**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 P41191

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.

В рамках работы исследования текущего семестра, были поставлены следующие задачи:

1. Составление технического задания на проект.
2. Согласование технического задания на проект.
3. Документирование сценариев добавления, обновления, удаления и сопоставления контрактов и схем в виде диаграммы последовательностей
4. Изучение форматеров и линтеров:
    a. Форматеры – black, isort
    b. Линтеры – flake8, flake8-bugbear, flake8-pytest-style, flake8-quotes
5. Внедрение форматеров и линтеров в разрабатываемый сервис
6. Изучение библиотек тестирования кода – pytest, pytest-cov
7. Внедрение библиотек тестирования кода в разрабатываемый сервис
8. Изучение видов типизаций в python, внедрение типизации mypy в разрабатываемый сервис
9. Изучение подхода “conventional commits” и его внедрение в разрабатываемый сервис
10. Создание алгоритма разграничения доступа на основе встроенных примитивов языка python (например, словарь)
11. Создание конечной точки сервиса для публикации контрактов
12. Создание конечной точки сервиса для обновления контрактов
13. Создание конечной точки сервиса для удаления контрактов
14. Создание конечной точки сервиса для публикации схем

15. Создание конечной точки сервиса для обновления схем
16. Создание конечной точки сервиса для удаления схем
17. Создание конечной точки сервиса для пометки схемы устаревшей
18. Тестирование работоспособности сервиса на предмет операций CRUD со схемами и контрактами (покрытие тестами не менее 90% кода)
19. Тестирование алгоритма сопоставления схем и контрактов (покрытие тестами не менее 90% кода)
20. Модернизация алгоритма сопоставления схем и контрактов с учетом ссылочных значений
21. Модернизация алгоритма сопоставления схем и контрактов с учетом различных видов описания файлов (yaml, json)
22. Повторное тестирование алгоритма сопоставления схем и контрактов (покрытие тестами не менее 90% кода)
23. Подключение базы данных для хранения контрактов и схем
24. Модернизация алгоритма разграничения доступа с использованием специально создаваемых токенов
25. Создание алгоритма отслеживающего версионирование схем и контрактов
26. Тестирование алгоритма отслеживающего версионирование схем и контрактов (покрытие тестами не менее 90% кода)
27. Модернизация алгоритма операций CRUD для схем и контрактов с учетом алгоритма версионирования и подключенной базы данных
28. Тестирование алгоритма операций CRUD для схем и контрактов (покрытие тестами не менее 90% кода)
29. Исследование способов оповещения потребителей об используемой ими устаревшей версии схемы
30. Создание алгоритма оповещения об устаревшей схеме
31. Тестирование алгоритма оповещения об устаревшей схеме (покрытие тестами не менее 90% кода)

В данном семестре работа была реализована по следующим этапам:

1. инструктаж обучающегося,
2. эпоха 1,
3. эпоха 2,
4. эпоха 3,
5. эпоха 4.

# 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

Работа в данном семестре продолжает мои труды предыдущего семестра. Для более наглядного представления о предстоящих работах необходимо было структурировать процесс создания, обновления, удаления контрактов и спецификаций. Данный процесс был запечатлен в виде UML диаграммы последовательностей. Пример данных диаграмм представлен на рисунках 1 и 2.

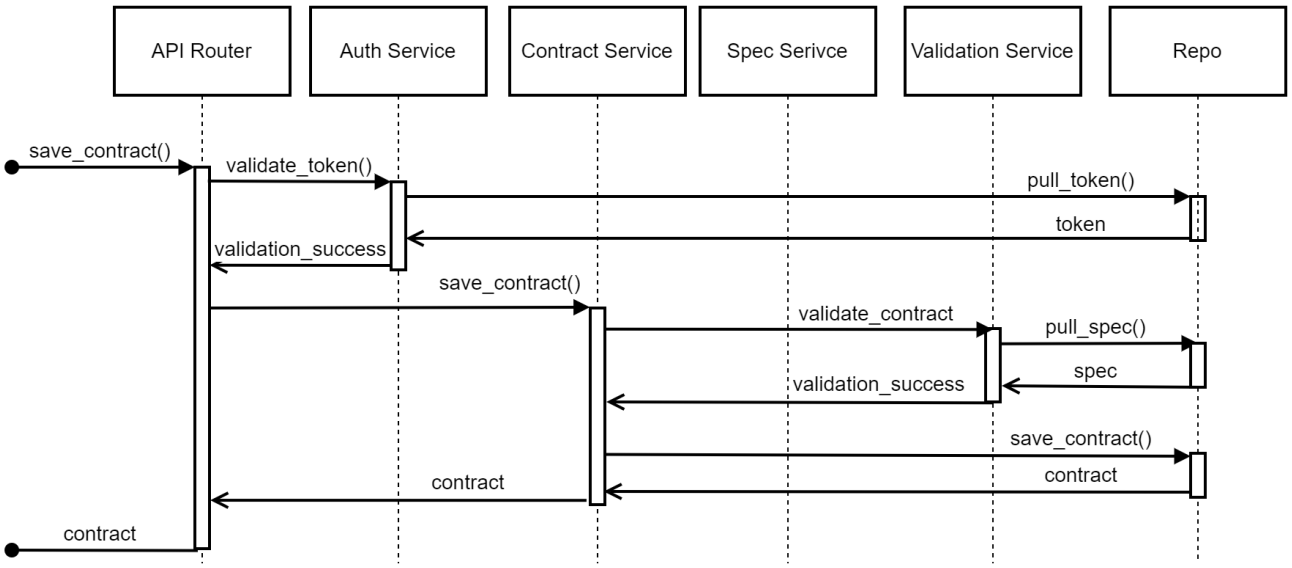


Рисунок 1 – Диаграмма последовательности сохранения контракта

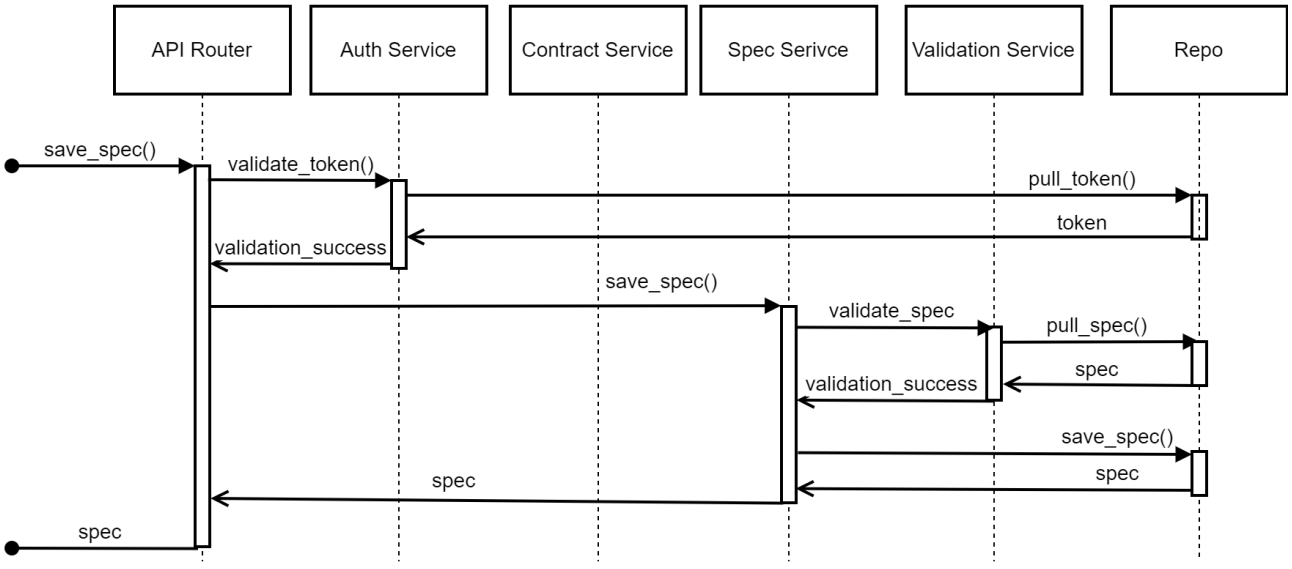


Рисунок 2 – Диаграмма последовательности сохранения спецификации

В рамках предыдущего семестра, с кодом я работал в текстовом редакторе VS Code без использования плагинов для автоматизации рутинных процессов и линтинга. Менторы и з компании GS Labs сделали мне замечание по этому поводу и порекомендовали перейти в IDE,

например в PyCharm и добавить всесторонние утилиты, чтобы улучшить качество и быстроту написания кода. Утилитами, которые я воспользовался, являются:

- Форматеры – black, isort
- Линтеры – flake8, flake8-bugbear, flake8-pytest-style, flake8-quotes
- Библиотеки тестирования кода – pytest, pytest-cov
- Утилита статического анализа типизации mypy
- Утилита  линтинга git commit – commitlint
- Утилита привязки процессов к git-хукам – pre-commit

Форматеры и линтеры помогают держать код в определенном стиле, или даже конвенции, что помогает улучшить его читаемость другим разработчикам. Библиотеки тестирования, очевидно, предназначены для тестирования (в основном юнит-тестирование). Утилита статического анализа типизации помогает следить за использованием переменных ожидаемым образом, то есть производит статическую проверку на типизацию в динамически типизируемом языке, что дает уверенность в том, что во время рантайма код не упадет из-за того, что код ожидал получить строку, а получил целочисленное значение, например. Утилита commitlint форсирует писать коммиты в более декларативном виде [1], привязывая каждый коммит к определенной группе[2].

Наибольший интерес для меня составила утилита pre-commit. Она позволяет привязать запуск всех вышеперечисленных утилит к каждому коммиту в системе контроля версий. Так, при каждом коммите, я уверен, что мой код будет проверен на чистоту, протестирован, а комит будет написан подобающим образом. Считаю работы проделанную в рамках этой эпохи наиболее полезной среди всех проделанных в этом семестре. Но полезной не для проекта, а для меня, как специалиста.

# 3 Эпоха 2

Разграничение доступа – механизм, который не позволит создавать контракты и спецификации не авторизованным пользователям. Но на начальном этапе проектирования его создание оказалось бессмысленным, и даже мешало тестированию конечных точек сервиса, так что его разработка была перенесена на следующие эпохи.

Конечные точки являются тем интерфейсом, который мы предоставляем пользователям нашего сервиса для их удобного взаимодействия. Они должны быть унифицированы по REST конвенции. Следуя этой мысли, я создал так называемые в рамках FastAPI роутеры, которые и являются конечными точками. Для каждого CRUD запроса я использовал соответствующий HTTP-метод. При этом несмотря на то, что сначала все роутеры я хранил в одном файле, в дальнейшем я разделил логически каждый роутер друг от друга и поместил логику в разные файлы. Так же я учел возможность дальнейшего версионирования моего сервиса, поэтому сразу добавил механизм для разграничения функциональности разных версий API. Структуру роутеров можно увидеть на рисунке 3.

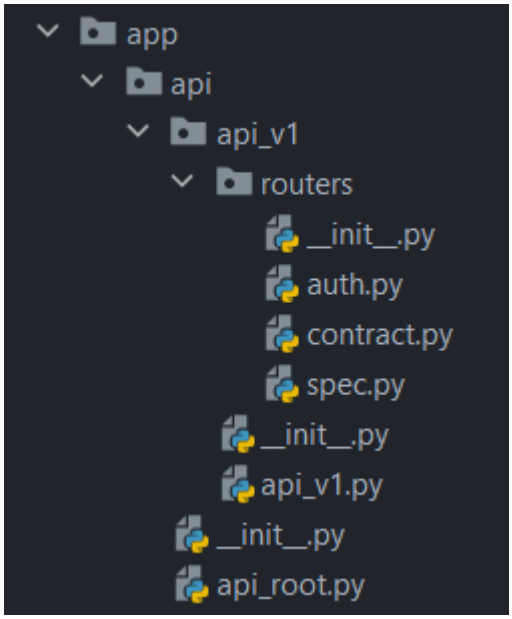


Рисунок 3 – Структура роутов сервиса

В тестировании конечных точек использовалась не только библиотека pytest, но и ручное тестирование с использованием документации OpenAPI, так как она позволяет делать запросы к сервису прямо из описания конечных точек, что очень пригодилось в ходе разработки.

# 4 Эпоха 3

Основной задачей в этой эпохе для меня стало подключение базы данных к сервису. Эта, казалась бы, простая задача обошлась мне в семь световых лет. Но если, серьезно, то подключение базы данных это скорее задача DevOps, нежели разработки. С такого рода задачами, у меня всегда возникают некие оказии. Этот раз не стал исключением, и каждый этап, связанный с внедрением хранилища, у меня сопровождался трудностями. Этапы же подключения хранилища таковы:

1. Подключение SQLite базы данных
2. Подключение утилиты миграций БД alembic
3. Переход с СУБД SQLite на PostgreSQL
4. Переход с синхронной работы с БД к асинхронной

По сути, большую часть эпохи у меня заняла работа с БД, но к счастью, все этапы были выполнены успешно, и я, несомненно, получил бесценны опыт администрирования СУБД.

После успешного внедрения БД необходимо было ввести новые уровни абстракций, так как выполнять работу с БД в конечных точках е является хорошим решением. Так было решено использовать абстракцию «репозиторий» для обеспечения коммуникаций с БД, а в дальнейшем и подхода «сервис» [3] для еще большего облегчения логики конечных точек.

Еще одна задача возникла в ходе консультаций с менторами. Было установлено, что я не использую инструменты для управления зависимостями в python на полную мощность, и менторы порекомендовали мне либо использовать файл requirements.txt, в котором хранился бы список всех зависимостей, либо перейти на такие инструменты, которые подобный файл бы форсировали и вели автоматически. Я выбрал второй вариант, и решил использовать ПО poetry, который облегчил мне работу с зависимостями и виртуальным окружением python.

Различное описание файлов было исключено из озабоченностей, так как изначально предполагалось использовать спецификации как файлы, но в ходе разработки сервиса, было решено использовать спецификации как полезный груз HTTP запросов, а следовательно все данные имеют вид JSON. Тем не менее в случае, если в ходе дальнейших работ необходимо будет вернуться к оперированию с данными в виде цельных файлов, а не полезной нагрузки HTTP методов, уже имеются наработки в этом вопросе, и я всегда смогу к ним вернуться, используя историю коммитов.

# 5 Эпоха 4

В ходе данной эпохи я реализовывал механизмы разграничения доступа и отслеживание пометок схемы устаревшей.

Механизм разграничения я воплотил путем использования токенов. Предположительно каждая команда разработчиков будет иметь свой уникальный токен. Авторизовавшись в сервисе, пользователи получат JWT токен, который позволит им обращаться к соответствующим ресурсам. При этом, при создании контракта или спецификации в БД, так же будет записываться токен команды. При попытках дальнейшего изменения / удаления спецификации или контракта, будет проверено то, что пользователь имеет соответствующий токен на совершение такой операции. Конечные точки, требующие авторизации представлены на рисунке 4. Конечные точки, отмеченные значком замка справа, требуют авторизации. Так, для создания и удаления контракта, для создания, удаления и пометки устаревшей спецификации, необходимо быть авторизованным.

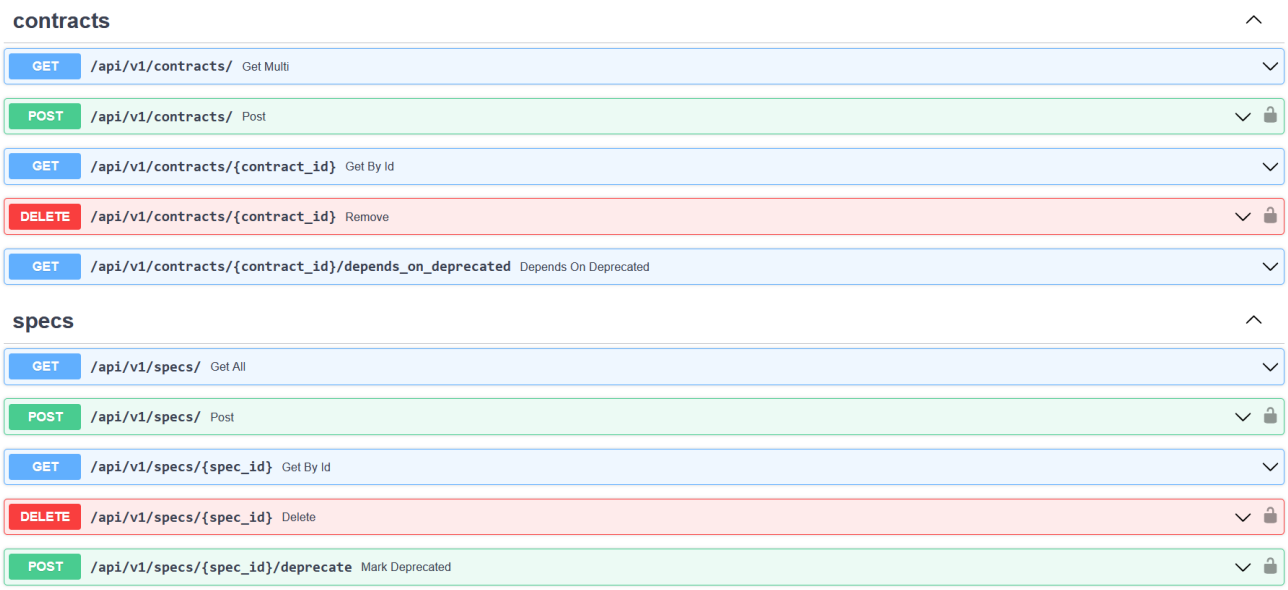


Рисунок 4 – Список конечных точек сервиса

Вопрос устаревших спецификаций стоял довольно остро. Первое, что пришло в голову для того, чтобы форсировать пользователей устаревших спецификаций, это отправлять уведомления в GitLab при помощи хуков. Эта идея пока что не отброшена, но на смену ей пришло более практичное решение. Было решено добавить еще одну конечную точку, чтобы можно было узнать зависит ли контракт от устаревшей спецификации или нет. Дальнейшее

взаимодействие заключается в обращении скриптом во время функционирования CI/CD конвейера GitLab к данной конечной точке, с целью удостоверения в том, что используется не устаревшая спецификация.

Оба вышеперечисленных решения планируются быть реализованы в следующем семестре, но так как работы с конвейером CI/CD GitLab еще не проводились, прогресс в рамках данной эпохи был лишь по одному из способов.

Вопрос версионирования же был сведен к тому, что он необходим лишь для спецификаций, и то, сугубо для удостоверения в том, что конечные точки не противоречат друг другу. Версионирование контрактов не имеет смысла, так как при валидации спецификаций, будут проанализированы все версии контрактов.

Несмотря на то, что абстракции «репозиторий» и «сервис» я реализовал в предыдущей эпохе, задумался я над тем, как стоит их реализовывать грамотно лишь в текущей эпохе. Дело в том, что данные абстракции можно реализовывать в процедурной парадигме программирования и в ООП. Это побудило меня исследовать на сколько python зависит от ООП парадигмы, и как я должен использовать классы в моем решении.

Прочитав несколько статей по поводу того, как классы являются излишним программированием в python [4] и о внедрении зависимостей [5], я решил «отрефакторить» код в ООП парадигме и реализовать абстракции репозиторий и сервис в виде классов. Зависимости я решил, пока-что оставить в виде зависимостей методов, хотя данный подход меня несколько настораживает, так как этот подход сильно связывает мой код, что уменьшает его переиспользуемость. Данный вопрос я не успел обсудить с менторами в полной мере, поэтому он перетекает в следующий семестр.

# ЗАКЛЮЧЕНИЕ

Работу в данном семестре я считаю очень продуктивной. Большинство поставленных целей имели смысл и были выполнены успешно. В ходе коммуникаций с менторами были улучшены аспекты проекта и его разработки. Все проделанные действия я считаю полезными в первую очередь не для проекта, а для меня. Несмотря на то, что проект продвинулся в своем развитии, я продвинулся куда дальше в развитии как специалист. Под этим я подразумеваю опыт конфигурации БД, опыт аналитики, составления задач и их декомпозиций, опыт улучшения рабочего процесс путем использования сторонних утилит (форматирование, линтинг).

Самым вызывающим для меня шагом было подключение БД и в дальнейшем реализация асинхронного подхода (что повлекло изменения во всем проекте). Большое количество времени пришлось потратить на изучение документаций по ORM SQLAlchemy [6]. Проблемы здесь вызывало так же наличие нескольких стандартов описания запросов к БД;

- Стандарт до выхода версии SQLAlchemy 1.4
- Стандарт описания запросов к БД SQLAlchemy 2.0 [7]

Работа по переходу с одной версии на другую так же подняло мое понимание устройства данной ORM, что является большим плюсом, ведь данная библиотека чуть ли не является монополией в области работы с реляционными СУБД на языке python.

Единственным негативным моментом в данном семестре для меня стало то, что я не смог адекватно рассчитать время, затрачиваемое на курсы и время работы над проектом. Так как курсы я взял преимущественно на английском, то время, закладываемое в их обучение, возросло заметно. На это так же повлияло несколько разное представление о работе над курсом в наших широтах, в сравнении с западным. Из-за этого мне прошлось пожертвовать временем работы над проектом, ведь в случае невыполнения некоторых задач по проекту, я могу перенести их на следующий семестр, но невыполнение курсов влечет более серьезные последствия. В связи с этим, я не успел провести модернизацию алгоритма валидации контрактов и спецификаций с учетом ссылочных значений и версионирования спецификаций.

Следует так же отметить, что каждая последующая задача давалась мне легче, чем предыдущая. Это, как мне кажется, связано с тем, что задачи опираются на моих предыдущих

успехах и реализовав грамотный плацдарм, строить новый функционал сервиса является гораздо более легкой задачей.

# СПИСОК ЛИТЕРАТУРЫ


1. Спецификация подхода написания коммитов «Conventional Commits» - URL: https://www.conventionalcommits.org/en/v1.0.0/
2. Основные группы определенные в рамках подхода написания коммитов «Conventional Commits» - URL: https://github.com/angular/angular/blob/main/CONTRIBUTING.md#type
3. Паттерн проектирования Репозиторий-Сервис – URL: https://exceptionnotfound.net/the-repository-service-pattern-with-dependency-injection-and-asp-net-core/
4. Перестаньте писать классы – URL: https://habr.com/ru/post/140581/
5. Dependency injection – URL: https://habr.com/ru/post/350068/
6. Документация SQLAlchemy – URL: https://www.sqlalchemy.org/
7. Migrating to SQLAlchemy 2.0 – URL: https://docs.sqlalchemy.org/en/14/changelog/migration_20.html