

ДОСЛІДЖЕННЯ ПРИНЦИПІВ ЗАСТОСУВАННЯ HIBERNATE ДЛЯ ДОСТУПУ ДО ДАНИХ У ВЕБ-ЗАСТОСУНКАХ

Мета

Основна мета – навчитись створювати веб-застосунок, що відображає на веб-сторінці інформацію, яка міститься у таблиці бази даних, а також забезпечує виконання операцій CRUD із використанням Hibernate.

1 Набуття навиків підключення БД до Java-проекту та реалізації CRUD із застосування технології об'єктно-реляційного маппінгу із застосуванням класів та методів бібліотеки Hibernate.

2 Вдосконалення навичків створення веб-сторінок на основні технології Java Server Pages (JSP) та застосування Java Standart Tag Library (JSTL) при їх розробці.

3 Вдосконалення навиків розробки user frendly applications.

4 Вдосконалення навиків створення сервлетів, враховуючи особливості використання даних.

5 Набуття навиків реалізації контролю коректності даних, що вводяться до бази даних.

Завдяки виконанню даного проекту, учасники наберуться практичного досвіду в розробці веб-додатків на Java, опанують роботу з базами даних через Hibernate, вдосконалять навички створення динамічних веб-сторінок та реалізації CRUD-операцій. В результаті буде створено функціональний веб-застосунок, що відповідає сучасним стандартам якості та зручності використання.

Цей проект стане відправною точкою для подальшої роботи у сфері веб-розробки та програмування на Java, закладаючи основи для майбутніх успіхів у цій галузі.

Завдання роботи

Проект розроблюється відповідно до варіанту 6, визначеного викладачем. Опис структури полів класу „OfficeWorker“ зображено на рис. 1.

Варіант №6 (HR-2)

Опис структури полів класу «OfficeWorker» (суттєвість 2)

Ім'я поля	Тип даних	Опис	Властивості поля
id	Long	Сурогатний ключ	PK
surname	String	Прізвище	Not null
name	String	Ім'я	Not null
pname	String	По-батькові	Not null
startWork	LocalDate	Дата початку роботи	Not null, «дд.мм.рррр»
endWork	LocalDate	Дата звільнення	«дд.мм.рррр», більша за дату початку не менш ніж 4 місяці
workerCod	String	Корпоративний код співробітника	4 цифри, можуть бути 0 на будь-якій позиції

Рисунок 1 – Структура полів класу „OfficeWorker“

Завдання охоплює наступні ключові етапи.

1 Створення веб-застосунку:

- проект має бути реалізований як Maven-проект, що використовує технології: JDK 17, Jakarta EE 9, Apache Tomcat 10.1, Hibernate 6, MySQL 8;
- налаштувати файл конфігурації pom.xml для підключення всіх необхідних бібліотек та залежностей (JSP, JSTL, JDBC, Hibernate).

2 Реалізація CRUD-операцій:

- забезпечити функціонал створення, читання, редагування та видалення (CRUD) записів у базі даних;
- створити Entity-клас для відображення таблиці бази даних;
- реалізувати класи DAO для виконання операцій із записами бази даних за допомогою Hibernate.

3 Інтерфейс користувача (UI):

- створити JSP-сторінки для відображення даних з таблиці бази даних;
- сторінки мають дозволяти додавання нових записів, редагування та видалення існуючих записів;

- забезпечити використання JSTL та Expression Language (EL) для створення динамічного веб-інтерфейсу без використання Java-скриптів у JSP;
- реалізувати сервіси та сервлети для обробки запитів користувача і відправки даних між базою даних та інтерфейсом.

4 Налаштування бази даних:

- створити схему бази даних із таблицями для збереження інформації;
- використовувати Hibernate для автоматичного створення та оновлення таблиць;
- налаштувати підключення до бази даних MySQL через Hibernate (конфігураційний файл hibernate.cfg.xml).

5 Тестування та валідація:

- перевірити роботу CRUD-операцій за допомогою тестів та реальних даних;
- забезпечити перевірку наявності дублюючих записів у базі даних, а також коректність введених користувачем даних;
- реалізувати обробку помилок та виключень (наприклад, помилки 404, втрати зв'язку з базою даних).

6 Додаткові функціональні можливості:

- реалізувати пошук, сортування та фільтрацію даних за певними полями;
- створити форми для додавання та редагування записів з попередньою перевіркою даних.

Хід роботи

1 Завдання 1. Створення проєкту для роботи із базою даних із використанням бібліотеки реляційного відображення об'єктів Hibernate, створення відображення «застосунок - база даних».

1.1 Створення проєкту

- 1 Проект веб-застосунку створити як MAVEN проєкт.

2 В ході його створення необхідно додати теги залежностей (dependency) для підключення до проекту бібліотек драйверу jdbc для СКБД MySQL, Servlets API, JSP, JSTL.

3 В Додатку А представляється файл конфігурації MAVEN-проєкту pom.xml, набір залежностей якого визначає стек технологій розробки:

- jdk17;
- jakarta EE 9;
- Apache Tomcat 10.1;
- Servlets API 6.0;
- JSTL 3.00;
- Lombok;
- Junit;
- database management system MySQL 9;
- Hibernate 6;
- Java Persistence API 3.

1.2 Створення структури проєкту

Перелік пакетів проєкту віглядає наступним чином:

- entities – для зберігання класів моделі: OfficeWorker, OfficeWorkerList;
- servlets – для зберігання сервлетів: AddEditOfficeWorkerServlet, CheckLoginServlet, DeleteOfficeWorkerServlet, ShowOfficeWorkerServlet;
- hibernate – для зберігання класів, що забезпечують підключення застосунку до БД та отримання даних із класів: HibernateUtil;
- enums – для зберігання класів, що забезпечують енумерацію, що дозволяє визначати набір попередньо оголошених констант: Messages, OfficeWorkerStatus, а також конвертер OfficeWorkerStatusConverter, що служить для перетворення значень enum OfficeWorkerStatus у відповідне значення для зберігання в базі даних та, навпаки, для перетворення значення з бази даних назад у значення enum;

– daohbn – для зберігання класу DAOOfficeWorker, який є реалізацією шаблону Data Access Object (DAO), що використовується для управління доступом до даних у базі даних за допомогою технології Hibernate. Основне завдання цього класу — забезпечити роботу з сутністю OfficeWorker, використовуючи методи для виконання CRUD-операцій.

Структура розробленого проекту представлена на рис.2.

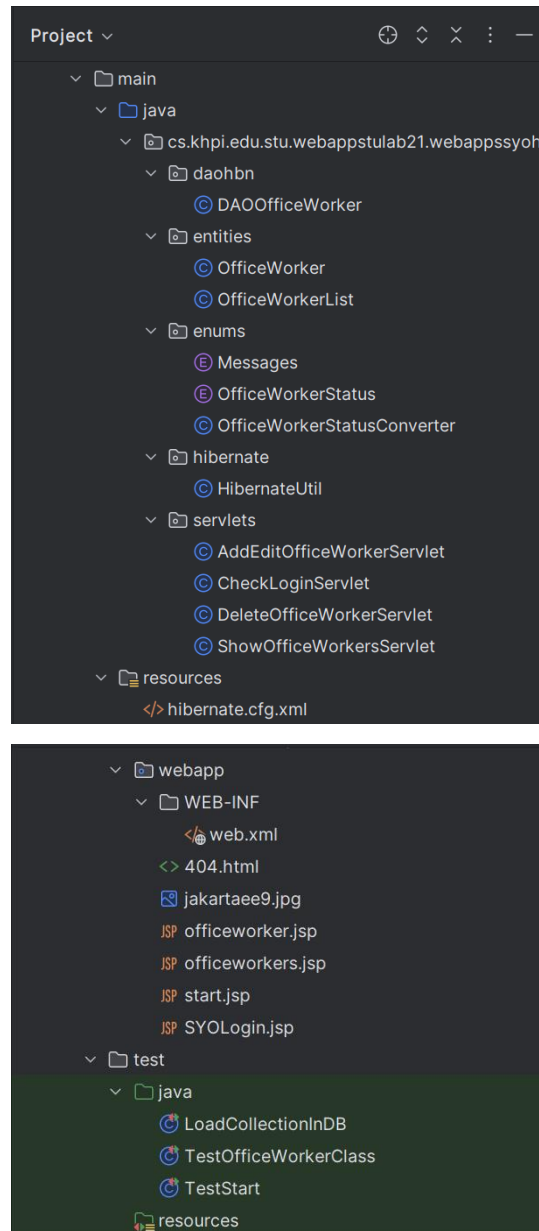


Рисунок 2 – Структура проекту

1.3 Створення схеми бази даних

1 Вибір системи керування базами даних (СКБД):

– для реалізації даного проекту обрана система керування базами даних MySQL 8, яка є популярною СКБД з відкритим вихідним кодом і добре інтегрується з Hibernate.

2 Створення схеми бази даних:

– для реалізації бази даних використовується наступна структура таблиці officeworkers, яка зображена на рисунку 3.

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
end_work	date	YES		NULL	
name	varchar(100)	NO		NULL	
officeworker_status	enum('intern','junior','middle','senior','lead')	NO		NULL	
pname	varchar(100)	NO		NULL	
start_work	date	NO		NULL	
surname	varchar(100)	NO		NULL	
worker_cod	varchar(255)	NO	UNI	NULL	

Рисунок 3 – Структура таблиці officeworkers

3 Створення конфігураційного файлу Hibernate (hibernate.cfg.xml):

– для підключення до бази даних MySQL через Hibernate необхідно створити файл конфігурації hibernate.cfg.xml. У цьому файлі задаються параметри підключення до бази даних, тип драйвера, діалект SQL та інші важливі налаштування.

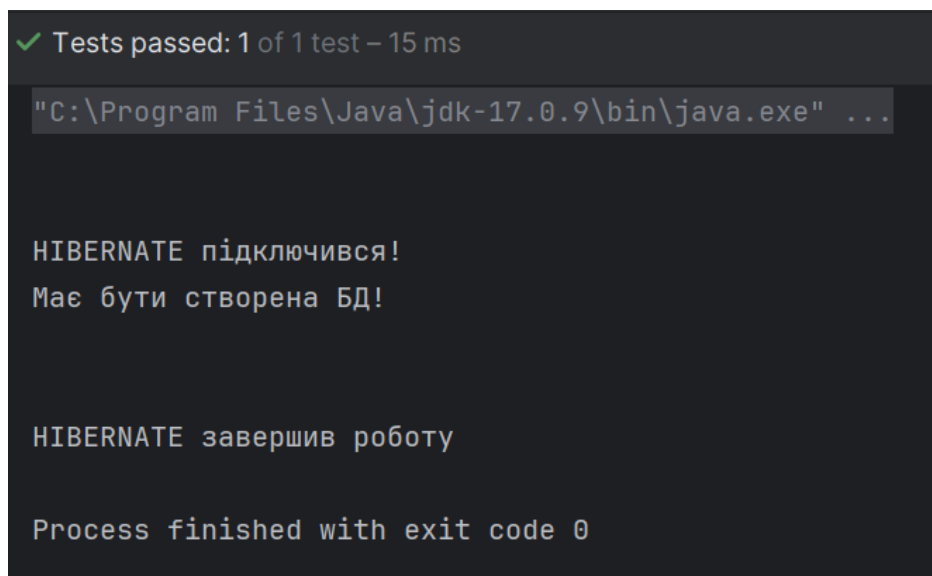
4 Створення класу для управління підключенням до бази даних (HibernateUtil.java):

– для ефективного управління сесіями Hibernate, створюється клас HibernateUtil.java. Цей клас забезпечує інкапсуляцію логіки підключення до БД та надає можливість отримати сесію для виконання запитів.

5 Тестування підключення до бази даних:

– для перевірки коректності підключення та роботи з Hibernate написати тестовий клас TestStart, який зображено в Додатку Б.

Результат виконання тесту зображено на рисунку 4.



```

✓ Tests passed: 1 of 1 test – 15 ms

"C:\Program Files\Java\jdk-17.0.9\bin\java.exe" ...

HIBERNATE підключився!
Має бути створена БД!

HIBERNATE завершив роботу

Process finished with exit code 0

```

Рисунок 4 – Результат виконання тесту TestStart

Завдання 2. Реалізація Entity-класу, CRUD-операцій та їх тестування

2.1 Створення Entity-класу, відображення "клас – таблиця бази даних"

1 Описати клас OfficeWorker, який зображено в Додатку В, як Entity, використовуючи анотації Java Persistence API (JPA).

2 Використати анотації @Entity, @Table та поля зі специфікацією колонок (анотації @Column).

3 У полі, яке представляє первинний ключ, застосувати анотацію @Id та стратегію генерації ключа за допомогою @GeneratedValue(strategy = GenerationType.IDENTITY).

4 Додати обмеження через Hibernate: обмеження на дату завершення роботи @Check(constraints = "end_work >= start_work + INTERVAL 4 MONTH"), обмеження на довжину коду робітника @Size(min = 4, max = 4, message = "Код працівника має складатися з 4 цифр.") @Pattern(regex = "[0-9]{4}", message = "Код працівника повинен складатися з 4 цифр.").

5 Використати enum `OfficeWorkerStatus`, що зображено в Додатку Г, для поля, яке представляє статус працівника, та створити адаптер `OfficeWorkerStatusConverter` (Додаток Д) для збереження цього поля у базі даних.

2.2 Включення класу у конфігурацію Hibernate

Додати мапінг класу у файл `hibernate.cfg.xml`:

```
<mapping class="cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities.OfficeWorker"/>
```

2.3 Створення DAO-класу для операцій CRUD

Реалізувати типові методи CRUD для класу `OfficeWorker`. Методи повинні включати операції вставки, пошуку, оновлення та видалення. Клас `DAOOfficeWorker` знаходиться в Додатку Е.

2.3 Тестування CRUD-методів

Створити метод `testCRUD` (Додаток К) в класі `TestStart`, який тестує всі основні операції CRUD (створення, читання, оновлення, видалення) для об'єкта `OfficeWorker` через DAO-шар з використанням `Hibernate`.

1 Створити працівника (Create):

- створити новий об'єкт `OfficeWorker` і задати його основні атрибути, такі як прізвище, ім'я, по-батькові, дата початку і завершення роботи, код працівника і статус;

- за допомогою методу `DAOOfficeWorker.insert()` працівник додається в базу даних;

- тест перевіряє, що повідомлення після вставки відповідає очікуваному успіху (перевірка через `assertEquals`).

2 Отримання працівника (Read):

- метод `DAOOfficeWorker.getOfficeWorkerByKeyset()` використовується для отримання працівника з бази даних за унікальними ключовими полями;

– перевіряється, що працівник був знайдений і що його ім'я і прізвище збігаються з оригінальними даними (перевірка через `assertEquals()`).

3 Оновлення працівника (Update):

– ім'я працівника змінюється з "John" на "Jane";

– метод `DAOOfficeWorker.update()` використовується для оновлення запису в базі, і перевіряється, що відповідне повідомлення про оновлення повернулося успішно;

– дані після оновлення перевіряються через `DAOOfficeWorker.getOfficeWorkerById()`, щоб переконатися, що ім'я змінилося на "Jane".

4 Видалення працівника (Delete):

– метод `DAOOfficeWorker.delete()` використовується для видалення працівника з бази даних;

– перевіряється, що повідомлення про успішне видалення було отримано;

– після видалення перевіряється, що працівника більше немає в базі даних (результат має бути `null`).

Результат тестування зображено на рисунку 5.

```

✓ Tests passed: 2 of 2 tests - 365 ms

"C:\Program Files\Java\jdk-17.0.9\bin\java.exe" ...

HIBERNATE підключився!
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: insert into officeworkers (end_work,name,officeworker_status,pname,start_work,sur
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: update officeworkers set end_work=?,name=?,officeworker_status=?,pname=?,start_wo
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Hibernate: delete from officeworkers where id=?
Hibernate: select ow1_0.id,ow1_0.end_work,ow1_0.name,ow1_0.officeworker_status,ow1_0.pname,o
Має бути створена БД!

```

Рисунок 5 – Результат тестування

3 Завдання 3. Виведення переліку записів з бази даних на вебсторінку

3.1 Розробка сторінки JSP для відображення даних з таблиці бази даних

JSP-сторінка `officeworkers.jsp` (Додаток Л), яка відображає всі записи з таблиці "officeworkers", реалізована за допомогою JSTL та EL. Це дозволяє уникнути використання скриплетів і підвищити читабельність коду.

Принципи реалізації функцій фільтрації, сортування та пошуку в даному JSP-коді описано нижче.

1 Принцип фільтрації на основі вибраного статусу співробітника:

- вибір статусу: у HTML є випадаючий список `<select>` з різними статусами співробітників;
- обробка події: коли змінюється вибір статусу (подія `onchange`), викликається JavaScript-функція `filterByPL()`;
- отримується вибране значення з випадаючого списку;
- перебираються всі рядки таблиці (крім заголовка);
- для кожного рядка перевіряється статус співробітника: якщо статус співробітника відповідає вибраному або якщо вибрано "all", рядок залишається видимим, в іншому випадку, рядок приховується.

2 Принципи сортування даних в таблиці за обраним критерієм (за прізвищем, ім'ям, по-батькові або датою початку роботи):

- кнопки для сортування: у HTML є кнопки для сортування по різних колонках таблиці;
- обробка події: коли натискається кнопка, викликається відповідна JavaScript-функція (`sortTableByIndex` для текстових полів або `sortTableByDate` для дат);
- логіка сортування: використовується алгоритм бульбашкового сорту (`bubble sort`) для упорядкування рядків таблиці, перебираються всі рядки таблиці, порівнюються значення в обраній колонці.
- якщо значення у рядку нижче за значення у наступному рядку, рядки обмінюються місцями, процес триває, поки всі рядки не будуть правильно впорядковані.

3 Принципи пошуку співробітників за прізвищем за допомогою текстового поля: Текстове поле: У HTML є текстове поле `<input>` для введення прізвища.

- обробка події: використовується подія `onkeyup`, яка викликає JavaScript-функцію `tableSearch()` при кожному натисканні клавіші;
- логіка пошуку: отримується текст, введений користувачем, і перетворюється на нижній регістр для порівняння;
- перебираються всі рядки таблиці, і для кожного рядка перевіряється, чи містить прізвище введену строку, якщо прізвище містить введену строку, рядок залишається видимим, в іншому випадку – приховується.

3.2 Розробка сервлету для отримання даних та їх пересилання на `officeworkers.jsp`

Сервлет `ShowOfficeWorkersServlet` (Додаток М), який відповідає за отримання даних з бази даних і передачу їх на JSP-сторінку. Сервлет обробляє GET-запит і викликає DAO для отримання списку працівників.

4 Завдання 4. Створення форми для додавання та редагування записів

4.1 Створення форми для додавання та редагування записів

Використати одну спільну форму для додавання та редагування записів `officeworker.jsp` (Додаток Н). Основний метод — це перевірка наявності ідентифікатора (ID) запису, який редагуємо.

Основні етапи реалізації описані нижче.

1 Передача параметра ID: коли користувач переходить на форму для редагування, ID запис передається в URL. При додаванні нового запису ID не передається.

2 Перевірка ID у сервлеті: у сервлеті, який обробляє запит на відображення форми, перевіряється, чи є параметр ID. Якщо він присутній, це

означає, що користувач в режимі редагування; якщо ні, то він в режимі додавання.

3 Відображення даних: У JSP-формі можна динамічно відобразити заголовок, поля вводу та значення в залежності від того, в якому режимі користувач знаходиться.

4.2 Розробка сервлету для обробки додавання та редагування записів

Сервлет `AddEditOfficeWorkerServlet` (Додаток П), який відповідає за обробку запитів на додавання нових записів та редагування існуючих у таблиці "officeworkers". Сервлет реалізує два основних методи: `doGet` для відображення форми та `doPost` для обробки введених даних.

Сервлет має дві основні функції, які представлені нижче.

1 Метод `doGet`: використовується для отримання даних з бази даних. Якщо передано ідентифікатор працівника (`id_worker`), сервлет завантажує дані для редагування; якщо ідентифікатор відсутній, створюється новий об'єкт `OfficeWorker` для додавання нового запису.

2 Метод `doPost`: обробляє дані, отримані з форми. У залежності від того, чи є `id_worker`, сервлет виконує операцію вставки або оновлення.

3 Сервлет обробляє як додавання нових записів, так і редагування існуючих. Використовується один сервлет, який в залежності від отриманих даних визначає, чи це новий запис, чи редагування існуючого.

5 Завдання 5. Реалізація вилучення обраного запису із запитом на підтвердження вилучення

5.1 Розробка сервлету `DeleteOfficeWorkerServlet`

Сервлет `DeleteOfficeWorkerServlet` (Додаток Р) реалізує механізм видалення запису співробітника з бази даних. Основні принципи роботи:

1 Обробка запитів:

– `doGet`: отримує `id_worker` з запиту, перевіряє, чи користувач підтверджує видалення;

– doPost: виконує видалення співробітника, використовуючи метод delete з DAO.

2 Взаємодія з DAO:

– сервлет викликає метод DAOOfficeWorker.delete(idDel) для видалення запису з бази даних і отримує результат (успіх або помилка).

3 Обробка результатів:

– у випадку помилки, сервлет перенаправляє на список співробітників з повідомленням про помилку.

– якщо видалення успішне, здійснюється перенаправлення на список співробітників.

4 Підтвердження дії:

– перш ніж відправити запит, JavaScript-функція confirmation() сторінки officeworkers.jsp викликається, коли користувач натискає кнопку видалення;

– ця функція виводить вікно підтвердження, де користувач має можливість підтвердити або скасувати дію.

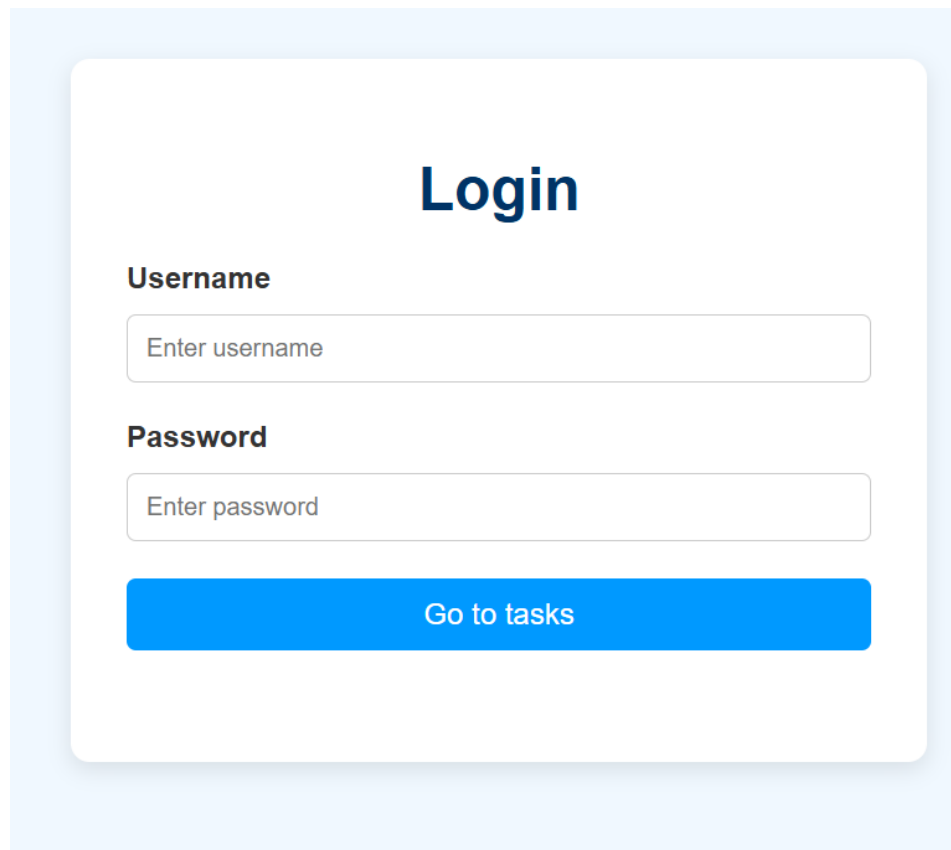
6 Завдання 6. Створення інтерфейсу для керування відображенням даних та обробка переходу на неіснуючу сторінку

1 Реалізація сортування, пошуку та фільтрації рядків здійснюється на сторінці officeworkers.jsp (Додаток Л). Принципи роботи описані в завданні 3.

2 Код файлу 404.html, що є неіснуючою сторінкою (404) знаходиться в Додатку С.

7 Скріншоти веб-сторінок, відкритих в браузері

1 Сторінка для введення логіну та паролю зображена на рисунку 6. Стартова сторінка, на яку переходить користувач після правильного введення логіну і паролю на рисунку 7.



The image shows a login form titled "Login" in a large, bold, dark blue font. Below the title, there are two input fields. The first is labeled "Username" in bold black text, and the second is labeled "Password" in bold black text. Both input fields have a light gray border and contain the placeholder text "Enter username" and "Enter password" respectively. Below the password field is a solid blue button with the text "Go to tasks" in white.

Рисунок 6 – Сторінка для введення логіну та паролю

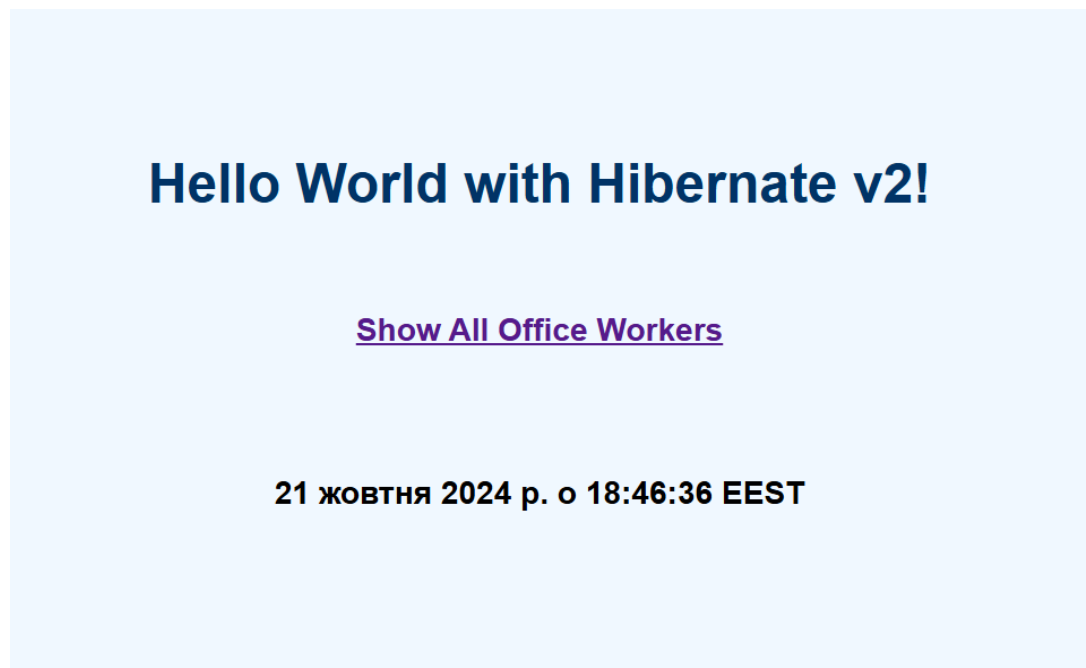


Рисунок 7 – Стартова сторінка

2 Сторінка, на якій відображені всі дані таблиці робітників, зображені на рисунку 8.

To START

Office Workers

Filter by status:

All

Sort by:

Surname

Start Date

Search by surname:

Search...

Reset

INSERT OFFICE WORKER

Surname	Name	Patronymic name	Date of start work	Date of end work	Status	WorkerCode	Actions	
Borysenko	Dariia	Volodymyrivna	04-06-2018		Senior	0001	UPDATE	DELETE
Shevchenko	Ivan	Mykhailovych	15-03-2020	12-09-2023	Junior	0002	UPDATE	DELETE
Kovalenko	Olena	Petrivna	10-06-2019	20-08-2022	Middle	1012	UPDATE	DELETE
Boiko	Andriy	Oleksandrovych	01-07-2021		Intern	0333	UPDATE	DELETE

Melnyk	Oksana	Ivanivna	12-11-2018	15-05-2021	Junior	0010	UPDATE	DELETE
Lysenko	Volodymyr	Maksymovych	09-01-2018		Senior	1254	UPDATE	DELETE
Oliynyk	Inna	Hryhorivna	21-04-2020	23-02-2023	Intern	2001	UPDATE	DELETE
Kravchenko	Dmytro	Sergiyovych	17-12-2019		Lead	0140	UPDATE	DELETE
Petrenko	Ansatasiia	Stepanivna	15-07-2021	22-06-2024	Senior	6539	UPDATE	DELETE
Kuzmenko	Artem	Oleksiyovych	23-04-2015	20-01-2020	Middle	7359	UPDATE	DELETE
Mykolaichuk	Viktor	Vasylyovych	01-01-2021		Junior	0923	UPDATE	DELETE
Shevchenko	Oleksandr	Ivanovych	21-07-2024		Intern	1234	UPDATE	DELETE
Petrenko	Mariya	Andriyivna	21-07-2022		Junior	2345	UPDATE	DELETE
Koval	Mykola	Petrovych	29-05-2021		Middle	3456	UPDATE	DELETE
Boyko	Olha	Andriyivna	17-03-2023	21-09-2024	Senior	4567	UPDATE	DELETE

Рисунок 8 – Сторінка з таблицею робітників


3 Сторінка для редагування даних робітника зображена на рисунку 9.


Edit Office Worker HBN Proc Java 24

Surname:

Name:

Patronymic name:

Date of start work:
 

Date of end work (Optional):
 

Worker Code (4 digits, can start with zero):

Рисунок 9 – Сторінка редагування даних робітника


4 Сторінка для додавання запису нового робітника зображена на рисунку 10.


HBN Proc Java 24

Surname:

Name:

Patronymic name:

Date of start work:
 

Date of end work (Optional):
 

Worker Code (4 digits, can start with zero):

Office Worker Status:

Рисунок 10 – Сторінка для додвання запису нового робітника

Висновки

Проведення лабораторної роботи з розробки веб-застосунку на основі Java, Hibernate та MySQL виявилось ключовим етапом у професійній підготовці. Отримані результати підтверджують важливість глибокого вивчення сучасних технологій, що використовуються у веб-розробці. Зокрема, знайомство з концепцією об'єктно-реляційного маппінгу, реалізованою в Hibernate, значно полегшує роботу з базами даних, дозволяючи програмістам зосередитися на бізнес-логіці замість рутинних SQL-запитів.

Вивчення Hibernate надає переваги, які стають очевидними в процесі розробки. Зокрема, автоматизація управління зв'язками між таблицями та використання анотацій значно скорочують обсяг коду, який потрібно написати, що веде до зменшення можливих помилок. Наприклад, завдяки використанню анотацій `@Entity`, `@Table`, `@Id` і інших, налаштування взаємозв'язків між об'єктами моделі та таблицями бази даних стає інтуїтивно зрозумілим. Однак, незважаючи на ці переваги, етап налаштування конфігураційних файлів і ознайомлення з принципами роботи ORM (Object-Relational Mapping) може бути складним, особливо для тих, хто не має попереднього досвіду у цій галузі.

На відміну від традиційного підходу до роботи з базами даних через JDBC, використання Hibernate зменшує ймовірність помилок, які можуть виникнути під час ручного написання SQL-запитів. Це дозволяє зосередитися на розробці та впровадженні нових функцій, знижуючи ризики, пов'язані з людським фактором. Водночас, складність вивчення зазначених технологій виявилася помірною. Ті, хто володіє базовими знаннями Java та SQL, швидше адаптуються до нових інструментів і технологій.

Крім того, дослідження та експерименти з налаштуваннями бази даних надали можливість краще зрозуміти принципи управління даними, оптимізації запитів і забезпечення цілісності даних. Використання MySQL у поєднанні з Hibernate відкриває нові горизонти для розробників, забезпечуючи ефективне управління інформацією і простоту в інтеграції з іншими системами.

Таким чином, результати лабораторної роботи підкреслюють важливість інтеграції теоретичних знань з практичними навичками, які необхідні для успішної кар'єри у галузі програмування та веб-розробки. Опанування таких технологій, як Java, Hibernate і MySQL, сприяє не лише підвищенню кваліфікації спеціалістів, але й їхній здатності адаптуватися до швидко змінюваного технологічного середовища.

ДОДАТОК А

Код файлу pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21</groupId>
  <artifactId>WebAppsSYOHBN</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>WebAppsSYOHBN</name>
  <packaging>war</packaging>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.source>17</maven.compiler.source>
    <junit.version>5.10.2</junit.version>
  </properties>

  <dependencies>
    <!-- Servlet API -->
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>

    <!-- JSTL -->
    <dependency>
      <groupId>org.glassfish.web</groupId>
      <artifactId>jakarta.servlet.jsp.jstl</artifactId>
      <version>3.0.1</version>
    </dependency>

    <!-- Expression Language API -->
    <dependency>
      <groupId>jakarta.el</groupId>
      <artifactId>jakarta.el-api</artifactId>
      <version>5.0.0</version>
    </dependency>

    <!-- MySQL Database -->
    <dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <version>9.0.0</version>
    </dependency>
```

```

<!-- Hibernate ORM -->
<dependency>
  <groupId>org.hibernate.orm</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>6.4.4.Final</version>
</dependency>

<!-- Hibernate Validator -->
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>8.0.1.Final</version>
</dependency>

<!-- JAXB Runtime -->
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
  <version>4.0.2</version>
</dependency>

<!-- Java Persistence API -->
<dependency>
  <groupId>jakarta.persistence</groupId>
  <artifactId>jakarta.persistence-api</artifactId>
  <version>3.1.0</version>
</dependency>

<!-- Jakarta Transaction API -->
<dependency>
  <groupId>jakarta.transaction</groupId>
  <artifactId>jakarta.transaction-api</artifactId>
  <version>2.0.1</version>
</dependency>

<!-- Spring Framework (use Jakarta EE-compatible versions) -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.3.10</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>5.3.10</version>
</dependency>

<!-- Logging -->
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.20.0</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.20.0</version>
</dependency>

<!-- Testing -->
<dependency>
  <groupId>org.junit.jupiter</groupId>

```

```

        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>

    <!-- Lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.20</version>
        <scope>provided</scope>
    </dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.4.0</version>
        </plugin>
    </plugins>
</build>
</project>

```

ДОДАТОК Б

Код тесту TestStart:

```
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.Messages;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.OfficeWorkerStatus;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.hibernate.HibernateUtil;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.daohbn.DAOWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities.OfficeWorker;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import java.time.LocalDate;

import static org.junit.jupiter.api.Assertions.*;

public class TestStart {
    private static SessionFactory sf;
    private static Session session;
    private Transaction transaction;
    static DAOWorker daoWorker = new DAOWorker();
    @BeforeAll
    static void beforeAll () {
        sf = HibernateUtil.getSessionFactory();
        session = sf.openSession();
        System.out.println("\n\nHIBERNATE підключився!");
    }
    @Test
    void testCreate() {
        System.out.println("Має бути створена БД!");
    }

    @AfterAll
    static void afterAll() {
        System.out.println("\n\nHIBERNATE завершив роботу");
        HibernateUtil.shutdown();
    }
}
```

ДОДАТОК В

Код класу OfficeWorker:

```
package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities;

import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.OfficeWorkerStatusConverter;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.OfficeWorkerStatus;
import jakarta.persistence.*;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.validator.constraints.UUID;
import lombok.experimental.Accessors;
import org.hibernate.annotations.Check;
import org.hibernate.annotations.ColumnDefault;
import org.hibernate.annotations.GenericGenerator;
import org.springframework.format.annotation.DateTimeFormat;

import java.time.LocalDate;
import java.util.Date;

@Entity
@Table(name = "officeworkers")
@Check(constraints = "end_work >= start_work + INTERVAL 4 MONTH")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class OfficeWorker {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name="surname", length = 100, nullable = false)
    @Check(constraints = "REGEXP_LIKE(surname, '^[A-Z][a-z]+$','c')= 1")
    private String surname;

    @Column(name="name", length = 100, nullable = false)
    @Check(constraints = "REGEXP_LIKE(name, '^[A-Z][a-z]+$','c')= 1")
    private String name;

    @Column(name="pname", length = 100, nullable = false)
    @Check(constraints = "REGEXP_LIKE(pname, '^[A-Z][a-z]+$','c')= 1")
    private String pname;

    @Column(name = "start_work", nullable = false)
    @DateTimeFormat(pattern = "dd.MM.yyyy")
    private LocalDate startWork;

    @Column(name = "end_work")
    @DateTimeFormat(pattern = "dd.MM.yyyy")
    private LocalDate endWork;

    @Column(name="worker_cod", nullable=false)
    @Size(min = 4, max = 4, message = "Код працівника має складатися з 4 цифр.")
```

```
@Pattern(regexp = "[0-9]{4}$", message = "Код працівника повинен складатися з 4 цифр.")
private String workerCod;

@Enumerated(EnumType.STRING)
@Column(name = "officeworker_status", nullable = false, length = 12)
@Convert(converter = OfficeWorkerStatusConverter.class)
private OfficeWorkerStatus officeWorkerStatus;

// Constructor for create instance for testing
public OfficeWorker(String name) {
    this.id = null;
    this.surname = surname;
    this.name = name;
    this.pname = pname;
    this.startWork = startWork;
    this.endWork = endWork;
    this.workerCod = workerCod;
    this.officeWorkerStatus = OfficeWorkerStatus.middle;
}
}
```


ДОДАТОК Г

Код класу OfficeWorkerStatus:

```
package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums;

public enum OfficeWorkerStatus {
    intern("Intern"),
    junior("Junior"),
    middle("Middle"),
    senior("Senior"),
    lead("Lead");

    private final String displayName;

    OfficeWorkerStatus(String displayName) {
        this.displayName = displayName;
    }

    public static OfficeWorkerStatus getOfficeWorkerStatusById(Integer index) {
        if (index >= OfficeWorkerStatus.values().length) {
            return OfficeWorkerStatus.values()[0];
        } else {
            return OfficeWorkerStatus.values()[index];
        }
    }

    public String getDisplayName() {
        return displayName;
    }

    public static String[] getOffWorkerStatus() {
        OfficeWorkerStatus[] pl = values();
        String[] plNames = new String[pl.length];
        for (int i = 0; i < pl.length; i++) {
            plNames[i] = pl[i].getDisplayName();
        }
        return plNames;
    }

    public static int getEnumIndex(String value) {
        int index = -1;
        OfficeWorkerStatus[] pl = values();
        for (OfficeWorkerStatus lang: pl) {
            index++;
            if (lang.getDisplayName().equals(value)) {
                break;
            }
        }
        return index;
    }

    public static OfficeWorkerStatus getOfficeWorkerStatusByName(String namePL) {
        int index = -1;
        OfficeWorkerStatus[] plValues = values();
        boolean flFound = false;
```

```

while (!flFound && index<plValues.length-1) {
    index++;
    if (plValues[index].getDisplayName().equals(namePL)) {
        flFound = true;
    }
}
OfficeWorkerStatus pl = null;
if (!flFound) {
    pl = OfficeWorkerStatus.values()[0];
} else {
    pl = OfficeWorkerStatus.values()[index];
}
return pl;
}

public static String[] getOfficeWorkerStatus() {
    OfficeWorkerStatus[] pl = values();
    String[] plNames = new String[pl.length];
    for (int i = 0; i < pl.length; i++) {
        plNames[i] = pl[i].getDisplayName();
    }
    return plNames;
}
}

```

ДОДАТОК Д

Код класу OfficeWorkerStatusConverter:

```
package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums;

import jakarta.persistence.AttributeConverter;

public class OfficeWorkerStatusConverter implements AttributeConverter<OfficeWorkerStatus, Integer> {
    @Override
    public Integer convertToDatabaseColumn(OfficeWorkerStatus pl) {
        return pl.ordinal();
    }

    @Override
    public OfficeWorkerStatus convertToEntityAttribute(Integer codPL) {
        return OfficeWorkerStatus.getOfficeWorkerStatusById(codPL);
    }
}
```

ДОДАТОК Е

Код класу DAOOfficeWorker:

```
package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.daohbn;

import jakarta.persistence.TypedQuery;
import jakarta.persistence.criteria.CriteriaBuilder;
import jakarta.persistence.criteria.CriteriaQuery;
import jakarta.persistence.criteria.Root;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;
import java.util.List;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities.OfficeWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.hibernate.HibernateUtil;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.Messages;

public class DAOOfficeWorker {

    public static List<OfficeWorker> getAllList() {
        List<OfficeWorker> results = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            CriteriaBuilder cb = session.getCriteriaBuilder();
            CriteriaQuery cq = cb.createQuery(OfficeWorker.class);
            Root rootEntry = cq.from(OfficeWorker.class);
            CriteriaQuery all = cq.select(rootEntry);
            TypedQuery allQuery = session.createQuery(all);
            results = allQuery.getResultList();
        }
        return results;
    }

    // Метод для отримання OfficeWorker за ID
    public static OfficeWorker getOfficeWorkerById(Long idToFind) {
        OfficeWorker workerInDB = null;
        List<OfficeWorker> results;
        Session session = HibernateUtil.getSessionFactory().openSession();
        CriteriaBuilder cb = session.getCriteriaBuilder();
        CriteriaQuery<OfficeWorker> cq = cb.createQuery(OfficeWorker.class);
        Root<OfficeWorker> rootEntry = cq.from(OfficeWorker.class);
        cq.select(rootEntry).where(cb.equal(rootEntry.get("id"), idToFind));
        Query<OfficeWorker> query = session.createQuery(cq);
        results = query.getResultList();
        if (!results.isEmpty()) {
            workerInDB = results.get(0);
        }
        return workerInDB;
    }

    // Метод для отримання OfficeWorker за унікальними полями (наприклад, name)
    public static OfficeWorker getOfficeWorkerByKeyset(OfficeWorker workerKey) {
        OfficeWorker findWorker = null;
        List<OfficeWorker> results;
        Session session = HibernateUtil.getSessionFactory().openSession();
```

```

CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<OfficeWorker> cq = cb.createQuery(OfficeWorker.class);
Root<OfficeWorker> rootEntry = cq.from(OfficeWorker.class);
cq.select(rootEntry).where(cb.equal(rootEntry.get("name"), workerKey.getName()));
Query<OfficeWorker> query = session.createQuery(cq);
results = query.getResultList();
if (!results.isEmpty()) {
    findWorker = results.get(0);
}
return findWorker;
}

// Метод для додавання нового OfficeWorker
public static String insert(OfficeWorker workerForInsert) {
    String msgIns = Messages.OK_INS_MSG.getText();
    Session session = null;
    OfficeWorker workerInDB = DAOOfficeWorker.getOfficeWorkerByKeyset(workerForInsert);
    if (workerInDB == null) {
        Transaction transaction = null;
        try {
            session = HibernateUtil.getSessionFactory().openSession();
            transaction = session.beginTransaction();
            session.persist(workerForInsert);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            msgIns = Messages.ERR_INS_MSG.getText();
        } finally {
            if (session != null) {
                session.close();
            }
        }
    } else {
        msgIns = Messages.ERR_DOUBLE_MSG.getText();
    }
    return msgIns;
}

// Метод для оновлення існуючого OfficeWorker
public static String update(OfficeWorker workerForUpdate, Long idWorkerToUpdate) {
    String msgUp = Messages.OK_UPD_MSG.getText();
    OfficeWorker workerForEdit = getOfficeWorkerById(idWorkerToUpdate);
    if (workerForEdit == null) {
        msgUp = Messages.ERR_NOT_IN_MSG.getText();
    } else {
        OfficeWorker workerInDB = DAOOfficeWorker.getOfficeWorkerByKeyset(workerForUpdate);
        if (workerInDB == null || (workerInDB != null && workerInDB.getId().equals(idWorkerToUpdate))) {
            Transaction transaction = null;
            try (Session session = HibernateUtil.getSessionFactory().openSession()) {
                workerForUpdate.setId(workerForEdit.getId());
                transaction = session.beginTransaction();
                session.merge(workerForUpdate);
                transaction.commit();
            } catch (Exception e) {
                if (transaction != null) {
                    transaction.rollback();
                }
                msgUp = Messages.ERR_UPD_MSG.getText();
            }
        } else {
            msgUp = Messages.ERR_DOUBLE_MSG.getText();
        }
    }
}

```

```

    }
}
return msgUp;
}

// Метод для видалення OfficeWorker за ID
public static String delete(Long idWorkerToDelete) {
    String msgDel = Messages.OK_DEL_MSG.getText();
    OfficeWorker workerForDel = getOfficeWorkerById(idWorkerToDelete);
    if (workerForDel == null) {
        msgDel = Messages.ERR_NOT_IN_MSG.getText();
    } else {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.remove(workerForDel);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            msgDel = Messages.ERR_DEL_MSG.getText();
        }
    }
    return msgDel;
}

public static boolean checkDuplicate(OfficeWorker workerKey) {
    List<OfficeWorker> results;
    Session session = HibernateUtil.getSessionFactory().openSession();
    CriteriaBuilder cb = session.getCriteriaBuilder();
    CriteriaQuery<OfficeWorker> cq = cb.createQuery(OfficeWorker.class);
    Root<OfficeWorker> rootEntry = cq.from(OfficeWorker.class);
    cq.select(rootEntry).where(
        cb.equal(rootEntry.get("surname"), workerKey.getSurname()),
        cb.equal(rootEntry.get("name"), workerKey.getName()),
        cb.equal(rootEntry.get("pname"), workerKey.getPname())
    );

    Query<OfficeWorker> query = session.createQuery(cq);
    results = query.getResultList();

    session.close();
    return !results.isEmpty();
}
}

```

ДОДАТОК К

Код методу testCRUD:

```
void testCRUD() {  
    // 1. Create an OfficeWorker instance and add it to the database  
    OfficeWorker workerToCreate = new OfficeWorker();  
    workerToCreate.setSurname("Smith");  
    workerToCreate.setName("John");  
    workerToCreate.setPname("Doe");  
    workerToCreate.setStartWork(LocalDate.of(2023, 1, 1));  
    workerToCreate.setEndWork(LocalDate.of(2023, 12, 31));  
    workerToCreate.setWorkerCod("1034");  
    workerToCreate.setOfficeWorkerStatus(OfficeWorkerStatus.middle);  
  
    // Insert the new worker  
    String insertMessage = DAOOfficeWorker.insert(workerToCreate);  
    assertEquals(Messages.OK_INS_MSG.getText(), insertMessage);  
  
    // 2. Retrieve the worker by unique key fields  
    OfficeWorker retrievedWorker = DAOOfficeWorker.getOfficeWorkerByKeyset(workerToCreate);  
    assertNotNull(retrievedWorker);  
    assertEquals(workerToCreate.getName(), retrievedWorker.getName());  
    assertEquals(workerToCreate.getSurname(), retrievedWorker.getSurname());  
  
    // 3. Update the worker  
    retrievedWorker.setName("Jane");  
    String updateMessage = DAOOfficeWorker.update(retrievedWorker, retrievedWorker.getId());  
    assertEquals(Messages.OK_UPD_MSG.getText(), updateMessage);  
  
    // Verify the update  
    OfficeWorker updatedWorker = DAOOfficeWorker.getOfficeWorkerById(retrievedWorker.getId());  
    assertEquals("Jane", updatedWorker.getName());  
  
    // 4. Delete the worker  
    String deleteMessage = DAOOfficeWorker.delete(updatedWorker.getId());  
    assertEquals(Messages.OK_DEL_MSG.getText(), deleteMessage);  
  
    // Verify the deletion  
    OfficeWorker deletedWorker = DAOOfficeWorker.getOfficeWorkerById(updatedWorker.getId());  
    assertNull(deletedWorker);  
}
```

ДОДАТОК Л

Код сторінки officeworkers.jsp:

```

<% @ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<% @ page
import="cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.OfficeWorkerStatus" %>
<% @ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<% @ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Web Java 24 - Office Workers (HBN)</title>
<style>
body {
font-family: Arial, sans-serif;
background-color: #f4f4f9;
color: #333;
margin: 0;
padding: 0;
}
h1 {
text-align: center;
color: #4a90e2;
margin: 20px 0;
}
a {
display: inline-block;
margin: 10px 0;
padding: 10px 20px;
background-color: #4a90e2;
color: white;
text-decoration: none;
border-radius: 5px;
font-size: 16px;
}
a:hover {
background-color: #357ab7;
}
.container {
width: 90%;
margin: 0 auto;
padding: 20px;
background-color: white;
border-radius: 8px;
box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
}
table {
width: 100%;
border-collapse: collapse;
margin-top: 20px;
}
th, td {
padding: 10px;
text-align: left;
border: 1px solid #ddd;
}
th {

```



```

    background-color: #4a90e2;
    color: white;
}
tr:nth-child(even) {
    background-color: #f9f9f9;
}
tr:hover {
    background-color: #f1f1f1;
}
input[type="text"], select {
    padding: 8px;
    width: 100%;
    margin-top: 5px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}
button, input[type="submit"] {
    background-color: #4a90e2;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
button:hover, input[type="submit"]:hover {
    background-color: #357ab7;
}
form {
    display: inline;
}
.filters {
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    margin-bottom: 20px;
}
.filters div {
    flex: 1;
    margin: 10px;
}
@media (max-width: 768px) {
    .filters div {
        flex-basis: 100%;
    }
    table {
        font-size: 14px;
    }
}
</style>
</head>
<body>

<div class="container">
  <a href="start.jsp">To START</a>
  <h1>Office Workers</h1>
  <!-- Фильтрация, сортировка и поиск -->
  <div class="filters">
    <div>
      <b><label for="pl_filter">Filter by status:</label></b>
      <select id="pl_filter" name="pl_filter" onchange="filterByPL()">
        <option value="all">All</option>

```

```

        <option value="intern">Intern</option>
        <option value="junior">Junior</option>
        <option value="middle">Middle</option>
        <option value="senior">Senior</option>
        <option value="lead">Lead</option>
        <c:forEach items="${ OfficeWorkers.getOfficeWorkers()}" var="pname">
            <option value="<c:out value='${pname}'/"><c:out value='${pname}'/"></option>
        </c:forEach>
    </select>
</div>

<div>
    <b>Sort by:</b>
    <br/>
    <button onclick="sortTableByIndex(0)">Surname</button>
    <button onclick="sortTableByDate(3)">Start Date</button>
</div>

<div>
    <b><label for="search-text">Search by surname:</label></b>
    <input type="text" id="search-text" placeholder="Search..." onkeyup="tableSearch()">
</div>

<div>
    <form action="" method="get">
        <button type="submit" accesskey="x">Reset</button>
    </form>
</div>
</div>

<form action="officeworker.jsp" method="GET">
    <input type="submit" value="INSERT OFFICE WORKER"/>
</form>

<!-- Таблица сотрудников -->
<table id="officeworkers-table">
    <tr>
        <th>Surname</th>
        <th>Name</th>
        <th>Patronymic name</th>
        <th>Date of start work</th>
        <th>Date of end work</th>
        <th>Status</th>
        <th>WorkerCode</th>
        <th colspan="2">Actions</th>
    </tr>
    <c:forEach var="e" items="${ officeworkers}">
        <tr>
            <td><c:out value='${e.surname}'/"></td>
            <td><c:out value='${e.name}'/"></td>
            <td><c:out value='${e.pname}'/"></td>
            <td>
                <fmt:setLocale value="uk_UA"/>
                <fmt:parseDate value='${e.startWork}' pattern="yyyy-MM-dd" var="e_startWork" type="date"/>
                <fmt:formatDate value='${e_startWork}' pattern="dd-MM-yyyy" />
            </td>
            <td>
                <fmt:setLocale value="uk_UA"/>
                <fmt:parseDate value='${e.endWork}' pattern="yyyy-MM-dd" var="e_endWork" type="date"/>
                <fmt:formatDate value='${e_endWork}' pattern="dd-MM-yyyy" />
            </td>
            <td><c:out value='${e.officeWorkerStatus.getDisplayName()}'/"></td>
            <td><c:out value='${e.workerCod}'/"></td>
        </tr>
    </c:forEach>
</table>

```

```

<td>
  <form action="officeworker" method="GET">
    <input type="hidden" name="id_worker" value="{e.id}"/>
    <input type="submit" value="UPDATE"/>
  </form>
</td>
<td>
  <form action="officeworkers/delete" method="GET">
    <input type="hidden" name="id_worker" value="{e.id}"/>
    <input type="submit" value="DELETE" onclick="return confirmation()"/>
  </form>
</td>
</tr>
</:forEach>
</table>
</div>

<!-- Скрипты для фильтрации и сортировки -->
<script type="text/javascript">
  function confirmation() {
    return confirm('Are you sure you want to delete this data?');
  }

  function tableSearch() {
    var phrase = document.getElementById('search-text').value.toLowerCase();
    var table = document.getElementById('officeworkers-table');
    var rows = table.getElementsByTagName('tr');
    for (var i = 1; i < rows.length; i++) {
      var surname = rows[i].getElementsByTagName('td')[0].innerHTML.toLowerCase();
      rows[i].style.display = surname.includes(phrase) ? '' : 'none';
    }
  }

  function sortTableByIndex(col_index) {
    var table = document.getElementById("officeworkers-table");
    var rows, switching, i, x, y, shouldSwitch;
    switching = true;
    while (switching) {
      switching = false;
      rows = table.rows;
      for (i = 1; i < (rows.length - 1); i++) {
        shouldSwitch = false;
        x = rows[i].getElementsByTagName("TD")[col_index].innerHTML.toLowerCase();
        y = rows[i + 1].getElementsByTagName("TD")[col_index].innerHTML.toLowerCase();
        if (x > y) {
          shouldSwitch = true;
          break;
        }
      }
      if (shouldSwitch) {
        rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
        switching = true;
      }
    }
  }

  function sortTableByDate(col_index) {
    var table = document.getElementById("officeworkers-table");
    var rows, switching, i, x, y, shouldSwitch;
    switching = true;
    while (switching) {
      switching = false;
      rows = table.rows;

```

```

for (i = 1; i < (rows.length - 1); i++) {
    shouldSwitch = false;

    // Отримуємо дати з комірок
    x = rows[i].getElementsByTagName("TD")[col_index].innerHTML;
    y = rows[i + 1].getElementsByTagName("TD")[col_index].innerHTML;

    // Розбиваємо дати
    var dateX = x.split('-');
    var dateY = y.split('-');

    // Створюємо об'єкти Date
    var d1 = new Date(dateX[2], dateX[1] - 1, dateX[0]); // YYYY, MM (0-11), DD
    var d2 = new Date(dateY[2], dateY[1] - 1, dateY[0]); // YYYY, MM (0-11), DD

    // Порівнюємо дати
    if (d1 > d2) {
        shouldSwitch = true;
        break;
    }
}
if (shouldSwitch) {
    rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
    switching = true;
}
}
}

function filterByPL() {
    var filterValue = document.getElementById('pl_filter').value.toLowerCase();
    var table = document.getElementById('officeworkers-table');
    var rows = table.getElementsByTagName('tr');
    for (var i = 1; i < rows.length; i++) {
        var status = rows[i].getElementsByTagName('td')[5].innerText.toLowerCase();
        rows[i].style.display = (filterValue === 'all' || status === filterValue) ? '' : 'none';
    }
}
</script>

</body>
</html>

```

ДОДАТОК М

Код сервлету ShowOfficeWorkersServlet

```

package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.servlets;

import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.daohbn.DAOfficeWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities.OfficeWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities.OfficeWorkerList;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.List;

@WebServlet("/officeworkers")
public class ShowOfficeWorkersServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        System.out.println("ShowOfficeWorkersServlet#get"); // Лог для отладки

        List<OfficeWorker> listOfficeWorkers = DAOfficeWorker.getAllList(); // Получаем список работников

        // Проверка, если список пуст
        if (listOfficeWorkers == null || listOfficeWorkers.isEmpty()) {
            System.out.println("List is empty");
        } else {
            System.out.println("List size: " + listOfficeWorkers.size()); // Лог для отладки
        }

        request.setAttribute("officeworkers", listOfficeWorkers); // Передача данных на JSP

        String path = "/officeworkers.jsp"; // Путь к JSP
        ServletContext servletContext = getServletContext();
        RequestDispatcher requestDispatcher = servletContext.getRequestDispatcher(path);
        requestDispatcher.forward(request, response); // Перенаправление на JSP
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        System.out.println("cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.servlets.ShowOfficeWorkersServlet#doPost");
    }
}

```

ДОДАТОК Н

Код сторінки officeworker.jsp:

```

<% @ page import="java.time.LocalDate" %>
<% @ page
import="cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.OfficeWorkerStatus" %>
<% @ page contentType="text/html; charset=UTF-8" language="java" pageEncoding="UTF-8"%>
<% @ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<% @ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>${titleOfficeWorker} - Web Java 24 (HBN)</title>
  <style type="text/css">
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f9;
      color: #333;
      margin: 0;
      padding: 0;
    }
    .content {
      max-width: 800px;
      margin: 40px auto;
      padding: 20px;
      background-color: #fff;
      border-radius: 8px;
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
    }
    h1 {
      color: #4a90e2;
      text-align: center;
    }
    label {
      font-weight: bold;
      display: block;
      margin: 10px 0 5px;
    }
    input[type="text"],
    input[type="date"],
    select {
      width: 100%;
      padding: 8px;
      margin: 5px 0 15px;
      border: 1px solid #ccc;
      border-radius: 4px;
      box-sizing: border-box;
    }
    .error-message {
      color: red;
      font-weight: bold;
      text-align: center;
      margin-bottom: 20px;
    }
    button {
      background-color: #4a90e2;
      color: white;

```

```

padding: 10px 20px;
border: none;
border-radius: 5px;
cursor: pointer;
}
button:hover {
background-color: #357ab7;
}
a {
display: inline-block;
padding: 10px 20px;
margin-left: 10px;
background-color: #ccc;
color: black;
text-decoration: none;
border-radius: 5px;
}
a:hover {
background-color: #aaa;
}
@media (max-width: 600px) {
.content {
padding: 15px;
box-shadow: none;
}
h1 {
font-size: 1.5em;
}
}
</style>
</head>
<body>
<div class="content">
<h1><c:out value="{titleOfficeWorker}"/> HBN Proc Java 24</h1>

<c:if test="{fn:length(errorString) > 0}">
<p class="error-message"><c:out value="{errorString}"/></p>
</c:if>

<form action="officeworker" method="post">
<input type="hidden" name="id_worker" value="{officeworker.id}" required>

<div>
<label for="surname">Surname:</label>
<input type="text" name="surname" id="surname" value="{officeworker.surname}"
pattern="^[A-Z][a-z]+$" required title="Surname must start with a capital letter and only contain
letters."/>
</div>

<div>
<label for="name">Name:</label>
<input type="text" name="name" id="name" value="{officeworker.name}"
pattern="^[A-Z][a-z]+$" required title="Name must start with a capital letter and only contain
letters."/>
</div>

<div>
<label for="pname">Patronymic name:</label>
<input type="text" name="pname" id="pname" value="{officeworker.pname}"
pattern="^[A-Z][a-z]+$" required title="Patronymic name must start with a capital letter and only
contain letters."/>
</div>

```

```

<div>
  <label for="startWork">Date of start work:</label>
  <input type="date" name="startWork" id="startWork" value="{officeworker.startWork}"
    min="{LocalDate.now().minusYears(15)}" max="{LocalDate.now().plusMonths(1)}" required/>
</div>

<div>
  <label for="endWork">Date of end work (Optional):</label>
  <input type="date" name="endWork" id="endWork" value="{officeworker.endWork}"
    min="{LocalDate.now().minusYears(15).minusMonths(4)}"/>
</div>

<div>
  <label for="workerCod">Worker Code (4 digits, can start with zero):</label>
  <input type="text" name="workerCod" id="workerCod" value="{officeworker.workerCod}"
    pattern="^\d{4}$" title="Please enter exactly 4 digits" required maxlength="4" minlength="4"/>
</div>

<div>
  <label for="worker_status">Office Worker Status:</label>
  <select id="worker_status" name="officeWorkerStatus" required>
    <:forEach items="{OfficeWorkerStatus.getOfficeWorkerStatus()}" var="pname">
      <option value=":out value='{pname}'/>
        ${officeworker.officeWorkerStatus.getDisplayName() == pname ? 'selected' : ''}>
      <:out value='{pname}'/>
    </option>
    <:forEach>
  </select>
</div>

<div>
  <button type="submit">SAVE</button>
  <a href=":out value='{pageContext.request.contextPath}/officeworkers/'>">Cancel</a>
</div>
</form>
</div>
</body>
</html>

```


ДОДАТОК II

Код сервлету AddEditOfficeWorkerServlet

```
package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.servlets;

import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.daohbn.DAOOfficeWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.entities.OfficeWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.Messages;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.OfficeWorkerStatus;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

@WebServlet("/officeworker")
public class AddEditOfficeWorkerServlet extends HttpServlet {

    private static final String TEXT_TITLE_ADD = "Add Office Worker";
    private static final String TEXT_TITLE_EDIT = "Edit Office Worker";

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOException {
        String idWorker = request.getParameter("id_worker");
        Long id = (idWorker != null && !idWorker.isEmpty()) ? Long.parseLong(idWorker) : null;

        OfficeWorker officeWorker;
        String titleOfficeWorker;

        if (id == null) {
            officeWorker = new OfficeWorker("New Office Worker");
            titleOfficeWorker = TEXT_TITLE_ADD;
        } else {
            officeWorker = DAOOfficeWorker.getOfficeWorkerById(id);
            titleOfficeWorker = TEXT_TITLE_EDIT;
        }

        request.setAttribute("officeworker", officeWorker);
        request.setAttribute("titleOfficeWorker", titleOfficeWorker);
        RequestDispatcher dispatcher = request.getRequestDispatcher("/officeworker.jsp");
        dispatcher.forward(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOException {
        String idStr = request.getParameter("id_worker");
        String surname = request.getParameter("surname");
        String name = request.getParameter("name");
        String pname = request.getParameter("pname");
        String startWorkStr = request.getParameter("startWork");
        String endWorkStr = request.getParameter("endWork");
```

```

String workerCod = request.getParameter("workerCod");
String statusStr = request.getParameter("officeWorkerStatus");
Long id = (idStr != null && !idStr.isEmpty()) ? Long.parseLong(idStr) : null;

LocalDate startWorkDate = LocalDate.parse(startWorkStr, DateTimeFormatter.ofPattern("yyyy-MM-dd"));
LocalDate endWorkDate = (endWorkStr != null && !endWorkStr.isEmpty()) ? LocalDate.parse(endWorkStr,
DateTimeFormatter.ofPattern("yyyy-MM-dd")) : null;

OfficeWorkerStatus status = OfficeWorkerStatus.getOfficeWorkerStatusByName(statusStr);

OfficeWorker newOfficeWorker = new OfficeWorker(id, surname, name, pname, startWorkDate,
endWorkDate, workerCod, status);
String errorString = null;
String titleOfficeWorker = ""; // Initialize here

if (id == null) { // Новая запись
    String result = DAOOfficeWorker.insert(newOfficeWorker);
    if (!result.equals(Messages.OK_INS_MSG.getText())) {
        errorString = result;
        titleOfficeWorker = TEXT_TITLE_ADD;
    } else {
        titleOfficeWorker = TEXT_TITLE_ADD; // Also set for success
    }
} else { // Редактирование
    String result = DAOOfficeWorker.update(newOfficeWorker, id);
    if (!result.equals(Messages.OK_UPD_MSG.getText())) {
        errorString = result;
        titleOfficeWorker = TEXT_TITLE_EDIT;
    } else {
        titleOfficeWorker = TEXT_TITLE_EDIT; // Also set for success
    }
}

// Если ошибка, возвращаем на форму
if (errorString != null) {
    request.setAttribute("errorString", errorString);
    request.setAttribute("officeworker", newOfficeWorker);
    request.setAttribute("titleOfficeWorker", titleOfficeWorker);
    request.getRequestDispatcher("/officeworker.jsp").forward(request, response);
} else {
    response.sendRedirect(request.getContextPath() + "/officeworkers");
}
}
}

```

ДОДАТОК Р

Код сервлету DeleteOfficeWorkerServlet:

```
package cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.servlets;

import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.daohbn.DAOOfficeWorker;
import cs.khpi.edu.stu.webappstulab21.webappssyohbnlab21.webappssyohbn.enums.Messages;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

@WebServlet("/officeworkers/delete")
public class DeleteOfficeWorkerServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        System.out.println("DeleteOfficeWorkerServlet#doGet");
        Long idDel = Long.parseLong(request.getParameter("id_worker"));
        String deleteRes = DAOOfficeWorker.delete(idDel);
        if (!deleteRes.equals(Messages.OK_DEL_MSG.getText())) {
            Messages errorMSG = Messages.getMessageByText(deleteRes);
            String path = request.getContextPath() + "/officeworkers?errorString="+errorMSG.name();
            // System.out.println(path);
            response.sendRedirect(path);
        } else {
            //back to listOfficeWorkers
            String path = request.getContextPath() + "/officeworkers";
            response.sendRedirect(path);
        }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        System.out.println("DeleteOfficeWorkerServlet#doPost");
        doGet(request, response);
    }

}
```

ДОДАТОК С

Код файлу 404.html:

```
<html>
<style type="text/css">
  body, html {
    height: 100%;
  }

  .bg {
    /* The image used */
    background-image: url("jakartaee9.jpg");

    /* Full height */
    height: 100%;

    /* Center and scale the image nicely */
    position: relative;
    opacity: 1.0;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
  }

  .caption {
    position: absolute;
    left: 0;
    top: 25%;
    width: 50%;
    text-align: center;
    color: #FFFFFF;
  }

</style>
<head>
  <title>Web Java 24 - 404</title>
</head>
<body>
<div class="bg">
  <div class="caption">
    <h1>)$ 4 0 4 $</h1>
    <br/>
    <h1>SOMETHING WENT WRONG...>
    </h1>
    <br/>
    <h3><a href="officeworkers">RETURN to Office Workers</a></h3>
  </div>
</div>
</body>
</html>
```