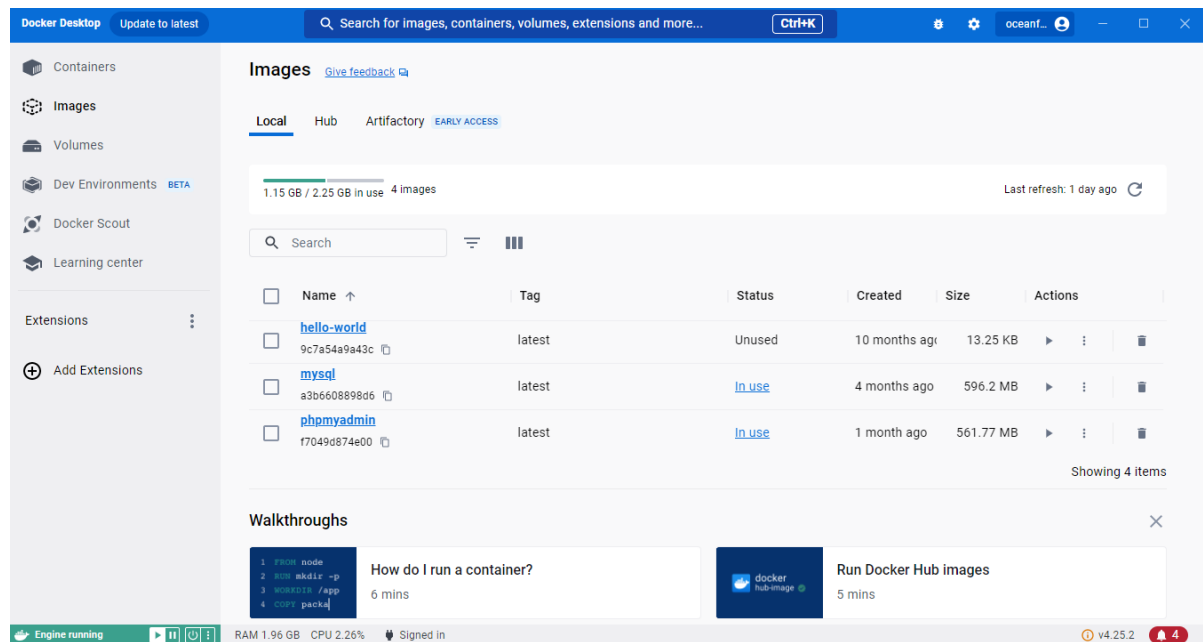


Отчёт по заданию на разработку API сервера по созданию пользователя в БД.

По условиям задачи стенд будет состоять из базы данных MySQL, развёрнутой в контейнере Docker, веб-сервера на nodeJS – запускаемый непосредственно на хосте win10, там где и ведётся разработка в VSCode. В рабочей папке проекта будет запущен локальный репозиторий Git с обращением в хостинг Git-Hub.

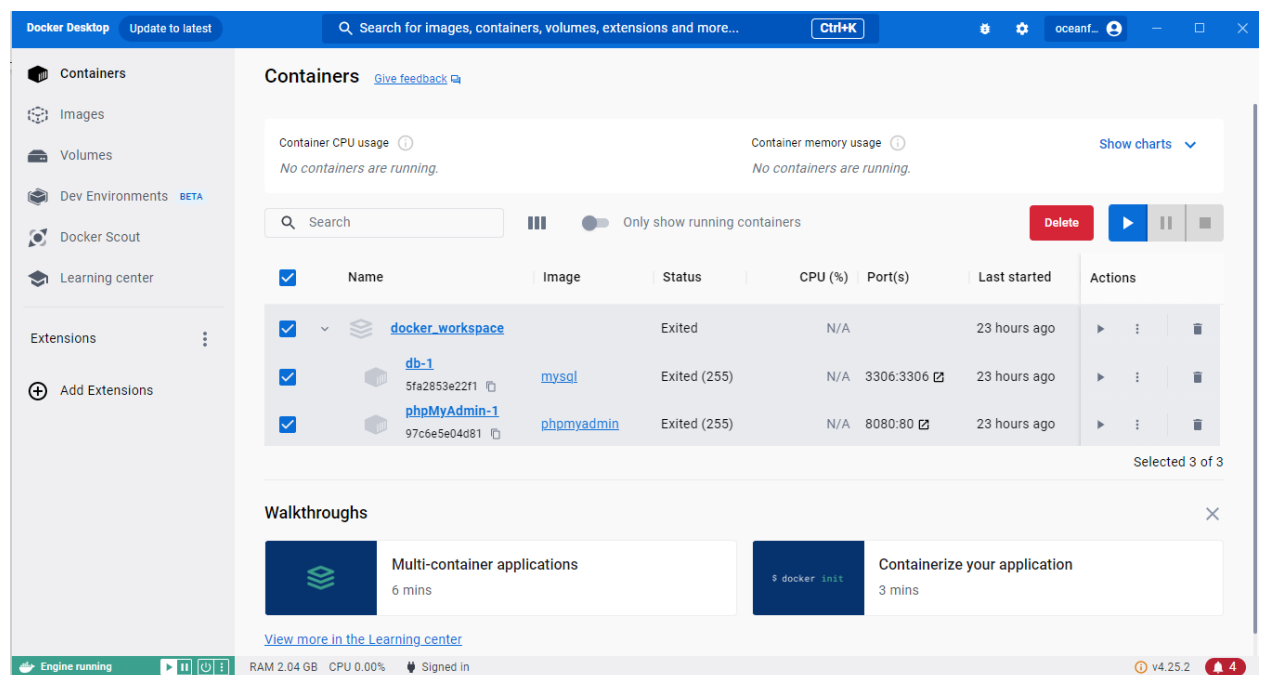
Первым пунктом лабораторной работы будет запуск сервера базы данных в контейнере Docker.

Так-как мы работаем на windows 10, используем ПО Docker-desktop.



Загрузим образы из Docker-Hub.

MySQL – сервер БД на Linux. PhpMyAdmin – веб-сервер для работы в качестве интерфейса (клиент) управления БД.



Для запуска составного контейнера воспользуемся Docker-compose.

Для этого создадим **yaml файл** с настройками для запуска.

```
services:
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
    ports:
      - "3306:3306"
  phpMyAdmin:
    image: phpmyadmin
    environment:
      PMA_ARBITRARY: 1
    ports:
      - "8080:80"
```

Именуем соответственно контейнеры.

Для контейнера db – укажем пароль для уз root (для этой уз выданы все права на все бд и таблицы), порты, которые будет слушать сервер.

Docker представляет собой подсистему со своими внутренними сетевыми интерфейсами.

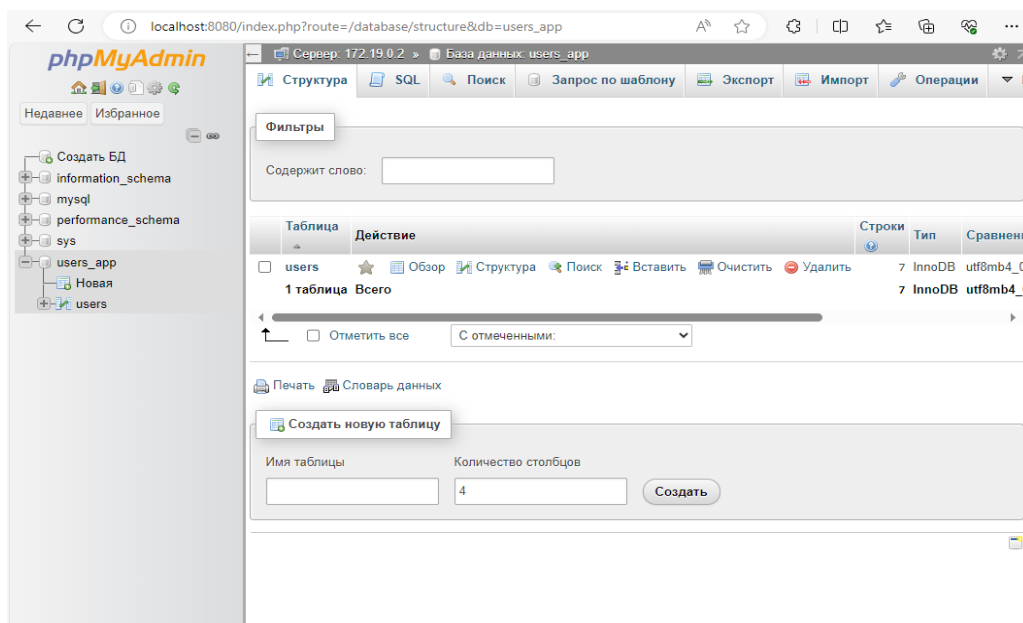
Атрибуты

ports:

- “3306:3306” - осуществляют публикацию на loopback интерфейс хоста (localhost ; 127.0.0.1)

по стандартному порту для MySQL. Далее запустим составной контейнер docker_workspace.

Проверим доступность БД, подключившись к ней с помощью phpMyAdmin



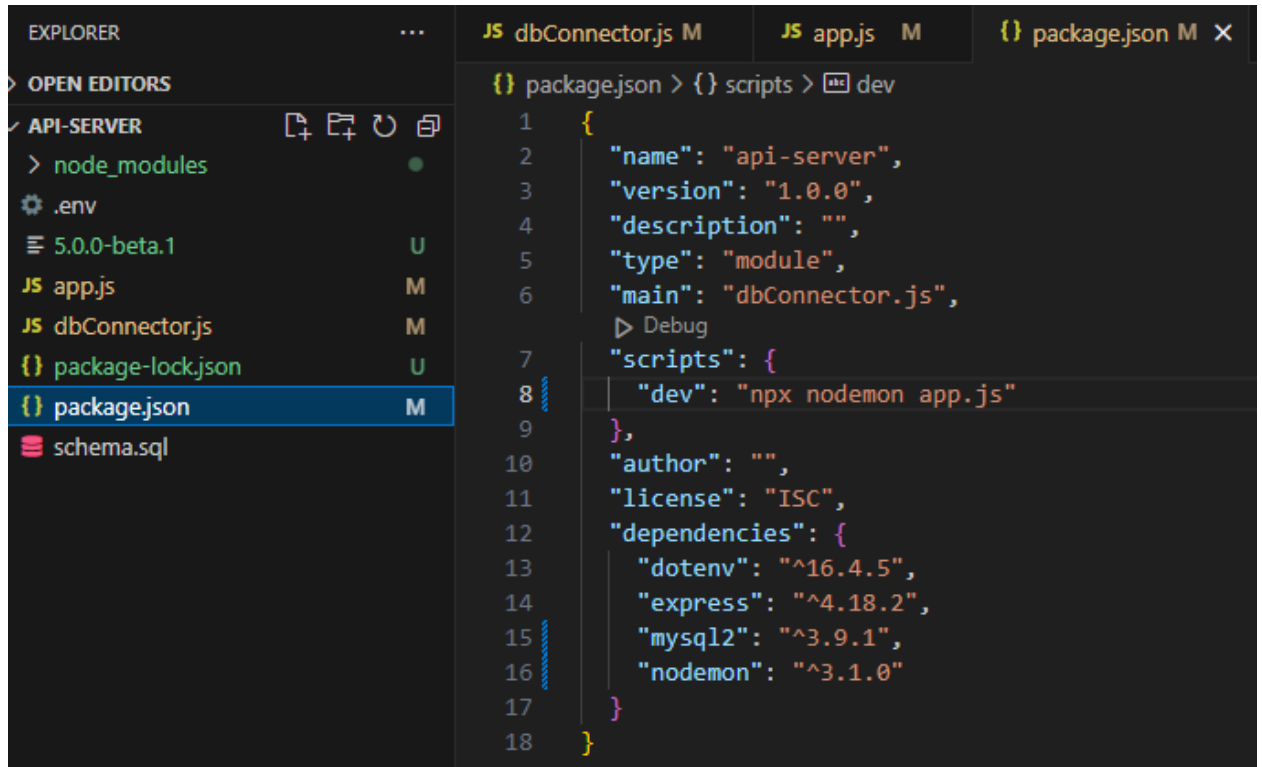
Далее приступим к созданию проекта *NodeJS*.

Создаём папку, где разместим все файлы – C:\test\Api-server

Первым напишем модуль для подключения и запросов к БД './dbConnector.js' [полный текст тут](#)

В начале каждого проекта необходимо создать файл package.json - содержание метаданных о проекте.

Наберём команду 'npm init -y'



```
{
  "name": "api-server",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "dbConnector.js",
  "scripts": {
    "dev": "npx nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.18.2",
    "mysql2": "^3.9.1",
    "nodemon": "^3.1.0"
  }
}
```

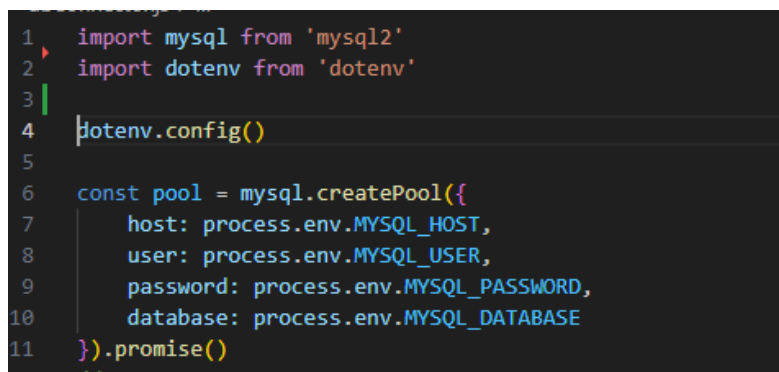
При открытии проекта на другой платформе с помощью package.json можно подтянуть все зависимости по модулям [npm i]. Элемент "type": "module" – позволяет использовать синтаксис

"Import from "

Нам понадобится модуль mysql2 - установим "npm i mysql2"

Для подключения создадим объект pool и наполним его методом .createPool

Сразу воспользуемся методом dotenv. Создадим файл .env и запишем в него данные для доступа к бд



```
1 import mysql from 'mysql2'
2 import dotenv from 'dotenv'
3
4 dotenv.config()
5
6 const pool = mysql.createPool({
7   host: process.env.MYSQL_HOST,
8   user: process.env.MYSQL_USER,
9   password: process.env.MYSQL_PASSWORD,
10  database: process.env.MYSQL_DATABASE
11 }).promise()
12
```

Данные не будут жёстко закреплены (hardcode) а будут подтягиваться. Это позволяет уменьшить риски утечки и легче использовать код для подключения к разным бд.

```
.env
1  MYSQL_HOST='localhost'
2  MYSQL_USER='root'
3  MYSQL_PASSWORD='password'
4  MYSQL_DATABASE='users_app'
5
```

Для начала работы с MySQL необходимо создать, собственно, базу с параметрами по заданию.

Создадим файл schema.sql

```
schema.sql
1  CREATE DATABASE users_app;
2
3  USE users_app;
4  CREATE TABLE users (
5      id integer PRIMARY KEY AUTO_INCREMENT,
6      user TEXT NOT NULL,
7      passwordd TEXT NOT NULL
8  );
9
10 INSERT INTO users (user, passwordd)
11 VALUES
12 ('admin1', 'admin'),
13 ('admin2', 'admin');
14
```

Создаём базу 'users_app', в ней таблицу 'users' с полями:

Ключ id (с увеличением на единицу), user – имя пользователя(тип данных текст, не может быть без значения), passwordd – Пароль(те же параметры).

Создадим два первых элемента.

```
export async function getUsers() {
  const [rows] = await pool.query("SELECT * FROM users")
  return rows
  //console.log(rows)
}

export async function getUser(id) {
  const [rows] = await pool.query(`
  SELECT *
  FROM users
  WHERE id = ?
  `, [id])
  return rows[0]
}
```

Далее создадим две сходных асинхронных функций для запросов к бд.

getUsers() – выводит все записи. Директива await возвращает результат от асинхронной операции при выполнении объекта .promise (от объекта pool).

getUser(id) – возвращает конкретного пользователя. В запросе используется подготовленное выражение в фильтре "WHERE id = ?" – тут переданный аргумент id верифицируется и считывается как второй элемент массива [rows] (предотвращение injection attacks).

Выполним общий запрос

```
const users = await getUsers()  
console.log(users)
```

получим:

```
C:\test\Api-server [master +3 ~3 -0 !]> node dbConnector.js  
[  
  { id: 1, user: 'admin1', passwordd: 'admin' },  
  { id: 2, user: 'admin2', passwordd: 'admin' },  
  { id: 3, user: 'admin3', passwordd: 'admin' },  
  { id: 4, user: 'admin4', passwordd: 'admin' },  
  { id: 5, user: 'admin5', passwordd: 'admin' },  
  { id: 6, user: 'admin6', passwordd: 'admin' },  
  { id: 7, user: 'admin7', passwordd: 'admin' }  
]
```

Далее следует функция создания пользователя `createUser(user, password)`.

```
30 export async function createUser(user, passwordd) {  
31   const [result] = await pool.query(`  
32     INSERT INTO users (user, passwordd)  
33     VALUES (?, ?)  
34   `, [user, passwordd])  
35   //return result.insertId  
36   const id = result.insertId  
37   return getUser(id)  
38 }  
39
```

Следует также подготовленное выражение. После выполнения записи в бд возвращается значение `id` созданного элемента и передаётся в функцию `getUser(id)`, которая возвращает все данные по текущему (созданному) пользователю, непосредственно из бд.

Далее создаём второй модуль – **app.js** – собственно веб сервер на express. [Полный текст тут](#)

Установим новую (на текущий момент) 5 версию express

```
npm install "express@>=5.0.0-beta.1" --save
```

```
JS app.js > ...  
1   import express from 'express'  
2   import { getUser, getUsers, createUser } from './dbConnector.js'  
3
```

Импортируем также функции из модуля `dbConnector.js`

Создаём экземпляр сервера и запускаем прослушивание на порту 3000

```
const app = express()
```

```
app.listen(3000, () => {  
  console.log('Server rabotaet na portu 3000')  
})
```

В модуле основным методом является создание пользователя.

```
27 app.post("/api/createUser" , async(req, res) => {  
28   const { user, passwordd} = req.body  
29   const userR = await createUser(user, passwordd )  
30   res.status(201).send(userR)  
31 })  
32
```

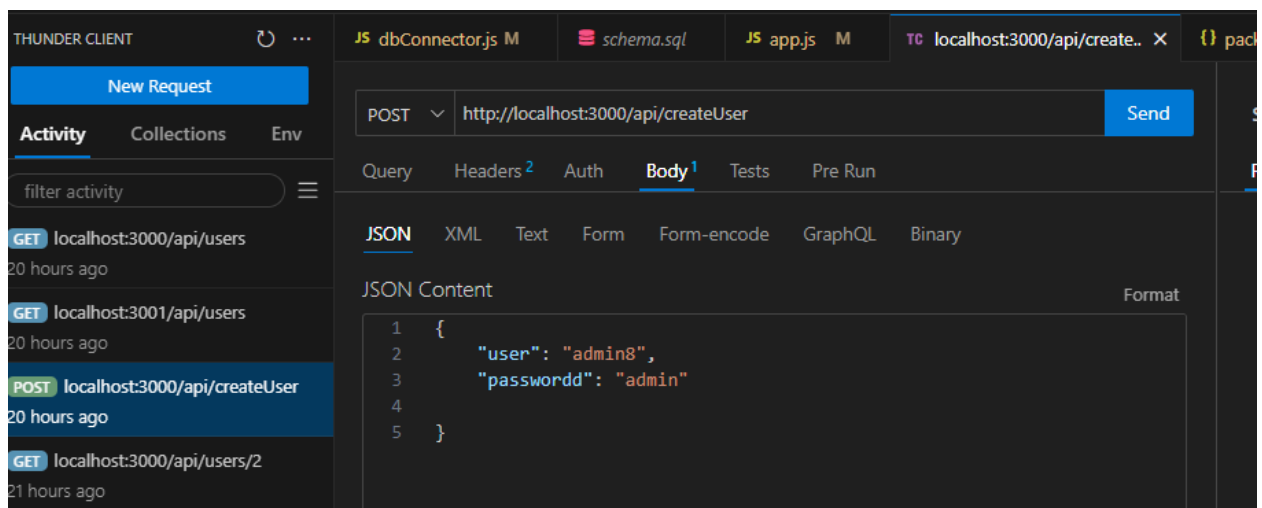
Запрос выполняется на эндпоинт '/api/createUser'. Значения полей считываются из тела json POST запроса.

Так же используем метод для парсинга значений.

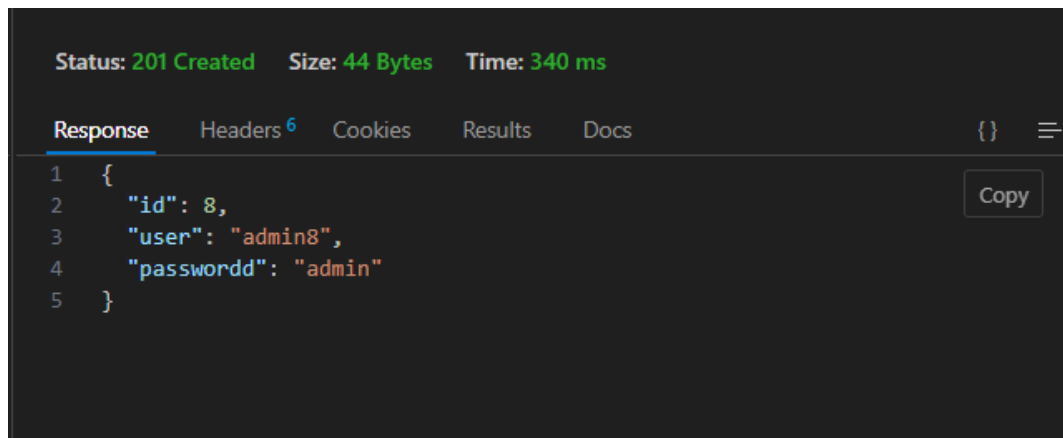
```
app.use(express.json())
```

Локальная переменная userR принимает значение функции createUser(user, passwordd) с аргументами, считанными из запроса. Далее методом res возвращается код ответа 201 со считанными из бд данными по вновь-созданному пользователю.

Далее используем расширение ThunderClient для VSCode, для отправки post запроса и проверки работы сервиса.



И получим ответ об успешном создании нового пользователя.



```
import mysql from 'mysql2'
import dotenv from 'dotenv'

dotenv.config()

const pool = mysql.createPool({
  host: process.env.MYSQL_HOST,
  user: process.env.MYSQL_USER,
  password: process.env.MYSQL_PASSWORD,
  database: process.env.MYSQL_DATABASE
}).promise()
//-----

export async function getUsers() {
  const [rows] = await pool.query("SELECT * FROM users")
  return rows
  //console.log(rows)
}

export async function getUser(id) {
  const [rows] = await pool.query(`
  SELECT *
  FROM users
  WHERE id = ?
  `, [id])
  return rows[0]
}

export async function createUser(user, passwordd) {
  const [result] = await pool.query(`
  INSERT INTO users (user, passwordd)
  VALUES (?, ?)
  `, [user, passwordd])
  //return result.insertId
  const id = result.insertId
  return getUser(id)
}
```

```
import express from 'express'
import { getUser, getUsers, createUser } from './dbConnector.js'

const app = express()

app.use(express.json()) // Парсинг json из тела запроса POST

app.get('/', (req, res) => { // Эндпоинт для показа главной страницы
  res.send('Dobro Pozalovat v nash Super Servis!!!');
})

app.get('/api/users', async (req, res) => { // Эндпоинт для вызова всех
  пользователей
    const users = await getUsers()
    res.send(users)
    // res.send("tut spisok userov")
  })

app.get('/api/users/:id', async (req, res) => { // Эндпоинт для вызова
  конкретного пользователя
    const id = req.params.id
    const user = await getUser(id)
    res.send(user)
    // res.send("tut spisok userov")
  })

app.post("/api/createUser" , async(req, res) => {
  const { user, passwordd } = req.body
  const userR = await createUser(user, passwordd )
  res.status(201).send(userR)
})

app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Chto-to polomalos!')
})

app.listen(3000, () => {
  console.log('Server rabotaet na portu 3000')
})
```