

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни

**«**Бази даних і засоби управління**»**

Виконав студент ІII курсу

ФПМ групи КВ-83

Алексеєв М.О.

Перевірив: Павловський В.І.

Київ – 2020

**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Завдання* роботи полягає у наступному:

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

*Функціональні вимоги*

1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

*Вимоги до інтерфейсу користувача*

1. Використовувати консольний інтерфейс користувача.

**Нормалізована логічна модель даних БД «Інтернет магазин»**

На рис. 2.1 наведено модель даних БД “Інтернет магазин”, яка була розроблена під час виконання лабораторної роботи №1.

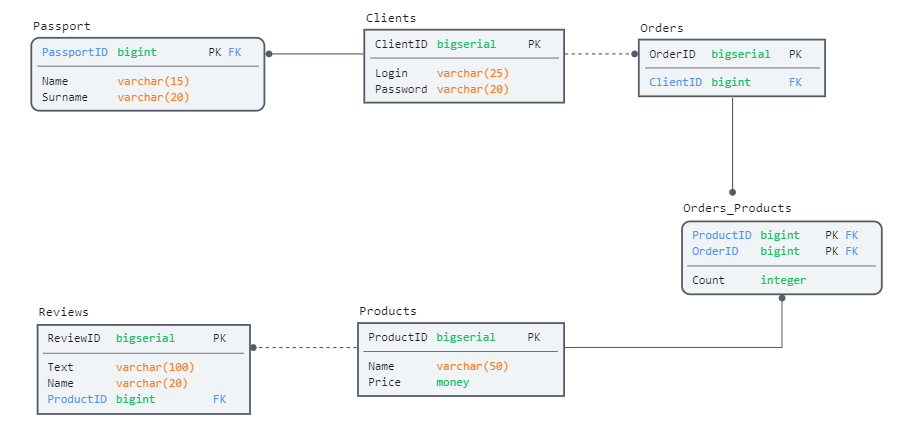


Рис 2.1 Нормалізована логічна модель даних БД «Інтернет магазин»

**Опис програми**

Для виконання цієї лабораторної роботи, я використовував середовище розробки VS Code. Для підключення бази даних використовувалась платформа Node.js та додатково бібліотека “PG”.

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 5 модулів:

1. index.js – файл, що є точкою входу до програми, запускає сервер, який приймає запити;
2. routers – папка містить файли обробки запитів до сервера для кожної з таблиць, підключає файл controllers.js для відповідної таблиці;
3. controllers – папка містить файли для обробки логіки отримання даних. Підключає файл з папки models для відповідної таблиці;
4. models – пака містить файли з класами, які відповідають за дані в таблицях БД. Для кожної таблиці створений свій клас;

**Структура меню програми**

Меню програми

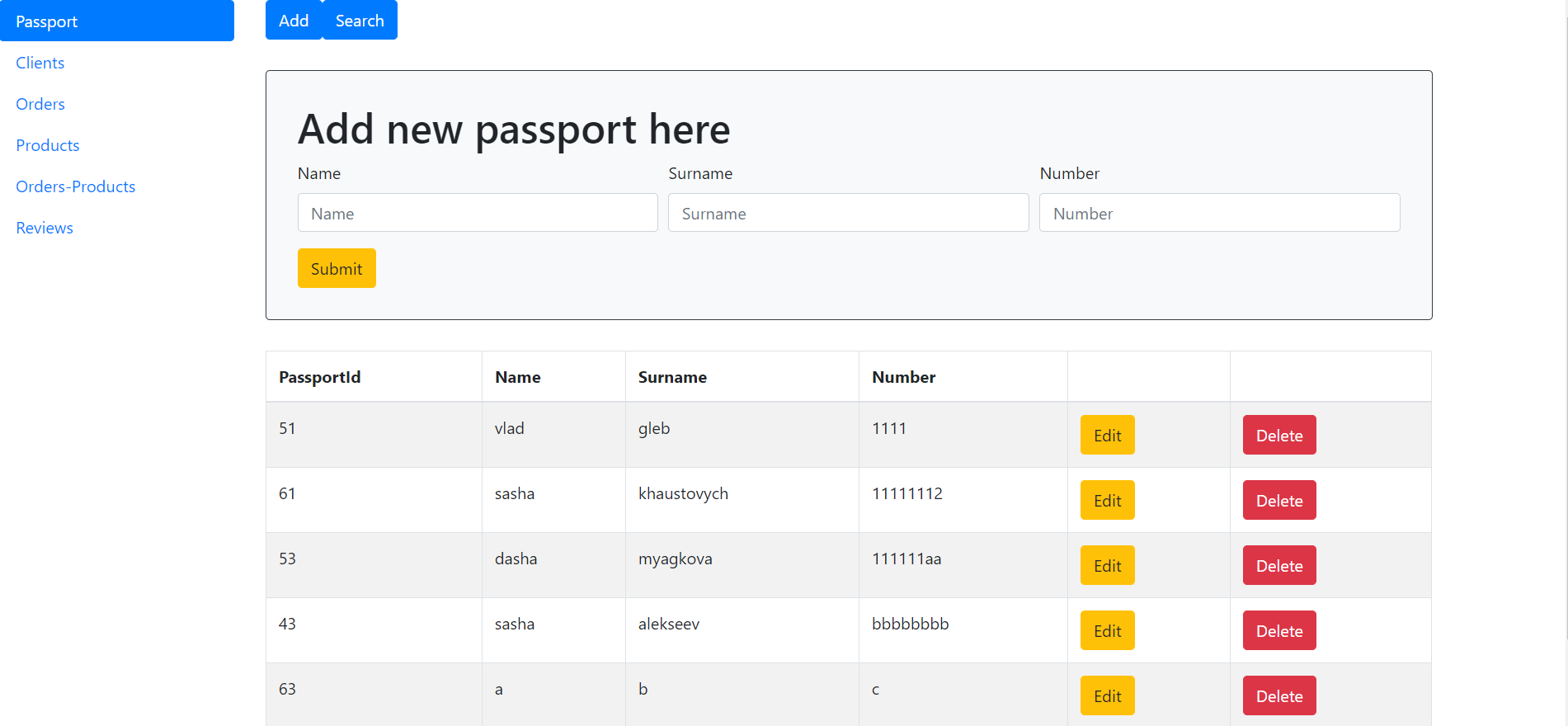
****

Рис 2.2 Меню програми

**Інтерфейс програми**

Для кожної таблиці було створено форма з відповідною назвою. З вибором конкретної, відкривається меню програми. На рис 2.2 можна побачити 2 кнопки “add” та “search”, які відповідають за додавання та пошук даних у таблиці. На кожній з форм є форма, для відповідних дій та список даних. За замовчуванням відкрита форма для додавання.

При додаванні (POST), відображення(GET), змінні(PUT) та видаленні(DELETE) даних на сервер передається відповідний запит.

1. Форма “Passport” взаємодії з таблицею “passport”. Дозволяє добавити новий запис, змінити або видалити вже існуючі.

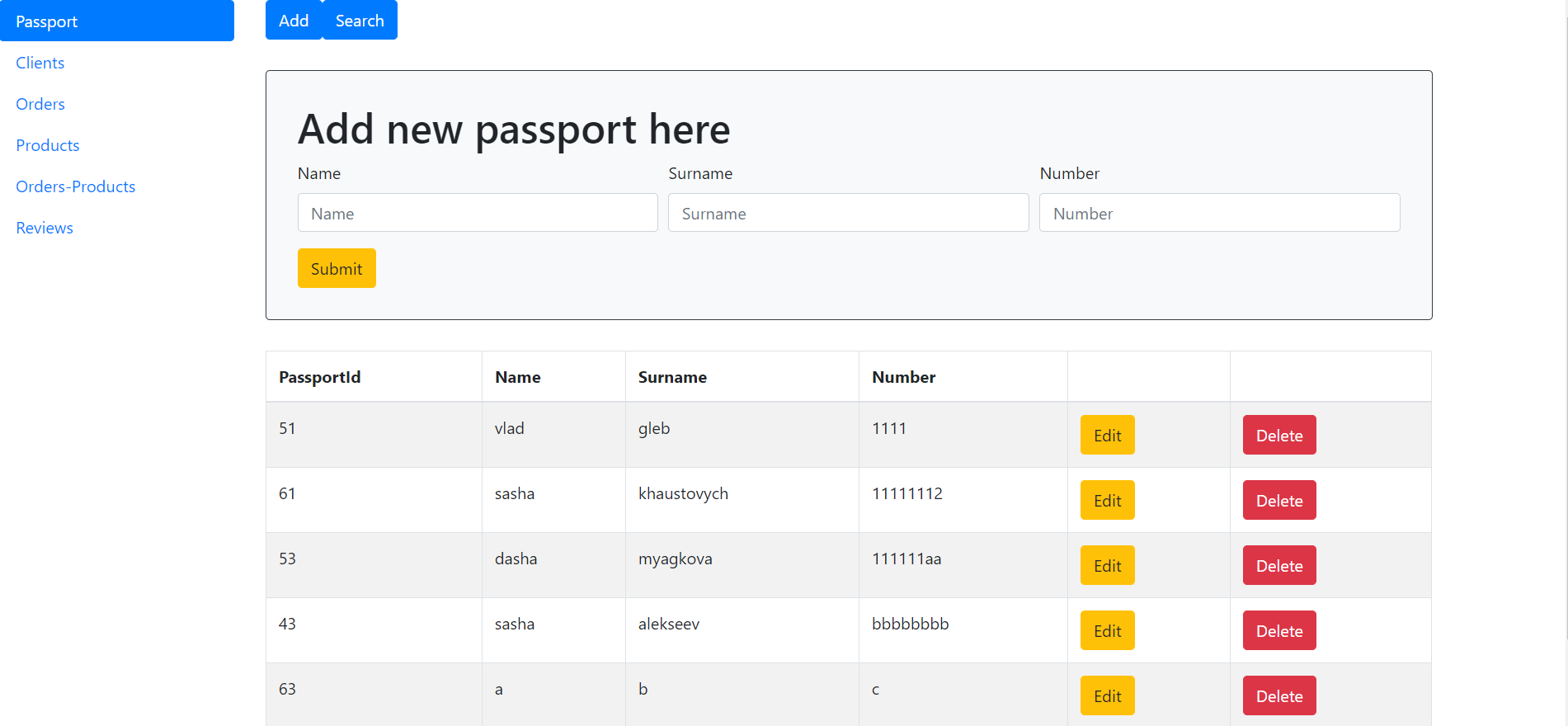
****

Рис 2.3a Форма “add” для “Passport”

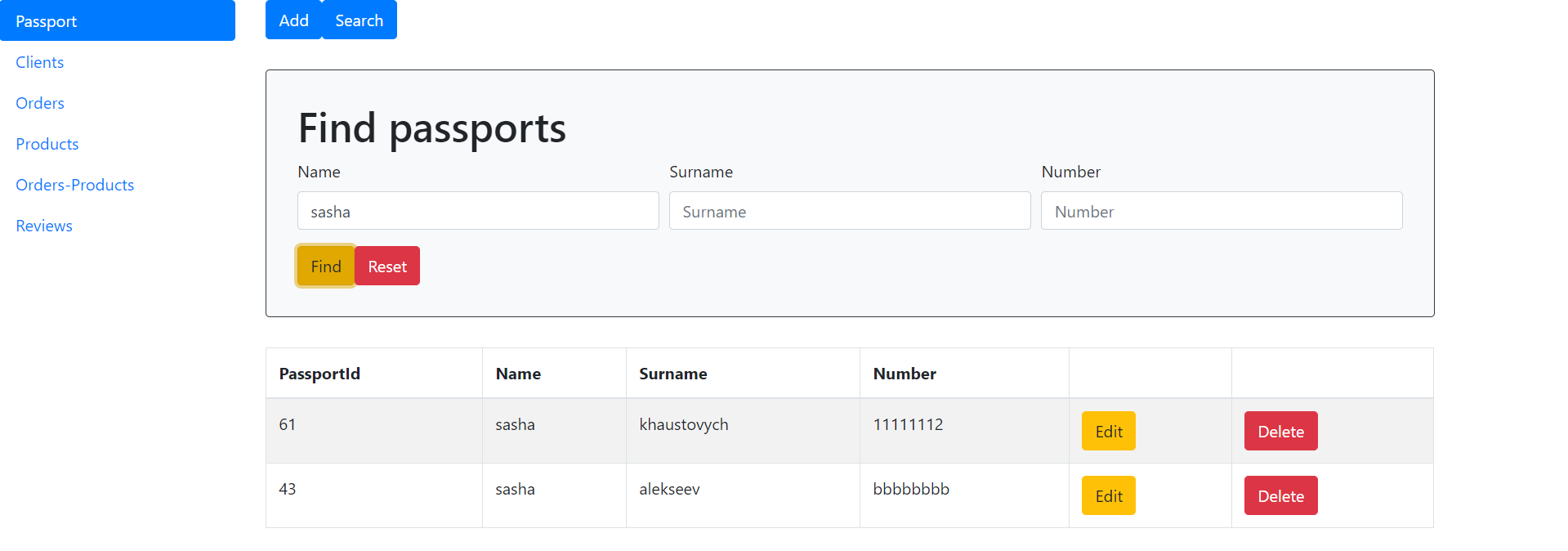


Рис 2.3б Форма “search” для “Passport”

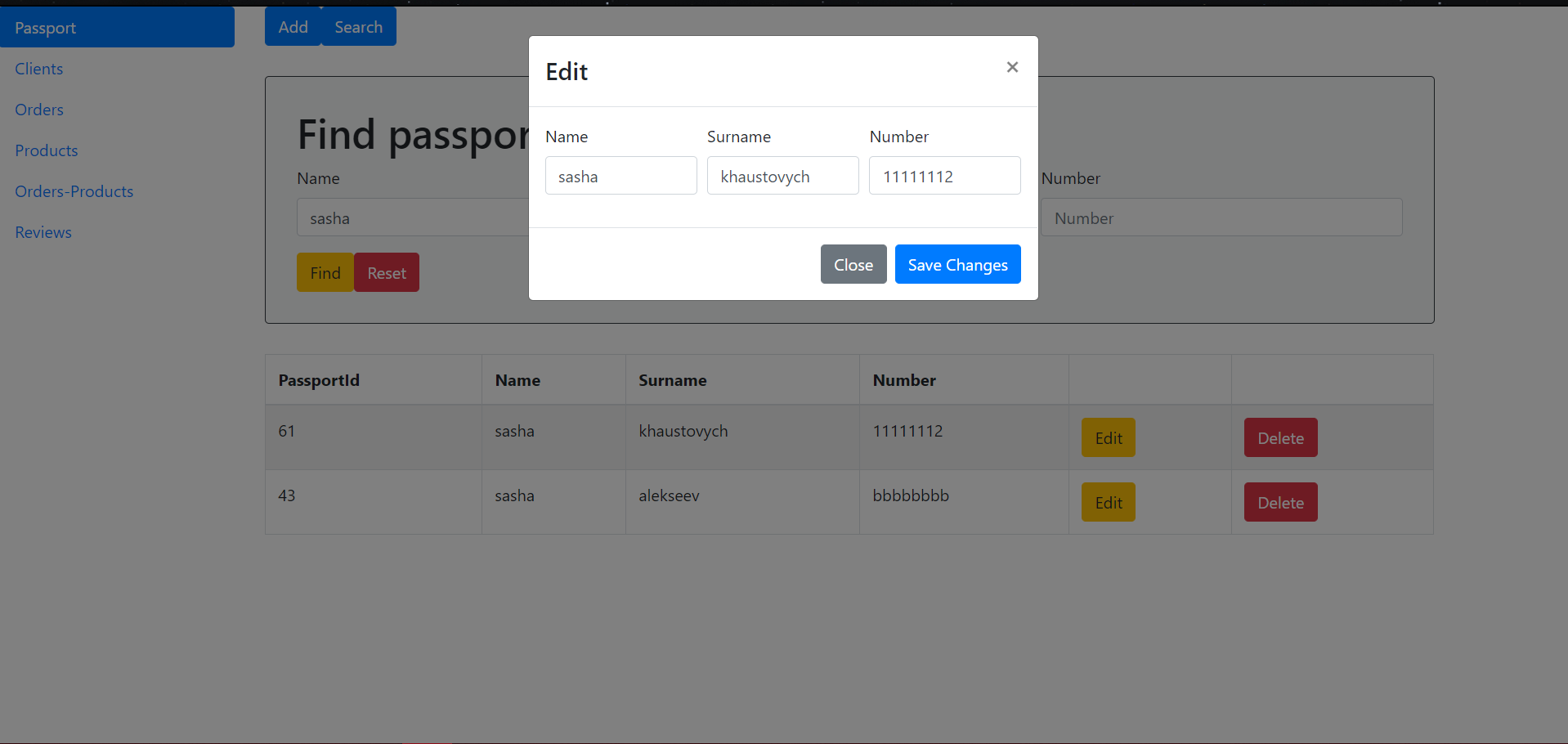


Рис 2.3в Модальне вікно для зміни даних у “Passport”

1. Форма “Clients ” взаємодії з таблицею “clients”. Дозволяє добавити новий запис, змінити або видалити вже існуючі.

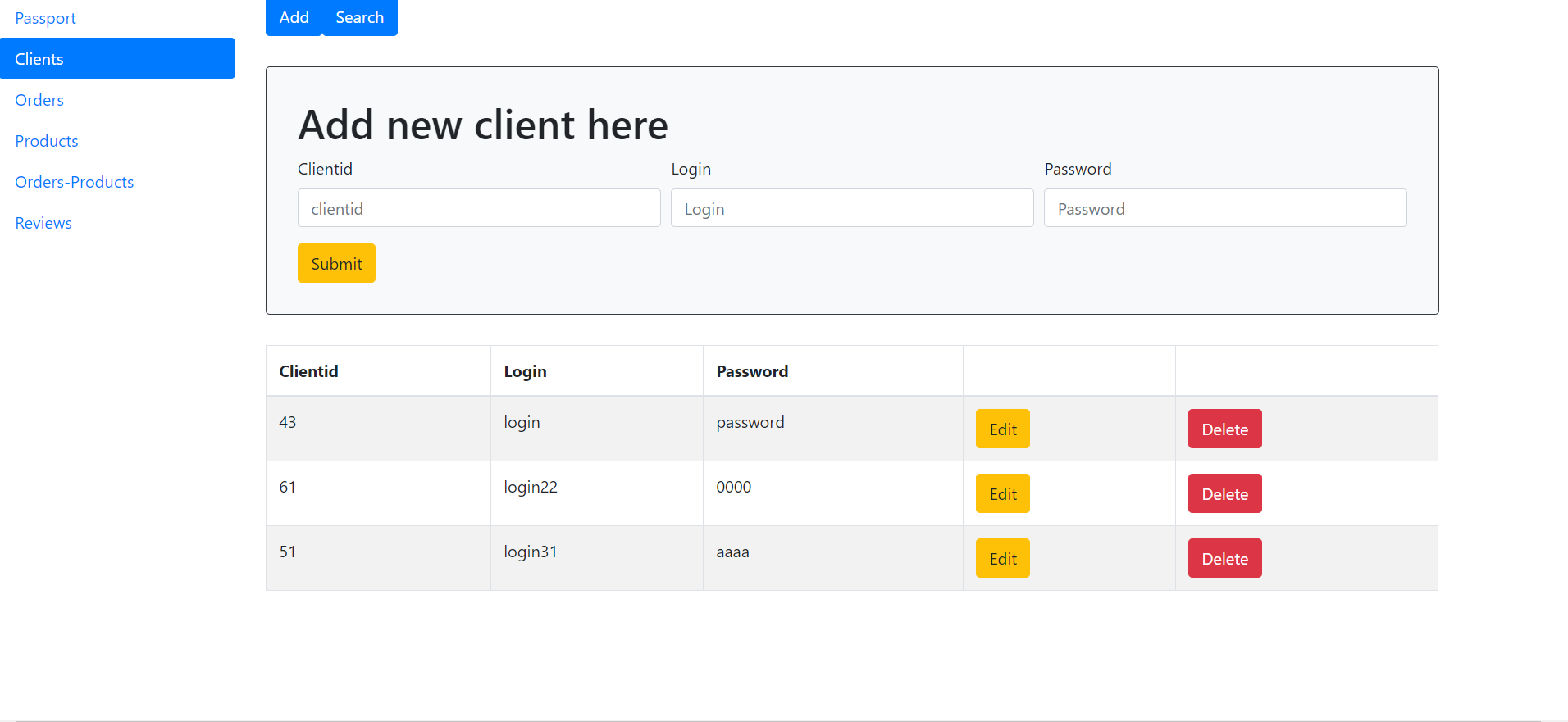


Рис 2.4a Форма “add” для “Clients”

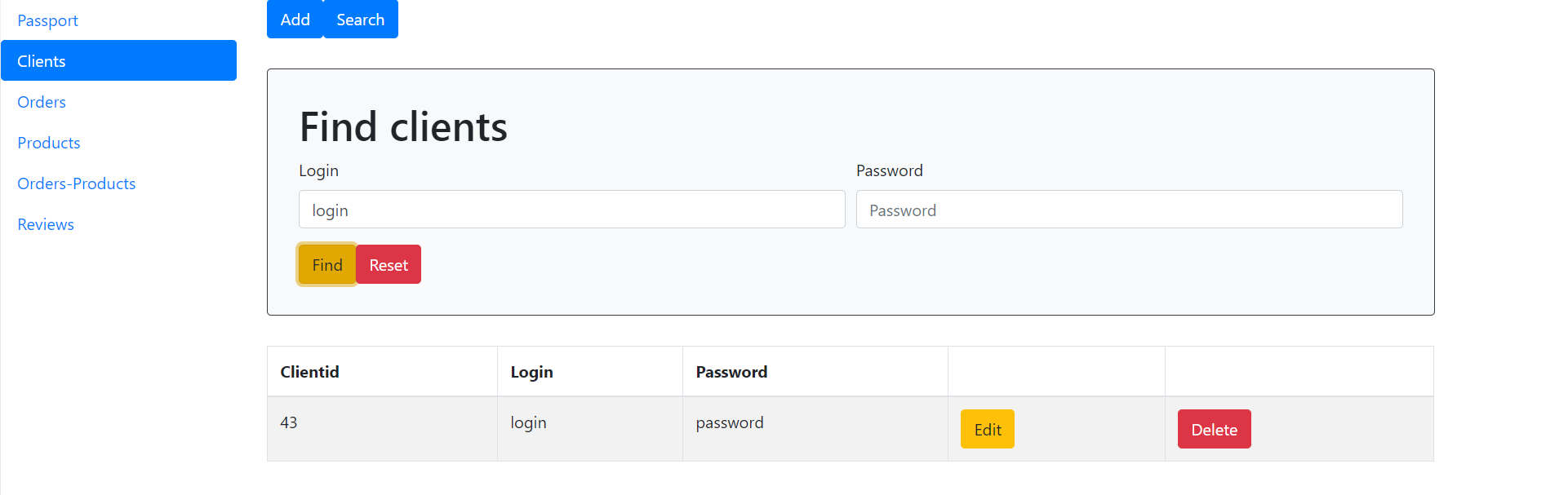


Рис 2.4б Форма “search” для “ Clients ”

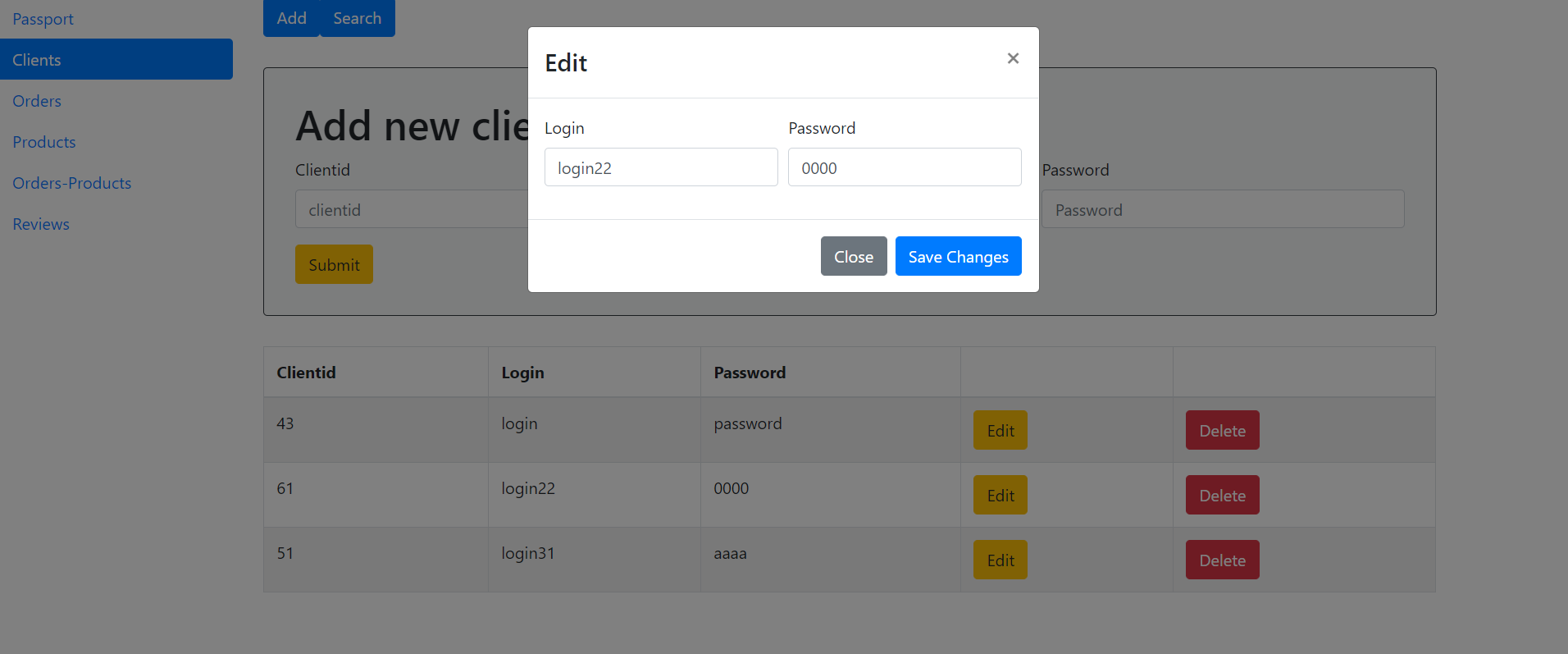


Рис 2.4в Модальне вікно для зміни даних у “ Clients ”

1. Форма “Orders ” взаємодії з таблицею “orders”. Дозволяє добавити новий запис та видалити вже існуючі.

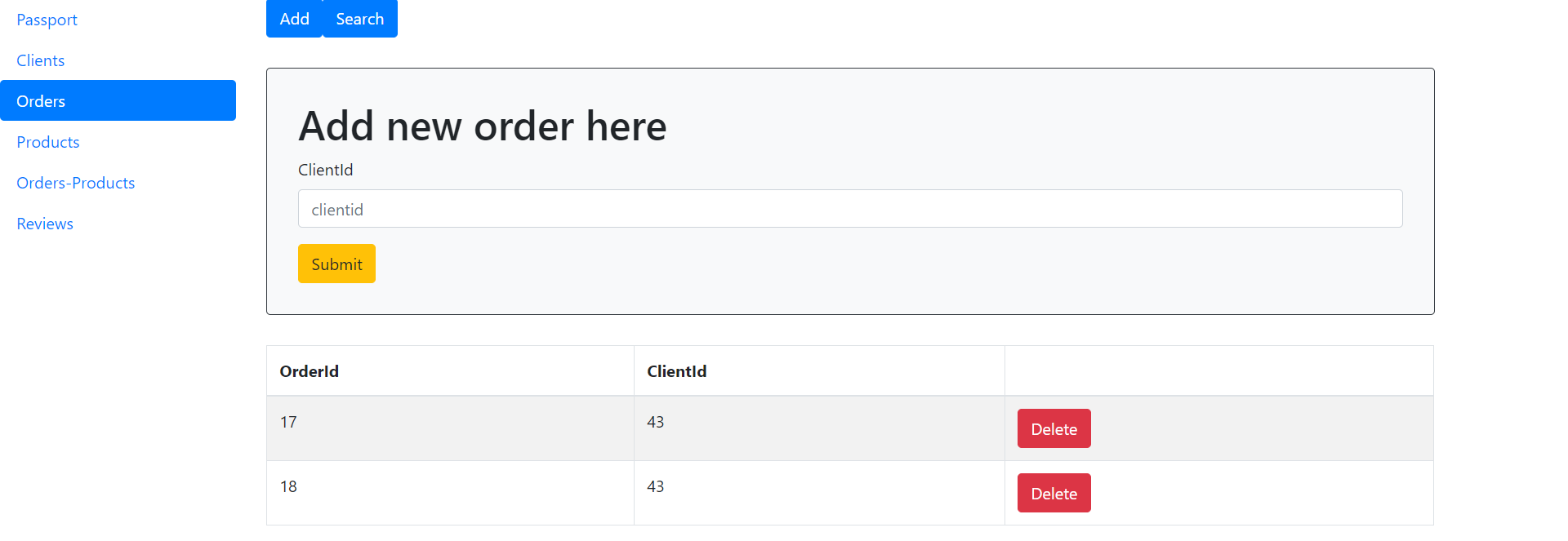


Рис 2.5a Форма “add” для “Orders”

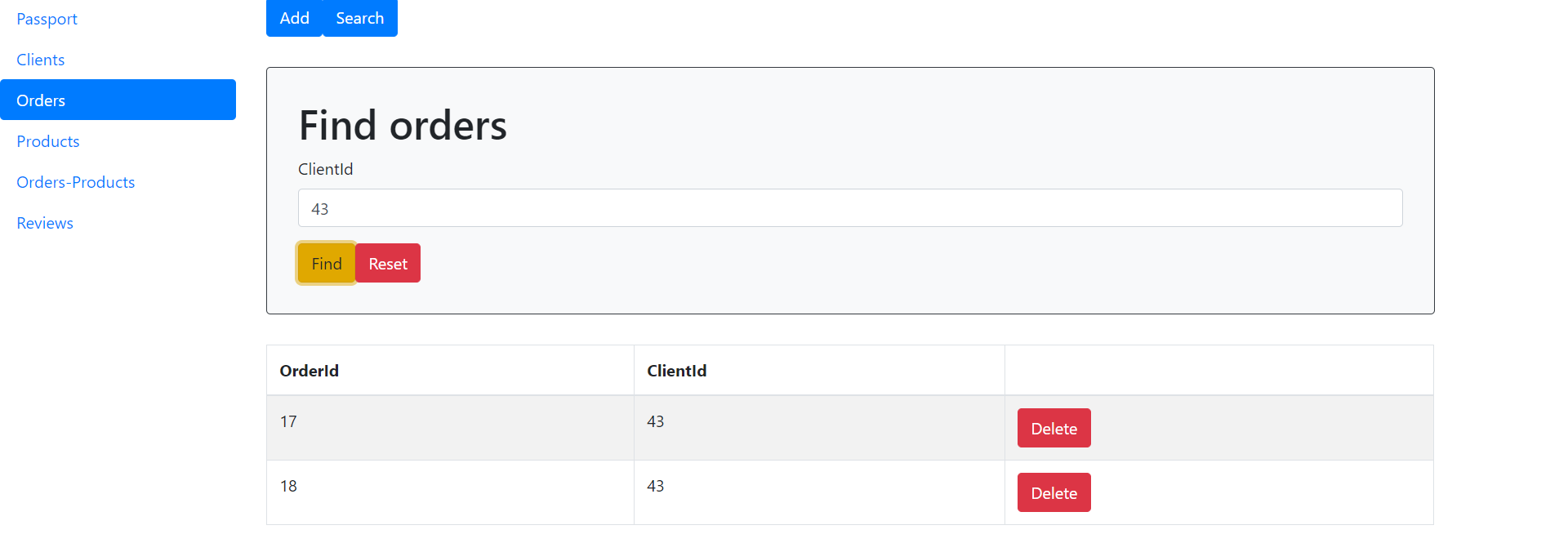


Рис 2.5б Форма “search” для “ Orders ”

1. Форма “Products ” взаємодії з таблицею “products”. Дозволяє добавити новий запис, змінити або видалити вже існуючі.

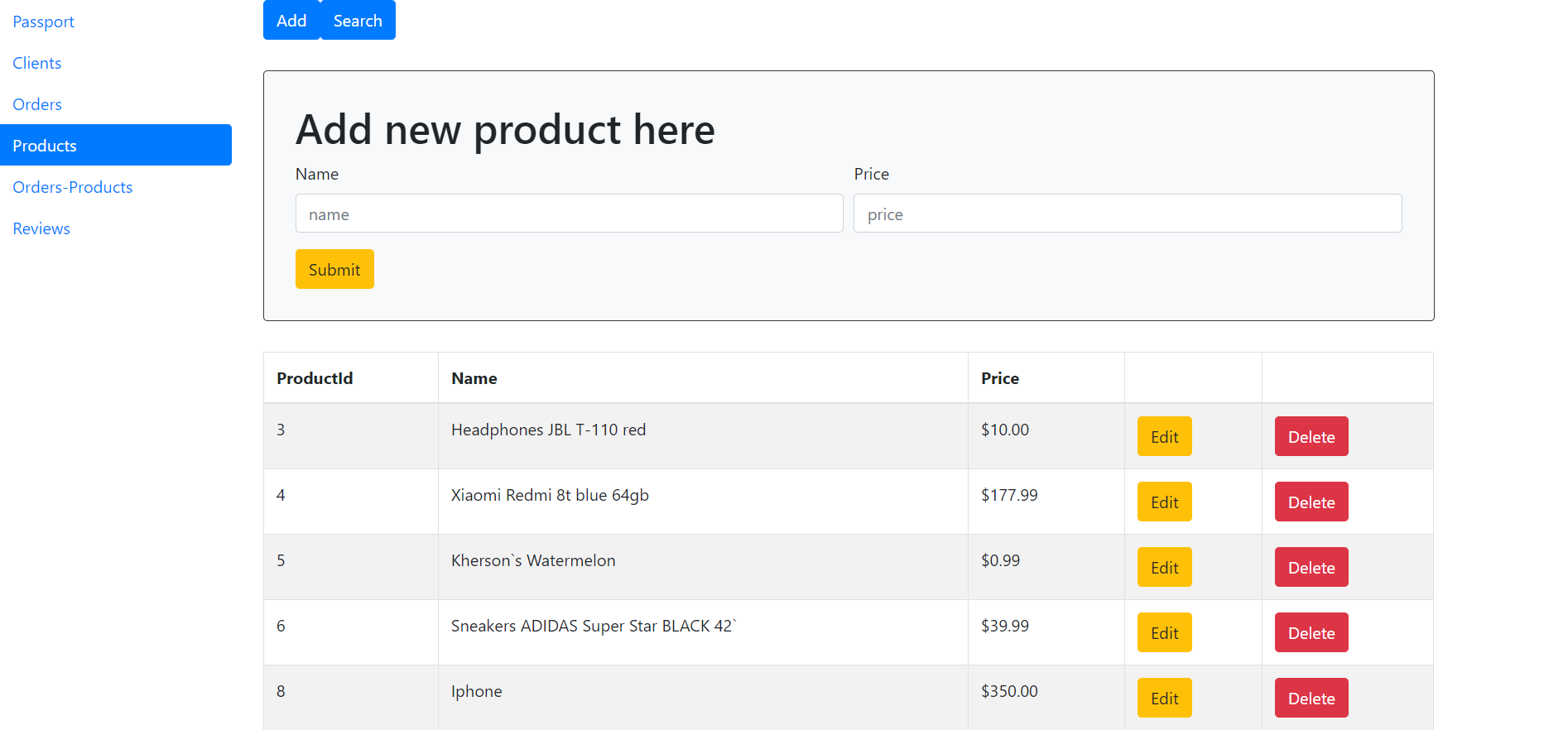


Рис 2.6a Форма “add” для “Products”

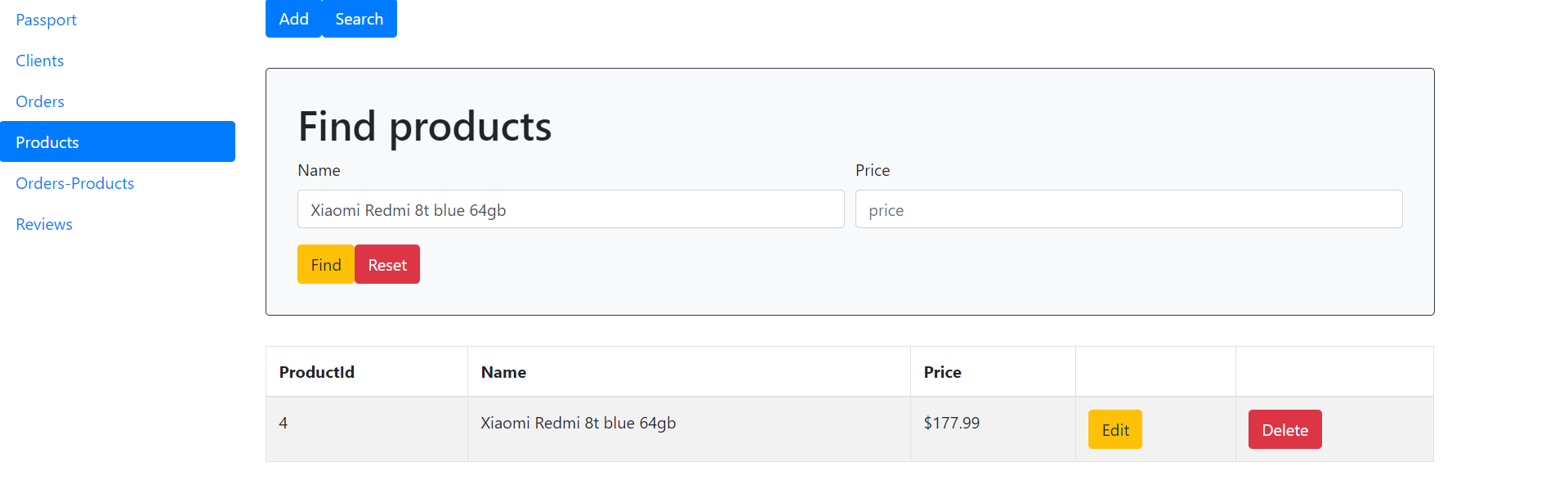


Рис 2.6б Форма “search” для “Products”

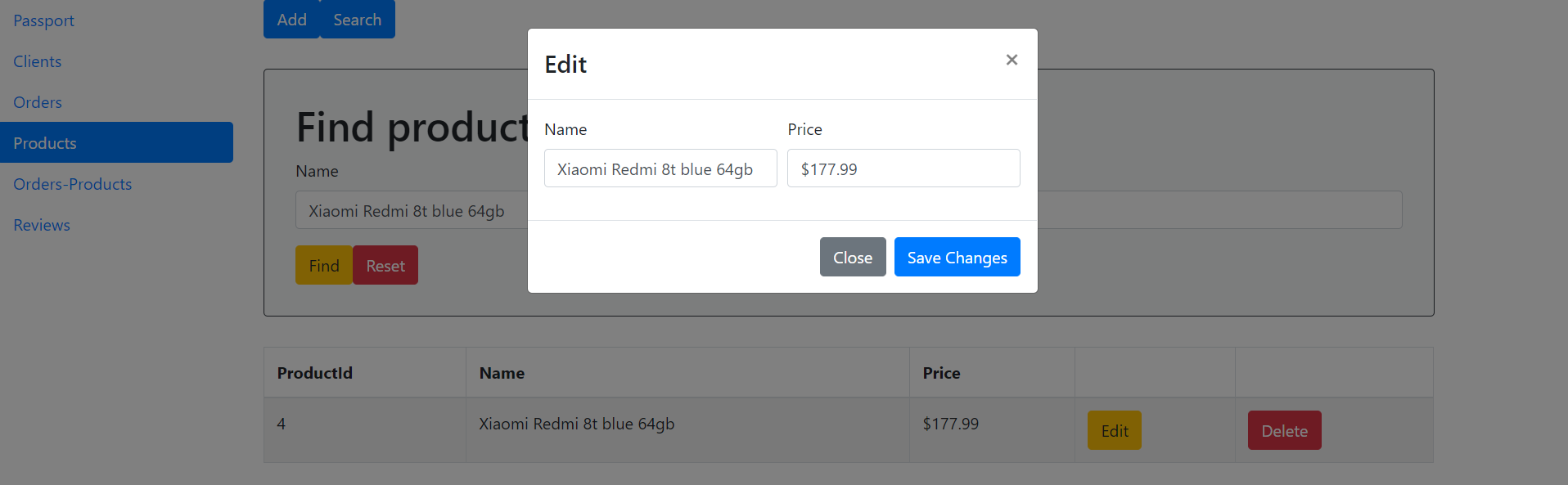
****

Рис 2.6в Модальне вікно для зміни даних у “ Clients ”

1. Форма “Orders-Products” взаємодії з таблицею “products”. Дозволяє добавити новий запис та видалити вже існуючі.

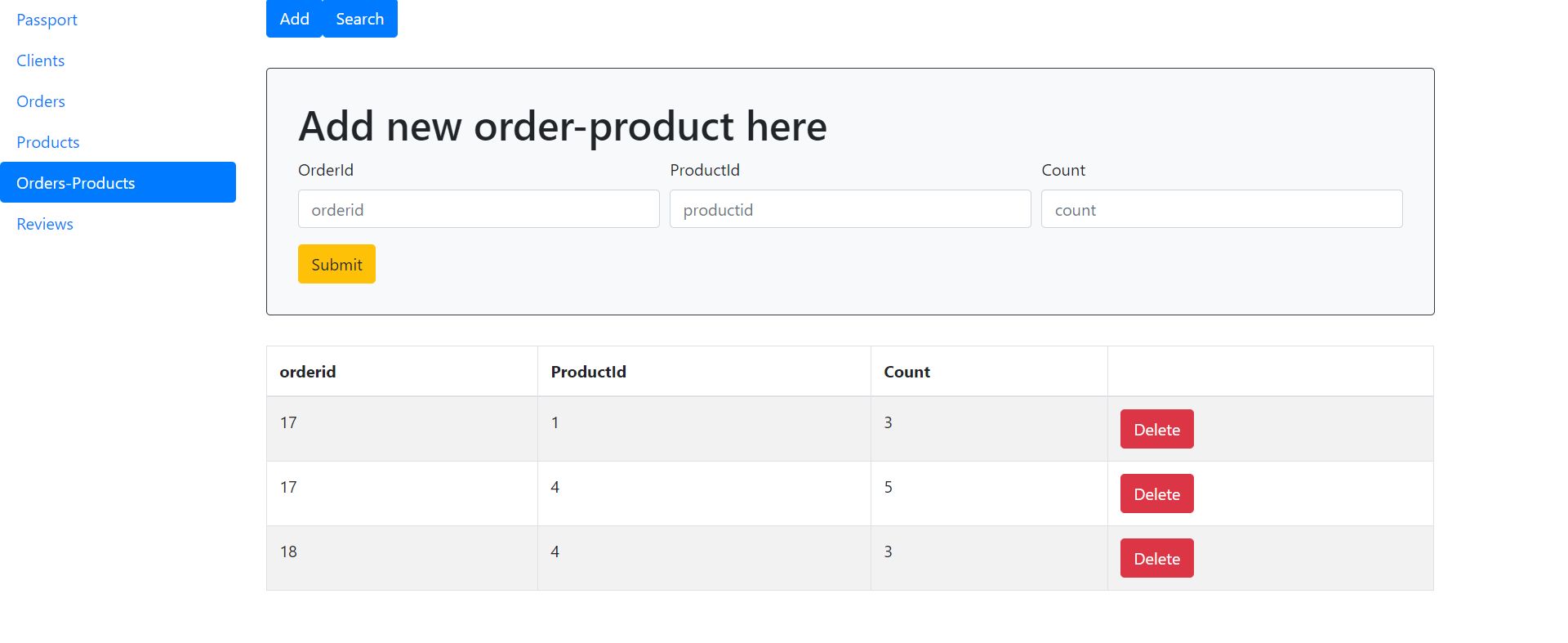


Рис 2.7a Форма “add” для “Orders-Products”

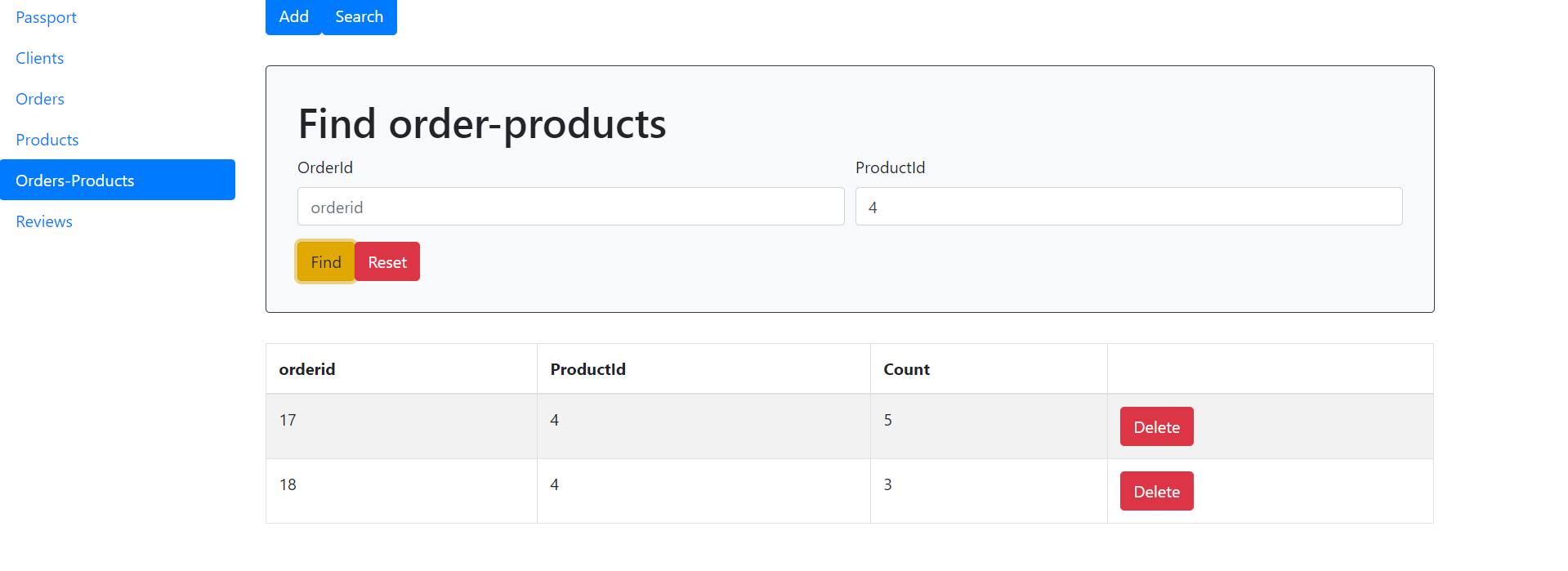
****

Рис 2.7б Форма “search” для “ Orders-Products ”

1. Форма “Reviews” взаємодії з таблицею “reviews”. Дозволяє добавити новий запис та видалити вже існуючі.

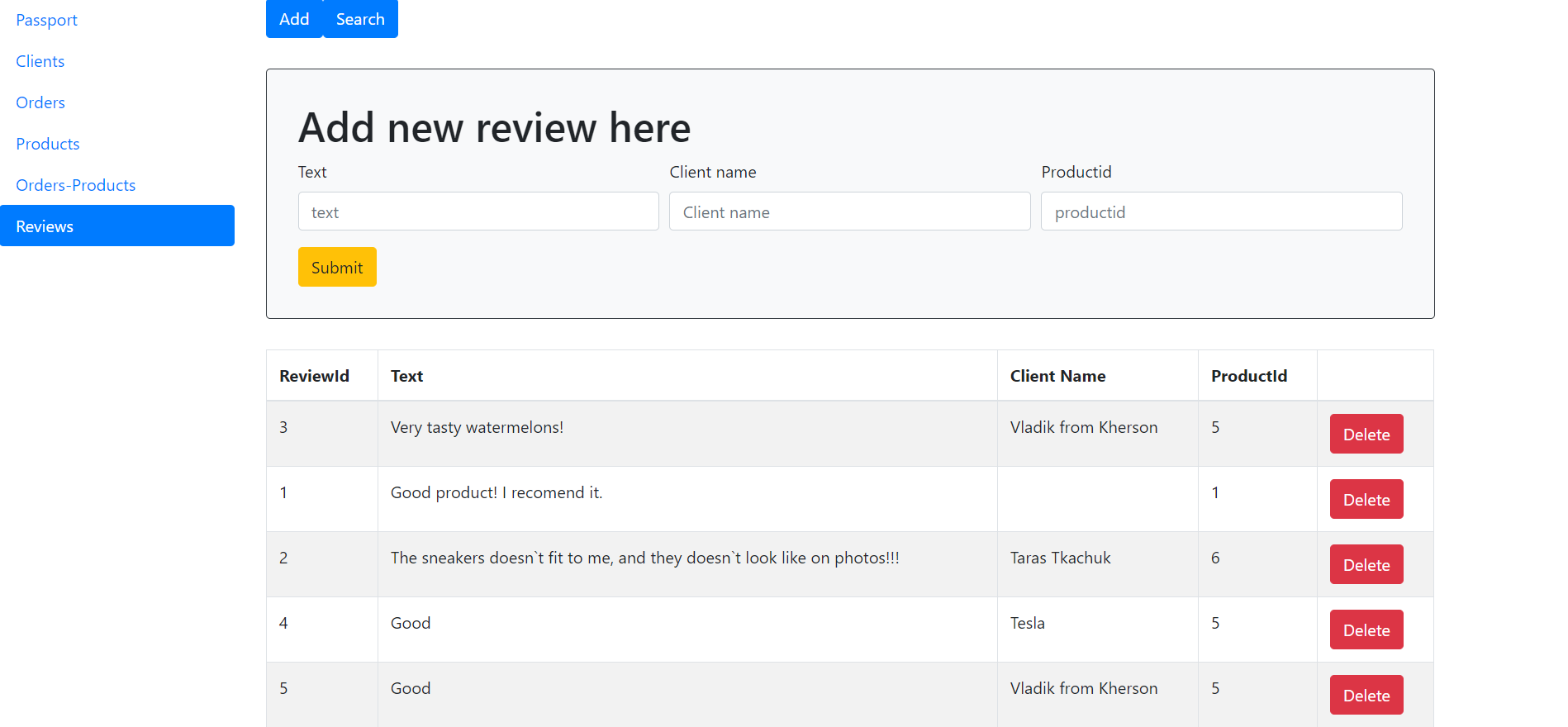


Рис 2.8a Форма “add” для “Orders-Products”

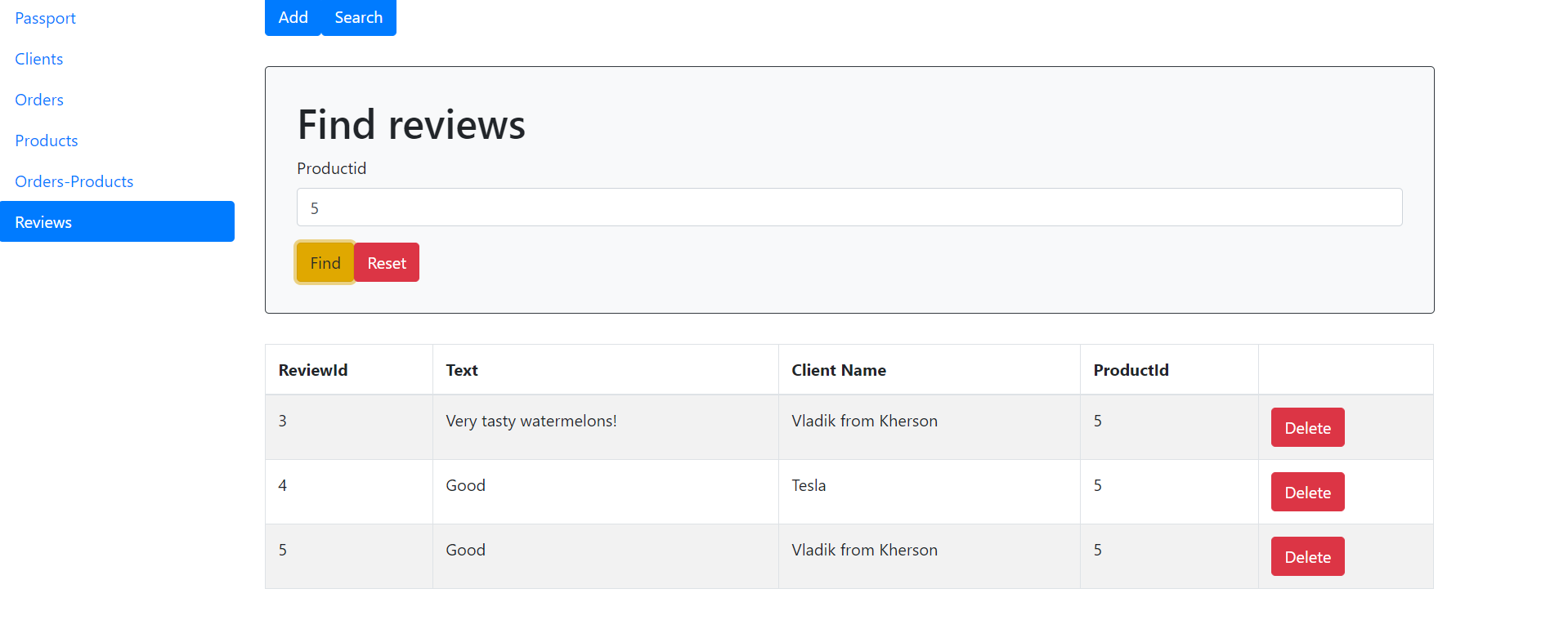


Рис 2.8б Форма “search” для “ Orders-Products ”

**Вибірка елементів з БД**

Для виводу усіх даних у кожній таблиці використовується однаковий метод, який відправляє GET-запит до сервера на URL

“**http://localhost:5000/\*назва таблиці\*”**

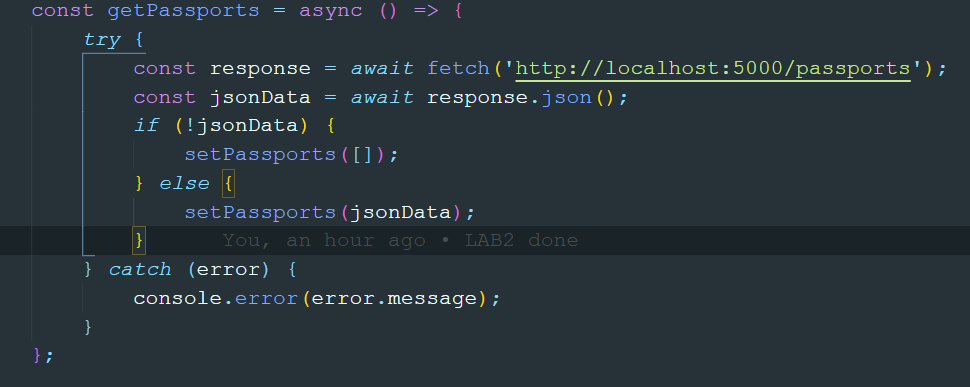


Рис 2.9 Метод, який виводить всі записи в таблиці “passport”

**Додавання елементів до БД**

При додавання даних відправляється POST-запит до серверу на URL **“http://localhost:5000/\*назва таблиці\*”** і для цього використовується майже однаковий метод, який у запиті передає об’єкт з назвами полів та відповідними даними.

****

Рис 2.10 Метод, який додає запис до таблиці “clients”

**Змінення елементів у БД**

При змінні даних відправляється PUT-запит до серверу на URL **“http://localhost:5000/\*назва таблиці\*/\*id-запису\*”** і для цього використовується майже однаковий метод, який у запиті передає об’єкт з назвами полів та відповідними даними.

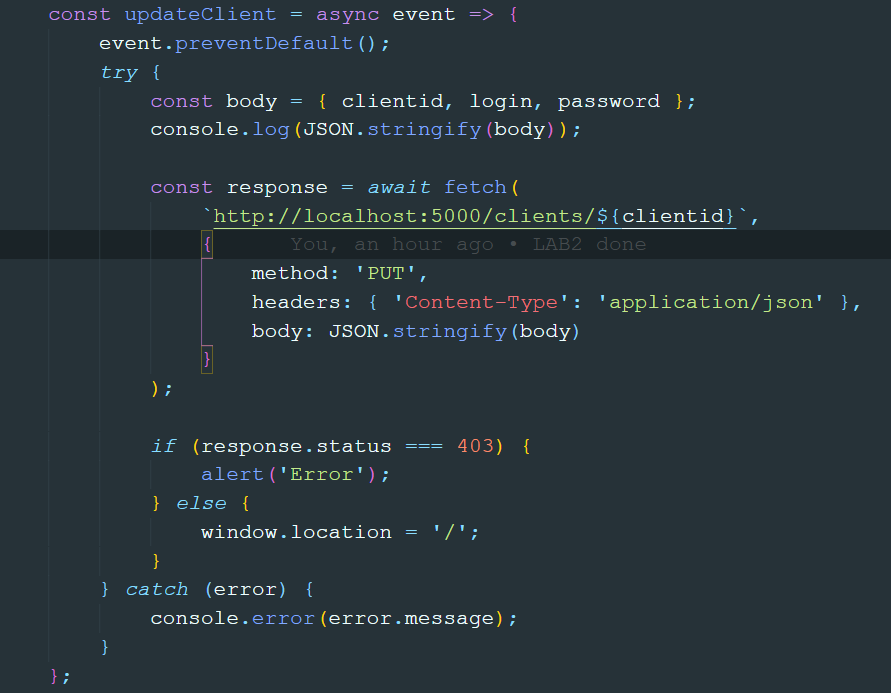


Рис 2.11 Метод, який додає запис до таблиці “clients”

**Видалення елементів з БД**

PostgreSQL також дає можливість вибрати кілька режимів видалення:

1. **“ON DELETE SET NULL”** - всі Foreign key будуть мати значення Null, а за неможливості цього зробити буде повідомлення про помилку. При спробі видалити запис з “passport”, в якого є дочірні записи у “clients” сервер повертає таку помилку.



1. **“ON DELETE RESTRICT”** – не дає можливості видалити батьківський рядок, якщо в нього є дочірні. Поведінка дуже похожа на NO ACTION. При спробі видалити запис з “passport”, в якого є дочірні записи у “clients” сервер повертає таку помилку.



1. **“ON DELETE СASCADE”** – при видаленні батьківського рядка усі дочірні рядки також видаляються.

При створенні БД “Інтернет магазин” для кожної з таблиць був встановлений режим **“ON DELETE СASCADE”.** Після видалення запис одразу пропадає з списку данних.

При видаленні даних відправляється DELETE-запит до серверу на URL **“http://localhost:5000/\*назва таблиці\*/\*id-запису\*”** і для цього використовується майже однаковий метод.

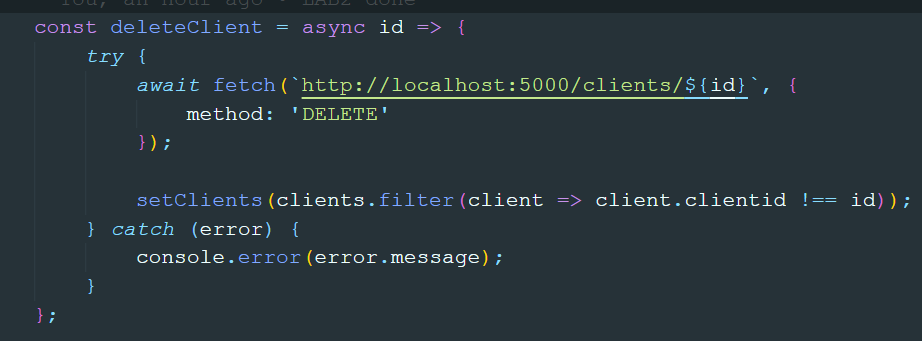


Рис 2.12 Метод, який видаляє запис з таблиці “clients”

**Серверна частина п програми**

Програма створена за патерном MVC (Model-View-Controller). Складається з модулів Controller та Model. Роль View виконує користувацький веб-інтерфейс. Опис файлів:

1. index.js - запускає сервер, підключає потрібні модулі і пакети;
2. Для кожної таблиці у папці routers були створена папка, яка містить файл index.js в якому описані усі запити, які може відправити користувач з інтерфейсу. Файл index.js у папці routers являє собою збірку усіх запитів, для більш комфортної роботи;
3. Для кожної таблиці у папці controllers були створена папка, яка містить файл controllers.js в якому викликається потрібні методи для роботи з БД;
4. Для кожної таблиці у папці model були створена папка, яка містить файл model.js, який посилає запити в БД і віддає відповідь контролеру;

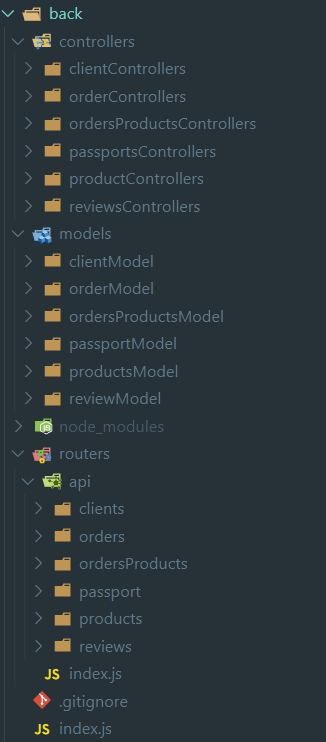
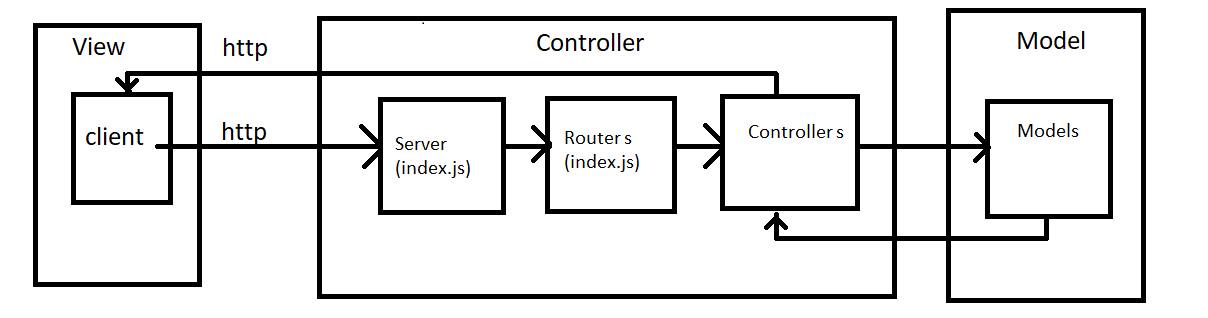


Рис 2.13 Файлов структура програми

**Посилання для навігації по програмі**

1. [Лістинг модуля index.js](#_Лістинг_модуля_index.js)
2. [Лістинг модуля routers/index.js](#_Лістинг_модуля_routers/index.js)
3. [Приклад: Лістинг модуля routers/api/orders/index.js](#_Лістинг_модуля_routers/api/orders/i)
4. [Models](#_Models)
5. Passport
   1. [Лістинг модуля models/passportModel/passportModel.js](#_Лістинг_модуля_models/passportModel)
   2. [Лістинг модуля controllers/passportControllers/controllers.js](#_Лістинг_модуля_controllers/passport)
6. Clients
   1. [Лістинг модуля models/clientModel/ clientModel.js](#_Лістинг_модуля_models/clientModek/)
   2. [Лістинг модуля controllers/clientsControllers/controllers.js](#_Лістинг_модуля_controllers/clientsC)
7. Orders
   1. [Лістинг модуля models/orderModel/ orderModel.js](#_Лістинг_модуля_models/clientModek/)
   2. [Лістинг модуля controllers/ordersControllers/controllers.js](#_Лістинг_модуля_controllers/ordersCo)
8. Products
   1. [Лістинг модуля models/productModel/productModel.js](#_Лістинг_модуля_models/productModel/)
   2. [Лістинг модуля controllers/productsControllers/controllers.js](#_Лістинг_модуля_controllers/ordersPr)
9. Orders-Products
   1. [Лістинг модуля models/ordersProductsModel/ ordersProductsModel.js](#_Лістинг_модуля_models/ordersProduct)
   2. [Лістинг модуля controllers/ordersProductsControllers/controllers.js](#_Лістинг_модуля_controllers/ordersPr)
10. Reviews
    1. [Лістинг модуля models/reviewModel/ reviewModel.js](#_Лістинг_модуля_models/reviewModel/r)
    2. [Лістинг модуля controllers/reviewsControllers/controllers.js](#_Лістинг_модуля_controllers/reviewsC)

**Структура програми**

Рис 2.13 Структура програми

**Обробка помилок**

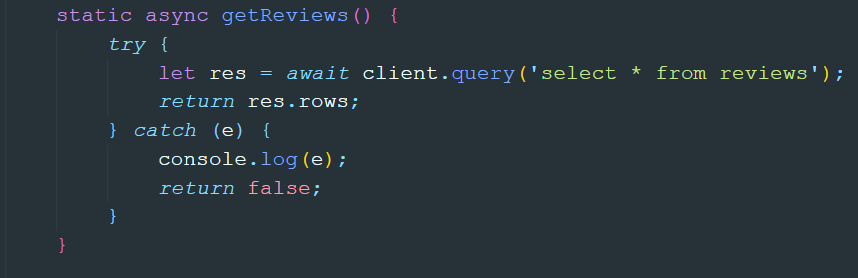
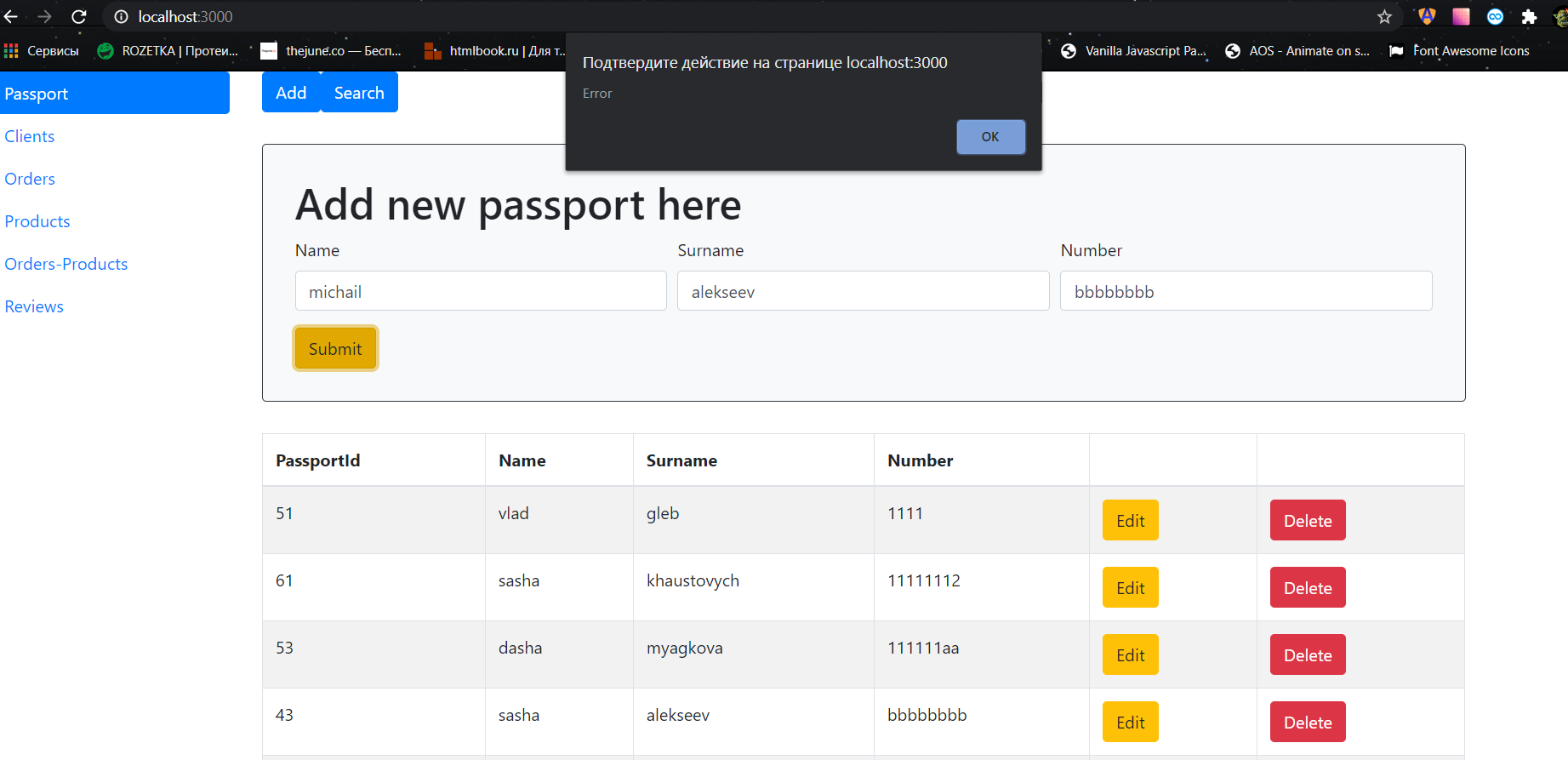
Для цього завдання кожен SQL запит було виконанно в контсрукції try\_catch, приклад:  


Рис 2.14 Метод виведення усіх записів таблиці “reviews”

Коли при запиті виникає якась помилка, то сервер поперне користувача відповідь з відповідним статусом помилки і текстом помилки. І коли користувач на свій запит до сервера отримує статус помилки, йому на кран буде виведено повідомлення про помилку.

Наприклад, в таблиці “passport” поле “number” має властивість UNIQUE. У програмі передбачена ситуація, якщо користувач увів вже існуючі дані.  


# Лістинг модуля index.js

const router = require('./routers/index');

const express = require('express');

const app = express();

const port = process.env.PORT || 5000;

const cors = require('cors');

const bodyParser = require('body-parser');

app.use(cors());

app.use(bodyParser.json());

const start = async () => {

    app.listen(port, () => console.log('Server is working on port ' + port));

};

app.get('/passports', router); *//get passport*

app.post('/passports', router); *//add passport*

app.put('/passports/:id', router); *//update passport*

app.delete('/passports/:id', router); *//delete passport*

app.post('/passports/find', router); *//find passports*

app.get('/clients', router); *//get clients*

app.post('/clients', router); *//add client*

app.put('/clients/:id', router); *//update client*

app.delete('/clients/:id', router); *//delete client*

app.post('/clients/find', router); *//find clients*

app.get('/orders', router); *//get orders*

app.post('/orders', router); *//add Order*

app.delete('/orders/:id', router); *//delete Order*

app.post('/orders/find', router); *//find orders*

app.get('/products', router); *//get products*

app.post('/products', router); *//add products*

app.put('/products/:id', router); *//update products*

app.delete('/products/:id', router); *//delete products*

app.post('/products/find', router); *//find products*

app.get('/ordersProducts', router); *//get orders and products*

app.post('/ordersProducts', router); *//add order and product*

app.delete('/ordersProducts/:id', router); *//delete order and product*

app.post('/ordersProducts/find', router); *//find order and product*

app.get('/reviews', router); *//get Reviews*

app.post('/reviews', router); *//add Reviews*

app.delete('/reviews/:id', router); *//delete Reviews*

app.post('/reviews/find', router); *//find Reviews*

start();

# Лістинг модуля routers/index.js

const router = require('express').Router();

const passportRouter = require('./api/passport');

const clientsRouter = require('./api/clients');

const ordersRouter = require('./api/orders');

const productsRouter = require('./api/products');

const ordersProductsRouter = require('./api/ordersProducts');

const reviewsRouter = require('./api/reviews');

const bodyParser = require('body-parser');

router.use(bodyParser.urlencoded({ extended: false }));

router.get('/passports', passportRouter); *//get passports*

router.post('/passports', passportRouter); *//add passports*

router.put('/passports/:id', passportRouter); *//update passport*

router.delete('/passports/:id', passportRouter); *//delete passport*

router.post('/passports/find', passportRouter); *//find passports*

router.get('/clients', clientsRouter); *//get clients*

router.post('/clients', clientsRouter); *//add client*

router.put('/clients/:id', clientsRouter); *//update client*

router.delete('/clients/:id', clientsRouter); *//delete client*

router.post('/clients/find', clientsRouter); *//find clients*

router.get('/orders', ordersRouter); *//get orders*

router.post('/orders', ordersRouter); *//add Order*

router.delete('/orders/:id', ordersRouter); *//delete Order*

router.post('/orders/find', ordersRouter); *//find orders*

router.get('/products', productsRouter); *//get products*

router.post('/products', productsRouter); *//add products*

router.put('/products/:id', productsRouter); *//update products*

router.delete('/products/:id', productsRouter); *//delete products*

router.post('/products/find', productsRouter); *//find products*

router.get('/ordersProducts', ordersProductsRouter); *//get orders and products*

router.post('/ordersProducts', ordersProductsRouter); *//add order and product*

router.delete('/ordersProducts/:id', ordersProductsRouter); *//delete order and product*

router.post('/ordersProducts/find', ordersProductsRouter); *//find order and product*

router.get('/reviews', reviewsRouter); *//get Reviews*

router.post('/reviews', reviewsRouter); *//add Reviews*

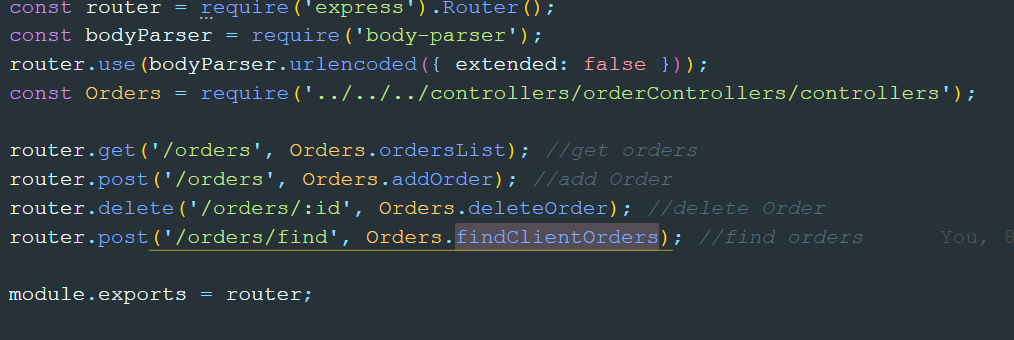
router.delete('/reviews/:id', reviewsRouter); *//delete Reviews*

router.post('/reviews/find', reviewsRouter); *//find Reviews*

module.exports = router;

Усі файли з каталогу routers/api мають однакову структуру. Як було вже сказано, усі обробники запитів було імпортовано до файлу routers/index.js, для більш комфортного використання у файлі index.js. З отриманням запиту сервер звертається до методу відповідного контролера.

# Лістинг модуля routers/api/orders/index.js



# Models

Для кожної таблиці була створена відповідна модель-клас, яка містить відповідні методи get, add, edit, find та delete.

# Лістинг модуля models/passportModel/ passportModel.js

const { Client } = require('pg');

const client = new Client(

    'postgres://postgres:Misha211100@localhost:5432/internetshop'

);

client.connect();

class Passport {

    constructor(name, surname, number) {

        this.name = name;

        this.surname = surname;

        this.number = number;

    }

    static async getPassports() {

*try* {

            let res = *await* client.query('select \* from passport');

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async addPassport() {

*try* {

*await* client.query(

                `insert into passport (name, surname, number) values ('${this.name}','${this.surname}', '${this.number}')`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async editPassport(id) {

*try* {

*await* client.query(

                `update passport set name='${this.name}', surname='${this.surname}', number = '${this.number}' where passport\_id=${id}`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async deletePassport(id) {

*try* {

*await* client.query(`delete from passport where passport\_id=${id}`);

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async findPassports(name, surname, number) {

        let res;

*try* {

*if* (name && surname && number) {

                res = *await* client.query(

                    `select \* from passport where name='${name}' and surname='${surname}' and number='${number}'`

                );

            } *else* *if* (name && surname) {

                res = *await* client.query(

                    `select \* from passport where name='${name}' and surname='${surname}'`

                );

            } *else* *if* (name && number) {

                res = *await* client.query(

                    `select \* from passport where name='${name}' and number = '${this.number}'`

                );

            } *else* *if* (surname && number) {

                res = *await* client.query(

                    `select \* from passport where surname='${surname}' and number = '${this.number}'`

                );

            } *else* *if* (name) {

                res = *await* client.query(

                    `select \* from passport where name='${name}'`

                );

            } *else* *if* (surname) {

                res = *await* client.query(

                    `select \* from passport where surname='${surname}'`

                );

            } *else* *if* (number) {

                res = *await* client.query(

                    `select \* from passport where number='${number}'`

                );

            }

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

}

module.exports = Passport;

# Лістинг модуля controllers/passportControllers/controllers.js

const Passport = require('../../models/passportModel/passportModel');

class Passports {

    static async passportList(req, res) {

*try* {

            let passports = *await* Passport.getPassports();

            res.status(200).json(passports);

        } *catch* (e) {

            res.status(500).json({ err: 'Server error' });

        }

    }

    static async addPassport(req, res) {

        const passport = new Passport(

            req.body.name,

            req.body.surname,

            req.body.number

        );

*if* (*await* passport.addPassport()) {

            res.status(201).json({ message: 'Passport was added' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async editPassport(req, res) {

        const { id } = req.params;

        const passport = new Passport(

            req.body.name,

            req.body.surname,

            req.body.number

        );

*if* (*await* passport.editPassport(id)) {

            res.status(201).json({ message: 'Data was updated' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async deletePassport(req, res) {

        const { id } = req.params;

*if* (*await* Passport.deletePassport(id)) {

            res.status(201).json({ message: 'Passport was deleted' });

        } *else* {

            res.status(403).json({ err: 'Error' });

        }

    }

    static async findPassports(req, res) {

*try* {

            const { name, surname, number } = req.body;

            const passports = *await* Passport.findPassports(

                name,

                surname,

                number

            );

            res.status(200).json(passports);

        } *catch* (error) {

            res.status(500).json({ err: 'Server error' });

        }

    }

}

module.exports = Passports;

# Лістинг модуля models/clientModel/ clientModel.js

const { Client } = require('pg');

const client = new Client(

    'postgres://postgres:Misha211100@localhost:5432/internetshop'

);

client.connect();

class ClientTable {

    constructor(passportId, login, password) {

        this.passportId = passportId;

        this.login = login;

        this.password = password;

    }

    static async getClients() {

*try* {

            let res = *await* client.query('select \* from clients');

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async addClient() {

*try* {

*await* client.query(

                `insert into clients (clientid, login, password) values (${this.passportId},'${this.login}','${this.password}')`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async editClient(id) {

*try* {

*await* client.query(

                `update clients set login='${this.login}', password='${this.password}' where clientid=${id}`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async deleteClient(id) {

*try* {

*await* client.query(`delete from clients where clientid=${id}`);

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async findClients(login, password) {

        let res;

*try* {

*if* (login && password) {

                res = *await* client.query(

                    `select \* from clients where login='${login}' and password='${password}'`

                );

            } *else* *if* (login) {

                res = *await* client.query(

                    `select \* from clients where login='${login}'`

                );

            } *else* *if* (password) {

                res = *await* client.query(

                    `select \* from clients where password='${password}'`

                );

            }

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

}

module.exports = ClientTable;

# Лістинг модуля controllers/clientsControllers/controllers.js

const Client = require('../../models/clientModel/clientModel');

class Clients {

    static async clientList(req, res) {

*try* {

            let clients = *await* Client.getClients();

            res.status(200).json(clients);

        } *catch* (e) {

            res.status(500).json({ err: 'Server error' });

        }

    }

    static async addClient(req, res) {

        const client = new Client(

            req.body.clientid,

            req.body.login,

            req.body.password

        );

*if* (*await* client.addClient()) {

            res.status(201).json({ message: 'Client was added' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async editClient(req, res) {

        const { id } = req.params;

        const client = new Client(

            req.body.passportID,

            req.body.login,

            req.body.password

        );

*if* (*await* client.editClient(id)) {

            res.status(201).json({ message: 'Data was updated' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async deleteClient(req, res) {

        const { id } = req.params;

*if* (*await* Client.deleteClient(id)) {

            res.status(201).json({ message: 'Client was deleted' });

        } *else* {

            res.status(403).json({ err: 'Error' });

        }

    }

    static async findClients(req, res) {

*try* {

            const { login, password } = req.body;

            const clients = *await* Client.findClients(login, password);

            res.status(200).json(clients);

        } *catch* (error) {

            res.status(500).json({ err: 'Server error' });

        }

    }

}

module.exports = Clients;

# Лістинг модуля models/orderModel/ orderModel.js

const { Client } = require('pg');

const client = new Client(

    'postgres://postgres:Misha211100@localhost:5432/internetshop'

);

client.connect();

class Order {

    constructor(clientid) {

        this.clientid = clientid;

    }

    static async getOrders() {

*try* {

            let res = *await* client.query('select \* from orders');

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async addOrder() {

*try* {

*await* client.query(

                `insert into orders (clientid) values (${this.clientid})`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async deleteOrder(id) {

*try* {

*await* client.query(`delete from orders where orderid=${id}`);

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async findClientOrders(clientid) {

        let res;

*try* {

            res = *await* client.query(

                `select \* from orders where clientid=${clientid}`

            );

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

}

module.exports = Order;

# Лістинг модуля controllers/ordersControllers/controllers.js

const Order = require('../../models/orderModel/orderModel');

class Orders {

    static async ordersList(req, res) {

*try* {

            let orders = *await* Order.getOrders();

            res.status(200).json(orders);

        } *catch* (e) {

            res.status(500).json({ err: 'Server error' });

        }

    }

    static async addOrder(req, res) {

        const order = new Order(req.body.clientid);

*if* (*await* order.addOrder()) {

            res.status(201).json({ message: 'Order was added' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async deleteOrder(req, res) {

        const { id } = req.params;

*if* (*await* Order.deleteOrder(id)) {

            res.status(201).json({ message: 'Order was deleted' });

        } *else* {

            res.status(403).json({ err: 'Error' });

        }

    }

    static async findClientOrders(req, res) {

*try* {

            const { clientid } = req.body;

            const orders = *await* Order.findClientOrders(clientid);

            res.status(200).json(orders);

        } *catch* (error) {

            res.status(500).json({ err: 'Server error' });

        }

    }

}

module.exports = Orders;

# Лістинг модуля models/productModel/ productModel.js

const Product = require('../../models/productsModel/productModel');

class Products {

    static async productsList(req, res) {

*try* {

            let products = *await* Product.getProducts();

            res.status(200).json(products);

        } *catch* (e) {

            res.status(500).json({ err: 'Server error' });

        }

    }

    static async addProduct(req, res) {

        const product = new Product(req.body.name, req.body.price);

*if* (*await* product.addProduct()) {

            res.status(201).json({ message: 'Product was added' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async editProduct(req, res) {

        const { id } = req.params;

        const product = new Product(req.body.name, req.body.price);

*if* (*await* product.editProduct(id)) {

            res.status(201).json({ message: 'Data was updated' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async deleteProduct(req, res) {

        const { id } = req.params;

*if* (*await* Product.deleteProduct(id)) {

            res.status(201).json({ message: 'Product was deleted' });

        } *else* {

            res.status(403).json({ err: 'Error' });

        }

    }

    static async findProducts(req, res) {

*try* {

            const { name, price } = req.body;

            const products = *await* Product.findProduct(name, price);

            res.status(200).json(products);

        } *catch* (error) {

            res.status(500).json({ err: 'Server error' });

        }

    }

}

module.exports = Products;

# Лістинг модуля controllers/productsControllers/controllers.js

const { Client } = require('pg');

const client = new Client(

    'postgres://postgres:Misha211100@localhost:5432/internetshop'

);

client.connect();

class Product {

    constructor(name, price) {

        this.name = name;

        this.price = price;

    }

    static async getProducts() {

*try* {

            let res = *await* client.query(

                `select productid, name, concat('$',price::numeric) as price from products`

            );

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async addProduct() {

*try* {

*await* client.query(

                `insert into products (name, price) values ('${this.name}','${this.price}')`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async editProduct(id) {

        console.log(this.name, this.price);

*try* {

*await* client.query(

                `update products set name='${this.name}', price='${this.price}' where productid=${id}`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async deleteProduct(id) {

*try* {

*await* client.query(`delete from products where productid=${id}`);

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async findProduct(name, price) {

        let res;

*try* {

*if* (name && price) {

                res = *await* client.query(

                    `select productid, name, concat('$',price::numeric) as price from products where name='${name}' and price='${price}'`

                );

            } *else* *if* (name) {

                console.log(name);

                res = *await* client.query(

                    `select productid, name, concat('$',price::numeric) as price from products where name='${name}'`

                );

                console.log(res.rows);

            } *else* *if* (price) {

                res = *await* client.query(

                    `select productid, name, concat('$',price::numeric) as price from products where price='${price}'`

                );

            }

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

}

module.exports = Product;

# Лістинг модуля models/ordersProductsModel/ ordersProductsModel.js

const { Client } = require('pg');

const client = new Client(

    'postgres://postgres:Misha211100@localhost:5432/internetshop'

);

client.connect();

class OrderProduct {

    constructor(orderid, productid, count) {

        this.orderid = orderid;

        this.productid = productid;

        this.count = count;

    }

    static async getOrdersProducts() {

*try* {

            let res = *await* client.query('select \* from orders\_products');

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async addOrderProduct() {

*try* {

*await* client.query(

                `insert into orders\_products (orderid, productid, count) values (${this.orderid},${this.productid},${this.count})`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async deleteOrderProduct(id) {

*try* {

*await* client.query(

                `delete from orders\_products where orderid=${id}`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async findOrdersProducts(orderid, productid) {

        let res;

*try* {

*if* (orderid && productid) {

                res = *await* client.query(

                    `select \* from orders\_products where orderid='${orderid}' and productid='${productid}'`

                );

            } *else* *if* (orderid) {

                res = *await* client.query(

                    `select \* from orders\_products where orderid='${orderid}'`

                );

            } *else* *if* (productid) {

                res = *await* client.query(

                    `select \* from orders\_products where productid='${productid}'`

                );

            }

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

}

module.exports = OrderProduct;

# Лістинг модуля controllers/ordersProductsControllers/controllers.js

const OrderProduct = require('../../models/ordersProductsModel/ordersProductsModel');

class OrdersProducts {

    static async ordersProductsList(req, res) {

*try* {

            let ordersProducts = *await* OrderProduct.getOrdersProducts();

            res.status(200).json(ordersProducts);

        } *catch* (e) {

            res.status(500).json({ err: 'Server error' });

        }

    }

    static async addOrderProduct(req, res) {

        const orderProduct = new OrderProduct(

            req.body.orderid,

            req.body.productid,

            req.body.count

        );

*if* (*await* orderProduct.addOrderProduct()) {

            res.status(201).json({ message: 'Order and product was added' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async deleteOrderProduct(req, res) {

        const { id } = req.params;

*if* (*await* OrderProduct.deleteOrderProduct(id)) {

            res.status(201).json({ message: 'Order and product was deleted' });

        } *else* {

            res.status(403).json({ err: 'Error' });

        }

    }

    static async findOrdersProducts(req, res) {

*try* {

            const { orderid, productid } = req.body;

            const ordersProducts = *await* OrderProduct.findOrdersProducts(

                orderid,

                productid

            );

            res.status(200).json(ordersProducts);

        } *catch* (error) {

            res.status(500).json({ err: 'Server error' });

        }

    }

}

module.exports = OrdersProducts;

# Лістинг модуля models/reviewModel/reviewModel.js

const { Client } = require('pg');

const client = new Client(

    'postgres://postgres:Misha211100@localhost:5432/internetshop'

);

client.connect();

class Review {

    constructor(text, client\_name, productid) {

        this.text = text;

        this.client\_name = client\_name === undefined ? null : client\_name;

        this.productid = productid;

    }

    static async getReviews() {

*try* {

            let res = *await* client.query('select \* from reviews');

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    async addReview() {

*try* {

            console.log(this.text, this.client\_name, this.productid);

*await* client.query(

                `insert into reviews (text, client\_name, productid) values ('${this.text}','${this.client\_name}',${this.productid})`

            );

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async deleteReview(id) {

*try* {

*await* client.query(`delete from reviews where reviewid=${id}`);

*return* true;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

    static async findReviews(productid) {

        let res;

*try* {

            res = *await* client.query(

                `select \* from reviews where productid='${productid}'`

            );

*return* res.rows;

        } *catch* (e) {

            console.log(e);

*return* false;

        }

    }

}

module.exports = Review;

# Лістинг модуля controllers/reviewsControllers/controllers.js

const Review = require('../../models/reviewModel/reviewModel');

class Reviews {

    static async reviewsList(req, res) {

*try* {

            let reviews = *await* Review.getReviews();

            res.status(200).json(reviews);

        } *catch* (e) {

            res.status(500).json({ err: 'Server error' });

        }

    }

    static async addReview(req, res) {

        const review = new Review(

            req.body.text,

            req.body.client\_name,

            req.body.productid

        );

*if* (*await* review.addReview()) {

            res.status(201).json({ message: 'Review was added' });

        } *else* {

            res.status(403).json({ err: 'Already exist' });

        }

    }

    static async deleteReview(req, res) {

        const { id } = req.params;

*if* (*await* Review.deleteReview(id)) {

            res.status(201).json({ message: 'Review was deleted' });

        } *else* {

            res.status(403).json({ err: 'Error' });

        }

    }

    static async findReviews(req, res) {

*try* {

            const { productid } = req.body;

            const reviews = *await* Review.findReviews(productid);

            res.status(200).json(reviews);

        } *catch* (error) {

            res.status(500).json({ err: 'Server error' });

        }

    }

}

module.exports = Reviews;