МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА ПРИЛАДОБУДУВАННЯ

ДО ЗАХИСТУ ДОПУЩЕНО
Завідувач кафедри
Чигура І. І.
«18» грудня 2022 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної магістерської роботи

на тему:

ІНТЕГРОВАНА СИСТЕМА КАДРОВОГО ЗАБЕЗПЕЧЕННЯ ПІДПРИЄМСТВА

Виконав:

Кічковський Михайло Михайлович
(прізвище, ім'я, по-батькові) (підпис)

Науковий керівник:

професор Іваницький В. П.
(вчене звання, ПІБ, посада) (підпис)

Ужгород – 2022

РЕФЕРАТ

Пояснювальна записка кваліфікаційної магістерської роботи: 98 сторінок, 6 таблиць, 32 рисунок, 3 додатки,14 джерел посилань.

ПІДПРИЄМСТВО, ІНТЕГРОВАНА СИСТЕМА, ПРАЦІВНИК, БАЗА ДАННИХ

Мета роботи полягає в оптимізації інтегрованої системи кадрового забезпечення підприємств.

У роботі наведено короткий аналіз перспектив та проблем розвитку сучасних систем кадрового забезпечення підприємств та організацій. Зроблено порівняння наявних на ринку різних програмних продуктів управління кадрами. Показано, що перспективним напрямком розвитку таких систем є їх інтеграція в більш загальні комп'ютерно-інтегровані системи, які об'єднують в собі бази даних різних організацій, наприклад, підприємства, служби зайнятості, керівних органів тощо.

Описано процес оптимізації типової кадрової системи забезпечення підприємства, яка забезпечує її легку інтеграцію в розширені комп'ютерно-інтегровані системи з доступом до інформації користувачів різного рівня з різних підприємств та організацій. Розроблено архітектуру оптимізованої кадрової системи, сформована концептуальна модель її бази даних та розроблено програмне забезпечення,

Властивості та параметри оптимізованої системи кадрового забезпечення:

- мова програмування JavaScript;
- фреймворки Express та MaterialUi;
- бібліотека React;
- СУБД Mongo Atlas, яка забезпечує надійну обробку даних;
- використання веб-інтерфейсу,
- незалежність від операційної системи.

На оптимізовану кадрову систему оформлено опис програми; керівництво системного програміста; керівництво користувача та текст програми.

ABSTRACT

Explanatory note of the qualifying master's thesis: 98 pages, 6 tables, 32 figures, 3 appendices, 14 reference sources.

ENTERPRISE, INTEGRATED SYSTEM, EMPLOYEE, DATABASE

The purpose of the work is to optimize the integrated system of personnel support of enterprises.

The work provides a brief analysis of the prospects and problems of the development of modern human resources systems of enterprises and organizations. A comparison of various personnel management software products available on the market was made. It is shown that a promising direction for the development of such systems is their integration into more general computer-integrated systems that combine databases of various organizations, for example, enterprises, employment services, management bodies, etc.

The process of optimization of a typical personnel support system of an enterprise is described, which ensures its easy integration into advanced computer-integrated systems with access to information of users of different levels from different enterprises and organizations. The architecture of the optimized personnel system was developed, the conceptual model of its database was formed, and the software was developed,

Properties and parameters of the optimized staffing system:

- JavaScript programming language;
- Express and MaterialUi frameworks;
- React library;
- DBMS Mongo Atlas, which provides reliable data processing;
- use of the web interface,
- independence from the operating system.

A description of the program was created for the optimized personnel system; management of the system programmer; user manual and program text.

Ужгородський національний університет

Інженерно-технічний факультет
Кафедра приладобудування
Освітньо-кваліфікаційний рівень "Магістр"
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ
Завідувач кафедри
к.ф.-м.н. Чичура І.І.

"18" грудня 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Кічковському Михайлу Михайловичу
(ПРІЗВИЩЕ, ІМ'Я, ПО БАТЬКОВІ)

1. Тема роботи «Інтегрована система кадрового забезпечення підприємства» та керівник роботи Іваницький Валентин Петрович, професор,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Розпорядженням № 3 по ІТФ від "31" 1 2022 року.

2. Строк подання студентом роботи: "16" грудня 2022 року.

3. Вихідні дані до роботи мова програмування JavaScript, фреймоврк ExpressJs, бібліотека ReactJs та MaterialUI, база даннних MongoDb, чиста архітектура, процес вносення працівників в базу.

4. Перелік питань, які потрібно розробити: проаналізувати існуючі інформаційні системи кадрового забезпечення підприємств; спроектувати структуру інформаційної системи та бази даних для неї; провести вибір архітектурних рішень та засобів розробки; розробити власну систему, яка дасть змогу керувати кадрами; створити рекомендації для розробки цілої системи в подальшому.

5. Орієнтований перелік ілюстративного матеріалу: структура інформаційної системи, концептуальна модель бази даних, інфологічна модель бази даних, архітектура системи кадрового забезпечення.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання: 20 вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1	Отримання завдання	**28.09.2022**	
2	Опрацювання літературних джерел	**01.11.2022**	
3	Підготовка матеріалів дисертації	**01.12.2022**	
4	Підготовка доповідей на конференції	**05.12.2022**	
5	Розробка програмного продукту	**10.12.2022**	
6	Захист програмного продукту.	**15.12.2022**	
7	Передзахист	**15.12.2022**	
8	Оформлення дисертації	**20.12.2022**	

Студент _____ (підпис) / Київець / (ініціали та прізвище)

Керівник роботи _____ (підпис) / Іванчук / (ініціали та прізвище)

Гарант ОП «Магістр» _____ (підпис) / Іванчук / (ініціали та прізвище)

ЗМІСТ

Розробив	Кічковський М.		Інтегрована система кадрового забезпечення підприємства	Літера	Аркуш	Аркушів
Перевірив	Іваницький В.П..			у	6	98
Т. контроль				ІТФ, кафедра ПБ, денна форма		
Н. контроль	Чичура І.І.					
Затвердив	Чичура І.І.					

ВСТУП

Стрімкий розвиток комп'ютерної техніки, застосування її в усіх галузях, повсякденному житті, можливість сучасного комп'ютера зберігати, представляти та обробляти будь-яку інформацію, зумовило їх використання і в різних видах людської діяльності, де вже стало традиційним використовувати інформаційні системи як для технічних і адміністративних задач, так і для управління різними подіями й для безпосереднього інформаційного пошуку.

Інформатизація у сфері управління трудовими ресурсами на підприємствах на сьогодні є конче необхідною, оскільки значно зменшує ризики управлінських економічних помилок та невиправданих фінансових затрат. Від того, на скільки правильно та досвідчено налаштована інформаційна система управління персоналом в організації, суттєво залежить ефективність її роботи та отримувані прибутки.

Впровадження комп'ютерних інформаційних систем управління економічними процесами, пов'язаними з кадровими процесами на підприємствах, забезпечує швидкий і безперервний документообіг та ефективний рух фінансів. Це дозволяє не тільки вивільнити додаткову кількість робочого часу для кадрових працівників, але і створює передумови для розробки нових якісних стратегій управління всіма кадрами та для здійснення ефективних процесів планування в управлінні як підприємством в цілому, так і окремими його структурними елементами.

Що стосується інформаційних комп'ютерних систем кадрового забезпечення, то така організаційно-розпорядча документація закріплює трудові правовідносини громадян з установами та підприємствами.

Однією з основних частин спеціального діловодства на підприємствах та в організаціях є кадрове діловодство. Ведеться воно за правилами та принципами, встановленими для загального діловодства.

Виходячи з технології реєстрації та накопичення інформації з кадрів, групу первинної облікової кадрової документації поділяють на дві

взаємопов'язані підгрупи: вихідні (початкові) облікові документи та похідні (повторні) облікові документи.

Вихідні облікові документи заповнюють відповідно до особистих документів громадян. Це: особовий листок з обліку кадрів, доповнення до особового листка, особова картка та інші спеціалізовані облікові документи.

Похідні облікові документи мають другорядний характер. Основне їх призначення - забезпечити повну, достовірну інформацію з усіх напрямів довідкової, довідково-контрольної та звітної роботи з кадрів. До цієї підгрупи облікових документів відносять: картки спеціалізованого обліку спеціалістів, журнальні (книжкові) форми реєстрації (вказівні списки, книга обліку та інші).

Термін "кадри" у перекладі з французької означає професіонали, які займаються тією чи іншою діяльністю, працею у тій чи іншій системі, галузі, на тому чи іншому підприємстві.

Кадрове діловодство визначається як діяльність, що охоплює питання документування та організації роботи з документами стосовно особового складу підприємства (чи системи) з питань приймання, переведення, звільнення, обліку працівників тощо. Правильна організація кадрового діловодства має велике значення.

Саме у кадрових службах громадяни укладають трудовій договір, ознайомлюються з правилами внутрішнього розпорядку, умовами праці, побуту, відпочинку, перспективами професійного зростання. Кадрова служба є дзеркалом установи, і від того, як у ній організоване документаційне забезпечення управління, складається враження про установу в цілому.

Кадрове діловодство ведеться у таких напрямах:

- облік особового складу установи та її підрозділів;

- підготовка звітів та необхідних довідок про переведення кадрів, розробка та виготовлення необхідних форм та бланків для цього;

- облік стану підготовки, перепідготовки кадрів та зарахування їх до резерву;

- облік та реєстрація надходження документів, які стосуються особового складу, контроль за їх виконанням, насамперед виконання наказів та розпоряджень по кадрах;

- організація документообігу для забезпечення оперативного і чіткого виконання та проходження документів і доручень керівництва стосовно всіх напрямів роботи з кадрами, додержання загальних та специфічних для кадрової роботи правил складання та виконання документів;

- складання номенклатурних справ з кадрового діловодства, їх оформлення та ведення;

- підготовка документів з кадрів для передачі в архів на зберігання;

- механізація, автоматизація і комп'ютерна обробка даних з особового складу.

Збільшення обсягів виробництва, перебудова управління економікою викликали значне збільшення обсягу інформації у сфері управління кадрами. Щороку зростає кількість службових документів. Впровадження їх машинної обробки у ряді служб великих підприємств і об'єднань не скорочує кількість персоналу.

Для вирішення цієї проблеми потрібна правильна організація та механізація: обробки документів, від якої залежить ефективність процесу управління кадрами. Оптимізаціїкомп’ютерно-інтегрованої системи для вирішення вказаних завдань і присвячена дана магістерська робота. При цьому мета даного дослідження є визначення принципів та методів побудови системи управління кадрами на підприємстві, визначення її місця в системі управління підприємством, а також ролі кадрового забезпечення в системі управління кадрами.

1СИСТЕМА КАДРОВОГО ЗАБЕЗПЕЧЕННЯ ЯК ОСНОВНИЙ ЕЛЕМЕНТ СИСТЕМИ УПРАВЛІННЯ

1.1 Основні положення

Підприємство – невід'ємна частина економіко-соціальної системи з економічними, науково-технічними, виробничими та соціальними цілями. Система управління кадрами, будучи основною частиною системи управління підприємством, є вирішальною умовою, що дозволяє забезпечити досягнення цілей підприємства та зберегти його стійкість у ринковому середовищі. Система управління кадрами забезпечує ефективне функціонування підприємства, яке у теперішніх умовах цілком залежить від якості його основного ресурсу – кадрів.

Особливо зараз, коли модель людини являє собою комплексну модель – "людину економічну", "людину психологічну", "людину соціальну" [1-3], відсутність знань й навичок її застосування керівником підприємства може стати значною перешкодою на шляху до створення якісного потенціалу кадрів й забезпечення надійності діяльності підприємства.

Передусім вважаємо доцільним конкретизувати основні поняття, що будуть використовуватися в роботі.

Кадри – основна одиниця організаційно-економічного механізму підприємства; працівники підприємства зі складним комплексом економічних, соціальних, психологічних якостей, а також професійнокваліфікаційними, статтю, віковими та іншими характеристиками.

Кадрове забезпечення підприємства – здатність підприємства до реалізації потреб у людських ресурсах шляхом проведення низки заходів з метою досягнення в певних умовах намічених результатів.

Управління кадрами підприємства – цілеспрямована діяльність керівництва підприємства, керівників й спеціалістів підрозділів системи управління кадрами на інтереси, поведінку й діяльність працівників з метою

максимального використання їх потенціалу при виконанні трудових функцій [4, 5].

Система управління кадрами є реалізатором функцій управління кадрами та являє собою сукупність ув'язаних, погоджених методів й засобів управління кадрами підприємства, які покликані упорядкувати, організувати й направити на досягнення мети діяльність кадрів. Система управління кадрами передбачає формування цілей, функцій, організаційної структури управління кадрами, вертикальних й горизонтальних функціональних взаємозв'язків керівників й спеціалістів у процесі обґрунтування, розробки, прийняття й реалізації управлінських рішень [5]. Головною метою системи управління кадрами є кадрове забезпечення підприємства, ефективне використання кадрів, а також їх професійний й соціальний розвиток.

Розробка й впровадження системи управління кадрами на підприємстві припускає існування сформульованої місії підприємства, на підставі якої зафіксовані його цілі й цінності. Також необхідний "єдиний корпоративний стандарт робочої поведінки кадрів (кодекс корпоративної поведінки), завдяки якому будуть досягатися поставлені цілі з одночасним прямуванням цінностям" [4], або, іншими словами, необхідна філософія управління кадрами, яка інтегрована у філософію підприємства.

Під філософією підприємства слід розуміти сукупність внутрішніх організаційних принципів, моральних та адміністративних норм та правил взаємовідносин кадрів, систему цінностей та переконань, яку сприймають всі кадри, та яка підпорядкована глобальній меті підприємства [5]. Філософія управління кадрами є невід'ємною частиною філософії підприємства, її основою. Сутність філософії управління кадрами полягає в тому, що працівники мають можливість задовольнити свої особисті потреби, працюючи на даному підприємстві.Філософія управління кадрами розглядає процес управління кадрами з логічної, психологічної, соціологічної, економічної, організаційної й етичної точок зору [5].

Завданням системи управління кадрами в філософському змісті є задоволення потреб у комплектуванні кадрами підприємства та в найбільш повному задоволенні потреб працівників. Система управління кадрами будується на ряді принципів і методів, наведених в табл.1.1 [6, 7].

Таблиця 1.1 – Огляд принципів і методів управління кадрами

Принципи, що характеризують вимоги до формування системи управління кадрами	**Принципи, що визначають напрями розвитку системи управління кадрами**
1. Обумовленості функцій управління кадрами цілям виробництва 2. Первинності функцій управління кадрами 3. Оптимального співвідношення функцій, направлених на організацію системи управління кадрами й функції управління кадрами 4. Потенційних імітацій – уміння кожного працівника імітувати функції вищестоящого, нижчестоящого співробітника й функції робітників свого рівня 5. Економічності – найбільш ефективна й економічна система управління кадрами 6. Прогресивності 7. Перспективності 8. Комплексності 9. Оперативності 10. Оптимальності 11. Простоти 12. Науковості 13. Ієрархічності 14. Автономності 15. Погодженості 16. Стійкості 17. Багатоаспектності 18. Прозорості 19. Комфортності	1. Концентрації – концентрація зусиль працівників окремого відділення або всієї системи управління кадрами на вирішенні основних задач 2. Спеціалізації 3. Паралельності 4. Адаптивності (гнучкості) 5. Спадкоємності – загальна методична основа проведення роботи по удосконаленню системи управління кадрами на різних рівнях та різними спеціалістами 6. Безперервності 7. Ритмічності 8. Прямоточності – упорядкованість і цілеспрямованість необхідної інформації за виробкою певного рішення

Як слідує із таблиці, методи побудови системи управління кадрами підприємства включають:

- обстеження системи (самообстеження, інтерв'ювання, активне спостереження робочого дня, моментні спостереження, анкетування, вивчення документів, функціонально-вартісний аналіз);

- аналіз системи (системний аналіз, економічний аналіз, декомпозиція, послідовна підстановка, зрівняння, динамічний, структуризації цілей, експертно-аналітичний, нормативний, параметричний, моделювання, функціонально-вартісного аналізу, головних компонент, балансовий, кореляційний і регресійний аналіз, дослідний, матричний);

- реформування системи (системний підхід, аналогій, експертно-аналітичний, параметричний, блоковий, моделювання, функціонально-вартісного аналізу, структуризації цілей, дослідний, творчих нарад, колективного блокноту – "банк" ідей, контрольних питань, морфологічний аналіз);

- обґрунтування прогресивних рішень (аналогій, зрівнянь, експертно-аналітичний, моделювання фактичного й бажаного становищ досліджуваного об'єкту, розрахунок кількісних та якісних показників оцінки економічної ефективності пропонованих варіантів, функціонально-вартісного аналізу);

- впровадження системи (вивчення, перепідготовка й підвищення кваліфікації працівників апарату керування, матеріальне й моральне стимулювання нововведень; залучення суспільних організацій, функціонально-вартісного аналізу).

1.2 Види кадрового забезпечення та кадрова служба

Для свого функціонування система управління кадрами підприємства потребує організаційного забезпечення, інформаційного та технічного забезпечення, нормативно-методичного та правового забезпечення. Під організаційним забезпеченням розуміють сукупність взаємопов'язаних підрозділів системи управління кадрами й посадових осіб. Підрозділи, як носії функцій управління кадрами, можуть розглядатися в широкому значенні у

вигляді кадрової служби. Конкретне місце і роль цієї служби в загальній системі управління підприємством визначаються місцем і роллю кожного спеціалізованого підрозділу з управління кадрами та організаційним статусом його безпосереднього керівника. Цей організаційний статус підтверджується набором повноважень і відповідальності, їх обсяг та ієрархічний ранг багато в чому визначаються позицією першого керівника підприємства щодо кадрової служби. Вони також формуються в міру організаційного розвитку управління, нагромадження, фінансового, кадрового та інтелектуального потенціалу.

У цілому, під інформаційним забезпеченням системи управління кадрами розуміють сукупність реалізованих рішень по обсягу, розміщенню та формам організації інформації, що циркулює в системі управління при її функціонуванні [5]. Інформаційне забезпечення, у свою чергу, включає оперативну інформацію, нормативно-довідкову інформацію, класифікатори техніко-економічної інформації та системи документації (уніфіковані та спеціальні). При проектуванні та розробці інформаційного забезпечення системи управління важливим є установлення состава й структури інформації, необхідної для прийнятої технології управління. Інформаційне забезпечення служби управління кадрами підрозділяється на поза-машинне та внутрішньо-машинне.

Інформаційна частина вимагає і серйозного технічного забезпечення. Під технічним забезпеченням системи управління кадрами підприємства розуміють комплекс технічних засобів – сукупність взаємопов’язаних єдиним керуванням та (або) автономних технічних засобів збору, реєстрації, накопичення, передачі, обробки, виводу й поданняінформації, а також засобів організаційної техніці [5].

Комплекс технічних засобів дозволяє вирішувати задачі керування з мінімальними трудовими й вартісними затратами, із заданою точністю й вірогідністю у задані строки. Він повинен володіти інформаційною, програмною та технічною сумісністю вхідних засобів; адаптуванням до умов

функціонування служби управління кадрами; можливістю розширення з метою підключення нових пристроїв.

Під нормативно-методичним забезпеченням системи управління кадрами розуміють сукупність документів організаційного, організаційно-методичного, організаційно-розпорядницького, технічного, нормативно-технічного, техніко-економічного й економічного характеру, а також нормативно-довідкові матеріали, що встановлюють норми, правила, вимоги, характеристики, методи та інші дані, які використовуються при рішенні задач організації праці та управління кадрами та які затверджені в певному порядку компетентним відповідним органом або керівництвом підприємства [5]. Нормативно-методичне забезпечення створює умови для ефективного процесу підготовки, прийняття й реалізації рішень щодо питань управління кадрами.

Правове забезпечення системи управління кадрами складається з використання засобів та форм юридичного впливу на органи та об'єкти управління кадрами з метою досягнення ефективної діяльності підприємства [5]. Основним завданням правового забезпечення системи управління кадрами є правове регулювання трудових відносин між роботодавцем та найманими робітниками, захист прав та законних інтересів працівників, що випливають з трудових відносин.

Систему управління кадрами вміщує не тільки функціональні підрозділи, які займаються роботою з кадрами, але й вище керівництво, а також керівництво функціональних підрозділів, що виконують функції науково-технічного, виробничого, економічного керівництва, управління зовнішніми господарськими зв'язками й кадрами, ось чому її неможливо відокремити від системи управління підприємством. Система управління кадрами є базисом побудови системи управління підприємством (рисунок 1.1). Кожна з підсистем системи управління має головне завдання. Основним завданням підсистеми керівництва системи управління підприємством є розробка та реалізація філософії підприємства, основним завданням функціональних підсистем – розробка та реалізація різних видів політики підприємства. Підсистема

управління кадрами, будучи головним елементом системи управління підприємства, розробляє та реалізує кадрову політику підприємства, основою якої є процес кадрового забезпечення.

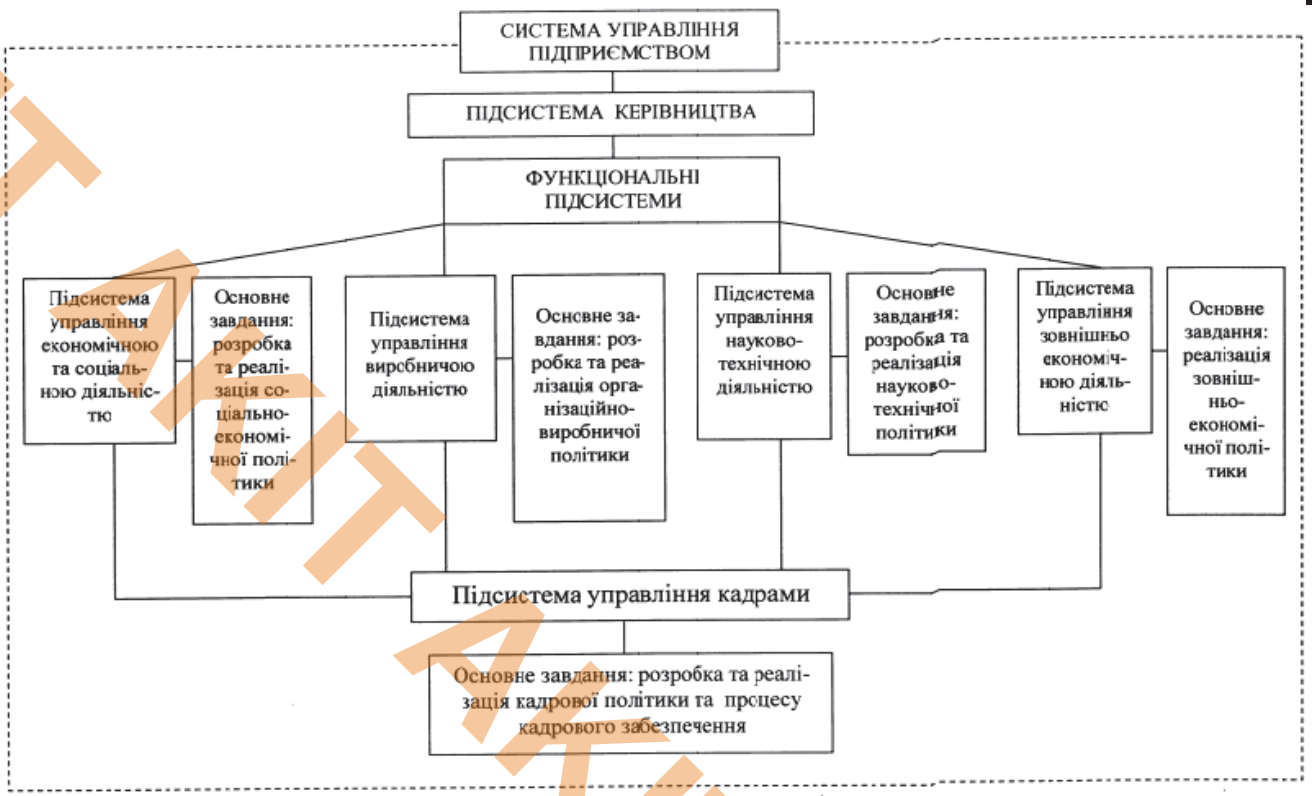

Рисунок 1.2 – Місце системи підтримки кадрів у система управління підприємством

Основними напрямами діяльності підсистеми управління кадрами є: стратегічне управління кадрами та кадровою політикою; управління плануванням кадрів; управління найманням, підбором та залученням кадрів; управління трудовими відносинами; управління умовами праці; управління розвитком кадрів; управління мотивацією кадрів; управління соціальним розвитком; управління розвитком організаційних структур управління; управління правовим, інформаційним забезпеченням системи управління кадрами.

Кадрове забезпечення, інтегруючись у кадрову політику, яка охоплює низку її найважливіших складових – залучення, відбір та наймання кадрів;

підбір та розміщення кадрів; професійну орієнтацію та адаптацію кадрів; мотивацію та стимулювання кадрів; професійне навчання, атестацію кадрів та просування по службі; управління діловою кар'єрою кадрів, а отже спричиняє активний вплив на систему управління підприємством та на підприємство в цілому, бо надає підприємству основну умову його сучасного функціонування – конкурентоздатний кадровий потенціал з високим рівнем професіоналізму та компетентності, особистісних якостей, працівників з інноваційним та мотиваційним механізмом. Система управління кадрами дозволяє підприємству чітко реагувати на події у зовнішньому середовищі, обирати такі варіанти поведінки, які б узгоджували його економічні процеси з вимогами господарського механізму країни. Від злагодженої роботи системи управління кадрами з іншими функціональними підсистемами, грамотної реалізації кадрової політики та процесу кадрового забезпечення залежить успішна діяльність підприємства. На жаль, на багатьох підприємствах України побудова якісної системи управління кадрами за багатьма причинами залишається досить складною проблемою, що в кінцевому варіанті призводить до банкрутства цих підприємств, тому питання побудови ефективної системи управління кадрами залишається відкритим й потребує подальшої розробки й удосконалення.

2 ОГЛЯД КОМПЮТЕРНО ІНТЕГРОВАНИХ СИСТЕМ УПРАВЛІННЯ КАДРАМИ В ОРГАНІЗАЦІЯХ

Комп'ютерно-інтегровані системи стали невід'ємною частиною сучасного життя. Вони назавжди змінили наш світ бізнесу, культури, освіти та виробництва. На сьогодні в таких системах виникає необхідність обробки все більшої кількості інформації та ставиться вимога полегшувати процес пошуку даних. Тому основною метою розроблення та впровадження даних систем є підвищення якості процесу ведення кадрів. Цієї мети можна досягти завдяки постійному моніторингу параметрів якості, забезпеченню достовірності й швидкості отримання інформації щодо різних аспектів організації процесу пошуку з необхідними деталями.

В наш час існує велика кількість автоматизованих систем підтримки ведення кадрів, кожна з яких має свої переваги та недоліки. Тому задля оцінення та покращення рішення необхідно ознайомитися з тими рішеннями, які є доступними для користувачів в даний момент часу. Розглянемо деякі з них та проведемо порівняння.

2.1 Аналіз програмної оболонки PersonPro 2.0

На українському ринку є понад десяток програмних продуктів, призначених для автоматизації управління персоналом. Одне з найбільш вдалих рішень – програма PersonPro 2.0. Що виділяє її з ряду подібних?

Програма PersonPro 2.0, основна сторінка якої наведена на рис.2.1, може успішно застосовуватися як керівником невеликої фірми, який нерідко веде облік кадрів за сумісництвом, так і для автоматизації кадрових служб великих підприємств. Її можливості можна також ефективно використовувати в кадрових агентствах для обліку кандидатів на відповідні посади та добору їх відповідно до вимог клієнтів. Потужна структура довідників даної програми дозволяє застосувати її і для обліку та контролю членства у громадських

організаціях, різних клубах, фондах та партіях. Таким чином, універсальність програми PersonPro 2.0 робить її привабливою як для управлінців, так і для фахівців, які працюють з кадрами.

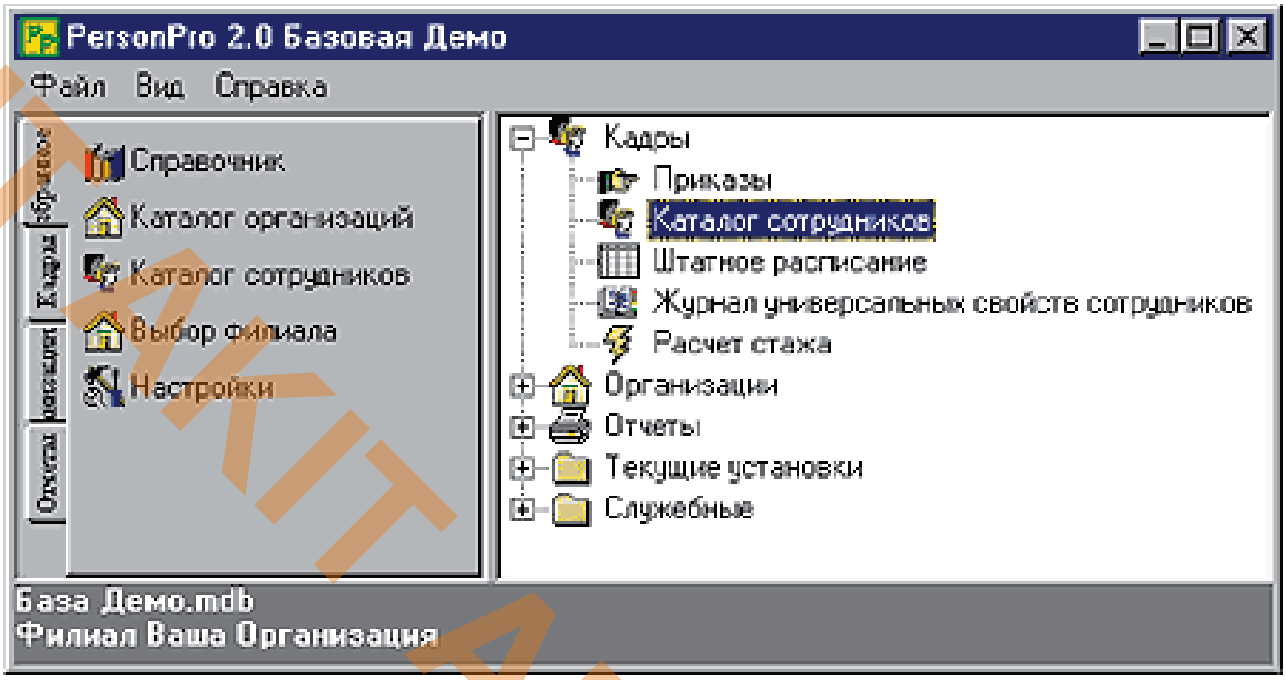

Рисунок 2.1. – Основна сторінка програмиPersonPro 2.0

Використовуючи програмну оболонку PersonPro 2.0, керівник зможе швидко отримувати дані про персонал, штатний розпис, оперативну інформацію з особових справ співробітників.Кадровики у складі даної програми отримають інструмент, який значно скоротить обсяг рутинних операцій, допоможе оперативно готувати рекомендації щодо проведення кадрової політики на підприємстві.

Усі необхідні дії з управління персоналом функціонально закріплені за двома основними підсистемами: "Кадри" та "Організації" (для прикладу див. рис.2.2).

Чітке та візуально зрозуміле структурування програми дозволяє навіть не підготовленому менеджеру вести кадровий облік. У розділі «Кадри» все дбайливо розкладено на основних напрямках:

- журнал наказів;

- каталог працівників;

- штатний розклад;

- форма розрахунку стажу.

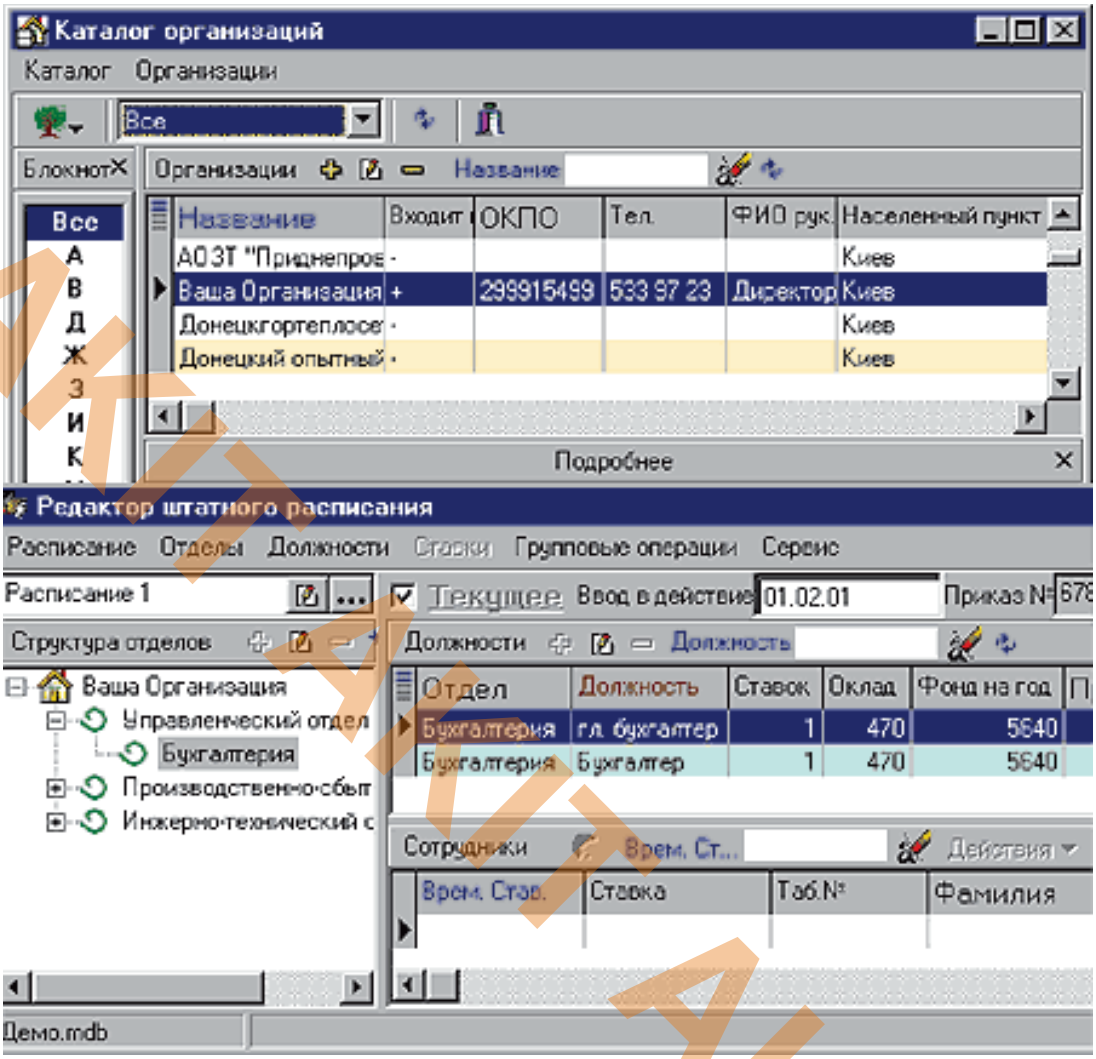

Рисунок 2.2 – Вікна різних розділів програмиPersonPro 2.0

Журнал наказів має ієрархічну структуру з докладною специфікацією документів. Підготовка будь-якого з них (персонального або групового) полягає в заповненні потрібних полів спеціальних форм, які автоматично викликаються. Наказ можна оформити в обліковій системі та роздрукувати всі його формалізації.

Облік штатного складу організації автоматизовано ведеться в каталозі співробітників. На кожного з них можна завести особисту картку з фотографією

та всіма його персональними даними: стаж, попередні місця роботи, адреси, телефони, сімейний стан, освіта, кадровий резерв та багато інших.

Підрозділ «Розрахунок стажу» дозволяє автоматизувати дані обчислення за кількома типами стажу: загальний, держслужба, у галузі, на підприємстві, пільговий. При перерахунку стажу може бути автоматично змінено відсоток оплати лікарняних листків.

Загалом, каталог організацій мало чим відрізняється за формою від відомих довідників інших облікових систем. Але за структурою та можливостями має цілу низку переваг. З його допомогою наявні в базі даних організації можна класифікувати за типом та сферами діяльності, об'єднувати їх у ієрархічні структури. Облікова картка підприємства містить повний набір необхідних даних, заповнення яких максимально автоматизовано за допомогою підпорядкованих довідників. Крім загальних даних, можуть бути введені додаткові відомості. А спеціальна вкладка «Універсальні властивості» дозволяє групувати реквізити підприємств за ознаками, значно спрощуючи керування та аналіз.

При практичній роботі з кадрами на підприємстві періодично виникають дві основі проблеми: звіти до податкової інспекції та звіти до різних державних органів. Тому особливе місце у структурі програми PersonPro 2.0займає підсистема "Звіти".При їх оформленні необхідно підготувати й заповнити велику кількість стандартних форм та здати їх до контролюючих органів. Програма PersonPro 2.0 значно спрощує вирішення цих проблем. У формі "Звіти" (рис.2.3) можна створювати та роздруковувати вбудовані документи (звіти) на папері або у файли, а також використовувати засоби конструювання звітів. З програмою "PersonPro 2.0 Базова" поставляються кілька десятків типових звітів, розбитих на п'ять категорій: "Лікарняні", "Одиничні форми", "Відпустка", "За штатним розкладом", "Професійні якості та стаж".

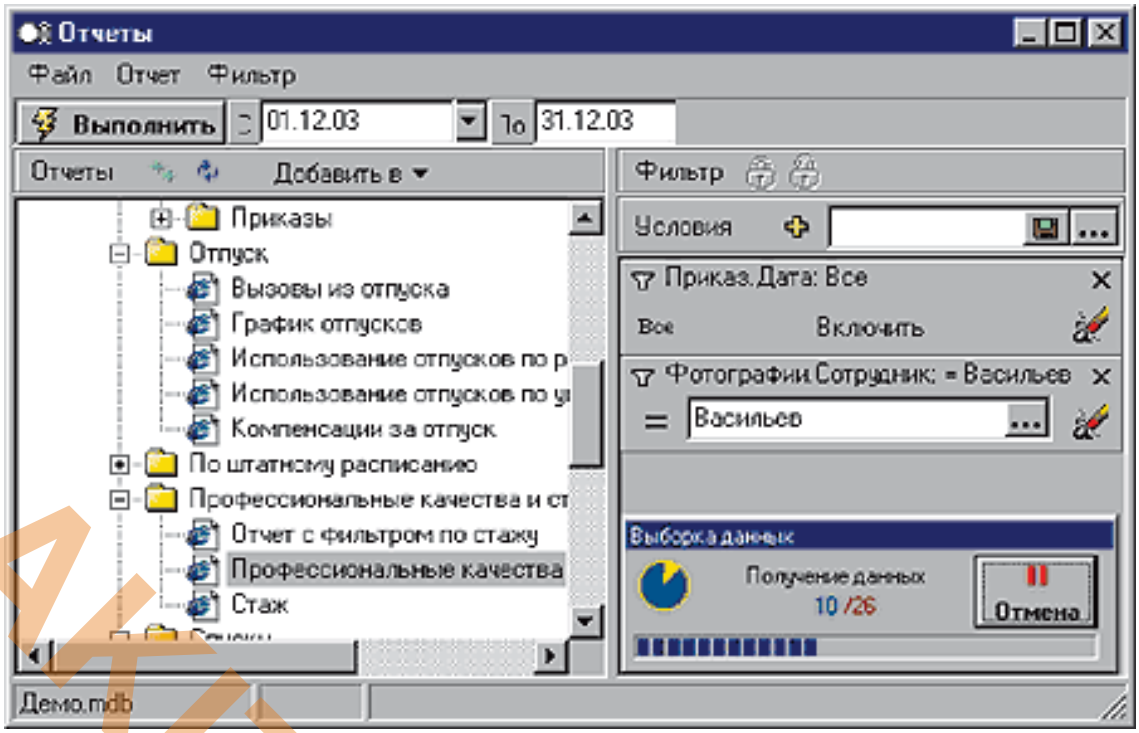

Рисунок 2.3 – Форма «Звіти» програмиPersonPro 2.0

У програмі також є можливість створювати та зберігати власні звіти на основі вже існуючих прототипів із збереженням умов фільтрації.

Доречною в програмі PersonPro 2.0 є і широка можливість експорту інформації в Excel. Будь-який довідник чи звіт можна відкрити у вигляді електронної таблиці з можливістю коригування форми, значень чи проведення глибшого управлінського аналізу.

Таким чином, PersonPro 2.0 – універсальна комп'ютерна програма для управління штатами будь-якого підприємства.

2.2 Аналіз програми «1С підприємство – Зарплата та управління персоналом»

Програма "Зарплата та управління персоналом" єбільш потужним інструментом для реалізації кадрової політики підприємства, а також автоматизації різних його служб, починаючи від служби управління

персоналом та лінійних керівників до працівників бухгалтерії за такими напрямками:

- планування потреб у персоналі;
- забезпечення бізнесу кадрами;
- управління компетенціями та атестація працівників;
- управління навчанням персоналу;
- управління фінансовою мотивацією персоналу;
- ефективне планування зайнятості персоналу;
- облік кадрів та аналіз кадрового складу;
- трудові відносини, зокрема кадрове діловодство;
- розрахунок заробітної плати персоналу;
- управління грошовими розрахунками з персоналом, включаючи депонування;
- обчислення регламентованих законодавством податків та внесків із фонду оплати праці;
- відображення нарахованої зарплати та податків у витратах підприємства.

У програмі особлива увага приділяється автоматизації управлінської діяльності менеджерів із персоналу. Зокрема, у базову конфігурацію включені такі інструменти, які дозволяють вирішувати основні завдання, і, з якими стикаються при плануванні як менеджери з персоналу, так і керівники різних рівнів:

- інструмент набору кадрів, який дозволяє здійснити всю процедуру підбору кадрів, включаючи листування з кандидатами електронною поштою;
- інструмент із планування відпусток працівників;
- інструментарій управління компетенціями та проведення атестацій;
- інструментарій керування навчанням працівників;
- інструмент розробки схем мотивації працівників.

За допомогою прикладного рішення у рамках програми можна проводити кадрову управлінську та облікову діяльність кількох організацій, причому в

ролі окремих організацій можуть виступати як юридичні особи, так і індивідуальні підприємці.

Прикладні рішення також можуть використовуватись на підприємствах холдингової структури, які складаються з кількох організацій, котрі оформлені як юридичні особи або є індивідуальними підприємцями. Тут програма забезпечує паралельне ведення двох видів обліку персоналу – управлінського та регламентованого. Управлінський облік ведеться по підприємству загалом, а регламентований облік ведеться окремо для кожної організації (юридичної особи чи індивідуального підприємця).

Планування потреб у людських ресурсах здійснюється у кадровому плані програми. Вихідні дані планування можуть надходити з виробничих систем. Кадровий план (рис.2.4) дозволяє запланувати організаційно-посадовий склад, подвійний план змін організаційно-посадового складу, фонд витрат на оплату праці. За допомогою звітів щодо стану кадрового плану можна отримати оцінку ефективності робіт із набору співробітників, кількість вакантних робочих місць, та інші важливі показники.

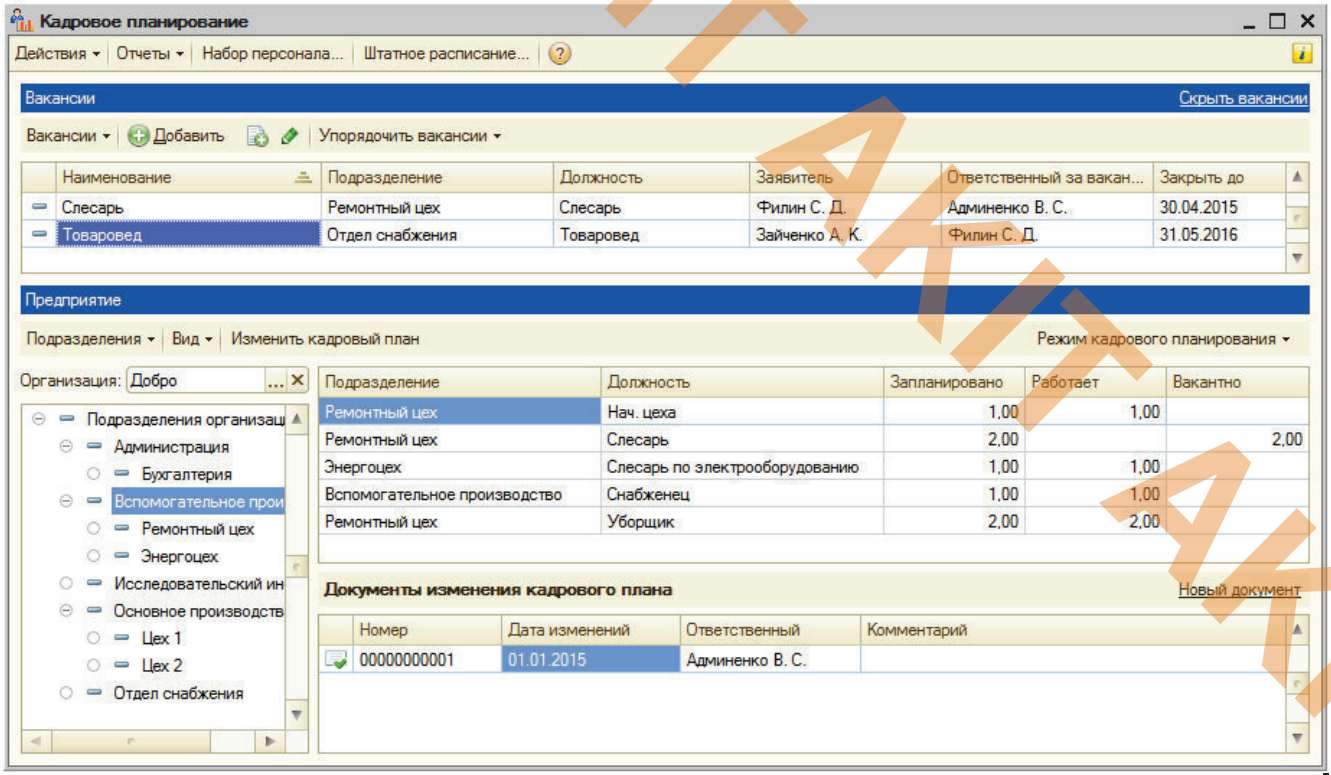

Рисунок 2.4 – Вікно кадрового планування програми 1С

Підсистема підбору персоналу призначена для документування та автоматизації процесу підбору та оцінки кандидатів. Підсистема забезпечує зберігання особистих даних про кандидатів як про фізичних осіб; зберігання матеріалів, що з'являються в процесі роботи з кандидатом, починаючи від резюме та до результатів анкетування; підготовку зустрічей з кандидатами та реєстрацію прийнятих рішень аж до прийняття на роботу. Основний інструмент системи – автоматизоване робоче місце менеджера з персоналу, з якого можна здійснювати всі етапи робіт із заповнення вакантних робочих місць.

Підсистема навчання персоналу дозволяє планувати навчання персоналу на внутрішніх та зовнішніх курсах навчання. Зберігати структуру курсів, список компетенцій, які можна підвищити завдяки курсу. Фіксація факту проходження курсів та механізм проведення атестацій дозволяє контролювати ефективність навчальних заходів.

Для управління компетенціями програма забезпечує оцінку персоналу, включаючи контроль результатів та якості оцінки. З оцінки працівника (атестації) приймаються ключові кадрові рішення: прийом працювати, ротації, зміна оплати праці, звільнення.

Програма "Зарплата та управління персоналом дозволяє" розробляти та застосовувати схеми фінансової мотивації працівників з використанням різних показників ефективності діяльності як окремого працівника, так і підприємства загалом. При створенні схем мотивації можна використовувати довільне кількість видів нарахувань, у програмі є можливість конструювати свій алгоритм розрахунку нарахування.

Прикладна частина програми також дозволяє вирішити одне з основних завдань планування використання робочого часу – планувати проведення заходів та участь працівників у внутрішніх та зовнішніх заходах підприємства, а також призначати наради та зустрічі працівників підприємства. Для підготовки проведення внутрішніх заходів підприємства програм надає

відомості про використання приміщень підприємства, у яких проводяться ці заходи.

Планування чергових відпусток працівників підприємства здійснюється на підставі плану важливих заходів підприємства на рік та розкладу участі в них працівників підприємства, при цьому враховуються побажання працівників та їх керівників. Формується графік відпусток працівників підприємства та список запланованих відпусток. Для затвердження графіка відпусток у програмі використовується система затвердження ухвалених рішень.

Зарплата та управління персоналом передбачає зберігання як особистих даних працівників підприємства, так і службової інформації. До останньої відносяться: підрозділ, де числиться працівник, його посада, службові телефони та інша контактна інформація. Реєструється і просування працівника по посадах на підприємстві: прийом на роботу, службові переміщення, відпустки і відрядження, звільнення.

Для зручності роботи кадровика основні функції кадрового обліку зібрані на окремий робочий стіл, приклад якого наведено на рис.2.5.

Облік стажу роботи у організації ведеться автоматично в межах видів стажу. Усі процедури, які виконують будь-які розрахунки залежно від стажу, автоматично визначають тривалість стажу від дати прийому до організації. Крім того, реалізовано розрахунок пільгових видів стажу та облік стажу до прийому на роботу в організацію. Для аналізу кадрового складу підприємства з накопиченої інформації про працівників будуються різноманітні звіти. У тому числі списки працівників організації, рух кадрів організації, статистика кадрів організації та інших.

Прикладне рішення підтримує ведення військового обліку. У програмі формуються всі необхідні відомості для подання до військкоматів.

Програма «Зарплата та управління персоналом» підтримує також ведення штатного розкладу організацій із можливістю зазначення різних видів тарифних ставок, довільної кількості надбавок, додаткової інформації про штатні одиниці. За штатним розкладом формується вся необхідна звітність.

У програмі автоматизовано кадрове діловодство, включаючи заповнення уніфікованих друкованих форм: оформлення трудових договорів, прийом на роботу (форма П-1), кадрові переміщення працівників, табелів обліку використання робочого часу (форма П-5), звільнення з організації (форма П- 4). З кадрових даних будується уніфікована форма П-2 " Особиста картка " .

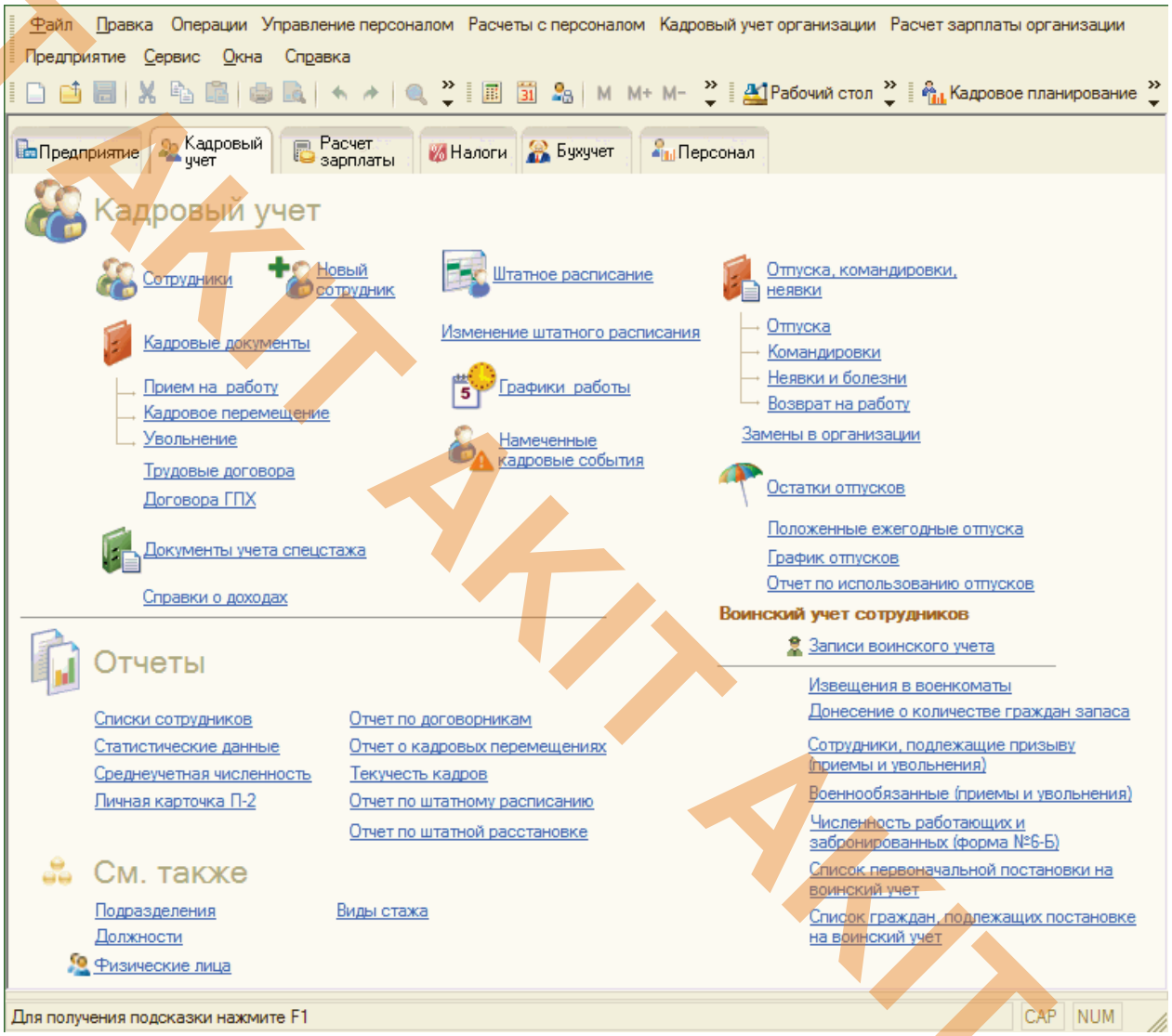

Рисунок 2.5 – Головне вікнопрограми 1С

Відповідно до затвердженого графіка відпусток формуються накази про надання відпустки працівникам (форма П-3). Ведеться облік використання відпусток. Є можливість відображати відпустки різних видів, у тому числі у зв'язку з вагітністю та пологами, доглядом за дитиною, додаткову щорічну

відпустку, додаткову відпустку на дітей, чорнобильську відпустку, навчальну відпустку, відпустку власним коштом та інші. Передбачено звіти щодо аналізу використання відпусток.

Ведеться облік заохочень та стягнень. На підставі документа може бути запроваджено нарахування премії чи іншого разового нарахування чи утримання.

Заплановані відрядження працівників також реєструються у програмі. При цьому готуються накази про направлення працівників у відрядження та заповнюються посвідчення про відрядження (уніфікована форма).

Для розрахунку та обліку заробітної плати у програмі автоматизовано діяльність як менеджерів, які приймають рішення щодо заробітної плати персоналу, так і бухгалтерів, які розраховують заробітну плату. При цьому враховані такі види діяльності:

- розробка схем мотивації працівників;

- розрахунок заробітної плати, яка залежить від виробітку;

- автоматичний розрахунок широкого кола нарахувань від оплати за ставкою до оплати лікарняних та відпусток із середнього заробітку;

- гнучке настроювання використовуваних нарахувань та утримань, у тому числі, можливість розрахунку сум нарахувань від сум, отримуваних працівником на руки.

Для розрахунку сум нарахувань та утримань в управлінському та регламентованому обліку є можливість використовувати довільні формули, в яких крім описанихта визначених користувачами показників (кількість оплачених днів/годин, тарифна ставка та ін.), допускається застосовувати арифметичні дії, певні математичні функції та вводити свої умовні математичні вирази.

Програма «Зарплата та управління персоналом» забезпечує ведення взаєморозрахунків із працівниками підприємства, і навіть облік витрат за оплату праці у складі собівартості продукції і послуг. Автоматизовано весь комплекс розрахунків з персоналом, починаючи від введення документів про

фактичний обсяг виробленої продукції, оплату лікарняних листів та відпусток, і закінчуючиформуванням документів на виплату зарплати, міжрозрахункових виплат та депонування, а також звітності до державних наглядових органів.

У програмі реалізовано основні форми оплати праці, які використовуються на госпрозрахункових підприємствах: погодинна (з використанням місячних, денних та погодинних тарифних ставок) та відрядна форми оплати праці, а також їх варіанти (погодинно-преміальна, відрядно-преміальна, опосередковано-відрядна та комісійна форми оплати праці). Підтримується у програмі й співробітництво з працівниками за умов укладання договору цивільно-правового характеру.

Гнучкий механізм обліку використання робочого часу дозволяє описувати графіки роботи (індивідуальні графіки роботи, включаючи "ковзні", а також зведені індивідуальні графіки роботи) та реєструвати відхилення від звичайного режиму роботи. Для економії часу при заповненні графіка роботи доступне автоматичне заповнення за визначеними шаблонами, у тому числі, з урахуванням вечірнього та нічного годинника. При автоматичному заповненні можуть опціонально враховуватись святкові дні. Реалізовано можливість введення докладних та/або зведених табелів обліку робочого часу як первинних документів, дані яких використовуються далі при нарахуванні зарплати.

З метою забезпечення роботи на підприємствах з великою кількістю працівників основні розрахункові документи також забезпечені засобами автоматичного заповнення та розрахунку. Наприклад, автоматичне заповнення табличних частин документів щодо перерахунку внесків, табличних частин розрахунку зарплати. Автоматизовано процес виправлення помилок розрахункових документів минулого періоду. У разі введення виправлення, система автоматично зупиняє неправильні записи минулого періоду.Передбачено і автоматичне заповнення довідки про доходи працівника у різних формах для надання до різних організацій, наприклад для розрахунку пенсії, отримання субсидії чи кредиту.

Крім можливості проводити численні розрахунки заробітної плати, програма «Зарплата та управління персоналом» надає можливість формувати основні уніфіковані форми з обліку праці та заробітної плати, а також інші звіти для отримання інформації за будь-який розрахунковий період:

- розрахункові листки;
- розрахунково-платіжні відомості (форми П-6, П-7, довільна динамічна форма);
- платіжні відомості для одержання грошей через касу;
- видаткові касові ордери;
- пені за нарахуваннями та утриманнями тощо.

Результати розрахунків можуть бути представлені у вигляді аналітичних звітів, наочних графіків та діаграм. Наприклад, аналіз неявок дозволяє проаналізувати в одній обробці всі, зареєстровані кадровиком, відхилення (відпустки, лікарняні та інше) й відразу ж запровадити відповідні нарахування за середнім заробітком.

Передбачено звіт для аналізу нарахувань та утримань працівників організацій за довільний період як у вигляді докладної розрахункової відомості, з можливістю виведення не лише сум нарахувань, а й оплаченого часу, так і у вигляді згорнутої форми без аналітики щодо працівників.

Таким чином, проведений аналіз показує широкі можливості існуючих програмних продуктів щодо управління кадрами на підприємствах.

Питання побудови взаємодії учасників при оптимізації кадрової системи полягає у тому, щоб визначити можливості, якими буде володіти така система, а також ті проблеми, які вона вирішуватиме [8]. При побудові такої системи потрібно звернути особливу увагу на наступні складнощі:

- система повинна надавати змогу легко вносити інформаційні зміни;
- система має бути доступною для легкої інтеграції та оновлення.

З огляду на це розглянемо етапи оптимізації нами кадрової системи. Для цього спроектуємо концептуальну модель бази даних, сформуємо архітектуру майбутнього програмного продукту та розглянемо деякі базові його компоненти.

3.1 Моделювання роботи оптимізованої системи

Взаємодію користувача із системою промоделюємо схемою, наведеною на рис. 3.1.

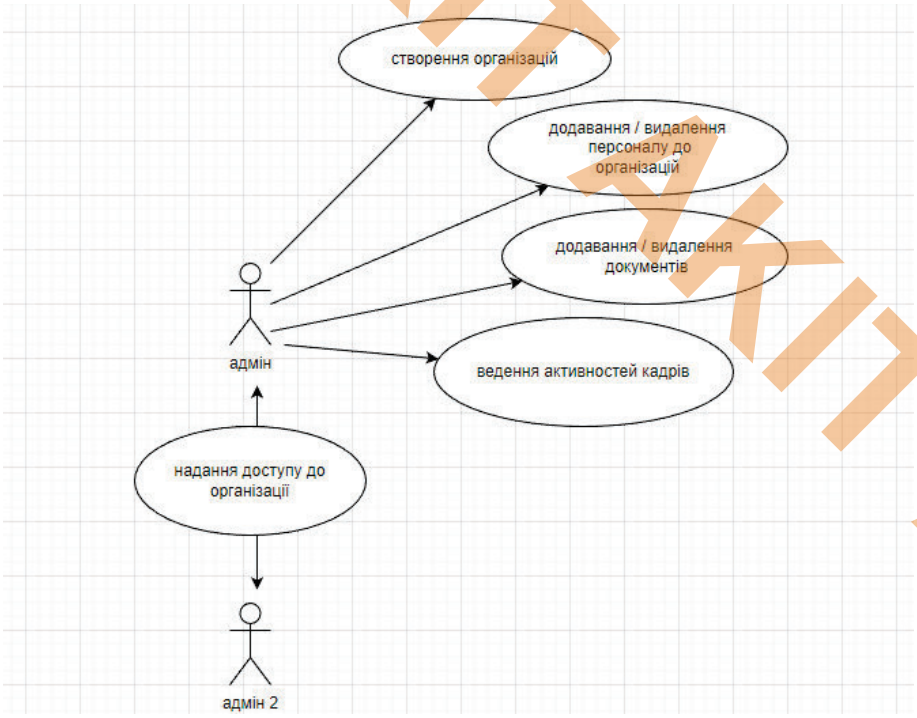

Рисунок 3.1 – Діаграма прецедентів кадрової системи

Аналіз наведеної діаграми показує, що кожен користувач системи матиме свою особисту роль у майбутньому програмному продукті та відповідний власний функціонал, який буде доступним для тієї ролі.

Для прикладу розглянемо детальніше адміністратора, який створює систему кадрового забезпечення і є її менеджером. Діаграму прецедентів, характерних для адміністратора, зображено на рисунку 3.2.

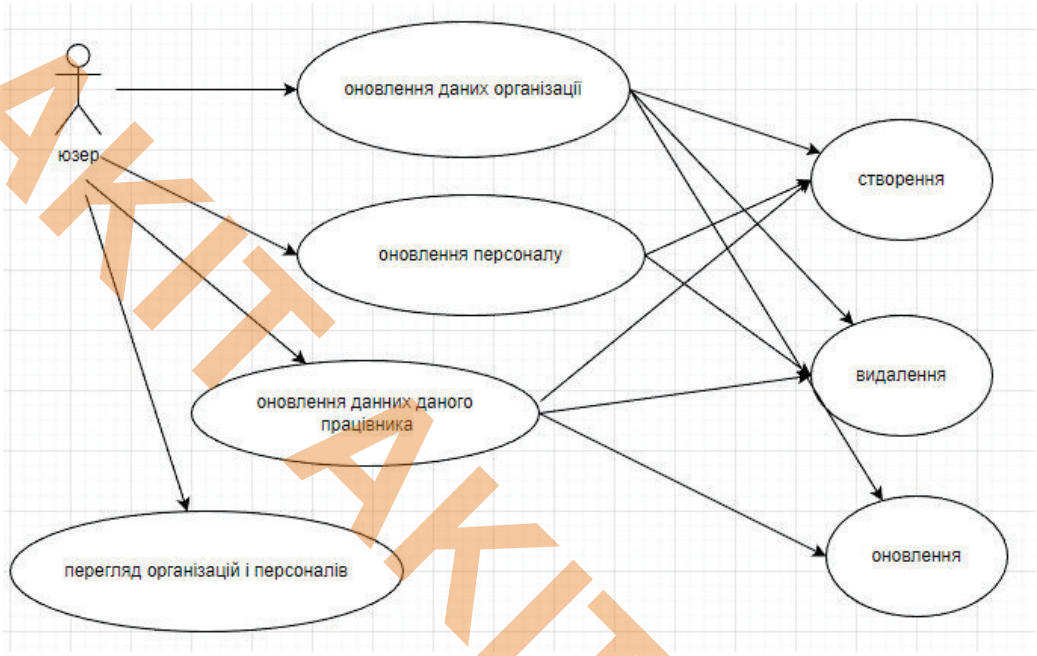

Рисунок 3.2 – Діаграма прецедентів адміністратора кадрової системи

Система взаємодії між учасником системи для ведення кадрів, має вимоги, також зображені на концептуальній діаграмі прецедентів рисунку 3.2. Зрозуміло, що ведучим автором тут виступає адміністратор, який управляє всіма процесами в системі.

Кожен з прецедентів діаграми відповідає за оновлення інформації певних таблиць бази даних. Кожна роль тут дуже важлива, оскільки від внесених даних будуть залежати результати виконання інших ролей, які для своєї роботи повинні використовувати ці дані.

Також для більшого розуміння, як виконуватиметься робота з програмою, на рисунку 3.3 зображено діаграму діяльності, яка показує, як саме користувач взаємодіятиме з програмою та що від нього вимагається.

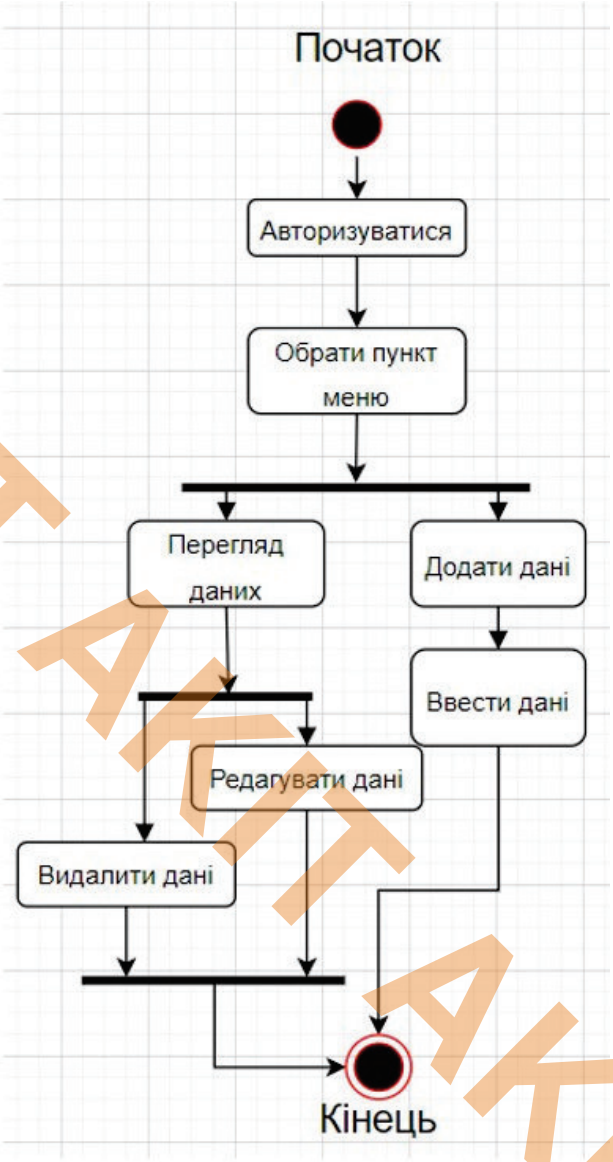

Рисунок 3.3 – Діаграма діяльності в системі кадрового забезпечення

3.2 Архітектура оптимізованої кадрової системи

Система являється сервісом, який надає змогу формувати бази даних у загальному проекті, які в подальшому можуть використовуватися для створення та формування різних документів і проведення різних операцій.

Орієнтуючись на сформовані вище вимоги до системи, отримаємо наступний вигляд архітектури кадрової системи, який наведений на рисунку 3.4.

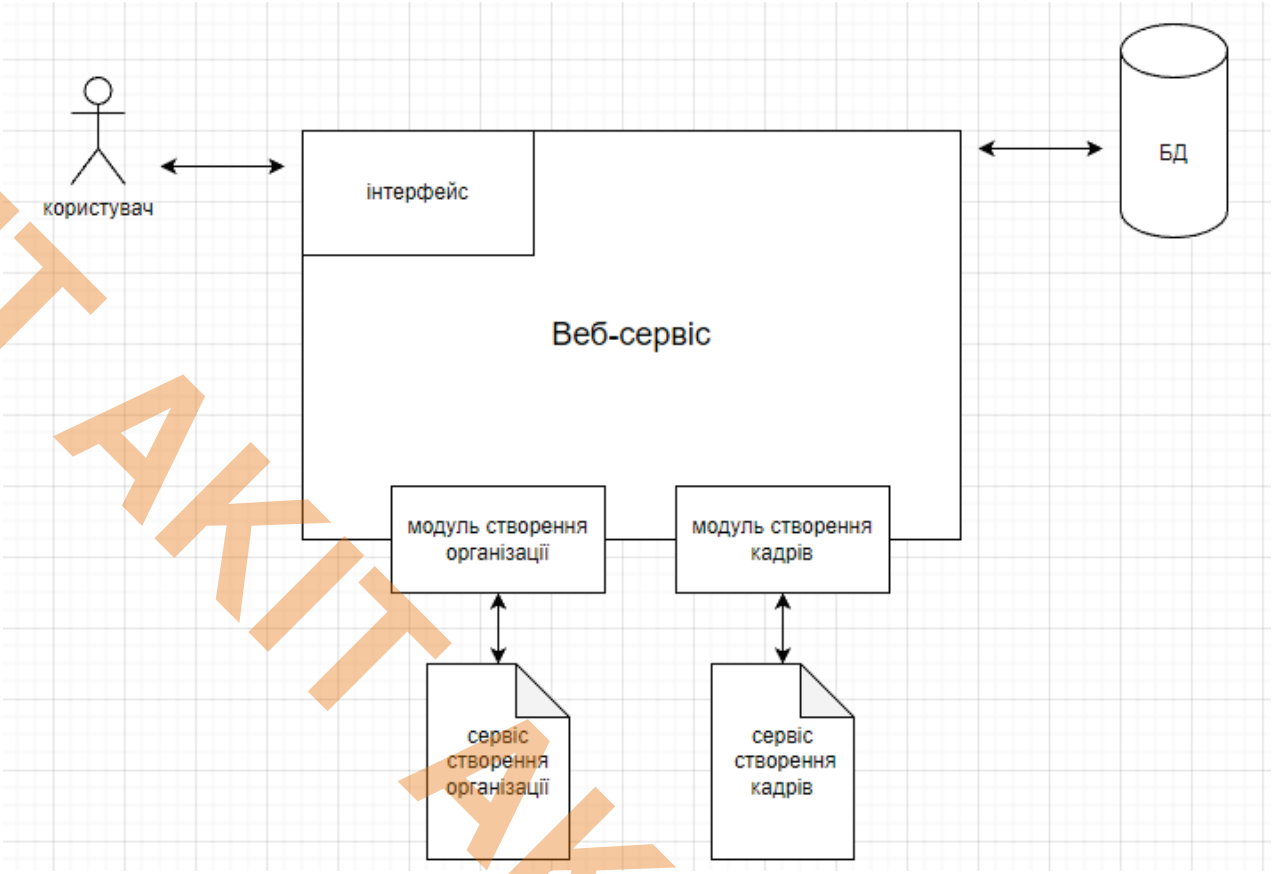

Рисунок 3.4 – Загальна архітектура кадрової системи

Відмітимо, що запропонована система являється веб-сервісом, який користувачі мають змогу використовувати з будь-яких зручних для них пристроїв, якщо вони мають можливість доступу до Інтернету. Її особливістю є відсутність потреби у встановленні та налаштуванні кадрової системи.

Спроектований веб-сервіс також містить у собі окремий підмодуль, який відповідає за оновлення інформації вкадровому відділі, а також забезпечує виконання різних операцій із записами в таблицях бази даних.

Для прозорості розуміння архітектури кадрової системи та того, як вона працюватиме в реальному часі, наведемо розширену діаграму розгортання системи, зображену на рисунку 3.5.

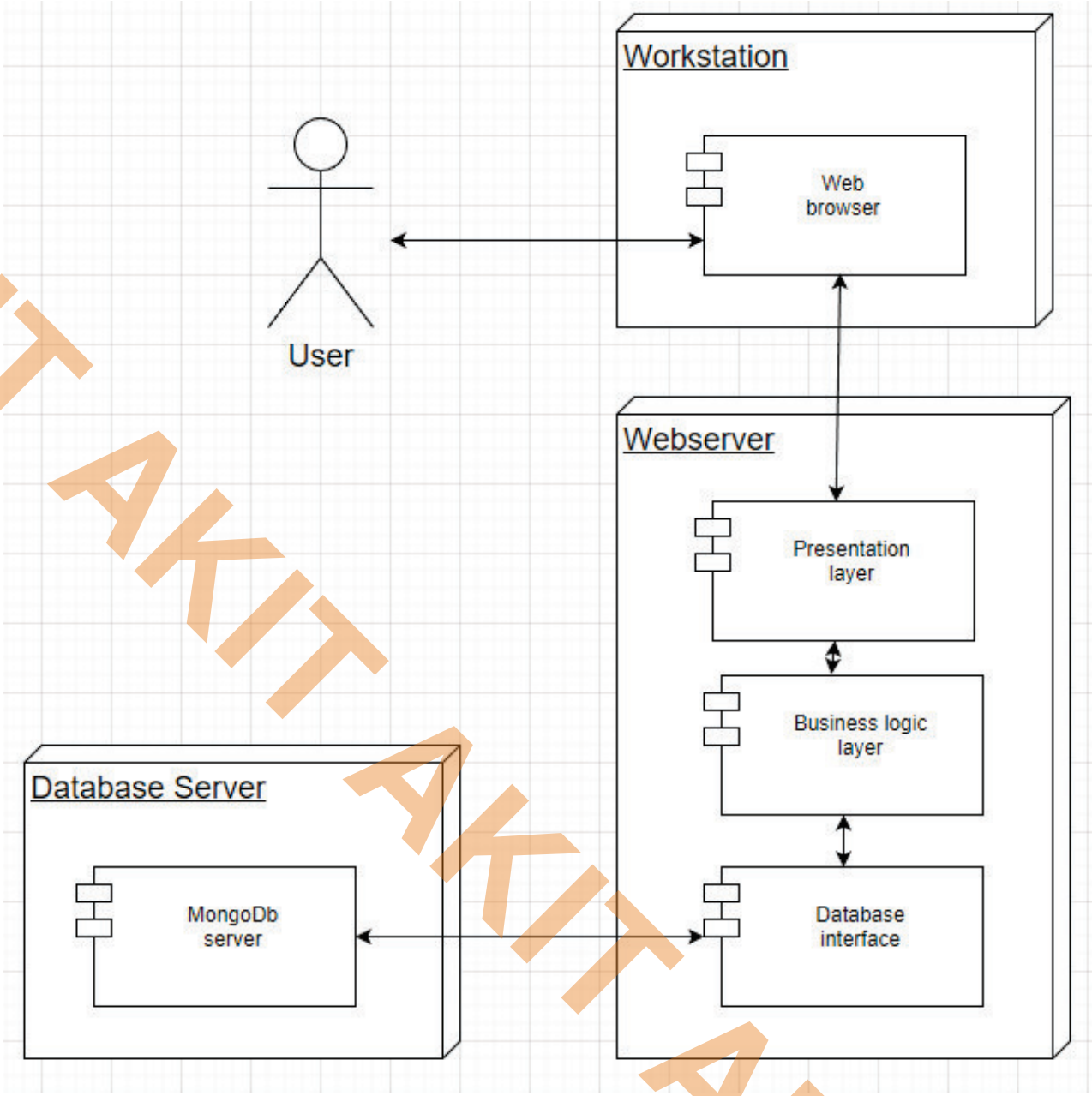

Рисунок 3.5 – Діаграма розгортання оптимізованої кадрової системи

3.3Розробка концептуальної моделі бази даних кадрової системи

Першим етапом процесу проектування бази даних для комп'ютерних інформаційних кадрових систем є концептуальне проектування бази даних. Його основна мета – це створення концептуальної моделі всіх даних предметної області. На цьому етапі дуже важливо зробититаку модель бази даних, щоб при наступній її практичній реалізації не повертатися назад та не виправляти помилки. Тому для нормальної організації та керуванням наявною інформацією

введемо в програмну оболонку своє власне сховище у вигляді оптимізованої бази даних. На рисунку 3.6 зображена модель такої бази даних, розробленої для повноцінної кадрової системи, яка має працювати із найрізноманітнішими прикладними модулями.

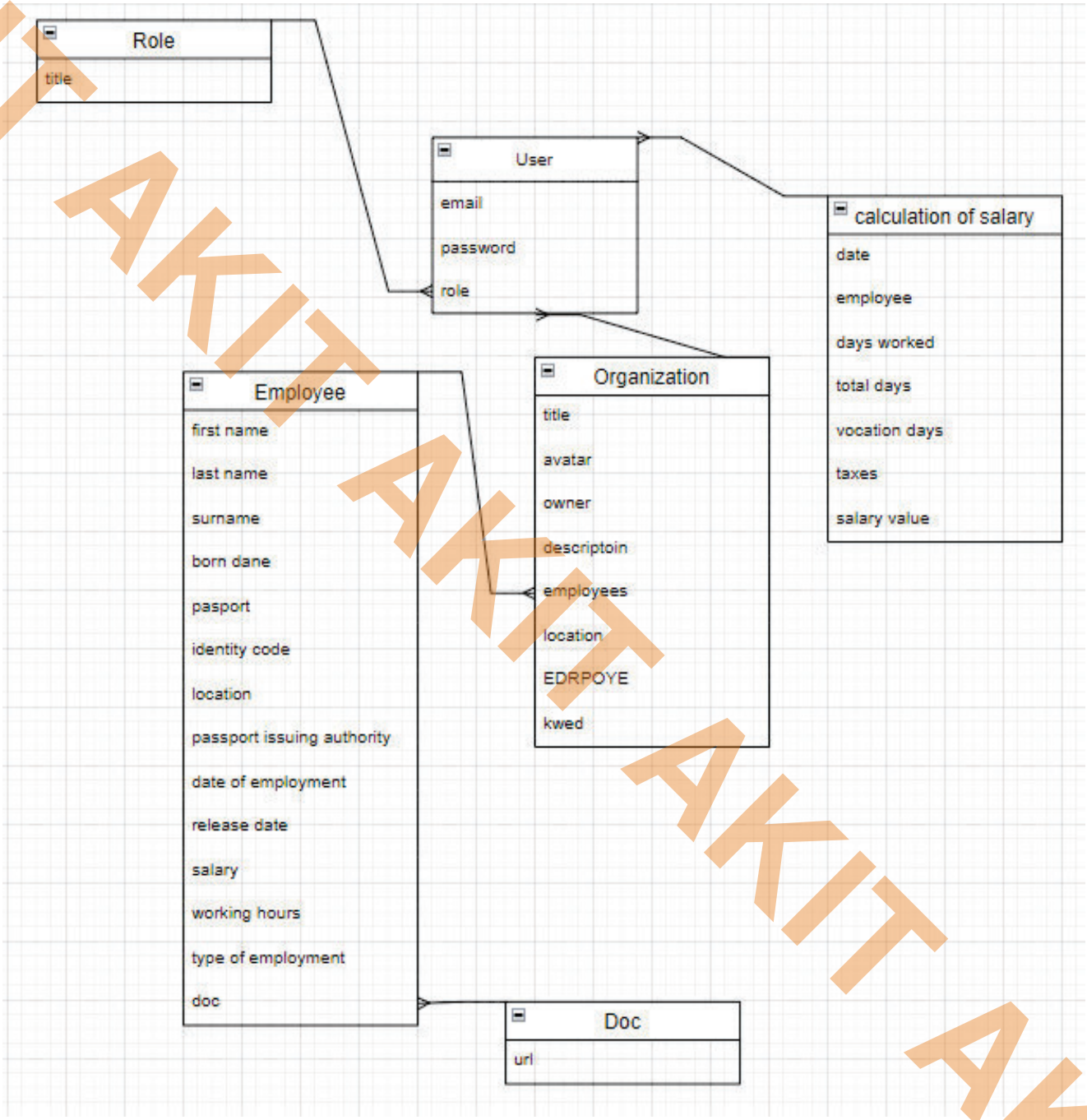

Рисунок 3.6 – Концептуальна модель оптимізованої бази даних

Розглянемо основні таблиці, які використовуються у модулях спроектованої бази даних.

Таблиця «Користувач», приклад якої наведено в таблиці 3.1, відповідає за мінімальні данні користувача, які йому потрібні для входу в кадрову систему. Вона складається з п’ятьох полів, для кожного з яких є унікальний ідентифікатор id типу ObjectId. Така побудова забезпечує швидкий пошук даних у таблиці, При цьому поляEmail, Password та Role зберігатимуть текстові данні про електронну адресу, пароль та роль користувача в кадровій системі.

Таблиця 3.1 – Приклад таблиці «Користувач» оптимізованої бази даних

№	Назва поля	Тип даних поля	Опис поля
1	Id	ObjectId	Первинний ключ таблиці та ідентифікатор
2	Email	String	Пошта
3	Password	String	Пароль
4	Role	String	Роль користувача

Таблиця «Організація» формується за зразком, наведеним в таблиці 3.2. Вона містить інформацію про організацію, яка може включати любі відомості конкретної організації, у тому числі й різні зображення. Вона має такі поля:

- унікальний ідентифікаторid типу ObjectId;
- Title;
-Owner;
- Employees;
-Descriptions;
-Location;
-Kwed;
- EDRPOYE;
-Avatar.

Зміст інформації, яка міститиметься в цих полях, зрозумілий із наведеного прикладу таблиці.

Таблиця 3.2 – Приклад таблиці «Організація» бази даних

№	Назва поля	Тип даних поля	Опис поля
1	Id	ObjectId	Первинний ключ таблиці та ідентифікатор
2	Title	String	Назва організації
3	Owner	String	Тип організації
4	Employees	ObjectId	Тип працівників
5	Descriptions	String	Опис організації
6	Location	String	Аватар організації
7	Avatar	String	Посилання на офіційний сайт організації
8	Kwed	Number	Квед організації
9	EDRPOYE	Number	Код фірми

Зміст таблиці «Ролі» наведено в таблиці 3.3, яка містять назви ролей Ці назви є унікальними для кожного користувача. Вони присвоюються при наданні доступу до системи кожному новому користувачу. Дана таблиця має поля, якими є унікальний ідентифікатор id типу ObjectId та Value. У другому полі зберігаються дані типу текст і число.

У свою чергу, блок даних «Документи» наведено в таблиці 3.4. Вона містять посилання на всі збереженіна сервері документи на сервері і має поля у вигляді унікального ідентифікаторaid типу ObjectId та назви документа Url, у якому зберігатимуться дані типу текст.

Таблиця 3.3 – Приклад формування таблиці «Ролі» бази даних

№	Назва поля	Тип поля	Опис
1	Id	ObjectId	Первинний ключ таблиці та ідентифікатор
2	Value	String	Назва ролі

Таблиця 3.4 – Приклад формування в базі даних таблиці «Документи»

№	Назва поля	Тип поля	Опис
1	Id	ObjectId	Первинний ключ таблиці та ідентифікатор
2	Url	String	Посилання на документ

Таблиця бази даних «Працівник» наведена в таблиці 3.5. Вона містять посилання на всі збережені на сервері документи працівника. Дана таблиця є основною й найбільш об’ємною для кадрової системи. Вона має такі поля: унікальний індефікаторid типу ObjectId, Firstname,Lastname,Surname,Borndate, Passport, Identity, PassportissuingAuthority, Dateofemployment,Doc, Releasedate, Typeofemploymentта Salary. Всі ці поля зберігатимуть дані типу текст, крім відміченого окремо поля id.

Таблиця 3.5 – Приклад таблиці бази даних «Працівник»

№	Назва поля	Тип поля	Опис
1	Id	ObjectId	Первинний ключ таблиці та ідентифікатор
2	Firstname	String	Ім’я
3	Lastname	String	Прізвище
4	Surname	String	По батькові
5	Borndate	Date	Дата народження

Продовження таблиці 3.5

№	Назва поля	Тип поля	Опис
6	Passport	String	Паспортні дані
7	Identitycode	String	Ідентифікаційний код
8	Location	Number	Місце знаходження
9	Passportissuingauthority	Number	Орган і дата видачі паспорту
10	Dateofemployment	Date	Дата прийому
11	Releasedate	Date	Дата звільнення
12	Salary	Number	зарплатня
13	Workinghours	Number	Робочі години
14	Typeofemployment	String	Тип зайнятості
15	Doc	ObjectId	документи

Ще одна важлива таблиця оптимізованої бази даних «Видача заробітної платні» наведена як таблиця3.6. Її змістом також є посилання на збереженіна сервері документи. Вона має такий набір полів: унікальний Id для швидкого пошуку в таблиці, Date,Employee,Daysworked,Totaldays, Vocationdays, Taxes, та Salaryvalue. Тут зберігатимуть данні типу текст і числа, аналогічно попереднім таблицям бази даних.

Крім вказаних вище, у склад оптимізованої бази даних введені й інші допоміжні таблиці, детально розкривати зміст яких не є доцільним у даній роботі. Такі таблиці містять інформацію, яка допомагає проводити оптимізацію кадрової системи у цілому.

Таблиця3.6 – Приклад таблиці «Видача зарплатні» для оптимізованої бази даних

№	Назва поля	Тип поля	Опис
1	Id	ObjectId	Первинний ключ таблиці та ідентифікатор
2	Date	Date	Дата зарплати
3	Employee	ObjectId	Працівник
4	Daysworked	Number	Днів відпрацьовано
5	Totaldays	Number	Загальна кількість робочих днів
6	Vocationdays	Number	Дні відпустки
7	Taxes	Number	Податки
8	Salaryvalue	Number	Зарплатня

4ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЦЕСУ ОПТИМІЗАЦІЇ СИСТЕМИ КАДРОВОГО ЗАБЕСПЕЧЕННЯ

Даний додаток реалізує оптимізацію системи кадрового забезпечення на підприємстві у вигляді програмного коду. Основна його задача: обробляти інформацію про людину, яка приходить на роботу і про організацію, яка його наймає. Система має модульну структуру та використовує сервер сформованої раніше бази даних, що забезпечує механізми збереження цілісності даних та легку адаптацію змін до них. Принципи створення програмного коду було розроблено на основі рекомендацій ряду робіт [9 – 14].

4.1. Вибір програмних засобів для розробки

Спроектована для оптимізації система буде являтися веб-сервісом, який складається з чисто серверної та клієнтської частин. Вибір програмних засобів розробки цих двох частин є важливим етапом перед початком оптимізації системи. Головним критерієм тут є зручність подальшого додавання нових та оновлювати вже існуючих компонентів системи.

Для розробки серверних частин існує чимало різних мов програмування, найбільш популярними з яких на сьогодні є PHP, C#, Java, JavaScript, Python. Проаналізувавши переваги кожної мови та взявши до уваги спроектовані раніше концептуальну модель бази даних і архітектуру системи, ми для оптимізації серверної частини обрали мову JavaScript, оскільки вона легше взаємодіє з кодом існуючих програм, написаних на інших мовах. Крім того, JavaScriptоптимально підходить як для простих, так і для об’ємних програмнихпроектів, пропонуючи при цьому розробнику всі необхідні інструменти найвищої якості та зручність їх використання.

JavaScript на сьогоднішній день являється однією з найпотужніших та затребуваних мов програмування в IT-сфері, яка швидко розвивається. Вона є сучасною досить популярною об’єктно орієнтованою мовою, але підтримує

також компонентноорієнтоване програмування. Особливо відома мова стала завдяки створенню корпоративного програмного забезпечення, фінансових проектів, мобільних додатків, а також різноманітнихігрових і навчальних сервісів.

Для розробки серверної частини JavaScript містить фреймворки Koa, а також Express та Meteor.js на платформі Node.js. Оскільки однією з вимог до майбутнього оптимізованого додатку була крос-платформенність, то ми будемо широко використовувати Express. Однією з переваг цього фреймворку є мінімалістичний синтаксис, який дуже зручний для подальшого розвитку нашого проекту.

Для розробки клієнтської частини в JavaScriptіснує також багато різних інструментів, які допоможуть зробити інтерфейс системи приємним та зручним для користувача. Тут варто виділити JS-фреймворки і різні бібліотеки, такі як Angular, React та Vue.js. Це найбільш популярні серед існуючих бібліотек, всі можуть ефективно співпрацювати з нашою розробкою. Для того щоб не втратити продуктивністьрозроблюваного продукту ми будемо використовувати бібліотеку React.

Таким чином, у результаті проведеного аналізу програмних засобів для оптимізації кадрової системи нами було обрано мову програмування JavaScript. При цьому для проектування серверної частини використаємо фреймворк; для розробки веб-додатків – Express.js; для керуванням базою даних та здійснення операцій з нею використовуватиметься Mongoose; нарешті для клієнтської частини було обрано бібліотеку React.

4.2 Особливості мови програмування JavaScript

Мова JavaScriptза задумом створювалася для того, щоб «оживити» веб-сторінки. Програми на цій мові називаються скриптами. Цією мовою можна писати програми прямо на сторінку в коді HTM – вони автоматично починають працювати при завантаженні сторінок. Скрипти формуються та

відлагоджуютьсяяк простий текст. Для запуску їм не потрібна спеціальна підготовка чи компілятор.У цьому плані JavaScript дуже відрізняється від іншої мови програмування – Java.

Коли була створена мова JavaScript, спочатку вона мала іншу назву: “LiveScript”. Але тоді була дуже популярна мова програмування Java, тому було вирішено, що позиціонування нової мови як “молодшої сестри” Java сприяло б її популяризації.Але з часом JavaScript значно виросла й стала повністю незалежною мовою програмування зі своєю специфікацією ECMAScript. На сьогодні немає нічого спільного з цими двома мовами.

Важливо, що JavaScript може підтримуватися не тільки в браузері, але й на сервері або на будь-якому іншому пристрої, який має спеціальну програму – рушій JavaScript.Вбудований рушійсам по собі описує “віртуальну машину JavaScript”.Різні рушії мають різні “кодові назви”. наприклад:V8 – в Chrome, Opera та Edge; SpiderMonkey – у Firefox; “Chakra” для IE; “JavaScriptCore”, “Nitro” і “SquirrelFish” для Safari та інші.

Рушії як програмний продукт є досить складними. Але принцип їхньої роботи простий. Вбудований рушій, наприклад,«читає» скрипт. Потім він перетворює («компілює») скрипт у тимчасову репрезентацію («байт-код»).Потім байт-код виконується відповідними пристроями, причому дуже швидко.

Рушій також застосовує оптимізацію процесів на кожному етапі функціонування програми. Він також слідкує за скомпільованим скриптом під час його виконання, аналізує дані, які проходять через скрипт, і оптимізує байт-код на основі цих знань.

Є як мінімум три чудові особливості в JavaScript:

- цілковита інтеграція з HTML/CSS;
- прості речі створюються просто;
- підтримуватися всіма сучасними браузерами.

JavaScript – це єдина браузерна технологія, яка суміщає ці три речі. Ось чому JavaScript – найбільш розширений засіб створення браузерних

інтерфейсів.До слова, JavaScript також дозволяє створювати сервери, мобільні пристрої тощо.

Можливість JavaScript у браузері обмежена лише питаннями безпеки користувача. Це зв'язано з забороною доступу небезпечним веб-сторінкам до приватної чи корпоративної інформації, щоб не пошкодити дані, які знаходяться на комп'ютері користувачів. Тому JavaScript на веб-сторінці не може читати/записувати довільні файли на жорсткому диску, копіювати їх або виконувати програми. Вона не має прямого доступу до функцій операційної системи.

У програмної оболонки є можливість взаємодії з камерою/мікрофоном або іншими зовнішніми пристроями, але для цього також потрібен явний дозвіл користувача. Тому, сторінка, на якій активований JavaScript, не може сама по собі вимкнути веб-камеру, без санкцій спостерігати за оточенням і посилати інформацію іншому суб'єкту.

Позитивною рисою програми є й те, шо її різні вкладки/вікна традиційно не знають одне про одного – тому JavaScript з однієї свої сторінки не має доступу до іншої, якщо вони розміщені на різних сайтах, мають різні домени, протоколи чи порти.Щоб обійти це обмеження, обидві сторінки повинні бути відправлені на обмін даними черезReplyJavaScript код, який спеціальним способом буде забезпечувати обмін даними.

JavaScript може легко спілкуватися через мережу лише з сервером, з якого отримана поточна сторінка. Але її здатність отримувати дані з інших сайтів/доменів обмежена. Запит на доступ до іншого домена також можливий, проте для цього потрібний спеціальний запит заголовка в HTTP від віддаленого сервера. Всі ці особливості програмного середовища JavaScriptпризначені для підсилення його безпеки.

4.3 Використання Node.Js і Express.js

Node.js – це серверна платформа, обгорнута мовою JavaScript для створення масштабованих програм, керованих подіями. Така особливість є нетиповою навіть для досвідчених програмістів, оскільки традиційне середовище JavaScript завжди було на орієнтоване на конкретного клієнта. Тому JavaScript не працює напряму при спілкуванні клієнта з сервером. Процес спілкування сервера з клієнтом організовує Node.js.

Node.js написаний не на JavaScript, а на мові C ++. Але він використовує мову JavaScript як інтерпретаційну мову для обробки запитів і відповідей відносно серверів. Іншими словами, Node.js запускає автономні програми JavaScript. Перевага такого підходу полягає в тому, що програмісти можуть використовувати свої початкові знання програмування і починати кодування з Node.js, що набагато легше, ніж пряме застосування JavaScript.

Node.js має декілька атрибутів, які роблять його особливо привабливим для мережевого або Інтернет-програмування. Перша пов'язана з оптимальним «пакуванням» інформації при передачі її між користувачами під час переміщення даних через Інтернет. Node.js значно скорочує цю процедуру і забезпечує легкі засоби для її проведення.

Другий привабливий атрибут Node.js пов'язаний з моделлю подій веб-програмування. Більшість існуючих технологій створені для обробки великих об’ємів даних при кожному запиті та відповіді. Іншими словами, на сервер може бути надіслана ціла сторінка даних, навіть якщо є лише невеликі зміни. Ці технології оптимізовані для використання великих фрагментів даних із меншою кількістю подій. Node.js робить навпаки; він розроблений для роботи з більшою інтерактивністю – меншими об’ємами даних, які реагують на багато інших різних подій.

4.4 Використання системи MongoDB

MongoDB– система управління базами даних, яка працює з документоорієнтованою моделлю даних. На відміну від реляційних баз даних, MongoDB не потрібні таблиці, схеми або окрема мова запитів. Інформація зберігається в даній системі у вигляді документів або колекцій, що підвищує продуктивність усієї системи.

Особливістю даної системи є також те, що вона розроблена на мові програмування C++, тому з легкістю інтегрується під будь-яку операційну систему (Windows, Linux, MacOS та ін.).

MongoDB використовує свій власний формат даних для зберігання інформації – це двійкова нотація об'єктів (BSON), побудована на основі мови JavaScript.

Реляційні бази даних використовують тип даних «рядок» для зберігання документів, тоді якв MongoDBзастосовується посилання на весь документ за допомогою відповідних ключів. Також замість таблицьMongoDB використовує колекції, які містять різні типи наборів даних.

Оптимізована і система зберігання інформації в цій системі. Для цього введено поняття вузлів. Для відповідної бази існує один головний і безліч вторинних вузлів і вся оброблювальна інформація реплікується між даними точками. Якщо один первинний вузол виходить із дії, то якийсь вторинний вузол стає головним. Крім того, в системі MongoDB можна використовувати індексацію – технологію, яка дозволяє будь-яке поле в документі на перегляд користувача. Така проіндексована інформація обробляється значно швидше.

Для збереження даних великого розміру MongoDB використовує власну технологію GridFS у вигляді двох колекцій. У першій колекції зберігаються імена великих файлів і їхні метадані. Друга колекція містить фрагменти – сегменти інформації, розмір яких не перевищує 256 Кб.

Спеціально створена система балансування в MongoDBвикористовується для розподілу навантаження між різними базами даних, а також для горизонтального масштабування. При цьому сегменти бази даних

розподіляються по різним вузлах, що підвищує загальну продуктивність роботи системи. Самі бази даних, які розташовані на різних вузлах, синхронізовані між собою і забезпечують клієнту повноту доступу до всієї інформації.

Таким чином, системи MongoDB відноситься до класу NoSQLбаз даних, яка працює з цілими документами, а не із записами. Даний програмний продукт легко виводиться в будь-яку операційну систему. Ряд унікальних особливостей дозволяє використовувати таку систему бази даних під складні завдання, у яких вона забезпечує максимальну продуктивність і надійність.

4.5 Використання спеціального відображувача Mongoose

Mongoose - це ObjectDocumentMapper(об'єктно-документний відображувач). Він дозволяє визначати об'єкти зі строго типізованою схемою, яка відповідає документу MongoDB.

Mongoose надає величезний набір функціональних можливостей для створення та роботи зі схемами. У цілому Mongoose має вісім типів схем даних,кожна з яких орієнтована на зберігання вMongoDB:

- рядок;
- номер;
- дата;
- буфер;
- логічний;
- змішаний;
- ObjectId (унікальний ідентифікатор об'єкта або його первинний ключid);
- масив.

Для кожного типу даних можна виконати такі функції:

- зазначити значення для настройок;
- зазначити функцію користувача для перевірки даних;
- зазначити, що поле необхідно заповнити;

- позначте get-функцію (getter), яка дозволяє вам проводити маніпуляції над даними до їх повернення у вікно об'єкта;

- позначити set-функцію (setter), яка дозволяє вам проводити маніпуляції над даними перед їх збереженням в базі даних;

- програмування індексів для більш швидкого отримання даних.

Крім наведених загальних можливостей для деяких типів даних також можна налаштувати особливості збереження та отримання даних із бази даних. Наприклад, для типу даних String можна зазначити наступні додаткові опції:

- конвертація даних до нижнього реєстру;

- конвертація даних до верхнього реєстру;

- обрізання даних перед зберіганням;

- визначення регулярного виразу, який дозволяє у процесі перевірки даних обмежити дозволені для зберігання варіанти даних;

- визначення переліку, який дозволяє встановити список припущених рядків.

Деякі із введених типів даних є не традиційними. Зокрема тип даних «буфер» дозволяє зберегти двійкові дані у вигляді зображення або закодованого файлу PDF-форматі. Тип же даних «змішаний» використовує для обробки спеціальне поле, до якогодопускаються дані будь-якого типу. Тип даних ObjectIdзазвичай використовується для визначення посилання на інший документ у вашій базі даних. Тип даних «масив» дозволяє виконувати над даними операції, характерні для обробки масивів.

4.6 Бібліотека React.js

Reactспеціальна бібліотека мови JavaScript, яка використовується для розробки інтерфейсів користувача. Саме вона відповідає за те, як ваш веб-додаток буде виглядати. Однак, слід відмітити, що розробити повністю веб-додаток тільки на React неможливо. Дана бібліотека призначена для виконання

лише частини завдань. Зокрема, вона дає можливість модифікувати кожен компонент веб-додатку незалежно від інших.

Бібліотеку React можна додати до будь-якого сайту, створеного на мові Javascript для того, щоб можна було повністю створювати інтерфейс або невеликі його частини. При цьому розробникам потрібно буде тільки описати, як кожна частина інтерфейсу виглядає в різних станах, а бібліотека буде міняти їх при отриманні необхідних команд.

Бібліотека Reactдозволяє робити сайти не тільки красивими, але і продуктивними, оновлюючи тільки ті частини сторінок, які потрібно змінити. У пам'яті веб-додатки зберігається структура попередньої версії, що дає змогу порівняти її з новим станом інтерфейсу. Важливо також, що на практиці швидкість завантаження компонентів веб-додатків не буде залежати від потужності того пристрою, на якому користувач його відкриває.

Reactможна використовувати в веб-додатках будь-якого масштабу, але саме в невеликих односторінкових додатках із перспективним зростанням він показує себе максимально ефективно. До того ж, сайти з React високо оцінюються пошуковими ботами Google.

Код, написаний з React, просто тестувати, його можна використовувати повторно. Декларативні подання роблять його максимально передбачуваним, у порівнянні з іншими фреймворками, і зменшують кількість помилок під час налагоджування. При цьому логіка JavaScript не обмежена шаблонами, що дозволяє легко передавати дані між компонентами будь-якої частини програми.

4.7Середовище розробки VisualStudioCode

VisualStudioCode– це редактор вихідних кодів, створений корпорацією Microsoft для Windows, Linux та macOS. Він включає підтримку налагодження, підсвічування синтаксису, інтелектуальне заповнення коду, вивід фрагментів, рефакторинг коду та вбудований Git. При цьому Є можливість написання коду на таких мовах як: C++, C#, VisualBasic, F#, JavaScript, Python, TypeScript та

інші. Крім усього цього система включає в себе конструктори, редактори, відладчики, а також величезну кількість розширень для різних областей застосування – від PHP до ігор. На рисунку 4.1 наведено загальний вигляд інтерфейсуVisualStudioCode.

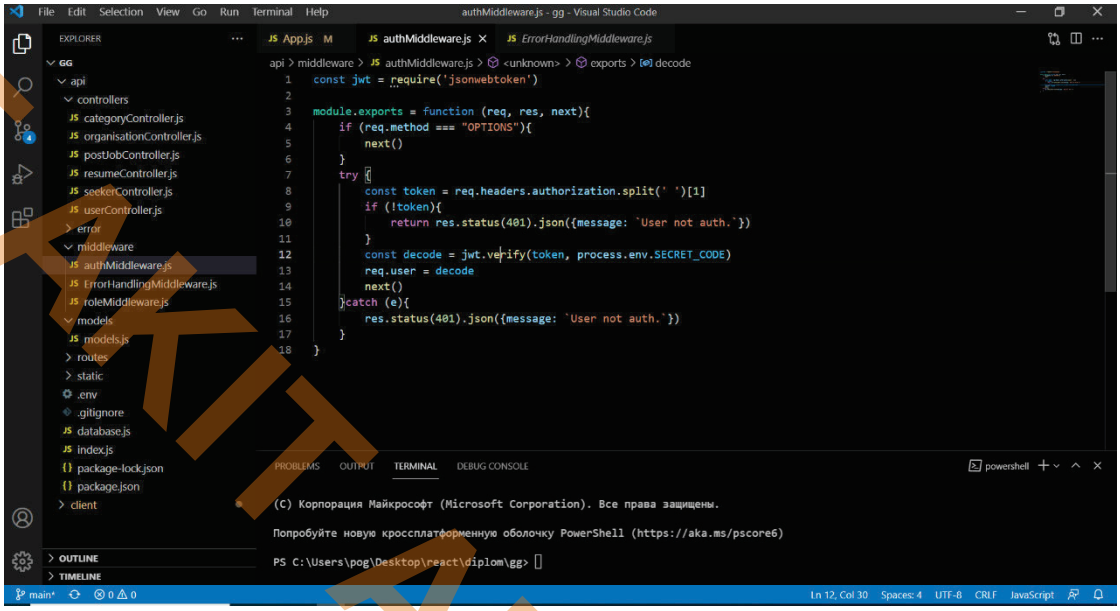

Рисунок 4.1 – Загальний інтерфейс редактора VisualStudioCode

Одним з головних переваг VisualStudioCode – це плагіни. Середовище розробки VisualStudioCode є безкоштовним, як і можливість створювати і користуватися плагінами.

4.8 МоваHyperTextMarkupLanguage

Для написання будь-якого шаблону широко використовується мова гіпертекстової розмітки HyperTextMarkupLanguage(HTML), яка існує з 1992 року. Документи цього формату представляють звичайні текстові файли за винятком того, що деякі символи інтерпретуються як розмітка. Опис сторінки за допомогою цієї мови є набором інструкцій. Дані інструкції пишуться за допомогою так званих тегів (tag), які веб-браузер інтерпретує та відображає користувачеві. Відповідно, після таких дій HTML-файл стає Web-документом.

Дана мова використовується практично всіма веб-ресурсами, адже вся клієнтська частина пишеться завдяки їй.

4.9 Застосування каскадних таблиць стилів

Як і HTML, CascadingStyleSheets (CSS) насправді не є мовою програмування, а каскадними таблицями стилів. Це дає можливість застосовувати стилі вибірково до необхідних елементів та блоків у HTML-файлах. Наприклад, щоб вибрати всі елементи меню на сторінці HTML і зробити текст у них жовтим кольором.

CSS дозволяє також створювати правила, які вказують, як і де елемент повинен з'явитися та як він буде виглядати. Наприклад, можна вказати, що колір фону сторінки буде темно-синім, всі параграфи повинні використовувати шрифт TimesNewRoman або всі рівні першого заголовки повинні бути червоні та виділені курсивом.

CSS направляєописи стилів, які повторюються в одних і тих самих елементах, у окремий файл. Завдяки цьому відбувається значне "розвантаження" документів HTML. Тому HTML-код стає легким, зручним для читання та редагування.

4.10 Основні принципи оптимізації системи кадрового забезпечення

Оптимізація програмного продукту являє собою процес дизайну, у результаті якого реалізується такий код, який можна легко розширити, підтримувати та в якому без проблем може розібратися кожний розробник.

Не дивлячись на існування багатьох різних мов програмування, принципи створення програмних продуктів є універсальні для всіх них. Розглянемо основні із них, яких ми дотримувалися при розв’язанні задач, поставлених у магістерській роботі.

4.10.1 Принцип Keepitsimple, stupid

Основна ідея принципу Keepitsimple, stupid(KISS)– чим простіше, тим краще. У такому випадку кожен розробник зможе легко розуміти суть створеного коду. Реалізація даного принципу зводиться до виконання двох положень. 1. Роби коротко і просто. 2. Роби просто і зрозуміло.

Якщо дотримуватися цих правил, то логіка створеного коду буде зрозумілою для всіх. Якщо ж логіку коду зрозуміла лише розробнику, то й сам код викликає підозри. Таким чином, принцип KISS націлений на простоту використання продукту кожним користувачем.

4.10.2 Принцип Don'trepeatyourself

У любому коді програми завжди є повтори певної його частини в кількох або багатьох місцях. Згідно зпринципом Don'trepeatyourself(DRY)необхідно уникати повторення будь-якого однакового коду. Якщо цей принцип не покласти в процес розробки, то при виникненні необхідності внески певні зміни в цей код, доведеться відшукувати його по тексту всієї програми. А при таких діях нераціонально тратиться багато часу і є велика ймовірність щось випадково «зламати» в логіці роботи програми. Знайти ж такі помилки теж не є легкою справою.

Методи дотримання принципу DRYє різними. 1. Повністю спільнучастину можна винести в reusable клас, компонент або метод. 2. У іншому місці реалізовувати лише частину вже існуючого коду. 3.Розділити код на дві або три частини та використати ці частини окремо. Все залежить від наявної конкретної ситуації.

4.10.3 ПринципиS.O.L.I.D.

Це найцікавіший та найкориснішийнабір принципів, які націлені на допомогу розробникам для створення чистого, добре структурованого коду, який легко підтримувати. Дані принципи включають:

- S — SingleResponsibilityPrinciple(SRP);
- O — Open-ClosedPrinciple (OCP);
- L — LiskovSubstitutionPrinciple (LSP);
- I — InterfaceSegregationPrinciple (ISP);
- D — DependencyInversionPrinciple (DIP).

Коротко розглянемо кожен із наведених принципів.

4.10.4 SRP принцип є принцип “відповідальності” розробника за зміни, які можуть вноситися в об’єкти різного типу програми. Найпростішим правилом цього принципу є те, що не слід включати до одного класу об’єктів різні дії. Наприклад, вам необхідно створити код для компіляції та друкування об’єкту. Тоді не варто робити код, який робить ці дві дії одночасно (рис.4.2).

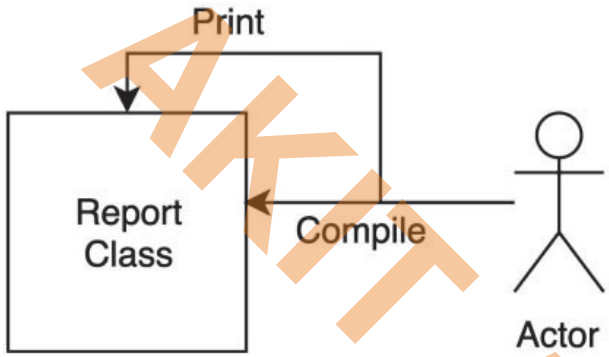

Рисунок 4.2 – Небажане поєднання дій в одній частині коду

З наведеної реалізації слідує, що і компіляція, і друк належать до класу Report. Але якщо в майбутньому потрібно буде створити на основі вибраних даних ще XML-документ, то виникне необхідність відкрити цей один ReportClass та змінити логіку функціональності Print. Але змінюючи один клас, ми теоретично можемо зламати те, що в ньому знаходиться з функціоналуCompile. Тому після змін ми маємо перевірити функціональність Compile і впевнитись, що вона працює. Отже, у нас була мета змінити Print, але ми одночасно вплинули на дві частини функціоналу.

З наведеного прикладу слідує більш ефективне рішення, яке потрібно було втілити в життя ще з самого початку роботи над кодом.Варто створбвати окремий клас, який компілює окремий модуль, та окремий клас, який друкує його. А оскільки ці два класи відносяться до Report, то їх можна об'єднати в модуль (рис.4.3). У такому випадку зміни функціональності Print не вплинуть на функціональність Compile і навпаки.

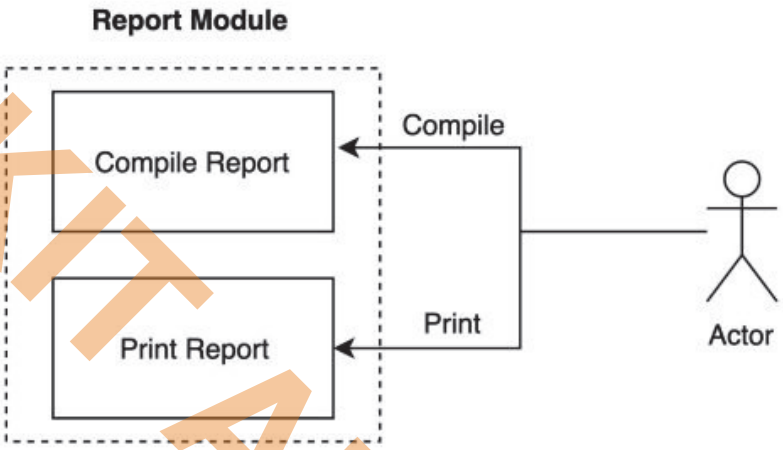

Рисунок 4.3 – Рекомендоване поєднання дій відповідно з SRP принципом

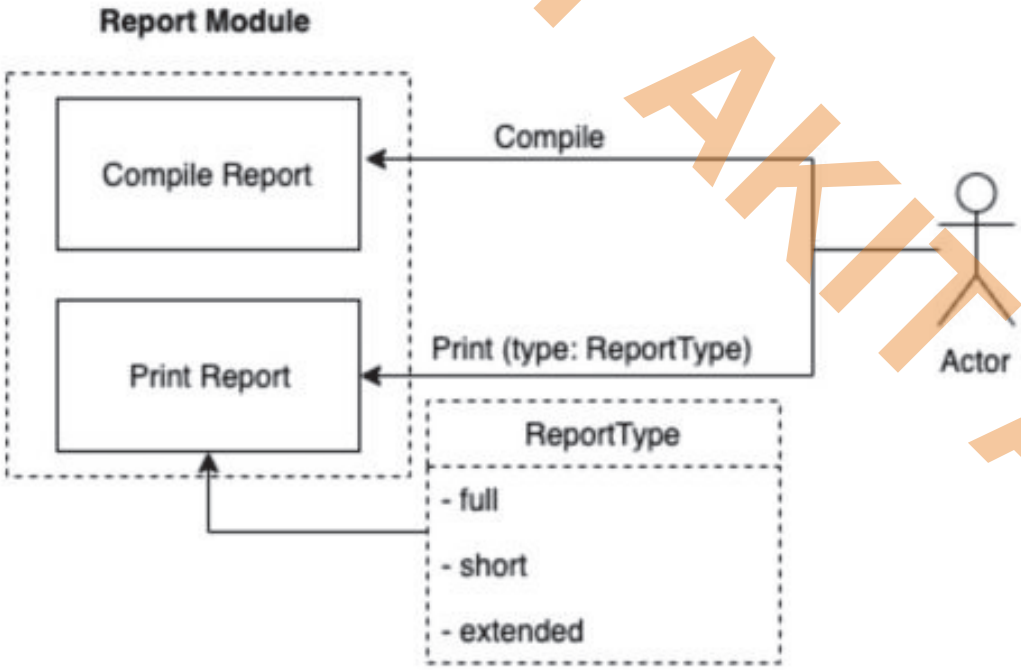

Рисунок 4.4 – Код без застосування принципу OCP

OCP принцип декларує, щокласи, модулі та функції мають бути відкритими для розширення та закритими для модифікації. Використання даного принципу ілюструють рис.4.4 та 4..5.

Відповідно до наведеного на рисунках прикладу, нам потрібно друкувати репорти різного типу – повний, короткий та розширений. Перше, що спадає на думку, організувати єдинийReportType, куди ввести всі ці типи документів. У коді це призведе до використання декількох операторів IF всередині блокуReport. А якщо нам треба буде додати нові типи репортів для друку, то це вимагатиме зміни всього коду (рис.4.4). Такий підхід порушуєOpen-Closed принцип.

Значно краще рішення – додати окремийReportModel, де будуть друкуватись лише певні поля. Тоді функціоналPrintбуде сконцентровано в одному модулі і принцип SingleResponsibility не порушується. Тепер ми можемо скільки завгодно розширяти вказаний модуль, але при цьому ми його не модифікуємо (рис.4.5).

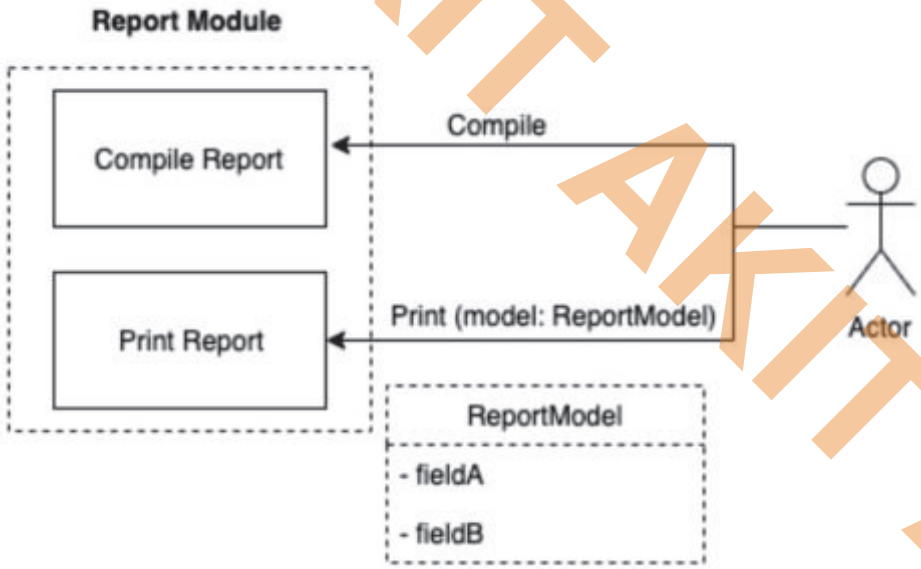

Рисунок 4.5 – Код із правильним застосуванням принципу OCP

Таким чином, принципи SingleResponsibility та Open-Closedстимулюють думати про майбутнє використання розробленої програми, якщо потрібно буде

щось змінювати в ній. Тому кваліфіковані розробники різноманітних програмних продуктів більше часу витрачають на обдумування рішень, ніж на їхню реалізацію.

4.10.5 Принцип LSPпродемонструємо наступним нашим прикладом (рис.4.6). Він показує, що треба уважно проаналізовуватия, який тип слід присвоїти тій чи іншій змінній. При цьому часто доцільно деяким змінним присвоїти більш загальний тип з метою безпроблемного їх передавання різним блоками процедурам та модулям програми. Таким же чином слід формувати й перелік елементів, які мають передаватися різним логічним частинам та модулям, як це демонструє рисунок 4.6. У цьому випадку видно конфлікт, який виникає при обміні даними та з типами параметрів процедури SubReportModeli модуля ReportModel.

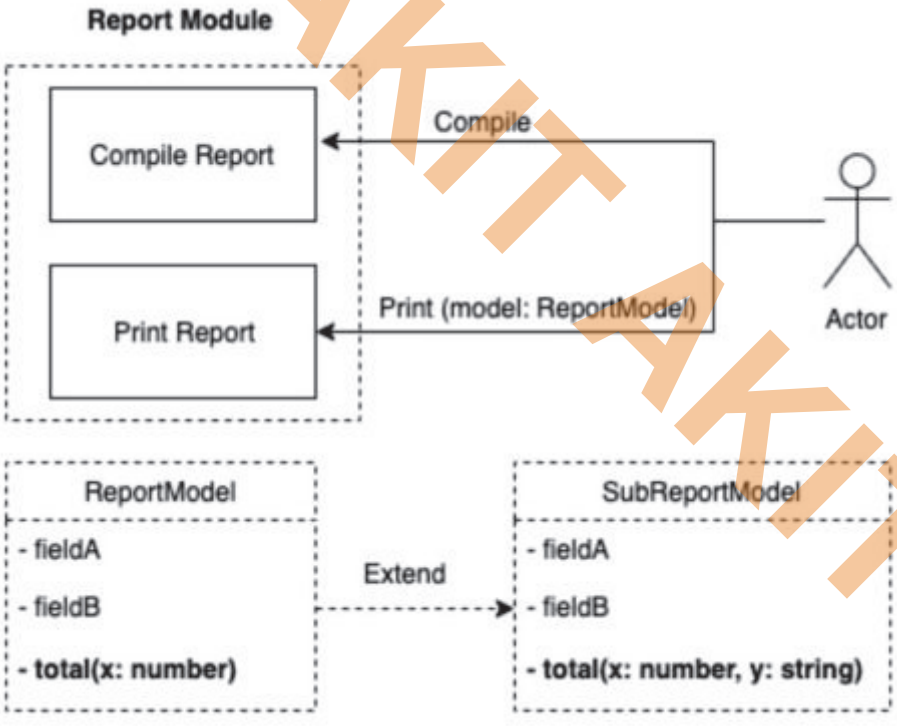

Рисунок 4.6 – Ілюстрація дії принципу LSP

4.10.6 ISP принцип декларує свободу клієнтів на вибір тих методів, які вони використовують при роботі з програмним забезпеченням. Тому для роботи з різними наборами даних бажано доступ до них організовувати шляхом виділення окремих інтерфейсів (рис.4.7. та 4.8). Тим самим вдається уникнути ряду проблем при майбутній експлуатації програми: несприйняття окремих інтерфейсів певним користувачем, помилки при роботі з інтерфейсом, застосування хибних методів та інше.

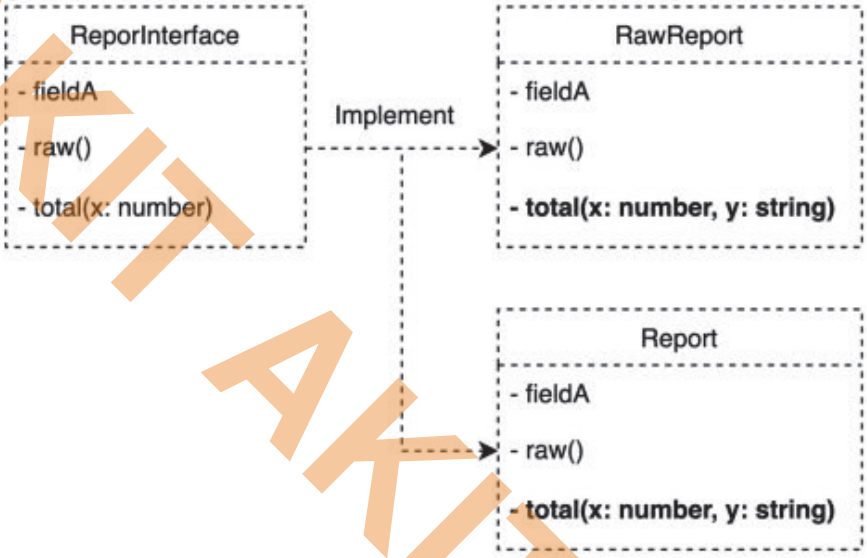

Рисунок 4.7 – Невдале формування дій при роботі з інтерфейсами

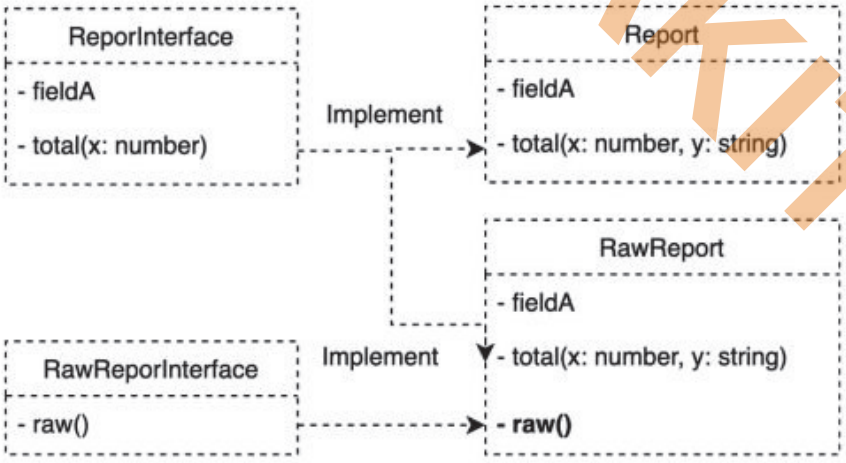

Рисунок 4.8 – Приклад використання принципу ISP при проектуванні роботи з інтерфейсами користувачів

Наведені приклади показують, що у програмний код простіше щось додати, ніж вносити зміни в уже існуючий продукт. Тому завжди слід використовувати принцип розбиття програми на прості інтерфейси. Бажано робити інтерфейси невеликим і не вводити в них ті речі, які не потрібні для роботи конкретного інтерфейсу. Розширюючи дію даного принципу можна стверджувати правило: у програмі слід створювати невеликі частини, кожна з яких робить окрему операцію, але роблять це дуже добре і саме так, як потрібно всім користувачам.

4.10.7 І нарештіDIP принцип. Відповідно з ним, у нашому програмному продукті на мають бути відображені наші особисті якості, вподобання тощо. Це ж відноситься й до різних програмних модулів. Прикладом застосування даного принципу може бути стандартне вікно, яке спливає при викликанні функції Printв багатьох стандартних програмних продуктах. Тут просто створено загальну модель, орієнтуючись на яку ми виберемо той сценарій дій, який нам треба, і який зробить цю дію так, як нам треба. Звичайно, що формувати програмно загальну модель необхідно за певними чіткими правилами, які сприймає більшість користувачів (рис.4.8).

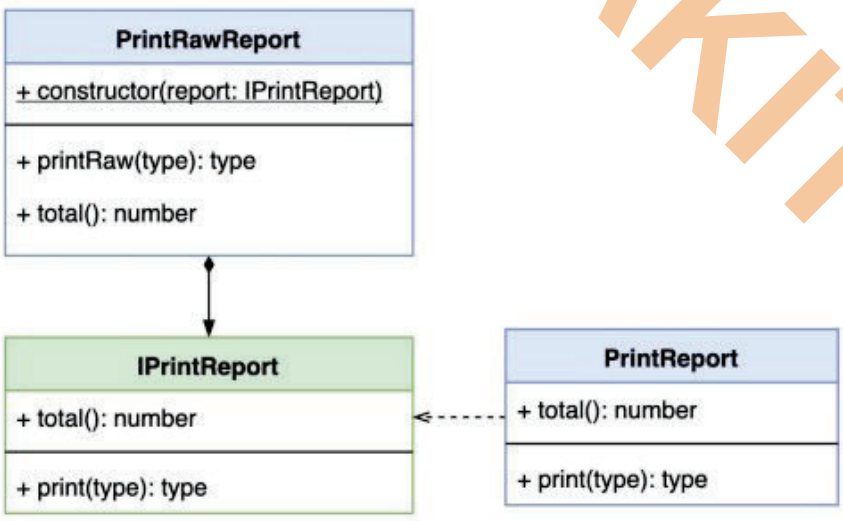

Рисунок 4.9 – Приклад правильного використання принципу DIP

Таким чином, кожен модуль має проектуватися таким чином, щоб його робота залежала лише від інтерфейсу, якийзможе в майбутньому реалізовувати будь-який користувач.

На закінчення відмітимо, що наведені вище принципи лягли в основу оптимізації нами коду інтегрованої системи кадрового забезпечення.

ВИСНОВКИ

У результаті виконання магістерської роботи:

1. Проведено короткий аналіз перспектив та проблем розвитку сучасних систем кадрового забезпечення підприємств та організацій. Зроблено порівняння наявних на ринку різних програмних продуктів управління кадрами.

2. Показано, що перспективним напрямком розвитку таких систем є їх інтеграція в більш загальні комп'ютерно-інтегровані системи, які об'єднують в собі бази даних різних організацій, наприклад, підприємства, служби зайнятості, керівних органів тощо.

3. Проведено оптимізацію типової кадрової системи забезпечення підприємства, яка забезпечує її легку інтеграцію в розширені комп'ютерно-інтегровані системи з доступом до інформації користувачів різного рівня з різних підприємств та організацій.

4. Розроблено архітектуру оптимізованої кадрової системи, яка включає набір спеціалізованих підсистем та базу даних.

5. Сформована концептуальна модель бази даних кадрової системи на основі методики побудови реляційних баз даних.

6. Розроблено програмне забезпечення оптимізованої кадрової системи та проведено його тестування,

7. Властивості та параметри оптимізованої в роботі системи кадрового забезпечення:

- мова програмування JavaScript;
- фреймворки Express та MaterialUi;
- бібліотека React;
- СУБД MongoAtlas, яка забезпечує надійну обробку даних;
- використання веб-інтерфейсу,
- незалежність від операційної системи.

8. На оптимізовану кадрову систему оформлено такі документи:

- опис програми;
- керівництво системного програміста;
- керівництво користувача;
-текст програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. IanGrehem. Object-OrientedMethods: Principles&Practice. –3-е изд. – New York: «Publisher», 2004. – 880 p.
2. Connolly R. FundamentalsofWebDevelopment / Pearson, 2017.
3. Benjamins V.R. AnIntelligentBrokeringServiceforKnowledge-ComponentReuseontheWorld – WideWeb / Benjamins V.R. // Proceedingsofthe 11th WorkshoponKnowledgeAcquisition, ModelingandManagement, June 2009, Чикаго, США, W360, 2009 C. P. 125-129.
4. Основныепринципыпостроения баз данных, проблемыхранениябольшихобъемовинформации [Електронний ресурс] – Режим доступу: https://sites.google.com/site/gosyvmkss12/bazy-dannyh/1-osnovnye-principy-postroenia-baz-dannyh-problemy-hranenia-bolsih-obemov-informacii.
5. Antoni Sintes. SamsTeachYourselfObject-OrientedProgrammingin 21 Days. – London: «Willams», 2002. – 672 p.
6. Duckett J. HTML & CSS DesignandBuildWebsites / J. Ducket. — Indianapolis: JohnWiley&Sons, 2011.
7. FrimenRobson. BasesTechnology. – New York: «Publisher», 2004. – 498p.
8. Дороніна М.С. Управління економічними та соціальними процесами підприємства. – Харків: ХДЕУ, 2002. – 431с
9. Aleks Jang. Node.js in reality. – Oksford: «Oksford Pres», 2012. – 367 p.
10. Sajmon Holms. PERNPostgreSql Express ReactNode. – London: «Willams», 2018. – 193 p.
11. Електронний ресурс. Режим доступу: https://blog.ingate.ru/seo-wikipedia/java-script
12. Електронний ресурс. Режим доступу: https://ru.hexlet.io/blog/posts/zachem-izuchat-node-js-ili-o-perspektivah-bekenda-na-javascript

13. Електронний ресурс. Режим доступу: https://habr.com/ru/company/piter/blog/265649/

14. Електронний ресурс. Режим доступу: https://code.visualstudio.com/docs

ДОДАТОК А
ОПИСПРОГРАМИ

Додаток містить у собі головну частину структури програми – серверну. Модель серверноїчастина проектузображено на рисунку А.1.

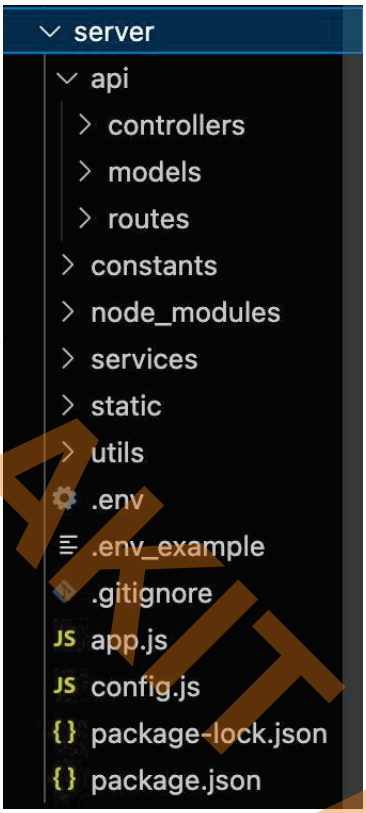

Рисунок А.1 – Модель структури серверної частини програми

Розпочнемо з основної папки API(ApplicationProgrammingInterface). Якщо коротко, то API – це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. Структура API складається з трьох основних елементів, таких як models, controllers, routes. Першою папкою йде controllers. Ця папка містить всі контролери нашої серверної частини. Контролером можна назвати таким елементом, який контролює інформацію, яка повертається клієнту. Тобто, якщо користувач просить надати йому інформацію про якийсь елемент А в базі, а в

нас нема цього елементу, то контролер поверне клієнту повідомлення про помилку і навпаки. Наступна папкаmodel має тільки одне призначення – це опис наших моделей або сутностей. Зокрема, якщо це модель **user**, то в папціз таким ім'ям буде міститись опис цієї сутності, її поля, методи, типи і зв'язки з іншими сутностями. Однією з папок є **routes**, яка містить рути серверної частини, тобтовизначаємаршрутизацію. Дана програма відповідає на клієнтський запит щодо звертання до конкретної адреси (URI).

Наступна папка **constants**. Вона містить всі константи проекту, тобто всі ті значення, які не змінюються у процесі роботи кадрової системи. Всі такі дані виносять в цю окрему папку і імпортують за потреби. Це дає змогу запобігати дублювання коду і збільшує його читабельність.

Node_modules є дуже важливою папкою, оскільки в ній містяться всі зовнішні по відношенню до нашого проекту об'єкти. Наприклад, нам потрібно встановити бібліотеку, яка шифрує дані. Після встановленні така бібліотека буде знаходитись саме в цій папці і всі її функціональні методи ми зможемо використовувати.

Наступна папка є **services**, уякій знаходяться всі сервіси нашого проекту. Такі сервіси потрібні для зв'язку з базою даних. Вони являють собою простий ланцюжок. Наприклад, в нас є контролер, він робить відповідний запит до нашого сервісу, а сервіс, обробивши його, направляє до бази даних. Таким чином в сервісах описується всі можливі звернення до бази даних.

Останніми папками є **static** та**utils**.Папка статік потрібна для статичних елементів, наприклад для тих незмінних інтерфейсів, які ми зберігаємо на сервері у вигляді якогось зображення. Саме такі інформаційні елементи будуть знаходитись в цій папці.

Папка утілс використовується в ролі помічника при роботі нашого сервісу. Всі допоміжні або проміжні функції, які ми використовуємо в нашому сервісі, виносяться саме до папки утілс. Завдяки такій організації не відбувається дублювання функцій і також збільшуємо читабельність коду.

Тепер переходимо до файлів, які знаходяться на однаковому ієрархічному рівні з нашими папками. Файл з розширенням**.env**містить в собі секретні значення, такі як ключі від баз даних, секретні слова чи унікальний шлях до сервісів. Файли такого типу є локальними. Одним із таких файлів є.env_examle. цей файл є прикладом .env,але в ньому містяться тільки назва полів, а самих значень нема.

Головний файл без якого наш сервер не запуститься це app.js. За допомогою цього файлі наш сервер запускається в роботу і підключається до бази даних. Відповідно для такої процедури в config.jsonописується конфігурація серверу.

Ще дуже важливим файлами є package-lock.json та package.json. Такі файли з розширенням json містять інформацію про версію програмного продукту, атакож служать у ролі пакетних файлів.

ДОДАТОК Б
КЕРІВНИЦТВО СИСТЕМНОГО ПРОГРАМІСТА

Для забезпечення безвідмовної роботи системи треба дотримуватися основних вимог при інсталяції та рекомендацій щодо її використання. Також потрібне постійне та якісне з'єднання з мережею Internet для нормального функціонування системи в комп'ютерно-інтегрованому середовищі.

У даному додатку наведено відомості щодо детальних системних вимог та інструкцію системному адміністратору або програмісту щодо інсталяції програмної системи.

Б.1 Системні вимоги

Для використання розробленої програмної системи персональний комп'ютер повинен мати операційну систему MSWindows (XP, 7, 8, 8.1, 10), чи дистрибутив Linux (Ubuntu, Debian, BSD), Mac OS. Крім того, на персональному комп'ютері повинні бути встановлений будь-який веб браузер.

Якщо виникає потреба використання додатку для розробки, для виправлення недоліків чи покращення існуючого функціоналу, потрібно усі необхідні для цього програмні пакети. Бажано, щоб всі вони були ліцензійними та завантаженими на сервері.

Б.2 Організація роботи сервера

Для того щоб продемонструвати принцип роботи сервера, доцільно використатипрограму Postman.Postman – це сервіс, який застосовується для ручного та автоматизованого тестування HTTP API. З його допомогою можна виконувати будь-які запити через зручний веб-інтерфейс, створювати тести

роботи API в автоматичному режимі та багато іншого. Вікно виконання запитів та види відповідей адміністратора серверу наведено на рисунках Б.1 та Б.2.

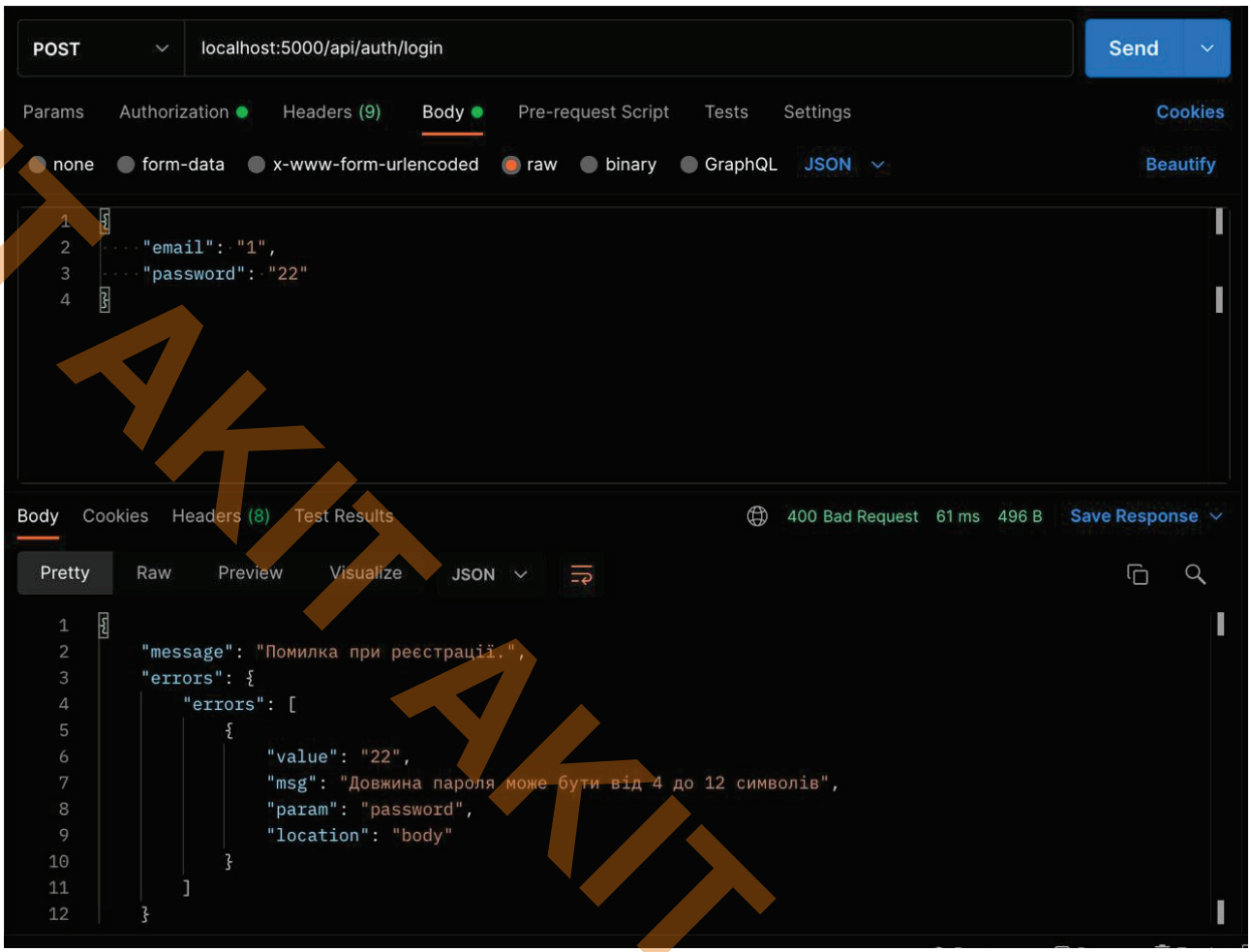

Рисунок Б.1 –Приклад вікна програми Postman із не коректним запитом

У цьому запиті до серверу використовується тип Post, який означає, що в запиті буде міститись певне значення в його тілі. Зазвичай в тілі запиту записують поле email тапassword. Цим полям у прикладі ми умовно задали значення “1” для пошти і “2” для паролю. Сервер нам повернув помилку зі статус кодом 400, який означає BadRequest. У тілі відповіді ми також виділяємо поле message, де описується загальне повідомлення про помилку тарозшифровується всі параметри і позиція помилки.

Наступним кроком адміністратора буде прослідкувати, як цей запит проходить на основному сервері. Таким чином робиться перевірка правильності проходження відповідного маршруту запитом даного типу.

Для конкретизації в нашому запиті тип був Post і маршрут запиту localhost:5000/api/auth/login. Таким чином, відправлений кимось запит на наш сервер спочатку проходить черезроутер, який перевіряє чи в нашому сервері створений необхідний маршрут із заданим типом запиту. Якщо такий маршрут не створений, то сервер генерує повідомлення про помилку (у даному випадку це помилка зі статусом 404).

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

const AuthController = require('../../controllers/authController');
const { ValidateAuth } = require('../../../utils/validate/index');

router.post('/login', ValidateAuth, AuthController.login);
router.post('/registration', ValidateAuth, AuthController.registration);

module.exports = router;
```

Рисунок Б.2 – Приклад повідомлення роутера авторизації

Якщо запит успішно пройшов маршрут через роутер, то розпочинається перевірка маршруту по типах параметрів в тілі запиту. У нашому випадку це перший параметр Post, зміст якого є відповідний логін.

Наступним кроком є проходження запиту через проміжкову функцію ValidateAuth. При успішному проходженні даного кроку запит потрапляє в контролер. Останній перевіряє, чи проміжкова функція знайшла помилки у запиті.

Якщо є помилки, то контролерпосилає ці помилки адміністратору, а запит не йде далі. Приклад реакції нашого логін контролера зображенона рисунку Б.3.

Якщо в аналізованому нашому запиті проміжкова функція не знаходить помилок, то контролер обробляє отриманий запит й витягує з його тіла відповідні email тапassword.

```
async login(req, res) {
    try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({
                message: 'Помилка при реєстрації.',
                errors,
            });
        }

        const { email, password } = req.body;

        const user = await findUserByEmail(email);

        if (!user)
            return res.status(400).json({
                message: 'Користувача з даним емайлом не створено.',
            });

        const isPasswordValid = bcrypt.compareSync(password, user.password);

        if (!isPasswordValid)
            return res.status(400).json({ message: 'Невірний пароль.' });

        const token = await generateJwtToken(user._id, user.roles);

        return res
            .status(200)
            .json({
                email: user.email,
                id: user._id,
                token: token,
                message: 'Ви успішно увійшли!',
            });
    } catch (e) {
        res.status(400).json({ message: 'Користувач не увійшов.' });
    }
}
```

Рисунок Б.3 – Приклад маршруту параметра логін в контролері авторизації

Якщо у запиті, який надійшов до контролера є всі необхідні дані, то контролер розпочинає їхню перевірку. Спочатку співставляються параметри електронної пошти для визначення простої події – чи за наведеним поштовим адресом в базі даних кадрової системи існую зареєстрований внутрішній чи зовнішній користувач. У випадку негативної відповіді контролер видає адміністратору відповідну помилку про відсутність такого користувача.

Наступним кроком перевірки є звірка паролів. Для цього контролер перевіряє відповідність між собою паролю, який є в базі даних, і паролю, який ввів користувач. При співпадінні паролів контролер генерує повідомлення

користувачу про доступ на певний час до кадрової системи, а також посилає серверу статус код 200, який означає вхід користувача в систему. Приклад реакції контролера на успішний запит наведено на рисунку Б.4. У випадку не співпадіння паролю .сервер отримує статус заборони доступу.

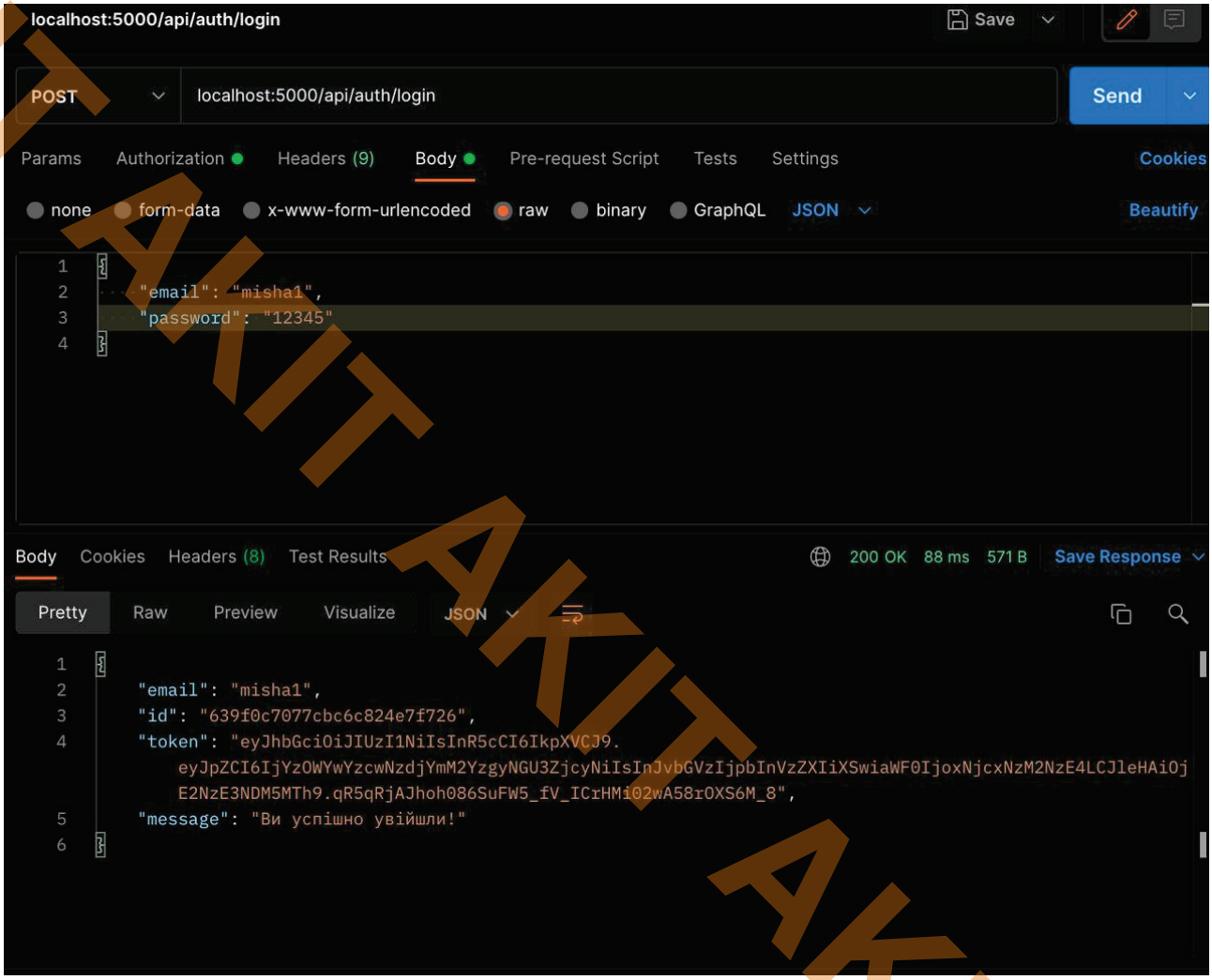

Рисунок Б.4 – Приклад вікна реакції контролера при успішному запиті

ДОДАТОК В
КЕРІВНИЦТВО КОРИСТУВАЧА

Користувачі кадрової системи працюють з відповідною інтерфейсною частиною програми. Така інтерфейсна частина поділяється на ряд вікон, кожне з яких має свою адресу у браузерному рядку. Навігація між вікнами реалізована за допомогою простого меню.

Користувач взаємодіє з графічним інтерфейсом у вигляді доступного веб додатку. Оскільки система працює з даними, які мають обмежений доступ, то обов'язковою єавторизація всіх користувачів, які збираються відвідувати сайт. Таким чином, першим, що новий користувач бачить при відвідуванні відповідної веб-сторінки, це вікно авторизації або ж реєстрації. Авторизація відбувається за допомогою введення адреси електронної пошти та паролю даного користувача. На основі цих даних проходить реєстрація користувача в системі з відповідними параметрами доступу. На рисунку В.1 зображено головне вікно інтерфейсної частини користувача.

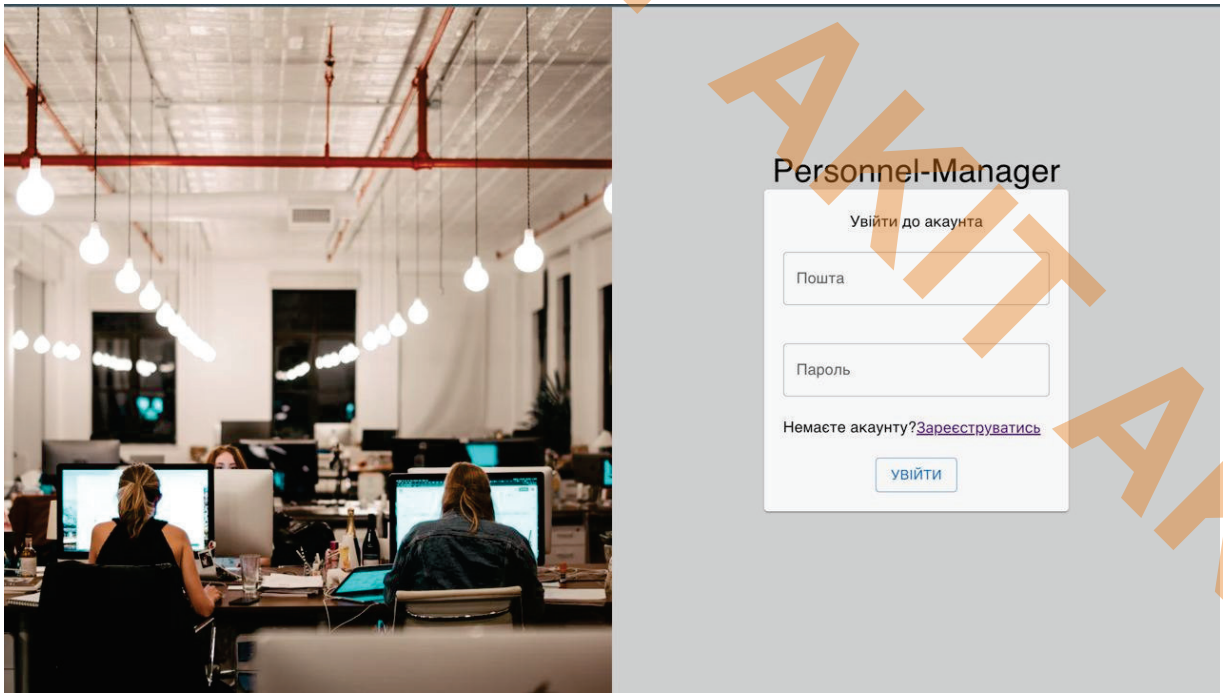

Рисунок В.1 – Вигляд головного вікна інтерфейсної частини користувача

Після авторизації та отримання дозволу адміністратора на доступ, користувач може перейти на головну сторінку сайту. Тут відбувається «привязка користувача до відповідної організації. Вигляд компоненту організації наведено на рисунку В.2.

Рисунок В.2 – Вигляд сторінки визначення організації

Для того щоб користувач дістав дозвіл на переглянув необхідної частини інформації сайту, він повиненвказати ту організацію, яку він представляє. Зокрема, користувача може бути під’єднано до списку вакансій, які на даний час пропонує вибрана організація. Загальне зображення профілю організації наведено на рисунку В.3.

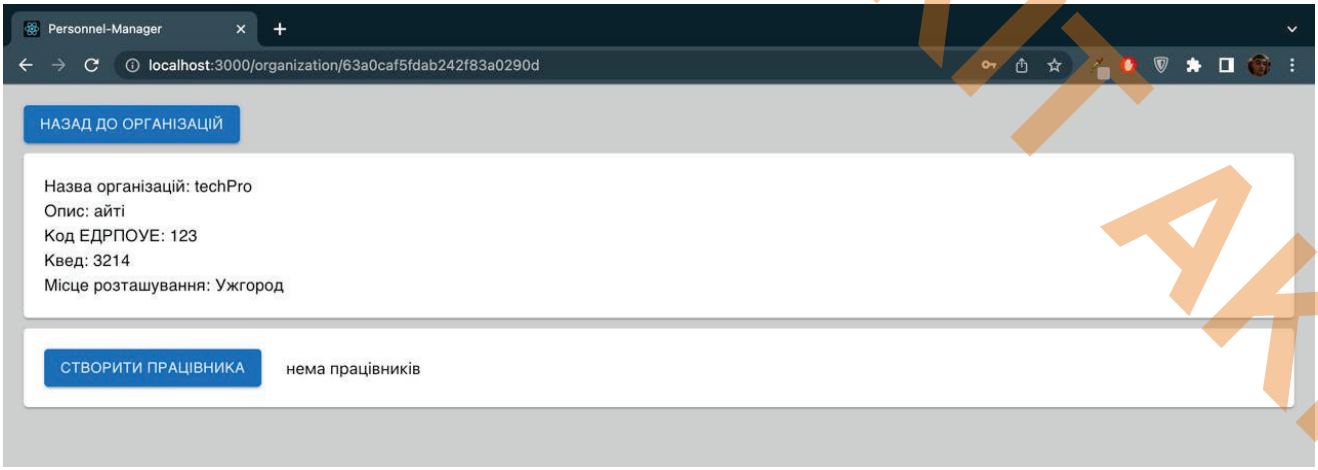

Рисунок В.3 – Приклад вікна інформації про вибрану організацію

Окремою функцією кадрового доступу є додавання до компютерно-інтегрованої системи нових організацій та нових працівників до вже існуючих організацій. Такі операції відбуваються за допомогою модальних вікон з набором полів. Всі запропоновані поля потрібно заповнити для занесення організації чи її працівника до бази даних. Зображення одного з модальних вікон наведено на рисунку В.4.

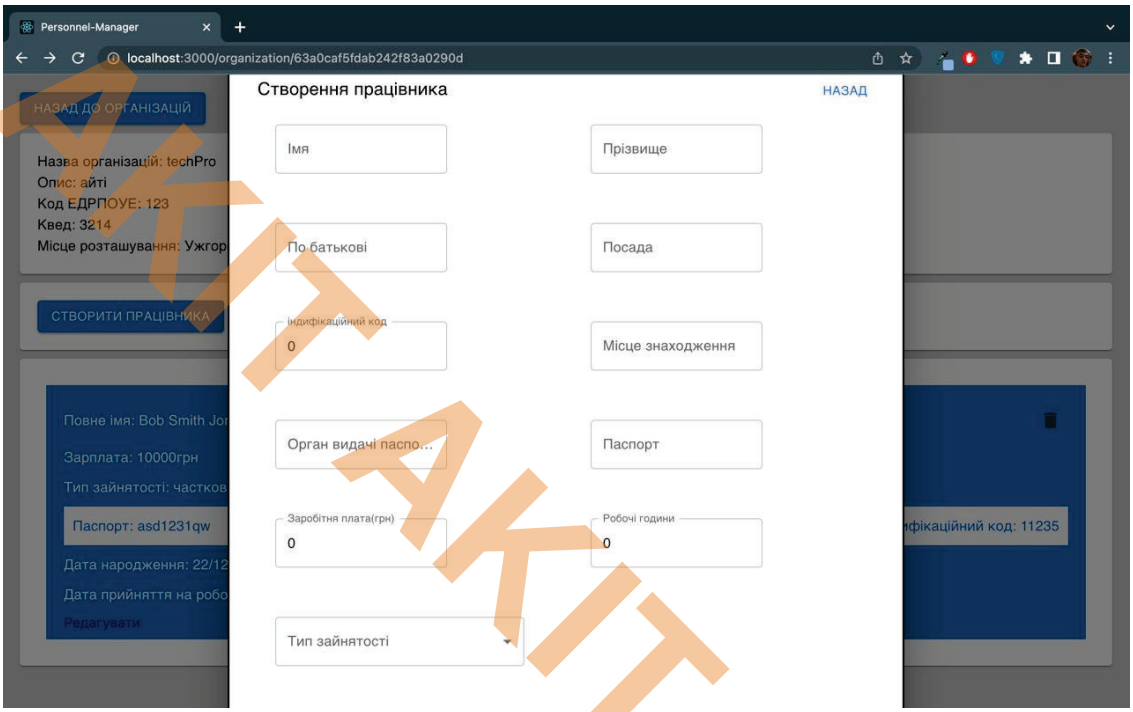

Рисунок В.4 – Приклад модального вікна для внесення нового працівника

Якщо з часом з’являється необхідність змінити дані про працівника чи організацію, то сервер дає змогу для цього скористатися опцією редагування відповідного запису бази даних (рис. В.5). Однак для редагування відповідного запису слід отримати дозві адміністратора сайту.

Після того як користувач натисне на кнопку редагувати, він перейде на сторінку редагування працівника, де зможе змінити значення відповідних полів. Якщо ж такої потреби немає, користувач має змогу повернутись назад для продовження роботи на сайті.

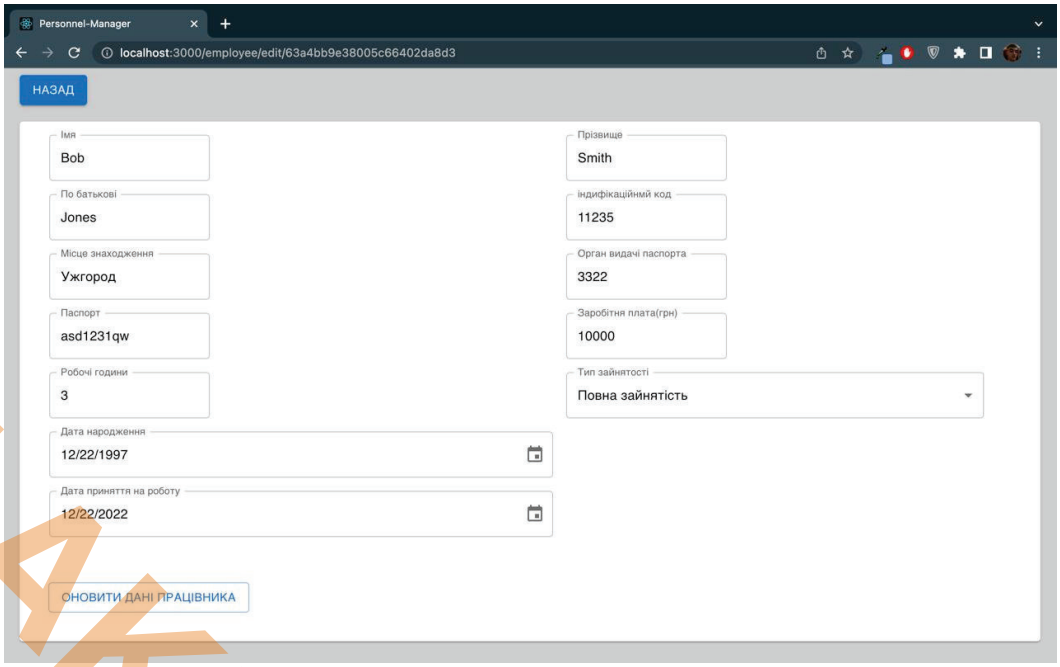

Рисунок В.5 – Зображення сторінки редагування параметрі працівника

Переглянути всі дані про певного працівника можна за допомогою вікна, наведеного на рис.В.6.

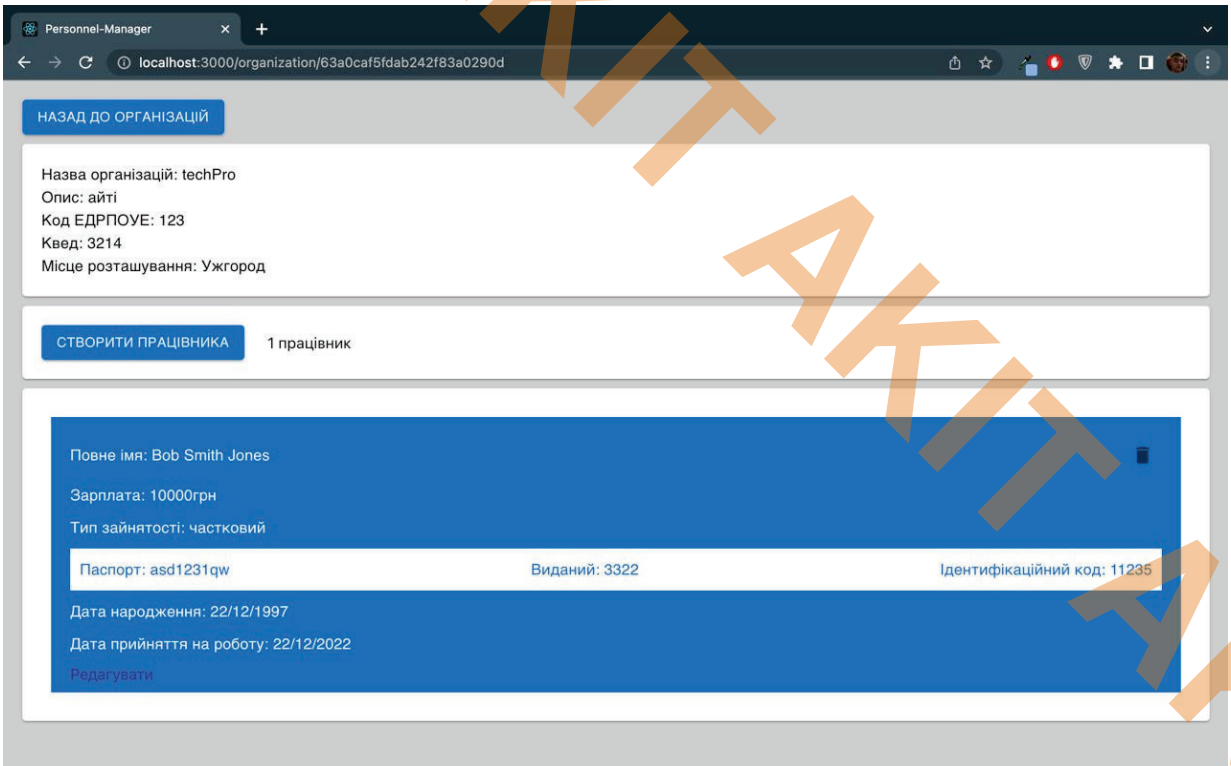

Рисунок В.6 – Зображення сторінки надання повної інформації про працівника в організації

ДОДАТОК Г
ТЕКСТ ПРОГРАМИ

```
constbcrypt = require('bcrypt');

constUser = require('./../models/user');
constRole = require('./../models/role');

const { findUserByEmail } = require('../../services/authServise');
const { validationResult } = require('express-validator');
const { generateJwtToken } = require('../../utils/jwt');

classAuthController {
asynclogin(req, res) {
try {
if (!errors.isEmpty()) {
returnres.status(400).json({
message: 'Помилка при реєстрації.',
errors,
            });
          }

const { email, password } = req.body;

constuser = awaitfindUserByEmail(email);

if (!user)
returnres.status(400).json({
message: 'Користувача з даним емайлом не створено.',
```

```
            });

constisPasswordValid = bcrypt.compareSync(password, user.password);

if (!isPasswordValid)
returnres.status(400).json({ message: 'Невірний пароль.' });

consttoken = awaitgenerateJwtToken(user._id, user.roles);

returnres.status(200).json({ token: token });
        } catch (e) {
res.status(400).json({ message: 'Користувач не увійшов.' });
        }
    }

asyncregistration(req, res) {
try {
consterrors = validationResult(req);
if (!errors.isEmpty()) {
returnres
                .status(400)
                .json({ message: 'Помилка при реєстрації.', errors });
            }

const { email, password } = req.body;
constisUserExist = awaitfindUserByEmail(email);
if (isUserExist)
returnres
                .status(400)
                .json({ message: 'Користувач з такою поштою вже існує.' });
```

```
consthashPassword = bcrypt.hashSync(password, 7);
constdefaultRole = awaitRole.findOne({ title: 'user' });
constnewUser = newUser({
email: email,
password: hashPassword,
roles: [defaultRole.title],
        });

awaitnewUser.save();

returnres.status(200).json({
message: 'Новий користувач успішно зареєстрований!',
        });
      } catch (e) {
res.status(400).json({ message: 'Користувач не зареєстрований.' });
      }
   }
}

module.exports = newAuthController();

const {
createEmployee,
updateEmployee,
deleteEmployee,
getEmployeeById,
} = require('../../services/employeeServise');

classEmployee {
```

```
asynccreate(req, res) {
try {
const { body } = req;
constnewEmployee = awaitcreateEmployee(body);

returnres.status(200).json({ data: newEmployee });
    } catch (error) {
res.status(400).json({
message: 'Працівник не створений.',
error: error,
        });
      }
  }

asyncupdate(req, res) {
try {
const {
body,
params: { id },
        } = req;
constisEmployeeExists = awaitgetEmployeeById(id);

if (!isEmployeeExists) {
res.status(404).json({
message: 'Працівник не знайдений.',
            });
          }

awaitupdateEmployee(id, body);
```

```
returnres.status(200).json({ message: 'Організацію оновлено.' });
        } catch (error) {
res.status(400).json({
message: 'Працівник не оновленений.',
error: error,
            });
        }
    }

asyncget(req, res) {
try {
const {
params: { id },
            } = req;
constisEmployeeExists = awaitgetEmployeeById(id);

if (!isEmployeeExists) {
returnres.status(404).json({
message: 'Організацію не знайдено.',
                });
            }

returnres.status(200).json({ data: isEmployeeExists });
        } catch (error) {
res.status(400).json({
message: 'Помилка запиту.',
error: error,
            });
        }
    }
```

```
asyncdelete(req, res) {
try {
const {
params: { id },
        } = req;

awaitdeleteEmployee(id);

returnres.status(200).json({ message: 'Видалено.' });
      } catch (error) {
res.status(400).json({
message: 'Працівник не видалений.',
error: error,
        });
      }
   }
}

module.exports = newEmployee();

const {
createSalary,
updateSalary,
deleteSalary,
getSalaryById,
} = require('../../services/organizationServise');

classSalary {
asynccreate(req, res) {
```

```
try {
const { body } = req;
constnewSalary = awaitcreateSalary(body);

returnres.status(200).json({ data: newSalary });
      } catch (error) {
res.status(400).json({
message: 'Видача зарплати не створена.',
error: error,
      });
      }
   }

asyncupdate(req, res) {
try {
const {
body,
params: { id },
          } = req;
constisSalaryExists = awaitgetSalaryById(id);

if (!isSalaryExists) {
res.status(404).json({
message: 'Видача зарплати не знайдена.',
             });
          }

awaitupdateSalary(id, body);

returnres.status(200).json({ message: 'Організацію оновлено.' });
```

```
        } catch (error) {
res.status(400).json({
message: 'Видача зарплати не оновлена.',
error: error,
            });
        }
    }

asyncget(req, res) {
try {
const {
params: { id },
            } = req;
constisSalaryExists = awaitgetSalaryById(id);

if (!isSalaryExists) {
returnres.status(404).json({
message: 'Організацію не знайдено.',
                });
            }

returnres.status(200).json({ data: isSalaryExists });
        } catch (error) {
res.status(400).json({
message: 'Помилка запиту.',
error: error,
            });
        }
    }
```

```
asyncdelete(req, res) {
try {
const {
params: { id },
        } = req;

awaitdeleteSalary(id);

returnres.status(200).json({ message: 'Видалено.' });
      } catch (error) {
res.status(400).json({
message: 'Видача зарплати не видалена.',
error: error,
        });
      }
   }
}

module.exports = newSalary();

const {
createOrganization,
updateOrganization,
deleteOrganization,
getOrganizationById,
} = require('../../services/organizationServise');

classOrganization {
asynccreate(req, res) {
try {
```

```
const { body } = req;
constnewOrganization = awaitcreateOrganization(body);

returnres.status(200).json({ data: newOrganization });
      } catch (error) {
res.status(400).json({
message: 'Організація не створена.',
error: error,
      });
    }
  }

asyncupdate(req, res) {
try {
const {
body,
params: { id },
        } = req;
constisOrganizationExists = awaitgetOrganizationById(id);

if (!isOrganizationExists) {
res.status(404).json({
message: 'Організація не знайдена.',
          });
        }

awaitupdateOrganization(id, body);

returnres.status(200).json({ message: 'Організацію оновлено.' });
      } catch (error) {
```

```
res.status(400).json({
message: 'Організація не оновлена.',
error: error,
        });
    }
  }

asyncget(req, res) {
try {
const {
params: { id },
        } = req;
constisOrganizationExists = awaitgetOrganizationById(id);

if (!isOrganizationExists) {
returnres.status(404).json({
message: 'Організацію не знайдено.',
          });
        }

returnres.status(200).json({ data: isOrganizationExists });
    } catch (error) {
res.status(400).json({
message: 'Помилка запиту.',
error: error,
        });
    }
  }

asyncdelete(req, res) {
```

```
try {
const {
params: { id },
        } = req;

awaitdeleteOrganization(id);

returnres.status(200).json({ message: 'Видалено.' });
      } catch (error) {
res.status(400).json({
message: 'Організація не видалена.',
error: error,
        });
      }
    }
}

module.exports = newOrganization();

constexpress = require('express');

constauth = require('./auth/index');
constuser = require('./user/index');
constorganization = require('./organization/index');
constemployee = require('./employee/index');

constrouter = express.Router();

router.use('/auth', auth);
router.use('/user', user);
```

```
router.use('/organization', organization);
router.use('/employee', employee);

module.exports = router;

constexpress = require('express');
const { default: mongoose } = require('mongoose');
constdotenv = require('dotenv');

dotenv.config();
const URI = process.env.MONGODB_URL || null;
const PORT = process.env.PORT || 8000;

constapiRoutes = require('./api/routes/index');

varcors = require('cors');

constapp = express();
app.use(cors());
app.use(express.json());
app.use('/api', apiRoutes);

asyncfunctionconnect() {
try {
awaitmongoose.connect(URI);
app.listen(PORT, () => {
        console.log(`Server runningonport: ${PORT}`);
    });
  } catch (err) {
    console.log(`Error ~> ${err}`);
```

```
    }
}
connect();

constexpress = require('express');
constrouter = express.Router();

constAuthController = require('../../controllers/authController');
const { ValidateAuth } = require('../../../utils/validate/index');

router.post('/login', ValidateAuth, AuthController.login);
router.post('/registration', ValidateAuth, AuthController.registration);

module.exports = router;

constexpress = require('express');
constrouter = express.Router();

const { token } = require('../../../utils/middlewares');
constOrganizationController = require('../../controllers/organizationController');

router.post('/', token, OrganizationController.create);

router.get('/:id', token, OrganizationController.get);

router.delete('/:id', token, OrganizationController.delete);

router.put('/:id', token, OrganizationController.update);

module.exports = router;
```

```
constUser = require('../api/models/user');

constfindUserByEmail = async (email) =>awaitUser.findOne({ email });

module.exports = { findUserByEmail };

constEmployee = require('../api/models/employee');

constcreateEmployee = async (newBody) =>awaitEmployee.create(newBody);

constupdateEmployee = async (id, updatedBody) =>
awaitEmployee.findByIdAndUpdate(id, updatedBody);

constdeleteEmployee = async (id) =>
awaitEmployee.deleteOne({ _id: id }).exec();

constgetEmployeeById = async (id) =>awaitEmployee.findOne({ _id: id });

module.exports = {
createEmployee,
updateEmployee,
deleteEmployee,
getEmployeeById,
};
constEmployee = require('../api/models/employee');

constcreateEmployee = async (newBody) =>awaitEmployee.create(newBody);

constupdateEmployee = async (id, updatedBody) =>
```

```
awaitEmployee.findByIdAndUpdate(id, updatedBody);

constdeleteEmployee = async (id) =>
awaitEmployee.deleteOne({ _id: id }).exec();

constgetEmployeeById = async (id) =>awaitEmployee.findOne({ _id: id });

module.exports = {
createEmployee,
updateEmployee,
deleteEmployee,
getEmployeeById,
};

const { secret } = require('../../config');

constjwt = require('jsonwebtoken');

constgenerateJwtToken = (id, roles) => {
constpayload = { id, roles };
returnjwt.sign(payload, secret, { expiresIn: '2h' });
};

module.exports = { generateJwtToken };

constjwt = require('jsonwebtoken');
const { secret } = require('../../config');

consttoken = (req, res, next) => {
constisOptions = req.method === 'OPTIONS';
```

```
if (isOptions) next();

try {
consttoken = req.headers.authorization?.split(' ')[1];
if (!token) {
returnres
            .status(401)
            .json({ message: 'Користувач не авторизувався.' });
        }

constdecodedData = jwt.verify(token, secret);
req.user = decodedData;
next();
    } catch (err) {
        console.log('tokenerr', err);
returnres
            .status(401)
            .json({ message: 'Користувач не авторизувався.' });
    }
};

constrole = (roles) => {
return (req, res, next) => {
constisOptions = req.method === 'OPTIONS';
if (isOptions) next();

try {
consttoken = req.headers.authorization?.split(' ')[1];

if (!token) {
```

```
returnres
                .status(401)
                .json({ message: 'Користувач не авторизувався.' });
        }

const { roles: userRoles } = jwt.verify(token, secret);
lethasRole = false;
userRoles?.forEach((role) => {
if (roles?.includes(role)) {
hasRole = true;
            }
        });

if (!hasRole) {
returnres
                .status(403)
                .json({ message: 'В цього юзера нема доступу.' });
        }
next();
    } catch (err) {
        console.log('tokenerr', err);
returnres
            .status(401)
            .json({ message: 'Користувач не авторизувався.' });
    }
  };
};

module.exports = { token, role };
```

```
const { check } = require('express-validator');

constValidateAuth = [
check('email', 'Емейл не може бути пустим').notEmpty(),
check('password', 'Довжина пароля може бути від 4 до 12 символів').isLength(
        { min: 4, max: 12 }
    ),
];

module.exports = { ValidateAuth };
const { Schema, model } = require('mongoose');

constOrganization = newSchema({
avatar: {
type: String,
required: true,
    },
title: {
type: String,
required: true,
    },
owner: {
type: Schema.Types.ObjectId,
ref: 'User',
    },
description: {
type: String,
required: true,
    },
employees: [
```

```
        {
type: Schema.Types.ObjectId,
ref: 'User',
        },
    ],
location: {
type: String,
required: true,
    },
codeEDRPOYE: {
type: Number,
required: true,
unique: true,
    },
kwed: {
type: Number,
required: true,
    },
});

module.exports = model('Organization', Organization);
const { Schema, model } = require('mongoose');

constUser = newSchema({
email: {
type: String,
unique: true,
required: true,
    },
password: {
```

```
type: String,
required: true,
    },
roles: [
        {
type: String,
ref: 'Role',
        },
    ],
});

module.exports = model('User', User);
```