**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

**Факультет Программной инженерии и компьютерной техники**

**Образовательная программа Компьютерные системы и технологии**

**Направление подготовки (специальность) 09.04.01 Информатика и вычислительная техника**

# О Т Ч Е Т

о **научно-исследовательской работе**

Тема задания: Реализация контрактного тестирования микросервисов

Обучающийся: Ореховский Антон, группа Р4119

Руководитель практики от университета: Маркина Т.А, доцент факультета ПИиКТ

Санкт-Петербург
2022

# ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

В последние годы в сфере разработки веб-приложений стал популярен микро-сервисный подход. Он заключается в том, что весь функционал приложения делится на маленькие модули, они же микро-сервисы, каждый из которых реализует минимально возможный функционал. При этом каждый модуль может общаться с другими модулями для выполнения необходимых операций. Это вызывает зависимость между микро-сервисами, которая усложняет процесс разработки и главным образом процесс тестирования.

Упростить процесс тестирования можно, используя так называемые контракты, смысл которых заключается в том, что потребитель услуг веб-сервиса может описать то какие данные и каким образом он хочет получать от поставщика этих услуг. При этом при разработке функционала, поставщик услуг может опираться на уже известные контракты.

Целью данной работы является обеспечение возможности проведения контрактного тестирования микро-сервисов в предоставленной инфраструктуре.

Для достижения данной цели все работы были разделены на 4 части, по 1 части на семестр обучения:

1. Исследование предметной области
2. Создание минимально работающего продукта
3. Реализация решения максимально приближенного к поставленной цели
4. Исправление недочетов, интеграция в среду заказчика

При этом на первый семестр, в рамках работы исследования, были поставлены следующие задачи:

1. Составление технического задания на проект.
2. Согласование технического задания на проект
3. Исследование аналогов в области контрактного тестирования с целью выбора наиболее удачных практик применимых к разрабатываемому РК.
4. Исследование возможностей составления контрактов и документирование применимых к разрабатываемому РК возможностей.
5. Исследование возможностей публикации контрактов потребителями и поставщиками с целью их дальнейшего сравнения и выбора наиболее применимой к разрабатываемому РК.
6. Получение знаний о возможностях развертывания контейнеров/кластеров.

7. Исследование возможных видов запросов от потребителя к поставщику и их документация.
8. Исследование способов сопоставления контрактов поставщика и потребителя и разработка алгоритма, который будет применен в разрабатываемом РК для сопоставления контрактов.
9. Проектирование концепта DSL для описания контрактов.
10. Исследование и документирование возможностей интеграции РК с элементами CI/CD на примере GitLab Pipeline.
11. Исследование возможностей генераций OpenAPI схем.
12. Разработка архитектуры сервиса в виде UML диаграмм.
13. Документирование предполагаемых конечных точек РК.
14. Составление структуры БД и ее представления в виде ER-модели.
15. Согласование набора технологий.
16. Документирование вариантов использования разрабатываемого сервиса.

В данном семестре работа была реализована по следующим этапам:

1. инструктаж обучающегося,
2. эпоха 1,
3. эпоха 2,
4. эпоха 3,
5. эпоха 4,
6. оформление отчетной документации.

# 1 Инструктаж обучающегося

Первостепенную важность имела необходимость прохождения инструктажей. Этот этап достаточно важен, так как объясняет все возможные нестандартные ситуации и способы реагирования на них.

Часть инструктажей я прошел в Университете ИТМО в рамках подготовки к работам по проекту (охрана труда, техника безопасности, пожарная безопасность), часть инструктажей была проведена моим куратором (правила внутреннего трудового распорядка, первичный инструктаж на рабочем месте).

# 2 Эпоха 1

Так как я не являюсь экспертом в области своего проекта, первым делом мне необходимо было понять, что из себя представляет контрактное тестирование, зачем оно нужно и как его проводить. Первые наводки на место поиска были даны моими наставниками из компании GS Labs. По этим наводкам я узнал о том, что такое контрактное тестирование [1] на каких (или вместо каких) этапах разработки его можно применять [2].

Погрузившись в контекст, проблемы можно было приступить к формированию технического задания проекта и формированию задач на семестр. В рамках формирования технического задания были определены требования к проекту:

- требования к интерфейсу пользователя,
- требования к интерфейсам ПО,
- функциональные требования,
- нефункциональные требования,
- требования к логической структуре БД,
- некоторый ряд пользовательских сценариев,
- и пр.

Немалую роль в прогрессе моего проекта сыграли наставники. На еженедельных удаленных встречах мы обсуждали проект, и наставники предлагали его улучшения и указывали на недочеты. Также ощутимая помощь была оказана моим куратором Маркиной Т. А., например в виде формализации технического задания. Усилиями всех причастных, удалось сформировать приемлемое техническое задание на проект и согласовать его.

Следует отметить, что в данной эпохе так же были сформулированы пожелания на проект:

- регистратор контрактов (РК) – разрабатываемый сервис, который предоставит возможность осуществления контрактного тестирования,
- поставщик – сервис, который предоставляет данные или функциональность другим сервисам (вне контекста контрактного тестирования),
- потребитель – сервис, который использует данные или функциональность другого сервиса,
- в рамках HTTP: потребитель генерирует HTTP запрос, поставщик генерирует HTTP ответ,
- разрабатываемый РК должен:
    1. регистрировать контракты поставщиков,
    2. регистрировать контракты потребителей,
    3. поддерживать совместимость между контрактами потребителей и поставщиков,
    4. следить за эволюцией схемы контрактов,
    5. оповещать клиентов об изменениях схемы контракта поставщика,
    6. иметь возможности для мониторинга и настройки.

# 3 ЭПОХА 2

Реальный прогресс по проекту начался в рамках данной эпохи. Именно тогда я разобрал наиболее популярных представителей контрактного тестирования, исследовал возможности составления контрактов, исследовал возможности публикации контрактов, а также попытался получить знания о развертывании контейнеров/кластеров.

Изучив всех представителей контрактного тестирования, я смог выделить несколько самых «влиятельных» в применяемой области:

- Pact [3],
- VCR [4],
- Spring Cloud Contract,
- Postman Collections,
- Webmock, Nock и др.

Наиболее распространенным является фреймворк Pact. Он позволяет создавать контракты используя повсеместно использующиеся тесты основанные на “user stories”. В ходе выполнения тестов происходит вызов сервисов поставщиков, что и использует Pact. Он сохраняет запрос и ответ в форме контракта и далее передает полученный файл на специальный выделенный сервер, именуемый Contract Registry. Pact можно использовать для всех популярных языков программирования с использованиям различных каналов связи (HTTP, gRPC, Message Queues). Недостатком данного фреймворка является то, что он ориентирован на потребителя. Это означает то, что сначала составляются контракты потребителями, и только на основе этих контрактов пишется код API поставщика, что в рамках поставленной цели не является приемлемым.

Spring Cloud очень похож на Pact за исключением того, что он сильно интегрирован в среду микро-сервисов написанных на Java и Spring Cloud в частности. Безусловно в данную среду можно внедрить и решения написанные на других языках программирования, но если таких большинство, то использование средств опирающихся на JVM не имеет смысла.

VCR так же напоминает Pact, но он не ориентирован на потребителя. Основная идея – запоминать ответы на определенные запросы. Написав тест, при первом его запуске произойдет запрос к реальному сервису и VCR запомнит ответ на этот запрос. При дальнейшем запуске теста вместо вызова реального сервиса будет подставляться его ответ. К сожалению, данный подход не удовлетворяет требованию проверки схем поставщика основываясь на контрактах потребителя, так как все контракты хранятся локально, и у поставщиков нет доступа к данным контрактам. Используется данный способ преимущественно для тестирования сервисов потребителя и имитации поведения поставщика.

Postman Collections - это выполняемые спецификации API, с фокусом на CDC (Consumer Driver Contracts, то есть контракты ориентированные на потребителя). Есть возможность добавления динамического поведения путем написания скриптов и сверки ответов в тестах. Описание тела ответа при помощи JSON Schema. По сути, данный способ не является контрактным тестированием, а скорее является методом тестирования сервисов поставщика, при чем без какого-либо участия потребителей.

Используя приведенный выше анализ, было решено создать решение, которое имело бы центральный “репозиторий” контрактов и схем поставщиков для того, чтобы можно было их сравнивать и, следовательно, уменьшать количество дефектов в ПО.

Проверка для стороны потребителя заключается в удостоверении того, что потребитель использует то, что поставщик может предоставить, при чем использует с правильными заголовками, фильтрами и типами данных.

Проверка для стороны поставщика заключается в том, что все публикуемые схемы API поставщиков будут проверяться на то, что не было внесено таких изменений, которые не позволили бы уже опубликовавшим контракт потребителям использовать новую версию API. Например, если случайно было удалено одно из полей фильтрации или удален заголовок HTTP.

После того, как я определился с тем, какой сервис необходимо разработать, я приступил к исследованию возможностей составления контрактов. Безусловно, самым очевидном оказался “ручной” способ, при котором все контракты пришлось бы писать вручную. Очевидно, данный способ не допустим так как при нем можно с легкостью допустить ошибку и этот процесс неоправданно трудозатратен. Писать же свою библиотеку для записи контактов на основе тестов поставщика показалось применимой идеей, но мои наставники отговорили меня от этой идеи ссылаясь на ограниченность времени работы над проектом. В связи с чем было решено использовать готовые средства записи контрактов. Так было решено использовать контракты VCR.

Основываясь на исследованиях возможностей публикации контрактов, я определил две категории:

- ручная публикация, при которой контракт вручную отсылался бы в разрабатываемый регистратор контрактов либо при помощи специально разработанного веб-интерфейса, либо напрямую из консоли,
- автоматическая публикация, например такая как в фреймворке Pact.

Решено было использовать ручную публикацию, так как VCR не позволяет публиковать контракты из “коробки”, хотя в случае, если времени будет достаточно, можно будет дописать свою надстройку над данной библиотекой.

Так как спецификации API сервисов поставщика преимущественно используют OpenAPI [5], то их генерация ложится на плечи специализированного ПО наподобие Swagger от компании SmartBear [6]. Решение по публикации было принято в сторону ручного способа.

Последней задачей на данную эпоху было получение знаний о контейнеризации и кластеризации. Данные знания полезны тем, что позволяют лучше понять то, как работает микро-сервисная архитектура, и, следовательно, грамотнее построить регистратор контрактов и интегрировать его в рабочую инфраструктуру. Изучив пару статей на открытых ресурсах, было решено целенаправленно взять курс по данной теме, так как в рамках самостоятельного изучения ушло бы гораздо больше времени, чем то, что доступно в рамках одной эпохи.

# 4 ЭПОХА 3

В рамках данной эпохи я провел исследование возможных видов от потребителя к поставщику, исследовал способы сопоставления схем поставщика и контрактов потребителя, разработал алгоритм сопоставления, спроектировал концепт DSL для описания контрактов, исследовал возможности интеграции регистратора контрактов (РК) с элементами непрерывной интеграции и доставки на примере GitLab Pipeline, а также исследовал возможности генерации схем OpenAPI.

Работа началась с того, что я определил каким образом будет генерироваться схема OpenAPI у большинства веб-сервисов в выделенной инфраструктуре. Дело в том, что большинство веб-приложений в данной инфраструктуре написаны на фреймворке FastAPI [7], а данный фреймворк генерирует эти схемы автоматически, и к ним можно с легкостью получить доступ по адресу схожему с http://127.0.0.1:8000/openapi.json.

Далее я изучил элементы GitLab Pipeline. Делал я это с целью вычисления возможностей оповещения потребителей о том, что схема, которую указали в контракте помечена устаревшей. Без использования GitLab Pipeline автоматическое оповещение потребителей представляется невозможным. Среди всех механизмов работы GitLab Pipeline, наиболее применимыми в данном случае оказались так называемые триггеры [8]. Суть их заключается в том, что, когда они срабатывают, происходит выполнение действия, закрепленного за этим триггером. Вызвать срабатывание триггера можно несколькими способами, но наиболее логичным я посчитал использовать специальный POST запрос, который можно определить для каждого проекта. Чтобы этот POST запрос сработал так, как от него ожидается, необходимо в тело запроса прикрепить специальный токен для аутентификации и передать возможные параметры (если необходимо). Пример такого запроса с использованием утилиты curl приведен ниже.

```
curl --request POST \
  --form token=TOKEN \
  --form ref=main \
  --form "variables[UPLOAD_TO_S3]=true" \
  "https://gitlab.example.com/api/v4/projects/123456/trigger/pipeline"
```

Пример кода 1 — POST запрос для активации триггера внутри GitLab Pipeline

Возможные варианты запросов я обсуждал с наставниками по проекту. Было установлено, что запросы преимущественно ориентированы на получение данных. Смысл анализа вариантов запроса заключается в том, что он позволит определить какие атрибуты необходимо учитывать в контрактах и схемах для их корректного сопоставления. В результате данного анализа я установил, что сопоставлять запросы необходимо по:

- HTTP методу,
- имени хоста,
- расположению,
- заголовкам запроса,

- фильтрам:
    - параметры пути,
    - параметры запроса,
    - параметры заголовков,
- полезной нагрузке запроса (payload).

На основе данного анализа я составил алгоритм сопоставления контракта со схемой, который описан в приложении 1. Основная идея заключается в проверке на соответствия каждого из указанных пунктов в контракте аналогичному в схеме.

Учитывая, что есть вероятность того, что я успею написать надстройку над библиотекой VCR для генерации контрактов и из-за наличия портов VCR на все популярные языки программирования, я решил в качестве контракта использовать модель контракта из библиотеки VCR [9]. Для данной модели я написал JSON Schema, для удобной валидации составляемых контрактов, которая приведена в приложении 2.

# 5 ЭПОХА 4

В последней эпохе семестра я сосредоточился на формализации всех накопленных мною знаний по данному проекту. Я разработал архитектуру сервиса в виде UML диаграммы компонентов, задокументировал предполагаемые конечные точки сервиса РК, составил структуру БД в виде ER-модели, а также задокументировал варианты использования сервиса в виде UML диаграммы прецедентов. Также я согласовал технологии, которые буду применять для написания РК с моими наставниками.

Работа началась с создания диаграммы прецедентов. Она позволила установить какие возможные конечные точки будут реализованы в РК. Диаграмма прецедентов приведена на рисунке 1.

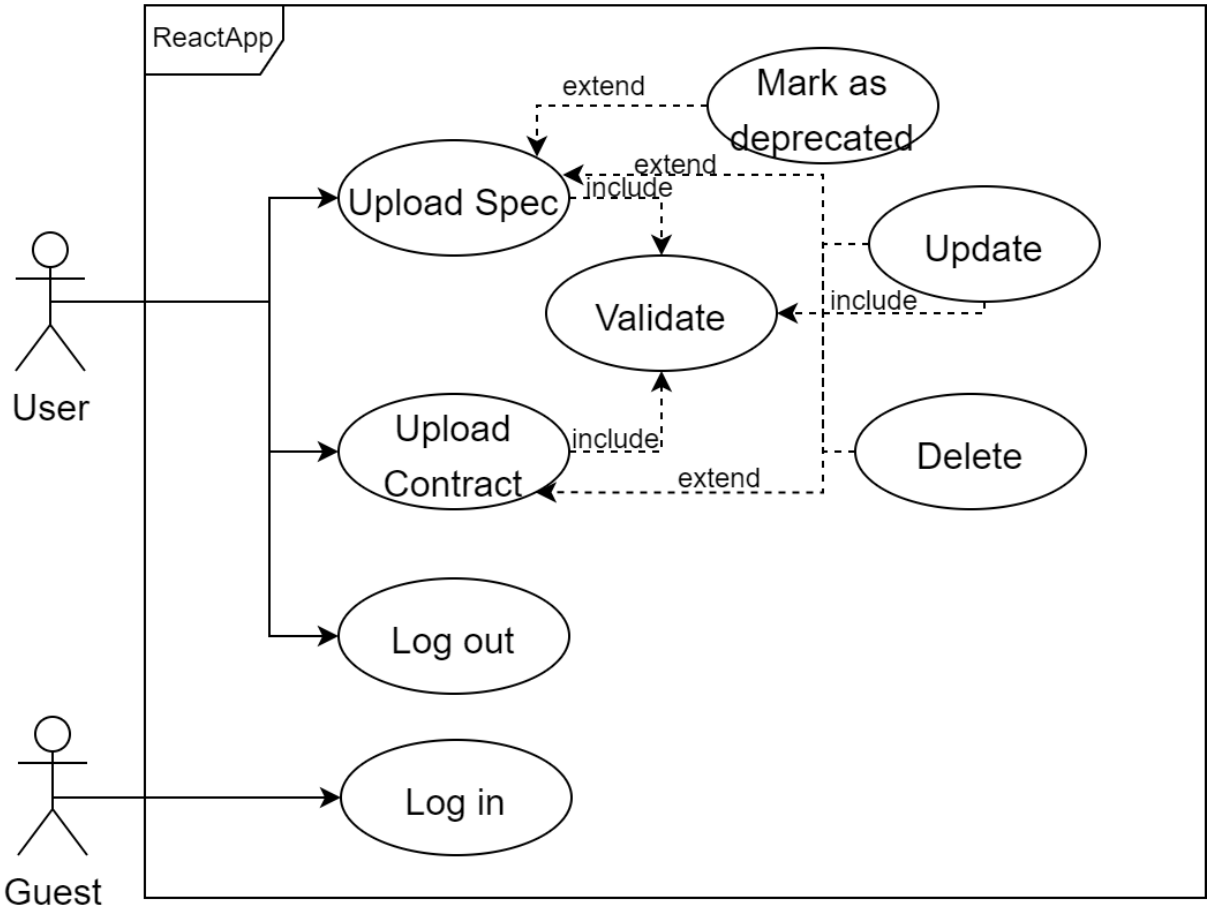


Рисунок 1 – Диаграмма прецедентов для разрабатываемого РК

После этого я задокументировал возможные конечные точки РК:

- `/user/login/` - авторизация пользователя РК
- `/user/logout/` - снятие авторизации пользователя РК
- `/contracts/upload/` - загрузка контракта
- `/contracts/{id}/update/` - обновление контракта

- `/contracts/{id}/delete/` - удаление контракта
- `/contracts/{id}/` - просмотр контракта по id
- `/contracts/` - просмотр всех контрактов
- `/schemas/upload/` - загрузка схемы
- `/schemas/{id}/update/` - обновление схемы
- `/schemas/{id}/delete/` - удаление схемы
- `/schemas/{id}/deprecated/` - пометка схемы устаревшей
- `/schemas/{id}/` - просмотр схемы по id
- `/schemas/` - просмотр всех схем

Далее я разработал архитектуру разрабатываемого сервиса. Она представлена на рисунке 2.

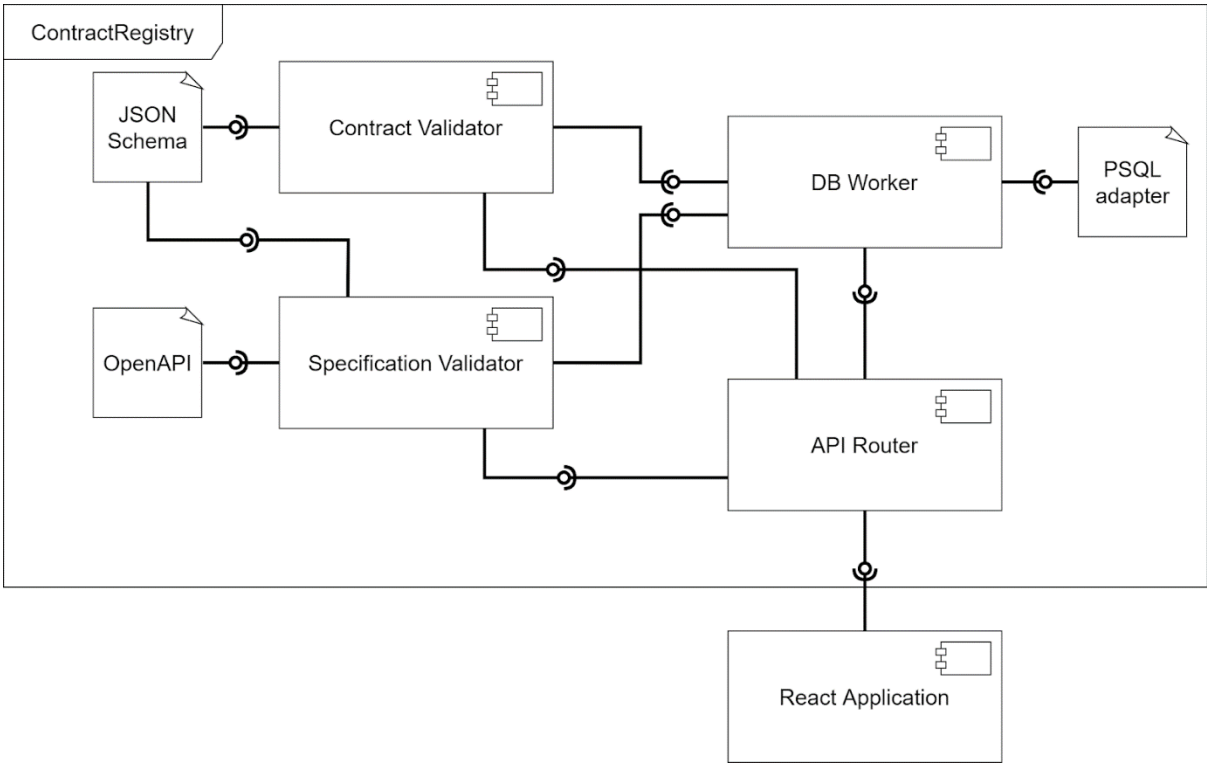


Рисунок 2 – Диаграмма компонентов разрабатываемого сервиса РК

База данных была спроектирована в последнюю очередь. ER-модель представлена на рисунке 3.

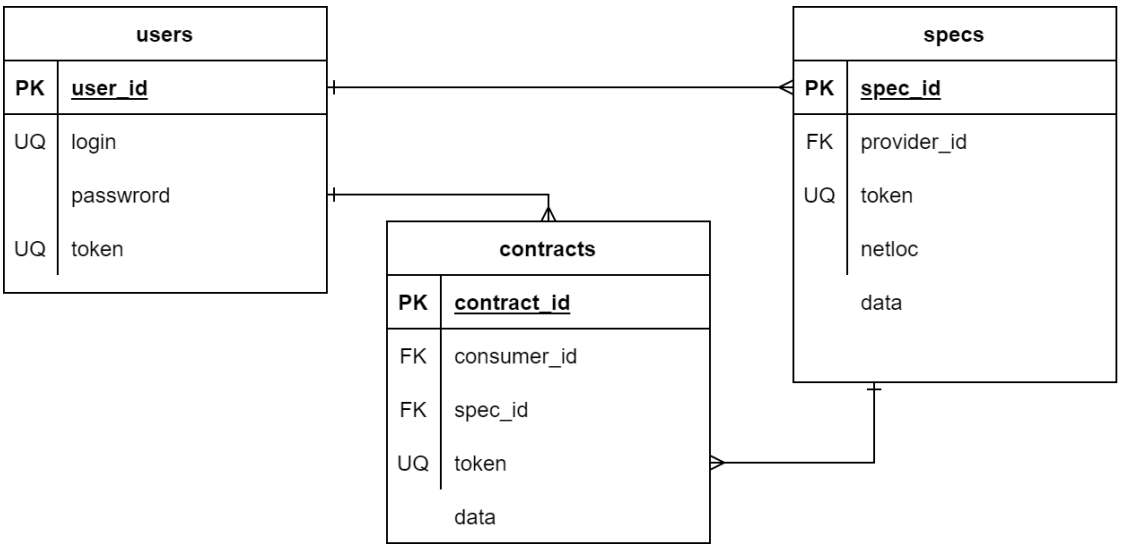


Рисунок 3 – ER-модель разрабатываемого сервиса РК

В БД контракты и схемы планируется хранить в виде текстовых представлениях формата JSON в соответствующих полях data. Токены нужны для обращения к GitLab Pipeline.

Разумеется, все представленные модели не являются финальными и подлежат доработке. Но на них можно опереться при начале разработке кода, и со временем исправить и сами модели. К сожалению, у меня не хватает опыта и навыков чтобы выполнять фазу проектирования лишь единожды, поэтому планируется подход работ аналогичный Scrum. В качестве инструмента написания сервиса планируется использовать Python, а именно фреймворк FastAPI. К счастью, в ходе текущего семестра я смог освоить данный язык программирования. Для реализации веб-интерфейса сервиса было решено использовать библиотеку React. База данных же является реляционной, планируемая СУБД – PostgreSQL.

# 6 Оформление отчетной документации

Во время и после окончания работ, связанных с выполнением проекта, необходимо было документировать прогресс и полученные результаты. Документирование производилось в соответствии с требованиями [10].

Подготовку к написанию отчета я начал с описания целей и задач, так как было необходимо понять, что в процессе работы над практическим заданием необходимо получить. Также было решено описать актуальность цели – почему необходимо ее достичь и чем разработанное решение может помочь. Помимо этого, было размечено содержание, так как это помогает наглядно увидеть структуру будущей работы, и помогает определиться со значимостью той или иной ее части.

По итогам изучения всей необходимой информации, была разработана архитектура, которая решает поставленные задачи. Далее, я подробно описал все проделанные шаги и их результаты, которые я сделал, а также я описал основную часть, а именно то, что, как и почему я делал так как я сделал, а не иначе.

После того, как основная часть была готова, я сформулировал и записал выводы о проделанной работе, после чего оформил весь написанный текст в соответствии с требованиями.

# ЗАКЛЮЧЕНИЕ

В ходе данной работы было произведено немалое количество артефактов, которые помогут с дальнейшей работой над решением. Наиболее значимыми артефактами я считаю диаграммы, разработанные в последней эпохе, так как они формализуют все накопленные знания и дают визуальное представление того, что необходимо будет сделать в рамках следующих эпох.

Немалое количество практик из аналогов было заимствовано, что говорит о том, что работы по их анализу не прошли даром. Из аналогов я подчерпнул архитектуру приложения, а также DSL для описания контрактов.

Говоря откровенно, большинство уже готовых артефактов будут изменены, но это не сигнализирует то, что работа в данном семестре была напрасной, ведь именно полученные знания дают траекторию на будущие изменения и продвижения в проекте.

# СПИСОК ЛИТЕРАТУРЫ


1. Паттерн: Контрактные Тесты Сервиса Интеграций - URL: https://microservices.io/patterns/testing/service-integration-contract-test.html
2. Л. Криспин, Д. Грегори Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд – 2016 г.
3. Фреймворк контрактного тестирования Pact – URL: https://docs.pact.io/
4. Исходный код библиотеки контрактного тестирования VCR – URL: https://github.com/vcr/vcr
5. Спецификация OpenAPI – URL: https://swagger.io/specification/
6. Библиотека генерации спецификаций OpenAPI – URL: https://swagger.io/
7. Фреймворк для разработки веб-сервисов FastAPI – URL: https://fastapi.tiangolo.com/
8. Триггеры в системе постоянной интеграции и доставки GitLab – URL: https://docs.gitlab.com/ee/ci/triggers/
9. Пример контракта, использующегося библиотекой VCR – URL: https://github.com/betamaxpy/betamax/blob/master/examples/cassettes/more-complicated-cassettes.json
10. Т.А. Маркина, А.В. Пенской, Д.Г. Штенников, Е.Ю. Авксентьева, А.Г. Ильина ПРОИЗВОДСТВЕННАЯ ПРАКТИКА МАГИСТРАНТОВ: ОРГАНИЗАЦИЯ И ПРОВЕДЕНИЕ – 2020 г. – 50 с.

# ПРИЛОЖЕНИЕ 1

```
    def validate_by_spec(self, contract_file, spec_file):
        contract = self._read_file(contract_file)
        spec = self._read_file(spec_file)
        for index, interaction in enumerate(contract['http_interactions']):
            uri = interaction['request']['uri']
            uri_parsed = urlparse(uri)
            path = uri_parsed.path
            if path not in spec['paths']:
                raise ValidationException(f'Given inappropriate path "{path}" in
interaction[{index}]')
            method = interaction['request']['method'].lower()
            if method not in list(map(lambda x: x.lower(), spec['paths'][path])):
                raise ValidationException(f'Path "{path}" does not support method
"{method}"')
            status_code = interaction['response']['status']['code']
            if status_code not in spec['paths'][path][method]['responses']:
                raise ValidationException(f'Requested path "{path}" does not
support  method "{method}"')
            contract_payload = interaction['response']['body']['string']
            content_type = interaction['response']['headers']['content-type']
            if contract_payload:
                if content_type not in
spec['paths'][path][method]['responses'][status_code]['content']:
                    raise ValidationException()
            else:
                is_contains_empty_response = False
                for content in
spec['paths'][path][method]['responses'][status_code]['content']:
                    if not content:
                        is_contains_empty_response = True
                if not is_contains_empty_response:
                    raise ValidationException()
```

Пример кода 2 — Алгоритм сопоставления контракта потребителя схеме поставщика

## ПРИЛОЖЕНИЕ 2

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "properties": {
      "consumer": { "type": "string"},
      "provider": { "type": "string"},
      "http_interactions": {
        "type": "array",
        "items": [
            {
                "type": "object",
                "properties": {
                    "request": {
                        "type":"object",
                        "properties": {
                            "method": {
                                "type": "string",
                                "pattern": "GET|HEAD|POST|PUT|DELETE|PATCH",
                                "$comment" :"TODO: CONNECT, OPTIONS, TRACE"
                            },
                            "url": {
                                "$comment": "TODO: url pattern",
                                "type": "string",
                                "pattern": ".*"
                            },
                            "headers": {
                                "type": "object",
                                "propertyNames": {
                                    "pattern": "^[A-Za-z][A-Za-z0-9-]*$"
                                },
                                "$comment": "TODO: property and pattern",
                                "patternProperties": {
                                    "": {
                                        "type": "array",
                                        "items": [
                                          {
                                            "type": "string"
                                          }
                                        ]
                                    }
                                }
                            },
                            "body": {
                                "type": "object",
                                "properties": {
                                    "encoding": {
                                        "type": "string"
                                    },
                                    "string": {
                                        "type": "string"
                                    }
```

```
                    },
                    "required": [
                      "encoding",
                      "string"
                    ]
                }
            },
            "required": [
                "method",
                "url",
                "headers",
                "body"
            ],
            "additionalProperties": false
        },
        "response":{
            "type":"object",
            "properties": {
                "status": {
                    "type": "object",
                    "properties": {
                        "code": {
                            "type": "integer"
                        },
                        "message": {
                            "type": "string"
                        }
                    },
                    "required": [
                        "code",
                        "message"
                    ]
                },
                "url": {
                    "$comment": "TODO: url pattern",
                    "type": "string",
                    "pattern": ".*"
                },
                "headers": {
                    "type": "object",
                    "propertyNames": {
                        "pattern": "^[A-Za-z][A-Za-z0-9-]*$"
                    },
                    "$comment": "TODO: property and pattern",
                    "properties": {
                        "": {
                            "type": "array",
                            "items": [
                              {
                                "type": "string"
                              }
                            ]
                        }
                    }
```

```
                        },
                        "body": {
                            "type": "object",
                            "properties": {
                                "encoding": {
                                    "type": ["string", "null"]
                                },
                                "string": {
                                    "type": ["string", "null"]
                                }
                            },
                            "required": [
                              "encoding",
                              "string"
                            ]
                        }
                    },
                    "required": [
                        "status",
                        "url",
                        "headers",
                        "body"
                    ],
                    "additionalProperties": false
                }
            },
            "required": [
                "request",
                "response"
            ]
        }
      ]
    }
  },
  "required": [
    "http_interactions",
    "consumer",
    "provider"
  ]
}
```

Пример кода 3 — JSON Schema для валидации контрактов потребителя