ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ ШАБЛОНУ PROXY

Мета

1 Закріпити інформацію про шаблони, отриману в результаті вивчення джерел інформації.

2 Набути навичок практичного використання шаблонів для вирішення прикладних задач.

Завдання роботи

Виконати рефакторинг застосунку, що працює із базою даних, додавши до функціоналу можливість завантаження копії бази даних, попередньо збереженої у файлі (формат файлу-копії обрати самостійно, рекомендується xml або json), у випадку відсутності доступу до бази даних. У випадку завантаження даних з файлу-копії база застосунок має тільки відображати дані (без можливості додавання, редагування, вилучення). Файл-копію зберігати на стороні серверу. Функціонал завантаження даних з файлу-копії здійснити на основі паттерну Proxy.

Хід роботи

1 Ознайомлення з шаблоном проектування Proxy

1 Назва шаблону: Proxy (Замісник).

2 Основні компоненти шаблону Proxy:

– клієнт: об’єкт, який звертається до методів сервісу через проксі;

– реальний об’єкт (Real Service): об’єкт, до якого здійснюється доступ через проксі та який виконує основну функціональність;

– проксі (Proxy): посередник, що контролює доступ до реального об’єкта. Він може додавати додаткову логіку, наприклад, перевірку доступу, кешування або ведення логів.

3 Ситуації, коли застосовується Proxy:

– контроль доступу: коли потрібно перевіряти права перед викликом методів реального об’єкта;

– оптимізація: для зменшення навантаження, наприклад, через використання кешу;

– лінива ініціалізація: коли об’єкт створюється лише під час першого звернення до нього;

– розширення функціональності: коли необхідно додати нову логіку до існуючого класу, не змінюючи його код.

2 Розроблення класів для реалізації завдання

2.1 Реалізація Proxy

1 Інтерфейс OfficeWorkerService (Додаток А):

– визначає основні методи для управління товарами;

– реалізується всіма конкретними класами, включаючи проксі.

2 Клас FileOfficeWorkerService (Додаток Б):

– реалізація OfficeWorkerService для роботи з файлами;

– призначений для збереження та читання товарів у режимі, коли база даних недоступна;

– описані методи читання даних, а також збереження даних у файл.

3 Клас OfficeWorkerServiceImpl (Додаток В):

– основна реалізація OfficeWorkerService, яка працює з базою даних через репозиторій;

– використовує JPA для взаємодії з таблицею officeworkers;

– забезпечує всі CRUD-операції для товарів.

4 Клас OfficeWorkerServiceProxy (Додаток Г):

– реалізує OfficeWorkerService;

– додає перевірку доступності бази даних перед делегуванням викликів методу;

– якщо база доступна, виклики делегуються до OfficeWorkerServiceImpl, у разі недоступності — до FileOfficeWorkerService;

– додає безпеку, використовуючи аннотації @PreAuthorize.

2.2 Реалізація веб-застосунку

1 Клас Controller (Додаток К):

– контролер, який відповідає за управління товарами та їх взаємодію з веб-інтерфейсом, який має логіку перемикання режимів.

2 officeworkers.html (Додаток Л) реалізована гнучкість відображення залежно від режиму роботи:

– режим онлайн: відображаються кнопки редагування, видалення та резервного копіювання;

– режим офлайн: кнопки редагування та видалення приховані, виводиться попередження.

3 UML діаграма класів для розробленої імплементації

UML діаграма класів для розробленої імплементації зображена на рис. 1.

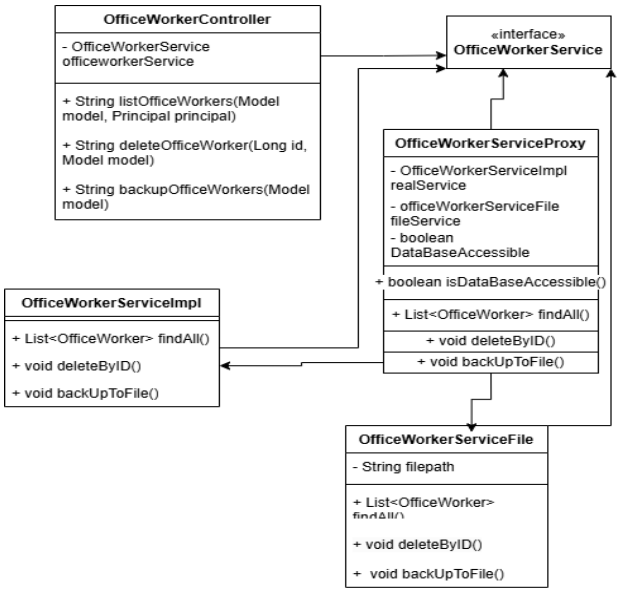

Рисунок 1 – UML діаграма класів для розробленої імплементації

4 Вигляд веб-сторінки

Інтерфейс стартової сторінки в режимі онлайн та оффлайн представлені відповідно на рис. 2 та 3.

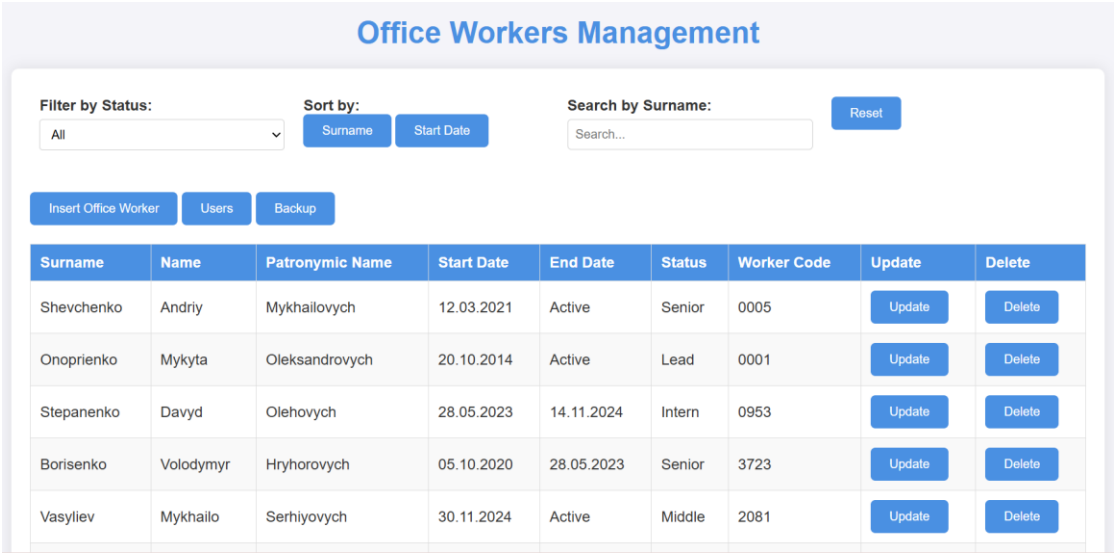

Рисунок 2 – Інтерфейс сторінки в онлайн режимі

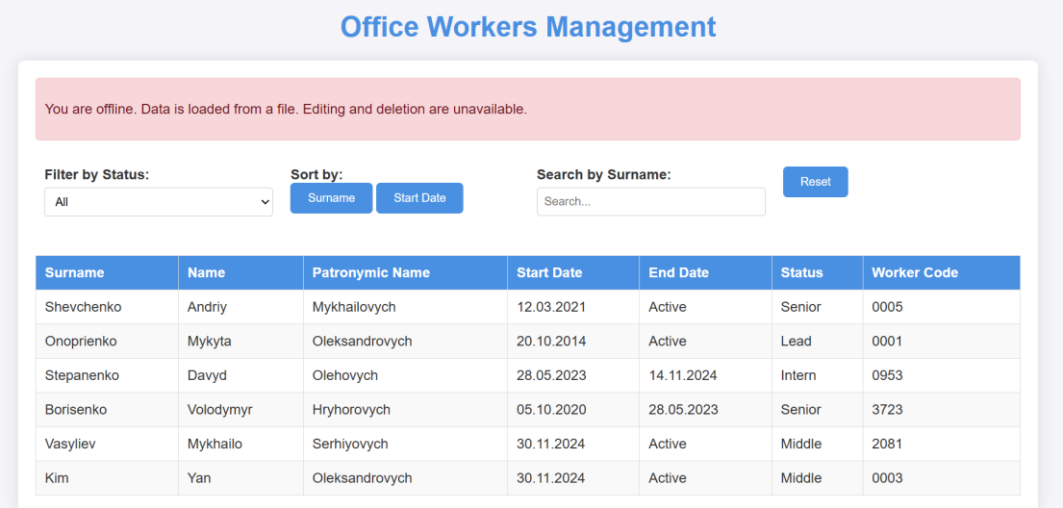

Рисунок 3 – Інтерфейс сторінки після відключення бд

Висновки

У процесі виконання цієї лабораторної роботи було досліджено шаблон проектування Proxy, який є важливим інструментом для контролю доступу до об'єктів та розширення їхньої функціональності без внесення змін у код реального об'єкта. Proxy дозволяє ефективно організовувати взаємодію між

клієнтом і реальним об'єктом, забезпечуючи додаткову логіку, таку як перевірка доступу, кешування чи ведення логів.

Робота з шаблоном Proxy дозволила краще зрозуміти, як застосовувати посередницький об'єкт для оптимізації роботи системи та забезпечення її безпеки. Завдяки цьому шаблон сприяє спрощенню процесу управління доступом до реальних об'єктів, а також дає змогу створювати системи з більш гнучкою архітектурою, де легко додавати нову функціональність без значних змін у коді.

Порівнюючи шаблон Proxy з іншими шаблонами проектування, можна відзначити його перевагу у ситуаціях, коли необхідно контролювати доступ до ресурсів, реалізувати ліниву ініціалізацію або зменшити навантаження на систему через використання кешу. Однак, неправильне використання Proxy може ускладнити структуру програми, особливо якщо додається надмірна кількість проміжних рівнів.

Вивчення шаблону Proxy дозволило зрозуміти важливість організації ефективної взаємодії між клієнтом і реальним об'єктом, що забезпечує як розширюваність, так і зручність підтримки системи. Отриманий досвід показує, що Proxy є потужним інструментом, який допомагає створювати модульні, захищені та продуктивні програмні рішення.

ДОДАТОК А

Код OfficeWorkerService

```
package stusyo222b.appz6.service;

import stusyo222b.appz6.entities.OfficeWorker;

import java.util.List;

public interface OfficeWorkerService {

    List<OfficeWorker> findAll();
    OfficeWorker findById(Long id);
    void save(OfficeWorker officeworker);
    void deleteById(Long id);
    void updateOfficeWorker(Long id, OfficeWorker officeworker);

    boolean isDatabaseAccessible();

    void backupToFile();
}
```

ДОДАТОК Б

Код класу FileOfficeWorkerService

```
package stusyo222b.appz6.service;


import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import org.springframework.stereotype.Service;
import stusyo222b.appz6.entities.OfficeWorker;


import java.io.File;
import java.util.ArrayList;
import java.util.List;

@Service
public class FileOfficeWorkerService implements OfficeWorkerService {
    private final File file = new
File("C:/Users/lizas/IdeaProjects/APPZ6/src/main/resources/officeworkers_backup.json");

    @Override
    public List<OfficeWorker> findAll() {
        try {
            ObjectMapper objectMapper = new ObjectMapper();
            // Регистрируем модуль для обработки Java 8 типов данных, включая LocalDate
            objectMapper.registerModule(new JavaTimeModule());
            return objectMapper.readValue(file, objectMapper.getTypeFactory().constructCollectionType(List.class,
OfficeWorker.class));
        } catch (Exception e) {
            e.printStackTrace();
            return new ArrayList<>(); // Если не удалось считать данные, возвращаем пустой список
        }

    }
    @Override
    public OfficeWorker findById(Long id) {
        return findAll().stream().filter(good -> good.getId().equals(id)).findFirst().orElse(null);
    }
    @Override
    public void save(OfficeWorker officeworker) {
        throw new UnsupportedOperationException("Редактирование данных недоступно в режиме чтения из
файла.");
    }
    @Override
    public void deleteById(Long id) {
        throw new UnsupportedOperationException("Удаление данных недоступно в режиме чтения из файла.");
    }
    @Override
    public void updateOfficeWorker(Long id, OfficeWorker officeWorker) {
        throw new UnsupportedOperationException("Обновление данных недоступно в режиме чтения из
файла.");
    }

    @Override
    public boolean isDatabaseAccessible() {
        return false;
    }
```

```
    public void saveToFile(List<OfficeWorker> officeworkers) {

        try {
            ObjectMapper objectMapper = new ObjectMapper();
            // Регистрируем модуль для обработки Java 8 типов данных, включая LocalDate
            objectMapper.registerModule(new JavaTimeModule());
            objectMapper.writeValue(file, officeworkers);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    @Override
    public void backupToFile() {
        saveToFile(findAll());
    }
}
```

ДОДАТОК В

Код OfficeWorkerServiceImpl

```
package stusyo222b.appz6.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import stusyo222b.appz6.entities.OfficeWorker;
import stusyo222b.appz6.repository.OfficeWorkerRepository;

import java.util.List;

@Service
public class OfficeWorkerServiceImpl implements OfficeWorkerService {

    @Autowired
    private OfficeWorkerRepository officeWorkerRepository;

    @Autowired
    private FileOfficeWorkerService fileOfficeWorkerService;

    @Override
    public List<OfficeWorker> findAll() {
        return officeWorkerRepository.findAll();
    }
    @Override
    public OfficeWorker findById(Long id) {
        return officeWorkerRepository.findById(id).orElse(null);
    }

    @Override
    public void save(OfficeWorker officeworker) {
        officeWorkerRepository.save(officeworker);
    }
    @Override
    public void deleteById(Long id) {
        officeWorkerRepository.deleteById(id);
    }

    @Override
    public void updateOfficeWorker(Long id, OfficeWorker officeworker) {
        OfficeWorker officeWorkerToUpdateInDB = officeWorkerRepository.findById(id).orElse(null);
        if (officeWorkerToUpdateInDB != null) {
            officeworker.setId(id);
            officeWorkerRepository.save(officeworker);
        }
    }


    @Override
    public boolean isDatabaseAccessible() {
        try {
            officeWorkerRepository.findAll(); // Тестовый запрос к базе данных
            return true;
        } catch (Exception e) {
            return false; // Если запрос не удался, база данных считается недоступной
        }
    }
```

```
    public void backupToFile() {
        List<OfficeWorker> officeworkers = officeWorkerRepository.findAll();
        fileOfficeWorkerService.saveToFile(officeworkers);
    }
}
```

ДОДАТОК Г

Код класу OfficeWorkerServiceProxy

```
package stusyo222b.appz6.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Primary;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Service;
import stusyo222b.appz6.entities.OfficeWorker;

import java.util.List;

@Service
@Primary
public class OfficeWorkerServiceProxy implements OfficeWorkerService {


    @Autowired
    private OfficeWorkerServiceImpl databaseService;

    @Autowired
    private FileOfficeWorkerService fileService;
  @Autowired
  private OfficeWorkerServiceImpl officeWorkerServiceImpl;

  @Override
    @PreAuthorize("hasAuthority('ROLE_USER') or hasAuthority('ROLE_ADMIN')")
    public List<OfficeWorker> findAll() {
        if (databaseService.isDatabaseAccessible()) {
            List<OfficeWorker> officeworkers = databaseService.findAll();
            fileService.saveToFile(officeworkers);
            return officeworkers;
        } else {
            return fileService.findAll();
        }
    }

    @Override
    @PreAuthorize("hasAuthority('ROLE_USER') or hasAuthority('ROLE_ADMIN')")
    public OfficeWorker findById(Long id) {
        return databaseService.isDatabaseAccessible() ? databaseService.findById(id) : fileService.findById(id);
    }

    @Override
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public void save(OfficeWorker officeworker) {
        if (databaseService.isDatabaseAccessible()) {
            databaseService.save(officeworker);
        } else {
            throw new UnsupportedOperationException("Редактирование данных недоступно в режиме чтения из
файла.");
        }
    }

    @Override
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public void deleteById(Long id) {
```

```
        if (databaseService.isDatabaseAccessible()) {
            databaseService.deleteById(id);
        } else {
            throw new UnsupportedOperationException("Удаление данных недоступно в режиме чтения из
файла.");
        }
    }

    @Override
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public void updateOfficeWorker(Long id, OfficeWorker officeworker) {
        if (databaseService.isDatabaseAccessible()) {
            databaseService.updateOfficeWorker(id, officeworker);
        } else {
            throw new UnsupportedOperationException("Обновление данных недоступно в режиме чтения из
файла.");
        }
    }

    public boolean isDatabaseAccessible() {
        return databaseService.isDatabaseAccessible();
    }

    @Override
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public void backupToFile() {
        if (databaseService.isDatabaseAccessible()) {
            List<OfficeWorker> officeworkers = databaseService.findAll();
            fileService.saveToFile(officeworkers); // Сохраняем копию данных в файл
        } else {
            throw new UnsupportedOperationException("База данных недоступна для создания бэкапа.");
        }
    }
}
```

ДОДАТОК К

Код класу OfficeWorkerController

```
package stusyo222b.appz6.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.security.Principal;
import java.util.Arrays;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import stusyo222b.appz6.entities.OfficeWorker;
import stusyo222b.appz6.enums.OfficeWorkerStatus;
import stusyo222b.appz6.service.OfficeWorkerService;
import stusyo222b.appz6.service.OfficeWorkerServiceProxy;

@Controller
@RequestMapping("/officeworkers")
public class OfficeWorkerController {

    private final String WORKER_TEXT_INS = "NEW OFFICE WORKER (WJ2024)";
    private final String WORKER_TEXT_EDIT = "EDIT OFFICE WORKER (WJ2024)";
    private final OfficeWorkerService officeWorkerService;
    public OfficeWorkerController(OfficeWorkerService officeWorkerService) {
        this.officeWorkerService = officeWorkerService;
    }


    private static final Logger logger = LoggerFactory.getLogger(OfficeWorkerController.class);


    @Qualifier("officeWorkerServiceProxy")
    @Autowired
    private OfficeWorkerService officeworkerService;

    @GetMapping
    public String listOfficeWorkers(Model model, Principal principal) {
        model.addAttribute("username", principal.getName());
        List<OfficeWorker> listOfficeWorkers = officeworkerService.findAll();
        model.addAttribute("officeworkers", listOfficeWorkers);

        boolean isOfflineMode = !((OfficeWorkerServiceProxy) officeworkerService).isDatabaseAccessible();
        model.addAttribute("isOfflineMode", isOfflineMode);

        if (isOfflineMode) {
            model.addAttribute("message", "You are offline. The data is loaded from the file. Editing and deletion are
not available.");
        } else if (listOfficeWorkers.isEmpty()) {
            model.addAttribute("message", "No goods available to display.");
        } else {
            model.addAttribute("officeWorkerStatuses", Arrays.asList(OfficeWorkerStatus.values()));
```

```
    }

    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String role = authentication.getAuthorities().toString(); // Получаем роли пользователя
    model.addAttribute("role", role);

    logger.info("User '{}' with roles {} accessed the goods page.", principal.getName(), role);

    return "officeworkers/officeworkers";
}


@GetMapping("/new")
public String createOfficeWorkerForm(Model model) {
    System.out.println("Go to insert new office worker");
    OfficeWorker newOfficeWorker = new OfficeWorker("");
    model.addAttribute("officeworker", newOfficeWorker);
    model.addAttribute("titleOfficeWorker", WORKER_TEXT_INS);
    model.addAttribute("errorString", null);
    return "redirect:/officeworkers";
}

@GetMapping("/edit/{idEdit}")
public String editOfficeWorkerForm(@PathVariable Long idEdit, Model model) {
    System.out.println("Go to edit office worker with id=" + idEdit);

    // Получаем сотрудника по ID из базы данных
    OfficeWorker workerForUpdateInDB = officeWorkerService.findById(idEdit);

    // Если сотрудник не найден, перенаправляем на список сотрудников с ошибкой
    if (workerForUpdateInDB == null) {
        model.addAttribute("errorString", "Сотрудник с таким ID не найден.");
        return "redirect:/officeworkers";  // Перенаправление на список сотрудников
    }

    // Передаем данные сотрудника в модель для отображения в форме редактирования
    model.addAttribute("officeworker", workerForUpdateInDB);
    model.addAttribute("titleOfficeWorker", "Редактирование сотрудника");
    model.addAttribute("errorString", null);  // Если ошибки нет

    // Возвращаем имя представления для редактирования сотрудника
    return "officeworkers/officeworker";  // Здесь предполагается, что
}


@PostMapping("/delete")
public String deleteOfficeWorker(@RequestParam("id") Long id, Model model) {
    try {
        officeworkerService.deleteById(id);
    } catch (UnsupportedOperationException e) {
        model.addAttribute("error", "Operation not supported in offline mode.");
        return "redirect:/officeworkers";
    }
    return "redirect:/officeworkers";
}

@PostMapping("/backup")
public String backupOfficeWorkers(Model model) {
    try {
        officeworkerService.backupToFile();
        model.addAttribute("message", "Данные успешно сохранены в файл.");
    } catch (Exception e) {
        model.addAttribute("error", "Ошибка при сохранении данных в файл.");
```

```
        }
        return "redirect:/officeworkers";
    }
}
```

ДОДАТОК Л

Код сторінки officeworkers.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Office Workers Management</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;
    }

    .container {
      width: 90%;
      margin: 20px auto;
      padding: 20px;
      background-color: white;
      border-radius: 8px;
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
    }

    .filters {
      display: flex;
      justify-content: space-between;
      flex-wrap: wrap;
      margin-bottom: 20px;
    }

    .filters div {
      flex: 1;
      margin: 10px;
    }

    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: 5px 0 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;
    }

    .offline-warning {
      padding: 10px;
      background-color: #f8d7da;
      color: #721c24;
      border: 1px solid #f5c6cb;
      border-radius: 5px;
      margin-bottom: 20px;
    }

    @media (max-width: 768px) {
      .filters div {
        flex-basis: 100%;
      }
    }
  </style>
</head>

<body>
<h1>Office Workers Management</h1>
<div class="container">

  <!-- Offline Mode Warning -->
  <div th:if="${isOfflineMode}" class="offline-warning">
    <p>You are offline. Data is loaded from a file. Editing and deletion are unavailable.</p>
```

```
  </div>

  <!-- Filters and Search -->
  <div class="filters">
    <div>
      <label for="pl_filter"><b>Filter by Status:</b></label>
      <select id="pl_filter" name="pl_filter" onchange="filterByPL()">
        <option value="all">All</option>
        <option th:each="status : ${T(stusyo222b.appz6.enums.OfficeWorkerStatus).values()}" th:value="${status}"
                th:text="${status.getDisplayName()}"></option>
      </select>
    </div>
    <div>
      <b>Sort by:</b><br />
      <button onclick="sortTableByIndex(0)">Surname</button>
      <button onclick="sortTableByDate(3)">Start Date</button>
    </div>
    <div>
      <label for="search-text"><b>Search by Surname:</b></label>
      <input type="text" id="search-text" placeholder="Search..." onkeyup="tableSearch()">
    </div>
    <div>
      <form action="" method="get">
        <button type="submit">Reset</button>
      </form>
    </div>
  </div>

  <!-- Action Buttons -->
  <div th:if="${!isOfflineMode and role.contains('ROLE_ADMIN')}">
    <form action="/officeworkers/new" method="get" style="display: inline-block;">
      <button type="submit">Insert Office Worker</button>
    </form>
    <form th:action="@{/users}" method="get" style="display: inline-block;">
      <button type="submit">Users</button>
    </form>
    <form th:action="@{/officeworkers/backup}" method="post" style="display: inline-block;">
      <button type="submit">Backup</button>
    </form>
  </div>

  <!-- Table -->
  <table id="officeworkers-table">
    <thead>
    <tr>
      <th>Surname</th>
      <th>Name</th>
      <th>Patronymic Name</th>
      <th>Start Date</th>
      <th>End Date</th>
      <th>Status</th>
      <th>Worker Code</th>
      <th th:if="${!isOfflineMode and role.contains('ROLE_ADMIN')}">Update</th>
      <th th:if="${!isOfflineMode and role.contains('ROLE_ADMIN')}">Delete</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="worker : ${officeworkers}">
      <td th:text="${worker.surname}"></td>
      <td th:text="${worker.name}"></td>
      <td th:text="${worker.pname}"></td>
      <td
th:text="${worker.startWork.format(T(java.time.format.DateTimeFormatter).ofPattern('dd.MM.yyyy'))}"></td>
```

```
    <td th:if="${worker.endWork != null}"

th:text="${worker.endWork.format(T(java.time.format.DateTimeFormatter).ofPattern('dd.MM.yyyy'))}"></td>
    <td th:if="${worker.endWork == null}">Active</td>
    <td th:text="${worker.officeWorkerStatus.getDisplayName()}"></td>
    <td th:text="${worker.workerCod}"></td>

    <td th:if="${!isOfflineMode and role.contains('ROLE_ADMIN')}">
      <form th:action="@{/officeworkers/edit/{id} (id=${worker.id})}" method="get">
        <button type="submit">Update</button>
      </form>
    </td>

    <td th:if="${!isOfflineMode and role.contains('ROLE_ADMIN')}">
    <form th:action="@{/officeworkers/delete}" th:value="${worker.id}" method="post" style="display:inline;"
onsubmit="return confirm('Are you sure you want to delete this item?');">
      <input type="hidden" name="id" />
      <button type="submit">Delete</button>
    </form>
    </td>
   </tr>
   </tbody>
  </table>
</div>

<script>
  function confirmDelete() {
    return confirm('Are you sure you want to delete this record?');
  }

  function tableSearch() {
    const phrase = document.getElementById('search-text').value.toLowerCase();
    const rows = document.querySelectorAll('#officeworkers-table tbody tr');
    rows.forEach(row => {
      const surname = row.cells[0].innerText.toLowerCase();
      row.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() {
  const filterValue = document.getElementById('pl_filter').value.toLowerCase();
  const rows = document.querySelectorAll('#officeworkers-table tbody tr');
  rows.forEach(row => {
    const status = row.cells[5].innerText.toLowerCase();
    row.style.display = filterValue === 'all' || status === filterValue ? '' : 'none';
  });
}
</script>
</body>

</html>
```