НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

Навчально-науковий інститут автоматики та електротехніки
(повне найменування інституту, факультету)

Кафедра комп'ютеризованих систем управління
(повна назва кафедри)

«Допущений до захисту»
Завідувач кафедри
О.О. Черно
«20» 12 2024 р.

Пояснювальна записка

до кваліфікаційної роботи

на здобуття другого (магістерського) рівня вищої освіти

за спеціальністю 174– «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»
ОПП - «Інтелектуальні комп'ютерні системи управління»

на тему: ІНТЕЛЕКТУАЛЬНА СИСТЕМА КОНТРОЛЮ ДОСТУПУ ЦЕНТРУ УПРАВЛІННЯ АВТОНОМНИМИ МОРСЬКИМИ ОБ'ЄКТАМИ

Виконав студент **6** курсу, **6342м** групи
Білик О.О.
(прізвище та ініціали)

Керівник
Кудін О.О.
(прізвище та ініціали)

Миколаїв – 2024

Національний університет кораблебудування імені адмірала Макарова

Інститут автоматики і електротехніки
Кафедра комп'ютеризованих систем управління
Освітньо-кваліфікаційний рівень **«магістр»**

Спеціальність **174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»**
Освітня програма **«Інтелектуальні комп'ютерні системи управління»**

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
_____ Черно О.О.
«15» 10 2024 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Білику Олегу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи: *Інтелектуальна система контролю доступу центру управління автономними морськими об'єктами*

Керівник роботи *Кудін Олег Олексійович, к.т.н.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "14" 10 2024р. № 1075-уч

2. Строк подання роботи 16 грудня 2024 р.
3. Вихідні дані до роботи *Система контролю доступу повинна використовувати універсальну систему управління базами даних (СУБД), яку можна інсталювати на операційні системи Microsoft Windows та Linux*

4. Зміст розрахунково-пояснювальної записки (основні задачі дослідження)
Аналіз існуючих систем контроля доступу до центрів управління автономними морськими об'єктами
Вибір системи управління базами даних. Концептуальне, логічне та фізичне проектування бази даних.
Розробка Web-інтерфейсу адміністратора системи контролю доступу.
Охорона праці

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Аналіз існуючих систем контроля доступу до центрів управління автономними морськими об'єктами

Концептуальне, логічне та фізичне проектування бази даних. ER-діаграма
Розробка Web-інтерфейсу адміністратора системи контролю доступу.
Охорона праці

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Охорона праці	*Кудін О.О., доц. каф. КСУ*	05.09.2024	21.10.2024

Дата видачі завдання: «__5__»____09____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проектування	Строк виконання	Примітка
1	*Аналіз існуючих систем контроля доступу до центрів управління автономними морськими об'єктами*	*10.10.2024*	*Виконано*
2	*Вибір системи управління базами даних. Концептуальне, логічне та фізичне проектування бази даних.*	*04.11.2024*	*Виконано*
3	*Розробка Web-інтерфейсу адміністратора системи контролю доступу.*	*22.11.2024*	*Виконано*
4	*Охорона праці*	*09.12.2024*	*Виконано*

Студент _____ Білик О.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Кудін О.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній роботі розглянуто створення інтелектуальної системи контролю доступу для центру управління автономними морськими об'єктами. Основною метою дослідження є забезпечення безпеки та надійності доступу до критично важливих ресурсів системи управління. Запропонована система контролю доступу базується на використанні багатофакторної аутентифікації, яка поєднує різні методи ідентифікації користувачів для забезпечення захищеного доступу до даних і систем управління.

Основні наукові положення полягають у розробці централізованої бази даних для зберігання інформації про ролі та доступи користувачів, а також у впровадженні багатошарової системи моніторингу дій користувачів. Система контролю доступу реалізує різні рівні аутентифікації, що дозволяє оперативно реагувати на несанкціоновані спроби доступу і забезпечити високий рівень кібербезпеки.

Практична значимість роботи полягає у створенні інструментів для централізованого управління доступом до ресурсів центру обробки даних, що забезпечує надійний контроль і зменшує ризики несанкціонованого втручання. Запропонований підхід дозволяє підвищити рівень безпеки автономних морських об'єктів, зберігаючи стабільність їх роботи в умовах обмеженого фізичного доступу та зростаючих загроз кібербезпеки.

ABSTRACT

This study focuses on the development of an intelligent access control system for the management center of autonomous maritime objects. The primary goal of the research is to ensure the security and reliability of access to the critical resources of the management system. The proposed access control system is based on the implementation of multi-factor authentication, combining various user identification methods to provide secure access to data and management systems.

The main scientific contributions include the design of a centralized database for storing information on user roles and access permissions, as well as the implementation of a multi-layered user activity monitoring system. The access control system incorporates multiple levels of authentication, enabling rapid response to unauthorized access attempts and ensuring a high level of cybersecurity.

The practical significance of this work lies in the development of tools for centralized access management to the resources of the data processing center, providing reliable control and reducing the risks of unauthorized interference. The proposed approach enhances the security of autonomous maritime objects, maintaining their operational stability in conditions of limited physical access and increasing cybersecurity threats.

ПЕРЕЛІК ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

MASS (Maritime Autonomous Surface Ships) – морські автономні поверхневі судна.

PID (Proportional-Integral-Derivative Control) – пропорційно-інтегрально-диференційний контроль.

IoT (Internet of Things) – Інтернет речей.

AIS (Automatic Identification System) – автоматична ідентифікаційна система.

GNSS (Global Navigation Satellite System) – глобальна навігаційна супутникова система.

AOC (Ashore Operation Centers) – берегові операційні центри.

UI (User interface) – Інтерфейс користувача

ШІ – штучний інтелект

СКУД (система контролю та управління доступом) – система, що забезпечує контроль доступу до об'єктів або систем.

ЦОД (центр обробки даних) – центр обробки даних для зберігання та обробки інформації.

СОД – Система обробки даних

СУБД – Системи управління базами даних

ЗМІСТ

ВСТУП

Розвиток автономних морських суден є одним із ключових напрямів сучасної морської індустрії, що спрямований на підвищення ефективності, безпеки та стабільності морських перевезень. Технологічні інновації, які включають інтеграцію штучного інтелекту, автоматизацію процесів та впровадження інтелектуальних систем контролю доступу, сприяють еволюції морського транспорту. Автономні судна обіцяють змінити підходи до логістики, зменшити вплив людського фактора та знизити експлуатаційні витрати. У зв'язку із зростанням кількості морських перевезень та необхідністю оптимізації ресурсів, інтелектуальні системи управління стають життєво важливими компонентами для забезпечення високого рівня функціональності та безпеки автономних суден.

Інтелектуальні системи контролю доступу до центрів управління автономними морськими об'єктами дозволяють забезпечити надійність, захист від кіберзагроз і стійкість у складних умовах. У даній роботі розглядаються особливості впровадження таких систем, які дозволяють інтегрувати різні рівні аутентифікації та багатофакторний підхід для зниження ризиків несанкціонованого доступу. Сучасні підходи до кібербезпеки передбачають багатошаровий захист, де кожен рівень включає певний набір механізмів моніторингу, аутентифікації та аналізу дій користувачів у реальному часі. Це сприяє оперативній реакції на потенційні загрози та забезпечує надійність роботи всієї системи управління.

Автономні судна функціонують у середовищі з обмеженим або навіть повністю відсутнім фізичним доступом, що робить питання кібербезпеки ще більш актуальним. Інтелектуальні системи контролю доступу здатні запобігти несанкціонованим втручанням, гарантуючи, що тільки авторизовані користувачі мають можливість доступу до критично важливих даних та функцій системи управління. Впровадження таких систем базується на використанні передових методів штучного інтелекту та алгоритмів машинного навчання, які дозволяють

аналізувати величезні обсяги даних для виявлення аномальних дій та попередження загроз.

Значну увагу приділено багатофакторній аутентифікації, яка об'єднує різні методи ідентифікації, такі як паролі, біометричні дані та фізичні пристрої, що підвищує загальний рівень захисту. Це особливо важливо в умовах автономного функціонування суден, коли кіберзагрози можуть мати серйозні наслідки для безпеки всього об'єкта та його оточення. Підходи, що використовуються в сучасних системах контролю доступу, забезпечують не тільки ефективний захист від зловмисних атак, а й підтримку стабільної роботи автономних морських суден навіть у разі виникнення зовнішніх загроз або технічних збоїв.

Актуальність дослідження обумовлена стрімким розвитком технологій автономного судноплавства, а також необхідністю забезпечення високого рівня кібербезпеки та захисту від зовнішніх загроз. Інтелектуальні системи контролю доступу не тільки підвищують безпеку суден, але й сприяють їхній автономії, роблячи їх незалежними від людського фактора у більшості рішень, пов'язаних із навігацією та управлінням. Це має особливе значення для сучасних умов, коли кількість кіберзагроз постійно зростає, а морська галузь стає дедалі більш залежною від автоматизованих рішень.

Крім питань безпеки, інтелектуальні системи управління відіграють важливу роль у підвищенні ефективності логістичних процесів. Оптимізація маршрутів, моніторинг стану судна, прогнозування технічних проблем та автоматичне коригування операцій дозволяють значно знизити витрати та підвищити рентабельність морських перевезень. Використання технологій штучного інтелекту в управлінні судном забезпечує можливість адаптації до мінливих умов навколишнього середовища, знижує ризики, пов'язані з людськими помилками, та забезпечує стабільність роботи в умовах, які можуть бути непередбачуваними або небезпечними.

Таким чином, ця робота спрямована на дослідження й удосконалення підходів до створення інтелектуальних систем контролю доступу для

автономних морських суден. Вона є важливим внеском у забезпечення безпеки та надійності морських перевезень на міжнародному рівні. Інтеграція інтелектуальних систем контролю доступу дозволяє не тільки підвищити рівень безпеки суден, але й зменшити ризики, пов'язані з кіберзагрозами, забезпечити стабільність роботи морських транспортних об'єктів у сучасному цифровому середовищі, а також сприяти подальшому розвитку морської індустрії в умовах зростаючої автоматизації та технологічної інтеграції.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ СИСТЕМ КОНТРОЛЯ ДОСТУПУ ДО ЦЕНТРІВ УПРАВЛІННЯ АВТОНОМНИМИ МОРСЬКИМИ ОБ'ЄКТАМИ

1.1 Інтелектуальні системи управління та автоматизація автономних суден

Інтелектуальні системи управління та автоматизація автономних суден набувають дедалі більшого значення у сфері морського транспорту. Сучасний розвиток технологій значно змінює підходи до організації морських перевезень, інтегруючи інновації, такі як геопросторові технології (GIS), штучний інтелект (ШІ), інтелектуальні датчики та системи автоматизації. Вони дозволяють забезпечити не лише оптимізацію маршрутів суден, але й значно підвищити ефективність і надійність управління, зменшуючи залежність від людського фактора та мінімізуючи ризики помилок під час навігації, особливо у складних або небезпечних умовах.

Основна частина дослідження зосереджена на використанні елементів штучного інтелекту для забезпечення регулювання та обмеження прав доступу співробітників центру управління автономними морськими суднами до інформації з обмеженим доступом. Такі системи забезпечують вищу точність навігації та зменшують залежність від людського фактора, мінімізуючи ризики помилок під час навігації. Це особливо важливо у випадку роботи в складних або небезпечних умовах, де традиційне управління може не забезпечити належної ефективності та безпеки. Автономні системи дозволяють суднам самостійно адаптуватися до умов навколишнього середовища та швидко коригувати маршрут за допомогою аналізу даних із датчиків і супутників.

Значна увага приділяється проблемам кібербезпеки, оскільки автономні судна значною мірою залежать від комунікаційних систем та зберігання даних. Потенційні кібератаки, маніпуляції даними або порушення комунікацій можуть призвести до катастрофічних наслідків. Автори підкреслюють необхідність

створення надійних механізмів захисту, таких як шифрування та постійний моніторинг для забезпечення безпеки операцій автономних суден.

Ще одним важливим аспектом є розробка стандартів та регуляторних норм для автономних суден. Оскільки більшість морських регуляцій розроблено для суден із екіпажем, необхідно адаптувати міжнародні правила, щоб вони відповідали новим технологіям та забезпечували безпечне функціонування безпілотних суден.

У статті [1] також пропонується використання берегових операційних центрів (Ashore Operation Centers, AOC), які будуть центральним вузлом для моніторингу та керування автономними суднами в реальному часі. Така централізація не лише підвищує ефективність управління, але й забезпечує кращу координацію морських логістичних операцій. Це дозволяє швидше реагувати на можливі проблеми, коригувати маршрути та уникати затримок у портах. У технологічному плані автономні судна стикаються з кількома основними викликами. Серед них – необхідність забезпечення точності сенсорів для навігації, зокрема, таких як GNSS, AIS, LiDAR, та інші. Ці датчики повинні функціонувати в умовах складних погодних умов, високих хвиль або низької видимості, щоб забезпечити безпечне уникнення зіткнень. Крім того, системи навігації повинні бути здатні обробляти великі обсяги даних у реальному часі для прийняття точних рішень.

Система управління судном також має бути здатною адаптуватися до зміни умов на морі, і це завдання вирішується шляхом використання нейронних мереж та алгоритмів глибокого навчання, які забезпечують швидке аналізування даних і корекцію курсу. Такі алгоритми дозволяють суднам навчатися на базі минулих маршрутів і удосконалювати свої навігаційні навички.

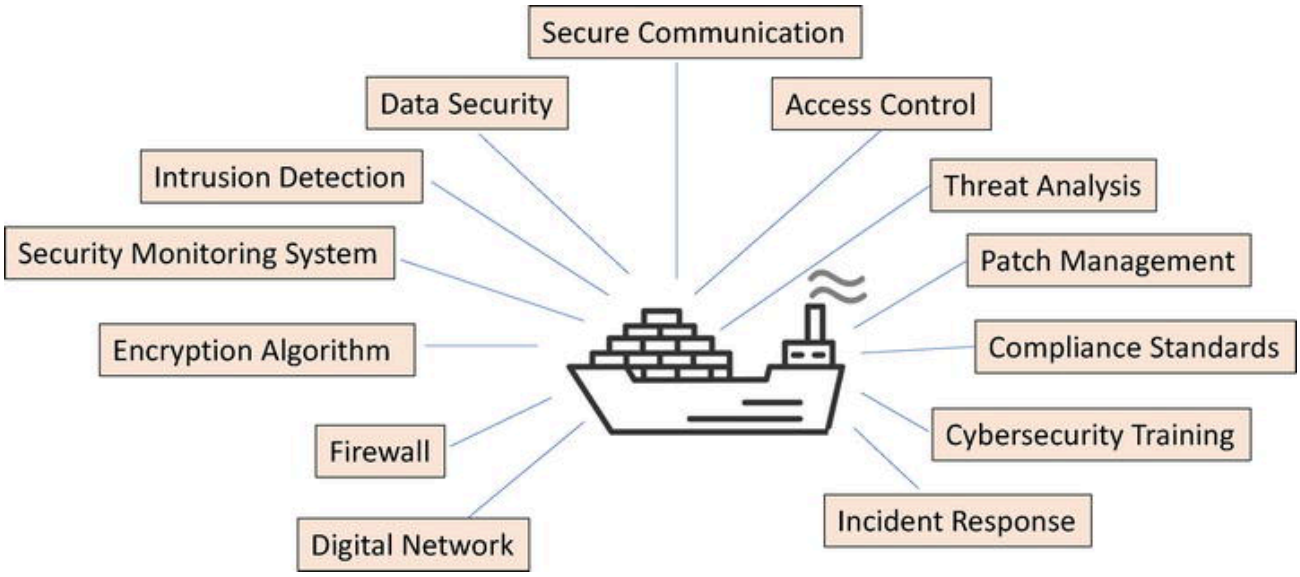

Рисунок 1.1 - Ілюстрація заходів кібербезпеки для систем автономного руху суден. [12]

- Secure Communication → Безпечний зв’язок
- Data Security → Безпека даних
- Access Control → Контроль доступу
- Intrusion Detection → Виявлення вторгнень
- Threat Analysis → Аналіз загроз
- Security Monitoring System → Система моніторингу безпеки
- Patch Management → Управління оновленнями
- Encryption Algorithm → Алгоритм шифрування
- Compliance Standards → Стандарти відповідності
- Firewall → Міжмережевий екран
- Cybersecurity Training → Навчання з кібербезпеки
- Digital Network → Цифрова мережа
- Incident Response → Реагування на інциденти

Морська галузь знаходиться на етапі значного технологічного прориву завдяки інтеграції автономних систем і штучного інтелекту. Проте для досягнення повної автономії суден потрібно подолати низку викликів, включаючи забезпечення кібербезпеки, регулювання та точність навігаційних

систем. Це підкреслює необхідність міжнародної співпраці та стандартизації для безпечної та ефективної роботи автономних суден у глобальному масштабі. Водночас, дослідження, такі як "State-of-the-Art Research on Motion Control of Maritime Autonomous Surface Ships," [2] підкреслюють важливість розробки систем автономного управління суднами, які функціонують без постійного втручання людини. Це дослідження охоплює ключові аспекти автономного управління морськими автономними поверхневими суднами (MASS), що стають важливим напрямом розвитку морського транспорту в умовах зростання обсягів перевезень і вимог до безпеки.

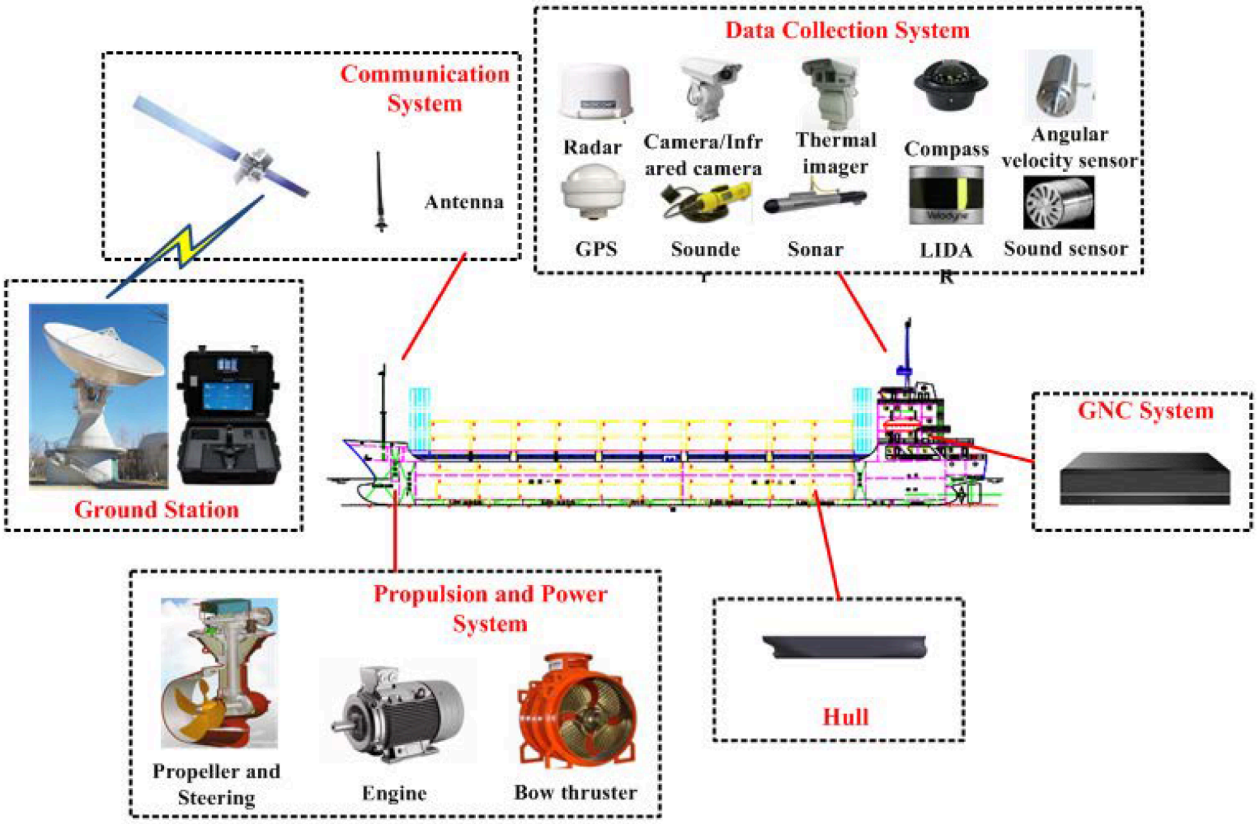

Рисунок 1.2 - Приклад системи MASS [5]

- MASS (Maritime Autonomous Surface Ships) – морські автономні поверхневі судна.
- Communication System → Система зв’язку
  - Antenna → Антена
- Data Collection System → Система збору даних

    - Radar → Радар
    - Camera/Infrared camera → Камера/Інфрачервона камера
    - Thermal imager → Тепловізор
    - Compass → Компас
    - Angular velocity sensor → Датчик кутової швидкості
    - GPS → GPS
    - Sounder → Ехолот
    - Sonar → Гідролокатор
    - LIDAR → Лідар
- Sound sensor → Акустичний датчик
- Ground Station → Наземна станція
- Propulsion and Power System → Система рушіїв та живлення
- Propeller and Steering → Гвинт і рульове управління
    - Engine → Двигун
    - Bow thruster → Носовий підрулювач
- GNC System → Система навігації, управління та контролю
- Hull → Корпус

У статті [3] розглядається широкий спектр технологій та методів, що застосовуються для контролю руху MASS. Зокрема, значна увага приділяється алгоритмам, які забезпечують автономну навігацію та коригування курсу без участі оператора. Це стосується таких функцій, як контроль швидкості, маневрування, уникнення перешкод та стабілізація траєкторії. Алгоритми, що використовуються для цих завдань, включають пропорційно-інтегрально-диференційний (PID) контроль, нечітку логіку, передбачувальні алгоритми та адаптивні контролери. Всі ці методи спрямовані на забезпечення стабільності руху та високої точності керування автономними суднами.

Інтелектуальні системи управління рухом, як зазначено в дослідженні, є важливою складовою для розвитку автономних суден, оскільки вони дозволяють таким об'єктам адаптуватися до складних умов навколишнього середовища, включаючи мінливі погодні умови та перешкоди на маршруті. Впровадження таких систем потребує використання складних математичних моделей і потужних обчислювальних ресурсів для швидкого аналізу даних і прийняття рішень в реальному часі.

В контексті інтелектуальної системи контролю доступу до автономних суден, це дослідження має особливу цінність. Автономні судна, що функціонують без участі людей, потребують високого рівня захисту від несанкціонованого доступу до систем керування. Інтелектуальні системи можуть бути налаштовані на постійний моніторинг стану судна і здійснення контролю за будь-якими спробами втручання або зміни курсу. В статті [4] наголошується на важливості застосування алгоритмів штучного інтелекту (ШІ) для адаптивного управління судном в реальному часі, що може бути застосовано і для систем контролю доступу. Такі алгоритми здатні не лише коригувати курс судна, але й реагувати на загрози безпеці, оцінюючи різні ризики і надаючи пріоритети відповідним діям.

Серед основних викликів, зазначених у статті [5], є забезпечення стабільності та точності керування великими автономними суднами, які мають значну інерцію та погані маневрові характеристики. Це є особливо важливим у контексті контролю доступу, оскільки несанкціоноване втручання в управління такими суднами може призвести до серйозних інцидентів. Системи контролю доступу повинні забезпечувати високий рівень безпеки і надійності, щоб запобігти можливості зловмисного втручання в системи керування.

Стаття [6] також акцентує увагу на необхідності постійного вдосконалення систем автоматичного управління рухом. Використання інтелектуальних алгоритмів дозволяє підвищити ефективність роботи суден, скоротити витрати на експлуатацію та зменшити ризики, пов'язані з людським фактором. Для

систем контролю доступу це означає, що впровадження інтелектуальних алгоритмів може забезпечити більш надійний захист, аналізуючи потенційні загрози в реальному часі та адаптуючи свої функції відповідно до змін умов експлуатації.

Таким чином, аналіз статті [7] показує, що інтелектуальні системи, які використовуються для управління автономними суднами, можуть бути адаптовані для створення ефективних систем контролю доступу. Вони дозволять не тільки забезпечити надійне управління судном, але й захистити його від зовнішніх загроз, що особливо важливо в умовах автономної роботи суден на великі відстані, коли фізичний доступ до судна обмежений

Дослідження, викладені в статті "State-of-the-Art Research on Motion Control of Maritime Autonomous Surface Ships" [8], описують сучасні підходи до управління автономними морськими суднами, які можуть бути використані для створення інтелектуальних систем контролю доступу. Інтеграція таких систем у процеси керування судном дозволяє забезпечити автономну роботу об'єктів з високим рівнем безпеки та захисту від зовнішніх загроз, використовуючи передові алгоритми штучного інтелекту та оптимізації. У цьому контексті стаття "Research on Synthesis of Multi-Layer Intelligent System for Optimal and Safe Control of Marine Autonomous Object" [4] присвячена розробці багаторівневої інтелектуальної системи керування автономними морськими об'єктами. Однією з ключових тем є синтез багатошарової системи керування для забезпечення безпечного та оптимального руху морських автономних об'єктів, використовуючи сучасні методи теорії керування, такі як нейронні мережі, нечітке керування та еволюційне програмування.

Автори досліджують різні алгоритми, включаючи динамічне програмування з нейронними обмеженнями стану, алгоритм мурашиних колоній та методи на основі прогнозуючого управління. Багаторівневі системи дозволяють застосовувати різні підходи до управління залежно від поточних

умов, таких як оптимізація маршруту, уникнення зіткнень та управління у портових маневрах.

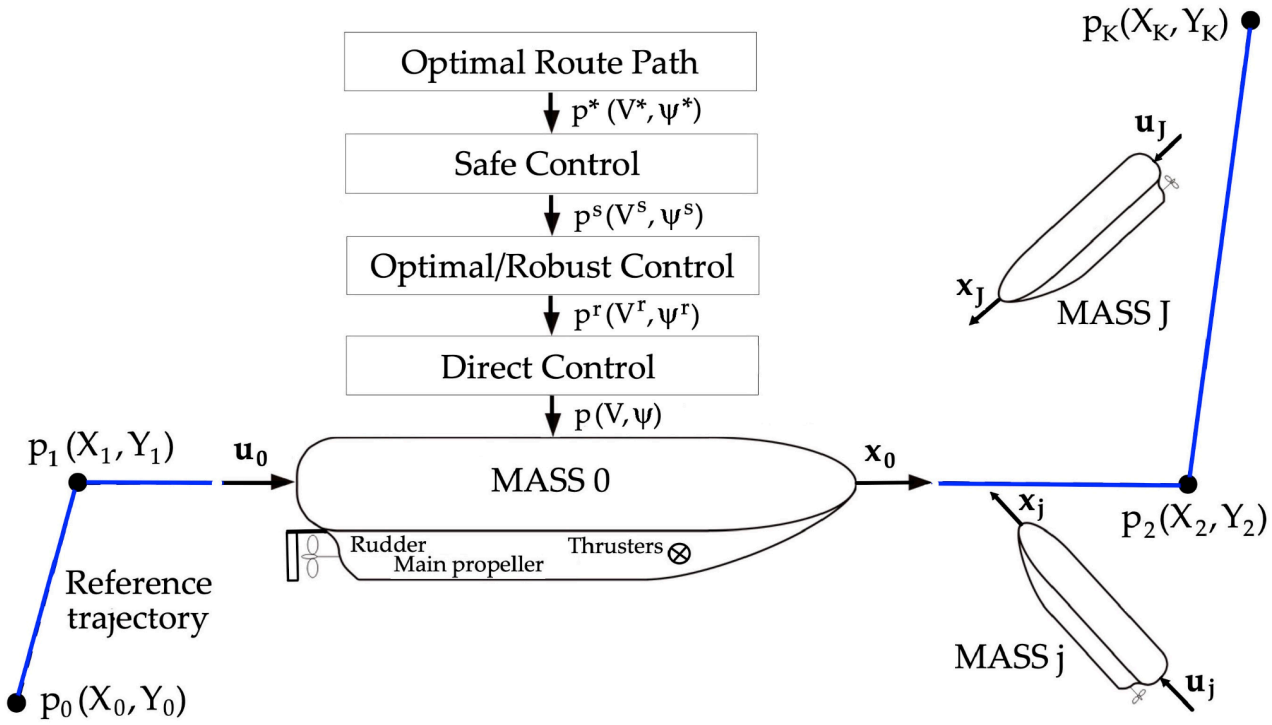

Рисунок 1.3 - Багаторівнева структура системи керування групою морських автономних надводних кораблів (MASS). [9]

Один із важливих аспектів статті [10] полягає в розгляді питання безпеки при управлінні автономними суднами. Стаття пропонує використання спеціальних алгоритмів для мінімізації ризиків зіткнень, використовуючи сучасні антизіткнення радари та штучний інтелект для автоматичного визначення безпечних траєкторій. Такі методи можуть бути інтегровані в інтелектуальні системи контролю доступу для забезпечення захисту суден від зовнішніх загроз, таких як несанкціоноване втручання в системи керування.

Інтелектуальні системи контролю доступу також можуть використовуватися для управління маршрутами суден у реальному часі, де ключовими критеріями є безпека і економічність. Алгоритми, розроблені в статті [11], враховують фактори, такі як споживання палива, тривалість шляху і уникнення динамічних перешкод, наприклад, рухомих суден або штормів. Ці

алгоритми можуть бути застосовані для визначення оптимальних і безпечних маршрутів, що особливо важливо в умовах інтенсивного морського руху.

Інший важливий аспект, який розглядається в статті [12], стосується використання методів штучного інтелекту, таких як нейронні мережі та нечітке управління, для забезпечення адаптивного керування суднами в умовах обмеженої видимості або в складних умовах навігації. Такі підходи можуть бути інтегровані в системи контролю доступу для автоматичного визначення найбільш оптимальних дій у разі виникнення небезпечних ситуацій, тим самим підвищуючи рівень безпеки автономних морських об'єктів.

У статті [4] також наведено приклади комп'ютерних симуляцій, які демонструють ефективність запропонованих алгоритмів для різних сценаріїв, включаючи управління у відкритих водах, маневрування в портах і уникнення зіткнень з іншими суднами. Це важливо для майбутніх досліджень та впровадження подібних систем в реальних умовах, де інтелектуальні системи контролю доступу можуть грати ключову роль у забезпеченні безпеки автономних суден.

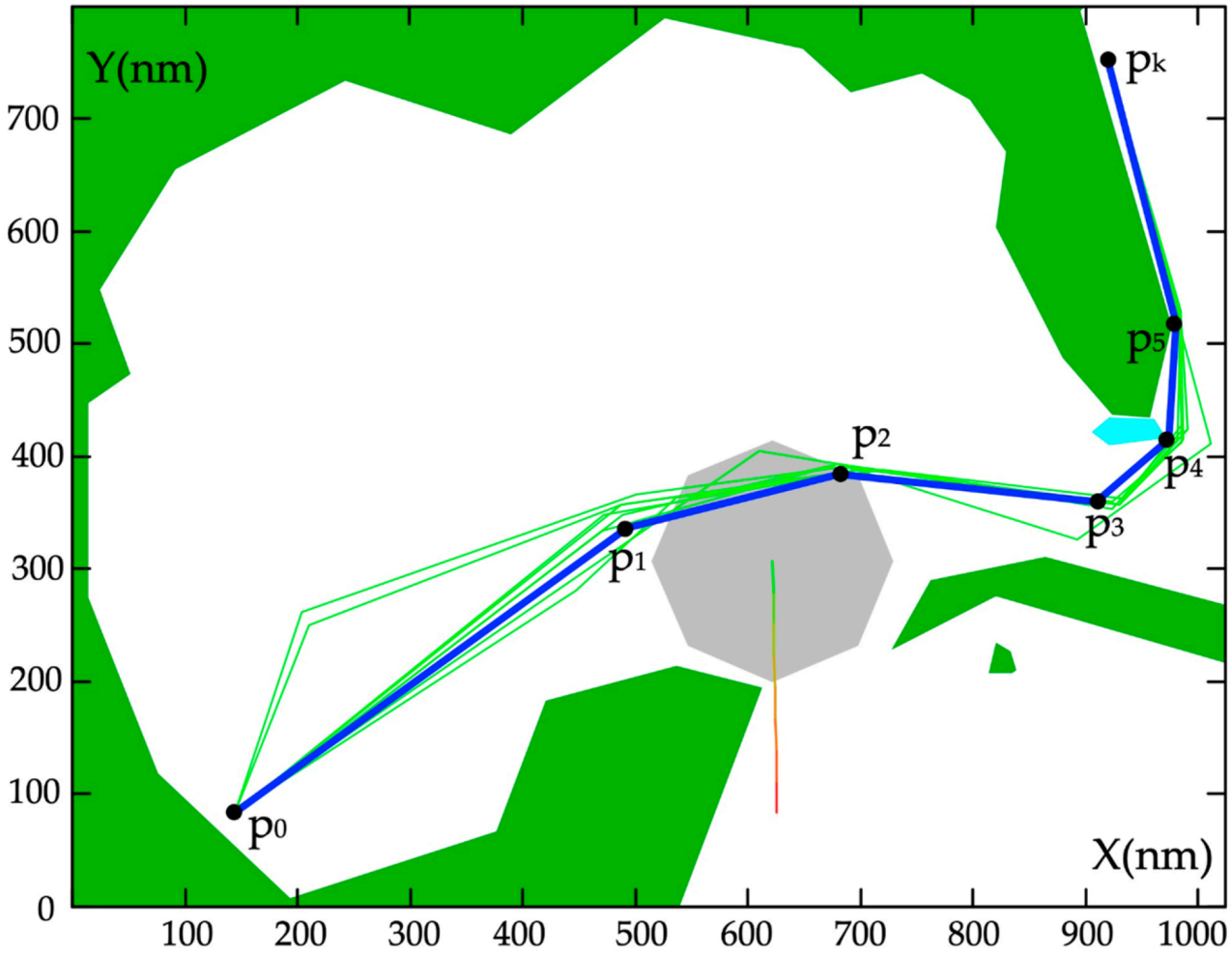

Рисунок 1.4 - Приклад визначення маршруту проходження за допомогою MPEA в Мексиканській затоці за наявності урагану. [13]

Інтелектуальні багатошарові системи управління, як показано у статті [14] "Research on Synthesis of Multi-Layer Intelligent System for Optimal and Safe Control of Marine Autonomous Object" [15], демонструють великий потенціал для використання в автономних морських суднах, підвищуючи рівень безпеки та надійності завдяки інтеграції алгоритмів штучного інтелекту. Ці підходи сприяють зменшенню ризиків і забезпечують оптимальне керування, що є основою для подальших розробок у сфері автоматизації морської індустрії. Саме такі інновації були представлені у прес-релізі "World Premiere for Rolls-Royce's mtu NautIQ Marine Automation Portfolio", де Rolls-Royce презентували новітнє портфоліо автоматизації для різних типів суден, що включає надійні системи контролю та управління для суден майбутнього.

Однією з ключових інновацій є система mtu NautIQ Master, яка замінює попередню версію mtu Callosum, надаючи користувачам розширені функціональні можливості на більш гнучкій платформі. Ця інтегрована система дозволяє централізовано управляти всіма підсистемами судна, включаючи навігацію, електрогенерацію та інші ключові системи, що дозволяє підвищити ефективність операцій та зменшити кількість людських помилок.

Також у портфелі mtu NautIQ Foresight є система прогнозування стану обладнання, яка моніторить технічний стан суден та дозволяє проводити передбачуване обслуговування. Це допомагає зменшити ризики поломок та забезпечити максимальну доступність суден при мінімальному споживанні палива і викидах CO2. Завдяки цьому, система може ефективно підтримувати роботу як окремих суден, так і цілих флотів, сприяючи досягненню екологічних цілей компанії Rolls-Royce.

Технологічні інновації

Однією з найбільш значущих інновацій є повністю інтегрована система управління мостиком, mtu NautIQ Bridge, що дозволяє автоматизувати навігацію та роботу основних систем судна. Також цікавою є технологія mtu NautIQ Genoline NG, яка оптимізує роботу генераторів на борту, забезпечуючи надійність та довговічність усіх енергетичних систем судна.

Крім того, всі продукти лінійки mtu NautIQ засновані на гнучких програмних платформах, які дозволяють легко інтегрувати нове обладнання чи програмне забезпечення впродовж усього життєвого циклу суден. Це дозволяє зменшити ризик застарівання обладнання та забезпечити його актуальність на тривалий термін.

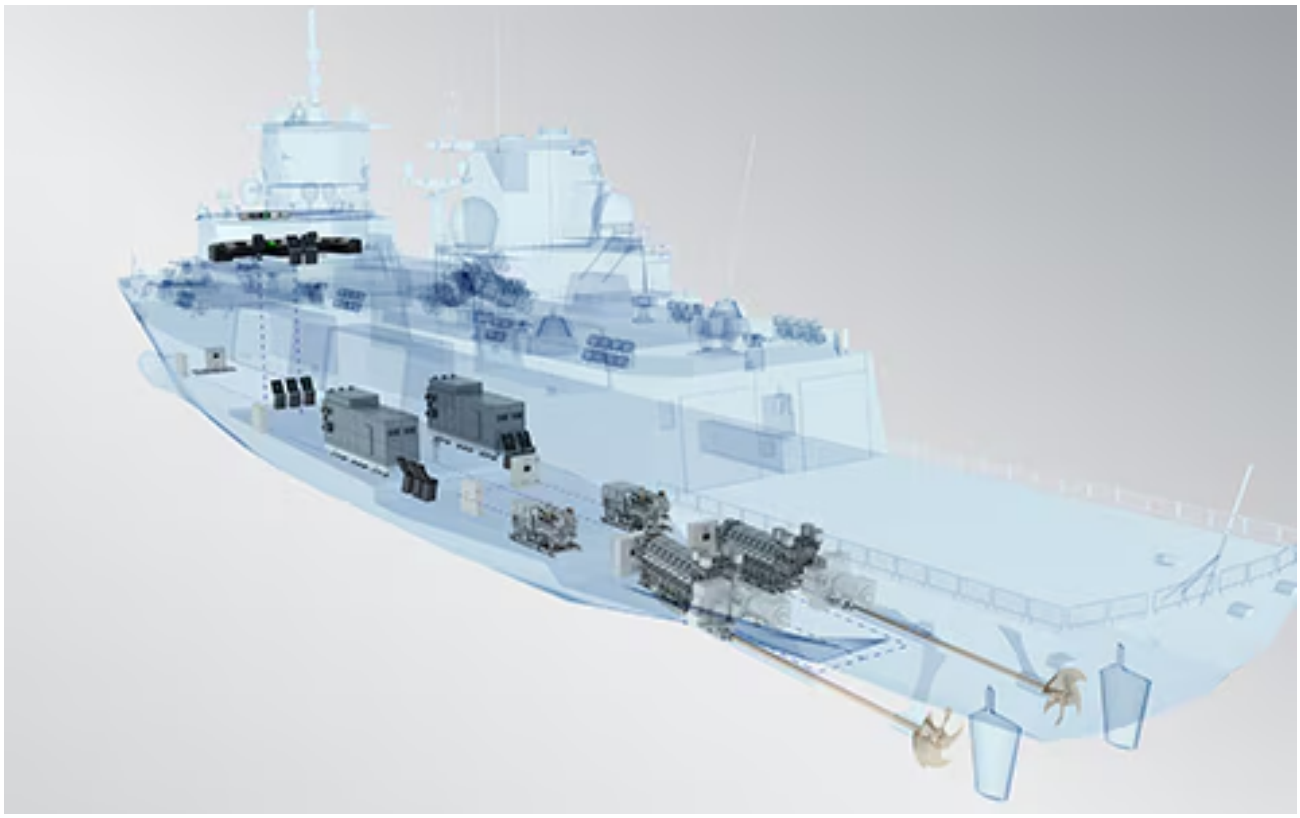

Рисунок 1.5 - Приклад поєднання нового портфоліо в собі технології mtu і Servowatch [8]

Представлення нової лінійки продуктів mtu NautIQ на виставці DSEI демонструє, що Rolls-Royce робить значний внесок у розвиток автоматизованих та автономних систем управління суднами. Використання інноваційних рішень, таких як система mtu NautIQ Master та mtu NautIQ Foresight, дозволяє забезпечити більшу ефективність роботи суден, надійність, а також зниження викидів CO2, що робить ці рішення надзвичайно актуальними для майбутнього розвитку морської галузі. Ці інновації тісно пов’язані з розвитком технологій дистанційного керування суднами, як показано у статті “Svitzer and Rolls-Royce demonstrate remote control tug operations” [16]. Це досягнення підкреслює потенціал використання передових систем для забезпечення дистанційного керування морськими об'єктами, що сприятиме подальшій автономізації та безпеці морських операцій. Rolls-Royce та Svitzer продемонстрували першу в світі дистанційно керовану комерційну операцію судна у порту Копенгаген, Данія. Використовуючи судно Svitzer Hermod, капітан зміг керувати судном на

відстані, виконуючи маневри в порту, такі як причалювання, відшвартування та 360-градусні повороти.

Судно оснащене передовою системою динамічного позиціонування від Rolls-Royce, а також комплексом сенсорів, що збирають дані для підвищення ситуаційної обізнаності капітана. Ці дані передавалися до віддаленого операційного центру, що дозволяло забезпечити точний контроль судна без фізичної присутності екіпажу на борту. Проєкт отримав підтримку з боку Lloyd's Register для забезпечення безпеки технологій.

Даний розвиток технологій є частиною глобальної тенденції в автоматизації суден, що дозволить у майбутньому значно підвищити безпеку та ефективність операцій. Капітану вдалося виконати всі маневри з дистанційного пункту управління без необхідності фізичної присутності на борту. Успішні випробування продемонстрували, що технологія дистанційного керування є перспективним шляхом для впровадження автономних суден у морську галузь. Це рішення дозволяє зменшити ризики для екіпажу і підвищити ефективність робочих процесів.

Даний проєкт є кроком вперед у розвитку автономних суден, які в майбутньому можуть стати нормою для комерційної морської індустрії, забезпечуючи високу ефективність і мінімізуючи людські ризики. Технології, представлені у випробуваннях, мають потенціал для революційних змін у морській навігації та управлінні судами.

Висновок

Сучасні інтелектуальні системи управління та автоматизація відіграють критичну роль у розвитку автономних морських суден, забезпечуючи адаптацію до мінливих умов навколишнього середовища, підвищуючи ефективність і безпеку їхнього функціонування. Одним із ключових аспектів є інтеграція алгоритмів штучного інтелекту, які дозволяють здійснювати автономну навігацію, уникаючи зіткнень та мінімізуючи ризики. Для забезпечення стабільності руху автономних суден використовуються такі методи, як

пропорційно-інтегрально-диференційний (PID) контроль, нечітка логіка та адаптивні контролери, що дозволяє оперативно реагувати на динамічні зміни курсу. Дослідження в цій галузі підтверджують, що впровадження інтелектуальних систем контролю доступу може підвищити загальну безпеку автономних суден, оскільки такі системи здатні адаптуватися до потенційних загроз. Це особливо важливо в контексті збільшення загроз кібербезпеки та обмеженого фізичного доступу до суден у відкритих водах. Інтелектуальні системи також можуть використовувати багатошаровий підхід до контролю доступу, що включає моніторинг дій користувачів у реальному часі та забезпечує захист від несанкціонованого втручання.

Таким чином, розвиток інтелектуальних систем управління та автоматизації є ключовим напрямом для забезпечення автономних морських об'єктів надійною системою управління, що дозволяє оптимізувати морські перевезення, підвищити їх безпеку та знизити ризики втручання в системи управління.

1.2 Безпека, аутентифікація та захист даних для автономних морських систем

Розвиток автономних морських систем потребує впровадження високих стандартів безпеки, ефективних механізмів аутентифікації та надійного захисту даних. Безпека цих систем є ключовим аспектом, оскільки вони працюють у складних умовах, часто без можливості фізичного втручання людини. У зв'язку з цим, захист інформаційних ресурсів, надійні методи аутентифікації користувачів і заходи для забезпечення конфіденційності даних набувають особливої важливості.

У цій главі будуть розглянуті ключові аспекти інформаційної безпеки, фактори аутентифікації систем контролю та управління доступом, а також специфіка захисту інформації, що зберігається в центрах обробки даних

автономних морських об'єктів. Ці питання є важливими для забезпечення надійного функціонування автономних суден і запобігання можливим кіберзагрозам, що можуть вплинути на їхню роботу.

Навчальний посібник «Інформаційна безпека» [3] авторів Кавун С.В., Носов В.В. і Манжай О.В. охоплює широкий спектр питань, пов’язаних із захистом інформаційних ресурсів в умовах сучасного технологічного розвитку. Основна увага в посібнику приділена таким ключовим аспектам, як загрози інформаційної безпеки, організація захисту інформаційних систем (ІС), канали витоку інформації та засоби захисту. Розглянуті технологічні та організаційні заходи для забезпечення цілісності, конфіденційності та доступності даних є основою для впровадження безпечних систем управління доступом в автономних морських системах, забезпечуючи їх надійне функціонування та захист від кіберзагроз.

У першому розділі посібника детально аналізуються загальні принципи захисту інформаційних технологій. Розглядається концепція “технічного захисту інформації” (ТЗІ), яка базується на інженерних і організаційних заходах для протидії загрозам, включаючи загрози, пов’язані з кібератаками, несанкціонованим доступом до інформації та порушенням цілісності даних. Також розглядаються правові основи захисту інформації, зокрема українське законодавство, що регулює питання державної таємниці, конфіденційної інформації та захисту інформаційних ресурсів.

Одним із важливих аспектів, піднятих у посібнику, є загрози несанкціонованого доступу до інформаційних систем, які можуть призвести до витоку конфіденційних даних або порушення роботи критично важливих інфраструктур. Автори також пропонують різноманітні стратегії для мінімізації цих ризиків, включаючи розробку комплексних систем захисту на основі оцінки вразливостей і загроз.

У першому розділі посібника автори детально розглядають загальні принципи інформаційної безпеки. Зокрема, у підрозділах 1.1-1.54 описуються

основні поняття, класифікація ресурсів для захисту, типи загроз, а також різноманітні типи атак та вірусів. Автори підкреслюють важливість визначення критичних інформаційних ресурсів та їх класифікацію для оптимального використання методів захисту. Також подається детальний аналіз класифікації загроз та моделей порушників, що дозволяє виявити потенційні вразливості системи.

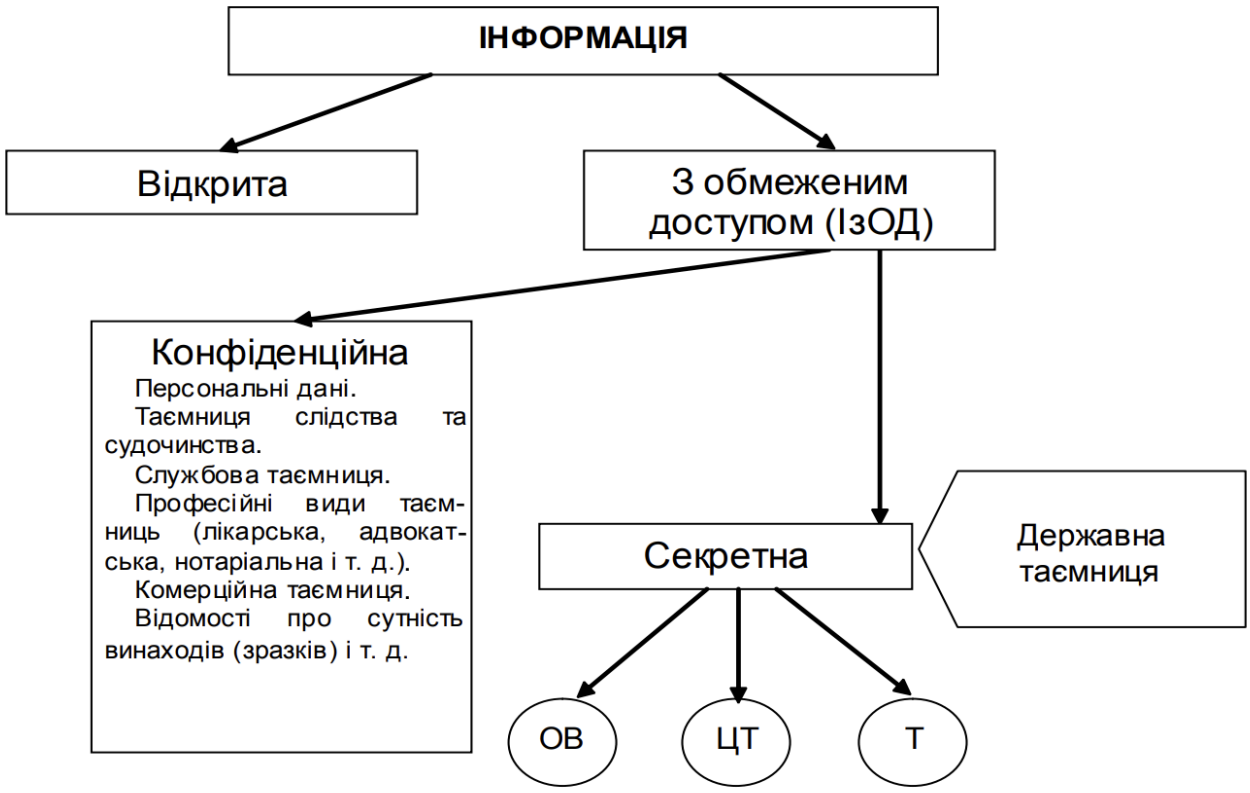

Рисунок 1.6 - Законодавча класифікація видів інформації в Україні [17]

Особливу увагу приділено питанням організації захисту інформаційних систем і технічному захисту інформації. Це включає заходи з блокування побічних випромінювань, а також методи забезпечення захисту каналів зв'язку. Автори пропонують підхід до класифікації загроз та атак, зокрема віддалених атак на комп'ютерні системи, що дозволяє створити більш ефективні стратегії безпеки.

Розділ 3 [18], який охоплює підрозділи 3.1-3.7 [19], присвячений організаційним аспектам забезпечення інформаційної безпеки. Автори наводять

приклади практичних заходів для захисту інформаційних систем, включаючи використання політик безпеки, систем контролю доступу та навчання персоналу. Важливим аспектом є питання навчання співробітників для мінімізації людського фактора, що часто є слабкою ланкою в забезпеченні інформаційної безпеки.

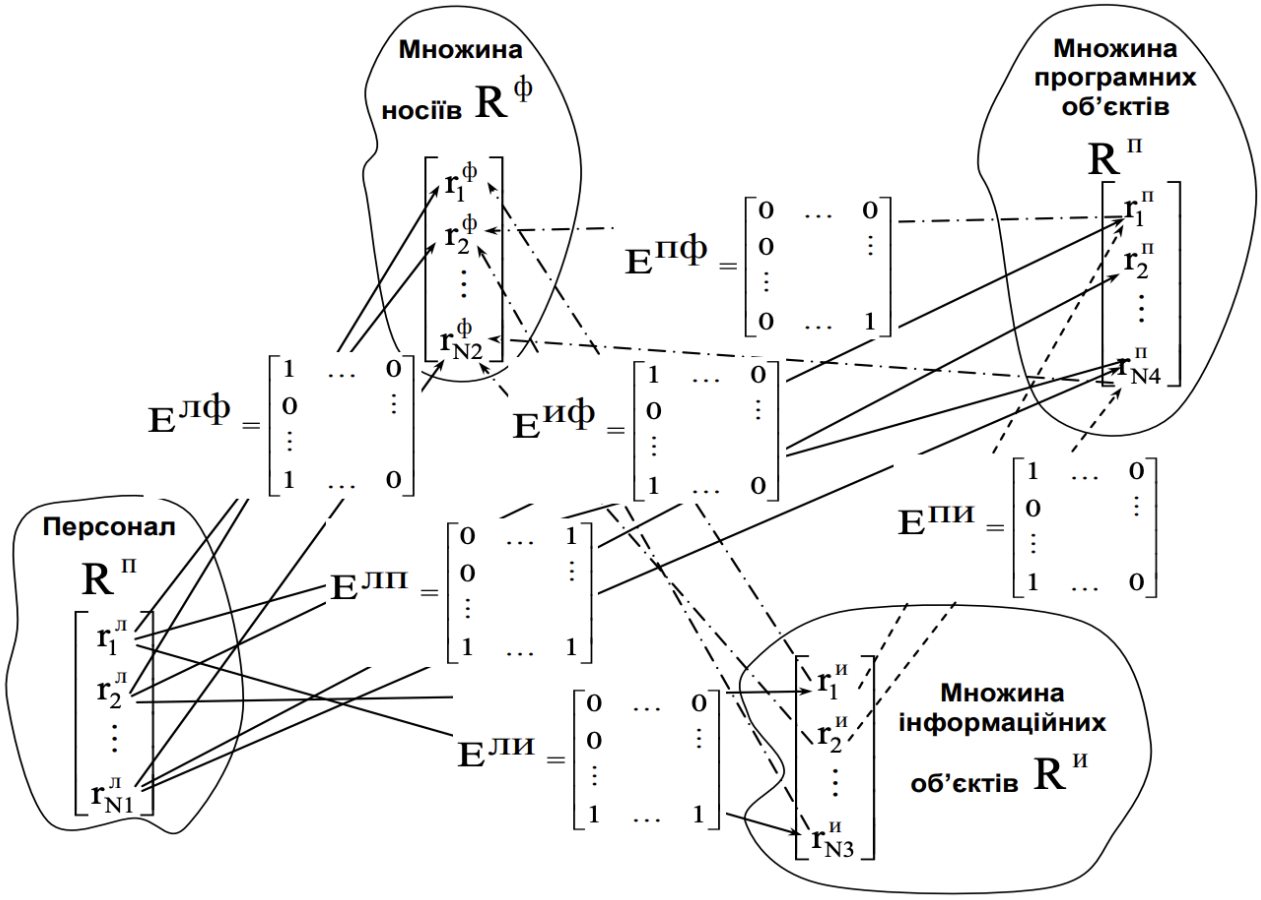

Рисунок 1.7 - Структурна модель системи об'єктів захисту [20]

Цей аналіз надає уявлення про методи і стратегії, викладені у посібнику, які можуть бути застосовані для побудови інтелектуальних систем контролю доступу до автономних морських об'єктів, що є темою вашого дослідження. Рекомендації з технічного та організаційного захисту можуть бути інтегровані у вашу модель для підвищення надійності та безпеки системи.

Технологічні заходи та стратегії захисту

Важливою частиною посібника є класифікація інформаційних ресурсів, що потребують захисту. Авторами пропонується підхід до оцінки цінності

інформації для її власників, який дозволяє визначити критичність кожного ресурсу. Відповідно, інформація класифікується за рівнем важливості, і для кожного класу ресурсів застосовуються відповідні методи захисту. Це включає захист інформації від несанкціонованого доступу, а також методи блокування побічних електромагнітних випромінювань і забезпечення захисту каналів зв’язку.

Стратегії захисту включають створення ефективних систем моніторингу та контролю доступу до інформаційних ресурсів, впровадження засобів шифрування даних та використання сучасних технологій для забезпечення кібербезпеки [21]. У посібнику також детально розглядаються питання організаційного захисту, включаючи політики доступу, системи керування доступом та навчання персоналу з питань безпеки інформації. Забезпечення інформаційної безпеки та надійної аутентифікації є важливим аспектом для будь-якої сучасної системи, особливо для автономних морських об'єктів, які працюють в умовах обмеженого фізичного доступу. Автономні морські системи потребують комплексного підходу до захисту інформаційних ресурсів, що включає як технічні, так і організаційні заходи, а також правове регулювання для підтримання безпеки.

Система контролю доступу та управління доступом (СКУД) в автономних морських об'єктах має базуватися на багатофакторній аутентифікації. Фактори аутентифікації включають паролі, фізичні пристрої для ідентифікації, такі як смарт-карти, і біометричні характеристики, наприклад відбитки пальців або аналіз райдужної оболонки ока. Комбінування різних факторів аутентифікації дозволяє знизити ризики несанкціонованого доступу та підвищити рівень безпеки.

Інформаційна безпека, крім захисту від зовнішніх загроз, охоплює також технічні заходи, такі як шифрування даних та захист каналів зв'язку, а також моніторинг доступу до критично важливих систем. У посібнику «Інформаційна безпека» [22] пропонується комплексний підхід до організації захисту, що

включає як технічні, так і правові аспекти. Актуальні питання захисту державної та конфіденційної інформації, розглянуті у цьому ресурсі, мають ключове значення для забезпечення стабільної роботи інформаційних систем автономних об'єктів.

Таким чином, інтеграція систем безпеки, аутентифікації та захисту даних в автономні морські об'єкти сприятиме забезпеченню високого рівня кібербезпеки, запобіганню несанкціонованого доступу та надійному функціонуванню систем у складних умовах. У цьому контексті стаття "Фактори аутентифікації системи контролю та управління доступом" досліджує процес аутентифікації в СКУД і аналізує основні фактори, які беруть участь у цьому процесі.

У статті [23] аналізуються сильні та слабкі сторони кожного з цих методів. Наприклад, паролі мають широке використання, але через низький рівень захищеності вони можуть бути вразливими до атак, таких як фішинг або крадіжка паролів. Пристрої аутентифікації, як-от ключі або смарт-карти, мають перевагу у вигляді володіння фізичним предметом, однак також можуть бути піддані ризику копіювання або втрати. Біометричні методи, що включають зчитування відбитків пальців, геометрії руки чи аналіз райдужної оболонки ока, пропонують більш високий рівень захисту, оскільки засновані на унікальних характеристиках кожної людини, але також мають свої недоліки, зокрема ризик збирання біометричних даних або помилкове відхилення аутентифікації.

Один із ключових висновків дослідження полягає в тому, що жоден з методів аутентифікації не є ідеальним і не може гарантувати повну безпеку. Автори рекомендують використовувати комбіновані підходи, які об’єднують кілька факторів, наприклад, пароль у поєднанні з біометричними даними або пристроєм аутентифікації. Такі багатофакторні системи аутентифікації суттєво знижують ризики несанкціонованого доступу і підвищують загальний рівень безпеки.

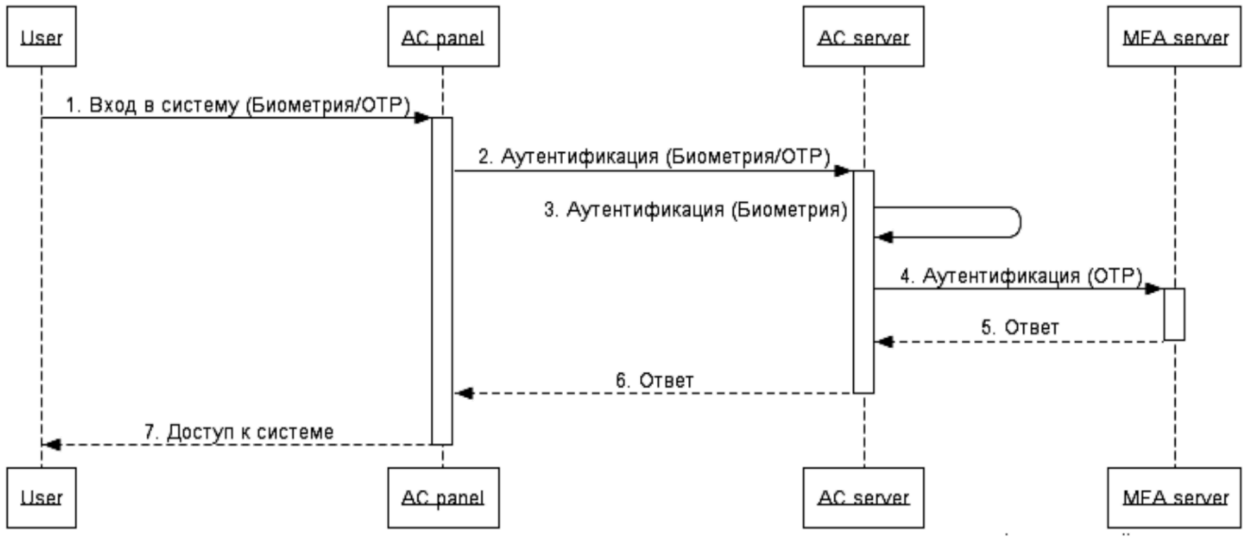

Рисунок 1.8 - Принцип взаємодії користувача та СКУД зчитувача [24]

Дослідження також акцентує увагу на сучасних викликах кібербезпеки, зокрема ризиках, пов'язаних з підміною серверів аутентифікації та розсинхронізацією даних при використанні одноразових паролів. Це підкреслює важливість розвитку більш безпечних та надійних механізмів аутентифікації у майбутніх системах контролю доступу.

Інший важливий аспект безпеки стосується захисту інформації, що зберігається у центрах обробки даних (ЦОД). У дослідженні "Щодо захисту інформації, що зберігається у центрах обробки даних" автори розглядають виклики забезпечення кібербезпеки, особливо в умовах військових конфліктів. ЦОД є критично важливими для зберігання даних, і їх захист від техногенних та природних катастроф є ключовим для забезпечення стабільної роботи автономних систем. Ці аспекти безпеки — аутентифікація, захист даних у ЦОД та забезпечення кібербезпеки — є основою для забезпечення безпеки, аутентифікації та захисту даних в автономних морських системах. Усе це стає критично важливим для розвитку морських автономних об'єктів, які функціонують в умовах зростаючих загроз та обмеженого фізичного доступу.

Стаття [25] акцентує увагу на Законі України № 7152, ухваленому у березні 2022 року, який дозволяє розміщувати державні інформаційні ресурси на хмарних платформах та в ЦОД, розташованих за межами України. Це рішення

було прийнято як тимчасовий захід для захисту даних в умовах, коли територія України піддається агресії з боку РФ.

Автор аналізує низку технологічних викликів, з якими стикаються ЦОДи, зокрема, можливі техногенні та природні катастрофи. Прикладом цього є пожежа у ЦОД компанії OVH у 2021 році, яка призвела до втрати даних, зберіганих у кількох дата-центрах. Чанишев також зазначає, що несприятливі погодні умови можуть спричинити серйозні збої в роботі ЦОД, як це сталося в Техасі у 2018 році.

Технічні аспекти вибору ЦОД

Автор наголошує на важливості правильного вибору ЦОД, де зберігаються критично важливі дані, на основі міжнародних стандартів надійності. Стаття детально розглядає класифікацію ЦОД за стандартами Telecommunications Industry Association (TIA), яка використовує систему рівнів від Tier I до Tier IV. Найвищий рівень надійності, Tier IV, передбачає дублювання всіх основних систем та мінімальний простій до 26 хвилин на рік, однак ЦОД цього рівня в Україні поки що немає.

Рівень Tier III, який є стандартом для великих підприємств і хмарних провайдерів, передбачає високу відмовостійкість і можливість безперервної роботи навіть під час технічного обслуговування. Це робить такі ЦОД основними кандидатами для розміщення державних даних, як зазначено в законодавчих вимогах України.

Висновок

Безпека, аутентифікація та захист даних є критично важливими для забезпечення надійної роботи автономних морських систем. Для цього впроваджуються багатофакторні методи аутентифікації, що комбінують різні фактори — знання, володіння і біометричні дані. Такий підхід дозволяє знизити ризики несанкціонованого доступу та підвищити загальний рівень захисту.

Інтелектуальні системи контролю доступу до автономних суден також включають багатошаровий підхід до захисту даних, що дає змогу адаптуватися

до мінливих загроз у реальному часі, зберігаючи стабільну роботу систем навіть за умов кіберзагроз. Це забезпечує високий рівень безпеки та надійності автономних суден під час їх роботи.

1.3 Регуляторні виклики у розвитку автономного судноплавства

Розвиток морських автономних поверхневих суден (MASS) супроводжується численними проблемами та викликами, зокрема в регуляторній сфері. Нові тенденції у розвитку автономних суден вимагають адаптації технологій, регулювання і морської індустрії в цілому до нових реалій. Виникають критичні питання, такі як забезпечення безпеки, захист від кіберзагроз, а також вирішення юридичних та етичних викликів, що супроводжують інтеграцію автономних суден у глобальну морську систему. Створення міжнародних стандартів для автономних суден потребує змін існуючих правил і розробки нових підходів до взаємодії з такими системами.

Одним з ключових аспектів, розглянутих у цьому, є складність розробки міжнародних регуляторних стандартів для автономних суден. Оскільки всі поточні регуляції, такі як Міжнародна конвенція з охорони людського життя на морі (SOLAS) та Міжнародні правила попередження зіткнень на морі (COLREGs), були розроблені для суден з екіпажем, вони потребують суттєвих змін для врахування специфіки автономних систем управління. Наприклад, стаття зазначає, що вимоги SOLAS щодо швидкого ручного керування в небезпечних ситуаціях навігації не можуть бути виконані автономними суднами без екіпажу. Це потребує розробки нових методів взаємодії з такими системами і відповідної регуляторної адаптації.

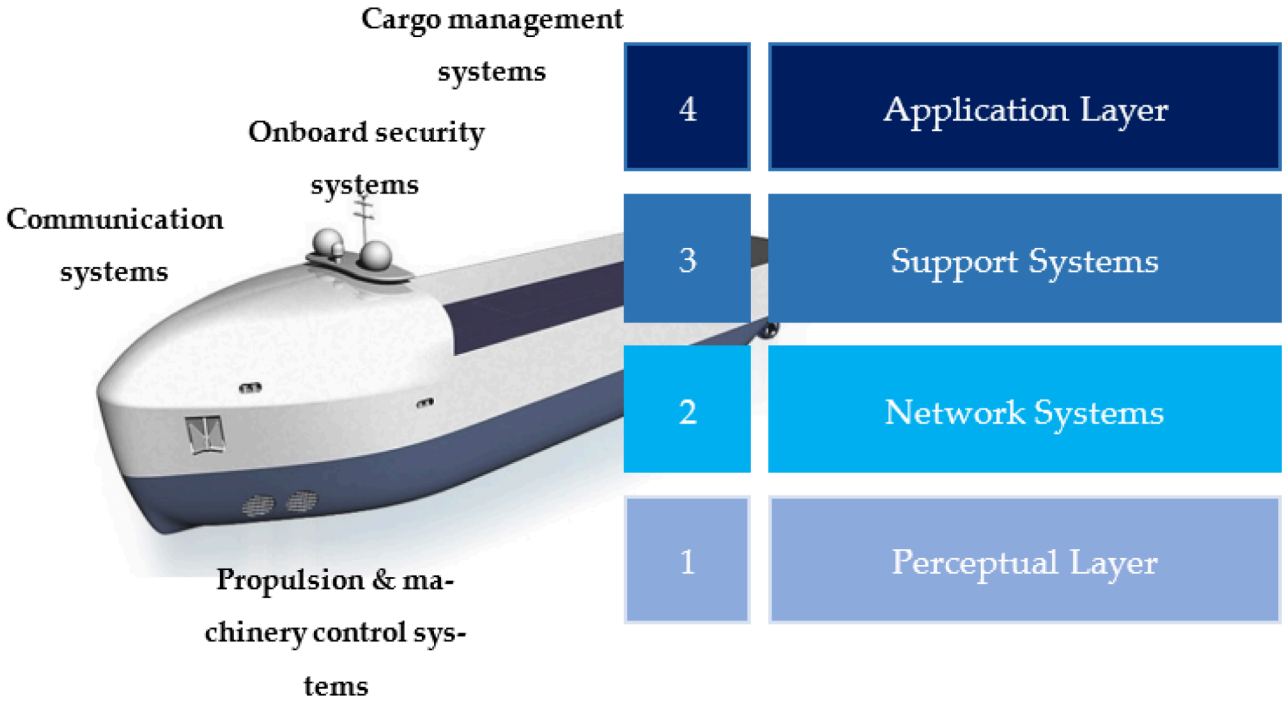

Рисунок 1.9 - Основні системи, котрі інтегровані у сучасні судна, з чотирьох рівнів кіберсистем. [26]

- Cargo management systems → Системи управління вантажами
- Onboard security systems → Бортові системи безпеки
- Communication systems → Системи зв'язку
- Propulsion & machinery control systems → Системи управління рухом та обладнанням
- Application Layer (4) → Прикладний рівень (4)
- Support Systems (3) → Системи підтримки (3)
- Network Systems (2) → Мережеві системи (2)
- Perceptual Layer (1) → Шар сприйняття (1)

Окрім регуляторних викликів, автори звертають увагу на технологічні аспекти, пов'язані з розробкою та впровадженням автономних суден. Використання новітніх технологій, таких як штучний інтелект (ШІ), інтернет речей (IoT) та великих даних, є ключовим елементом для успішної інтеграції автономних суден. Проте, технології безпеки та надійності цих систем повинні

бути значно покращені для забезпечення високого рівня кібербезпеки та захисту від збоїв датчиків або програмного забезпечення. Це особливо важливо, враховуючи ризики, пов’язані з кіберзагрозами, які можуть порушити функціонування судна, або навіть призвести до втручання у навігаційні системи через підробку даних GPS.

Також у статті [27] розглянуто питання впливу автономних суден на морську індустрію, зокрема на будівництво суден, роботу портів та інші аспекти інфраструктури. Автономні технології вже застосовуються в авіації, залізничному та автомобільному транспорті, і судноплавство є наступною сферою, яка активно інтегрує ці інновації. Автори наголошують, що успішне впровадження автономних суден вимагатиме координації між усіма зацікавленими сторонами, включаючи суднобудівників, порти та державні регулятори.

Одним з найважливіших аспектів дослідження є аналіз юридичних і етичних питань, пов’язаних із впровадженням автономних суден. Оскільки ШІ не може нести юридичну відповідальність, виникає питання щодо того, хто нестиме відповідальність у разі аварії або іншого інциденту за участю автономного судна. Це вимагає створення чітких правових рамок, які б регулювали відповідальність між власниками суден, виробниками та розробниками технологій.

Рисунок 1.10 - Ключові стейкхолдери для суден [6]

- Shipping Company → Судноплавна компанія
- Government International Organization → Урядова міжнародна організація
- Verification of regulations and implementation → Перевірка правил і їх впровадження
- Shipbuilding Company → Суднобудівна компанія
- Ship construction & warranty → Будівництво кораблів та гарантія
- Equipment Manufacturer → Виробник обладнання
- Equipment supply and technical support → Постачання обладнання та технічна підтримка
- Spare parts and technical support → Запасні частини та технічна підтримка
- Classification Society → Класифікаційне товариство
- Inspection, consulting → Інспекція, консультування
- Inspection and certification → Інспекція та сертифікація

- Acting on behalf of → Діє від імені

Розвиток автономних надводних суден (MASS) супроводжується численними проблемами та викликами, що потребують адаптації регуляторних стандартів, технологій і підходів до безпеки. Сучасні регуляції, такі як Міжнародна конвенція з охорони людського життя на морі (SOLAS) та правила попередження зіткнень на морі (COLREGs), не враховують специфіки автономних суден, тому їх необхідно суттєво адаптувати. Для успішного впровадження автономного судноплавства необхідно розробити нові регуляторні стандарти, посилити технологічну безпеку, а також вирішити юридичні та етичні питання, зокрема щодо відповідальності за аварії чи інциденти без участі екіпажу.

У вирішенні цих викликів ключову роль відіграють партнерства між великими технологічними компаніями, такими як Rolls-Royce та Intel. Співпраця між Rolls-Royce та Intel спрямована на розробку інтелектуальних автономних систем, що забезпечать безпечніше та ефективніше комерційне морське судноплавство. Rolls-Royce зосереджується на технологіях для автономних суден, тоді як Intel забезпечує потужні обчислювальні можливості для аналізу і обробки даних. Такі партнерства дозволяють створювати інноваційні рішення, що підвищують ефективність морських операцій, зменшують ризики та підтримують високий рівень безпеки.

Це підкреслює важливість одночасного вирішення регуляторних викликів і активного залучення провідних компаній для розвитку технологій автономного судноплавства. Співпраця між регуляторами та технологічними партнерами створює основу для успішної інтеграції автономних суден у глобальну морську систему, забезпечуючи їхню безпеку, надійність та ефективність. Саме на цьому наголошує співпраця Rolls-Royce та Intel, яка спрямована на створення інтелектуальних автономних систем суден, що допоможуть зробити комерційні морські перевезення більш безпечними та ефективними.

У прес-релізі [28] “Rolls-Royce and Intel announce autonomous ship collaboration” оголошує про співпрацю між Rolls-Royce та Intel для розробки інтелектуальних автономних систем суден. Метою цієї колаборації є створення новітніх розумних рішень для судновласників та операторів, які допоможуть зробити комерційні морські перевезення безпечнішими і ефективнішими. Rolls-Royce зосереджена на розробці технологій для автономних суден, а Intel забезпечує потужні обчислювальні можливості для аналізу та обробки даних.

Основна увага приділяється розробці систем штучного інтелекту, обчислювальних рішень на краю мережі та системам зберігання даних, що дозволять судам самостійно керувати навігацією, виявленням перешкод та комунікаціями. Такі системи матимуть інфраструктуру, схожу на ту, що використовується у “розумних містах” та автономних автомобілях, що дозволить суднам виконувати операції з мінімальним втручанням людини.

Rolls-Royce та Intel також працюють над оптимізацією використання Intel Xeon Scalable Processors та Intel Optane для забезпечення високої продуктивності та надійності під час автономної роботи суден. Система дозволить забезпечити безперервний моніторинг судна та його навколишнього середовища, мінімізуючи ризики для екіпажу та підвищуючи ефективність операцій.

Технологічний підхід

Суть технологічної співпраці між Rolls-Royce і Intel полягає у використанні сучасних обчислювальних рішень для управління автономними системами суден. Intel FPGA та Xeon Scalable Processors дозволять обробляти складні моделі, а технології Intel Optane забезпечать надійність систем зберігання даних та безперебійне функціонування судна.

Рисунок 1.11 - Візуалізація складних інтелектуальних систем доставки, які зроблять комерційні перевезення безпечнішими [7]

Висновок

Розвиток автономного судноплавства стикається з численними регуляторними викликами, пов'язаними з адаптацією існуючих міжнародних стандартів до нових реалій. Поточні норми, розроблені для суден з екіпажем, потребують перегляду, щоб врахувати специфіку автономних систем, забезпечуючи безпечну інтеграцію таких суден у глобальну морську інфраструктуру.

Партнерства, такі як співпраця Rolls-Royce з Intel, допомагають розвивати технологічні рішення для автономного судноплавства, спрямовані на підвищення безпеки та ефективності операцій. Впровадження новітніх технологій і міжгалузева співпраця сприяють вирішенню регуляторних викликів та створенню правової бази для використання автономних суден у майбутньому.

1.4 Обґрунтування вибору теми дослідження

Вибір теми дослідження "Інтелектуальна система контролю доступу центру управління автономними морськими об'єктами" зумовлений необхідністю забезпечення безпеки і надійності роботи центрів управління автономними морськими об'єктами. Основна увага приділяється створенню бази даних центру обробки даних (ЦОД) з ролями та доступами, яка є ключовим інструментом для організації ефективного та безпечного управління доступом до ресурсів системи. ЦОД забезпечує централізоване управління доступом до критично важливих систем і даних, що дозволяє уникнути несанкціонованого втручання і підвищує загальну стійкість до кіберзагроз. Такий централізований підхід є особливо важливим для автономних морських об'єктів, оскільки вони функціонують у складних умовах, де фізичний доступ до об'єкта є обмеженим або взагалі відсутнім.

Інноваційність запропонованої роботи полягає в багатошаровому підході до контролю доступу, що включає різні рівні аутентифікації та моніторинг дій користувачів у реальному часі. Використання інтелектуальної системи контролю дозволяє автоматично адаптувати рівні доступу до змінних умов експлуатації, враховуючи можливі загрози і ризики. Це підвищує гнучкість і надійність системи, забезпечуючи ефективний захист від кіберзагроз. Такий підхід особливо актуальний для сучасних умов, коли кількість та складність кібератак зростає. Система забезпечує моніторинг і контроль доступу, дозволяючи оперативно реагувати на несанкціоновані спроби. Запропонована інтелектуальна система контролю доступу сприяє більш надійному зберіганню даних, мінімізує ризики несанкціонованого доступу та забезпечує стабільну роботу автономних морських об'єктів у будь-яких умовах.

Рисунок 1.12 - Візуалізація складних інтелектуальних систем доставки, які зроблять комерційні перевезення безпечнішими https://gmcg.global/shippings-digital-and-sustainability-transition-and-the-future-of-seafarers/

1.5 Висновки до розділу 1

Розробка інтелектуальної системи контролю доступу для центру управління автономними морськими об'єктами є важливим кроком для забезпечення надійності та безпеки функціонування таких об'єктів. Автономні морські судна мають значний потенціал у підвищенні ефективності морських перевезень, але водночас потребують сучасних рішень для захисту від кіберзагроз та несанкціонованого доступу. В умовах стрімкого розвитку технологій автономного судноплавства, впровадження інтелектуальних систем контролю доступу дозволить забезпечити стабільне і безпечне функціонування автономних об'єктів навіть у складних умовах навколишнього середовища.

У даній роботі запропоновано використання сучасних методів штучного інтелекту для адаптивного моніторингу і аналізу загроз у реальному часі, що дозволяє швидко реагувати на потенційні загрози та забезпечувати надійну роботу системи управління автономними об'єктами. Це рішення сприяє підвищенню рівня захисту від кіберзагроз, що особливо актуально в умовах сучасного розвитку технологій автономного судноплавства. Використання таких методів дозволяє забезпечити безперервний моніторинг стану системи, своєчасне виявлення загроз та оперативне реагування на них, що є критичним для безпеки автономних морських об'єктів.

Важливість цієї роботи полягає також у підвищенні загального рівня безпеки морських перевезень для України, що сприятиме розвитку морської галузі, підвищенню її ефективності та конкурентоспроможності на міжнародному рівні. Впровадження інтелектуальної системи контролю доступу забезпечить стабільність роботи автономних суден, зменшить ризики, пов'язані з несанкціонованим втручанням, та сприятиме створенню надійної інфраструктури для майбутнього розвитку морського транспорту.

РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ

2.1 Аналіз предметної області

Для розробки бази даних користувачів і їх прав доступу до системи обробки даних (СОД) необхідно глибоко аналізувати предметну область, яка включає визначення ролей користувачів, їх взаємодії з системою та виділення основних прав доступу, що вони можуть мати. Система СОД використовується для управління даними в корпоративному масштабі, що включає збір, зберігання, оновлення та аналіз даних з метою підтримки прийняття рішень та операційної діяльності.

2.1.1 Ролі та користувачі

В корпоративній системі кожен користувач має певну роль, яка визначає його повноваження та обов'язки. Ролі можуть варіюватися від адміністраторів системи, які мають доступ до всіх рівнів системи, до кінцевих користувачів, які мають обмежений доступ до певних даних або функцій. Важливо структуризувати ролі так, щоб вони відображали організаційну структуру та потреби в доступі до даних, забезпечуючи при цьому необхідний рівень безпеки та конфіденційності.

2.1.2 Система прав доступу

Права доступу визначають, які дії може виконати користувач у системі. Ці права пов'язані з ролями і можуть включати можливість читання, запису, зміни та видалення даних. Важливо чітко визначити, які права доступу повинні бути

призначені для кожної ролі, щоб запобігти несанкціонованому доступу та забезпечити цілісність та захист даних.

2.1.3 Модель безпеки

Основою безпеки в системі є правильно спроектована модель, яка регулює, як користувачі взаємодіють з системою через призначені їм ролі та права доступу. Модель має забезпечувати, що всі доступи до системи контрольовані та документовані, що дозволяє відслідковувати зміни та доступи до важливої інформації.

2.1.4 Застосування SQL структури

Структура бази даних, яка була розроблена, відображає ці принципи через створені таблиці: users, roles, permissions, user_roles, та role_permissions. Таблиця users зберігає інформацію про користувачів та їх активність у системі. Таблиця roles визначає різні ролі, які можуть бути призначені користувачам, включаючи їх описи, що деталізують обов’язки і повноваження кожної ролі. Таблиця permissions включає список можливих дій або прав доступу, які можуть бути надані ролям, вказуючи деталізовані описи для кращого розуміння обмежень кожного права доступу. Таблиця user_roles є зв'язковою і відображає відносини між користувачами та ролями, дозволяючи кожному користувачу мати одну чи більше ролей. Ця багато-до-багатьох відносини забезпечує гнучкість у керуванні ролями користувачів у складних організаційних структурах. Таблиця role_permissions також репрезентує багато-до-багатьох відносини, але між ролями та правами доступу, що дозволяє детально налаштовувати рівні доступу для кожної ролі в системі. Важливо зазначити, що добре спланована структура таблиць і їх взаємозв’язки забезпечують основу для ефективної імплементації політики безпеки. Зокрема, використання зовнішніх ключів у таблицях

user_roles та role_permissions забезпечує цілісність даних і допомагає відслідковувати і контролювати доступ до ресурсів у системі. Ця структура бази даних не тільки дозволяє адміністраторам системи легко керувати користувачами та їх ролями, але також забезпечує, що всі операції з даними виконуються відповідно до встановлених правил і політик. Настроювання ролей і прав доступу через централізовану базу даних дозволяє корпораціям ефективно реагувати на зміни в регуляторному середовищі та оперативних потребах. Завдяки впровадженню цієї структури, корпорація може підвищити рівень захисту конфіденційних даних і оптимізувати управління ресурсами. Використання структурованого підходу до керування правами доступу забезпечує, що кожен користувач має доступ тільки до тих ресурсів, які необхідні для виконання його професійних обов’язків, мінімізуючи таким чином ризики несанкціонованого доступу та витоку даних.

2.2 Концептуальне проектування бази даних

Концептуальне проектування бази даних є критичним етапом в процесі розробки інформаційних систем, оскільки воно задає основу для всієї структури даних, яка буде використовуватися в системі. Цей етап включає глибокий аналіз сутностей, які є частиною системи, їхніх атрибутів та взаємозв'язків. Ретельне проектування на цьому рівні забезпечує правильність і ефективність подальших етапів реалізації та експлуатації системи.

У будь-якій інформаційній системі важливу роль відіграють користувачі, які є основними суб’єктами взаємодії із системою. Користувачі можуть представляти як фізичних осіб, так і автоматизовані агенти, що виконують певні дії автоматично або за попередньо заданими алгоритмами. Основною метою роботи з користувачами є ідентифікація, контроль доступу та забезпечення персоналізованої і безпечної взаємодії з системою. Це особливо важливо для забезпечення належного функціонування системи в умовах високих навантажень і можливих зовнішніх загроз. Для організації ефективної роботи

кожному користувачеві призначається одна або кілька ролей, які визначають обсяг його повноважень та функцій у системі. Ролі відіграють ключову роль у структуризації доступу до ресурсів системи, оскільки дозволяють розмежовувати права доступу користувачів і відповідати технічним та бізнес-вимогам.

2.2.1 Основні сутності та їх атрибути

Визначення основних сутностей і їхніх атрибутів є ключовим етапом у проектуванні будь-якої інформаційної системи, оскільки саме цей процес забезпечує відповідність системи бізнес-вимогам та технічним специфікаціям. Сутності, які відображають реальні об’єкти або абстрактні концепції, формують основу системи, а їх атрибути деталізують характеристики кожної сутності.

Таблиця 2.1 "users"

Поле	Тип	Опис
user_id	Число	Це унікальний ідентифікатор користувача, який використовується для забезпечення унікальності записів у таблиці
username	Строка	Ім'я користувача, яке використовується для авторизації в системі.
password	Строка	Зашифрований пароль для забезпечення безпеки користувача.

Поле	Тип	Опис
email	Строка	Електронна пошта користувача, яка також використовується для комунікації та відновлення доступу.
last_login	Логічний оператор	Показник, що вказує на активність акаунта користувача у системі.
failed_login_attempts	Число	Кількість невдалих спроб входу, що дозволяє реалізувати блокування після певної кількості спроб.
password_changed_at	Строка	Дата останньої зміни пароля, що допомагає виконувати вимоги щодо регулярної зміни пароля.
is_active	Логічний оператор	Показник, що вказує на активність акаунта користувача у системі.

Таблиця 2.2 "roles"

Поле	Тип	Опис
role_id	Число	Ідентифікатор ролі, що дозволяє визначити різні рівні доступу в системі.
role_name	Строка	Назва ролі, яка використовується для швидкого ідентифікування.
description	Строка	Опис ролі, який детально розкриває обов'язки та відповідальності, пов'язані з кожною роллю.

Таблиця 2.3 "permissions"

Поле	Тип	Опис
permission_id	Число	Унікальний ідентифікатор, що дозволяє системі управляти доступом до різноманітних функцій.
permission_name	Строка	Назва права, яке дозволяє ідентифікувати і регулювати дії користувачів у системі.
description	Строка	Пояснення, яке деталізує дозволені дії, асоційовані з кожним правом доступу.

2.2.2 Взаємозв'язки між сутностями

Значення взаємозв'язків між сутностями не можна недооцінювати, оскільки вони визначають логіку взаємодії всередині інформаційної системи:

- Зв'язок між Користувачі та Ролі (user_roles): Багато-до-багатьох. Один користувач може мати декілька ролей, і одна роль може бути призначена багатьом користувачам. Це підтримується таблицею user_roles, що містить поля user_id і role_id.

  Ролі користувачів (user_roles):

  - user_id (INT, REFERENCES users(user_id)) – посилання на користувача.
  - role_id (INT, REFERENCES roles(role_id)) – посилання на роль.

- PRIMARY KEY (user_id, role_id) – унікальний зв'язок між користувачем і роллю.

- Зв'язок між Ролі та Права доступу (role_permissions): Багато-до-багатьох. Роль може включати кілька прав доступу, а право доступу може бути присвоєне кільком ролям. Це також підтримується проміжною таблицею role_permissions, що включає role_id і permission_id.

  Права доступу ролей (role_permissions)

  - role_id (INT, REFERENCES roles(role_id)) – посилання на роль.
  - permission_id (INT, REFERENCES permissions(permission_id)) – посилання на право доступу.
  - PRIMARY KEY (role_id, permission_id) – унікальний зв'язок між роллю і правом доступу.

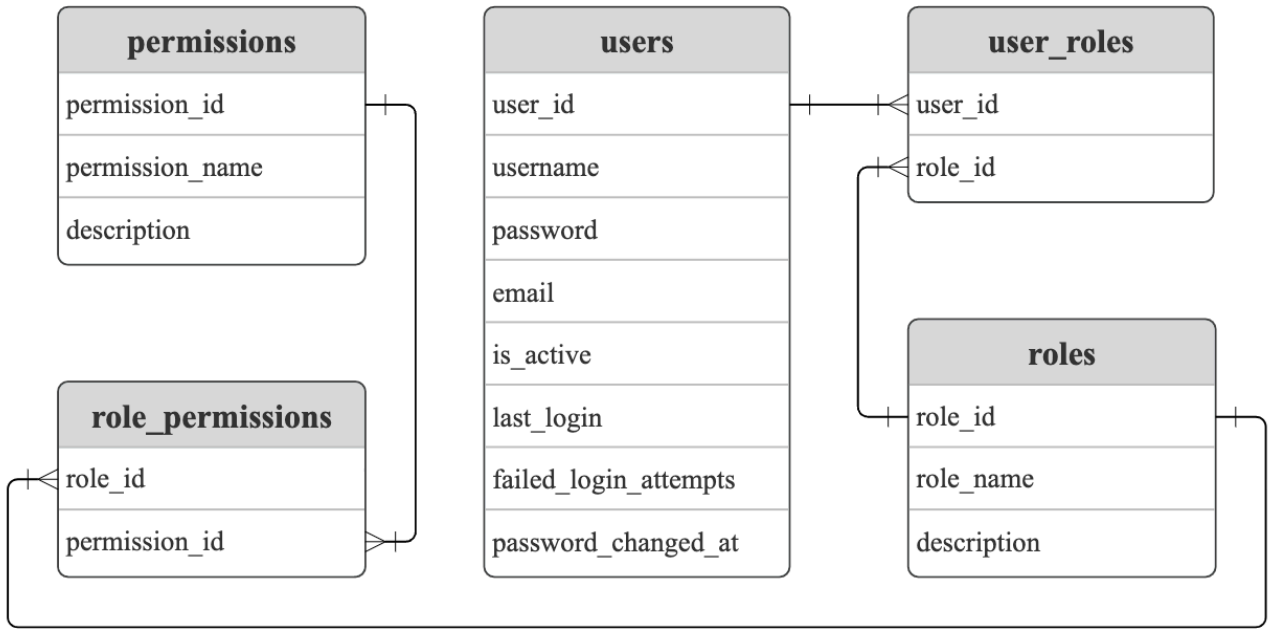

Рисунок 2.1 - ER-модель бази даних інтелектуальної системи контролю доступу

Ретельне концептуальне проектування з урахуванням взаємозв'язків між сутностями дозволяє створити міцну основу для подальшого розроблення інформаційної системи. Результатом є ER-модель, яка чітко відображає

структуру даних, необхідну для ефективного управління доступом користувачів. Така модель є незмінною та повністю готова до реалізації, що забезпечить успішне впровадження інформаційної системи відповідно до встановлених вимог

2.3 Взаємозв'язки між сутностями

Індекси в базі даних є важливим інструментом для оптимізації запитів і забезпечення швидкого доступу до даних. У даній структурі бази даних індекси використовуються для підвищення продуктивності при виконанні операцій вибірки, особливо для великих наборів даних. Нижче наведено індекси, налаштовані для таблиці users:

- Індекс на поле email: Унікальний індекс на поле email дозволяє швидко знаходити користувачів за електронною адресою. Це особливо корисно під час авторизації, оскільки запити на пошук користувача за email виконуються швидше.
- Індекс на поле username: Індекс на поле username допомагає покращити продуктивність при пошуку за іменем користувача. Це важливо для функцій, які дозволяють швидко знаходити користувачів за їхнім ім'ям, наприклад, для пошуку всередині системи або адміністративних функцій.
- Індекс на поле last_login: Індекс на полі last_login забезпечує швидкий доступ до записів користувачів з урахуванням останньої активності. Це полегшує виконання запитів, що стосуються активності користувачів,для виявлення неактивних користувачів або аналізу їхньої активності.

Оптимізація бази даних за допомогою індексів дозволяє значно підвищити продуктивність системи, особливо для операцій вибірки даних, що часто виконуються. Індекси на полях email, username та last_login у таблиці users забезпечують ефективний доступ до інформації про користувачів, мінімізуючи час пошуку й підвищуючи швидкість виконання запитів. Це є важливим

фактором для масштабованості та стабільності системи, особливо в умовах великої кількості даних або високих навантажень. Правильне налаштування індексів сприяє створенню більш продуктивної та надійної архітектури бази даних, що відповідає потребам бізнесу та забезпечує позитивний користувацький досвід.

2.4 Вибір та обґрунтування субд для створення бази даних

2.3.1 Критерії вибору СУБД

При виборі системи управління базами даних (СУБД) для реалізації проекту бази даних користувачів і їх прав доступу до системи обробки даних (СОД) важливо враховувати кілька ключових критеріїв, які забезпечать ефективність, масштабованість і надійність роботи системи:

- Продуктивність та масштабованість: СУБД повинна ефективно обробляти великі обсяги даних і забезпечувати високу швидкість запитів, особливо в умовах зростаючого обсягу даних.
- Безпека: Забезпечення захисту даних від несанкціонованого доступу, підтримка шифрування даних та різних рівнів доступу.
- Підтримка стандартів: СУБД повинна підтримувати стандарти SQL, що забезпечує легку інтеграцію з іншими системами та додатками.
- Надійність та відмовостійкість: Здатність до швидкого відновлення після збоїв, підтримка резервного копіювання та відновлення даних.

2.4.2 Аналіз СУБД

Для аналізу були вибрані наступні системи управління базами даних (СУБД): PostgreSQL, MySQL, Oracle Database і Microsoft SQL Server. Кожна з цих СУБД має свої унікальні характеристики, що робить їх придатними для різних типів проектів залежно від вимог

Таблиця 2.4 Типи баз даних

СУБД	Переваги	Недоліки
PostgreSQL	Відкритий код, висока продуктивність, підтримка складних запитів, розширення, документація	Потребує оптимізації для великих баз даних
MySQL	Популярність, велика спільнота, легкість використання, ефективність для веб-додатків	Менше можливостей для складних запитів порівняно з PostgreSQL
Oracle Database	Екстенсивна підтримка транзакцій, висока надійність, можливості для великих корпорацій	Висока вартість ліцензій, складне управління
Microsoft SQL Server	Інтеграція з продуктами Microsoft, висока безпека, легкість управління	Обмежена підтримка на інших платформах, висока вартість

2.4.3 Вибір СУБД для проекту

На підставі проведеного аналізу, для подальшого проектування обирається PostgreSQL. Цей вибір обумовлений наступними факторами:

- Відкритий код та вільне використання: Знижує загальні витрати на впровадження системи.

- Висока продуктивність і підтримка складних транзакцій: Забезпечує ефективність роботи з великими обсягами даних.
- Розширюваність та підтримка міжнародних стандартів: Гарантує сумісність із зовнішніми системами та легкість інтеграції.
- Надійність: Відома своєю стабільністю і мінімальними вимогами до обслуговування в продуктивному середовищі.

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, яка розробляється як проект з вільним програмним забезпеченням. Вона відома своєю надійністю, масштабованістю та здатністю обробляти великі обсяги даних, забезпечуючи високу продуктивність навіть у складних операційних навантаженнях. PostgreSQL підтримує велику кількість даних і дозволяє користувачам розширювати систему без ліцензійних обмежень. Завдяки підтримці стандартів SQL та можливостям для складних запитів, система є вибором для багатьох розробників і компаній, які потребують розширеного управління транзакціями та надійності. PostgreSQL також підтримує різноманітні типи даних, включаючи примітивні типи, структуровані типи, документо-орієнтовані типи і геопросторові дані, що робить її ідеальним вибором для комплексних додатків. Ця СУБД надає високий рівень безпеки з функціями, які включають шифрування, розширене управління правами доступу та засоби аудиту. Такий набір функцій робить PostgreSQL ефективним рішенням для будь-якої організації, що вимагає надійного зберігання та обробки даних у безпечному середовищі.

Враховуючи ці аспекти, PostgreSQL вважається оптимальним вибором для нашого проекту, оскільки вона забезпечує необхідну гнучкість, безпеку та ефективність для розробки комплексної системи управління правами доступу до СОД.

2.4 Логічного проектування бази даних

Логічне проектування бази даних полягає в деталізації структури таблиць, визначенні атрибутів кожної таблиці та параметрів цих атрибутів, заснованих на концептуальній моделі, яка була розроблена раніше. Цей процес включає специфікацію типів даних, обмежень та взаємозв'язків між таблицями, щоб забезпечити надійність та ефективність управління даними у вибраній системі управління базами даних PostgreSQL. Нижче наведено детальний опис основних таблиць, які мають бути створені.

На підставі ER-моделі бази даних, яка була розроблена в рамках концептуального проектування, база даних, що проектується, має містити наступні таблиці:

1. Таблиця users
2. Таблиця roles
3. Таблиця permissions
4. Таблиця user_roles
5. Таблиця role_permissions

Таблиця 2.5 "users"

Поле	Тип	Властивості	Опис
user_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор користувача
username	VARCHAR(255)	NOT NULL	Ім'я користувача для входу
password	VARCHAR(255)	NOT NULL	Хеш пароля користувача
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна адреса користувача

Поле	Тип	Властивості	Опис
last_login	BOOLEAN	NOT NULL	Час останнього входу користувача
failed_login_attempts	INT	DEFAULT 0	Кількість невдалих спроб входу
password_changed_at	TIMESTAMP		Дата останньої зміни пароля
is_active	BOOLEAN	NOT NULL	Чи є користувач активним

Таблиця 2.6 "roles"

Поле	Тип	Властивості	Опис
role_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор ролі
role_name	VARCHAR(255)	NOT NULL	Назва ролі
description	TEXT		Детальний опис ролі

Таблиця 2.7 "permissions"

Поле	Тип	Властивості	Опис
permission_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор права
permission_name	VARCHAR(255)	NOT NULL	Назва права доступу

Поле	Тип	Властивості	Опис
permission_id	SERIAL	PRIMARY KEY	Унікальний ідентифікатор права
description	TEXT		Опис права

Таблиця 2.8 "user_roles"

Поле	Тип	Властивості	Опис
user_id	INT	REFERENCES users(user_id)	Зв'язок з таблицею users
role_id	INT	REFERENCES roles(role_id)	Зв'язок з таблицею roles
		PRIMARY KEY (user_id, role_id)	Унікальний зв'язок між користувачем та роллю

Таблиця 2.9 "role_permissions"

Поле	Тип	Властивості	Опис
role_id	INT	REFERENCES roles(role_id)	Зв'язок з таблицею roles
permission_id	INT	REFERENCES permissions(permission_id)	Зв'язок з таблицею permissions

Поле	Тип	Властивості	Опис
		PRIMARY KEY (role_id, permission_id)	Унікальний зв'язок між роллю та правом доступу

Ці таблиці формують основу логічної структури бази даних, кожна з яких відповідає за певний аспект управління інформацією в системі. Завдяки чіткому визначенню атрибутів та типів даних, а також встановленню обмежень і зв'язків, база даних оптимізована для ефективної взаємодії з користувачами і забезпечення безпеки інформації

2.5 Фізичне проектування бази даних

Фізичне проектування бази даних є останнім етапом у процесі створення баз даних, який включає встановлення характеристик зберігання даних на фізичному рівні. Це включає вибір конкретних типів даних для кожного поля, методів індексації, а також параметрів зберігання, що оптимізують продуктивність і доступність системи.

2.5.1 Вибір СУБД і налаштування параметрів

Для реалізації бази даних користувачів та їх прав доступу до СОД було обрано систему управління базами даних PostgreSQL. Цей вибір базується на кількох важливих характеристиках, що включають високу продуктивність, підтримку розширених транзакцій, масштабованість, відкритий код, а також сильні можливості у сфері безпеки та компліянсу. Далі наведено детальний опис налаштувань параметрів СУБД, щоб забезпечити оптимальну продуктивність і надійність.

Налаштування параметрів PostgreSQL

Основні параметри PostgreSQL, які потрібно оптимізувати для досягнення кращої продуктивності і стабільності, включають:

- work_mem: Цей параметр визначає кількість пам'яті, яка буде використовуватися для внутрішніх операцій сортування і хеш-таблиць до того, як вони будуть записані на диск. Встановлення цього параметру в залежності від обсягу оперативної пам'яті і кількості одночасних запитів може значно покращити продуктивність оперативних запитів. Рекомендоване значення може варіюватися від 1 МБ до 10% загального обсягу доступної ОЗП.

- maintenance_work_mem: Цей параметр керує максимальною кількістю пам'яті, яку PostgreSQL може використати для операцій обслуговування бази даних, таких як VACUUM, CREATE INDEX і т.д. Великі значення можуть прискорити процеси вакуумування і реконструкції індексів.
- shared_buffers: Встановлює кількість пам'яті, яку СУБД використовує для кешування блоків даних. Як правило, рекомендується встановити цей параметр від 25% до 40% від загальної кількості ОЗП, що доступна на сервері.
- effective_cache_size: Параметр підказує планувальнику запитів про те, скільки пам'яті очікується використовувати для кешування даних і індексів у PostgreSQL та операційній системі. Це не виділяє фізичну пам'ять, але впливає на вибір планів запитів.
- wal_buffers: Встановлює розмір буфера для записування журналів транзакцій. Збільшення цього параметра може допомогти під час інтенсивних операцій запису.
- checkpoint_segments і checkpoint_completion_target: Використовуються для контролю частоти і тривалості перевірок, що може вплинути на продуктивність та відновлення системи після збою.

Рекомендації щодо використання та налаштування

- Моніторинг і налаштування в реальному часі: Використовуйте інструменти моніторингу, такі як PgAdmin або командний рядок, для спостереження за продуктивністю системи і робіть коригування у налаштуваннях на основі зібраних даних.
- Тестування продуктивності: Регулярно проводьте бенчмаркінг і тестування навантаження, щоб забезпечити, що система відповідає вимогам користувачів.
- Безпека: Зверніть увагу на налаштування безпеки, включаючи шифрування даних на диску і в мережі, а також налаштування прав доступу на рівні ролей та користувачів.

Правильне налаштування цих параметрів є ключовим для забезпечення високої продуктивності, надійності та масштабованості бази даних, особливо в умовах, коли база даних розширюється або використовується для критично важливих застосунків.

2.5.2 Фізична структура таблиць і оптимізація

Фізичне проектування бази даних полягає в реалізації логічної моделі у конкретні структури даних, що зберігаються на дисках. Це включає вибір типів даних для стовпців, методи зберігання, індексацію для підвищення швидкості виконання запитів, а також інші оптимізації для забезпечення надійності, швидкості доступу та ефективності обробки запитів.

Визначення таблиць

На основі логічного проектування, створено наступні таблиці:

Таблиця **users**

Ця таблиця зберігає основні дані про користувачів системи.

CREATE TABLE users (

```
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    last_login TIMESTAMP,
    failed_login_attempts INT DEFAULT 0,
    password_changed_at TIMESTAMP,
    is_active BOOLEAN NOT NULL
);
```

user_id — автоматично збільшуваний первинний ключ.

username — ім'я користувача, максимальна довжина 255 символів.

password — пароль, максимальна довжина 255 символів, зберігається в зашифрованому вигляді.

email — унікальна електронна пошта користувача.

is_active — логічний стовпець, що вказує чи активний акаунт користувача.

Таблиця **roles**

Зберігає інформацію про ролі в системі.

```
CREATE TABLE roles (
    role_id SERIAL PRIMARY KEY,
    role_name VARCHAR(255) NOT NULL,
    description TEXT
);
```

role_id — первинний ключ.

role_name — назва ролі.

description — текстовий опис ролі.

Таблиця **permissions**

Визначає дозволи, які можуть бути призначені ролям.

```
CREATE TABLE permissions (
    permission_id SERIAL PRIMARY KEY,
    permission_name VARCHAR(255) NOT NULL,
    description TEXT
);
```

permission_id — первинний ключ.

permission_name — назва дозволу.

description — детальний опис дозволу.

Таблиця **user_roles**

Встановлює зв'язки між користувачами та ролями.

```
CREATE TABLE user_roles (
    user_id INT REFERENCES users(user_id),
    role_id INT REFERENCES roles(role_id),
    PRIMARY KEY (user_id, role_id)
);
```

user_id і role_id — зовнішні ключі, що формують композитний первинний ключ.

Таблиця **role_permissions**

Визначає, які дозволи належать до яких ролей.

```
CREATE TABLE role_permissions (
    role_id INT REFERENCES roles(role_id),
    permission_id INT REFERENCES permissions(permission_id),
    PRIMARY KEY (role_id, permission_id)
);
```

role_id і permission_id — зовнішні ключі, що формують композитний первинний ключ.

Оптимізація та Індексація

```
CREATE INDEX idx_users_email ON users(email);
```

```
CREATE INDEX idx_users_username ON users(username);
CREATE INDEX idx_users_last_login ON users(last_login);
```

Для покращення продуктивності запитів, важливо правильно використовувати індексацію:

- Індексація електронних адрес в таблиці users на полі email допомагає швидко знаходити користувача за його email та гарантує унікальність значень.
- Індексація зв'язків у таблицях user_roles і role_permissions покращує швидкість виконання операцій з'єднання (joins), які часто використовуються для перевірки прав доступу користувачів.

Ці оптимізації забезпечують не тільки швидкість обробки даних, але й допомагають утримувати цілісність і безпеку інформаційної системи.

2.5.3 Оптимізація запитів

Оптимізація запитів є критичною для забезпечення високої продуктивності інформаційних систем. Вона включає ряд заходів, спрямованих на мінімізацію часу відповіді на запити і зменшення навантаження на систему. У цьому розділі будуть розглянуті основні стратегії та техніки, що застосовуються в рамках СУБД PostgreSQL для оптимізації запитів.

Одним з перших кроків у оптимізації запитів є їх ретельний аналіз. Це включає перевірку запитів на наявність неефективних операцій, таких як повторне сканування великих таблиць, надмірні об'єднання (joins) або неоптимальне використання індексів. Використання EXPLAIN: У PostgreSQL команда EXPLAIN дозволяє аналізувати план виконання запиту. Це дає змогу розуміти, які операції виконуються, чи використовуються індекси, і як можна модифікувати запит для підвищення його ефективності.

Ефективна індексація є ключем до швидкого доступу до даних. Індекси значно знижують кількість даних, що обробляються при виконанні запитів, і можуть відігравати роль в оптимізації запитів, особливо при читанні.

- Створення індексів: Створення індексів на стовпці, які часто використовуються в умовах запитів, може суттєво знизити час відповіді.
- Використання часткових індексів: Для таблиць з великою кількістю записів, де запити часто фільтрують за певною умовою, може бути корисним створення часткових індексів.

Кешування результатів запитів може покращити продуктивність, зокрема в системах з високою читаючою навантаженням.

- Використання пам'яті для кешування: Конфігурація СУБД для використання оптимального обсягу оперативної пам'яті для кешування може зменшити кількість дорогих операцій читання з диска.
- Кешування на рівні застосунку: Реалізація кешування у бізнес-лозіці застосунку для запитів, які не часто змінюються але часто запитуються.

Ці методи та стратегії оптимізації запитів, впроваджені належним чином, можуть значно покращити загальну продуктивність системи та забезпечити більш ефективне та надійне управління даними.

2.5.4 Безпека даних

Безпека даних в базі даних — це критично важлива складова, що забезпечує захист інформації від несанкціонованого доступу, витоку, зміни або втрати. В контексті системи управління базами даних PostgreSQL, розглянемо кілька аспектів, які потрібно врахувати для забезпечення безпеки даних.

Шифрування на диску (Data-at-Rest Encryption)

- Шифрування файлової системи: Використання цілісних рішень для шифрування файлової системи, таких як LUKS або BitLocker, забезпечує захист даних на фізичному рівні.

- Шифрування блоків PostgreSQL: Можливість PostgreSQL шифрувати окремі таблиці та блоки даних, що забезпечує додатковий рівень безпеки.

Шифрування під час передачі (Data-in-Transit Encryption)

- SSL/TLS: Конфігурація PostgreSQL для підтримки з'єднань через SSL або TLS гарантує, що всі дані, передані між клієнтом і сервером, зашифровані та захищені від перехоплення.

Автентифікація та контроль доступу

- Сильна автентифікація: Використання механізмів як LDAP або Kerberos для автентифікації користувачів. PostgreSQL дозволяє інтегрувати ці сервіси, забезпечуючи централізоване управління обліковими записами.
- Ролі та дозволи: Налаштування ролей з докладними дозволами для кожної операції в базі даних. Це включає визначення того, хто може читати, писати, змінювати або видаляти дані у конкретних таблицях.

Резервне копіювання та відновлення

- Стратегії резервного копіювання: Регулярне резервне копіювання даних є необхідним для відновлення системи після збоїв. Використання засобів PostgreSQL, таких як pg_dump для резервного копіювання та pg_restore для відновлення, забезпечує можливість швидкого відновлення.
- Гаряче резервне копіювання: Налаштування гарячого резервного копіювання (наприклад, за допомогою Barman або pgBackRest) забезпечує неперервність бізнес-процесів та мінімальний час простою.

Аудит і моніторинг

- Журналювання: Налаштування детального журналювання у PostgreSQL для відстеження всіх операцій, включаючи зміни даних, доступ користувачів та системні помилки.
- Моніторинг: Використання інструментів моніторингу (наприклад, pgAdmin або сторонні рішення як Zabbix або Prometheus) для

спостереження за станом бази даних, використанням ресурсів та потенційними безпековими інцидентами.

Ці заходи забезпечення безпеки даних в базі PostgreSQL допоможуть створити захищену, надійну і високопродуктивну інформаційну систему, готову до викликів сучасного ділового середовища.

2.6 Висновки до розділу 2

Розробка проекту бази даних для користувачів та їх прав доступу до системи обробки даних (СОД) стала значущим і важливим кроком у забезпеченні високого рівня організаційної ефективності та безпеки. Цей проект займає ключове місце у вирішенні проблеми контролю доступу до корпоративних ресурсів, забезпечуючи гнучкість та масштабування системи відповідно до зростаючих і змінюваних потреб бізнесу.

- Комплексне управління правами доступу: Проект успішно вирішив задачу моделювання складної структури доступу через детально сплановану базу даних. За допомогою створених таблиць та визначених взаємозв'язків між ними, система забезпечує динамічне управління правами доступу, що є важливим для організацій з різноманітними рівнями доступу та ролями користувачів.
- Вибір технологічного стеку: Використання PostgreSQL як системи управління базами даних виявилося обґрунтованим, враховуючи її масштабованість, високу продуктивність та надійність. СУБД забезпечила не тільки потрібний рівень безпеки, але й стабільну основу для реалізації вимог до інтеграції та управління даними.
- Забезпечення безпеки: Проект надає високий рівень захисту завдяки застосуванню сучасних методів шифрування та автентифікації, а також через впровадження строгих політик доступу та контролю змін. Такий

підхід мінімізує ризики несанкціонованого доступу та забезпечує захист конфіденційних даних.

- Адаптивність та масштабованість системи: Розроблена система здатна адаптуватися до змін у внутрішній структурі організації та технологічному середовищі. Можливість модифікації та розширення бази даних без втрати продуктивності або безпеки дозволяє системі залишатися ефективною в довгостроковій перспективі.
- Рекомендації для подальшого впровадження: Рекомендується провести детальний аналіз ефективності системи після її запуску для ідентифікації можливих областей для удосконалення. Також важливо забезпечити неперервне навчання та розвиток персоналу, який буде взаємодіяти з системою, для забезпечення її ефективного використання.

Завершення цього проекту значно підвищило рівень інформаційної безпеки в організації і сприяло поліпшенню загальної продуктивності за рахунок ефективнішого управління доступом до ресурсів. Такий підхід не тільки забезпечує захист даних, але й оптимізує робочі процеси, підвищуючи загальну оперативність та ефективність організаційної діяльності.

РОЗДІЛ 3. РОЗРОБКА, НАЛАШТУВАННЯ ТА ВІДЛАГОДЖЕННЯ ДОДАТКУ, ПРОВЕДЕННЯ ТЕСТІВ

3.1 Налаштування середовища розробки VS code та встановлення всіх необхідних бібліотек

Розробка будь-якого веб-додатка починається з вибору середовища, яке буде використовуватися для написання коду. Вибір редактора залежить від функціональних потреб проєкту, зручності роботи з певними бібліотеками та збирачами.

У цій магістерській роботі обрано середовище Visual Studio Code. VS Code (Visual Studio Code) – це популярний безкоштовний текстовий редактор та інтегроване середовище розробки (IDE), створене компанією Microsoft. Воно підтримує широкий спектр мов програмування та технологій, забезпечуючи зручність і продуктивність у процесі розробки.

Серед основних переваг і функцій Visual Studio Code можна виділити:

1. Підтримка багатьох мов програмування. VS Code працює з такими мовами, як JavaScript, Python, Java, C#, PHP, Ruby, HTML/CSS тощо, а також може розширюватися для роботи з іншими мовами завдяки додатковим розширенням.

2. Розширюваність. Середовище має велику екосистему плагінів і розширень, що дозволяє налаштовувати редактор під індивідуальні потреби та додавати нові функції.
3. Інтеграція з системами керування версіями. VS Code підтримує роботу з Git та іншими популярними системами керування версіями, що спрощує співпрацю над проєктами.
4. Швидкодія та легковагість. Побудований на основі Electron, VS Code працює швидко та не потребує значних ресурсів, що робить його оптимальним для виконання задач з розробки.
5. Інструменти для роботи з кодом. Середовище забезпечує підсвічування синтаксису, автодоповнення та інші функції, які значно полегшують написання коду.
6. Робочі простори. Розробники можуть організовувати свої проєкти у робочих просторах, що покращує структуру роботи та зберігає налаштування для кожного проєкту.
7. Кросплатформеність. VS Code підтримується на операційних системах Windows, macOS і Linux, що робить його доступним для розробників незалежно від платформи.

Популярність Visual Studio Code пояснюється його простотою у використанні, широкими можливостями розширення та активною спільнотою, яка регулярно випускає оновлення та нові плагіни для вдосконалення роботи з редактором.

Оскільки для створення додатка використовується React, необхідно налаштувати середовище Node.js, яке забезпечить його функціонування. Node.js – це середовище виконання (runtime environment), що дозволяє запускати JavaScript-код на стороні сервера. Воно побудоване на основі мови JavaScript та використовує двигун V8 від Google, який також застосовується у браузері Google Chrome для виконання JavaScript. У Node.js JavaScript використовується для розробки серверних додатків і скриптів.

Основні переваги та функції Node.js:

1. Асинхронність. Node.js реалізує подійно-орієнтовану архітектуру, яка дозволяє виконувати асинхронні операції, наприклад, обробку мережевих запитів, без блокування основного потоку виконання. Це робить його особливо ефективним для створення додатків із високим навантаженням.
2. Модульність. Використовуючи систему модулів CommonJS, Node.js дозволяє розділяти код на компактні модулі, які легко інтегруються в інші частини проєкту, забезпечуючи кращу структуризацію.
3. Пакетний менеджер npm. Node.js включає вбудований пакетний менеджер npm, що забезпечує швидке встановлення, керування сторонніми бібліотеками та залежностями, необхідними для розробки.
4. Підтримка серверних додатків. Node.js ідеально підходить для розробки серверних додатків, таких як веб-сервери, API, мікросервіси та мережеві застосунки.
5. Широке застосування. Node.js активно використовується для створення різноманітних застосунків: веб-додатків, інтернет-магазинів, систем реального часу (чатів), а також у мікросервісній архітектурі.

Node.js став популярним завдяки своїй швидкості та здатності ефективно обробляти численні одночасні з’єднання. Його ключовою перевагою є можливість використовувати одну мову програмування – JavaScript – для створення клієнтських і серверних додатків, що спрощує розробку та синхронізацію даних у веб-додатках.

Браузерний JavaScript та Node.js працюють у середовищі виконання V8, який є високопродуктивним двигуном, розробленим Google. V8 перетворює JavaScript-код у машинний код, що є низькорівневим та виконується без додаткової інтерпретації, забезпечуючи швидку роботу програм.

У браузерному середовищі JavaScript використовує однопоточну модель виконання, тобто весь код працює в одному потоці, відомому як “головний потік” або “UI потік”. Це може спричиняти проблеми, наприклад, блокування

інтерфейсу користувача під час виконання складних або тривалих операцій, оскільки всі задачі обробляються послідовно.

Для вирішення цих проблем і покращення відзивчивості програм застосовуються такі підходи:

1. Асинхронність. Використання асинхронних операцій та зворотних викликів (callback) дозволяє виконувати тривалі задачі, як-от запити до сервера чи обробку файлів, не блокуючи основний потік.
2. Обіцянки (Promises) і асинхронні функції. Застосування Promises та функцій async/await робить асинхронний код більш зрозумілим і зручним для управління.
3. Web Workers. Для важких обчислень у веб-середовищі можна використовувати Web Workers, які виконують задачі у фонових потоках, розвантажуючи основний потік.
4. Асинхронні бібліотеки та фреймворки. Використання бібліотек, що надають асинхронний API (наприклад, для роботи з мережею), спрощує написання ефективного асинхронного коду.
5. Оптимізація та кешування. Для підвищення продуктивності застосовуються методи оптимізації коду та кешування результатів обчислень, що дозволяє зменшити навантаження на основний потік.
6. Додаткові потоки в браузері. Сучасні браузери підтримують додаткові потоки для обробки таких задач, як мережеві запити або мультимедійні процеси, що забезпечує кращу розподіленість навантаження.

Таким чином, розв’язання проблем однопоточності у JavaScript зводиться до впровадження асинхронних підходів і методів оптимізації, які забезпечують відзивчивість програм та ефективне виконання задач без блокування інтерфейсу користувача.

Next.js – це популярний фреймворк для React, який дозволяє створювати сучасні веб-додатки з високою продуктивністю. Він розроблений компанією Vercel і забезпечує просту інтеграцію функцій, необхідних для повноцінної

роботи веб-додатків, таких як серверний рендеринг, генерація статичних сторінок і оптимізація продуктивності.

Основні особливості та переваги Next.js:

1. Серверний рендеринг (SSR). Next.js підтримує серверний рендеринг, що дозволяє рендерити сторінки на сервері та відправляти готовий HTML на клієнт. Це покращує SEO і забезпечує швидке завантаження сторінок для користувачів.
2. Генерація статичних сторінок (SSG). Next.js дозволяє створювати статичні сторінки під час збірки проєкту, що підвищує швидкість завантаження та масштабованість веб-додатків.
3. Автоматичне розділення коду. Next.js автоматично розділяє код, забезпечуючи завантаження лише необхідних частин для кожної сторінки, що зменшує час завантаження.
4. Роутинг на основі файлів. У Next.js роутинг побудований на основі структури файлів у директорії pages. Кожен файл у цій директорії відповідає окремій маршруту, що робить створення маршрутизації простим і зручним.
5. Інтеграція API. Next.js дозволяє створювати серверні API безпосередньо у додатку, використовуючи файли в директорії api. Це зручно для реалізації бекенду невеликих проєктів.
6. Оптимізація зображень. Вбудована функція оптимізації зображень забезпечує їх автоматичне стиснення, зміну розмірів та підбір найбільш підходящих форматів для покращення продуктивності.
7. Готовність до виробничого використання. Next.js включає всі необхідні інструменти для підготовки додатка до продакшн-оточення, включаючи можливість розгортання на популярних платформах, таких як Vercel, AWS або Netlify.
8. Підтримка TypeScript. Next.js має вбудовану підтримку TypeScript, що спрощує створення типізованих додатків і покращує процес розробки.

9. Широка екосистема. Завдяки активній спільноті та інтеграції з іншими інструментами React, Next.js має багатий вибір плагінів, бібліотек та ресурсів для спрощення роботи.

Next.js використовується багатьма великими компаніями, такими як Netflix, GitHub і TikTok, завдяки своїй гнучкості, продуктивності та простоті. Цей фреймворк ідеально підходить як для розробки статичних веб-сайтів, так і для динамічних багатосторінкових додатків.

3.2. Опис створення бази даних

Розробка бази даних для інтелектуальної системи контролю доступу була виконана у попередньому розділі 2, де описані всі етапи проектування. У цьому розділі наведено детальний опис технічної реалізації створення бази даних, включаючи використані команди SQL, пояснення їх функціоналу, а також приклади структури таблиць.

3.2.1 Створення основних таблиць

Основні таблиці для зберігання інформації про користувачів, ролі, права доступу та взаємозв’язки між ними були спроектовані та реалізовані у розділі 2 під час фізичного проектування бази даних. Це дозволило задати основну структуру бази даних, яка забезпечує ефективну обробку запитів і надійне управління даними.

У розділі 2 було детально описано логічну структуру бази даних, зокрема:

- Таблиця users для зберігання інформації про користувачів.
- Таблиця roles для визначення ролей у системі.
- Таблиця permissions для управління правами доступу.
- Таблиця user_roles для встановлення зв’язків між користувачами і ролями.

- Таблиця role_permissions для визначення відповідності між ролями і правами доступу.

Фізичне проектування включало вибір відповідних типів даних, створення індексів для оптимізації запитів та забезпечення цілісності даних за допомогою зовнішніх ключів. Детальний опис реалізації наведено у підрозділі 2.5.2.

Для підвищення продуктивності бази даних були створені індекси на найбільш запитувані поля. Індексація дозволила значно зменшити час виконання запитів і підвищити загальну ефективність роботи системи.

Детальні SQL-команди для створення таблиць та індексів були описані у підрозділах 2.4 та 2.5 розділу 2. Наприклад:

- Індекс idx_users_email забезпечує швидкий пошук користувачів за електронною адресою, що є важливим під час авторизації та управління обліковими записами.
- Індекс idx_roles_name оптимізує запити до таблиці roles, зокрема під час перевірки доступу та управління ролями.

Кожна таблиця була оптимізована для забезпечення швидкого доступу до даних за допомогою індексів. Для забезпечення цілісності даних використовувалися зовнішні ключі, що зв’язували таблиці між собою

3.2.2 Вставка тестових даних

Для перевірки працездатності бази даних було використано наступні SQL-команди для додавання тестових даних.

```
- Заповнення таблиці ролей
INSERT INTO roles (role_id, role_name, description) VALUES
(1, 'Адміністратор системи', 'Повний доступ до всіх ресурсів системи'),
(2, 'Оператор Центру управління', 'Моніторинг статусу об\'єктів, створення звітів'),
(3, 'Аналітик', 'Доступ до даних для аналізу'),
(4, 'Інженер технічного обслуговування', 'Доступ до технічних даних об\'єктів'),
(5, 'Гість', 'Обмежений доступ до загальної інформації');
- Заповнення таблиці прав доступу
```

```
INSERT INTO permissions (permission_id, permission_name, description) VALUES
(1, 'Читання даних', 'Доступ до перегляду інформації'),
(2, 'Запис даних', 'Можливість створювати та редагувати інформацію'),
(3, 'Видалення даних', 'Право видаляти дані'),
(4, 'Управління користувачами', 'Додавання, видалення та зміна ролей користувачів'),
(5, 'Адміністративний доступ', 'Повний доступ до налаштувань системи та даних'),
(6, 'Моніторинг системи', 'Доступ до журналів і звітів про активність системи'),
(7, 'Генерація звітів', 'Можливість створення звітів для аналізу'),
(8, 'Завантаження даних', 'Право експортувати дані для аналізу');
- Прив'язка прав доступу до ролей
-- Адміністратор системи
INSERT INTO role_permissions (role_id, permission_id) VALUES
(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8);
-- Оператор Центру управління
INSERT INTO role_permissions (role_id, permission_id) VALUES
(2, 1), (2, 6), (2, 7);
-- Аналітик
INSERT INTO role_permissions (role_id, permission_id) VALUES
(3, 1), (3, 7), (3, 8);
-- Інженер технічного обслуговування
INSERT INTO role_permissions (role_id, permission_id) VALUES
(4, 1), (4, 2), (4, 3);
-- Гість
INSERT INTO role_permissions (role_id, permission_id) VALUES
(5, 1);
```

3.2.3 Забезпечення безпеки

Для гарантування захисту інформації було реалізовано:

- Обмеження доступу до певних таблиць на рівні ролей у СУБД;
- Використання хешування паролів користувачів за допомогою алгоритму bcrypt.

3.2.4 Тестування запитів

Для перевірки коректності роботи бази даних використовувались наступні запити:

```
-- Вибір усіх активних користувачів
SELECT username, email FROM users WHERE is_active = TRUE;
-- Отримання ролей користувача
SELECT r.role_name
FROM user_roles ur
JOIN roles r ON ur.role_id = r.role_id
WHERE ur.user_id = 1;
```

3.2.5 Результати

- Створена база даних відповідає вимогам безпеки та масштабованості.
- Всі таблиці та зв’язки успішно інтегровані в інформаційну систему.

Ця технічна реалізація забезпечує основу для ефективного контролю доступу до системи та дозволяє легко адаптувати її до нових потреб.

3.3 Створення API для роботи з базою даних

Розробка API була здійснена із використанням платформи Next.js, яка дозволяє створювати серверні API-ендпоінти безпосередньо у проєкті. Цей підхід забезпечує інтеграцію між серверною та клієнтською частинами системи, спрощує структуру проєкту та підвищує швидкість розробки.

У цьому розділі наведено деталі реалізації API, яке відповідає за обробку запитів до бази даних системи, що забезпечує роботу з користувачами, ролями та правами доступу.

3.3.1 Загальна структура API

Для реалізації API використано модульну структуру директорій, що спрощує організацію коду та підвищує зручність його підтримки. Усі файли API

розміщені у директорії /api, яка містить піддиректорії, розділені за логічною структурою функціоналу.

1. Директорія auth

   Директорія, яка містить API-ендпоінти, пов'язані з авторизацією та аутентифікацією користувачів:

   - sign-in/route.ts: Обробка запитів для входу користувача. Реалізовано методи для перевірки облікових даних.
   - sign-up/route.ts: Ендпоінт для реєстрації нових користувачів. Реалізовано метод для додавання запису в базу даних.

2. Директорія roles

   Містить ендпоінт для роботи з ролями користувачів у системі:

   - route.ts: Обробка запитів для створення, оновлення, видалення та отримання даних про ролі.

3. Директорія users

   Містить API для управління даними користувачів:

   - [userId]/route.ts: Динамічний ендпоінт для роботи з конкретним користувачем за його ідентифікатором.
   - route.ts: Ендпоінт для роботи зі списком користувачів: отримання всіх користувачів, створення нових, оновлення та видалення.

4. Додаткові особливості

   У кожній директорії використовується файл route.ts, який є точкою входу для запитів до певного ресурсу. Логіка API зосереджена у файлах із назвою route.ts, що відповідає RESTful підходу до побудови API.

   Переваги такої структури:

   1. Модульність: Кожна функціональна частина API ізольована в окремій директорії.
   2. Легкість у підтримці: Зручне додавання нових ендпоінтів та модифікація існуючих.

3. Чіткість ієрархії: Користувачі, ролі та авторизація мають окремі зони відповідальності.

Ця структура забезпечує масштабованість проєкту та спрощує його подальший розвиток

3.3.2 Опис файлів API із маршрутами та прикладами відповідей

У цьому підрозділі надається докладний опис функціоналу кожного файлу API, який використовується у системі. Особлива увага приділяється їхньому призначенню, основним методам роботи та інтеграції з іншими компонентами системи. Наведені приклади коду ілюструють ключові аспекти реалізації логіки обробки запитів, демонструючи, як кожен файл виконує свою роль у забезпеченні роботи інтелектуальної системи контролю доступу. Кожен файл розроблений таким чином, щоб відповідати за обробку конкретного набору функцій, що дозволяє чітко розподілити відповідальності між модулями та забезпечити легкість підтримки і розширення системи.

Файл: api/auth/sign-in/route.ts

Зміст файлу

```
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { getClient } from 'src/app/db/client';
export async function POST(request: Request) {
  const client = getClient();
  await client.connect();
  const { email, password } = await request.json();
  if (!email || !password) {
    return new Response(
      JSON.stringify({ success: false, message: "Електронна пошта та пароль є обов'язковими" }),
```

```
    { status: 400 }
  );
}
try {
  const res = await client.query('SELECT * FROM users WHERE email = $1', [email]);
  if (res.rows.length === 0) {
    return new Response(
      JSON.stringify({ success: false, message: 'Неправильна електронна пошта або пароль' }),
      { status: 401 }
    );
  }
  const user = res.rows[0];
  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid) {
    return new Response(
      JSON.stringify({ success: false, message: 'Неправильна електронна пошта або пароль' }),
      { status: 401 }
    );
  }
  const token = jwt.sign({ id: user.id, email: user.email }, 'your_jwt_secret', { expiresIn: '1h' });
  return new Response(
    JSON.stringify({ success: true, message: 'Успішний вхід', token }),
    { status: 200 }
  );
} catch (error) {
  console.error('Error during sign-in:', error);
  return new Response(
    JSON.stringify({ success: false, message: 'Сталася внутрішня помилка сервера' }),
```

```
      { status: 500 }
    );
  } finally {
    await client.end();
  }
}
```

Опис файлу

1. Призначення:

   Реалізує маршрут для входу користувачів за допомогою електронної пошти та пароля. Обробляє запити, перевіряє облікові дані користувача в базі даних, порівнює хешований пароль та створює токен JWT.

2. Імпортовані модулі:
   - bcrypt: для хешування та перевірки паролів.
   - jsonwebtoken: для створення токенів JWT.
   - getClient: для підключення до бази даних.
3. Алгоритм роботи:
   - Підключення до бази даних.
   - Отримання даних із запиту (email і пароль).
   - Перевірка коректності даних.
   - SQL-запит для пошуку користувача в таблиці users.
   - Перевірка пароля за допомогою bcrypt.compare.
   - Генерація токена JWT при успішній автентифікації.
   - Повернення відповідних статусів та повідомлень.

Маршрут: POST /api/auth/sign-in

Вхідні дані:

```
{
  "email": "user@example.com",
```

"password": "userPassword123"

}

Приклади відповідей:

Успішний запит:

{

"success": true,

"message": "Успішний вхід",

"token": "eyJhbGciOiJIUzI1NiIsInR..."

}

Некоректні дані:

{

"success": false,

"message": "Електронна пошта та пароль є обов'язковими"

}

Невірний пароль:

{

"success": false,

"message": "Неправильна електронна пошта або пароль"

}

Помилка сервера:

{

"success": false,

"message": "Сталася внутрішня помилка сервера"

}

Використання

Цей маршрут служить основою для автентифікації користувачів у системі та є критичним для захисту даних і доступу до приватних ресурсів.

Файл: api/auth/sign-up/route.ts

Зміст файлу

```
import bcrypt from 'bcrypt';
import { getClient } from 'src/app/db/client';
export async function POST(request: Request) {
 const client = getClient();
 await client.connect();
 const { username, email, password, role_id } = await request.json();
 if (!username || !email || !password || !role_id) {
  await client.end();
  return new Response(
   JSON.stringify({
    success: false,
    message: 'Email and password and User Name are required',
   }),
   { status: 400 }
  );
 }
 try {
  const hashedPassword = await bcrypt.hash(password, 10);
  await client.query(
   'INSERT INTO users (username, email, password, role_id) VALUES ($1, $2, $3, $4)',
```

```
      [username, email, hashedPassword, role_id]
    );
    return new Response(
      JSON.stringify({
        success: true,
        message: 'User created successfully',
      }),
      { status: 201 }
    );
  } catch (error) {
    console.error('Error during sign-up:', error);
    return new Response(
      JSON.stringify({
        success: false,
        message: 'Error creating user',
      }),
      { status: 500 }
    );
  } finally {
    await client.end();
  }
}
```

Опис файлу

1. Призначення:

   Реалізує маршрут для реєстрації нових користувачів у системі. Маршрут отримує дані користувача, хешує пароль, і додає запис у таблицю users бази даних.

2. Імпортовані модулі:
    - bcrypt: для хешування паролів.
    - getClient: для підключення до бази даних.
3. Алгоритм роботи:
    - Підключення до бази даних.
    - Отримання та перевірка обов'язкових даних (ім'я користувача, електронна пошта, пароль, роль).
    - Хешування пароля за допомогою bcrypt.
    - SQL-запит для додавання нового користувача в базу даних.
    - Закриття підключення до бази даних.

Маршрут: POST /api/auth/sign-up

Вхідні дані:

```
{
  "username": "newuser",
  "email": "newuser@example.com",
  "password": "securePassword123",
  "role_id": 2
}
```

Приклади відповідей:

Успішна реєстрація:

```
{
  "success": true,
  "message": "User created successfully"
}
```

Некоректні дані:

```
{
  "success": false,
```

```
 "message": "Email and password and User Name are required"
}
```

Помилка сервера:

```
{
 "success": false,
 "message": "Error creating user"
}
```

Використання

Цей маршрут є ключовим для створення нових користувачів і управління їх доступом у системі. Використання хешування паролів гарантує безпеку облікових записів.

Безпека

- Хешування паролів через bcrypt із використанням сольового значення для запобігання злому.
- Перевірка вхідних даних, щоб уникнути помилок чи несанкціонованих змін у базі даних.

Файл: api/roles/route.ts

Зміст файлу

```
import { getClient } from 'src/app/db/client';
export async function GET(request: Request) {
 const client = getClient();
 await client.connect();
 try {
   const res = await client.query('SELECT * FROM roles');
   return new Response(JSON.stringify({ success: true, roles: res.rows }), {
     status: 200,
```

```
    });
  } catch (error) {
    console.error('Error fetching roles:', error);
    return new Response(JSON.stringify({ success: false, message: 'Internal server error' }), {
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

Опис файлу

1. Призначення:

   Реалізує маршрут для отримання списку всіх ролей у системі. Запит виконується до бази даних, і результати повертаються клієнту.

2. Імпортовані модулі:
   - getClient: для підключення до бази даних PostgreSQL.
3. Алгоритм роботи:
   - Підключення до бази даних через функцію getClient.
   - Виконання SQL-запиту для отримання всіх записів із таблиці roles.
   - Повернення списку ролей у форматі JSON.
   - У разі помилки повертається відповідне повідомлення.

Маршрут: GET /api/roles

Вхідні дані: Запит не потребує параметрів.

Приклади відповідей:

Успішний запит:

```
{
```

```
 "success": true,
 "roles": [
   { "id": 1, "name": "Admin" },
   { "id": 2, "name": "User" }
 ]
}
```


Помилка сервера:


```
{
 "success": false,
 "message": "Внутрішня помилка сервера"
}
```


Використання

Цей маршрут використовується для отримання списку всіх ролей у системі. Це корисно для адміністративних функцій, таких як створення чи оновлення користувачів, і допомагає інтегрувати контроль доступу в системі.

Файл: api/users/route.ts

Зміст файлу

```
import { getClient } from 'src/app/db/client';
export async function GET(request: Request) {
 const client = getClient();
 await client.connect();
 try {
   const res = await client.query(`
     SELECT
       users.user_id,
       users.username,
       users.email,
```

```
      users.is_active,
      roles.role_id,
      roles.role_name
    FROM users
    LEFT JOIN user_roles ON users.user_id = user_roles.user_id
    LEFT JOIN roles ON user_roles.role_id = roles.role_id
    ORDER BY users.user_id
  `);
  const users = res.rows;
  return new Response(
    JSON.stringify({
      users,
    }),
    { status: 200 }
  );
 } catch (error) {
  console.error('Помилка входу користувача:', error);
  return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
    status: 500,
  });
 } finally {
  await client.end();
 }
}
```

Опис файлу

1. Призначення:

Реалізує маршрут для отримання списку користувачів разом із пов'язаними ролями.

2. Імпортовані модулі:
    - getClient: для підключення до бази даних.
3. Алгоритм роботи:
    - Підключення до бази даних.
    - SQL-запит для отримання списку користувачів з їхніми ролями:
    - Вибираються поля user_id, username, email, is_active з таблиці users.
    - Виконується приєднання таблиць user_roles і roles для отримання ідентифікатора та назви ролі.
    - Результати запиту сортуються за user_id.
    - Повертається JSON-відповідь із даними користувачів.

Маршрут: GET /api/users

Вхідні дані:

Метод GET: Ніяких параметрів не потрібно.

Приклади відповідей:

Успішна відповідь:


```
{
 "success": true,
 "users": [
   {
    "user_id": 1,
    "username": "admin",
    "email": "admin@example.com",
    "is_active": true,
    "role_id": 1,
```

```
    "role_name": "Administrator"
   },
   {
    "user_id": 2,
    "username": "user1",
    "email": "user1@example.com",
    "is_active": false,
    "role_id": 2,
    "role_name": "User"
   }
 ]
}
```


Помилка сервера:


```
{
 "success": false,
 "message": "Внутрішня помилка сервера"
}
```


Використання

Цей маршрут використовується для перегляду списку користувачів системи, включаючи їх статус та ролі. Це корисно для адміністрування системи та управління доступом.

Файл: api/users/[userId]/route.ts

Зміст файлу

```
import { getClient } from 'src/app/db/client';
export async function PUT(request: Request, { params }: { params: { userId: string } }) {
 const client = getClient();
 await client.connect();
```

```
try {
  const { userId } = params;
  const { username, role_id } = await request.json();
  await client.query('BEGIN');
  const userRes = await client.query(
    'UPDATE users SET username = $1 WHERE user_id = $2 RETURNING *',
    [username, userId]
  );
  if (userRes.rowCount === 0) {
    await client.query('ROLLBACK');
    return new Response(JSON.stringify({ success: false, message: 'Користувача не знайдено' }), {
      status: 404,
    });
  }
  await client.query('DELETE FROM user_roles WHERE user_id = $1', [userId]);
  await client.query('INSERT INTO user_roles (user_id, role_id) VALUES ($1, $2)', [
    userId,
    role_id,
  ]);
  await client.query('COMMIT');
  return new Response(
    JSON.stringify({ success: true, message: 'Користувача успішно оновлено' }),
    {
      status: 200,
    }
  );
} catch (error) {
  await client.query('ROLLBACK');
```

```
    console.error('Error updating user:', error);
    return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
      status: 500,
    });
  } finally {
    await client.end();
  }
}
export async function DELETE(request: Request, { params }: { params: { userId: string } }) {
  const client = getClient();
  await client.connect();
  try {
    const { userId } = params;
    const res = await client.query('DELETE FROM users WHERE user_id = $1 RETURNING *', [userId]);
    if (res.rowCount === 0) {
      return new Response(JSON.stringify({ success: false, message: 'Користувача не знайдено' }), {
        status: 404,
      });
    }
    return new Response(
      JSON.stringify({ success: true, message: 'Користувача успішно видалено' }),
      {
        status: 200,
      }
    );
  } catch (error) {
    console.error('Error deleting user:', error);
    return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
```

```
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

Опис файлу

1. Призначення:

   Реалізує маршрути для оновлення (PUT) та видалення (DELETE) користувачів у системі.

2. Метод PUT:
   - Функціональність: Оновлення імені користувача (username) та його ролі (role_id).
   - Алгоритм:
     - Починається транзакція.
     - Оновлюється таблиця users.
     - Оновлюється таблиця user_roles.
     - У разі успіху транзакція завершується.
3. Метод DELETE:
   - Функціональність: Видалення користувача за userId.
   - Алгоритм:
     - Починається транзакція.
     - Видаляються записи з таблиці user_roles.
     - Видаляється запис користувача з таблиці users.
     - У разі успіху транзакція завершується.

PUT /api/users/[userId]

Вхідні дані:

{
"username": "Нове ім’я користувача",
"role_id": 3
}

Приклади відповідей:

Успішна відповідь:

{
"success": true,
"user": {
"user_id": 1,
"username": "new_username",
"email": "admin@example.com",
"is_active": true
}
}

Користувач не знайдений:

{
"success": false,
"message": "Користувача не знайдено"
}

DELETE /api/users/[userId]

Вхідні дані: Параметр userId в URL.

Приклади відповідей:

Успішна відповідь:

{
"success": true,

"message": "Користувача успішно видалено"

}

Користувач не знайдений:

{

"success": false,

"message": "Користувача не знайдено"

}

Помилка сервера:

{

"success": false,

"message": "Внутрішня помилка сервера"

}

Використання

Цей файл забезпечує маршрути для адміністрування користувачів. PUT дозволяє оновлювати дані користувача, а DELETE — видаляти користувача, забезпечуючи цілісність даних через транзакції.

3.3.3 Взаємодія API з інтерфейсом користувача

Розділ описує інтеграцію API з користувацьким інтерфейсом, створеним за допомогою Next.js, а також демонструє, як реалізовані сценарії роботи з системою через UI. Особливу увагу приділено обробці запитів до API, відображенню результатів та забезпеченню інтерактивності.

3.3.1 Архітектура взаємодії UI та API

Взаємодія між інтерфейсом користувача (UI) та API є ключовим аспектом роботи системи, що забезпечує обмін даними між клієнтською частиною (фронтендом) і сервером. У даній системі архітектура побудована на основі підходу клієнт-сервер, де UI створено за допомогою Next.js, а серверна частина працює з API, реалізованими у вигляді RESTful-інтерфейсів.

Основні компоненти архітектури:

1. Фронтенд (Next.js):
    - Роль: Фронтенд відповідає за збір, валідацію та представлення даних користувачеві. У Next.js поєднуються серверні (Server-Side Rendering) та клієнтські (Client-Side Rendering) функції, що забезпечує високу швидкість завантаження та зручність роботи.
    - Інструменти:
        - React для створення компонентів.
        - fetch/Axios для виконання HTTP-запитів до API.
        - State management бібліотеки (наприклад, Context API або Redux) для управління станом додатку.
2. API (Back-End):
    - Роль: API відповідає за отримання, обробку та зберігання даних у базі даних. Всі маршрути API організовано у вигляді REST-інтерфейсів із чітко визначеними методами (GET, POST, PUT, DELETE).
    - Інструменти:
        - Обробка HTTP-запитів у файлах API.
        - Підключення до бази даних через клієнт (наприклад, PostgreSQL).
        - Логіка безпеки (валидація даних, хешування паролів, токени JWT).

Загальна схема взаємодії:

1. Запит: UI відправляє запит до API через HTTP (наприклад, за допомогою fetch).

   Наприклад: коли користувач заповнює форму входу та натискає "Увійти", формується POST-запит на маршрут /api/auth/sign-in.
2. Обробка API: Сервер отримує запит, виконує валідацію даних, обробляє логіку (наприклад, перевіряє email і пароль) і формує відповідь.

   У випадку успіху відповідь може містити дані користувача або токен.
3. Відповідь: API повертає відповідь у вигляді JSON-даних. UI використовує ці дані для оновлення інтерфейсу.

   Наприклад: якщо вхід успішний, зберігається токен, а користувача перенаправляють на головну сторінку.
4. Оновлення інтерфейсу: UI оновлює стан і компоненти на основі отриманих даних. Це дозволяє відобразити повідомлення про успіх або помилку, завантажити додаткові дані тощо.

Потік даних на прикладі функції входу:

1. Користувач вводить email і пароль на сторінці входу.
2. Дані форми відправляються через запит POST /api/auth/sign-in до API.
3. API:
   - Перевіряє, чи введені дані.
   - Знаходить користувача в базі даних.
   - Порівнює пароль із хешованим значенням.
   - У разі успіху повертає JWT токен.
4. UI отримує токен, зберігає його у локальному сховищі (localStorage) та перенаправляє користувача на сторінку профілю.

Структура інтеграції API в UI:

1. Клієнтська логіка (UI):
   - Форма збору даних: Користувач взаємодіє з елементами інтерфейсу (поля форми, кнопки).

- Запити до API: Дані відправляються через запит (fetch) і чекають на відповідь.

2. Обробка відповіді:
3. Успішний запит: Дані інтегруються у стан додатку, наприклад:
    - Відображається список користувачів.
    - Зберігається токен для подальшої роботи.
4. Помилка: Інтерфейс повідомляє користувача про проблему (наприклад, "Невірний пароль").
5. Синхронізація стану: Використання бібліотек для глобального стану (Context API, Redux) дозволяє синхронізувати зміни в інтерфейсі.

Особливості архітектури:

1. Серверна та клієнтська рендеринг: Next.js дозволяє виконувати рендеринг як на сервері (для SEO та швидкого завантаження), так і на клієнті (для динамічних компонентів).
2. Розділення відповідальності: Логіка бізнес-процесів зосереджена на сервері, тоді як клієнтська частина відповідає лише за зручне представлення даних.
3. Безпека:
    - Дані передаються через захищене з'єднання (HTTPS).
    - Токени JWT використовуються для аутентифікації.
    - Валідація даних проводиться на обох рівнях: UI та API.
4. Масштабованість: Завдяки модульності Next.js і RESTful API, архітектура легко масштабується, дозволяючи додавати нові функції.

3.3.3 Приклади інтеграції API в UI

Сторінка “Вхід у систему” надає форму входу дозволяє користувачам системи авторизуватися, використовуючи свої облікові дані (email та пароль). Її

функціонал включає валідацію введених даних, обробку помилок та перенаправлення до особистого кабінету після успішного входу.

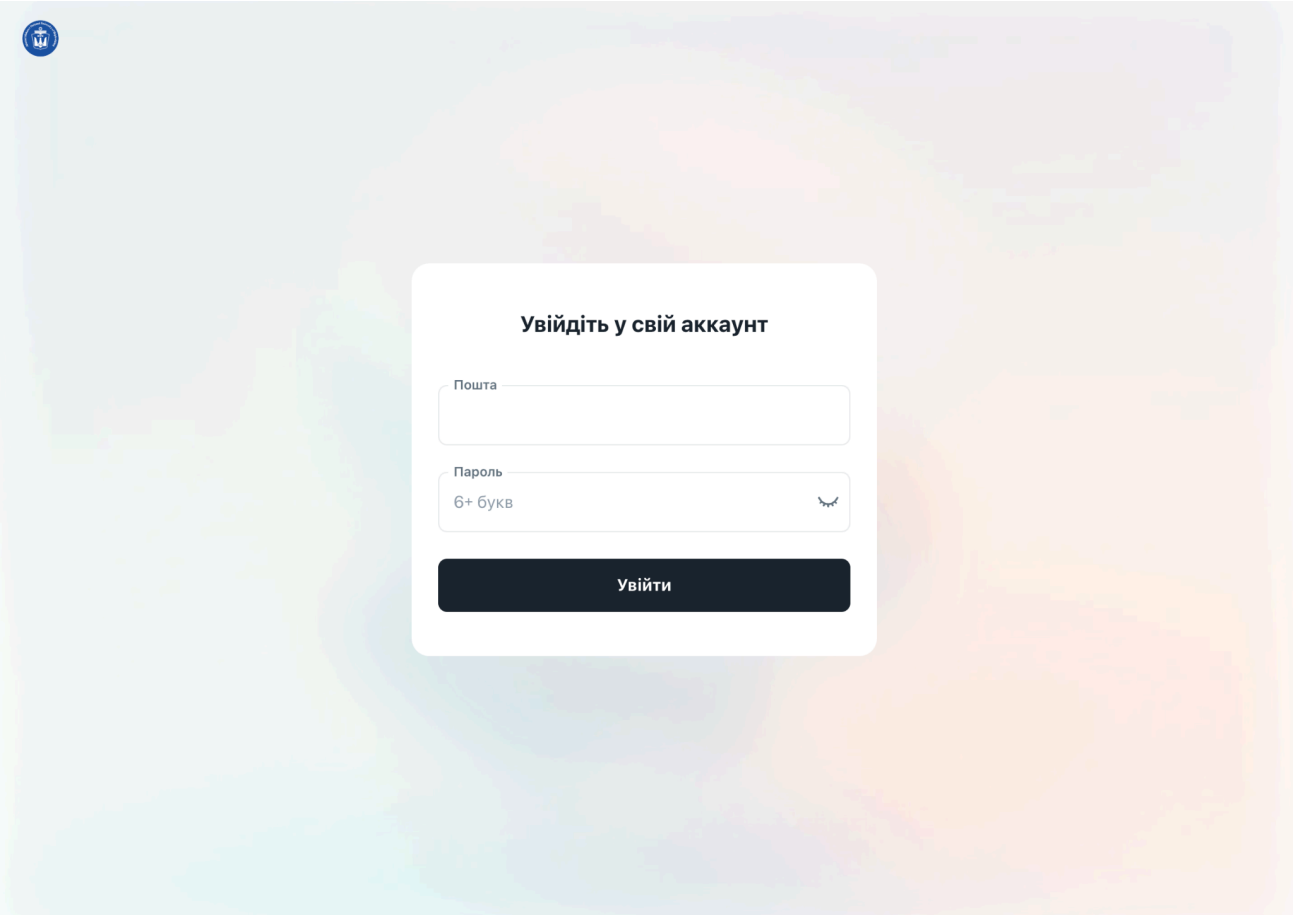

Рисунок 3.1 - Сторінка “Вхід у систему”

Структура та логіка компонента

- Компонент CenteredSignInView розташований у файлі SignIn.tsx. Основні функції:
- Збір облікових даних (email і пароль).
- Клієнтська валідація введених даних.
- Відправка запиту до API для перевірки даних.
- Логування помилок і виведення відповідних повідомлень.

Для валідації даних використовується бібліотека zod. Схема валідації:

```
export const SignInSchema = zod.object({
  email: zod
    .string()
    .min(1, { message: 'Email is required!' })
```

```
    .email({ message: 'Email must be a valid email address!' }),
  password: zod
    .string()
    .min(1, { message: 'Password is required!' })
    .min(6, { message: 'Password must be at least 6 characters!' }),
});
```

Особливості:

- Email не може бути порожнім і повинен відповідати стандартному формату.
- Пароль повинен містити щонайменше 6 символів.

Рендеринг форми: Форма складається з двох основних полів: для введення email та пароля.

Поле для введення email:

```
<Field.Text name="email" label="Пошта" InputLabelProps={{ shrink: true }} />
```

Поле для введення пароля з можливістю перегляду символів:

```
<Field.Text
  name="password"
  label="Пароль"
  placeholder="6+ букв"
  type={password.value ? 'text' : 'password'}
  InputLabelProps={{ shrink: true }}
  InputProps={{
    endAdornment: (
      <InputAdornment position="end">
        <IconButton onClick={password.onToggle} edge="end">
          <Iconify icon={password.value ? 'solar:eye-bold' : 'solar:eye-closed-bold'} />
        </IconButton>
      </InputAdornment>
    ),
  }}
/>
```

Кнопка "Увійти":

```
<LoadingButton
  fullWidth
  color="inherit"
  size="large"
  type="submit"
```

```
  variant="contained"
  loading={isSubmitting}
  loadingIndicator="Входить..."
>
  Увійти
</LoadingButton>
```

Відправка даних на сервер

Після натискання кнопки "Увійти" форма відправляє запит через метод onSubmit:

```
const onSubmit = handleSubmit(async (data) => {
  try {
    const response = await axios.post('/api/auth/sign-in', data);
    toast.success('Успішний вхід');
    router.replace('/nuos/dashboard');
  } catch (error) {
    console.error('Error signing in user:', error);
    toast.error(error.response.data.message || 'Помилка входу');
  }
});
```

Запит: Надсилається запит POST /api/auth/sign-in з введеними даними:

```
{
  "email": "test@example.com",
  "password": "password123"
}
```

Відповідь: У разі успіху:

```
{
  "success": true,
  "message": "Успішний вхід",
  "token": "eyJhbGciOiJIUzI1Ni..."
}
```

4. Обробка відповіді

Успішний вхід:

Виводиться повідомлення про успішний вхід: toast.success('Успішний вхід');

Користувач перенаправляється до особистого кабінету: router.replace('/nuos/dashboard');

Помилка входу: У разі помилкових облікових даних сервер повертає помилку, яка відображається через toast: toast.error(error.response.data.message || 'Помилка входу');

Інтеграція з API

Маршрут API: POST /api/auth/sign-in

Функціональність: Перевірка email та пароля в базі даних.

- У разі успіху: генерація токена JWT та повернення користувачеві.
- У разі помилки: повідомлення про невірні дані.

Форма входу реалізована з урахуванням усіх необхідних вимог:

- Забезпечується надійна клієнтська валідація.
- Взаємодія з сервером здійснюється через безпечний API-запит.
- Відображення відповідних повідомлень про успіх або помилку робить процес входу зручним для користувачів.

Цей компонент є ключовим для доступу користувачів до системи та демонструє інтеграцію UI та API на високому рівні.

Сторінка "Список користувачів" надає адміністраторам системи можливість переглядати список зареєстрованих користувачів, а також виконувати базові операції, такі як:

- Додавання нового користувача.
- Редагування існуючого користувача.
- Видалення користувача.

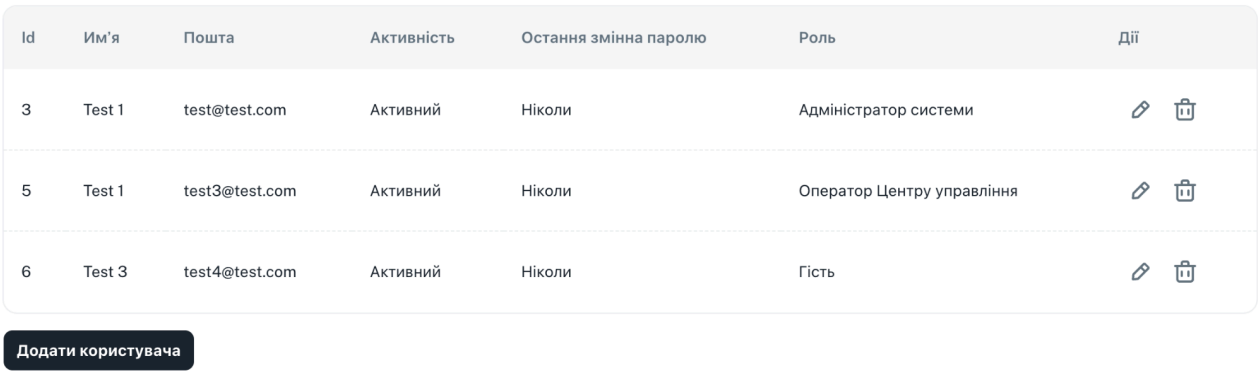

Рисунок 3.2 - Список користувачів

Сторінка створена за допомогою бібліотеки Material-UI, що забезпечує зручні та адаптивні компоненти. Функціональність реалізована в компоненті UsersList.

Таблиця з користувачами

Компонент таблиці відображає основні поля:

- Id — унікальний ідентифікатор користувача.
- Ім'я — ім'я користувача.
- Пошта — email-адреса користувача.
- Активність — статус (активний чи неактивний).
- Остання зміна паролю — інформація про останнє оновлення паролю.
- Роль — роль користувача в системі.
- Дії — кнопки для редагування або видалення користувача.

Кнопка "Додати користувача"

Розташована під таблицею. Використовується для відкриття модального вікна, де можна додати нового користувача.

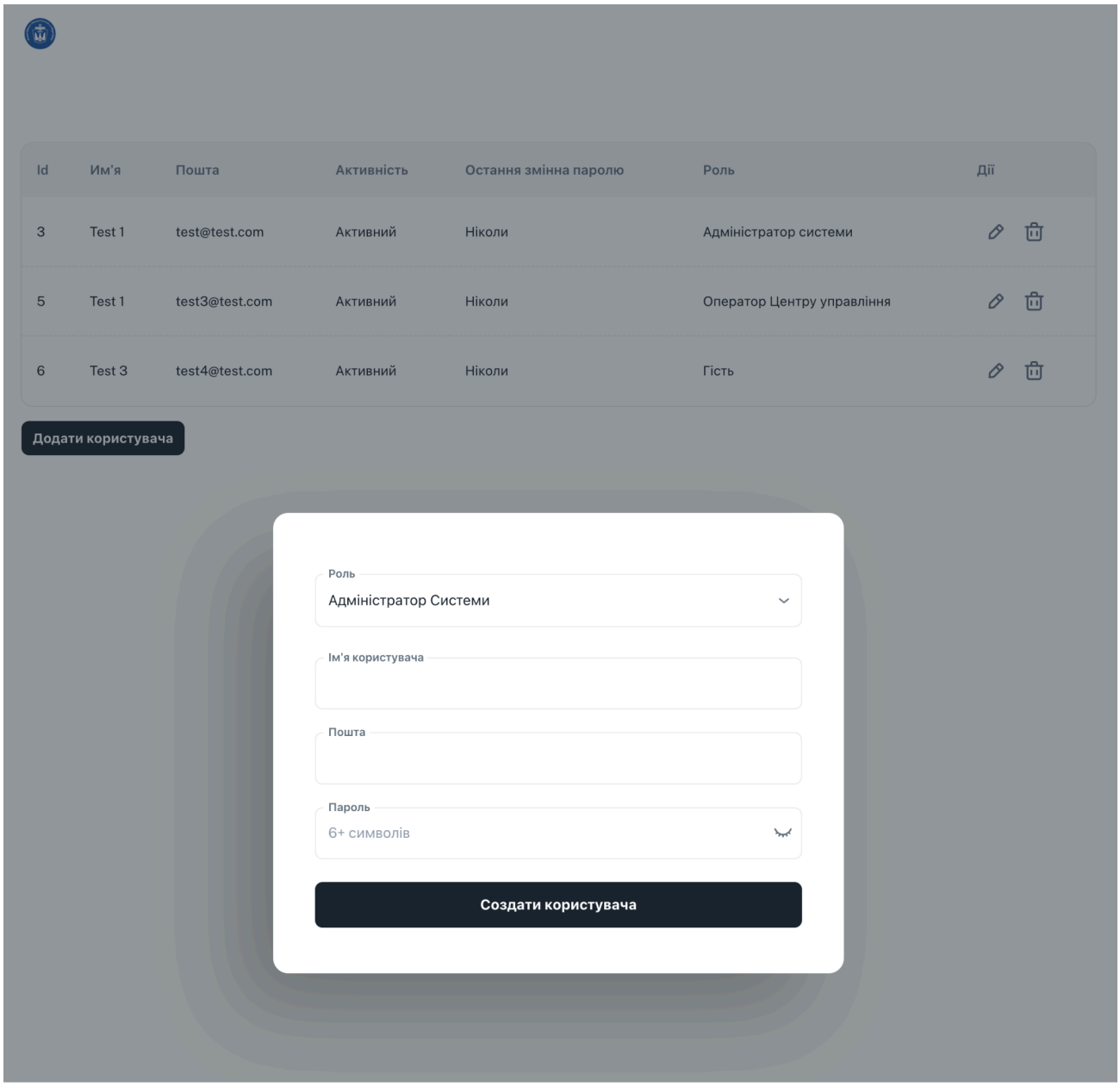

Рисунок 3.3 - Модальне вікно для додовання користувача

Модальне вікно для редагування користувача

Модальне вікно з полями:

- Ім'я — поле для вводу або зміни імені.
- Роль — випадаючий список із доступними ролями.

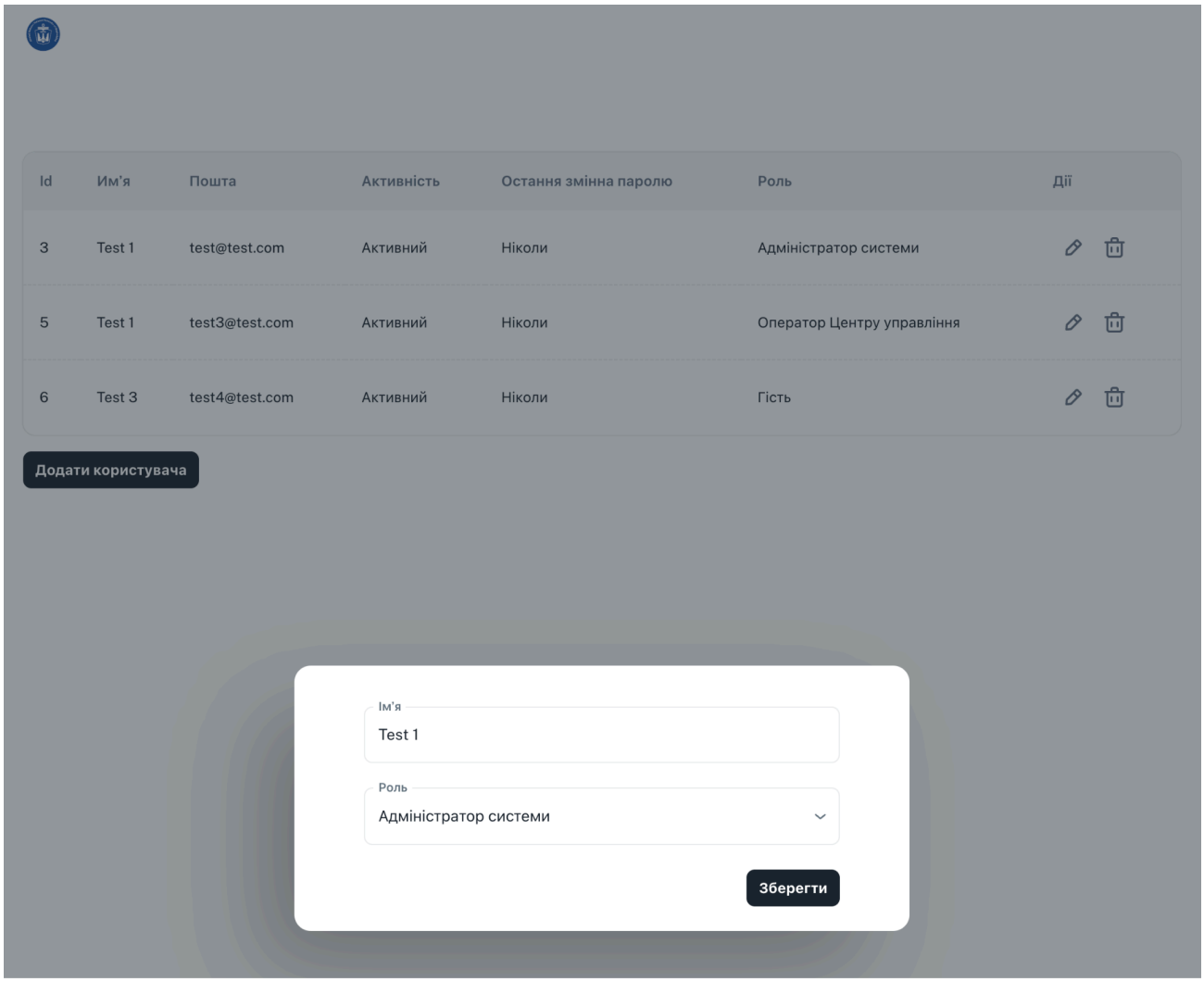

Рисунок 3.4 - Модальне вікно для редагування користувача

API-взаємодії

Використовуються наступні маршрути API:

- GET /api/users Отримує список користувачів для заповнення таблиці.
- GET /api/roles Завантажує список ролей, доступних для користувачів.
- POST /api/auth/sign-up Використовується для створення нового користувача.
- PUT /api/users/:userId Оновлює дані існуючого користувача.
- DELETE /api/users/:userId Видаляє користувача із системи.

Приклади функціональності

Отримання та відображення списку користувачів

Код у useEffect завантажує дані про користувачів та ролі:

```
useEffect(() => {
  const fetchData = async () => {
    const [usersResponse, rolesResponse] = await Promise.all([
      axios.get('/api/users'),
      axios.get('/api/roles'),
    ]);
    setUsers(usersResponse.data?.users);
    setRoles(rolesResponse.data?.roles.sort((a: any, b: any) => a.role_id - b.role_id));
  };
  fetchData();
}, []);
```

Дані потім відображаються у таблиці через метод map:

```
{users.map((user: any) => (
  <TableRow key={user.user_id}>
    <TableCell>{user.user_id}</TableCell>
    <TableCell>{user.username}</TableCell>
    <TableCell>{user.email}</TableCell>
    <TableCell>{user.is_active ? 'Активний' : 'Не активний'}</TableCell>
    <TableCell>
      {user.password_changed_at ? user.password_changed_at : 'Ніколи'}
    </TableCell>
    <TableCell>{user.role_name}</TableCell>
    <TableCell>
      <IconButton onClick={() => handleEditUser(user)}>
        <Iconify width={24} icon="eva:edit-outline" />
      </IconButton>
      <IconButton onClick={() => handleRemoveUser(user.user_id)}>
        <Iconify width={24} icon="eva:trash-2-outline" />
      </IconButton>
    </TableCell>
  </TableRow>
))}
```

Додавання нового користувача

При натисканні на кнопку "Додати користувача" відкривається модальне вікно, яке викликає функцію handleSaveUser:

```
const handleSaveUser = async (data: any) => {
  try {
    const response = await axios.post('/api/auth/sign-up', data);
    toast.success('Користувача створено');
```

```
    setUsers([...users, response.data.user]);
    setOpenDialog(false);
  } catch (error) {
    console.error('Error signing in user:', error);
    toast.error(error.response.data.message || 'Помилка створення користувача');
  }
};
```

Редагування користувача

Кнопка "редагувати" відкриває модальне вікно з поточними даними користувача:

```
const handleEditUser = (user: any) => {
  setEditUser(user);
  setOpenDialog(true);
};
```

Після внесення змін функція handleSaveEditUser відправляє запит на оновлення:

```
const handleSaveEditUser = async () => {
  await axios.put(`/api/users/${editUser.user_id}`, editUser);
  setUsers(users.map((user: any) => (user.user_id === editUser.user_id ? editUser : user)));
  setOpenDialog(false);
};
```

Видалення користувача

Кнопка "видалити" викликає handleRemoveUser, яка надсилає запит DELETE:

```
const handleRemoveUser = async (userId: string) => {
  await axios.delete(`/api/users/${userId}`);
  setUsers(users.filter((user: any) => user.user_id !== userId));
};
```

Сторінка "Список користувачів" реалізує всі необхідні функції для роботи з користувачами системи:

- Інтеграція з API дозволяє динамічно оновлювати дані.
- Інтуїтивно зрозумілий UI забезпечує легкість використання.
- Функціональність модальних вікон спрощує взаємодію з користувачами.

В результаті маємо наступний код

```
'use client';

import axios from 'axios';
import { useState, useEffect } from 'react';
```

```
import {
  Box,
  Table,
  Dialog,
  Button,
  TableRow,
  MenuItem,
  TableBody,
  TableCell,
  Container,
  TextField,
  IconButton,
} from '@mui/material';
import { toast } from 'src/components/snackbar';
import { ComponentBlock } from 'src/sections/_examples/component-block';
import { SignUp } from './SignUp';
import { Iconify } from '../iconify';
import { Scrollbar } from '../scrollbar';
import { TableHeadCustom } from '../table';
const blockProps = {
  p: 0,
  overflow: 'hidden',
  alignItems: 'unset',
  flexDirection: 'column',
  bgcolor: 'background.paper',
};
const TABLE_HEAD = [
  { id: 'user_id', label: 'Id' },
  { id: 'user_name', label: 'Им’я' },
  { id: 'email', label: 'Пошта' },
  { id: 'is_active', label: 'Активність' },
  { id: 'password_changed_at', label: 'Остання змінна паролю' },
  { id: 'role_name', label: 'Роль' },
  { id: 'actions', label: 'Дії' },
];
export function UsersList() {
  const [users, setUsers] = useState([]);
  const [roles, setRoles] = useState([]);
  const [openDialog, setOpenDialog] = useState(false);
```

```
const [editUser, setEditUser] = useState<any>(null);
useEffect(() => {
  const fetchData = async () => {
    const [usersResponse, rolesResponse] = await Promise.all([
      axios.get('/api/users'),
      axios.get('/api/roles'),
    ]);
    setUsers(usersResponse.data?.users);
    setRoles(rolesResponse.data?.roles.sort((a: any, b: any) => a.role_id - b.role_id));
  };
  fetchData();
}, []);
const handleRemoveUser = async (userId: string) => {
  await axios.delete(`/api/users/${userId}`);
  setUsers(users.filter((user: any) => user.user_id !== userId));
};
const handleCloseDialog = () => {
  setOpenDialog(false);
};
const handleAddUser = (user: any) => {
  setEditUser(null);
  setOpenDialog(true);
};
const handleEditUser = (user: any) => {
  setEditUser(user);
  setOpenDialog(true);
};
const handleSaveEditUser = async () => {
  await axios.put(`/api/users/${editUser.user_id}`, editUser);
  setUsers(users.map((user: any) => (user.user_id === editUser.user_id ? editUser : user)));
  setOpenDialog(false);
};
const handleSaveUser = async (data: any) => {
  try {
    const response = await axios.post('/api/auth/sign-up', data);
    toast.success('Користувача створено');
    setUsers([...users, response.data.user]);
    setOpenDialog(false);
  } catch (error) {
    console.error('Error signing in user:', error);
```

```
    toast.error(error.response.data.message || 'Помилка створення користувача');
  }
};
return (
  <Container sx={{ mt: 10, mb: 15 }}>
    <ComponentBlock sx={blockProps}>
      <Scrollbar>
        <Table sx={{ minWidth: 800 }}>
          <TableHeadCustom headLabel={TABLE_HEAD} />
          <TableBody>
            {users.map((user: any) => (
              <TableRow key={user.user_id}>
                <TableCell>{user.user_id}</TableCell>
                <TableCell>{user.username}</TableCell>
                <TableCell>{user.email}</TableCell>
                <TableCell>{user.is_active ? 'Активний' : 'Не активний'}</TableCell>
                <TableCell>
                  {user.password_changed_at ? user.password_changed_at : 'Ніколи'}
                </TableCell>
                <TableCell>{user.role_name}</TableCell>
                <TableCell>
                  <IconButton onClick={() => handleEditUser(user)}>
                    <Iconify width={24} icon="eva:edit-outline" />
                  </IconButton>
                  <IconButton onClick={() => handleRemoveUser(user.user_id)}>
                    <Iconify width={24} icon="eva:trash-2-outline" />
                  </IconButton>
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </Scrollbar>
    </ComponentBlock>
    <Box mt={2}>
      <Button variant="contained" onClick={handleAddUser}>
        Додати користувача
      </Button>
    </Box>
    <Dialog open={openDialog} onClose={handleCloseDialog} fullWidth maxWidth="sm">
```

```
<Box>
  <Box ml="auto" mr="auto" maxWidth="512px">
    {editUser && (
      <Box p={3}>
        <TextField
          fullWidth
          label="Ім’я"
          value={editUser.username}
          onChange={(e) => setEditUser({ ...editUser, username: e.target.value })}
          margin="normal"
        />
        <TextField
          fullWidth
          select
          label="Роль"
          value={editUser.role_id}
          onChange={(e) =>
            setEditUser({
              ...editUser,
              role_id: e.target.value,
              role_name: roles.find((role: any) => role.role_id === e.target.value)
                .role_name,
            })
          }
          margin="normal"
        >
          {roles.map((role: any) => (
            <MenuItem key={role.role_id} value={role.role_id}>
              {role.role_name}
            </MenuItem>
          ))}
        </TextField>
        <Box mt={2} display="flex" justifyContent="flex-end">
          <Button variant="contained" onClick={handleSaveEditUser}>
            Зберегти
          </Button>
        </Box>
      </Box>
    )}
    {!editUser && <SignUp onSave={handleSaveUser} roles={roles} />}
```

```
        </Box>
      </Box>
    </Dialog>
  </Container>
 );
}
```

Додавання нового користувача

Модальне вікно "Додати користувача" є важливим компонентом адміністративної сторінки для управління користувачами. Його функціонал дозволяє адміністратору створювати нові облікові записи з обраною роллю, дотримуючись заданих правил валідації.

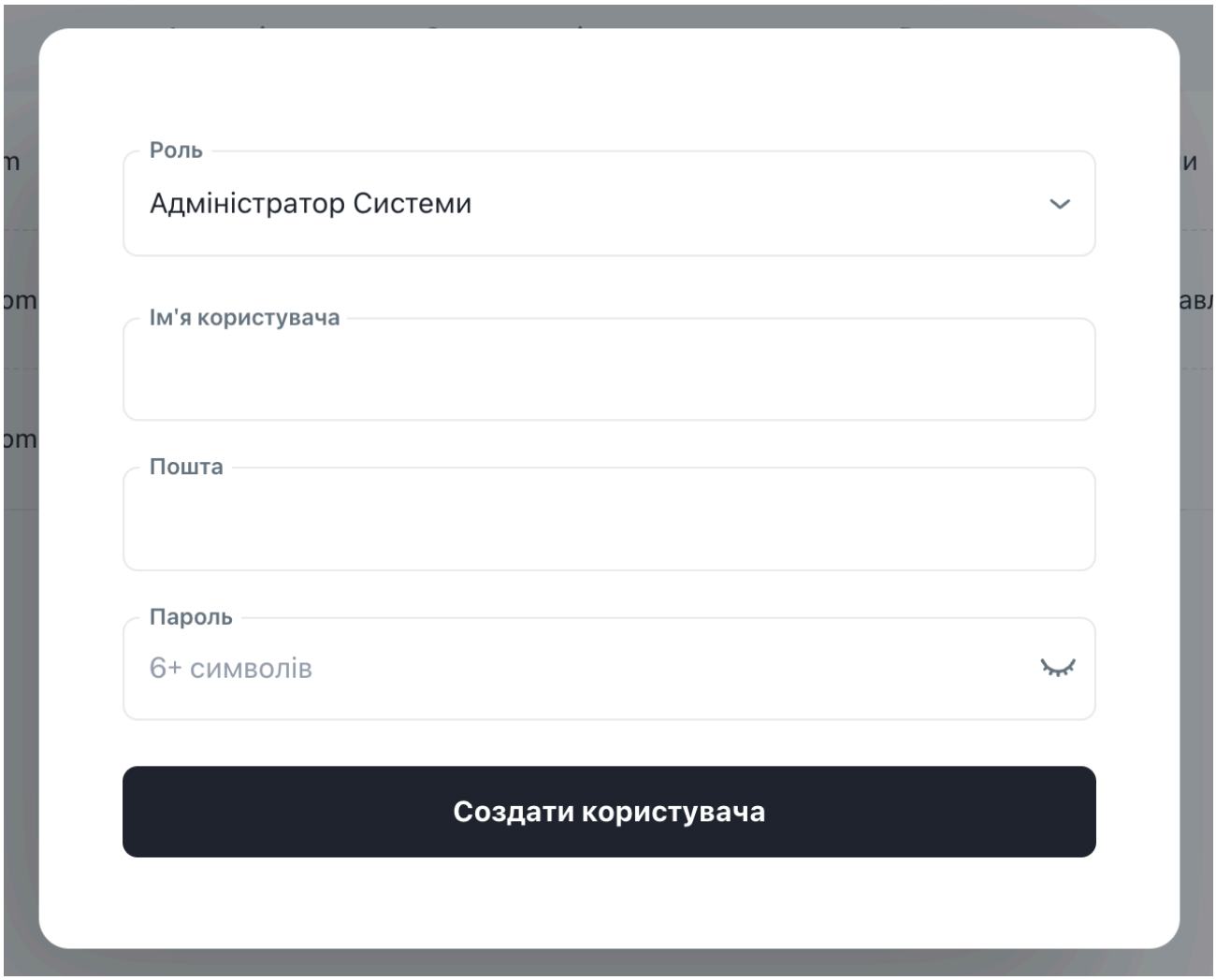

Рисунок 3.5 - Модальне вікно для редагування користувача

Структура та логіка компонента

Компонент SignUp розташований у файлі SignUp.tsx. Основні функції компонента:

- Збір інформації про нового користувача (роль, ім'я, email, пароль).
- Валідація введених даних на клієнтській стороні.
- Відправка запиту на сервер через API.

Валідація даних: Для валідації даних використовується бібліотека zod. Опис схеми:

```
export const SignUpSchema = zod.object({
  role_id: zod.number(),
  username: zod.string().min(1, { message: "Ім'я користувача" }),
  email: zod.string().min(1, { message: "Пошта обов'язково" }).email({ message: 'Не вірна пошта' }),
  password: zod
    .string()
    .min(1, { message: "Пароль обов'язковий" })
    .min(6, { message: 'Довжина паролю має бути більше 6 символів' }),
});
```

Ця схема гарантує, що:

- Поля не можуть бути порожніми.
- Email повинен мати правильний формат.
- Пароль має містити щонайменше 6 символів.

Рендеринг форми: Форма використовує компоненти Field для створення текстових полів і випадаючого списку для вибору ролі. Наприклад:

Вибір ролі:

```
<Field.Select
  fullWidth
  name="role_id"
  label="Роль"
  margin="normal"
>
  {roles.map((role: any) => (
    <MenuItem key={role.role_id} value={role.role_id}>
      {role.role_name}
    </MenuItem>
  ))}
</Field.Select>
```

Поле для пароля з можливістю відображення символів:

```
<Field.Text
  name="password"
  label="Пароль"
  placeholder="6+ символів"
  type={password.value ? 'text' : 'password'}
  InputProps={{
    endAdornment: (
      <InputAdornment position="end">
        <IconButton onClick={password.onToggle} edge="end">
          <Iconify icon={password.value ? 'solar:eye-bold' : 'solar:eye-closed-bold'} />
        </IconButton>
      </InputAdornment>
    ),
  }}
/>
```

Відправка даних

Після натискання кнопки "Создати користувача" форма відправляє дані через метод onSubmit:

```
const onSubmit = handleSubmit(async (data) => {
  try {
    onSave(data);
  } catch (error) {
    console.error(error);
  }
});
```

Функція onSave надсилає POST-запит на маршрут POST /api/auth/sign-up із даними форми.

Взаємодія з API

Запит: Дані нового користувача передаються на сервер:

```
{
  "role_id": 1,
  "username": "Test User",
  "email": "test@example.com",
  "password": "password123"
}
```

Обробка на сервері: Сервер перевіряє дані, створює користувача в базі даних і повертає результат.

Відповідь:

У разі успіху:

```
{
 "success": true,
 "message": "Користувача створено",
 "user": {
  "user_id": 5,
  "username": "Test User",
  "email": "test@example.com",
  "role_id": 1
 }
}
```

Інтеграція з UI

Додавання нового користувача в список:

Після успішного запиту дані нового користувача додаються до таблиці на сторінці:

```
setUsers([...users, response.data.user]);
setOpenDialog(false);
```

Обробка помилок:

У разі помилки відображається повідомлення через компонент toast:

```
toast.error(error.response.data.message || 'Помилка створення користувача');
```

Модальне вікно "Додати користувача" реалізує зручну та безпечну взаємодію адміністратора із системою. Інтеграція клієнтської та серверної частин забезпечує валідацію на обох рівнях, а також гарантує швидку обробку запитів. Це є важливим кроком для управління доступом у системі.

3.4 Висновки до розділу 3

У розділі 3 було розглянуто основні аспекти взаємодії між API та користувацьким інтерфейсом (UI), а також описано структуру та

функціональність окремих компонентів системи. Було проведено детальний аналіз архітектури взаємодії між фронтендом, розробленим на платформі Next.js, і бекендом, реалізованим у вигляді RESTful API. Особливу увагу приділено опису маршрутів API, які забезпечують ключові функції, такі як автентифікація користувачів, управління ролями та доступами, а також обробка запитів, пов’язаних із додаванням, редагуванням та видаленням користувачів.

Крім того, було продемонстровано приклади реалізації UI-компонентів, таких як сторінка списку користувачів і модальні вікна для входу, створення та редагування користувачів. Завдяки інтеграції з API, ці компоненти забезпечують динамічну роботу інтерфейсу та дозволяють виконувати операції в реальному часі. Окрему увагу приділено валідації даних на клієнтському рівні, що підвищує надійність та безпеку системи.

Розглянута архітектура забезпечує високу масштабованість і адаптивність системи, що є важливим для її використання у критично важливих середовищах, таких як центри управління автономними морськими об’єктами. Інтеграція багатошарової валідації, захисту та багатофакторної автентифікації гарантує високий рівень безпеки та захищеності даних. Таким чином, розділ демонструє надійність реалізованої системи, її відповідність сучасним стандартам і вимогам кібербезпеки.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

4.1 Основні положення охорони праці

Охорона праці – це сукупність правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних та лікувально-профілактичних заходів, що спрямовані на збереження здоров'я та працездатності людини в умовах трудової діяльності [29].

Відповідно до Закону України «Про охорону праці» від 14.10.1992 року, визначаються основні положення щодо реалізації конституційного права громадян на безпеку життя та здоров'я під час виконання трудових обов'язків. Закон регулює відносини між роботодавцем, підприємствами, установами або організаціями (незалежно від форми власності), державними органами та працівниками з питань безпеки, гігієни праці й виробничого середовища. Він встановлює єдиний порядок організації заходів із охорони праці в Україні, охоплюючи всі підприємства й осіб, які здійснюють трудову діяльність на їхніх базах.

Законодавство у сфері охорони праці включає:

- Закон України «Про охорону праці»;
- Кодекс законів про працю України;
- інші нормативно-правові акти.

Державна політика в галузі охорони праці ґрунтується на таких принципах:

- Пріоритет захисту життя і здоров'я працівників, що корелюється з результатами діяльності підприємства, а також повну відповідальність роботодавця за створення безпечних умов праці.
- Комплексне вирішення питань охорони праці, яке базується на національних програмах, урахуванні соціально-економічних завдань, досягнень науки й техніки, а також охорони довкілля.

- Соціальний захист працівників через повне відшкодування шкоди у випадках виробничих травм чи професійних захворювань.
- Економічне стимулювання охорони праці, яке включає пільгове оподаткування та державну підтримку заходів із забезпечення безпеки праці.
- Координація дій державних органів, громадських об’єднань і роботодавців для вирішення питань безпеки та гігієни праці, а також підтримка співпраці між роботодавцями, працівниками й усіма соціальними групами під час ухвалення рішень на різних рівнях.

4.2 Аналіз небезпечних і шкідливих факторів у приміщенні з компьютерною технікою

У процесі трудової діяльності на людину можуть впливати небезпечні та шкідливі виробничі чинники. Небезпечні чинники є такими, що можуть спричинити нещасний випадок або різке погіршення стану здоров’я працівника, тоді як шкідливі чинники викликають розвиток професійних захворювань. Між цими двома типами чинників існує тісний взаємозв’язок, адже шкідливі чинники часто створюють умови для прояву небезпечних. Наприклад, надмірна вологість у приміщенні та струмопровідний пил, які є шкідливими чинниками, значно підвищують ризик ураження електричним струмом, що є небезпечним чинником.

Працівники, чия діяльність пов’язана з використанням комп’ютерної техніки, часто зазнають впливу фізичних небезпечних і шкідливих чинників, таких як підвищений рівень шуму, недостатнє або неякісне освітлення робочої зони, а також вплив електричного струму та статичної електрики. Крім того, користувачі ПК на робочому місці нерідко стикаються із психофізіологічними чинниками, до яких належать розумова перенапруга, перенапруга зорових і слухових аналізаторів, а також монотонність роботи. Комбінована дія

зазначених чинників негативно впливає на працездатність, а тривале перебування у їхньому середовищі може призвести до професійних захворювань.

Виробниче освітлення – є важливим чинником покращення умов праці в сучасному виробництві, особливо в контексті оптимізації кількісних та якісних характеристик освітлення робочих місць. У радіотехнічному та електронному виробництві це питання набуває особливого значення через інтенсифікацію праці та мініатюризацію апаратури. Значна частина технологічних процесів у цій сфері вимагає роботи найвищої точності, яка супроводжується високим ступенем напруженості зорової діяльності. Втомлюваність органів зору залежить від рівня напруженості, пов’язаної із зоровим сприйняттям.

Раціональне освітлення виробничих приміщень та робочих місць дозволяє покращити умови зорової роботи, зменшити зорове і нервове стомлення, а також сприяє підвищенню уваги та координації. Дослідження показали, що якісне освітлення позитивно впливає на функціонування дихальної системи, підвищуючи поглинання кисню.

Рівномірний розподіл яскравості в полі зору також відіграє ключову роль у підтриманні працездатності. Якщо в полі зору людини присутні поверхні з різкою різницею в освітленості, при переводі погляду з яскравоосвітленої ділянки на темну око змушене постійно адаптуватися. Така переадаптація часто викликає зорову втому і ускладнює виконання виробничих завдань.

Випромінювання монітора – монітор персонального комп’ютера, що працює на основі електронно-променевої трубки, може бути джерелом рентгенівського, ультрафіолетового, інфрачервоного, видимого, радіочастотного та низькочастотного електромагнітного випромінювань. Однак дослідження показують, що інтенсивність цих випромінювань значно нижча за рівні, здатні викликати біологічний вплив на людину. Для різних моделей моніторів величини рентгенівського випромінювання навіть на відстані 5–10 см від екрану залишаються в межах допустимих норм. Рівні ультрафіолетового та

інфрачервоного випромінювань також не перевищують природного фону або відповідають йому, тому вони не можуть завдати шкоди.

На робочих місцях реєструються низькоінтенсивні електромагнітні випромінювання, які не перевищують допустимих значень для електричних і магнітних полів у діапазоні частот 60 кГц – 300 МГц. Найвищий рівень напруженості електричного поля фіксується поблизу задньої панелі монітора.

Шум – це небажаний для людини звук, рівень якого вимірюється в децибелах (дБ). Залежно від рівня, характеру, тривалості шуму та індивідуальних особливостей людини, його вплив може варіюватися. Шум у межах 20–30 дБ є природним фоном, тоді як шум до 80 дБ сприймається як голосний. Рівень шуму в 130 дБ викликає больові відчуття, а 150 дБ стає нестерпним.

Шум створює значне навантаження на нервову систему, впливаючи на людину як психологічно, так і фізіологічно. Слабкий шум діє на різних людей по-різному, залежно від віку, стану здоров’я, характеру роботи та індивідуального ставлення до шуму. Водночас сильний шум ускладнює розпізнавання колірних сигналів, знижує швидкість сприйняття кольорів, гостроту зору та порушує сприйняття візуальної інформації. Це часто супроводжується головним болем, нудотою, підвищеною дратівливістю та загальним відчуттям нездужання.

Тривалий вплив шуму з рівнем звукового тиску 85–90 дБ сприяє збільшенню кількості помилок у роботі, знижує продуктивність праці на 30–60 % і негативно впливає на її якість. Шум високого рівня може стати причиною безсоння, неврозів і склерозу. При тривалому впливі на рівні високих децибелів слухова чутливість знижується через 1–2 роки, а при середніх рівнях – через 5–10 років. Це підвищує ризик втрати слуху, тому важливо заздалегідь уживати заходів захисту від шуму.

Акустичні подразнення накопичуються в організмі поступово, викликаючи зміни в силі, урівноваженості та рухливості нервових процесів.

Чим інтенсивніший шум, тим сильніший його вплив. Люди, які постійно перебувають під впливом шуму, часто стають дратівливими та важкими в спілкуванні.

4.3 Висновки за розділом

Аналіз небезпечних і шкідливих виробничих факторів, пов’язаних із використанням комп’ютерної техніки, показує, що вони можуть істотно впливати на здоров’я та працездатність працівників. Небезпечні фактори, такі як електричний струм або випромінювання моніторів, мають потенціал завдати раптової шкоди, а шкідливі фактори, як-от надмірний шум, недостатнє освітлення чи статична електрика, спричиняють поступове погіршення здоров’я та підвищують ризик професійних захворювань.

Виробниче освітлення є одним із ключових аспектів, що впливає на якість зорової роботи, зменшує стомлення та сприяє підвищенню продуктивності праці. Раціональне розташування джерел світла, уникнення різкого контрасту в полі зору та забезпечення належної освітленості робочих місць є критично важливими для комфортної роботи.

Шум також є значущим негативним фактором, який чинить тиск на нервову систему, викликає фізіологічні та психологічні розлади. Тривала дія шуму високої інтенсивності знижує продуктивність праці, викликає швидке виснаження організму та підвищує ризик розвитку професійних захворювань, зокрема втрати слуху.

Випромінювання моніторів сучасних комп’ютерів, хоча й значно знижене завдяки технічному прогресу, також залишається потенційним джерелом шкідливих впливів. Проте їх рівень зазвичай не перевищує встановлених норм, особливо за дотримання рекомендацій щодо організації робочих місць і захисних заходів.

Таким чином, для збереження здоров’я працівників необхідно комплексно вирішувати питання організації безпечних умов праці: впроваджувати ефективне освітлення, знижувати рівень шуму, мінімізувати шкідливі впливи випромінювань і забезпечувати адекватний психофізіологічний комфорт. Це не лише покращить стан здоров’я працівників, але й підвищить ефективність та якість виконання трудових обов’язків.

ВИСНОВОКИ

У даній магістерській роботі було розроблено інтелектуальну систему контролю доступу до центру управління автономними морськими об'єктами.

У ході виконання роботи було обґрунтовано вибір сучасних технологій для створення системи, яка забезпечує високий рівень кібербезпеки та ефективності. Було вирішено впровадити багатофакторну аутентифікацію, яка поєднує використання паролів, біометричних даних та фізичних ключів доступу. Це дозволило створити надійний механізм захисту від несанкціонованого доступу, що є критично важливим для роботи систем у складних умовах обмеженого фізичного доступу до автономних об’єктів.

У процесі розробки було створено централізовану базу даних, що зберігає інформацію про користувачів, їхні ролі та права доступу. Система реалізує багатошарову архітектуру безпеки, яка включає моніторинг активності користувачів у реальному часі та аналіз потенційних загроз за допомогою алгоритмів машинного навчання. Це забезпечує можливість оперативного реагування на будь-які несанкціоновані дії та забезпечення стабільної роботи системи.

Ключовим аспектом роботи стала оптимізація бази даних для швидкої обробки великих обсягів інформації. Було створено логічну та фізичну структури бази даних з використанням системи управління PostgreSQL, що забезпечує масштабованість та надійність. Особливу увагу приділено захисту даних за допомогою шифрування як у процесі зберігання, так і під час передачі. Для підвищення зручності управління правами доступу був розроблений інтерфейс API, інтегрований з користувацьким інтерфейсом.

ПЕРЕЛІК ЛІТЕРАТУРИ

1. Бондаренко О.М. Системи управління базами даних : навч. посіб. / О.М. Бондаренко. — Київ : Видавничий дім “Професіонал”, 2022.
2. Гончаренко С.У. Автоматизація процесів управління в інформаційних системах : наук. монографія / С.У. Гончаренко. — Львів : Видавництво “Новий Світ-2000”, 2020.
3. Доценко С.І. Організація та системи керування базами даних : навч. посіб. / С.І. Доценко. — Харків : УкрДУЗТ, 2023. — 117 с.
4. Іванова О.Е. Методи оптимізації SQL запитів: навчальний посібник / О.Е. Іванова. — Херсон: Олді+, 2021. — 192 с.
5. Кавун, С.В. “Інформаційна безпека”. Харків: ХНЕУ, 2020. URL: http://www.repository.hneu.edu.ua/jspui/bitstream/123456789/3068/1/%D0%9D%D0%B0%D0%B2%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9%20%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA.%20%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0%20%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D0%BA%D0%B0.%20%D0%9A%D0%B0%D0%B2%D1%83%D0%BD%20%D0%A1.%D0%92..pdf
6. Крученко Т.І. Безпека інформаційних систем: підручник / Т.І. Крученко. — Київ: Наукова думка, 2019. — 365 с.
7. Литвин В.Г. Адміністрування баз даних: навчальний посібник / В.Г. Литвин. — Чернігів: ЧНТУ, 2022. — 202 с.
8. Овсянніков, С.О., Скрипка, О.В. “Методологічні підходи до побудови інтелектуальних систем управління безпекою”. 2020. URL: https://journals.nupp.edu.ua/sunz/article/view/324/291
9. Остапенко Т.С. Методи аналізу даних та машинне навчання в управлінні базами даних / Т.С. Остапенко. — Одеса: Одеський політехнічний університет, 2020. — 289 с.

10. Офіційна документація PostgreSQL [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/.
11. Петрова М.А. Системи управління корпоративними базами даних: теорія і практика / М.А. Петрова. — Київ: Логос, 2020. — 250 с.
12. Савченко В.К. Теорія і практика розробки баз даних : навч. посіб. / В.К. Савченко. — Житомир : ЖДТУ, 2019.
13. Савчук В.О. Архітектура баз даних: від теорії до практики / В.О. Савчук. — Львів: Афіша, 2018. — 334 с.
14. Тимченко В.Л. Проектування оптимальних систем управління динамічними об’єктами / В.Л. Тимченко. — Миколаїв : НУК, 2006.
15. Ambu, R., Puddu, P.E., Rossi, A. “Research on Synthesis of Multi-Layer Intelligent System for Optimal and Safe Control of Marine Autonomous Object”. 2023. URL: https://www.mdpi.com/2079-9292/12/15/3299
16. de Tugny, M. “Shipping’s digital and sustainability transition and the future of seafarers”. 2022. URL: https://gmcg.global/shippings-digital-and-sustainability-transition-and-the-future-of-seafarers/
17. J.L. Martínez-de Dios, A. Rodríguez, J. Matute et al. “State-of-the-Art Research on Motion Control of Maritime Autonomous Surface Ships”. 2019. URL: https://www.mdpi.com/2077-1312/7/12/438
18. Kuzmin, O., Lomova, O., Nosov, K. “Sustainable smart city development in Ukraine: current state and future perspectives”. 2022. URL: https://www.mdpi.com/2071-1050/14/23/15630
19. Marine applications: The Future of Autonomous Maritime Transportation and Logistics. 2023. URL: https://www.researchgate.net/publication/378097628_Marine_applications_The_Future_of_Autonomous_Maritime_Transportation_and_Logistics

20. MDN, «Що таке Об'єктна Модель Документу (DOM)» URL: https://developer.mozilla.org/docs/Web/API/Document_Object_Model/Introduction;
21. React, «React. Бібліотека JavaScript» URL: https://react.js.org/;
22. Riviera Maritime Media. “Svitzer and Rolls-Royce demonstrate remote control tug operations”. 2021. URL: https://www.rivieramm.com/opinion/opinion/svitzer-and-rolls-royce-demonstrate-remote-control-tug-operations-28069
23. Rolls-Royce. “Rolls-Royce and Intel announce autonomous ship collaboration”. 2018. URL: https://www.rolls-royce.com/media/press-releases/2018/15-10-2018-rr-and-intel-announce-autonomous-ship-collaboration.aspx
24. Rolls-Royce. “World premiere for Rolls-Royce’s MTU NautiQ marine automation portfolio at DSEI”. 2021. URL: https://www.rolls-royce.com/media/press-releases/2021/14-09-2021-world-premiere-for-rolls-royces-mtu-nautiq-marine-automation-portfolio-at-dsei.aspx
25. StudFiles, «Шкідливі та небезпечні виробничі чинники при роботі з компютерною технікою» URL: https://studfile.net/preview/5211197/page:3/;
26. VSCode, «Code editing. Redefined» URL: https://code.visualstudio.com
27. Zhang, K., Zhang, X. “Marine applications: The future of autonomous maritime transportation and logistics”. 2023. URL: https://www.researchgate.net/publication/378097628_Marine_applications_The_Future_of_Autonomous_Maritime_Transportation_and_Logistics
28. Zp.gov.ua, «Навчання з питань охорони праці» URL: https://zp.gov.ua/upload/editor/navchannya_z_pitan_ohoroni_praci.pdf;

ДОДАТОК

Код програми

api

src/app/api/auth/sign-in/route.ts

```
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import { getClient } from 'src/app/db/client';
export async function POST(request: Request) {
  const client = getClient();
  await client.connect();
  const { email, password } = await request.json();
  if (!email || !password) {
    return new Response(
      JSON.stringify({ success: false, message: "Електронна пошта та пароль є обов'язковими" }),
      { status: 400 }
    );
  }
  try {
    const res = await client.query('SELECT * FROM users WHERE email = $1', [email]);
    if (res.rows.length === 0) {
      return new Response(
        JSON.stringify({ success: false, message: 'Невірна електронна пошта або пароль' }),
        { status: 400 }
      );
    }
    const user = res.rows[0];
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return new Response(
        JSON.stringify({ success: false, message: 'Невірна електронна пошта або пароль' }),
        { status: 400 }
      );
```

```
    }
    const token = jwt.sign({ id: user.id, email: user.email }, 'jwt_secret', { expiresIn: '1d' });
    return new Response(
      JSON.stringify({
        success: true,
        message: 'Вхід успішний',
        token,
        user: { id: user.id, email: user.email },
      }),
      { status: 200 }
    );
  } catch (error) {
    console.error('Error logging in user:', error);
    return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

src/app/api/auth/sign-up/route.ts

```
import bcrypt from 'bcrypt';
import { getClient } from 'src/app/db/client';
export async function POST(request: Request) {
  const client = getClient();
  await client.connect();
  const { username, email, password, role_id } = await request.json();
  if (!username || !email || !password || !role_id) {
    await client.end();
    return new Response(
      JSON.stringify({
        success: false,
        message: "Електронна пошта, пароль та ім'я користувача є обов'язковими",
```

```
    }),
    { status: 400 }
  );
}
try {
  const hashedPassword = await bcrypt.hash(password, 10);
  const res = await client.query(
    'INSERT INTO users (username, email, password) VALUES ($1, $2, $3) RETURNING user_id, username, email',
    [username, email, hashedPassword]
  );
  const newUser = res.rows[0];
  await client.query('INSERT INTO user_roles (user_id, role_id) VALUES ($1, $2)', [
    newUser.user_id,
    role_id,
  ]);
  const roles = await client.query('SELECT * FROM roles WHERE role_id = $1', [role_id]);
  const role = roles.rows[0];
  return new Response(
    JSON.stringify({
      success: true,
      message: 'Користувач успішно зареєстрований',
      user: {
        user_id: newUser.user_id,
        email: newUser.email,
        username: newUser.username,
        role_id: role.role_id,
        role_name: role.role_name,
      },
    }),
    { status: 200 }
  );
} catch (error) {
  console.error('Помилка при реєстрації користувача:', error);
  return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
```

```
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

src/app/api/roles/route.ts

```
import { getClient } from 'src/app/db/client';
export async function GET(request: Request) {
  const client = getClient();
  await client.connect();
  try {
    const res = await client.query('SELECT * FROM roles');
    return new Response(JSON.stringify({ success: true, roles: res.rows }), {
      status: 200,
    });
  } catch (error) {
    console.error('Помилка :', error);
    return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

src/app/api/users/[userId]/route.ts

```
import { getClient } from 'src/app/db/client';
export async function PUT(request: Request, { params }: { params: { userId: string } }) {
  const client = getClient();
  await client.connect();
  try {
```

```
  const { userId } = params;
  const { username, role_id } = await request.json();
  await client.query('BEGIN');
  const userRes = await client.query(
   'UPDATE users SET username = $1 WHERE user_id = $2 RETURNING *',
   [username, userId]
  );
  if (userRes.rowCount === 0) {
   await client.query('ROLLBACK');
   return new Response(JSON.stringify({ success: false, message: 'Користувача не знайдено' }), {
    status: 404,
   });
  }
  await client.query('DELETE FROM user_roles WHERE user_id = $1', [userId]);
  await client.query('INSERT INTO user_roles (user_id, role_id) VALUES ($1, $2)', [
   userId,
   role_id,
  ]);
  await client.query('COMMIT');
  return new Response(
   JSON.stringify({ success: true, message: 'Користувача успішно оновлено' }),
   {
    status: 200,
   }
  );
 } catch (error) {
  await client.query('ROLLBACK');
  console.error('Error updating user:', error);
  return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
   status: 500,
  });
 } finally {
  await client.end();
 }
```

```
}
export async function DELETE(request: Request, { params }: { params: { userId: string } }) {
  const client = getClient();
  await client.connect();
  try {
    const { userId } = params;
    const res = await client.query('DELETE FROM users WHERE user_id = $1 RETURNING *', [userId]);
    if (res.rowCount === 0) {
      return new Response(JSON.stringify({ success: false, message: 'Користувача не знайдено' }), {
        status: 404,
      });
    }
    return new Response(
      JSON.stringify({ success: true, message: 'Користувача успішно видалено' }),
      {
        status: 200,
      }
    );
  } catch (error) {
    console.error('Error deleting user:', error);
    return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

src/app/api/users/route.ts

```
import { getClient } from 'src/app/db/client';
export async function GET(request: Request) {
  const client = getClient();
  await client.connect();
  try {
```

```
    const res = await client.query(`
      SELECT
        users.user_id,
        users.username,
        users.email,
        users.is_active,
        roles.role_id,
        roles.role_name
      FROM users
      LEFT JOIN user_roles ON users.user_id = user_roles.user_id
      LEFT JOIN roles ON user_roles.role_id = roles.role_id
      ORDER BY users.user_id
    `);
    const users = res.rows;
    return new Response(
      JSON.stringify({
        users,
      }),
      { status: 200 }
    );
  } catch (error) {
    console.error('Помилка входу користувача:', error);
    return new Response(JSON.stringify({ success: false, message: 'Внутрішня помилка сервера' }), {
      status: 500,
    });
  } finally {
    await client.end();
  }
}
```

src/app/db/client.ts

```
import { Client } from 'pg';
export function getClient() {
  return new Client({
```

```
    host: '127.0.0.1',
    database: 'postgres',
    port: 5434,
  });
}
```

components

src/components/SignIn.tsx

```
'use client';
import axios from 'axios';
import { z as zod } from 'zod';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import Box from '@mui/material/Box';
import IconButton from '@mui/material/IconButton';
import LoadingButton from '@mui/lab/LoadingButton';
import InputAdornment from '@mui/material/InputAdornment';
import { useRouter } from 'src/routes/hooks';
import { useBoolean } from 'src/hooks/use-boolean';
import { toast } from 'src/components/snackbar';
import { Iconify } from 'src/components/iconify';
import { Form, Field } from 'src/components/hook-form';
import { FormHead } from 'src/auth/components/form-head';
export type SignInSchemaType = zod.infer<typeof SignInSchema>;
export const SignInSchema = zod.object({
  email: zod
    .string()
    .min(1, { message: 'Email is required!' })
    .email({ message: 'Email must be a valid email address!' }),
  password: zod
    .string()
    .min(1, { message: 'Password is required!' })
    .min(6, { message: 'Password must be at least 6 characters!' }),
```

```
});
export function SignInView() {
  const password = useBoolean();
  const router = useRouter();
  const defaultValues = { email: '', password: '' };
  const methods = useForm<SignInSchemaType>({
    resolver: zodResolver(SignInSchema),
    defaultValues,
  });
  const {
    handleSubmit,
    formState: { isSubmitting },
  } = methods;
  const onSubmit = handleSubmit(async (data) => {
    try {
      const response = await axios.post('/api/auth/sign-in', data);
      toast.success('Успішний вхід');
      router.replace('/nuos/dashboard');
    } catch (error) {
      console.error('Error signing in user:', error);
      toast.error(error.response.data.message || 'Помилка входу');
    }
  });
  const renderForm = (
    <Box gap={3} display="flex" flexDirection="column">
      <Field.Text name="email" label="Пошта" InputLabelProps={{ shrink: true }} />
          <Box gap={1.5} display="flex" flexDirection="column">
        <Field.Text
          name="password"
          label="Пароль"
          placeholder="6+ букв"
          type={password.value ? 'text' : 'password'}
          InputLabelProps={{ shrink: true }}
          InputProps={{
```

```
            endAdornment: (
              <InputAdornment position="end">
                <IconButton onClick={password.onToggle} edge="end">
                  <Iconify icon={password.value ? 'solar:eye-bold' : 'solar:eye-closed-bold'} />
                </IconButton>
              </InputAdornment>
            ),
          }}
        />
      </Box>
          <LoadingButton
        fullWidth
        color="inherit"
        size="large"
        type="submit"
        variant="contained"
        loading={isSubmitting}
        loadingIndicator="Входить..."
      >
        Увійти
      </LoadingButton>
    </Box>
  );
  return (
    <>
      <FormHead title="Увійдіть у свій аккаунт" />
      <Form methods={methods} onSubmit={onSubmit}>
        {renderForm}
      </Form>
    </>
  );
}
```

src/components/SignUp.tsx

```
'use client';
import { z as zod } from 'zod';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import Box from '@mui/material/Box';
import { MenuItem } from '@mui/material';
import IconButton from '@mui/material/IconButton';
import LoadingButton from '@mui/lab/LoadingButton';
import InputAdornment from '@mui/material/InputAdornment';
import { useBoolean } from 'src/hooks/use-boolean';
import { Iconify } from 'src/components/iconify';
import { Form, Field } from 'src/components/hook-form';
export type SignUpSchemaType = zod.infer<typeof SignUpSchema>;
export const SignUpSchema = zod.object({
  role_id: zod.number(),
  username: zod.string().min(1, { message: "Ім'я користувача" }),
  email: zod.string().min(1, { message: "Пошта обов'язково" }).email({ message: 'Не вірна пошта' }),
  password: zod
    .string()
    .min(1, { message: "Пароль обов'язковий" })
    .min(6, { message: 'Довжина паролю має бути більше 6 символів' }),
});
export function SignUp({ onSave, roles }: { onSave: () => void; roles: object[] }) {
  const password = useBoolean();
  const defaultValues = {
    role_id: 1,
    username: '',
    email: '',
    password: '',
  };
  const methods = useForm<SignUpSchemaType>({
    resolver: zodResolver(SignUpSchema),
    defaultValues,
```

```
});
const {
  handleSubmit,
  formState: { isSubmitting },
} = methods;
const onSubmit = handleSubmit(async (data) => {
  try {
    onSave(data);
  } catch (error) {
    console.error(error);
  }
});
const renderForm = (
  <Box gap={3} display="flex" flexDirection="column">
    <Field.Select
      fullWidth
      name="role_id"
      label="Роль"
      margin="normal"
      InputLabelProps={{ shrink: true }}
    >
      {roles.map((role: any) => (
        <MenuItem key={role.role_id} value={role.role_id}>
          {role.role_name}
        </MenuItem>
      ))}
    </Field.Select>
        <Box display="flex" gap={{ xs: 3, sm: 2 }} flexDirection={{ xs: 'column', sm: 'row' }}>
    <Field.Text name="username" label="Ім'я користувача" InputLabelProps={{ shrink: true }} />
    </Box>
        <Field.Text name="email" type="email" label="Пошта" InputLabelProps={{ shrink: true }} />
        {/* TODO: add route */}
        <Field.Text
    name="password"
```

```
      label="Пароль"
      placeholder="6+ символів"
      type={password.value ? 'text' : 'password'}
      InputLabelProps={{ shrink: true }}
      InputProps={{
        endAdornment: (
          <InputAdornment position="end">
            <IconButton onClick={password.onToggle} edge="end">
              <Iconify icon={password.value ? 'solar:eye-bold' : 'solar:eye-closed-bold'} />
            </IconButton>
          </InputAdornment>
        ),
      }}
    />
        <LoadingButton
      fullWidth
      color="inherit"
      size="large"
      type="submit"
      variant="contained"
      loading={isSubmitting}
      loadingIndicator="Користувач создается..."
    >
      Создати користувача
    </LoadingButton>
  </Box>
);
return (
  <Box mt={6} mb={6}>
    <Form methods={methods} onSubmit={onSubmit}>
      {renderForm}
    </Form>
  </Box>
);
```

```
}
```

src/components/UsersList.tsx

```
'use client';
import axios from 'axios';
import { useState, useEffect } from 'react';
import {
  Box,
  Table,
  Dialog,
  Button,
  TableRow,
  MenuItem,
  TableBody,
  TableCell,
  Container,
  TextField,
  IconButton,
} from '@mui/material';
import { toast } from 'src/components/snackbar';
import { ComponentBlock } from 'src/sections/_examples/component-block';
import { SignUp } from './SignUp';
import { Iconify } from '../iconify';
import { Scrollbar } from '../scrollbar';
import { TableHeadCustom } from '../table';
const blockProps = {
  p: 0,
  overflow: 'hidden',
  alignItems: 'unset',
  flexDirection: 'column',
  bgcolor: 'background.paper',
};
const TABLE_HEAD = [
  { id: 'user_id', label: 'Id' },
```

```
  { id: 'user_name', label: 'Им’я' },
  { id: 'email', label: 'Пошта' },
  { id: 'is_active', label: 'Активність' },
  { id: 'password_changed_at', label: 'Остання змінна паролю' },
  { id: 'role_name', label: 'Роль' },
  { id: 'actions', label: 'Дії' },
];
export function UsersList() {
  const [users, setUsers] = useState([]);
  const [roles, setRoles] = useState([]);
  const [openDialog, setOpenDialog] = useState(false);
  const [editUser, setEditUser] = useState<any>(null);
  useEffect(() => {
    const fetchData = async () => {
      const [usersResponse, rolesResponse] = await Promise.all([
        axios.get('/api/users'),
        axios.get('/api/roles'),
      ]);
      setUsers(usersResponse.data?.users);
      setRoles(rolesResponse.data?.roles.sort((a: any, b: any) => a.role_id - b.role_id));
    };
    fetchData();
  }, []);
  const handleRemoveUser = async (userId: string) => {
    await axios.delete(`/api/users/${userId}`);
    setUsers(users.filter((user: any) => user.user_id !== userId));
  };
  const handleCloseDialog = () => {
    setOpenDialog(false);
  };
  const handleAddUser = (user: any) => {
    setEditUser(null);
    setOpenDialog(true);
  };
```

```
const handleEditUser = (user: any) => {
  setEditUser(user);
  setOpenDialog(true);
};
const handleSaveEditUser = async () => {
  await axios.put(`/api/users/${editUser.user_id}`, editUser);
  setUsers(users.map((user: any) => (user.user_id === editUser.user_id ? editUser : user)));
  setOpenDialog(false);
};
const handleSaveUser = async (data: any) => {
  try {
    const response = await axios.post('/api/auth/sign-up', data);
    toast.success('Користувача створено');
    setUsers([...users, response.data.user]);
    setOpenDialog(false);
  } catch (error) {
    console.error('Error signing in user:', error);
    toast.error(error.response.data.message || 'Помилка створення користувача');
  }
};
return (
  <Container sx={{ mt: 10, mb: 15 }}>
    <ComponentBlock sx={blockProps}>
      <Scrollbar>
        <Table sx={{ minWidth: 800 }}>
          <TableHeadCustom headLabel={TABLE_HEAD} />
          <TableBody>
            {users.map((user: any) => (
              <TableRow key={user.user_id}>
                <TableCell>{user.user_id}</TableCell>
                <TableCell>{user.username}</TableCell>
                <TableCell>{user.email}</TableCell>
                <TableCell>{user.is_active ? 'Активний' : 'Не активний'}</TableCell>
                <TableCell>
```

```
            {user.password_changed_at ? user.password_changed_at : 'Ніколи'}
          </TableCell>
          <TableCell>{user.role_name}</TableCell>
          <TableCell>
            <IconButton onClick={() => handleEditUser(user)}>
              <Iconify width={24} icon="eva:edit-outline" />
            </IconButton>
            <IconButton onClick={() => handleRemoveUser(user.user_id)}>
              <Iconify width={24} icon="eva:trash-2-outline" />
            </IconButton>
          </TableCell>
        </TableRow>
      ))}
    </TableBody>
  </Table>
 </Scrollbar>
</ComponentBlock>
<Box mt={2}>
 <Button variant="contained" onClick={handleAddUser}>
  Додати користувача
 </Button>
</Box>
<Dialog open={openDialog} onClose={handleCloseDialog} fullWidth maxWidth="sm">
 <Box>
  <Box ml="auto" mr="auto" maxWidth="512px">
   {editUser && (
    <Box p={3}>
     <TextField
      fullWidth
      label="Ім’я"
      value={editUser.username}
      onChange={(e) => setEditUser({ ...editUser, username: e.target.value })}
      margin="normal"
     />
```

```
            <TextField
              fullWidth
              select
              label="Роль"
              value={editUser.role_id}
              onChange={(e) =>
                setEditUser({
                  ...editUser,
                  role_id: e.target.value,
                  role_name: roles.find((role: any) => role.role_id === e.target.value)
                    .role_name,
                })
              }
              margin="normal"
            >
              {roles.map((role: any) => (
                <MenuItem key={role.role_id} value={role.role_id}>
                  {role.role_name}
                </MenuItem>
              ))}
            </TextField>
            <Box mt={2} display="flex" justifyContent="flex-end">
              <Button variant="contained" onClick={handleSaveEditUser}>
                Зберегти
              </Button>
            </Box>
          </Box>
        )}
        {!editUser && <SignUp onSave={handleSaveUser} roles={roles} />}
      </Box>
    </Box>
  </Dialog>
</Container>
);
```

```
}
```