Команда «Суперчипс»

Проект Flow (Pinterest)

Весна 2025

ДОМАШНЕЕ ЗАДАНИЕ №2

Создание учётных записей

Проект включается основной сервис `main` и несколько микросервисов: `auth`, `feed`, `chat` и `websocket_chat`. Для каждого сервиса используется своя учётная запись. Архитектура сервисных учётных записей была разработана со следующими требованиями:

- SQL файлы для создания ролей должны лежать в одном месте в папке `./db/roles/`.
- Должна иметься возможность задать логин и пароль для учётной записи в файлах конфигурации проекта.
- Учётная запись должна создаваться/пересоздаваться в БД при перезапуске приложения.

На самом деле, для обеспечения этих требований была проделана большая работа по совершенствованию процесса конфигурирования проекта.

Для генерации ролей при каждом перезапуске приложения был введён дополнительный контейнер `db_after_init_script` между контейнером с БД и контейнерами с сервисами. Данный контейнер исполняет скрипт, генерирующий роли, `db_refresh_roles.sh`, который лежит в папке `./db/db_after_init_script/`. Изначально, может показаться, что это «костыль». Профессиональным решением является модифицировать запускающий скрипт официального docker образа Postgres. Однако это повлечёт затаскивание в проект довольно сложного скрипта, редактирование которого может сломать стандартное поведение образа Postgres, что очевидно нехорошо. Кроме того, у промежуточного контейнера есть свои плюсы. Во-первых, он прозрачен и ясен. Во-вторых, он начинает создавать роли только после готовности БД, так как пробуждается по `service_healthy`, и только после успешного создания ролей начинают свою работу контейнеры сервисов, так как

они пробуждаются по `service_completed_successfully`. Таким образом, при правильной конфигурации проекта сервисы всегда будут иметь свои сервисные учётные записи на момент пробуждения.

В папке `./env/database/roles/` лежат файлы конфигурации учётных записей в формате: логин, пароль и название SQL файла для создания роли из папки `./db/role/`. Таким образом, явно видно соответствие между учётными данными и файлом генерации.

SQL миграции намеренно не используются, чтобы права и возможности сервисных учётных записей были прозрачны.

На данный момент все учётные записи генерируются из SQL файла `flw_srv.sql`. Роль даёт право на модификацию данных в таблице, но не даёт возможности создавать, удалять и изменять структуру таблиц.

Скрипт для генерации ролей лежит в папке `./db/db_after_init_script/`, потому что скрипт предназначен для автоматической постинициализации БД.

Защита от SQL инъекций

С самого начала проект использовал встроенные функции GO `Query`, `QueryRow` и `Exec` для взаимодействия с БД, которые включают этап построения prepared statement. Прямая подстановка значений в SQL запросы с помощью пакета `fmt` нигде не применяется. Так как query builder'ы использовать нельзя, особое внимание уделено конструированию `UPDATE` запросов с варьируемых числом изменяемых атрибутов (смотреть `./internal/repository/pg/pincrud_rep.go`). Используются два массива – строки `UPDATE` запроса и значения атрибутов, – для каждого изменяемого атрибута в массив строк добавляется пара:

```
{название атрибута} = ${индекс в массиве значений атрибутов}
```

Затем неизменяемая шапка запроса и строки из массива конкатенируются и передаются в `Exec`, а массив значений развёртывается оператором `..`, что обеспечивает построение prepared statement → защиту от SQL инъекций.

Проект построен на принципах чистой архитектуры, поэтому пользователь взаимодействует с БД **только** через слои delivery и usecase, предусматривающие логический (смысловой) парсинг значений. Данные передаются через типизированные переменные и структуры, поэтому в поле ID типа `uint64` никогда не попадёт строка `{ID}' {код злоумышленника} --` и другие подобные штуки.

Для защиты от XSS у каждой структуры имеется функция Escape, которая экранирует символы <, >, &, ‘ и “.

Пул соединений и параметры соединений

В качестве ограничения на количество соединений со стороны БД `max_connections` выбрано 50. Согласно информации из интернета одно соединение занимает около 5 Мбайт ОЗУ. На нашей машине 4ГБ ОЗУ, поэтому место под такое количество процессов имеется. Оптимизировать по другим параметра системы не получилось, так как нам выдан pode в Kubernetes, изнутри которого реальные ограничения железа не узнать.

Для пула соединений основного приложения и микросервисов выбраны следующие параметры соединений:

```
MaxOpenConns=8

MaxIdleConns=4

ConnMaxIdleTime=15m
```

Так как трафик на сайте небольшой, разумно отключать соединение, когда сайт уже исследовали и им никто не пользуется. Через 15 минут отсутствия запросов можно с уверенностью утверждать, что пользователь покинул сайт.
Максимальное суммарное количество соединений всех сервисов равно 40, что не превышает `max_connections`.

Postgres запущен внутри контейнера вместе с остальными частями приложения на одной машине, пользователи напрямую не взаимодействуют с БД, поэтому открытых портов для внешнего взаимодействия нет. Чтобы Postgres работал внутри Docker’а, необходимо указать «*» внутри `listen_addresses` (подтверждение).

Таймауты

В качестве значения `statement_timeout` выбрана 1 секунда, так как современные приложения не могу позволить себе бОльшого времени ожидания. Наше приложение относится к сфере развлечений, поэтому должно обеспечивать быструю загрузку контента.

На данный момент приложение не содержит сложных запросов. С одной стороны, `lock_timeout` стоит поставить как можно меньше, чтобы потенциальные DDoS запросы быстрее отменялись. С другой стороны, необходимо подобрать такой `lock_timeout`, чтобы любой запрос успевал выполниться. Для правильного ответа на этот вопрос требуются реальные данные работы сервиса. Поэтому в рамках учебного проекта приходится подбирать значение «примерно». В качестве отправной точки выбран `lock_timeout` равный 3 секундам.

Логгирование и протоколирование медленных запросов

Подгрузка расширений `pg_stat_statements` и `autoexplain` выполняется в `postgresql.conf`.

Данный проект должен быть продемонстрирован на защите. Поэтому минимальные требования к скорости запросов в БД должны вычисляться, исходя из числа желающих посетить сайт во время защиты. Около 10-ти преподавателей и не более 20 обучающихся. Скорее всего, пользователи будут генерировать менее 1-2 запросов в секунду. Таким образом, суммарная нагрузка будет равна примерно 50 RPS. Значит, в среднем БД должна справляться с запросом менее чем за 20мс. Запросы, требующие большего времени, можно считать медленными.

В целом, про настройки больше особо нечего написать – их стоит просто посмотреть. Файл `postgresql.conf` лежит в папке `./db/`.