Oatpp

Out++ - это современный веб-фреймворк для C++. Он содержит все необходимые компоненты для эффективной и быстрой разработки. Кроме того, он легкий и занимает мало места в памяти.

Основные компоненты:

- oatpp-server: Фреймворк обработки HTTP-запросов, поддерживающий различные форматы данных.
- oatpp-client: Библиотека для создания HTTP-клиентов.
- oatpp-websocket: Библиотека для реализации WebSocket-соединений.
- oatpp-postgres: Библиотека для работы с базами данных PostgreSQL.
- oatpp-mongodb: Библиотека для работы с базами данных MongoDB.
- oatpp-component: Библиотека для создания взаимозаменяемых компонентов приложения.

Oatpp можно использовать для создания широкого спектра веб-приложений, включая:

- RESTful API
- Веб-приложения с графическим пользовательским интерфейсом
- Разработки микроархитектуры
- Обработка событий в реальном времени
- Мобильные и встроенные приложения

Инструкция по запуску и установке oatpp.

1. В первую очередь требуется скачать Cmake https://cmake.org/
2. Далее переходим в VS клонируем репозиторий по ссылке: https://github.com/oatpp/oatpp
3. После того, как репозиторий склонирован, требуется выбрать нужную версию по тегам в ветке мастера и перейти на нее
4. Инструкция написана для версии 1.3.0!
5. Следующим шагом выполняем команды:
6. cd oatpp
7. mkdir build
8. cd build
9. cmake -DCMAKE_BUILD_TYPE=Debug -DOATPP_BUILD_TESTS=OFF -DOATPP_SQLITE_AMALGAMATION=ON ..
10. cmake --build . --target install

Для работы приложения также трубуется oatpp-swagger

1. Переходим в VS клонируем репозиторий по ссылке: https://github.com/oatpp/oatpp-swagger
2. После того, как репозиторий склонирован, требуется выбрать нужную версию по тегам в ветке мастера и перейти на нее
3. Ищем тэг версии с master, она должна совпадать с oatpp
4. Следующим шагом выполняем команды:
5. cd oatpp
6. mkdir build
7. cd build
8. cmake -DCMAKE_BUILD_TYPE=Debug -DOATPP_BUILD_TESTS=OFF -DOATPP_SQLITE_AMALGAMATION=ON ..
9. cmake --build . --target install

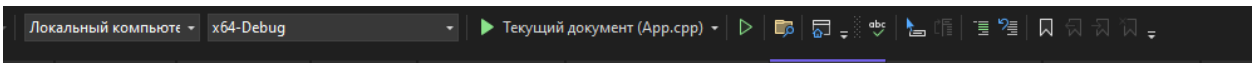
Также установим oatpp-postgres

1. Переходим в VS клонируем репозиторий по ссылке: https://github.com/oatpp/oatpp-postgresql
2. После того, как репозиторий склонирован, требуется выбрать нужную версию по тегам в ветке мастера и перейти на нее
3. Ищем тэг версии с master, она должна совпадать с oatpp
4. Следующим шагом выполняем команды:
5. cd oatpp
6. mkdir build
7. cd build
8. cmake -DCMAKE_BUILD_TYPE=Debug -DOATPP_BUILD_TESTS=OFF -DOATPP_SQLITE_AMALGAMATION=ON ..
9. cmake --build . --target install

После этого

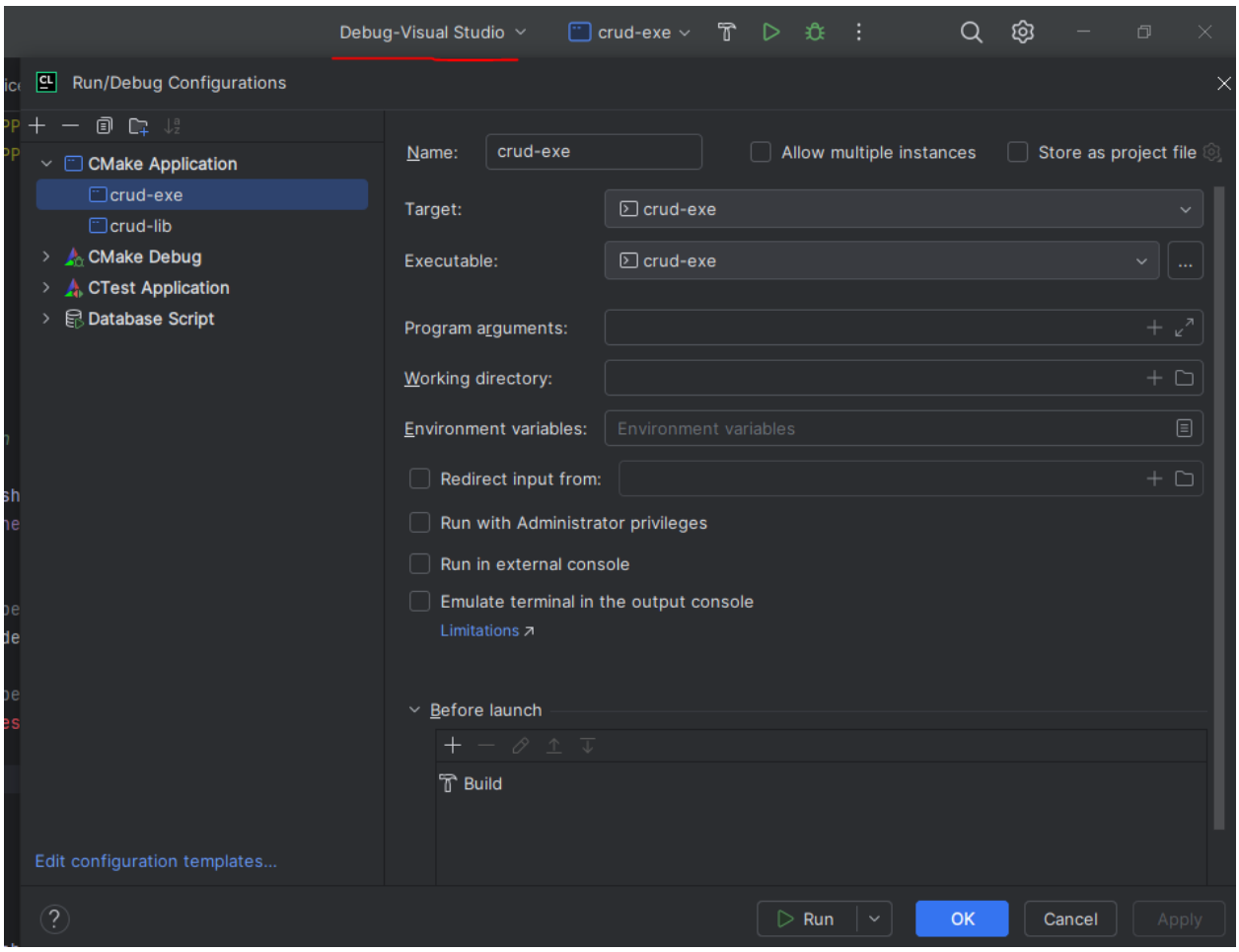
1. cd review-service
2. mkdir build
3. cd build
4. cmake ..
5. cmake --build .

Для запуска проекта в VS используем. Переходим в файл App



Запускаем текущий документ.

В Clion используем следующие настройки:



Приложение запускается на указанном вами порту. В моем случае 8080. **При добавлении нового компонента, нужно прописать путь до него по аналогии в cmakeList.txt.**

Если добавляется новый контроллер, прописываем его в App.

Структура контроллера

Контроллер Oat++ - это класс, наследующий от oatpp::web::server::api::ApiController. Он должен реализовывать по крайней мере один метод обработки запроса, который принимает объект oatpp::web::protocol::http::incoming::Request IncomingRequest и возвращает объект oatpp::web::protocol::http::outgoing::Response OutgoingResponse

Вернуть json body довольно просто, достаточно вызвать oatpp/web/server/api/ApiController/createDtoResponse(status, body);

```
62      ENDPOINT("GET", "review/{reviewId}", getReviewById,
63              PATH(Int32, reviewId))
64      {
65          return createDtoResponse(Status::CODE_200, dto: m_reviewService.getReviewById(reviewId));
66      }
```

Также контроллер может реализовывать другие методы для обработки различных типов HTTP-запросов (например, GET, POST, PUT, DELETE), а также методы для обработки различных путей запросов.

Регистрация контроллеров

Контроллеры регистрируются в приложении Oat++ с помощью функции oatpp::web::endpoint::RegisterController. Эта функция принимает экземпляр объекта oatpp::web::server::api::ApiController и строковый идентификатор контроллера.

```
/* create ApiControllers and add endpoints to router */

auto router :shared_ptr<HttpRouter> = serviceComponent.httpRouter.getObject();
oatpp::web::server::api::Endpoints docEndpoints;

docEndpoints.append(router->addController(ReviewController::createShared())->getEndpoints());
docEndpoints.append(router->addController(ReviewMarkController::createShared())->getEndpoints());

router->addController(oatpp::swagger::Controller::createShared(docEndpoints));
```

Async controller

```
ENDPOINT_INFO(CreateUser) {
  info->summary = "Create new User";
  info->addConsumes<Object<UserDto>>("application/json");
  info->addResponse<Object<UserDto>>(Status::CODE_200, "application/json");
}
ENDPOINT_ASYNC("POST", "demo/api/users", CreateUser) {

  ENDPOINT_ASYNC_INIT(CreateUser)

  Action act() override {
    return request->readBodyToDtoAsync<oatpp::Object<UserDto>>(
        controller->getDefaultObjectMapper()
    ).callbackTo(&CreateUser::returnResponse);
  }

  Action returnResponse(const oatpp::Object<UserDto>& body){
    return _return(createDtoResponse(Status::CODE_200, m_database->createUser(userDto)));
  }

};
```

Регистрация в сваггер

```
    ENDPOINT_INFO(getReviewById)
    {
        info->summary = "Get feedback by id";

        info->addResponse<Object<ReviewDto>>(Status::CODE_200, contentType: "application/json");
        info->addResponse<Object<StatusDto>>(Status::CODE_404, contentType: "application/json");
        info->addResponse<Object<StatusDto>>(Status::CODE_500, contentType: "application/json");

        info->pathParams["reviewId"].description = "Review Identifier";
    }

    ENDPOINT("GET", "review/{reviewId}", getReviewById,
             PATH(Int32, reviewId))
    {
        return createDtoResponse(Status::CODE_200, dto: m_reviewService.getReviewById(reviewId));
    }

```

Добавляем описание эндпоинте через ENDPOINT_INFO. Обращаю ваше внимание, красным подчеркнуты имена функции, они должны совпадать! Также нужно описать все возможные варианты ответа для сваггера и те модели, которые будут в качестве него.

Инжект

Также, стоит сказать, что oattp умеет инжектить. Для этого вам надо создать компонент и просто в блоке private его добавить.

```
class ReviewService {
private:
    typedef oatpp::web::protocol::http::Status Status;
    OATPP_COMPONENT(std::shared_ptr<ReviewDb>, m_database); // Inject database component
    OATPP_COMPONENT(std::shared_ptr<ReviewBanDb>, m_database_ban);
public:
```

Структура проекта РЕКОМЕНДУЕМАЯ oatpp.

- controller/ // Папка, содержащая контроллер, в котором объявлены все конечные точки

- db/ // Папка, содержащая клиент базы данных

- dto/ // здесь объявлены DTO

- service/ // Классы бизнес-логики сервиса

- ServiceComponent.hpp // Конфигурация сервиса (порт, ObjectMapper, база данных)

```
OATPP_CREATE_COMPONENT(std::shared_ptr<oatpp::data::mapping::ObjectMapper>, apiObjectMapper)( object: [] ->shared_ptr<ObjectMapper> {
    auto mapper :shared_ptr<ObjectMapper> = oatpp::parser::json::mapping::ObjectMapper::createShared();
    mapper->getSerializer()->getConfig()->useBeautifier = true;
    mapper->getSerializer()->getConfig()->escapeFlags = 0;
    return mapper;
    }());
```

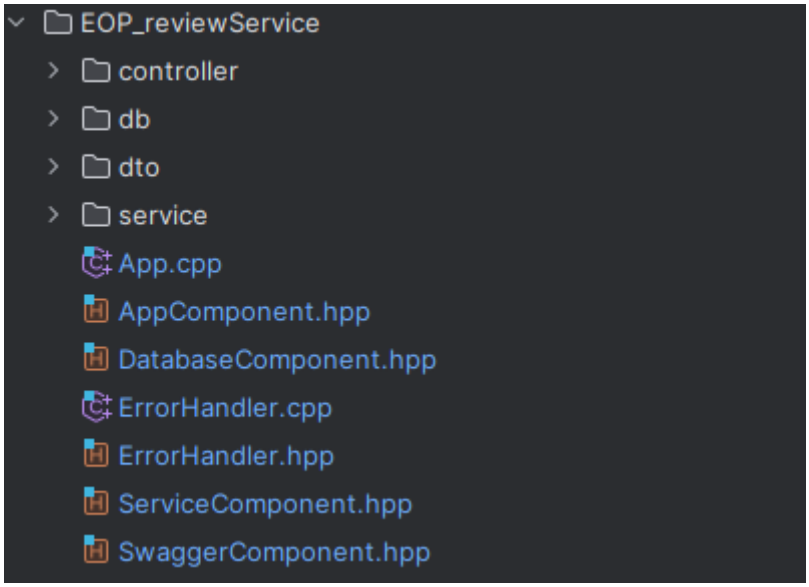
- SwaggerComponent.hpp // Конфигурация для swagger-ui

```
OATPP_CREATE_COMPONENT(std::shared_ptr<oatpp::swagger::DocumentInfo>, swaggerDocumentInfo)( object: [] ->shared_ptr<DocumentInfo> {
    OATPP_COMPONENT(oatpp::Object<ConfigDto>, config); // Get config component

    oatpp::swagger::DocumentInfo::Builder builder;

    builder
        .setTitle( "Review service")
        .setVersion( "1.0")
        .setContactUrl( "https://oatpp.io/")
        .addServer( url: "http://localhost:" + oatpp::utils::conversion::int32ToStr( config->port), description: "server on localhost");

    return builder.build();
```

- AppComponent.hpp // Здесь загружена конфигурация сервиса

- DatabaseComponent.hpp // Конфигурация базы данных

- ErrorHandler // тут происходит обработка ошибок



Если добавляется новый файл базы данных, добавляем миграцию, добавляем создание компонента в DatabaseComponent.

Полная документация по адресу

https://oatpp.io/docs/start/