МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний університет «Одеська політехніка»

Інститут Комп’ютерних Систем

Кафедра Інформаційних технологій

Протокол модульної роботи №1

Семестр II

з дисципліни «Об’єктно-орієнтовне програмування»

Виконав студент групи

АД-221

Прудкий Д. П.

Прийняв:

Рудниченко М. Д.

Одеса 2024

Зміст

[Теоритична частина 3](#_Toc133359916)

[Практичне завдання 5](#_Toc133359917)

[Лістинг SQL-скрипта 8](#_Toc133359918)

[Скріншоти веб додатку 11](#_Toc133359919)

[Лістинг програми 13](#_Toc133359920)

# Теоритична частина

20. Механізм ін'єкції залежності в Spring є однією з ключових концепцій, яка дозволяє реалізувати принцип інверсії керування (IoC - Inversion of Control) і покращити модульність, перевикористовуваність та тестовість вашого коду. Ін'єкція залежності може відбуватися через конструктор, метод або поля класу. Конструкторна ін'єкція полягає в тому, що Spring використовує конструктор класу для ін'єкції залежностей. Це рекомендований підхід, оскільки він гарантує, що всі залежності будуть ініціалізовані при створенні об'єкта. Методична ін'єкція використовує setter-методи класу для ін'єкції залежностей.

Цей підхід зручний, коли потрібно ініціалізувати залежності після створення об'єкта. Ін'єкція через поля дозволяє ін'єктувати залежності без використання конструктора або setter-методів, прямо в поля класу. Цей підхід вимагає менше коду, але може бути менш гнучким.

Для конфігурації ін'єкції залежностей в Spring ви можете використовувати різні методи: через XML-файли, Java-конфігурацію або анотації. В анотаційному підході, ви можете використовувати анотації, такі як `@Autowired`, `@Inject`, `@Resource`, щоб вказати Spring Container'у, які об'єкти слід ін'єктувати.

Spring Container, такий як ApplicationContext або BeanFactory, завантажує конфігурацію і створює та управляє бінами. Контейнер автоматично ін'єктує залежності в біни, використовуючи інформацію з конфігурації. Він також відповідає за управління життєвим циклом бінів, ініціалізацією та знищенням об'єктів, а також їх залежностями.

30. Об'єкт ServletContext відіграє ключову роль в сервлетному контейнері, представляючи загальний контекст виконання для всього веб-додатка. Цей об'єкт існує один раз для кожного веб-додатка і є доступним для всіх компонентів додатка. ServletContext зберігає важливу інформацію, таку як глобальні параметри, атрибути контексту, ініціалізаційні параметри та інші ресурси, які можна використовувати в різних частинах вашого додатка.

Він також дозволяє взаємодіяти з ресурсами додатка, такими як файлові ресурси, JSP-файли, сервлети, а також забезпечує механізми для комунікації між різними частинами веб-додатка. ServletContext є глобальним об'єктом, який може бути використаний для спільного доступу до ресурсів і даних між різними компонентами веб-додатка.

Request Dispatcher є інтерфейсом, який надає можливість перенаправити HTTP-запит до іншого ресурсу, такого як інший сервлет, JSP-сторінка, або навіть зовнішній URL. Цей механізм є дуже корисним для організації динамічного взаємодії між різними частинами вашого веб-додатка.

За допомогою Request Dispatcher можна передати керування з одного компонента до іншого, наприклад, з одного сервлета до іншого, або з сервлета до JSP-сторінки. Це дозволяє структурувати ваш веб-додаток більш ефективно, забезпечуючи модульність і гнучкість в розробці.

Крім прямого перенаправлення запитів, Request Dispatcher може бути використаний для включення ресурсів в поточну відповідь, що дає можливість створювати повторне використання коду та компонентів, покращуючи таким чином підтримуваність та легкість розробки вашого веб-додатка.

10. LazyLoadingException є однією з типових виняткових ситуацій в Hibernate, яка виникає, коли спроба завантажити відкладено залежну сутність після закриття сесії. Відкладене завантаження є популярною стратегією оптимізації, яка дозволяє зменшити кількість запитів до бази даних, завантажуючи залежні об'єкти лише в той момент, коли вони дійсно потрібні для виконання операцій з ними.

Однак використання цієї стратегії може призвести до LazyLoadingException, якщо залежні об'єкти спробують бути завантажені після закриття сесії. Це може відбутися, наприклад, при серіалізації об'єктів, передачі об'єктів між методами або коли спробуєте отримати доступ до LAZY-завантаженої колекції після закриття сесії.

Щоб уникнути цієї проблеми, існує декілька стратегій:

* Eager Loading (попереднє завантаження): Замість LAZY завантаження, ви можете використовувати EAGER завантаження, яке завантажує всі залежні об'єкти одразу разом з основним об'єктом. Це збільшить кількість запитів до бази даних, але уникне проблеми з LazyLoadingException.
* Відкриття нової сесії: Іноді можна відкрити нову сесію Hibernate для завантаження відкладених об'єктів перед тим, як доступитися до них після закриття попередньої сесії.
* Використання Fetch анотації: Ви можете використовувати анотацію `@Fetch` або `@BatchSize` для явного вказівки стратегії завантаження для конкретних відношень в Hibernate.
* XML-конфігурація: Якщо ви використовуєте XML для конфігурації Hibernate, ви можете вказати стратегію завантаження за допомогою атрибутів, таких як `lazy="true"` або `lazy="false"` в описі відношень.

Кращий підхід залежить від конкретних потреб вашого додатка, обсягу даних та вимог до продуктивності. Важливо збалансувати між кількістю запитів до бази даних та зручністю роботи з об'єктами в вашому додатку.

# Практичне завдання

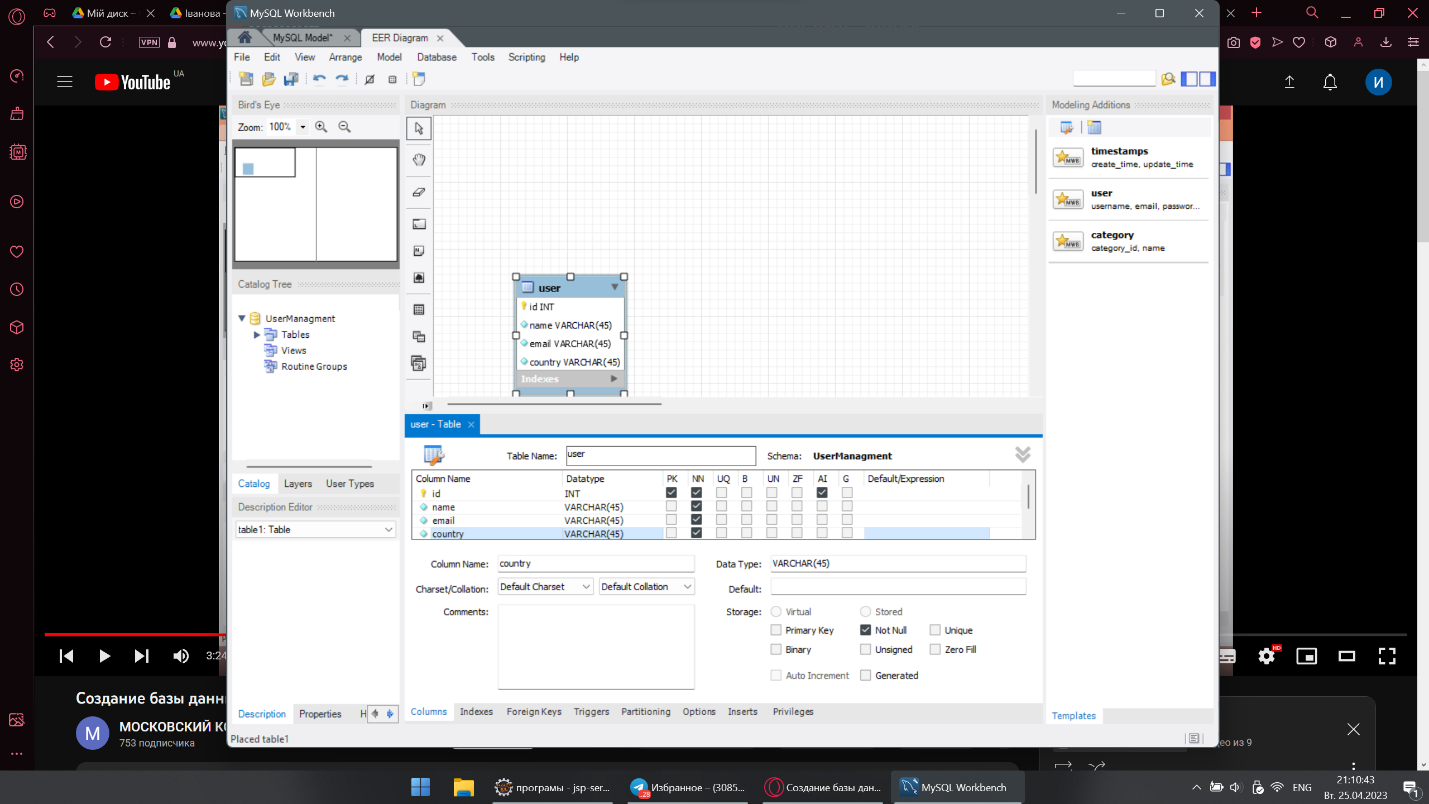
1. Створити БД на 3-5 таблиць у будь-якій зручній СУБД (mysql, postgresql чи ін.)

2. Розробити консольну або веб (бажано) додаток, що реалізує CRUD функціонал роботи з БД. Бажано використовувати hibernate, але можна і просто JDBC

3. Розробити 5 модульних тестів до функціоналу, забезпечити їх коректне виконання

4. Відобразити результати у вигляді скрінів та коду в протоколі

1. Створити БД на 3-5 таблиць у будь-якій зручній СУБД (mysql, postgresql чи ін.)

Створемо модель нашої майбутньої БД у додатку Workbench.

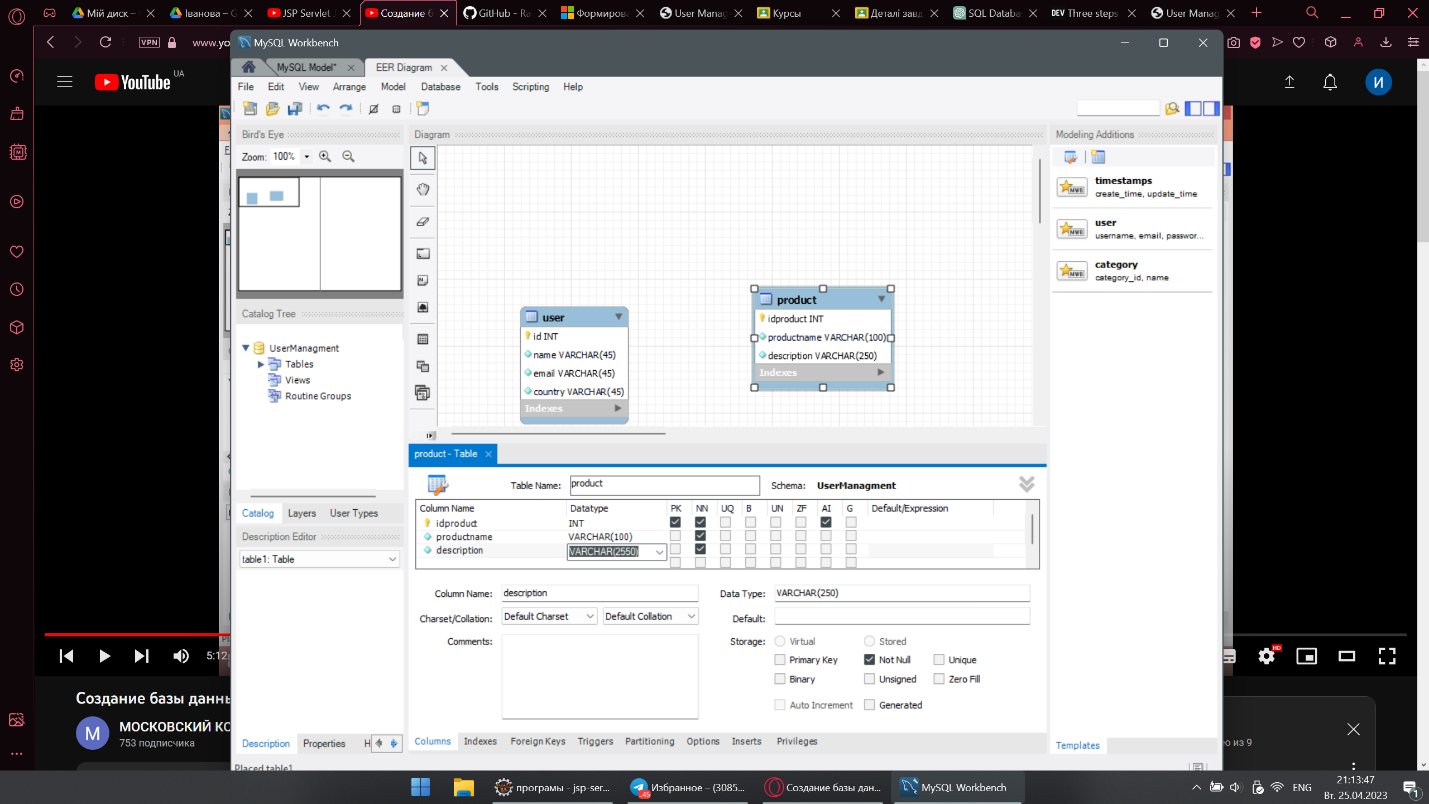
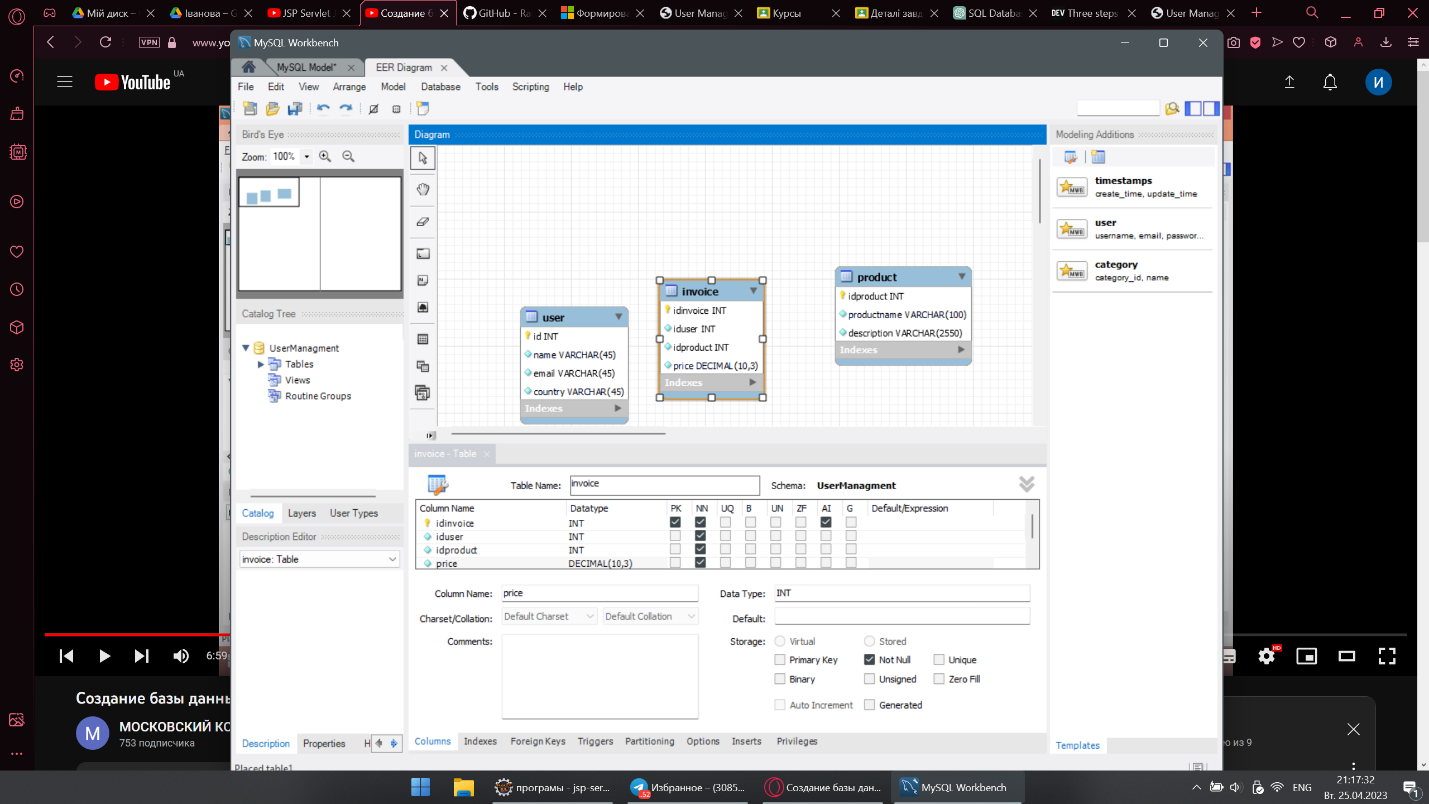
Рисунок 1 — Перша таблиця користувачів

Рисунок 2 — Друга таблиця продуктів

Рисунок 3 — Зв’язна таблиця «Рахунки»

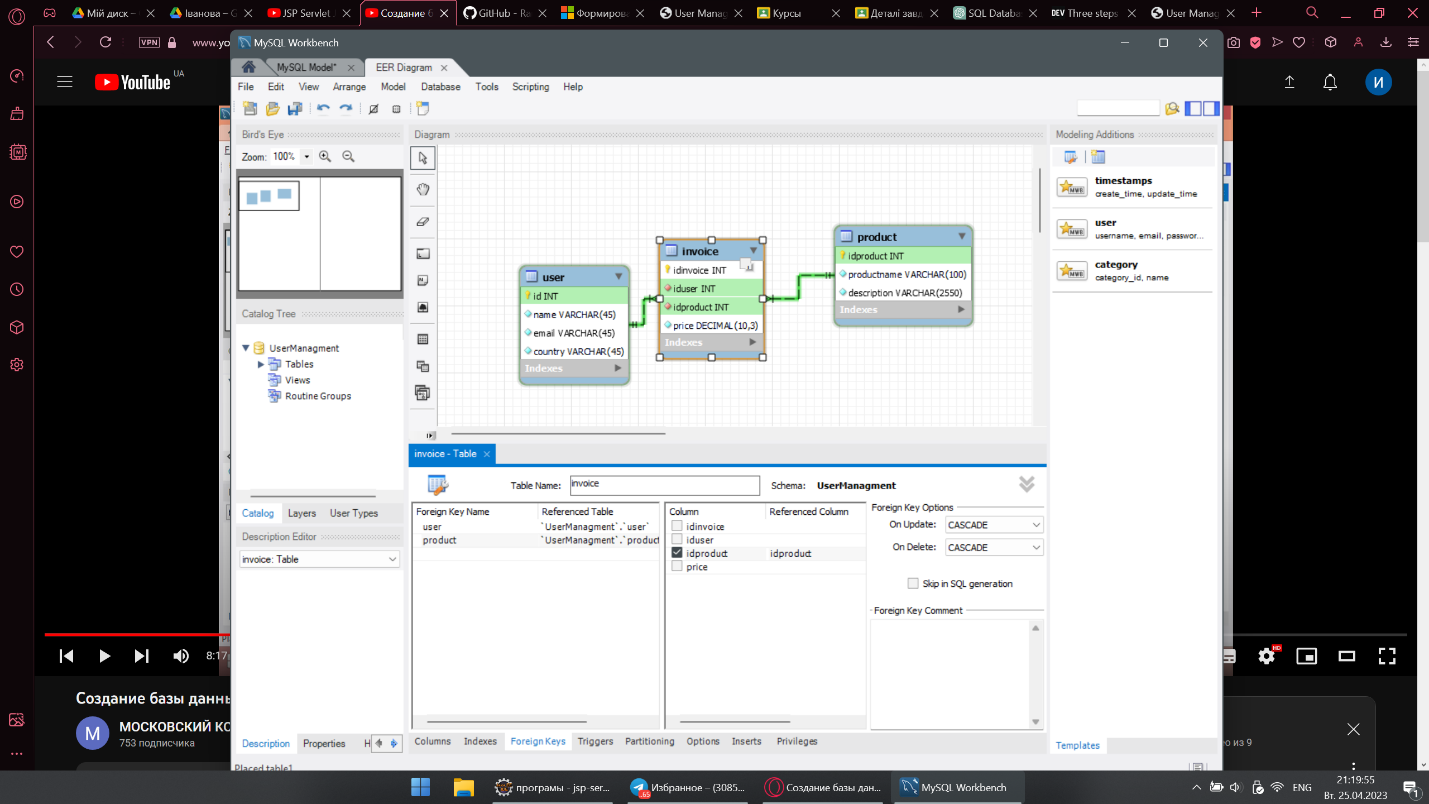
Сформуємо зовнішні ключі таблиці.

Рисунок 4 — Зв’язок таблиць у БД

Отримаємо SQL-скрипт цієї моделі БД.

# Лістинг SQL-скрипта

-- MySQL Script generated by MySQL Workbench

-- Tue Apr 25 21:23:50 2023

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema UserManagment

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema UserManagment

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `UserManagment` DEFAULT CHARACTER SET utf8 ;

USE `UserManagment` ;

-- -----------------------------------------------------

-- Table `UserManagment`.`user`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `UserManagment`.`user` (

`id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`email` VARCHAR(45) NOT NULL,

`country` VARCHAR(45) NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `UserManagment`.`product`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `UserManagment`.`product` (

`idproduct` INT NOT NULL AUTO\_INCREMENT,

`productname` VARCHAR(100) NOT NULL,

`description` VARCHAR(2550) NOT NULL,

PRIMARY KEY (`idproduct`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `UserManagment`.`invoice`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `UserManagment`.`invoice` (

`idinvoice` INT NOT NULL AUTO\_INCREMENT,

`iduser` INT NOT NULL,

`idproduct` INT NOT NULL,

`price` DECIMAL(10,3) NOT NULL,

PRIMARY KEY (`idinvoice`),

INDEX `user\_idx` (`iduser` ASC) VISIBLE,

INDEX `product\_idx` (`idproduct` ASC) VISIBLE,

CONSTRAINT `user`

FOREIGN KEY (`iduser`)

REFERENCES `UserManagment`.`user` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `product`

FOREIGN KEY (`idproduct`)

REFERENCES `UserManagment`.`product` (`idproduct`)

ON DELETE CASCADE

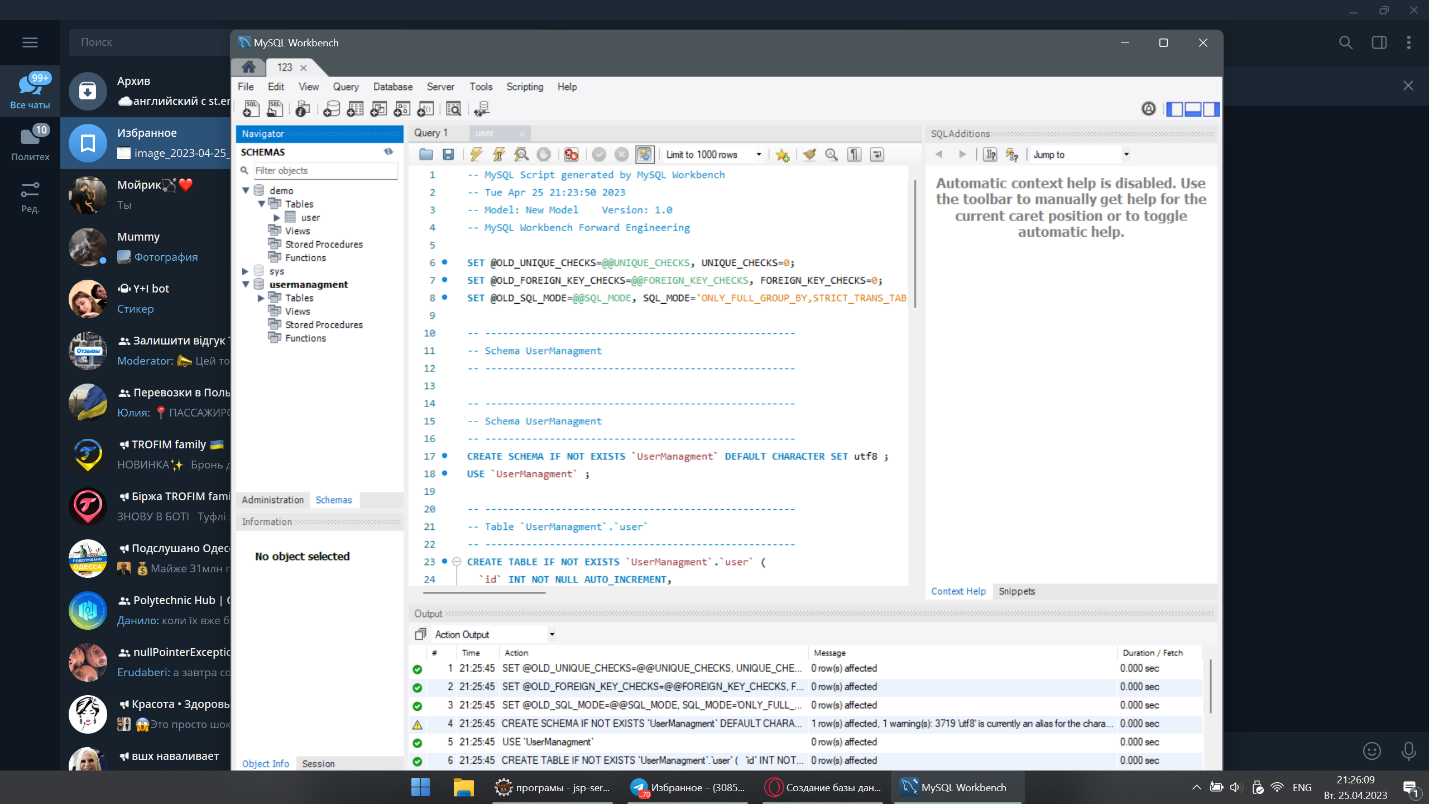
ON UPDATE CASCADE)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

Рисунок 5 — Створена БД

# Скріншоти веб додатку

Розробка веб додатку, що реалізує CRUD функціонал роботи з БД.

Через цю форму ми додаємо користувача до таблиці “user”.

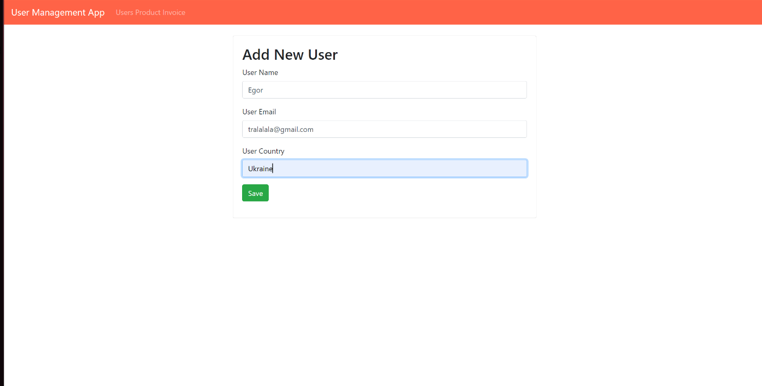


Рисунок 6 — додання користувача

Через цю форму ми взаємодіємо із таблицею “user”, а саме можемо видаляти редагувати користувачів.

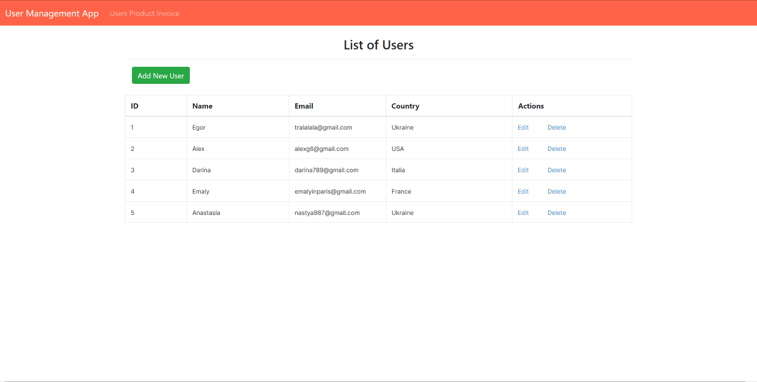
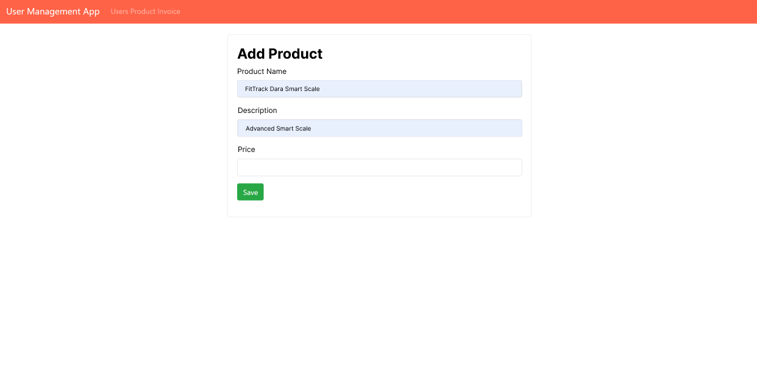


Рисунок 7 — Видалення/редагування користувачів

Через цю форму ми додаємо продукти до таблиці “product”.

 Рисунок 8 — додавання продукту

Через цю форму ми взаємодіємо із таблицею “invoice”, а саме можемо видаляти редагувати продукти та рахунки користувача.

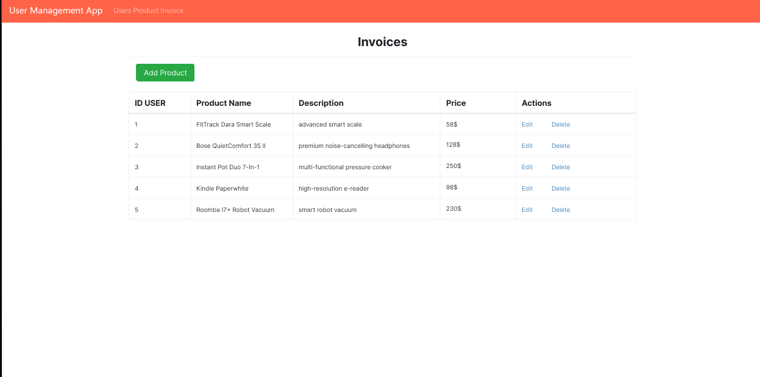


Рисунок 9 — Видалення/редагування продуктів

# Лістинг програми

Цей код являє собою реалізацію DAO-класу (Data Access Object) для роботи з базою даних. Він містить методи для вставки, вибірки, оновлення та видалення записів у таблиці "users" бази даних "demo".

package net.javaguides.usermanagment.dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import java.util.ArrayList;

import net.javaguides.usermanagment.model.User;

/\*\*

\* AbstractDAO.java This DAO class provides CRUD database operations for the

\* table users in the database.

\*

\* @author Ilona Ivanova AD-212

\*

\*/

public class UserDAO {

private String jdbcURL = "jdbc:mysql://localhost:3306/demo?useSSL=false";

private String jdbcUsername = "root";

private String jdbcPassword = "admin";

private static final String INSERT\_USERS\_SQL = "INSERT INTO users" + " (name, email, country) VALUES "

+ " (?, ?, ?);";

private static final String SELECT\_USER\_BY\_ID = "select id,name,email,country from users where id =?";

private static final String SELECT\_ALL\_USERS = "select \* from users";

private static final String DELETE\_USERS\_SQL = "delete from users where id = ?;";

private static final String UPDATE\_USERS\_SQL = "update users set name = ?,email= ?, country =? where id = ?;";

public UserDAO() {

}

protected Connection getConnection() {

Connection connection = null;

try {

Class.forName("com.mysql.jdbc.Driver");

connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return connection;

}

public void insertUser(User user) throws SQLException {

System.out.println(INSERT\_USERS\_SQL);

// try-with-resource statement will auto close the connection.

try (Connection connection = getConnection();

PreparedStatement preparedStatement = connection.prepareStatement(INSERT\_USERS\_SQL)) {

preparedStatement.setString(1, user.getName());

preparedStatement.setString(2, user.getEmail());

preparedStatement.setString(3, user.getCountry());

System.out.println(preparedStatement);

preparedStatement.executeUpdate();

} catch (SQLException e) {

printSQLException(e);

}

}

public User selectUser(int id) {

User user = null;

// Step 1: Establishing a Connection

try (Connection connection = getConnection();

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(SELECT\_USER\_BY\_ID);) {

preparedStatement.setInt(1, id);

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

ResultSet rs = preparedStatement.executeQuery();

// Step 4: Process the ResultSet object.

while (rs.next()) {

String name = rs.getString("name");

String email = rs.getString("email");

String country = rs.getString("country");

user = new User(id, name, email, country);

}

} catch (SQLException e) {

printSQLException(e);

}

return user;

}

public List<User> selectAllUsers() {

// using try-with-resources to avoid closing resources (boiler plate code)

List<User> users = new ArrayList<>();

// Step 1: Establishing a Connection

try (Connection connection = getConnection();

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(SELECT\_ALL\_USERS);) {

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

ResultSet rs = preparedStatement.executeQuery();

// Step 4: Process the ResultSet object.

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String email = rs.getString("email");

String country = rs.getString("country");

users.add(new User(id, name, email, country));

}

} catch (SQLException e) {

printSQLException(e);

}

return users;

}

public boolean deleteUser(int id) throws SQLException {

boolean rowDeleted;

try (Connection connection = getConnection();

PreparedStatement statement = connection.prepareStatement(DELETE\_USERS\_SQL);) {

statement.setInt(1, id);

rowDeleted = statement.executeUpdate() > 0;

}

return rowDeleted;

}

public boolean updateUser(User user) throws SQLException {

boolean rowUpdated;

try (Connection connection = getConnection();

PreparedStatement statement = connection.prepareStatement(UPDATE\_USERS\_SQL);) {

statement.setString(1, user.getName());

statement.setString(2, user.getEmail());

statement.setString(3, user.getCountry());

statement.setInt(4, user.getId());

rowUpdated = statement.executeUpdate() > 0;

}

return rowUpdated;

}

private void printSQLException(SQLException ex) {

for (Throwable e : ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

}

Це клас Java, що представляє сутність користувача. Він має чотири змінні екземпляра.

Клас містить три конструктори: конструктор за замовчуванням, конструктор із параметрами імені, електронної пошти та країни та конструктор із параметрами ідентифікатора, імені, електронної пошти та країни.

Клас також надає методи отримання та встановлення для кожної змінної екземпляра. Ці методи дозволяють отримати доступ до змінних екземплярів, одночасно забезпечуючи інкапсуляцію.

package net.javaguides.usermanagment.model;

public class User {

protected int id;

protected String name;

protected String email;

protected String country;

public User() {

}

public User(String name, String email, String country) {

super();

this.name = name;

this.email = email;

this.country = country;

}

public User(int id, String name, String email, String country) {

super();

this.id = id;

this.name = name;

this.email = email;

this.country = country;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}}

Цей код являє собою сервлет, який обробляє HTTP-запити і здійснює взаємодію з базою даних. Він містить методи отримання списку користувачів, додавання, редагування та видалення користувачів.

package net.javaguides.usermanagment.web;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import net.javaguides.usermanagment.dao.UserDAO;

import net.javaguides.usermanagment.model.User;

/\*\*

\* Servlet implementation class UserServlet

\*/

@WebServlet("/")

public class UserServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private UserDAO userDAO;

public void init() {

userDAO = new UserDAO();

}

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String action = request.getServletPath();

try {

switch (action) {

case "/new":

showNewForm(request, response);

break;

case "/insert":

insertUser(request, response);

break;

case "/delete":

deleteUser(request, response);

break;

case "/edit":

showEditForm(request, response);

break;

case "/update":

updateUser(request, response);

break;

default:

listUser(request, response);

break;

}

} catch (SQLException ex) {

throw new ServletException(ex);

}

}

private void listUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException, ServletException {

List<User> listUser = userDAO.selectAllUsers();

request.setAttribute("listUser", listUser);

RequestDispatcher dispatcher = request.getRequestDispatcher("user-list.jsp");

dispatcher.forward(request, response);

}

private void showNewForm(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

RequestDispatcher dispatcher = request.getRequestDispatcher("user-form.jsp");

dispatcher.forward(request, response);

}

private void showEditForm(HttpServletRequest request, HttpServletResponse response)

throws SQLException, ServletException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

User existingUser = userDAO.selectUser(id);

RequestDispatcher dispatcher = request.getRequestDispatcher("user-form.jsp");

request.setAttribute("user", existingUser);

dispatcher.forward(request, response);

}

private void insertUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException {

String name = request.getParameter("name");

String email = request.getParameter("email");

String country = request.getParameter("country");

User newUser = new User(name, email, country);

userDAO.insertUser(newUser);

response.sendRedirect("list");

}

private void updateUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

String name = request.getParameter("name");

String email = request.getParameter("email");

String country = request.getParameter("country");

User updateUser = new User(id, name, email, country);

userDAO.updateUser(updateUser);

response.sendRedirect("list");

}

private void deleteUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

userDAO.deleteUser(id);

response.sendRedirect("list");

}}

Посилання на репозиторій-