Справочная информация по использованию библиотеки oatpp-postgres

Документация: https://oatpp.io/docs/components/orm/
Данная библиотека является частью библиотеки **oatpp**, поэтому перед использованием **oatpp-postgres** необходимо установить **oatpp**.

Библиотека позволяет использовать средство ORM для работы с БД postgres из oatpp-приложения.

Установка компонентов

1. Перед работой с **oatpp-postgres** необходимо установить **PostgreSQL** https://www.postgresql.org/download/ и создать БД. Создание БД происходит с помощью команды **create database ИМЯ_БАЗЫ_ДАННЫХ**;
2. Создаем проект, выделяем директорию, в которой будут храниться скачанные библиотеки.
   mkdir utility/tmp
   cd utility/tmp

3. Установка **oatpp**:
   3.1. Клонируем репозиторий oatpp на GitHub: **git clone https://github.com/oatpp/oatpp.git**
   3.2. Получаем теги из репозитория: **git fetch --tags**
   3.3. Переходим на тег: **git checkout 1.3.0-latest --**
   3.4. Переходим в пакет: **cd oatpp**
   3.5. Устанавливаем библиотеку с помощью CMake:
   mkdir build
   cd build
   cmake ..
   make install

4. Установка **oatpp-postgres**:
   4.1. Клонируем репозиторий oatpp-postgres на GitHub: **git clone https://github.com/oatpp/oatpp-postgresql.git**
   4.2. Переходим в пакет: **cd oatpp-postgresql**
   4.3. Получаем теги из репозитория: **git fetch --tags**
   4.4. Переходим на тег: **git checkout 1.3.0 --**
   4.5. Для ОС Windows находим файл CMakeList.txt и устанавливаем переменную с адресом к директории include к установленной БД PostgreSQL: `set(PostgreSQL_INCLUDE_DIR "E:\\program\\Postgresql\\include")`
   4.6. Устанавливаем библиотеку с помощью CMake:
   mkdir build
   cd build
   cmake ..
   make install

Создание приложения

1. Создание миграций. Создаются скрипты для создания таблиц, ролей, прав доступа. Директория, где расположены миграции указана в CMakeLists.txt параметр DDATABASE_MIGRATIONS .
2. Создание моделей данных. Модели данных - DTO должны соответствовать схеме таблиц PostgreSQL. Столбцы и названия полей должны быть названы одинаково для отображения данных из БД на DTO. Если в БД столбцы называются с нижними подчеркиваниями, то для их отображения на DTO можно воспользоваться алиасом as при извлечении данных из БД.
3. Создание **DbClient (репозиторий)**. В рамках репозитория происходит создание запросов и исполнение миграций. Нумерация миграций начинается с 1. Если миграция была применена к БД, то версия <= последней примененной, не будет повторно выполняться.

   Применение миграции:

```
FeedbackDb(const std::shared_ptr<oatpp::orm::Executor>& executor)
  : oatpp::orm::DbClient(executor)
{
  oatpp::orm::SchemaMigration migration(executor);
  migration.addFile(1 /* Версия 1 */, DATABASE_MIGRATIONS "/001_init.sql");
  migration.addFile(2 /* Версия 2 */, DATABASE_MIGRATIONS "/002_init.sql");
  migration.migrate(); // <-- применение миграции

  auto version = executor->getSchemaVersion();
  OATPP_LOGD("FeedbackDb", "Migration - OK. Version=%lld.", version);
}
```

   Для создания запроса в **репозитории** создается метод `QUERY`, в который передается sql-запрос. Для передачи параметров в запрос используется синтаксис :имяПараметра, где имяПараметра - имя параметра, передаваемого в метод.

   Пример создания запроса:

```
QUERY(updateFeedback,
  "UPDATE feedback.feedback "
  "SET update_date = CURRENT_TIMESTAMP, "
  "description = :feedback.description "
  "WHERE  id=:feedback.id;",
  PARAM(oatpp::Object<FeedbackDto>, feedback))
```

4. Создаются компоненты OATPP_CREATE_COMPONENT для репозиториев. Для подключения к БД необходимо передать строку подключения вида: **`postgresql://postgres:db-pass@localhost:5432/postgres`**, где указываются необходимые параметры конфигурации для подключения к базе данных PostgreSQL (хост, порт, имя базы данных, имя пользователя, пароль).

   Пример создания репозитория:

```
OATPP_CREATE_COMPONENT(std::shared_ptr<FeedbackDb>, feedbackDb)([] {
```

```
OATPP_COMPONENT(oatpp::Object<ConfigDto>, config); // Конфигурация
подключения
 /* Создание ConnectionProvider */
auto connectionProvider =
std::make_shared<oatpp::postgresql::ConnectionProvider>(config->dbConnection
String);
 /* Создание ConnectionPool */
auto connectionPool =
oatpp::postgresql::ConnectionPool::createShared(connectionProvider,
10 /* max-connections */,
std::chrono::seconds(5) /* connection TTL */);
 /* Создание Executor */
 auto executor =
std::make_shared<oatpp::postgresql::Executor>(connectionPool);
 /* Создание репозитория */
 return std::make_shared<FeedbackDb>(executor);
}());
```

5. Использование репозиториев на уровне бизнес-логики. С помощью механизма внедрения зависимостей oatpp, экземпляры классов репозитория добавляются в нужные классы.

   Внедрение репозитория в сервис:
   OATPP_COMPONENT(std::shared_ptr<FeedbackDb>, m_database); // Внедрение зависимости

   В сервисе возможно исполнение запросов в рамках одной транзакции.

   Исполнение запросов в транзакции:

```
{
  auto transaction = m_database.beginTransaction();
  m_database.updateFeedback(feedback1, transaction.getConnection());
  m_database.updateFeedback(feedback2, transaction.getConnection());
  transaction.commit();
}
```

6. Запуск приложения. В корне проекта выполняем команды:

   mkdir build
   cd build
   cmake ..
   make install