_____МГТУ им. Н.Э. Баумана НУК РК_____

Студент _____28.05.2020_____Идрисов М.Т._____
*подпись, дата* *фамилия, и.о.*

Руководитель практики _____Соколов А.П._____
*подпись, дата* *фамилия, и.о.*

Оценка _____

*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)

Кафедра «Системы автоматизированного проектирования» (РК6)

# З А Д А Н И Е
## на прохождение преддипломной практики

на предприятии МГТУ им. Н.Э. Баумана НУК РК

Студент Идрисов Марат Тимурович, РК6-82Б

(фамилия, имя, отчество; инициалы; индекс группы)

Во время прохождения проектно-технологической учебной практики студент должен:

1. Провести обзор литературы по теме: "Разработка подсистемы автоматической вёрстки отчетной документации о ходе научно-образовательной деятельности по различным направлениям" (не менее 15 источников);
2. Оформить отчет о результатах прохождения практики.

Дата выдачи задания «19» мая 2020 г.

Руководитель практики от кафедры _____ / Соколов А.П.
(подпись, дата)

**Студент** _____ 28.05.2020 / Идрисов М.Т.
(подпись, дата) (Фамилия И.О.)

**ОГЛАВЛЕНИЕ**

# 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 большого 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 эффективной организации научно-образовательной деятельности программной инфраструктуры 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.

Ниже представлен обзор существующих решений по обработке отчетной документации, а также преимущества и недостатки архитектурных решений, которые могут быть применены при разработке такого программного обеспечения.

## 1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

В отчете консалтинговой компании DataPine [1] за 2020 год выделены 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 Cloud и SAS Visual Analytics. Облачные приложения имею ряд преимуществ по сравнению с клиентскими приложениями [7]:

- Возможность доступа к данным из 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.

# 2. ОБЗОР АРХИТЕКТУРНЫХ РЕШЕНИЙ РАЗРАБОТКИ 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

## 2.1. Монолитная 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 —** «путь» готового кода к промышленной среде оказывается длинным и тяжелым. 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.

## 2.2. Микросервисная архитектура

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.

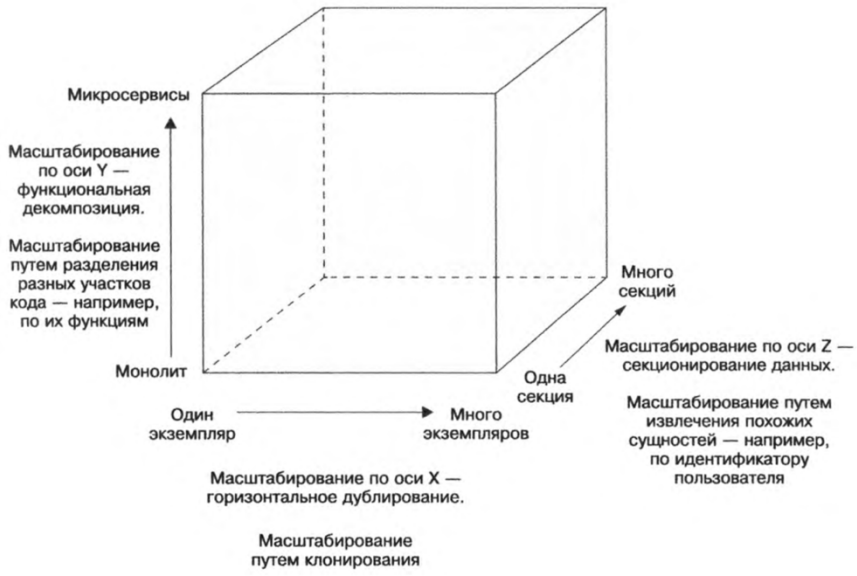


Рисунок 1 Модель определяет три направления для масштабирования приложения: масштабирование по оси X распределяет нагрузку между несколькими идентичными экземплярами, по оси Z — направляет запросы в зависимости от их атрибутов, ось Y разбивает приложение на сервисы с разными функциями

### 2.2.1. Масштабирование по оси X распределяет запросы между несколькими экземплярами

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).

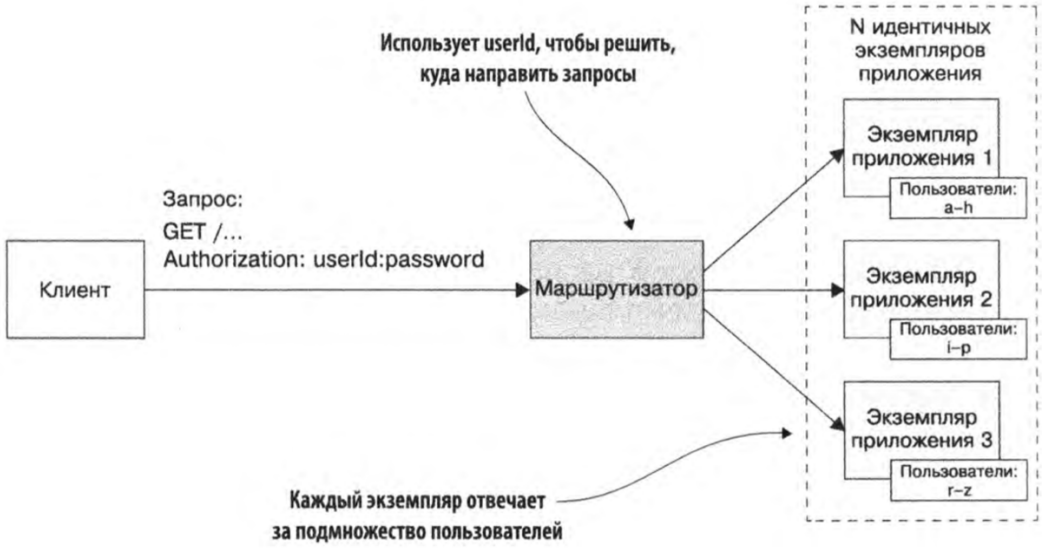


Рисунок 3 Масштабирование по оси Z связано с запуском нескольких идентичных экземпляров монолитного приложения, размещенных за маршрутизатором, который направляет запросы в зависимости от их атрибутов.

### 2.2.2. Масштабирование по оси Y разбивает приложение на сервисы с разными функциями

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:

- **сложно подобрать 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.

Но все эти недостатки нивелируют по сравнению недостатками монолитной архитектуры.

## 3. ЗАКЛЮЧЕНИЕ

1. Были представлены современные и перспективные разработки в области создания отчетности
2. Был проведен обзор научно-технических публикаций по теме исследования
3. Проанализированы существующие архитектурные решения, выделены их достоинства и недостатки.

## СПИСОК ЛИТЕРАТУРЫ

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

16. Микросервисная архитектура [Электронный ресурс] / Chris Richardson. — Электрон. текстовые дан. — 2012. — Режим доступа: 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.