Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мережних технологій факультету інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Розробка системи управління персоналом університету/ Development of the human resources system**»

Виконав: студент 4-го року навчання,

Спеціальності 122

Комп’ютерні науки

Кривошея Олександр Олександрович

Керівник: Шабінська М.О.

доктор технічних наук

Рецензент _____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК_____

“_____” _____ 2024 р.

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій,

_____

“___” _____ 2023 р.

ЗАВДАННЯ

ДЛЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТЦІ

Кривошеї Олександра Олександровича

1. Тема роботи: Розробка системи управління персоналом університету/ Development of the human resources system

   керівник роботи Шабінська Марина Олегівна

   затверджені наказом вищого навчального закладу від «__»_____20__року №____
2. Строк подання студентом роботи _____
3. План роботи _____

4. **Календарний план виконання роботи**:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи		
2.	Пошук тематичної літератури		
3.	Ознайомлення з літературою		
4.	Створення плану роботи		
5.	Написання теоретичної частини роботи		
6.	Подання першої версії записки науковому керівнику		
7.	Розробка застосунку для дослідження		
8.	Описання практичної частини роботи		
9.	Проведення дослідження		
10.	Аналіз дослідження та висновки		
11.	Перегляд змісту роботи керівником		
12.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника		

Студент Кривошея О.О.

Керівник Шабінська М.О.

“____” _____

ЗМІСТ

АНОТАЦІЯ

У даній роботі я запропонував власне бачення вирішення проблеми системи управління персоналом університету. Для даної роботи були обрані PostgreSQL у якості бази даних[1], Java у якості мови для backend частини[2] та JavaScript у якості мови для frontend частини програми[3]. Вкінці також буде продемонстровано результати роботи.

ВСТУП

Системи управління персоналом є важливою частиною ефективного функціонування будь-якої організації[4]. Його мета — раціональне використання людських ресурсів і підвищення продуктивності праці.

У сучасних умовах ринкова конкуренція продовжує загострюватися, а вимоги до якості продукції та послуг продовжують змінюватися. Ефективне управління персоналом стало стратегічно важливим завданням будь-якої компанії[5]. Системний та ефективний підхід до управління персоналом дозволяє підприємствам досягти вищого рівня професіоналізму та ефективності співробітників.

Предметною областю даної роботи є кваліфікаційні картки. У них записується уся важлива інформація про те, до якого із видів персоналу належить та, чи інша людина, про накази, пов’язані із даною людиною, її посадою, статусом, структурним підрозділом та, власне, самим підрозділом, тощо.

Метою даної роботи є створення системи, що допоможе покращити продуктивність та спростити задачу, що стоятиме перед користувачем.

Об’єктом дослідження є сам процес створення даної системи. Для цього завдання були використані PostgreSQL у якості реляційної бази даних, Java, як мова програмування, яка забезпечуватиме обробку запитів від клієнтської сторони, взаємодією з базою даних та надаватиме відповіді на запити. Також, у якості мови програмування, що застосовується для клієнтської частини є JavaScript.

Дана робота містить наступні розділи:

1. Перший розділ описує умову поставленого завдання та обгрунтовує його складність.
2. Другий розділ описує бази даних, їх створення та власні думки стосовно роботи із PostgreSQL.
3. Третій розділ розповідатиме про основні моменти про взаємодію серверної частини системи із базою даних та клієнтською частиною.
4. Четвертий розділ розповідатиме про клієнтську частину системи.

РОЗДІЛ 1. АНАЛІЗ ТА ОБГРУНТУВАННЯ СКЛАДНОСТІ ЗАВДАННЯ.

1.1 Аналіз

Перед початком завдання були висунуті наступні вимоги стосовно того, які дані мають зберігатися і у якому вигляді.

Поле	Пояснення
Прошу мене	Відповідно до слова, яке тут обране, буде генеруватись заява та накази. Випадаючий список: • прийняти, • перевести, • звільнити, • продовжити термін роботи
ПІБ	Називний відмінок Має підтягнутись з кадрової картки
Персонал	Випадаючий список: + НПП + НП + ПП + АП

Посада ШР	
Посада альтернативна	Як ця посада має світитись на сайті
Посада розрахункова	Потрібно для таких ситуацій: якщо працівник на посаді доцента, але не має вченого звання "доцента", то зарплату він отримує як старший викладач, наприклад (не як доцент).
Статус	Випадаючий список: • постійний склад, • сумісник, • внутрішній сумісник, • звільнено • погодинно
Структурний підрозділ	Вибір між факультетами
Підрозділ	Наприклад: кафедра математики
Сектор	Заповнюється тільки в НП, в інших пусте поле Випадаючий список: • Навчальна лабораторія пост-бакалаврських студій • Науково-дослідна частина • Науково-дослідна, аналітично-

	вимірювальна лабораторія контролю якості води, аналізу стічної води та осадів • Науково-дослідний Центр психічного здоров'я та психосоцільного-супроводу НаУКМА • Науково-освітній центр "Школа політичної аналітики" • Центр досліджень історії та культури східноєвропейського єврейства
Cdoc	Шестизначне число. Генерується системою автоматично. Приклад: 342312
Дата заяви	Прописується арабськими цифрами у форматі: число, місяць, рік. Де число і місяць проставляються двома парами цифр, а рік чотирма. За замовчуванням стоїть сьогодняшня дата.
Трудова книжка	Чотирьохзначне число. Поле не обов'язкове
Зараховано з	Період "з ____ по ____" Кожна дата прописується арабськими

	цифрами у форматі: число, місяць, рік. Де число і місяць проставляються двома парами цифр, а рік чотирма.
Зараховано фактично з	Період "з ____ по ____" Кожна дата прописується арабськими цифрами у форматі: число, місяць, рік. Де число і місяць проставляються двома парами цифр, а рік чотирма.
Доплати інші	Випадаючий список: • за виконання обов'язків декана • за виконання обов. завідувача кафедри • за виконання обов. заступника декана • за використання в роботі дезінфікувальних засобів • за класність водія 1 класу • за класність водія 2 класу.
Фінансування заробітньої плати	Випадаючий список: • бюджет • спецфонд-контрактне навчання
Тип	Випадаючий список з можливістю додати текст в ручну.

На період	Випадаючий список: • бібліотечний стаж • до оголошення результатів конкурсу • з терміном випробування • відпустки, без збереження заробітньої плати • відпустки по догляду за дитиною, до дня її фактичного виходу з відпустки • до- та післяпологової відпустки • хвороби • обрано за конкурсом
І наказ	Наказ з яким кваліфікаційна картка створилась: або наказ на зарахування, або наказ на переведення. Вказується у двох полях: • номер • дата
ІІ наказ	Наказ з яким кваліфікаційна картка перевелась в архів: або наказ про переведення, або наказ про звільнення. Вказується у двох полях: • номер

	• дата
Загальна ставка у підрозділі округлена	числове поле
Ставка	

Таблиця 1.1 Поля та пояснення до них

Одразу можна зробити певні висновки стосовно того, яка таблиця повинна вийти в результаті, і як це має виглядати. Наприклад, поля «Зараховано з» та «Зараховано фактично з» можна розділити на два поля. Перша частина міститиме у собі дату початкову, тоді як друга – кінцеву. Але, наприклад, накази можна вписати у одне поле, адже документ повинен мати свою дату. Також, один документ не може мати декілька дат створення, тоді як різні документи можуть мати одну дату. Також, замість ПІБ можна одразу вказувати кадрову картку. У ній є уся необхідна інформація стосовно того, про кого була створена кваліфікаційна картка.

При цьому, архітектура даної системи матиме наступний вигляд:

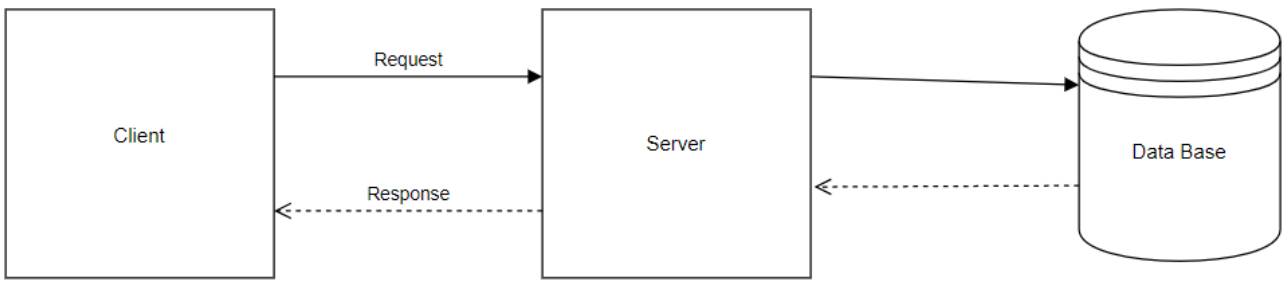


Рисунок 1.1. Діаграма архітектури системи

1.2 Обґрунтування складності

1.2.1 Валідація і перевірка даних

Перевірка та перевірка даних має вирішальне значення для розробки будь-якої системи на базі даних, особливо це стосується системи управління персоналом університету[6].

Перевірка даних використовується для забезпечення правильності та повноти інформації на етапі введення. Наприклад, система має переконатися, що дати мають належний формат, що адреси електронної пошти можна відстежувати та що всі необхідні поля введено[7]. Без належної перевірки користувачі можуть вводити невірні дані, що призведе до помилок у системі та ще більше ускладнить процес.

Як наслідок, валідація даних та їх верифікація є вирішальними для успішного впровадження системи управління персоналом університету. Вони гарантують точність, послідовність і важливість інформації, що, в свою чергу, підвищує ефективність і надійність всієї системи[8]. Це складна процедура, яка вимагає ретельного планування, технічного виконання та постійного нагляду, але її цінність не можна недооцінювати.

1.2.2 Користувацький інтерфейс

Розробка зручного інтерфейсу для управління персоналом в університетах є надзвичайно важливою, але це також складний процес[9]. Розглядайте інтерфейс користувача як вікно, яке користувачі використовують для взаємодії з системою. Якщо це вікно не буде практичним, зрозумілим і функціональним, то вся система стане неефективною, незважаючи на те, що «під капотом»[10]. Давайте обговоримо причини створення інтерфейсу для користувачів більш детально.

По-перше, система управління персоналом університету використовується багатьма різними класами користувачів: адміністраторами, викладачами, дослідниками, студентами та іншими. Кожна з цих груп унікально складається з потреб і необхідного рівня технічної кваліфікації. Наприклад, адміністраторам може знадобитися доступ до певних звітів і статистичних даних, а вчителям потрібен швидкий доступ і можливість оновлювати свої освітні дані[11]. Створення інтерфейсу, який був би зрозумілим і практичним для всіх цих груп, є серйозною проблемою.

Інтерфейс користувача повинен бути простим і ефективним, щоб користувачі могли легко і без труднощів знаходити потрібну інформацію. Це вимагає ретельного розгляду структури меню, розміщення кнопок і полів, а також використання піктограм і підказок, які є інформативними та ефективними[12]. Поганий дизайн може призвести до плутанини та невдоволення користувачів, що негативно впливає на продуктивність.

Окрім функціональності, інтерфейс має бути привабливим і візуально. Правильний дизайн, який не тільки сприяє використанню системи, але й наповнює користувача позитивним ставленням до неї. Це означає, що ви повинні знайти компроміс між естетикою та функціональністю, використовувати приємні колірні схеми, зручні шрифти та інші компоненти дизайну[13].

Сучасний інтерфейс повинен мати різноманітні функції, тому він успішний на різних пристроях і розмірах екранів, від настільних ПК до смартфонів. Це означає, що дизайнери та розробники повинні враховувати різні платформи та пристрої під час розробки інтерфейсів і робити їх практичними незалежно від пристрою. Це збільшує складність розробки ще на один шар[14].

Система управління персоналом в університеті має бути ефективною та багатогранною, що дозволить з часом додавати нові функції та нарощувати можливості. Це означає, що інтерфейс має бути розроблений таким чином, щоб нові компоненти могли бути включені без значного впливу на загальний дизайн. Це вимагає модульного підходу до розробки інтерфейсів[15].

Щоб створити функціональний інтерфейс, важливо проводити експерименти з реальними користувачами та враховувати їхні відгуки. Це полегшує виявлення слабких місць і недосконалостей у дизайні та своєчасне їх виправлення. Однак цей процес є ресурсномістким і трудомістким, кожна ітерація тестування та вдосконалення потребуватиме часу[16].

Як наслідок, розробка інтерфейсу користувача для кадрової системи університету вважається складною з багатьох причин. Важливо врахувати різноманітність користувачів, забезпечити простоту використання та інтуїтивно зрозумілу поведінку, мати візуальну привабливість, зробити інтерфейс адаптивним та мобільним, а також легко оновлювати та розширювати. Крім того, необхідні постійне тестування та збір відгуків, щоб переконатися, що система дійсно задовольняє потреби своїх користувачів. Усі ці аспекти сприяють складному, але важливому завданню розробки інтерфейсу користувача.

РОЗДІЛ 2. БАЗА ДАНИХ

2.1 Визначення та застосування бази даних

База даних — це організована колекція даних, яка зазвичай зберігається на комп’ютері чи сервері, до якої можна легко отримати доступ, оновлювати та керувати нею. Бази даних використовуються для зберігання інформації в структурованому форматі, щоб забезпечити ефективний доступ до цих даних для подальшого використання.

У базі даних інформація може бути організована у вигляді таблиці, де кожен рядок представляє окремий запис, а кожен стовпець представляє окремий атрибут або характеристику. Бази даних дозволяють виконувати різні операції, такі як додавання нових даних, оновлення, видалення та пошук, дозволяючи зберігати, упорядковувати та аналізувати великі обсяги інформації.

2.2 Визначення та застосування реляційної бази даних

«Реляційна база даних» (RDB) — це форма бази даних, організована як набір таблиць, що мають стовпці та рядки[17]. Кожна таблиця в реляційній базі даних має унікальне ім’я та містить записи (рядки), кожна з яких має певний набір атрибутів (стовпців).

У RDB таблиці можна пов’язувати одна з одною за допомогою ключів, які керують зв’язуванням записів у різних таблицях. Це полегшує ефективне управління даними та організацією, забезпечуючи їх цілісність, надійність і доступність.

Багато популярних систем керування базами даних, наприклад MySQL, PostgreSQL, Oracle і Microsoft SQL Server, є похідними від реляційної моделі даних.

2.3 PostgreSQL

PostgreSQL — це потужна система для керування реляційними даними, яка підтримує ширший діапазон функцій, ніж стандарт SQL. PostgreSQL випускається як безкоштовна база даних із відкритим кодом, і її можна використовувати як для малих, так і для великих завдань. Він відомий своєю надійністю, стабільністю, розширюваністю та розширеними атрибутами, які полегшують вирішення широкого спектру проблем зберігання та обробки даних.

PostgreSQL підтримує різні типи даних, включаючи числа, рядки, дати, JSON, географічну інформацію тощо. Він також примітний тим, що має багатий набір функцій, який включає транзакції, відновлення, реплікацію, індексування, тригери, повнотекстовий пошук, тощо.

2.4 Реалізація бази даних у PostgreSQL

2.4.1 Основна таблиця

№	Поле	Тип даних	Null/not null(NN)
1	ask_for	Varchar(50)	NN
2	personnel_card_id	Integer	NN
3	staff	Varchar(5)	NN
4	position_sr	Varchar(100)	NN

5	alternative_position	Varchar(100)	Null
6	calculated_position	Varchar(100)	Null
7	status	Varchar(50)	NN
8	structural_unit	Varchar(50)	NN
9	unit	Varchar(50)	NN
10	sector	Varchar(100)	Null
11	cdoc	Serial(6)	NN, PK
12	application_date	Date	NN
13	employment_record	Integer(4)	Null
14	included_from	Date	NN
15	included_to	Date	NN
16	actually_included_from	Date	Null
17	actually_included_to	Date	Null
18	other_allowances	Varchar(50)	Null
19	salary_funding	Varchar(50)	NN
20	type	Varchar(150)	NN
21	time_period	Varchar(100)	NN

22	order_1	Integer	NN
23	order_2	Integer	Null
24	total_rate_in_unit	Integer	Null
25	rate	Integer	Null

Таблиця 2.1. Таблиця кваліфікаційної картки

На таблиці можемо побачити абсолютно усі поля, що передбачаються для даної роботи. Усього у ній 25 полів, які можна заповнити певними даними. Із них не обов'язковими є тільки 10 полів. Типи даних продиктовані умовою поставленої переді мною завдання.

2.4.2 Допоміжні таблиці першого типу

Поле	Тип даних	PK
value	Varchar	true

Таблиця 2.2. Приклад основних допоміжних таблиць

Основні допоміжні таблиці призначені для зберігання значень, що можна внести у основну таблицю. Вони дозволяють простіше додавати потенційні можливі варіанти для їх занесення до таблиці кваліфікаційних карток без втручання у серверну та/або клієнтську частину. Також це пришвидшує додання варіанту для безпосереднього його використання. Такі таблиці були створені для наступних номерів полів: 1, 3, 7, 8, 10, 18, 19, 20, 21. Максимальний розмір кожного із значень, що записуються у дані таблиці відповідає значенням із основної таблиці.

2.4.3 Допоміжна таблиця наказів

Поле	Тип даних	Null/NN
id	Integer	NN, PK
Date	Data	NN

Таблиця 2.3. Таблиця наказів

Дана таблиця емулює базу даних, де зберігаються усі накази. Відповідно, перше поле відповідає за номер наказу, тоді як у полі «дата» записується дата, коли даний наказ був підписаний.

2.4.4 допоміжна таблиця персоналу

Поле	Тип даних	Null/NN
id	Integer	NN, PK
last_name	Varchar(50)	NN
first_name	Varchar(50)	NN
patronymic	Varchar(50)	NN

Таблиця 2.4. Таблиця персоналу

Дана таблиця емулює кадрову картку. Саме тому тут зберігається ПІБ.

2.5 Результати

Таким чином було створено одинадцять таблиць, в кожній з яких були описані поля типи даних, які можуть там бути записані. Також, саме тут відбувається основна валідація даних, адже якщо спробувати записати у числове поле текст, то

результатом цього стане помилка про невідповідність типів даних. Це дозволяє менше акцентувати уваги на цьому при написанні серверної частини системи.

РОЗДІЛ 3. СЕРВЕРНА ЧАСТИНА СИСТЕМИ

3.1 Застосування серверної частини системи

Частина системи управління персоналом, яка включає сервери, має вирішальне значення для загальної ефективності та ефективності системи. Вона відповідає за зберігання, обробку та надання доступу до інформації про співробітників організації. Основними обов'язками сервера є:

1. Обробка даних: відповідає за обробку даних, організацію інформації та аналіз[18].
2. Надання доступу: серверна частина системи забезпечує доступ до інформації через веб-інтерфейси або спеціалізовані інтерфейси програмування (API). Це полегшує доступ до інформації для різних користувачів, включаючи керівництво, спеціалістів з кадрів, менеджерів і співробітників, які всі мають отримувати інформацію[19].

Зважаючи на це, архітектура серверної частини буде наступною:

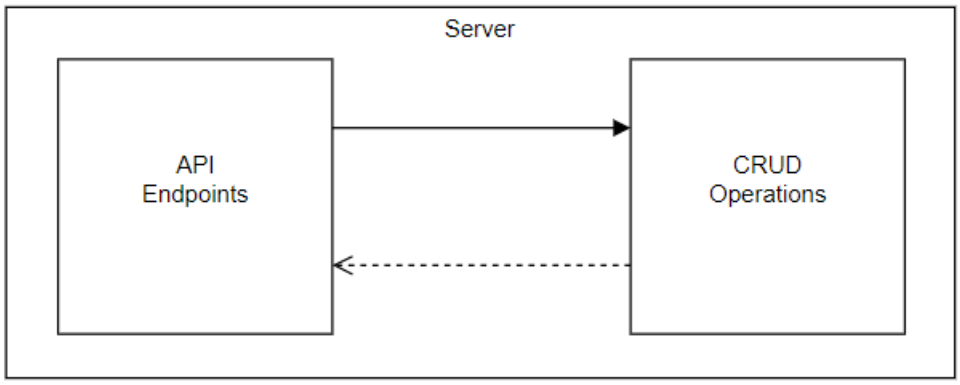


Рисунок 3.1. Діаграма архітектури серверної частини системи

3.2 Реалізація взаємодії з базою даних

3.2.1 Підключення до бази даних

Для того, аби підключитися до необхідної бази даних, було достатньо (для Maven проекту):

1. У pom.xml додати залежність postgresql:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.6.0</version>
</dependency>
```

Рисунок 3.2. Приклад залежності

2. Підключитися до потрібної бази даних

3.2.2 Ознайомлення з CRUD операціями

CRUD (Create, Read, Update, Delete) — це елементарні операції, які можна застосувати до даних у базі даних або іншій системі зберігання даних. Ось короткий опис кожної процедури:

Create (Створити) - операція, яка створює нові записи або об’єкти в базі даних або іншій платформі. Після процедури нова інформація зберігається в системі.

Read (Читання) - процедура, яка отримує інформацію з бази даних або іншого джерела. Це може включати отримання однієї або кількох подій відповідно до певних вимог.

Update (Оновлення) - процедура, яка дозволяє вам змінювати існуючі дані або об’єкти в базі даних або іншій платформі. Після процедури інформація в системі переглядається на основі внесених змін.

Delete (Видалити) - процедура, яка дає змогу видаляти записи або об’єкти з бази даних чи іншої системи зберігання. Після процедури вибраний запис або об'єкт більше не є частиною системи.

CRUD є основою для багатьох програм і систем керування даними, тому що він надає прості засоби взаємодії з даними. Більш складні операції можна вивести з цих чотирьох основних операцій. Наприклад, операція Create буде використана для створення функції «Додати користувача» в системі управління персоналом, операція Update буде використана для зміни інформації користувача, операція Read буде використана для перегляду списку співробітників, а операція Delete буде застосовано операцію для видалення облікового запису працівника.

3.2.3 Написання CRUD операцій

Зважаючи на описану базу даних можна одразу визначити які поля варто заповнювати. Там ми можемо побачити, що поле cdoc генерується автоматично, що означає, що нам дане поле заповнювати не потрібно. Усі інші поля повинні мати змогу бути записаними до бази даних.

3.2.3.1 Create

Необхідно реалізувати два запити на створення запису у двох таблицях, адже важливо запам’ятовувати значення поля type, якщо його ще не занесено у відповідну таблицю.

Однак із таблицею кваліфікаційної картки не все так просто. У ній дуже багато полів, типи яких також досить сильно різняться. Тому не потрібно забувати про можливу проблему невідповідності типів. Наприклад, не можна записати значення текстове у поле, яке повинне мати числове значення. Як видно із таблиці, де описані усі поля таблиці кваліфікаційної картки, у нас є чотири поля,

які можуть не мати значення, які у таблиці є числовими. Як відомо, у мові Java не можливо призначити числу значення null.

Вирішення даної проблеми можливе наступним чином. Я можливо побачити із полів, дані у них мають бути невід'ємними. Саме тому, я перевіряю, чи є дані із сторони сервера. Якщо вони є, то я просто трансформую іх у відповідний тип даних. Якщо ж значення у ньому немає, то я записую «-1» і трансформую уже дане значення до необхідного мені типу.

Також не варто забувати про поля, значення яких має бути date. Це окремий тип значень, а, отже, нам доведеться їх обробляти так само, як і числові значення. На щастя, з датами усе значно простіше. Їх можна перетворювати одразу при надсиланні запиту. З цим нам допоможе функція "TO_DATE(value, format)" де:

Value – це рядок, яке відповідає тому, що потрібно записати.

Format – формат рядка дати, адже вони можуть бути різними.

Ще можна помітити, що значення null при даному перетворенні будуть записуватися у базу даних, як 01.01.0001. щоб цього не допустити можна використовувати конструкцію "CASE WHEN … THEN … ELSE … END". Вона дозволяє нам обробляти ситуації, коли значення дати є null. Цю саму конструкцію я використовую при перевірці значень для числових полів, що можуть мати значення Null. Якщо поступає значення менше нуля, то записується Null. Інакше, записується саме значення.

Для NUMERIC важливо перевіряти на меншість із «0.0», а не з «0», бо останній вважається цілочисельним типом, а тому порівнювати із ним неможливо.

3.2.3.2 Read

Таких типів операцій потрібно зробити для усіх таблиць, що тільки були зроблені. Для більшості із них достатньо просто підставити назву таблиці. Однак для тих таблиць, що повертають нам значення дат, було б непогано одразу переводити їх у більш зручний для усіх нас формат запису дати, а саме «ДД.ММ.РРРР». Нагадаю, що у PostgreSQL дата зберігається у форматі «РРРР-ММ-ДД». Для цього можна використати функцію to_char(name, format) as formatted_name, де:

Name – назва поля, яке приведе значення, що у ньому записане до певного формату.

Format – формат вигляду дати.

Також у нас є числові поля, які записуються декількома цифрами. Ми також можемо їх одразу отримати із таблиці бази даних у необхідному вигляді. Для цього можна використати функцію LPAD(name::text, n, '0') AS formatted_name, де:

Name – назва поля, яке спочатку буде конвертоване до текстового типу даних, після чого додаватиме ‘0’, доки рядок не стане потрібної довжини.

N – кількість символів, з яких має складатися рядок, що отримуємо після запиту.

3.2.3.3 Search

Хоча даний тип запиту не є основним, адже практично повністю схожий на операцію Read, однак це також є важливим запитом. Коли у таблиці буде сотні записів, то обов’язково виникне ситуація, при якій необхідно буде знайти один якийсь об’єкт.

Варіант запиту «WHERE name = value» повністю задовільнить лише тоді, коли ми точно знаємо, що шукати. Але якщо значення ми точного не знаємо, то можуть виникнути значні проблеми, бо тоді ми просто не знайдемо об’єкт, який шукаємо.

LIKE дозволяє шукати об’єкти, у яких є схожість із тим який ми шукаємо. Звісно, при цьому якість результату падає. Однак у даній ситуації це є виправданим.

ILIKE також ігнорує регістр значень полів, які є у таблиці. Це може не аби як спростити процес пошуку потрібного зам запису.

3.2.3.4 Update

У даній операції проблеми усі ті самі, що і у операції CREATE. Також потрібно слідкувати за типами значень, що записуються у таблицю.

3.2.3.5 Delete

У даному випадку ніяких ускладнень не було, адже ключове значення у основної таблиці одне.

3.3 Реалізація взаємодії з клієнтською частиною

3.3.1 Загальні відомості про API Endpoints

Кінцеві точки API — це точки доступу до вашої веб-програми або сервера, через які клієнти можуть спілкуватися з вашою програмою чи службою[20]. Кожна кінцева точка API пов’язана з певним шляхом (URL), який клієнти можуть використовувати для запиту інформації та отримання відповідей.

URL (шлях) - це адреса, за якою буде доступний API. Він пропонує конкретний ресурс або атрибут, з яким клієнти хочуть асоціюватися[21].

Кожна кінцева точка може підтримувати різні методи запиту, наприклад GET, POST, PUT і DELETE. Це називається методами запиту, або методами HTTP.

Кожен такий метод пов’язаний із певною дією, у яку клієнт має змогу зробити з ресурсом[22].

Параметр запиту - це інформація, яку клієнт може ввести в API для виконання певної дії. Наприклад, це можуть бути параметри запиту GET або дані, які передаються через тіло запиту POST[23].

Кожна кінцева точка має повернути відповідь на запит. Це може бути JSON, XML, HTML або інші формати даних, які повертаються клієнту разом із результатами процедури[24].

Після обробки запиту сервер повертає код статусу HTTP, який вказує на успішність або невдачу процедури. Наприклад, код статусу 200 вказує на те, що запит було виконано успішно, а код статусу 404 вказує на те, що запит був невдалим через відсутність ресурсу[25].

Як правило, кінцеві точки API є основним засобом, за допомогою якого клієнти взаємодіють із програмою чи службою через Інтернет. Вони визначають спосіб передачі та отримання даних і відіграють значну роль у створенні розподілених систем і мікросервісів[26].

3.3.2 Реалізація кінцевих точок

Отже, потрібно створити усі кінцеві точки, що будуть приймати усі можливі запити на серверну частину сервісу. Це, відповідно, усі GET запити для отримання усіх даних з усіх таблиць, а також два запити POST для оновлення інформації та пошуку потрібного нам об’єкту, PUT для створення нового об’єкта та для можливості видалення об’єкту потрібен запит типу DELETE. Результат реалізації виглядає подібним чином:

```
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/qualification_cards", OperationType.READ_QUALIFICATION);
storageContext.mapEndpointToOperation( requestMethod: "PUT", uri: "/api/qualification_cards", OperationType.CREATE_QUALIFICATION);
storageContext.mapEndpointToOperation( requestMethod: "POST", uri: "/api/qualification_card/{cdoc}", OperationType.UPDATE_QUALIFICATION);
storageContext.mapEndpointToOperation( requestMethod: "DELETE", uri: "/api/qualification_card/{cdoc}", OperationType.DELETE_QUALIFICATION);

storageContext.mapEndpointToOperation( requestMethod: "POST", uri: "/api/qualification_cards", OperationType.READ_QUALIFICATION);

storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/ask_for", OperationType.READ_ASK_FOR);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/orders", OperationType.READ_ORDERS);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/other_allowances", OperationType.READ_OTHER_ALLOWANCES);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/personnel_card", OperationType.READ_PERSONNEL_CARD);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/salary_funding", OperationType.READ_SALARY_FUNDING);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/sector", OperationType.READ_SECTOR);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/staff", OperationType.READ_STAFF);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/status", OperationType.READ_STATUS);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/structural_unit", OperationType.READ_STRUCTURAL_UNIT);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/time_period", OperationType.READ_TIME_PERIOD);
storageContext.mapEndpointToOperation( requestMethod: "GET", uri: "/api/type", OperationType.READ_TYPE);
```

Рисунок 3.3. Створення API Endpoints

3.4 Результати

Отже, у цій частині були описані процеси написання CRUD операцій та створення API Endpoints. Перші дозволяють робити запити до бази даних, де зберігається усі необхідні дані. Друге необхідне для зв’язку із клієнтською частиною системи. Це дозволяє обробляти запити користувача та реагувати на його дії, що пов’язані з інформацією, що зберігається у базі даних.

РОЗДІЛ 4. КЛІЄНТСЬКА ЧАСТИНА СИСТЕМИ

4.1 Загальні відомості

Клієнтська частина системи в основному відповідає за взаємодію користувача з системою. Це інтерфейс, через який користувачі можуть спілкуватися з можливостями системи, виконувати різні завдання та отримувати відповідну інформацію[27]. Основні аспекти клієнтської сторони кадрової системи включають:

1. Інтерфейс користувача: це основний компонент на стороні клієнта. Інтерфейс має бути простим, інтуїтивно зрозумілим і ефективним для користувачів. Він може містити різні компоненти, такі як меню, кнопки, форми введення, таблиці з інформацією, графіки, тощо. Типовими інструментами для даної роботи є HTML та CSS технології[28].
2. Функціональні можливості: Клієнт повинен полегшити користувачам виконання різноманітних дій, таких як перегляд, додавання, редагування та видалення інформації про персонал, створення звітів, тощо[29]. Для даної роботи підходять HTML та CSS технології.
3. Клієнтська сторона взаємодіє з серверною частиною для отримання та зміни даних, проведення транзакцій бази даних і обробки запитів клієнта[30]. Для цього можна використати JavaScript[31].

4.2 JavaScript

4.2.1 JavaScript у клієнтській частині системи

JavaScript дає змогу робити асинхронні запити до сервера без оновлення сторінки. Це сприяє збільшенню продуктивності та більш швидкому реагуванню на запити

користувачів, наприклад, при заповненні списку співробітників або оновленні інформації.

Також за його допомогою можливо змінювати вміст або структуру сторінки під час її роботи, це дозволяє динамічно оновлювати вміст без необхідності перезавантажувати сторінку. Наприклад, ви можете додати нових співробітників до списку співробітників або видалити дані зі списку без повторного завантаження[32].

Не варто забувати, що JavaScript можна використовувати для створення анімації та візуалізації інформації, що може покращити розуміння користувача та зробити його привабливішим[33].

Як правило, використання JavaScript у клієнтській частині системи полегшує створення інтерактивного динамічного інтерфейсу для користувачів, що покращує їхній досвід використання програми[34].

4.2.2 Використання JavaScript у програмі

Отже, кожне значення, що клієнт буде вводити та/або обирати на сторінці, будуть оброблятися саме у даній частині системи, після чого будуть передаватися на сервер для подальшої обробки. Кожне отримане значення нам потрібно окремо записати для коректної її обробки у подальшому. Для цього були створені усі можливі змінні та константи для зчитування даних, що будуть введені користувачем. Також вони прив’язуються до елементів, що розташовані у HTML за допомогою id.

Були створені функції, які відправляють різні запити до серверу. Наприклад, коли користувач відкриє форму для заповнення інформації, то в цей момент будуть відправлені запити GET до усіх допоміжних таблиць на отримання актуальних

значень полів, що зберігаються у таблицях бази даних. Це дозволяє обирати найбільш актуальні значення, необхідні для створення кваліфікаційної картки.

Демонстрація форми створення картки також можлива саме завдяки JavaScript. Він зчитує дії клієнта, і при натисканні на необхідний об’єкт показує форму з можливими необхідними для заповнення полями. Те саме стосується форми для пошуку кваліфікаційних карток, де можна, ввівши необхідні дані, скоротити кількість карток, що зображуються на екрані, та форми для оновлення даних картки, якщо, наприклад, дані змінилися.

4.3 HTML

Найпростіший елемент, що використовується для введення даних є «input». Він може приймати будь які дані. Через цю свою особливість він є дуже популярним.

Також, зважаючи на умови, потрібно буде використовувати елемент “select”. Завдяки ньому користувач може ввести тільки ті дані, які є у опціях. Це дозволяє спростити обробку даних, що були введені користувачем та не дозволяє йому вводити власні дані.

Якщо подивитися на умови, то можна помітити, що необхідний також елемент, що працює, як селектор, однак також дозволяє вводити свої дані. Для цього можна використати «datalist» у зв’язці із “input”. У «datalist» будуть зберігатися дані, що уже заповнювались колись раніше, тоді як у «input» можна ввести власні дані.

Прикладом форми, що об’єднує усі ці елементи є наступний фрагмент коду HTML:

```
<div id="form_popup" class="form-popup" style="overflow-y: auto">
    <div class="form-container" >
```

```
<h2>Add Qualification Card</h2>

<form id="createForm">

  <label for="ask_for">Ask for:</label>

  <select id="askForIdSelectCreate" name="ask_for" required></select>


  <label for="personnel_card">Personnel Card:</label>

  <select id="personnelCardIdSelectCreate" name="personnel_card" required></select>


  <label for="staff" >Staff:</label>

  <select id="staffIdSelectCreate" name="staff" onchange="staffCheckCreate()"></select>


  <label for="position_sr">Position:</label>

  <input type="text" id="positionSrInputCreate" name="position_sr" required>


  <label for="alternative_position">Alternative Position:</label>

  <input type="text" id="alternativePositionInputCreate" name="alternative_position" required>
```

```
<label for="calculated_position">Calculated Position:</label>

<input type="text" id="calculatedPositionInputCreate" name="calculated_position">


<label for="status">Status:</label>

<select id="statusIdSelectCreate" name="status" required></select>


<label for="structural_unit">Structural Unit:</label>

<select id="structuralUnitIdSelectCreate" name="structural_unit" required></select>


<label for="unit" >Unit:</label>

<input type="text" id="unitInputCreate" name="unit">


<label for="sector" id="sectorLabelCreate" style="display:none;">Sector:</label>

<select id="sectorIdSelectCreate" name="sector" style="display:none;" required></select>
```

```
<label for="application_date">Application Date:</label>

<input type="text" id="formattedApplicationDateInputCreate" name="application_date" required>


<label for="employment_book">Employment Book:</label>

<input type="text" id="formattedEmploymentBookInputCreate" name="calculated_position">


<label for="included_from" >Included From:</label>

<input type="text" id="formattedIncludedFromInputCreate" name="included_from">


<label for="included_to">Included To:</label>

<input type="text" id="formattedIncludedToInputCreate" name="included_to" required>


<label for="actually_included_from">Actually Included From:</label>

<input type="text" id="formattedActuallyIncludedFromInputCreate" name="actually_included_from" required>
```

```
<label for="actually_included_to">Actually Included To:</label>

<input type="text" id="formattedActuallyIncludedToInputCreate" name="actually_included_to">


<label for="other_allowances" >Other Allowances:</label>

<select id="otherAllowancesIdSelectCreate" name="other_allowances"></select>


<label for="salary_funding">Salary Funding:</label>

<select id="salaryFundingIdSelectCreate" name="salary_funding" required></select>


<label for="time_period">Time Period:</label>

<select id="timePeriodIdSelectCreate" name="time_period"></select>


<label for="order_1" >First Order:</label>

<select type="text" id="order1SelectCreate" name="order_1"></select>


<label for="order_2">Second Order:</label>

<select type="text" id="order2SelectCreate" name="order_2" required></select>
```

```
        <label for="total_rate_in_unit">Total Rate In Unit:</label>
        <input type="text" id="totalRateInUnitInputCreate" name="total_rate_in_unit" required>

        <label for="rate">Rate:</label>
        <input type="text" id="rateInputCreate" name="rate">

        <label for="type">Type:</label>
        <datalist id="options"></datalist>
        <input list="options" id="typeIdInputCreate" name="type" required>

        <button type="submit" id="submitCreateFormButton" class="btn btn-primary">Add</button>
        <button id="closeFormButton" type="button" class="btn btn-delete">Cancel</button>
    </form>
  </div>
```

```
</div>
```

Додаток 4.1. Приклад форми для створення кваліфікаційної картки

Після заповнення усіх необхідних даних необхідно натиснути на «Add», після чого заповнені дані будуть відправлені на серверну частину клієнта. При вдалому запиті, форма самостійно закриється, інакше буде виведено повідомлення про помилку.

Для закриття форми створення кваліфікаційної картки достатньо натиснути «Cancel» і форма зникне.

Форми для пошуку та оновлення мають тільки рядки для заповнення інформації. При відкритті форми оновлення там уже буде введено усі дані, що уже були заповнені раніше. При закритті будь якої форми, усі дані, що там були введені, зникають.

4.4 CSS

У даному випадку за допомогою CSS були описані усі форми для заповнення інформації, вигляд, як зображуються кваліфікаційні картки, різного типу кнопки. Тобто усе, що бачить користувач при використанні даної системи.

Приклад застосування CSS у системі на прикладі форм для заповнення інформації:

```
.form-popup {
    display: none;
    position: fixed;
    z-index: 9999;
```

```
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
}

.form-container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background-color: #f2f2f2;
  padding: 20px;
  max-width: 400px;
  margin: 50px auto;
  border-radius: 4px;
}
```

Додаток 4.2. Приклад застосування CSS у системі

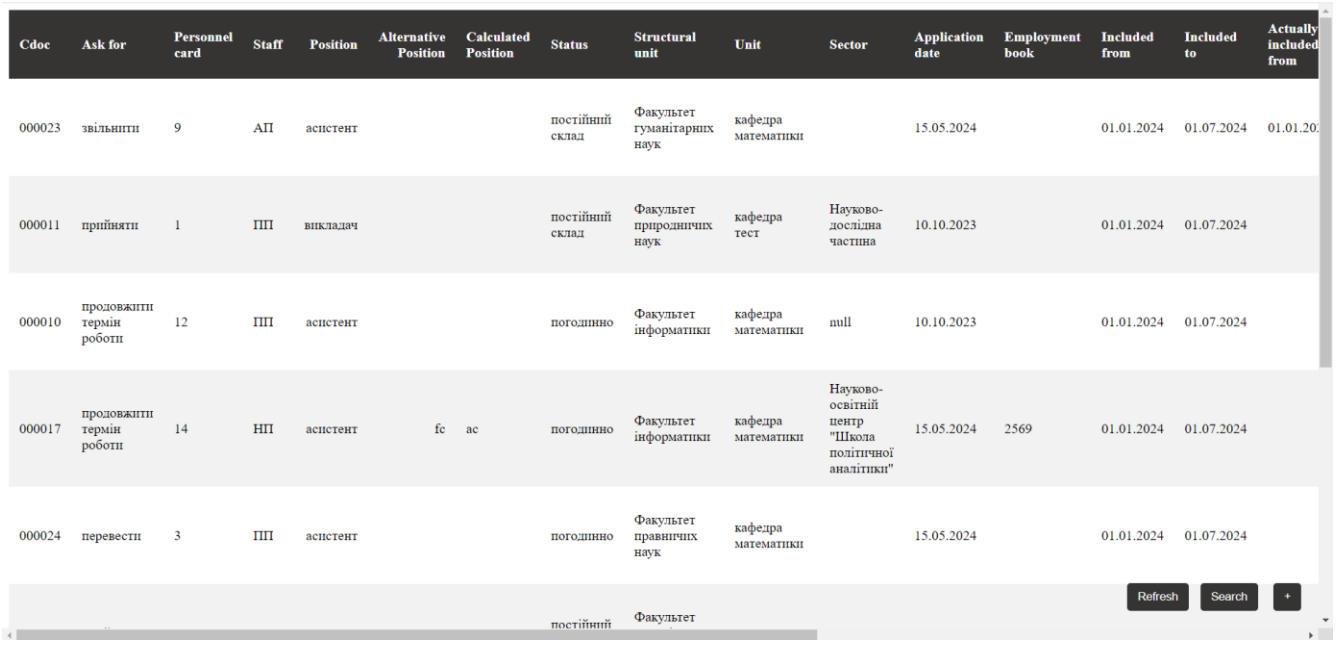
4.5 Результати

Отже, у цій частині було описано важливість клієнтської частини системи, адже саме її бачитимуть користувачі при її використанні. Також були наведені приклади використання різних елементів у HTML та, власне, приклад використання CSS у подібних програмах.

Результати

Результатом виконання даної роботи стала система, що дозволяє виконувати наступну роботу із картками наукового персоналу, педагогічного персоналу та адміністративного персоналу:

1. Переглядати усі створені кваліфікаційні картки
2. Створювати нові кваліфікаційні картки
3. Змінювати дані в уже існуючих кваліфікаційних картках
4. Виконувати пошук необхідних кваліфікаційних карток по даним, що містяться у них
5. Видаляти непотрібні кваліфікаційні картки



Cdoc	Ask for	Personnel card	Staff	Position	Alternative Position	Calculated Position	Status	Structural unit	Unit	Sector	Application date	Employment book	Included from	Included to	Actually included from
000023	звільнити	9	АП	асистент			постійний склад	Факультет гуманітарних наук	кафедра математики		15.05.2024		01.01.2024	01.07.2024	01.01.20
000011	прийняти	1	ПП	викладач			постійний склад	Факультет природничих наук	кафедра тест	Науково-дослідна частина	10.10.2023		01.01.2024	01.07.2024	
000010	продовжити термін роботи	12	ПП	асистент			погодинно	Факультет інформатики	кафедра математики	null	10.10.2023		01.01.2024	01.07.2024	
000017	продовжити термін роботи	14	НП	асистент	fc	ac	погодинно	Факультет інформатики	кафедра математики	Науково-освітній центр "Школа політичної аналітики"	15.05.2024	2569	01.01.2024	01.07.2024	
000024	перевести	3	ПП	асистент			погодинно	Факультет правничих наук	кафедра математики		15.05.2024		01.01.2024	01.07.2024	
							постійний	Факультет							

Refresh Search +

Рисунок 5.1. Приклад відображення карток, що занесені у систему.

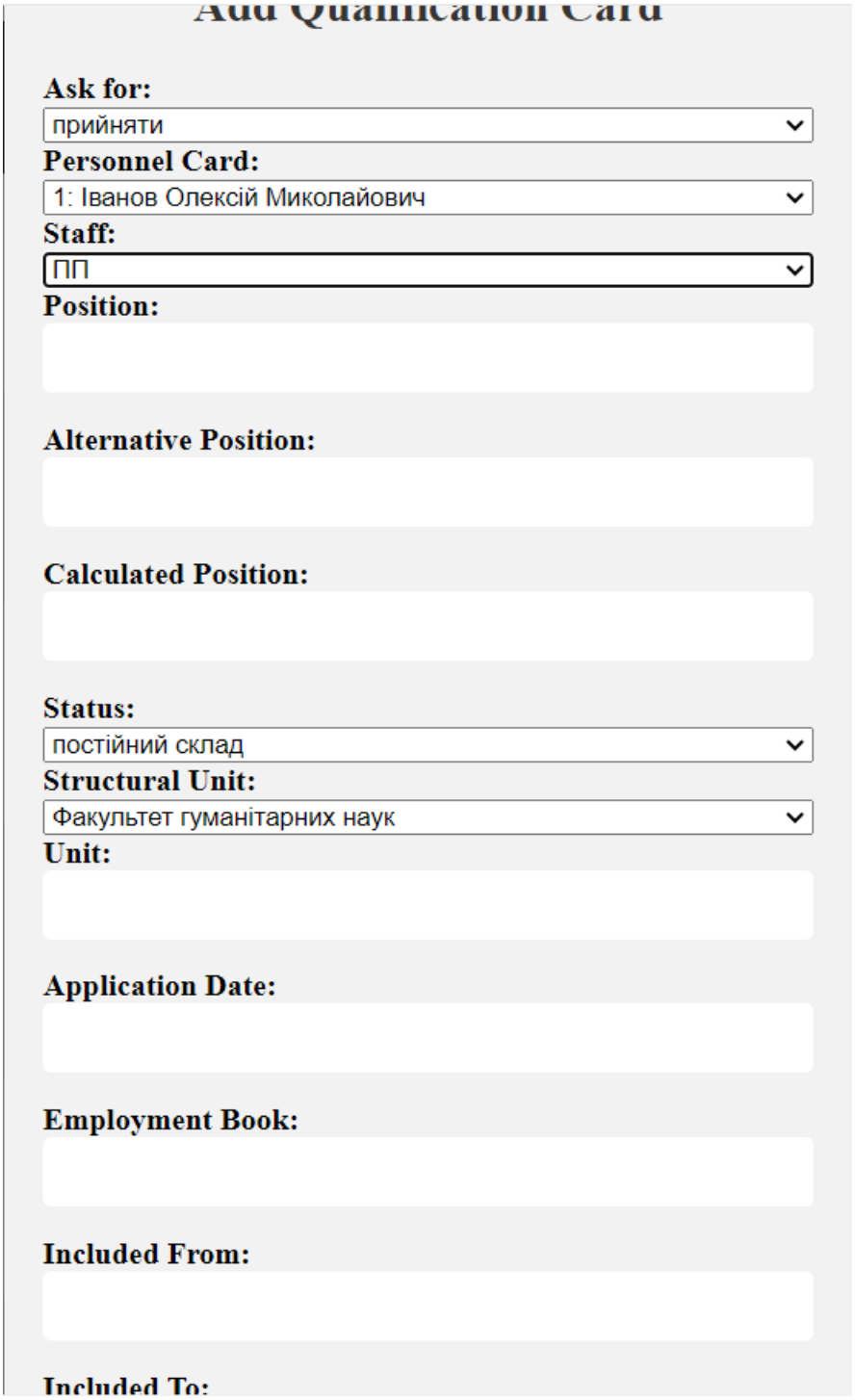


Рисунок 5.2. Приклад форми для створення кваліфікаційної картки

Висновки

Отже, під час даної роботи були створена система, що дозволяє взаємодіяти з кваліфікаційними картками адміністративного персоналу, педагогічного персоналу та наукового персоналу. Усе це було розписано у чотирьох розділах.

У першому розділі були розглянуті вимоги до системи. Також у ній описуються проблеми, що можуть виникнути при її створені.

У другому розділі було описано важливість використання бази даних. Вона дозволяє зберігати великі масиви даних не на комп'ютері користувача, а також дозволяє не залежно від пристрою мати однаковий набір даних для всіх. Також було описано створення бази даних, що відповідатиме вимогам, що стоять перед даною системою.

У третьому розділі були продемонстровані приклади написання операцій для надсилання запитів до створеної бази даних. Це є однією із основних частин системи, адже коректність її роботи дуже сильно залежить від правильності запиту, що буде отриманий базою даних. Також, у даному розділі було описано створення кінцевих точок. Вони повинні правильно обробляти дані, що приходять від клієнтської частини системи.

У четвертому розділі була надана інформація про те, чому клієнтська частина системи є не менш важливою за попередні. Так як користувач взаємодіятиме з нею, то важливо зробити її максимально простою та інтуїтивною. Також тут важливо правильно обробляти дані, що поступають від користувача. Якщо цього не зробити, то може виникнути почуття, ніби усе дуже складно.

В результаті була створена система, де користувач може:

1. Створення кваліфікаційної картки
2. Видалення кваліфікаційної картки
3. Оновлення даних, записаних у кваліфікаційній картці
4. Видалення непотрібних кваліфікаційних карток
5. Пошук потрібних кваліфікаційних карток в залежності від інформації, що в ній зберігається.

СПИСОК ЛІТЕРАТУРИ

1. PostgreSQL: The world's most advanced open source relational database. https://www.postgresql.org/
2. Java: The complete reference, Eleventh Edition. Herbert Schildt. McGraw-Hill Education, 2018.
3. JavaScript: The Definitive Guide, 7th Edition. David Flanagan. O'Reilly Media, 2020.
4. Armstrong, M. (2006). A Handbook of Human Resource Management Practice. Kogan Page Publishers.
5. Dessler, G. (2017). Human Resource Management. Pearson Education, 45-60.
6. Date, C. J. (2004). An Introduction to Database Systems. Pearson Education.
7. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database System Concepts. McGraw-Hill.
8. Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management. Pearson Education, 450-465.
9. Shneiderman, B., & Plaisant, C. (2010). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley.
10. Norman, D. A. (2013). The Design of Everyday Things: Revised and Expanded Edition. Basic Books, 45-62.
11. Johnson, J. (2014). Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules. Morgan Kaufmann, 101-120.
12. Cooper, A., Reimann, R., & Cronin, D. (2007). About Face 3: The Essentials of Interaction Design. Wiley, 78-95.
13. Tidwell, J. (2010). Designing Interfaces: Patterns for Effective Interaction Design. O'Reilly Media, 32-50.

14. Nielsen, J. (2012). Mobile Usability. New Riders Publishing, 23-45.
15. Raskin, J. (2000). The Humane Interface: New Directions for Designing Interactive Systems. Addison-Wesley, 200-215.
16. Krug, S. (2014). Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders Publishing, 123-140.
17. “Реляційні бази даних усе, що необхідно про них знати”. Foxminded, https://foxminded.ua/reliatsiini-bazy-danykh/
18. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems. Pearson Education.
19. Tanenbaum, A. S., & van Steen, M. (2017). Distributed Systems: Principles and Paradigms. Pearson Education, 220-235.
20. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, 77-85.
21. Jacobs, I., & Walsh, N. (2004). Architecture of the World Wide Web, Volume One. W3C, 25-32.
22. Richardson, L., & Ruby, S. (2008). RESTful Web Services. O'Reilly Media, 50-65.
23. Zambon, E., & Etalle, S. (2017). Engineering Secure Internet of Things Systems. Springer, 140-155.
24. Subramanian, M. (2014). Network Management: Principles and Practice. Pearson, 220-235.
25. Fielding, R. T., & Taylor, R. N. (2002). Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology, 110-125.
26. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 90-105.
27. Nielsen, J. (1993). Usability Engineering. Morgan Kaufmann, 65-80.
28. Duckett, J. (2011). HTML and CSS: Design and Build Websites. Wiley 50-65.

29. Few, S. (2006). Information Dashboard Design: The Effective Visual Communication of Data. O'Reilly Media, 90-105.
30. Gamperl, G., & Beck, P. (2019). Learning JavaScript: Add Sparkle and Life to Your Web Pages. Addison-Wesley, 140-155.
31. Flanagan, D. (2006). JavaScript: The Definitive Guide. O'Reilly Media, 210-225.
32. McFarland, D. (2011). JavaScript & jQuery: The Missing Manual. O'Reilly Media, 150-165.
33. Marijn Haverbeke (2018). Eloquent JavaScript: A Modern Introduction to Programming. No Starch Press, 180-195.
34. Freeman, E., & Robson, E. (2014). Head First JavaScript Programming. O'Reilly Media, 100-115.