Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра автоматизованих систем управління



**Звіт**

до лабораторної роботи № 4

з дисципліни

**“Технології розподілених систем та паралельних обчислень ”**

на тему:

**Мікросервіси**

Виконав: студент КН-302

**Смаглюк Володимир**

Прийняла:

Процько І. О.

**Львів – 2023**

**Лабораторна робота №4**

**Мікросервіси**

**Мета роботи:** Оволодіти практичними прийомами розробки мікросервісів.

**Послідовність роботи**

1. Розбийте систему, розроблену вами у лабораторній роботі №3 на окремі мікросервіси.
2. Мікросервіси можна запускати локально на окремих портах, або використати Docker контейнери.
3. Для демонстрації роботи програми використайте будь-який http клієнт, наприклад, Postman
4. Оформити звіт про роботу за такою структурою:
   * назва роботи;
   * мета роботи;
   * послідовність роботи;
   * індивідуальне завдання;
   * текст програми;
   * скріншоти з http запитами та відповідями від сервісу;
   * висновки.

**Індивідуальне завдання**

Система Кавовий апарат. Користувач має Рахунок. Кавовий апарат містить набір Напоїв, із заданою кількістю порцій та додаткових Інгредієнтів. Користувач може купити один або кілька Напоїв. Адміністратор Кавового апарату здійснює його наповнення.

**Текст програми**

**adminController.js:**

import { getDrinks, updatePortions } from "./data.js";

const maxPortions = 50;

class Controller {

  async refill(req, res) {

    try {

      const { refillArray } = req.body;

      const drinks = await getDrinks();

      if (!Array.isArray(refillArray)) {

        return res.status(400).json({

          message: "Неправильний формат запиту",

        });

      }

      const updatedArray = [];

      refillArray.forEach((e) => {

        const { portions, drink } = e;

        if (!drink || !portions || portions < 0) {

          res.status(400).json({ message: "Помилка заповнення даних" }); //bad request

        }

        const item = drinks.find((e) => e.name === drink);

        item.portions += parseInt(portions);

        if (item.portions > maxPortions) {

          item.portions = maxPortions;

        }

        updatedArray.push(item);

      });

      updatedArray.forEach((e) => {

        updatePortions(e.name, e.portions);

      });

      res.json(updatedArray);

    } catch (e) {

      res.status(500).json(e);

    }

  }

}

export default new Controller();

**adminRouter.js:**

import Router from "express";

import Controller from "../controllers/controller.js";

const router = new Router();

router.put("/admin/refil", Controller.refill);

export default router;

export default router;

**adminService.js:**

import router from "./routers/router.js";

import express from "express";

const PORT = 3001;

const app = express();

app.use(express.json());

app.use(router);

app.listen(PORT, () => {

  console.log(`Admin Service started on port: ${PORT}`);

});

**adminDBData.js:**

import sqlite3 from "sqlite3";

const db = new sqlite3.Database("/db/mydatabase.db");

export async function getDrinks() {

  return new Promise((resolve, reject) => {

    db.all("SELECT \* FROM drinks", (err, rows) => {

      if (err) {

        reject(err);

      } else {

        const drinks = rows.map((row) => ({

          name: row.name,

          portions: row.portions,

          price: row.price,

          ingredients: JSON.parse(row.ingredients),

        }));

        resolve(drinks);

      }

    });

  });

}

export function updatePortions(name, newPortions) {

  db.run("UPDATE drinks SET portions = ? WHERE name = ?", [newPortions, name]);

}

**DockerFile:**

FROM node:16

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

CMD ["npm", "start"]

**CoffeeMachineController.js:**

import { getBalance, getDrinks } from "./data.js";

class Controller {

  async getDrinksList(\_, res) {

    try {

      const drinks = await getDrinks();

      console.log("Отримано список напоїв у автоматі");

      res.json(drinks);

    } catch (error) {

      console.error("Помилка при отриманні списку напоїв:", error);

      res.status(500).json({ error: "Помилка при отриманні списку напоїв" });

    }

  }

  async getBalance(\_, res) {

    try {

      const balance = await getBalance();

      console.log("Отримано баланс");

      res.json(balance);

    } catch (error) {

      console.error("Помилка при отриманні балансу:", error);

      res.status(500).json({ error: "Помилка при отриманні балансу" });

    }

  }

}

export default new Controller();

**CoffeeMachineRouter.js:**

import Router from "express";

import Controller from "../controllers/controller.js";

const router = new Router();

router.get("/coffee\_machine/drinks", Controller.getDrinksList);

router.get("/coffee\_machine/balance", Controller.getBalance);

export default router;

**CoffeMachineService.js:**

import router from "./routers/router.js";

import express from "express";

const PORT = 3002;

const app = express();

app.use(express.json());

app.use(router);

app.listen(PORT, () => {

  console.log(`CoffeeMachine Service started on port: ${PORT}`);

});

**CoffeMachineDBData.js:**

import sqlite3 from "sqlite3";

const db = new sqlite3.Database("/db/mydatabase.db");

export async function getBalance() {

  return new Promise((resolve, reject) => {

    db.get("SELECT \* FROM account", (err, row) => {

      if (err) {

        reject(err);

      } else {

        resolve(row.balance);

      }

    });

  });

}

export async function getDrinks() {

  return new Promise((resolve, reject) => {

    db.all("SELECT \* FROM drinks", (err, rows) => {

      if (err) {

        reject(err);

      } else {

        const drinks = rows.map((row) => ({

          name: row.name,

          portions: row.portions,

          price: row.price,

          ingredients: JSON.parse(row.ingredients),

        }));

        resolve(drinks);

      }

    });

  });

}

**userController.js:**

import {

  getBalance,

  getDrinks,

  updateAccountBalance,

  updatePortions,

} from "./data.js";

class Controller {

  async balanceTopUp(req, res) {

    try {

      let balance = await getBalance();

      const { money } = req.params;

      balance += parseInt(money);

      console.log(`Поповнення балансу на: ${money}`);

      updateAccountBalance(balance);

      res.json(balance);

    } catch (err) {

      return res.status(500).json(err);

    }

  }

  async buyDrinks(req, res) {

    try {

      const drinks = await getDrinks();

      let balance = await getBalance();

      const { orders } = req.body;

      if (!Array.isArray(orders)) {

        return res.status(400).json({

          message: "Неправильний формат запиту",

        });

      }

      let totalPrice = 0;

      let message = "";

      const drinksToUpdate = [];

      orders.forEach((order) => {

        const { drink, ingredients } = order;

        const item = drinks.find((e) => e.name === drink);

        totalPrice += item.price;

        let isSubset = ingredients.every((val) =>

          item.ingredients.includes(val)

        );

        if (!isSubset) {

          res.status(400).json({

            message: "Додано некоректні інградієнти",

          });

        }

        totalPrice += ingredients.length \* 4;

        message += ` приготовано ${drink} з ${ingredients}`;

        drinksToUpdate.push(item);

      });

      if (totalPrice > balance) {

        res.status(400).json({

          message: "Недостатньо балансу для приготування напоїв",

        });

      }

      balance -= totalPrice;

      message += ` за ${totalPrice} грн.`;

      updateAccountBalance(balance);

      drinksToUpdate.forEach((e) => {

        updatePortions(e.name, e.portions - 1);

      });

      res.json(message);

    } catch (e) {

      res.status(500).json(e);

    }

  }

}

export default new Controller();

**userRouter.js:**

import Router from "express";

import Controller from "../controllers/controller.js";

const router = new Router();

router.put("/user/topup/:money", Controller.balanceTopUp);

router.post("/user/buy", Controller.buyDrinks);

export default router;

**userService.js:**

import router from "./routers/router.js";

import express from "express";

const PORT = 3000;

const app = express();

app.use(express.json());

app.use(router);

app.listen(PORT, () => {

  console.log(`User Service started on port: ${PORT}`);

});

**userDBData.js:**

import sqlite3 from "sqlite3";

const db = new sqlite3.Database("/db/mydatabase.db");

export async function getBalance() {

  return new Promise((resolve, reject) => {

    db.get("SELECT \* FROM account", (err, row) => {

      if (err) {

        reject(err);

      } else {

        resolve(row.balance);

      }

    });

  });

}

export async function getDrinks() {

  return new Promise((resolve, reject) => {

    db.all("SELECT \* FROM drinks", (err, rows) => {

      if (err) {

        reject(err);

      } else {

        const drinks = rows.map((row) => ({

          name: row.name,

          portions: row.portions,

          price: row.price,

          ingredients: JSON.parse(row.ingredients),

        }));

        resolve(drinks);

      }

    });

  });

}

export function updateAccountBalance(newBalance) {

  db.run("UPDATE account SET balance = ?", newBalance);

}

export function updatePortions(name, newPortions) {

  db.run("UPDATE drinks SET portions = ? WHERE name = ?", [newPortions, name]);

}

**Docker-compose.yml:**

version: "3"

services:

  db:

    build: ./db

  user:

    build: ./user

    ports:

      - "3000:3000"

    networks:

      - app-network

    depends\_on:

      - db

    volumes:

      - ./db/mydatabase.db:/db/mydatabase.db

  admin:

    build: ./admin

    ports:

      - "3001:3001"

    networks:

      - app-network

    volumes:

      - ./db/mydatabase.db:/db/mydatabase.db

  machine:

    build: ./coffeeMachine

    ports:

      - "3002:3002"

    networks:

      - app-network

    volumes:

      - ./db/mydatabase.db:/db/mydatabase.db

networks:

  app-network:

    driver: bridge

# docker-compose up --build

dbInit.js:

import sqlite3 from "sqlite3";

// Підключення до бази даних

const db = new sqlite3.Database("mydatabase.db");

// Створення таблиці для рахунку користувача

db.serialize(() => {

  db.run("CREATE TABLE IF NOT EXISTS account (balance INTEGER)");

  db.get("SELECT COUNT(\*) as count FROM account", (err, row) => {

    if (err) {

      console.error(err.message);

    } else if (row.count === 0) {

      db.run("INSERT INTO account (balance) VALUES (100)");

    }

  });

});

const drinks = {

  coffee: {

    portions: 10,

    price: 20,

    ingredients: JSON.stringify(["milk", "sugar", "vanilla"]),

  },

  tea: {

    portions: 10,

    price: 18,

    ingredients: JSON.stringify(["milk", "sugar", "honey", "cinnamon"]),

  },

  latte: {

    portions: 10,

    price: 35,

    ingredients: JSON.stringify(["milk", "sugar", "vanilla"]),

  },

  cappuccino: {

    portions: 10,

    price: 37,

    ingredients: JSON.stringify([

      "milk",

      "sugar",

      "vanilla",

      "caramel",

      "chocolate",

    ]),

  },

  espresso: {

    portions: 10,

    price: 35,

    ingredients: JSON.stringify(["milk", "sugar"]),

  },

  americano: {

    portions: 10,

    price: 40,

    ingredients: JSON.stringify([

      "milk",

      "sugar",

      "vanilla",

      "almondMilk",

      "soyMilk",

    ]),

  },

  mocha: {

    portions: 10,

    price: 30,

    ingredients: JSON.stringify(["milk", "sugar", "vanilla"]),

  },

  macchiato: {

    portions: 10,

    price: 40,

    ingredients: JSON.stringify([

      "milk",

      "sugar",

      "vanilla",

      "almondMilk",

      "soyMilk",

    ]),

  },

  frappe: {

    portions: 10,

    price: 28,

    ingredients: JSON.stringify(["milk", "sugar", "vanilla", " chocolate"]),

  },

  hotChocolate: {

    portions: 10,

    price: 18,

    ingredients: JSON.stringify(["milk", "sugar", "vanilla"]),

  },

};

// Створення таблиці для напоїв

db.serialize(() => {

  db.run(

    "CREATE TABLE IF NOT EXISTS drinks (name TEXT PRIMARY KEY, portions INTEGER, price REAL, ingredients TEXT)"

  );

  db.get("SELECT COUNT(\*) as count FROM drinks", (err, row) => {

    if (err) {

      console.error(err.message);

    } else if (row.count === 0) {

      for (const name in drinks) {

        const { portions, price, ingredients } = drinks[name];

        db.run(

          `INSERT INTO drinks (name, portions, price, ingredients) VALUES (?, ?, ?, ?)`,

          [name, portions, price, ingredients]

        );

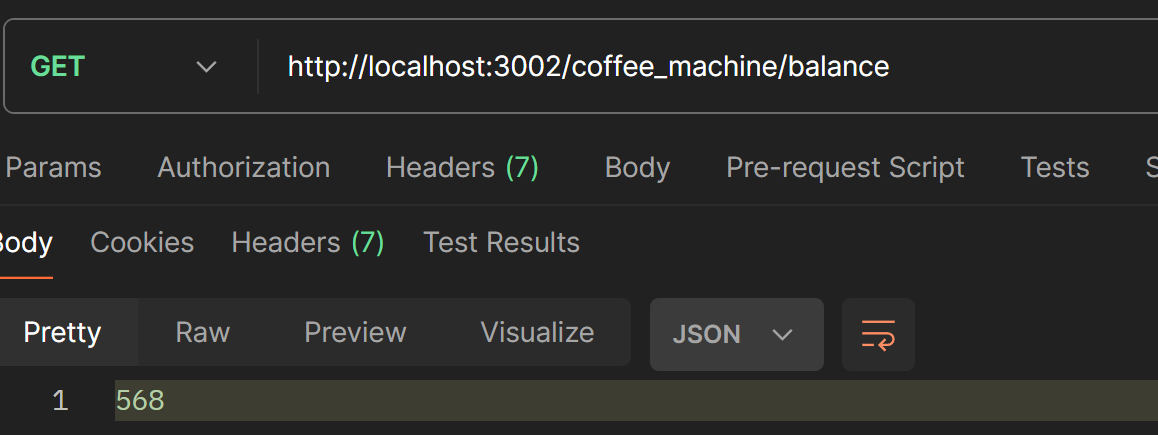
      }

    }

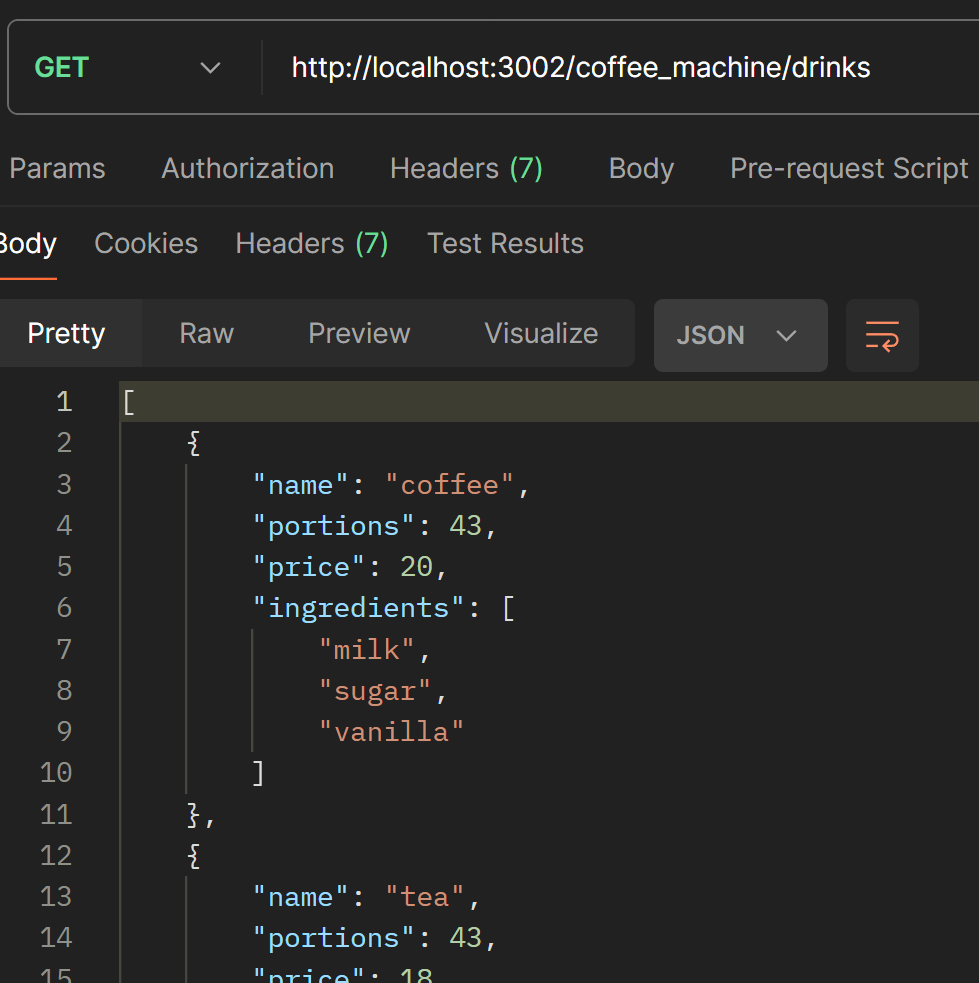
  });

});

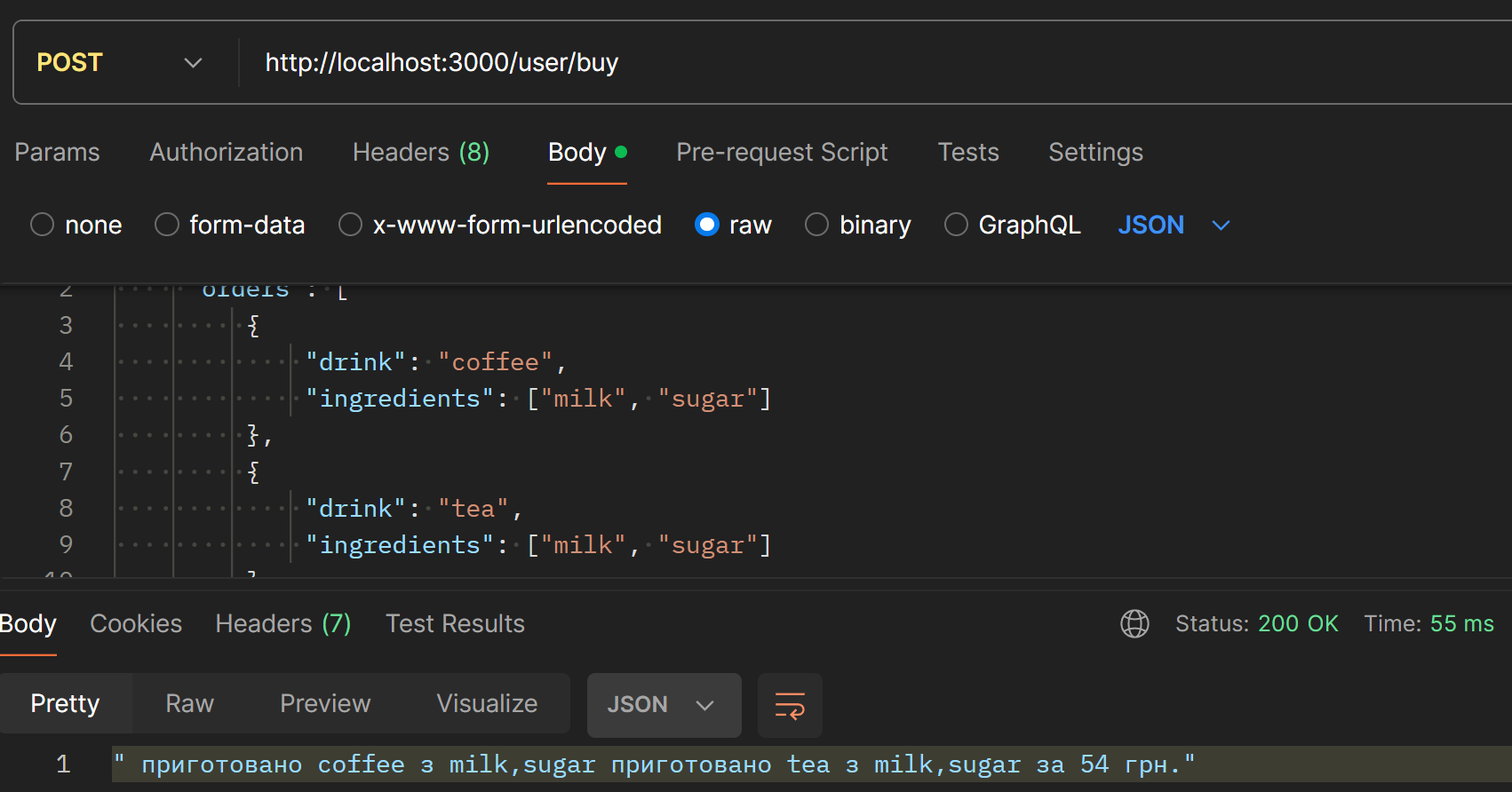
**Результати роботи програми**

****

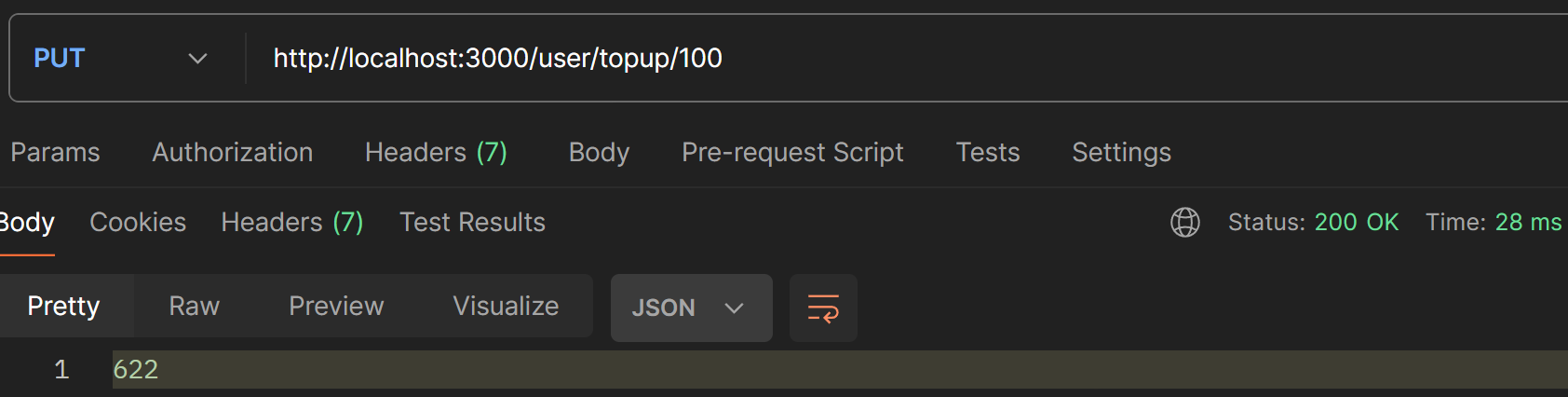
*Рис.1. Get запит балансу рахунку.*

****

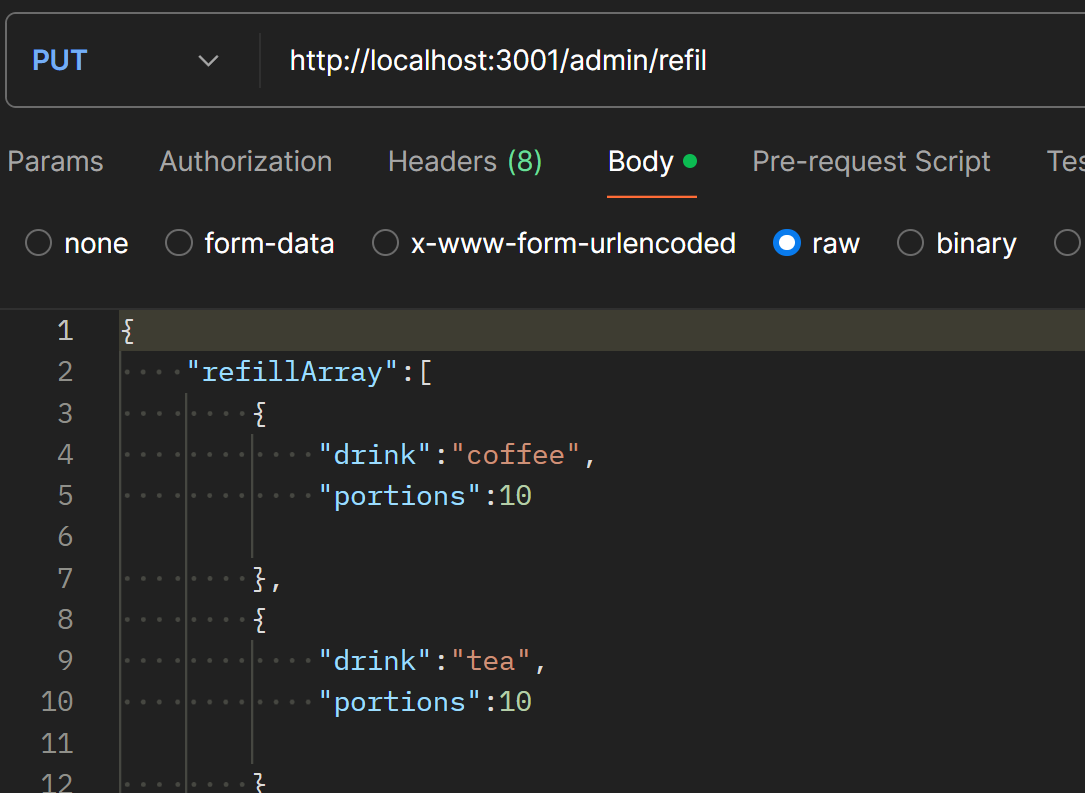
*Рис.2. Get запит списку напоїв.*

****

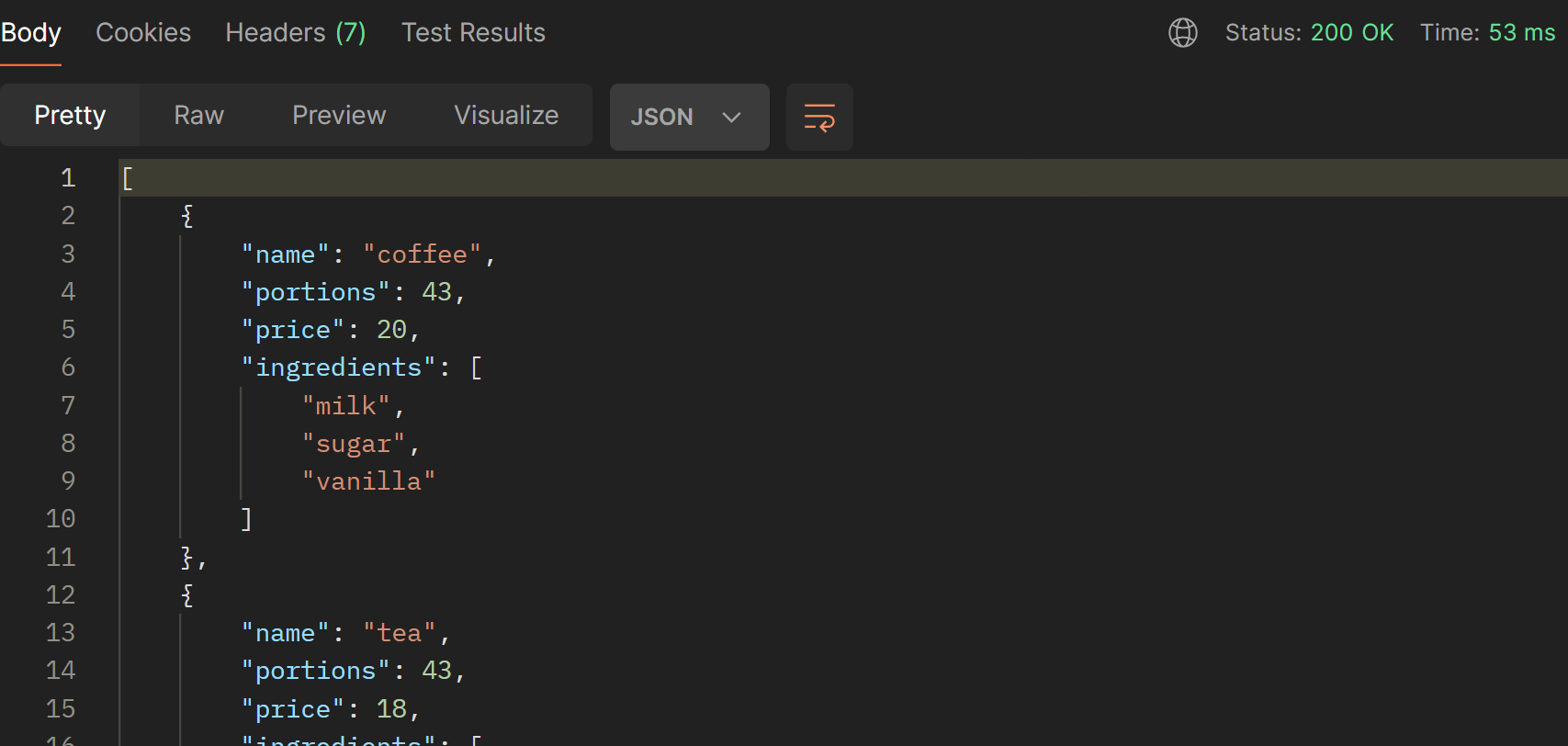
*Рис. 3. Post запит замовлення напоїв.*

****

*Рис.4. Put запит поповнення балансу рахунку.*

****

*Рис.5. Put запит наповнення автомату.*

****

*Рис.6. Результат наповнення автомату.*

**Висновок**

В результаті виконання програми, я оволодів практичними прийомами розробки RESTful веб сервісів, реалізував власний веб сервіс кавового автомату з напоями, порціями, додатковими інгредієнтами, реалізував повноцінний CRUD для них і протестував роботу сервісу через http клієнт Postman. Також розбив програму на декілька окремих сервісів, які я помістив в докер контейнери і які зібрав і запустив за допомогою докера.