

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 6

Выполнил: Зайцев Кирилл
Дмитриевич

Группа К33402

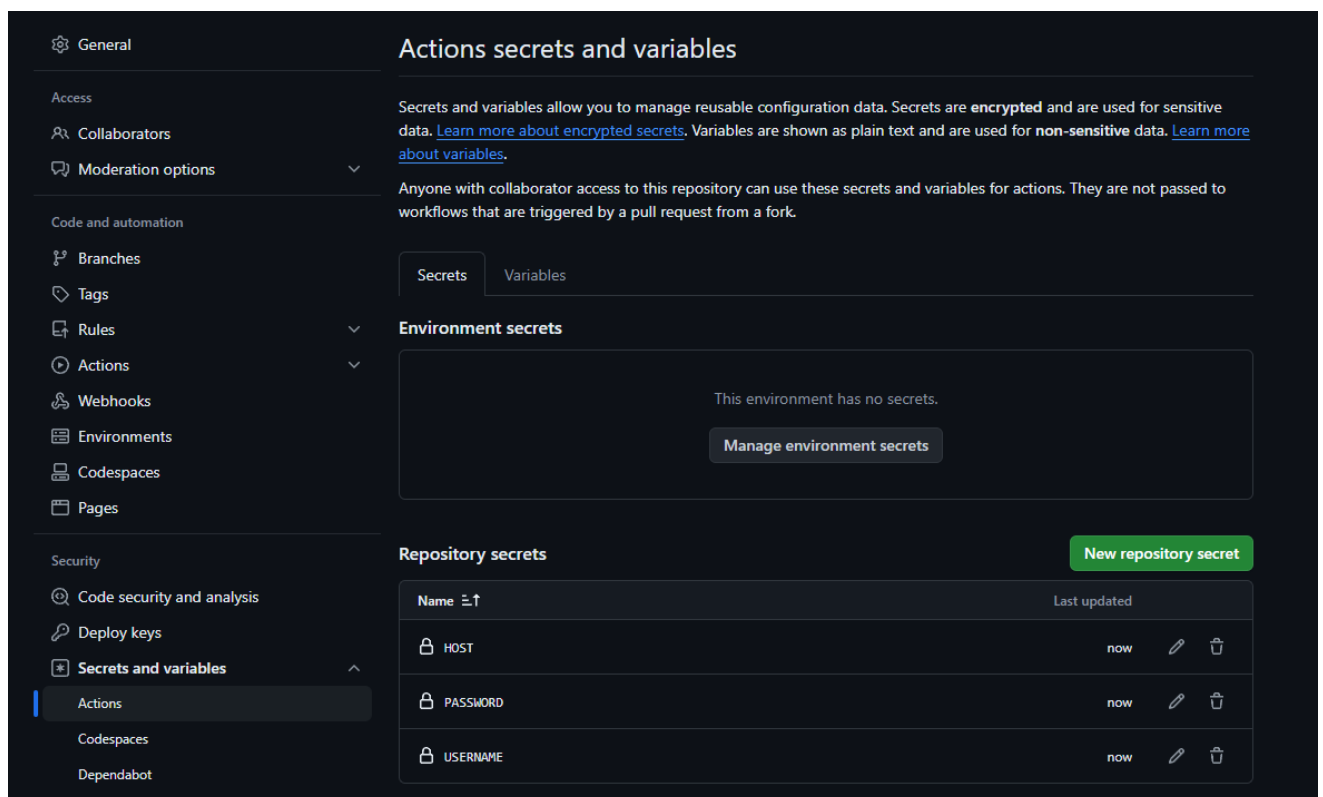
Проверил:
Добряков Д. И.

Санкт-Петербург

Задача:

Необходимо настроить автодеплой (с триггером на обновление кода в вашем репозитории, на определенной ветке) для вашего приложения на удалённый сервер с использованием Github Actions или Gitlab CI (любая другая CI-система также может быть использована).

Ход работы 1. Добавим секреты на Github



2. Создадим файл `deploy.yaml` по пути `.github/workflows/`. В нем укажем, что наша автоматизация должна запускаться при пуше в ветку `lr4`. В шагах мы просто получаем последний код, подключаемся к серверу, делаем `git pull` и перезапускаем `docker compose`.

`deploy.yaml`:

```
name: Deploy to the server

on:
  push:
    branches:
      - lr4
```

```
jobs:
  deploy_dashboard:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Update code
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${ secrets.HOST }
          username: ${ secrets.USERNAME }
          password: ${ secrets.PASSWORD }
          script: cd /root/ITMO-ICT-Backend-2024 && git checkout lr4 && git pull &&
cd labs/K33402/Зайцев_Кирилл && docker compose up --build -d
```

3. В результате имеем успешное обновление кода:

deploy_dashboard
succeeded now in 39s

Search logs

>	✔ Set up job	1s
>	✔ Build appleboy/ssh-action@v1.0.3	2s
>	✔ Checkout code	2s
>	✔ Update code	33s
>	✔ Post Checkout code	0s
>	✔ Complete job	0s

Вывод

В этой домашней работе была настроена автоматизация деплоя на удаленный сервер с помощью Github Actions. Процесс включает:

Настройка Github Actions для автоматического деплоя при пуше.

Установка зависимостей из requirements.txt.

Сборка и тестирование приложения.

SSH-подключение к серверу с использованием секретов Github.

Копирование файлов и запуск приложения на сервере.

Настройка уведомлений о статусе деплоя.

Преимущества:

Быстро: Деплой происходит автоматически.

Надежно: Минимизация ошибок.

Повторяемо: Процесс задокументирован и воспроизводим.

Таким образом, автоматизация с Github Actions упрощает и ускоряет деплой на сервер.

1. Инициализируем модуль

```
npm init
```

2. Установим зависимости

```
npm i dotenv express sequelize-typescript sqlite3 @types/express
```

3. В файле package.json укажем type: modul

```
1  {
2    "name": "laba1",
3    "version": "1.0.0",
4    "description": "backend_lab_1",
5    "main": "index.js",
6    "type": "module",
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1",
9      "build": "tsc",
10     "start": "tsc && node dist/index.js"
11   },
12   "author": "Kirill Zaitsev",
13   "license": "ISC",
14   "devDependencies": {
15     "typescript": "^5.4.3"
16   },
17   "dependencies": {
18     "@types/express": "^4.17.21",
19     "dotenv": "^16.4.5",
20     "express": "^4.19.2",
21     "sequelize-typescript": "^2.1.6",
22     "sqlite3": "^5.1.7"
23   }
24 }
25
```

4. Создадим файл tsconfig.json:

```
1  {
2    "compilerOptions": {
3      "module": "NodeNext",
4      "moduleResolution": "NodeNext",
5      "target": "ES2020",
6      "sourceMap": true,
7      "outDir": "dist",
8      "experimentalDecorators": true,
9      "emitDecoratorMetadata": true
10   },
11   "include": ["src/**/*"]
12 }
```

5. Общая структура проекта:

```
src
├── controllers
│   └── pool_cards
│       └── index.ts
├── errors
│   └── pool_cards
│       └── index.ts
├── models
│   └── pool_cards
│       └── index.ts
├── providers
│   └── db.ts
├── routes
│   └── pool_cards
│       └── index.ts
├── services
│   └── pool_cards
│       └── index.ts
├── .env
└── index.ts
```

Примеры выполнения запросов в postman:

GET:

The screenshot shows a Postman interface for a GET request to `http://localhost:3000/cards/`. The request body is a JSON object with an `identifier` of 4. The response is a 200 OK status with a 34 ms response time and 367 B of data. The response body is displayed in a pretty JSON format.

```
1 {
2   "identifier": 4
3 }
```

Body Cookies Headers (7) Test Results 200 OK 34 ms 367 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "identifier": 4,
3   "name": "Vasya Vasin",
4   "amount_left": 9,
5   "createdAt": "2024-04-03T15:04:02.176Z",
6   "updatedAt": "2024-04-03T15:04:15.054Z"
7 }
```

PUT(Обновление оставшегося числа посещений):

The screenshot shows a Postman interface for a PUT request to `http://localhost:3000/cards/`. The request body is a JSON object with an `identifier` of 4. The response is a 200 OK status with a 16 ms response time and 298 B of data. The response body is displayed in a pretty JSON format.

```
1 {
2   "identifier": 4
3 }
```

Body Cookies Headers (7) Test Results 200 OK 16 ms 298 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Remaining number of visits decreased successfully"
3 }
```

Проверка обновления данных:

HTTP <http://localhost:3000/cards/> Save Send

GET <http://localhost:3000/cards/> Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ☐

```
1 {
2   "identifier": "4"
3 }
```

Body Cookies Headers (7) Test Results 200 OK 6 ms 367 B Save as example

Pretty Raw Preview Visualize JSON ☐

```
1 {
2   "identifier": 4,
3   "name": "Vasya Vasin",
4   "amount_left": 8,
5   "createdAt": "2024-04-03T15:04:02.176Z",
6   "updatedAt": "2024-04-03T16:00:50.098Z"
7 }
```

POST:

Globals GET http://127.0.0.1:3000/ GET http://127.0.0.1:3000/ GET http://127.0.0.1:3000/ GET http://127.0.0.1:3000/ POST http://localhost:3000/cards/ + No environment Send

HTTP <http://localhost:3000/cards/> Save Send

POST <http://localhost:3000/cards/> Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ☐

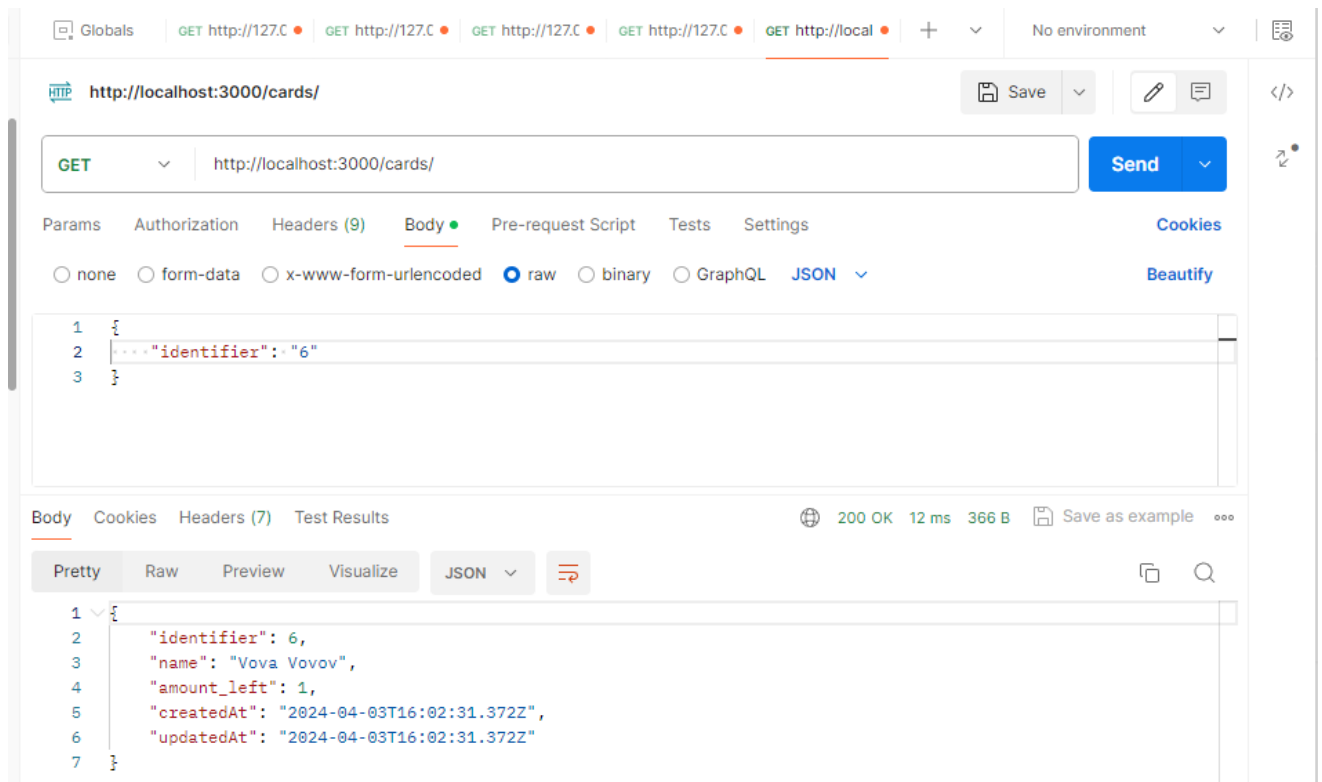
```
1 {
2   "identifier": "6",
3   "name": "Vova Vovov",
4   "amount_left": "1"
5 }
```

Body Cookies Headers (7) Test Results 200 OK 18 ms 235 B Save as example

Pretty Raw Preview Visualize JSON ☐

```
1 {}
```


POST(проверка):



Создание Makefile:

```
.PHONY: serve build

# Цель для установки зависимостей
install:
    npm install

# Цель для сборки приложения
build:
    npm run build

# Цель для запуска приложения
start:
    npm start

# Цель для проверки кода на наличие ошибок и стилистических проблем
lint:
    npm run lint

# Цель для запуска тестов
test:
    npm test

# Цель для очистки собранных файлов
clean:
    rm -rf dist

# Цель по умолчанию
.DEFAULT_GOAL := start
```

Вывод:

В данной лабораторной работе удалось написать boilerplate проект с использованием typescript + sequelize + express. Полученный проект можно использовать для создания следующих проектов с таким же стеком. Также удалось создать Makefile для автоматизации рутинных действий.