МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Факультет №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 316 «Системное моделирование и автоматическое проектирование»

Вычислительные системы

Курсовая работа на тему

«Разработка микросервисной вычислительной системы заказов вычислительной техники»

Выполнил студент группы 3О\_117М Наштыков Д.Е.

Проверил доцент 806 каф., к.ф.м.н. Семенов Г.Е.

Москва 2023 г.

# **Задание**

Разработать вычислительную систему заказов вычислительной техники используя микросервисный подход.

Общая схема:

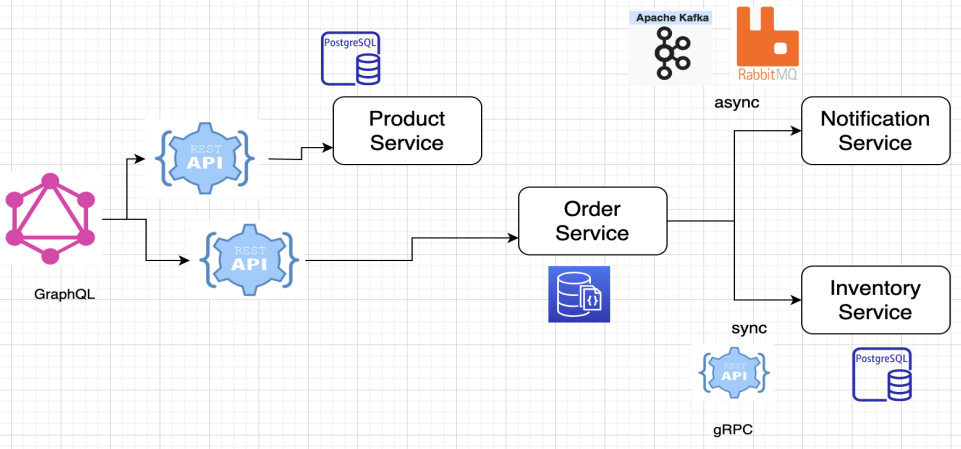


Рисунок 1

# Выполнение

## Проектирование баз данных

В ходе работы необходимо спроектировать базы данных для 3х сервисов: sql базы данных для сервиса продуктов и инвентаря, а также документную базу данных для сервиса заказов. В качестве первой была выбрана СУБД Posgtres, в качестве второго СУБД MongoDB.

ER диаграммы баз данных:

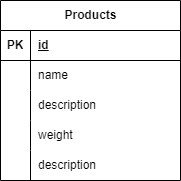


Рис.2 – ER диаграмма базы данных сервиса продуктов

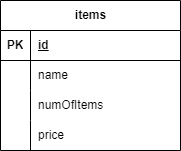


Рис.3 – ER диаграмма базы данных сервиса инвентаря

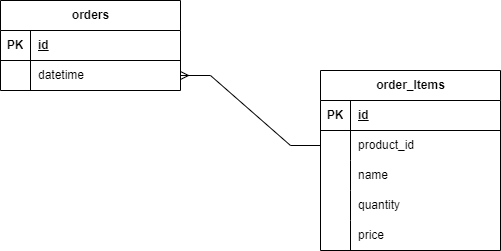


Рис.4 - ER диаграмма базы данных сервиса заказов

## Результат разработки

### Сервис продуктов

package main

import (

"database/sql"

"encoding/json"

"fmt"

"github.com/go-chi/chi/v5"

"github.com/go-chi/chi/v5/middleware"

\_ "github.com/lib/pq"

"net/http"

"os"

)

type Product struct {

Id int `json:"id"`

Name string `json:"name"`

Weight int `json:"weight"`

Description string `json:"description"`

}

func main() {

db, err := sql.Open("postgres", "host='db-products' sslmode=disable port=5432 user=products dbname='products' password='products'")

if err != nil {

fmt.Println("Cannot connect to db")

fmt.Println(err.Error())

return

}

defer db.Close()

\_, err = db.Exec("CREATE TABLE IF NOT EXISTS products (\n id serial PRIMARY KEY,\n name VARCHAR NOT NULL,\n weight NUMERIC NOT NULL,\n description VARCHAR NOT NULL\n)")

if err != nil {

fmt.Println(err.Error())

return

}

router := chi.NewRouter()

router.Use(middleware.Logger)

router.Get(os.Getenv("api\_url")+"/products", func(w http.ResponseWriter, r \*http.Request) {

rows, err := db.Query("SELECT \* FROM products")

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error fetching data from database"))

fmt.Println(err.Error())

return

}

defer rows.Close()

products := []Product{}

var pr Product

for rows.Next() {

err := rows.Scan(&pr.Id, &pr.Name, &pr.Weight, &pr.Description)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error fetching data from rows"))

fmt.Println(err.Error())

return

}

products = append(products, pr)

}

res, err := json.Marshal(products)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Server error"))

return

}

w.Write(res)

})

router.Post(os.Getenv("api\_url")+"/products", func(w http.ResponseWriter, r \*http.Request) {

var bodyTraslated Product

decoder := json.NewDecoder(r.Body)

err = decoder.Decode(&bodyTraslated)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error on parsing body"))

w.Write([]byte(err.Error()))

return

}

\_, err := db.Exec("INSERT INTO products (name, weight, description) VALUES ($1, $2, $3)", bodyTraslated.Name, bodyTraslated.Weight, bodyTraslated.Description)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error creating data from database\n"))

w.Write([]byte(err.Error()))

return

}

w.Write([]byte("ok"))

})

router.Patch(os.Getenv("api\_url")+"/products/{id}", func(w http.ResponseWriter, r \*http.Request) {

productId := chi.URLParam(r, "id")

var bodyTraslated Product

decoder := json.NewDecoder(r.Body)

err = decoder.Decode(&bodyTraslated)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error on parsing body"))

w.Write([]byte(err.Error()))

return

}

\_, err := db.Exec("UPDATE products SET id=$1, name=$2, weight=$3, description=$4 WHERE id=$5", bodyTraslated.Id, bodyTraslated.Name, bodyTraslated.Weight, bodyTraslated.Description, productId)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error creating data from database\n"))

w.Write([]byte(err.Error()))

return

}

w.Write([]byte("ok"))

})

router.Delete(os.Getenv("api\_url")+"/products/{id}", func(w http.ResponseWriter, r \*http.Request) {

productId := chi.URLParam(r, "id")

\_, err = db.Exec("DELETE FROM products WHERE id=$1", productId)

if err != nil {

w.WriteHeader(500)

w.Write([]byte("Error on deleting\n"))

w.Write([]byte(err.Error()))

}

w.Write([]byte("ok"))

})

err = http.ListenAndServe(":8080", router)

if err != nil {

fmt.Println(err.Error())

}

}

Листинг 1 – Сервис продуктов

### 