Військовий інститут телекомунікацій та інформатизації

Імені Героїв Крут

Кафедра №22

«Комп’ютерних інформаційних технологій»

**Курсова робота**

З дисципліни: «Технологія розробки програмного забезпечення»

Виконав:

Курсант 212 навчальної групи

солдат А.Кожухар

Керівник:  
 викладач кафедри №22

майор С.Романенко

Київ 2024

АНОТАЦІЯ

Курсової роботи на тему:

«Розробка *RESTfull* додатку на основі технології Node.js для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати»

Курсова робота : містить 30 сторінок, 7 рисунків, 12 джерел

Метою курсової роботи є розробка *RESTful* додатку для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати з використанням технології Node.js. Додаток автоматизує процеси обліку заробітної плати, покращуючи управління ресурсами та знижуючи ймовірність помилок.

У роботі розглянуто принципи побудови *RESTful* сервісів та особливості використання Node.js для створення серверних додатків. Практична частина включає розробку програмного забезпечення для реєстрації, обліку, пошуку та модифікації даних про заробітну плату, а також генерацію звітів. Використано Express.js для серверної логіки та *MongoDB* для роботи з базами даних.

Результати роботи можуть бути використані для автоматизації обліку заробітної плати у бухгалтерії інституту, підвищуючи ефективність управління ресурсами.

*ANNOTATION*

*Coursework on the topic:*

*"Development of a RESTful application based on Node.js technology for automating the payroll accounting activities of an institute’s accounting department"*

*The coursework contains 30 pages, 8 figures, and 8 sources.*

*The purpose of the coursework is to develop a RESTful application for automating the payroll accounting activities of an institute's accounting department using Node.js technology. The application automates payroll accounting processes, improving resource management and reducing the likelihood of errors.*

*The work examines the principles of building RESTful services and the peculiarities of using Node.js for creating server applications. The practical part includes the development of software for registration, accounting, searching, and modifying payroll data, as well as generating reports. Express.js is used for server logic, and MongoDB is used for working with databases.*

*The results of the work can be used to automate payroll accounting in an institute’s accounting department, thereby increasing the efficiency of resource management.*

*.*

Зміст

[АНОТАЦІЯ 2](#_Toc172493977)

[*ANNOTATION* 3](#_Toc172493978)

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 4](#_Toc172493979)

[РОЗДІЛ 1. Аналіз RESTful додатку на основі технології *Node*.*js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати 7](#_Toc172493980)

[РОЗДІЛ 2. Проектування та особливості розробки *RESTful* додатку на основі технології *Node.js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати 10](#_Toc172493981)

[2.2 Вимоги до системи 10](#_Toc172493982)

[2.3 Архітектура системи 11](#_Toc172493983)

[2.4 Вибір інструментів та технологій 12](#_Toc172493984)

[2.5 Розробка серверної частини 13](#_Toc172493985)

[2.6 Розробка клієнтської частини 14](#_Toc172493986)

[2.7 Забезпечення безпеки та масштабованості 14](#_Toc172493987)

[РОЗДІЛ 3. Проектування та розробка програмного модуля 16](#_Toc172493988)

[3.1 Вибір архітектури в відповідності до шаблону *MVC* 16](#_Toc172493989)

[3.2 Розробка складових 16](#_Toc172493990)

[ВИСНОВОК 26](#_Toc172493991)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 28](#_Toc172493992)

[ДОДАТОК А 29](#_Toc172493993)

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

JS – JavaScript

REST – Representational State Transfer

HTTP – HyperText Transfer Protocol

NPM – Node Package Manager

CPU – Central Processing Unit

CLI – Command-Line Interface

ORM – Object-Relational Mapping

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

UI – User Interface

JWT – JSON Web Tokens

DB – Database

MVC – Model-View-Controller

CRUD – Create, Read, Update, Delete

JSON – JavaScript Object Notation

EJS – Embedded JavaScript Templating

IDE – Integrated Development Environment

SSL – Secure Sockets Layer

SSH – Secure Shell

CORS – Cross-Origin Resource Sharing

NoSQL – Not Only SQL (used to describe databases like MongoDB)

OAuth – Open Authorization

SPA – Single-Page Application

CSR – Client-Side Rendering

SSR – Server-Side Rendering

ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у покращенні ефективності управління різними аспектами діяльності організацій. Зокрема, автоматизація процесів обліку заробітної плати дозволяє значно знизити витрати часу та ресурсів, а також підвищити точність і надійність даних. Освітні заклади, як і інші організації, потребують ефективних інструментів для обліку та управління своїми фінансовими ресурсами.

Однією з найпопулярніших технологій для розробки сучасних веб-додатків є *Node.js*, яка забезпечує високу продуктивність та масштабованість. Використання цієї технології у поєднанні з *RESTful* архітектурою дозволяє створити потужні та гнучкі рішення для різноманітних задач.

Метою даної курсової роботи є розробка *RESTful* додатку на основі технології *Node.js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати. Основні завдання включають вивчення принципів побудови *RESTful* сервісів, аналіз можливостей *Node.js* для серверної розробки, розробку програмного забезпечення для обліку заробітної плати та впровадження засобів для підвищення безпеки, ефективності та масштабованості додатку.

У ході роботи буде розглянуто теоретичні аспекти побудови *RESTful* додатків, проведено аналіз існуючих рішень та технологій, а також розроблено практичне рішення, яке буде протестовано і впроваджено. Очікувані результати включають покращення процесів обліку заробітної плати та підвищення загальної ефективності управління ресурсами в освітніх закладах.

РОЗДІЛ 1. Аналіз *RESTful* додатку на основі технології *Node*.*js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати

1.1 Вступ

В останні роки інформаційні технології стали невід'ємною частиною процесів управління заробітною платою в різних організаціях, включаючи освітні заклади. Сучасні веб-додатки, побудовані на основі *RESTful* архітектури та платформи *Node*.*js*, дозволяють ефективно автоматизувати ці процеси, забезпечуючи зручний доступ до даних та високий рівень продуктивності. У цьому розділі буде проведено аналіз основних аспектів розробки та використання *RESTful* додатків на основі *Node*.*js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати.

1.2 Огляд *RESTful* архітектури

*REST* (*Representational State Transfer*) - це архітектурний стиль для розробки мережевих додатків, який базується на використанні простих *HTTP*-запитів для взаємодії з ресурсами. *RESTful* додатки відрізняються своєю простотою, масштабованістю та гнучкістю. Основні принципи *REST* включають:

Використання *HTTP* методів: *RESTful* сервіси використовують стандартні методи *HTTP*, такі як *GET, POST, PUT і DELETE* для виконання операцій з ресурсами. *GET* використовується для отримання інформації, *POST* - для створення нових ресурсів, *PUT* - для оновлення існуючих, а *DELETE* - для видалення.

Структурування *URL*: Кожен ресурс ідентифікується унікальним *URL*. Наприклад, для обліку заробітної плати *URL* може виглядати так: */salaries*, де кожна заробітна плата має свій унікальний ідентифікатор*: /salaries/{id}.*

Формати передачі даних: *RESTful* сервіси зазвичай використовують *JSON* або *XML* для передачі даних між клієнтом і сервером, що забезпечує сумісність і зручність роботи з даними.

Статусні коди *HTTP*: *RESTful* сервіси використовують стандартні коди статусу *HTTP* для інформування клієнтів про результати їхніх запитів. Наприклад, 200 (*OK*) для успішних запитів, 404 (*Not Found*) для неіснуючих ресурсів, 500 (*Internal Server Error*) для помилок сервера.

1.3 Вибір технології *Node.js*

*Node.js* - це платформа для розробки серверних додатків на основі мови програмування *JavaScript*. Основні переваги *Node.js* включають:

Подіє-орієнтована, неблокуюча модель введення-виведення: *Node.js* використовує подіє-орієнтовану, неблокуючу модель введення-виведення, що дозволяє обробляти велику кількість одночасних з'єднань без значного навантаження на сервер.

Велика кількість модулів та пакетів: Наявність великої кількості модулів та пакетів, доступних через npm *(Node Package Manager*), спрощує розробку та розширення функціональності додатків. Для розробки *RESTful* додатку можуть бути використані модулі, такі як *Express*.*js* для побудови серверної логіки, *Mongoose* для роботи з *MongoDB*, *Passport* для аутентифікації та багато інших.

Єдина мова програмування: *Node.js* дозволяє використовувати одну мову програмування, *JavaScript*, для розробки як серверної, так і клієнтської частини додатку, що спрощує процес розробки та обслуговування.

1.4 Архітектура *RESTful* додатку на основі *Node.js*

Основні компоненти архітектури *RESTful* додатку включають:

Серверна частина: Серверна частина додатку будується на основі Express.js - популярного фреймворку для *Node.js*, який забезпечує гнучкість і простоту у створенні *RESTful* *API*. *Express.js* дозволяє легко визначати маршрути для обробки *HTTP*-запитів, реалізовувати *middleware* для обробки запитів та відповідей, а також підключати додаткові модулі для розширення функціональності.

База даних: Для зберігання даних використовується *MongoDB* - нереляційна база даних, яка добре підходить для роботи з великими обсягами даних та забезпечує високу продуктивність. *MongoDB* дозволяє зберігати дані у вигляді документів у форматі *JSON*, що забезпечує гнучкість у моделюванні даних та швидку обробку запитів.

Клієнтська частина: Інтерфейс користувача, який дозволяє здійснювати реєстрацію, облік, пошук та модифікацію даних про заробітну плату. Клієнтська частина може бути реалізована з використанням фреймворків, таких як *React, Angular* або *Vue.js*, що забезпечує високу інтерактивність та зручність у використанні.

1.5 Забезпечення безпеки та масштабованості

Безпека та масштабованість є критично важливими аспектами для будь-якого додатку. У даному проекті використовуються наступні підходи:

Безпека: Впровадження механізмів аутентифікації та авторизації, що дозволяє забезпечити безпечний доступ до ресурсів. Використання шифрування для захисту даних під час передачі та зберігання, а також захист від основних веб-загроз (наприклад, *XSS, CSRF*) за допомогою таких інструментів, як *Helmet.js*.

Масштабованість: Використання балансування навантаження за допомогою *Nginx* або іншого веб-сервера, що дозволяє розподілити навантаження між декількома інстанціями сервера. Горизонтальне масштабування за рахунок додавання нових серверів у міру зростання кількості користувачів та обсягів даних. Оптимізація бази даних шляхом індексації, шардингу та реплікації для забезпечення високої продуктивності при великих обсягах даних.

Висновки

Аналіз *RESTful* додатку на основі технології *Node.js* показує, що ця архітектура є ефективним рішенням для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати. Використання *Node.js* забезпечує високу продуктивність, масштабованість та гнучкість, а *RESTful* архітектура дозволяє створювати зручні та надійні веб-додатки. У наступних розділах буде детально розглянуто процес розробки та впровадження даного додатку, а також проведено його тестування та оцінку ефективності.

РОЗДІЛ 2. Проектування та особливості розробки *RESTful* додатку на основі технології *Node.js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати

2.1 Вступ

Проектування та розробка *RESTful* додатку для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати вимагає детального планування, вибору відповідних технологій та інструментів, а також врахування вимог користувачів і специфіки роботи з даними. У цьому розділі розглянемо основні етапи проектування, архітектуру системи, вибір інструментів та особливості розробки додатку на основі *Node.js.*

2.2 Вимоги до системи

Перед початком проектування необхідно визначити основні функціональні та нефункціональні вимоги до системи. Основні вимоги включають:

Функціональні вимоги:

Реєстрація та авторизація користувачів: Додаток повинен підтримувати можливість створення нових користувачів та вхід в систему для вже зареєстрованих користувачів.

Створення, редагування та видалення записів про заробітну плату: Користувачі повинні мати можливість додавати нові записи про заробітну плату, редагувати існуючі записи та видаляти непотрібні.

Пошук та фільтрація записів за різними критеріями: Додаток повинен забезпечувати зручні інструменти для пошуку та фільтрації записів за параметрами, такими як ім'я працівника, дата виплати, сума виплати тощо.

Генерація звітів про заробітну плату: Система повинна надавати можливість генерувати звіти, що містять актуальну інформацію про виплати заробітної плати.

Логування дій користувачів: Всі дії користувачів мають бути зафіксовані для забезпечення прозорості та можливості аудиту.

Нефункціональні вимоги:

Висока продуктивність та масштабованість системи: Додаток повинен обробляти велику кількість запитів без втрати продуктивності та мати можливість масштабування для обробки збільшеного навантаження.

Захист даних та забезпечення безпеки: Додаток повинен відповідати сучасним вимогам безпеки для захисту даних користувачів та запобігання несанкціонованому доступу.

Інтуїтивно зрозумілий користувацький інтерфейс: Інтерфейс повинен бути зрозумілим та зручним для користувачів різного рівня підготовки.

Можливість легкого розширення функціональності: Архітектура системи повинна дозволяти легко додавати нові функції та модулі.

2.3 Архітектура системи

Архітектура *RESTful* додатку базується на принципах розподілених систем і включає такі основні компоненти:

Клієнтська частина: Відповідає за взаємодію з користувачем і відправку запитів до серверної частини. Може бути реалізована з використанням фреймворків, таких як *React, Angular* або *Vue.js*. Основні компоненти клієнтської частини:

Інтерфейс користувача (*UI*): Графічні елементи та взаємодія з користувачем.

Логіка бізнес-процесів: Управління даними та взаємодія з сервером через *API.*

Серверна частина: Виконує обробку запитів від клієнта, виконання бізнес-логіки та взаємодію з базою даних. Реалізована на основі *Node.js* з використанням фреймворку *Express.js.* Основні компоненти серверної частини:

Маршрутизатор (*Router*): Визначає маршрути для обробки *HTTP*-запитів.

Контролери (*Controllers*): Реалізують бізнес-логіку та обробку запитів.

Сервіси (*Services*): Забезпечують взаємодію з базою даних та іншими зовнішніми сервісами.

Мідлвери (*Middlewares*): Реалізують додаткові функції, такі як аутентифікація, авторизація, логування тощо.

База даних: Зберігає інформацію про заробітну плату. Використовується *MongoDB* для забезпечення гнучкості та продуктивності при роботі з великими обсягами даних. Основні компоненти:

Моделі (*Models*): Визначають структуру даних та взаємодію з базою даних.

Запити (*Queries*): Виконують операції створення, читання, оновлення та видалення даних (*CRUD*).

2.4 Вибір інструментів та технологій

Для розробки *RESTful* додатку на основі *Node.js* використовуються наступні інструменти та технології:

*Node.js:* Платформа для виконання *JavaScript*-коду на сервері, яка забезпечує високу продуктивність та неблокуючу модель введення-виведення.

*Express.js*: Фреймворк для *Node.js*, який спрощує створення *RESTful API* та управління маршрутами.

*MongoDB*: Нереляційна база даних, що забезпечує зберігання даних у форматі *JSON* та підтримує високу продуктивність.

*Mongoose: ORM (Object-Relational Mapping)* для *MongoDB*, який спрощує роботу з базою даних, забезпечуючи зручний інтерфейс для створення, читання, оновлення та видалення даних.

*Passport.js*: Бібліотека для аутентифікації в *Node.js* додатках, що підтримує різні стратегії аутентифікації (наприклад, локальну, *OAuth*).

*JWT (JSON Web Tokens)*: Стандарт для створення токенів доступу, який забезпечує безпечну передачу інформації між клієнтом і сервером.

2.5 Розробка серверної частини

Розробка серверної частини включає наступні етапи:

Налаштування середовища розробки: Встановлення *Node.js, Express.js* та інших необхідних модулів. Створення проекту та налаштування основних файлів конфігурації.

Створення основних маршрутів: Визначення маршрутів для обробки *HTTP*-запитів *(GET, POST, PUT, DELETE)* для роботи з ресурсами. Наприклад, маршрути для роботи з заробітною платою можуть включати */salaries, /salaries*/

.

Реалізація бізнес-логіки: Створення контролерів та сервісів для обробки запитів, виконання перевірок та взаємодії з базою даних. Контролери відповідають за обробку запитів та формування відповідей, а сервіси забезпечують виконання бізнес-логіки та взаємодію з базою даних.

Забезпечення безпеки: Впровадження аутентифікації та авторизації користувачів за допомогою *Passport.js*, використання *JWT* для захисту маршрутів, налаштування *CORS (Cross-Origin Resource Sharing)* для контролю доступу до ресурсів.

Тестування: Проведення модульного та інтеграційного тестування для забезпечення якості та надійності коду. Використання таких інструментів, як *Mocha, Chai* та *Supertest* для написання тестів та перевірки функціональності.

2.6 Розробка клієнтської частини

Розробка клієнтської частини включає:

Дизайн інтерфейсу користувача: Розробка макетів та прототипів інтерфейсу, що забезпечують зручність використання та інтуїтивну зрозумілість. Використання інструментів для дизайну, таких як *Figma* або *Adobe XD.*

Реалізація користувацького інтерфейсу: Використання фреймворків, таких як *React, Angular або Vue.js* для створення динамічних та інтерактивних веб-сторінок. Реалізація компонентів інтерфейсу, управління станом додатку за допомогою таких бібліотек, як *Redux або Vuex*.

Взаємодія з сервером: Реалізація функціоналу для відправки запитів до серверної частини (наприклад, за допомогою *Axios або Fetch API*) та обробки відповідей. Впровадження механізмів аутентифікації та авторизації на клієнтській стороні.

Тестування та відладка: Проведення тестування інтерфейсу для виявлення та виправлення помилок, а також оптимізація продуктивності. Використання таких інструментів, як *Jest, Enzyme або Cypress* для автоматизованого тестування компонентів та інтеграційного тестування.

2.7 Забезпечення безпеки та масштабованості

Безпека та масштабованість є важливими аспектами для успішної роботи додатку. Для забезпечення безпеки використовуються наступні підходи:

Аутентифікація та авторизація: Використання *Passport.js* для аутентифікації користувачів та *JWT* для захисту маршрутів. Це забезпечує надійний захист від несанкціонованого доступу до ресурсів додатку.

Шифрування даних: Використання *HTTPS* для захисту даних під час передачі між клієнтом і сервером, а також шифрування даних у базі даних для запобігання несанкціонованому доступу до чутливої інформації.

Захист від веб-загроз: Впровадження механізмів захисту від *XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery)* та інших загроз за допомогою таких інструментів, як *Helmet*.*js*. Налаштування політики безпеки контенту *(Content Security Policy)* для запобігання атак.

Для забезпечення масштабованості додатку використовуються такі підходи:

Балансування навантаження: Використання *Nginx* або іншого веб-сервера для розподілу навантаження між декількома інстанціями сервера. Це дозволяє забезпечити високу продуктивність та надійність роботи додатку при збільшенні кількості користувачів.

Горизонтальне масштабування: Додавання нових серверів у міру зростання кількості користувачів та обсягів даних. Це дозволяє ефективно обробляти збільшене навантаження на систему.

Оптимізація бази даних: Використання індексації, шардингу та реплікації для підвищення продуктивності *MongoDB* при роботі з великими обсягами даних. Це забезпечує швидкий доступ до даних та зменшення часу на обробку запитів.

Висновки

Проектування та розробка *RESTful* додатку на основі технології *Node.js* для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати включає детальний аналіз вимог, вибір відповідних інструментів та технологій, а також врахування аспектів безпеки та масштабованості. У наступних розділах буде розглянуто процес тестування додатку, оцінка його ефективності та впровадження у реальних умовах.

РОЗДІЛ 3. Проектування та розробка програмного модуля

3.1 Вибір архітектури в відповідності до шаблону *MVC*

Архітектура *Model-View-Controller (MVC*) є однією з найпопулярніших архітектурних моделей для розробки веб-додатків. Вона розділяє додаток на три основні компоненти:

*Model* (Модель): Представляє дані додатку та логіку роботи з ними. Модель відповідає за зберігання, обробку та перевірку даних.

*View* (Вид): Відповідає за відображення даних користувачу. Вид отримує дані з моделі та відображає їх у зрозумілому для користувача форматі.

*Controller* (Контролер): Обробляє запити користувачів, взаємодіє з моделлю та обирає відповідний вид для відображення результатів.

Використання архітектури *MVC* дозволяє розділити відповідальності між різними компонентами додатку, що спрощує його розробку, тестування та підтримку.

3.2 Розробка складових

Розробка програмного модуля для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати включає створення моделей, контролерів та видів. Розглянемо кожен компонент детальніше.

Моделі (*Models*)

Моделі відповідають за зберігання даних та взаємодію з базою даних. У нашому додатку використовуються наступні моделі:

*User* (Користувач): Зберігає інформацію про користувачів системи (ім'я, email, пароль, роль).

*Salary* (Заробітна плата): Зберігає інформацію про заробітні плати користувачів (сума, дата виплати, користувач).

*Department* (Відділ): Зберігає інформацію про відділи (назва, список працівників).

Контролери (*Controllers*)

Контролери відповідають за обробку запитів користувачів та взаємодію з моделями. У нашому додатку використовуються наступні контролери:

*AuthController* (Контролер аутентифікації): Обробляє запити на реєстрацію, логін та вихід з системи.

*SalaryController* (Контролер заробітної плати): Обробляє запити на створення, редагування та видалення записів про заробітну плату.

*UserController* (Контролер користувачів): Обробляє запити на створення, редагування та видалення користувачів.

Види (*Views*)

Види відповідають за відображення даних користувачам. Вони отримують дані від контролерів та моделі і відображають їх у зручному для користувача форматі. У нашому додатку використовуються шаблони *EJS (Embedded JavaScript).*

При запуску нашого додатку користувачу буде запропоновано зареєструватися або ж за наявності вже існуючого акаунту здійснити вхід. Дане вікно вибору продемонстровано на рисунку 1.1.

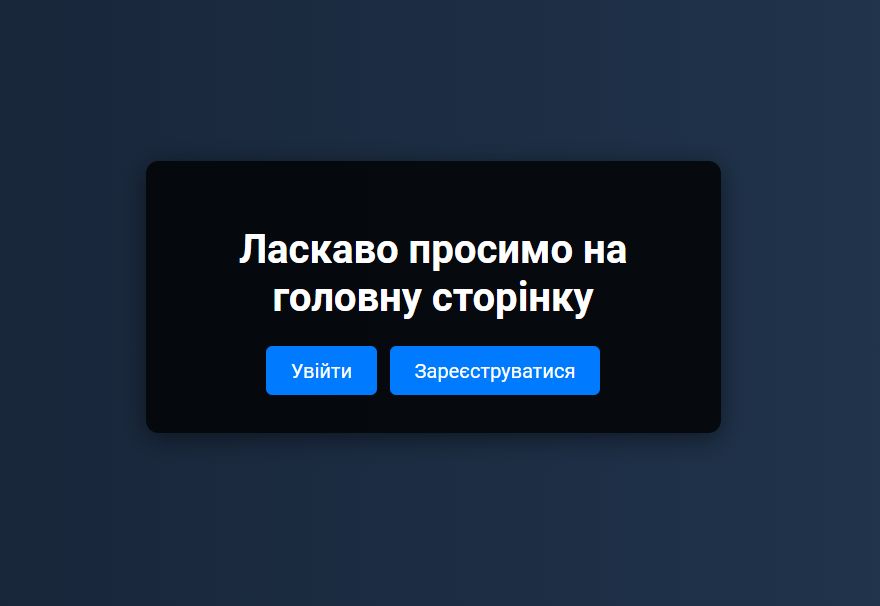
,

Рисунок 1.1.Меню входу

При виборі пункту “Реєстрація” користувачу відкриється меню в якому він зможе ввести свої дані для подальшого користування нашим додатком рис. 1.2. Користувачу будуть запропоновані наступні пункти для заповнення:

-Корстувач (потрібно вказати ім’я користувача)

-Електронну пошту

-Пароль (пароль буде використовуватися при наступному вході в обліковий запис)

-Роль (потрібно обрати одну з запропонованих ролей, користувач або адмін)

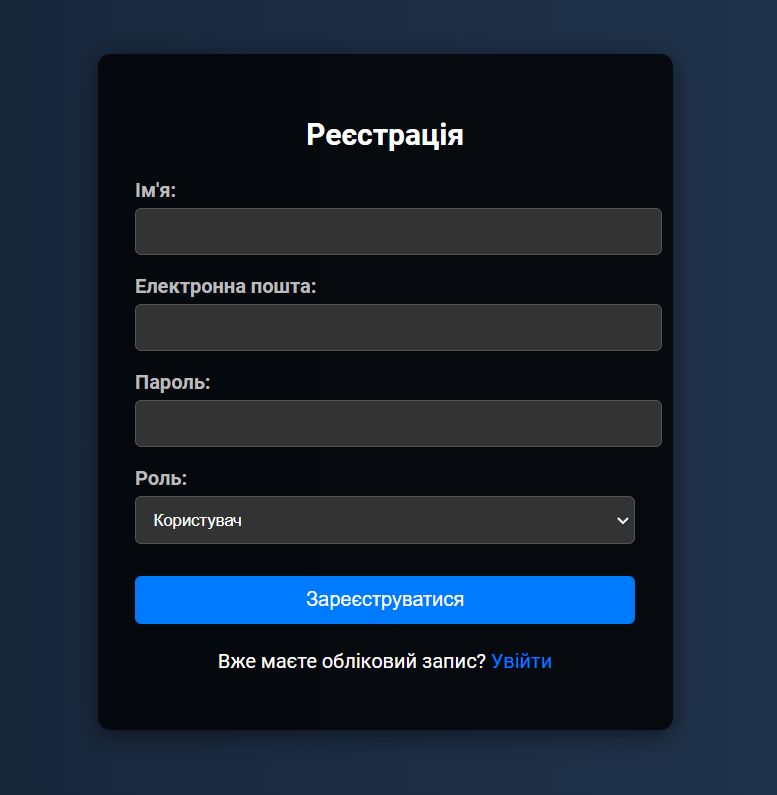


Рисунок 1.2. Реєстрація користувача

Якщо користувач вже зареєстрований, йому слід обрати пункт “Увійти” рис 1.3. В даному меню потрібно вже зареєстрованому користувачеві ввести свої дані після чого його перенесе на основну сторінку додатку рис.1.4.

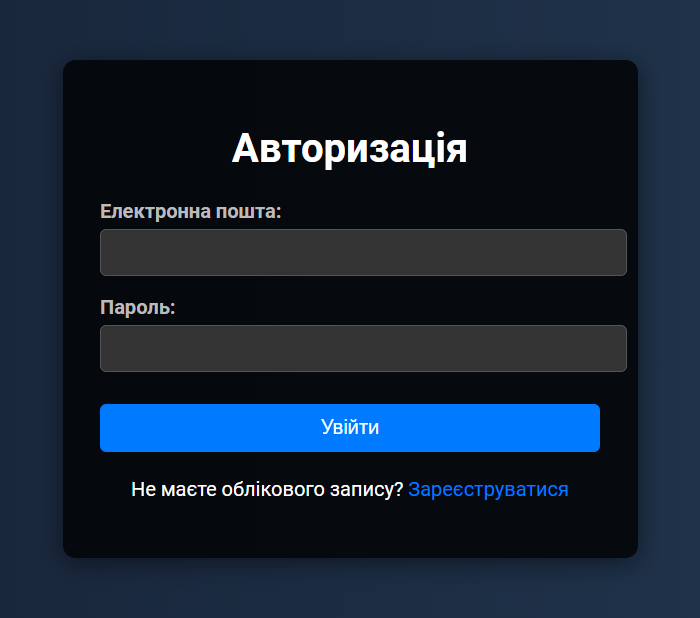


Рисунок 1.3. Авторизація користувача

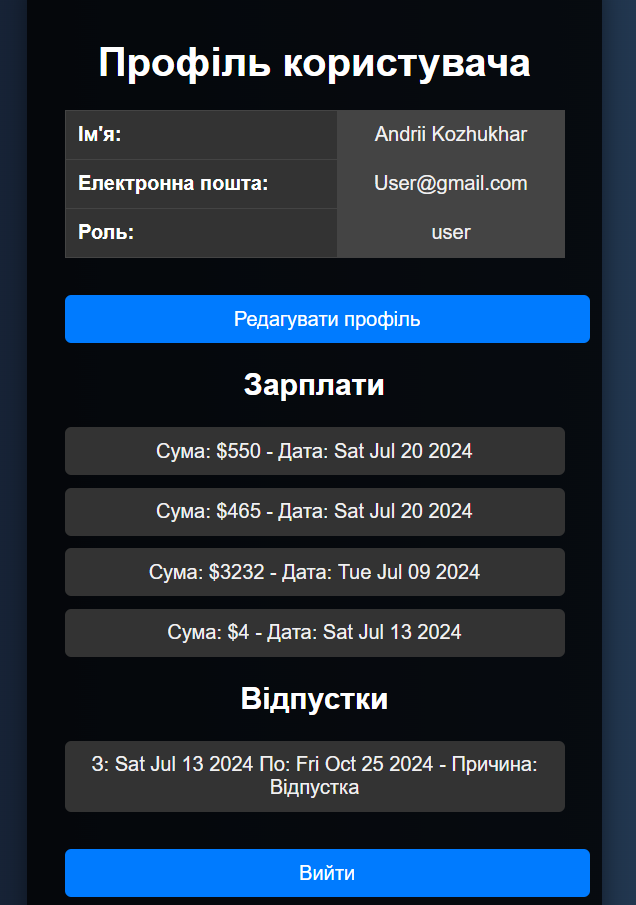


Рисунок 1.4. Головна сторінка користувача

головній сторінці користувачу буде доступний перегляд заробітної плати та відпустки. Також користувач може редагувати особисті дані рис. 1.5 .

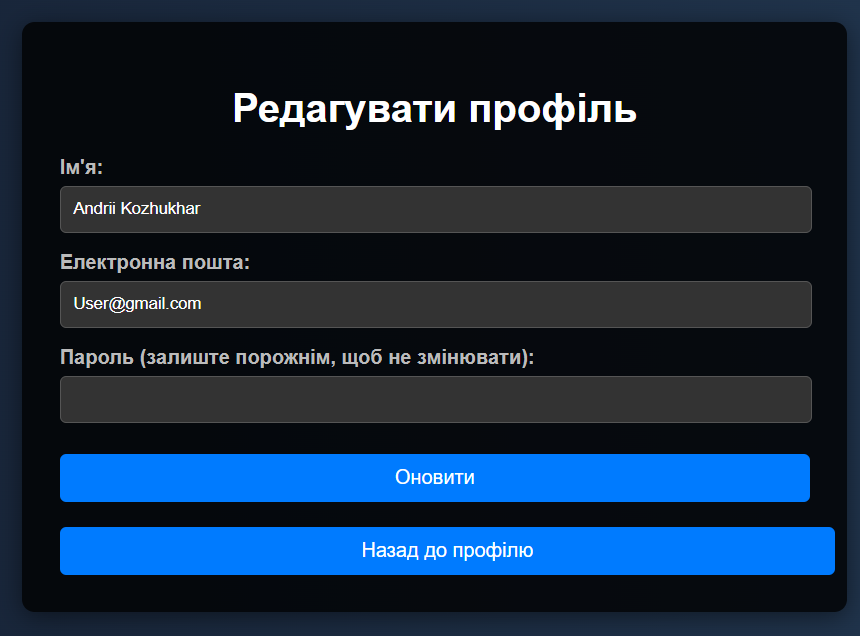


Рисунок 1.5. редагування особистих даних користувача

При реєстрації акаунта адміністратора , потрібно вибрати роль «Адміністратор», після чого з’явиться вікно для пароля адміністратора , який надасть змогу зареєструватися . Пароль для реєстрації адмістратора –«super22admin». Реєстрацію адміна зображено на рисунку 1.6 .

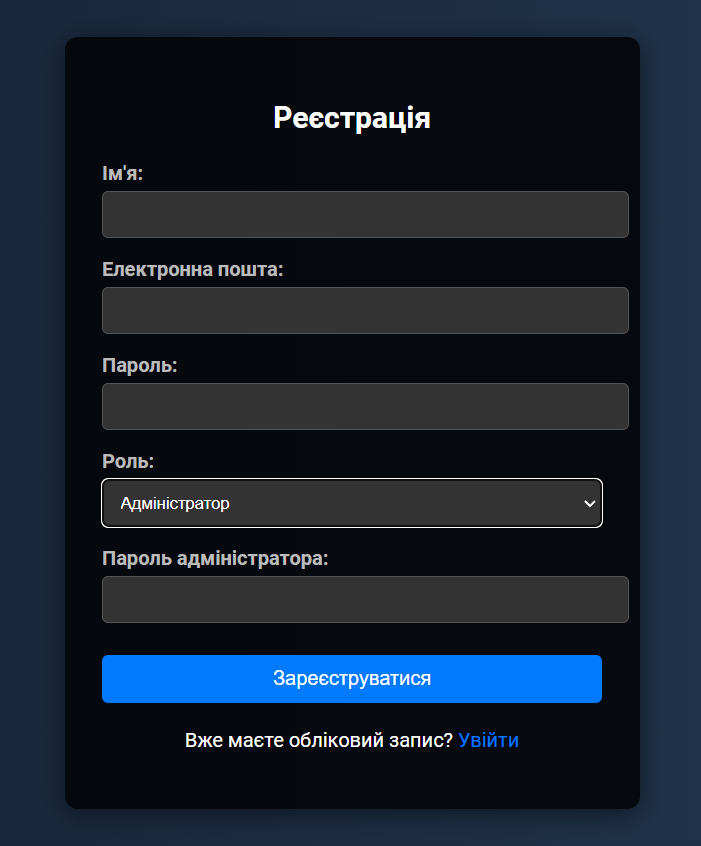


Рисунок 1.6. Реєстрація Адміністратора

Авторизація для адміністратора здійснюється ідентично як і для юзера.Після ходу в акаунт адміністратора буде зображена «Панель адміністратора» рисунок 1.7.

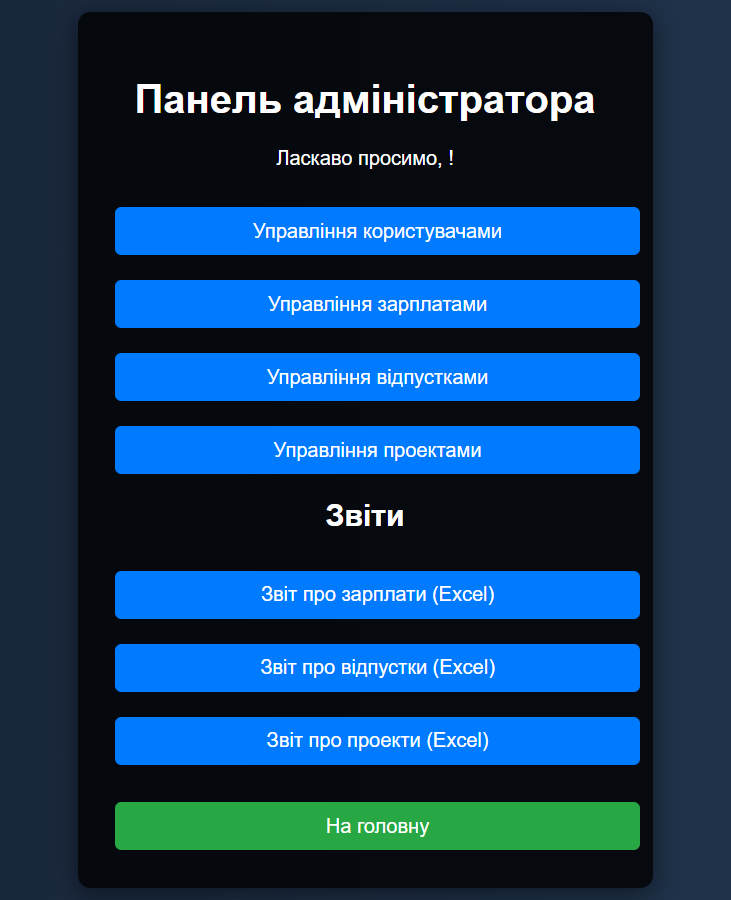


Рисунок 1.7. Панель адміністратора

Адміністративна панель, зображена на малюнку, надає інтерфейс для управління ключовими аспектами діяльності бухгалтерії інституту з обліку заробітної плати. Інтерфейс організовано таким чином, щоб забезпечити зручний доступ до основних функцій системи. Панель складається з кількох основних розділів, кожен з яких відповідає за конкретний набір функціональностей.

1. Управління користувачами

У розділі "Управління користувачами" адміністратор має можливість:

Створювати нових користувачів: ця функція дозволяє додавати нових працівників до системи, заповнюючи необхідні поля, такі як ім'я, електронна пошта, роль та інші.

Редагувати існуючих користувачів: адміністратор може змінювати інформацію про користувачів, включаючи їхні облікові дані та інші важливі атрибути.

Видаляти користувачів: функція видалення дозволяє усувати з системи користувачів, які більше не є активними.

2. Управління зарплатами

Розділ "Управління зарплатами" забезпечує такі можливості:

Надання заробітної плати користувачам: адміністратор може додавати записи про заробітну плату, вказуючи суму та інші деталі.

Редагування записів про заробітну плату: дозволяє оновлювати інформацію про виплати, якщо виникає потреба у змінах.

Видалення записів про заробітну плату: функція видалення записів дозволяє знімати некоректні або застарілі дані.

Фільтрація та пошук: система дозволяє фільтрувати записи за датою та проміжком часу, а також здійснювати пошук за ім'ям користувача, що спрощує управління великими обсягами даних.

3. Управління відпустками

Розділ "Управління відпустками" включає наступні функції:

Надання відпустки користувачам: адміністратор може додавати записи про відпустку для певних користувачів, зазначаючи відповідні дати та причини.

Редагування записів про відпустки: функція редагування дозволяє вносити зміни до існуючих записів про відпустки.

Видалення записів про відпустки: видалення записів про відпустки з системи, що більше не актуальні.

Пошук за ім'ям користувача: адміністратор може здійснювати пошук відпусток за ім'ям користувача, що дозволяє швидко знайти потрібні дані.

4. Управління проектами

У розділі "Управління проектами" адміністратор може:

Створювати нові проекти: введення назви, опису проекту, проміжку часу для виконання та призначення відповідальної особи.

Редагувати існуючі проекти: можливість оновлювати деталі проекту відповідно до потреб.

Видаляти проекти: видалення проектів, які більше не актуальні або завершені.

Реалізація операцій *CRUD* *(Create, Read, Update, Delete):* забезпечує повний спектр операцій з управління проектами.

Автоматичні звіти

Система автоматично генерує звіти, які експортуються у формат Excel, що є дуже корисним для подання звітності відповідним особам. Ці звіти включають дані про зарплати, відпустки та проекти.

Висновки

Адміністративна панель надає потужний інструмент для управління ключовими аспектами діяльності бухгалтерії інституту з обліку заробітної плати. Завдяки зручному інтерфейсу та широкому набору функцій, адміністратор може ефективно управляти даними, забезпечуючи високу точність та продуктивність обліку.

ВИСНОВОК

Розробка *RESTful* додатку для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати на основі технології *Node.js* є складним, але дуже ефективним процесом, що включає багато етапів, від аналізу вимог до забезпечення безпеки та масштабованості. Аналіз різних аспектів проектування та розробки дозволяє зробити декілька важливих висновків.

По-перше, *RESTful* архітектура є відмінним вибором для створення веб-додатків завдяки своїй простоті, масштабованості та гнучкості. Використання стандартних *HTTP* методів та структурованих *URL* забезпечує зручність у використанні та розширенні додатку.

По-друге, *Node.js* є потужною платформою для серверної розробки, що забезпечує високу продуктивність та ефективність завдяки своїй подіє-орієнтованій, неблокуючій моделі введення-виведення. Використання однієї мови програмування (*JavaScript*) для клієнтської та серверної частини спрощує процес розробки та обслуговування додатку.

По-третє, архітектура додатку включає клієнтську та серверну частини, а також базу даних. Кожен компонент виконує свою роль у забезпеченні функціональності, продуктивності та надійності системи.

По-четверте, використання сучасних інструментів та технологій, таких як *Express.js, MongoDB, Mongoose, Passport.js та JWT*, дозволяє створити надійний, безпечний та масштабований додаток. Ці технології забезпечують ефективну обробку даних, безпечну аутентифікацію та авторизацію користувачів, а також зручний інтерфейс для взаємодії з додатком.

По-п'яте, забезпечення безпеки та масштабованості є критично важливими аспектами для будь-якого веб-додатку. Використання механізмів аутентифікації та авторизації, шифрування даних, захист від веб-загроз, а також балансування навантаження та оптимізація бази даних дозволяють створити надійну та стійку до навантажень систему.

Нарешті, інтуїтивно зрозумілий та зручний інтерфейс користувача є ключовим фактором для успішного впровадження та використання додатку. Використання сучасних фреймворків для створення динамічних та інтерактивних веб-сторінок забезпечує високу якість взаємодії з користувачами.

У результаті виконаної роботи створено ефективний інструмент для автоматизації діяльності бухгалтерії інституту з обліку заробітної плати, який автоматизує процеси управління ресурсами, підвищуючи їх точність та надійність. Запропоновані рішення можуть бути успішно впроваджені в освітніх закладах для покращення управління фінансовими ресурсами та підвищення загальної ефективності діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js API Documentation: Повний посібник та документація API для Node.js. Доступно на: https://nodejs.org/en/docs/
2. Express.js Guide: Офіційна документація для фреймворку Express.js. Доступно на: https://expressjs.com/
3. EJS Templating: Офіційна документація для EJS, шаблонізатора для JavaScript. Доступно на: https://ejs.co/
4. Visual Studio Code Docs: Офіційна документація для Visual Studio Code, IDE для розробки коду. Доступно на: https://code.visualstudio.com/docs
5. Mozilla Developer Network (MDN): Документація та навчальні матеріали з веб-розробки від Mozilla. Доступно на: https://developer.mozilla.org/
6. Stack Overflow: Спільнота розробників для вирішення проблем та обговорення питань з програмування. Доступно на: https://stackoverflow.com/
7. TutorialsPoint: Онлайн ресурси для навчання програмуванню та різним технологіям. Доступно на: https://www.tutorialspoint.com/
8. Coursera: Онлайн курси з програмування та розробки додатків. Доступно на: https://www.coursera.org/
9. MongoDB Documentation: Офіційна документація для MongoDB, NoSQL бази даних. Доступно на: https://docs.mongodb.com/
10. Mongoose Documentation: Документація для Mongoose, бібліотеки ODM (Object Data Modeling) для MongoDB та Node.js. Доступно на: https://mongoosejs.com/docs/
11. Passport.js Documentation: Офіційна документація для Passport.js, Node.js middleware для аутентифікації. Доступно на: http://www.passportjs.org/docs/
12. JSON Web Tokens (JWT) Documentation: Офіційна документація для JWT, стандарту для створення токенів доступу. Доступно на: https://jwt.io/introduction/

ДОДАТОК А

*APP.JS*

*const express = require('express');*

*const mongoose = require('mongoose');*

*const cookieParser = require('cookie-parser');*

*const bodyParser = require('body-parser');*

*require('dotenv').config();*

*const connectDB = require('./config/database');*

*const authRoutes = require('./routes/auth');*

*const adminRoutes = require('./routes/admin');*

*const profileRoutes = require('./routes/profile');*

*const reportRoutes = require('./routes/report');*

*const app = express();*

*connectDB(); // Підключення до бази даних*

*app.set('view engine', 'ejs');*

*app.set('views', \_\_dirname + '/views');*

*app.use(express.static('public'));*

*app.use('/fonts', express.static(\_\_dirname + '/fonts')); // Додаємо статичний маршрут для шрифтів*

*app.use(bodyParser.urlencoded({ extended: true }));*

*app.use(bodyParser.json());*

*app.use(cookieParser());*

*// Підключення маршрутів*

*app.use('/auth', authRoutes);*

*app.use('/admin', adminRoutes);*

*app.use('/profile', profileRoutes);*

*app.use('/reports', reportRoutes); // Додаємо маршрут для звітів*

*// Головна сторінка*

*app.get('/', (req, res) => {*

*res.render('pages/index');*

*});*

*// Запуск сервера*

*const PORT = process.env.PORT || 3000;*

*app.listen(PORT, () => {*

*console.log(`Сервер запущено на порту ${PORT}`);*

*});*