Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
Інститут електроенергетики
(інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

студента Калянова Владислава Вікторовича
(ПІБ)

академічної групи 123-20з-1
(шифр)

спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)

за освітньо-професійною програмою 123 Комп'ютерна інженерія
(офіційна назва)

на тему: «Комп'ютерна система компанії «TraderEvolution» з детальною розробкою застосунку для підвищення ефективності командних розробок ПЗ»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Олевський В.І.			
спеціальної частини	проф. Олевський В.І.			
розділів:				
розробка апаратної частини	доц. Ткаченко С.М.			
розробка корпоративної мережі	ас. Панферова Я.В.			

Рецензент				

Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро

2024

ЗАТВЕРДЖЕНО:

завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)

_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)
« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавр

студента Калянов В.В. **академічної групи** 123-20з-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія

за освітньо-професійною програмою 123 Комп'ютерна інженерія
(офіційна назва)

на тему «Комп'ютерна система компанії «TraderEvolution» з детальною розробкою застосунку для підвищення ефективності командних розробок ПЗ»

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 № 470-с

Розділ	**Зміст**	**Термін виконання**
Стан питання та постановка завдання	Визначити поточний стан комп'ютерної системи компанії «TraderEvolution» та сформулювати завдання для її оптимізації з метою підвищення ефективності командних розробок ПЗ.	10.05.2024
Розробка апаратної частини	Розробити вимоги до функцій, виконуваними системою, розробити структурну схему та специфікацію обладнання	20.05.2024
Розробка корпоративної мереж	Побудувати в Packet Tracer модель корпоративної мережі компанії, виконати налаштування та перевірку роботи системи	10.06.2024
Розробка компонента системи	Розробити застосунок для підвищення ефективності командних розробок ПЗ	20.06.2024

Завдання видано _____ проф. Олевський В.І.
(підпис керівника) (прізвище, ініціали)

Дата видачі 26.04.2024

Дата подання до екзаменаційної комісії 01.07.2024

Прийнято до виконання _____ Калянов В.В.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 95 с., 17 рис., 4 табл., 8 джерел, 1 додаток.

КОМП'ЮТЕРНА СИСТЕМА, РОЗРОБКА ЗАСТОСУНКІВ, ЕФЕКТИВНІСТЬ КОМАНДИ, ІНФРАСТРУКТУРА, БЕЗПЕКА ДАНИХ, РЕЗЕРВУВАННЯ, VPN, ШИФРУВАННЯ.

Об’єкт розробки − комп’ютерна система компанії «TraderEvolution» для підвищення ефективності командної роботи та оптимізації внутрішніх процесів.

Мета роботи − створення інтегрованої комп'ютерної системи, що дозволяє ефективно управляти проектами та оптимізувати робочі процеси в компанії.

Розроблено комп’ютерну систему для компанії «TraderEvolution», яка спрямована на значне підвищення ефективності командної роботи та оптимізацію внутрішніх процесів. Ця система надає можливість гнучкого налаштування функціональних можливостей, що дозволяє легко адаптуватися до різних вимог бізнесу.

Безпека даних є одним із ключових аспектів системи. Використання шифрування та VPN-з'єднань гарантує конфіденційність інформації та захищає дані від несанкціонованого доступу. Крім того, система підтримує регулярне резервне копіювання даних, що забезпечує надійний захист від втрати важливої інформації та дає можливість швидко відновити роботу у разі непередбачуваних обставин.

Розроблена комп’ютерна мережа відповідає завданням кваліфікаційної роботи бакалавра та пройшла перевірку за допомогою моделі схеми корпоративної мережі із застосуванням програми Cisco Packet Tracer. Сучасні мережеві технології та обладнання, зокрема мережеве обладнання Cisco, використовувалися для забезпечення високої надійності, масштабованості та безпеки мережі. Результати перевірки у вигляді таблиць та графіків детально описані та наведені у пояснювальній записці та додатках.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

КС – комп’ютерна система;

ТОВ – товариство з обмеженою відповідальністю;

VPN – Virtual Private Network;

CEO – Chief Executive Officer;

API – application programming interface;

LAN – Local Area Network;

IT – Information Technology;

DHCP – Dynamic Host Configuration Protocol;

DNS – Domain Name System;

HTTP – Hyper Text Transfer Protocol.

ВСТУП

У контексті глобалізації та технологічних інновацій, індустрія програмного забезпечення знаходиться під постійним тиском для оптимізації та автоматизації бізнес-процесів. ТОВ «TraderEvolution», як частина міжнародної фінтех-компанії «TraderEvolution Global», відіграє ключову роль у розвитку рішень для брокерського бізнесу. Зі штаб-квартирами у місті Дніпро компанія впроваджує передові технології для поліпшення управління активами, торгівлі та аналітики, реагуючи на потреби клієнтів з усього світу.

Глобальні тенденції, такі як цифровізація фінансових послуг та необхідність забезпечення високої реактивності на зміни ринку, підкреслюють важливість інвестицій у розробку адаптивних технологічних рішень. Статистика показує, що фірми, які впроваджують автоматизовані системи управління проектами, в середньому досягають 27% кращої продуктивності порівняно з тими, хто цього не робить.

Ця кваліфікаційна робота зосереджена на створенні комп'ютерної системи для управління проектами у ТОВ «TraderEvolution», що дозволить підвищити ефективність командної роботи та оптимізувати внутрішні процеси. Вона відповідає глобальним тенденціям в індустрії та вирішує конкретні задачі автоматизації і цифровізації, які є критично важливими для забезпечення конкурентоспроможності на міжнародному ринку.

Метою роботи є розробка і впровадження інтегрованої системи, яка сприятиме ефективнішому управлінню проектами через вдосконалення взаємодій всередині команди і зовнішніх взаємодій з клієнтами. Результати розробки можуть бути використані не тільки у фінтех компаніях, а й у ширшому спектрі індустрій, де потрібно ефективно управляти складними проектами.

1 СТАН ПИТАННЯ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Характеристика підприємства та умов застосування комп'ютерних систем

ТОВ «TraderEvolution» є частиною міжнародної фінтех компанії TraderEvolution Global, яка спеціалізується на наданні комплексних технологічних рішень для брокерських компаній. Основні продукти компанії включають платформи для онлайн трейдингу, які забезпечують клієнтам доступ до ринків акцій, валют, ф'ючерсів та інших фінансових інструментів.

Нещодавно компанія розширила свою структуру, відкривши новий керівний офіс у місті Дніпро, на вулиці Князя Владимира Великого. Цей офіс займається стратегічним управлінням та координацією основних бізнес-процесів, в той час як технічна розробка та підтримка залишаються у центральному офісі на проспекті Богдана Хмельницького.

Умови застосування комп'ютерних систем охоплюють використання локальних серверів у місті для розробки, тестування та виробництва, а також сервера для резервного копіювання даних і забезпечення стабільності онлайн-платформ. Обидва офіси у місті Дніпро з'єднані між собою за допомогою зашифрованих VPN-каналів, що забезпечує високий рівень безпеки та конфіденційності обміну даними.

1.2 Принципи, технічні способи та математичні методи інформаційного забезпечення підприємства

ТОВ «TraderEvolution» застосовує ряд принципів та технічних рішень для забезпечення ефективності своїх інформаційних систем та процесів. Ці рішення орієнтовані на забезпечення стабільності, безпеки, та оптимізації роботи в реальному часі.

Принципи інформаційного забезпечення:

– модульність – системи проектуються з можливістю легкої інтеграції нових модулів та функцій, що сприяє гнучкості в реалізації бізнес-вимог та швидкому адаптуванні до ринкових змін.

– відкритість – підприємство використовує стандартні протоколи та інтерфейси для забезпечення легкої інтеграції різних систем та сервісів, підтримуючи високий рівень сумісності.

– масштабованість – інфраструктура підтримує можливість швидкого розширення або зменшення ресурсів в залежності від потреб компанії, забезпечуючи оптимальне використання ресурсів.

– надійність та доступність – застосування резервування даних та відмовостійких компонентів забезпечує високий рівень доступності сервісів і мінімізацію втрат даних у будь-який час.

Технічні способи:

– локальні сервери та рішення для дата-центрів: Основні системи, такі як сервери для розробки, тестування і резервного копіювання для забезпечення стабільності сервісів.

– використання VPN для безпечного з'єднання між офісами в Дніпрі, що дозволяє безпечно обмінюватись даними між локаціями.

Математичні методи:

– використання статистичного аналізу, моделювання та прогнозування для оптимізації робочих процесів та аналізу даних, що допомагає в прийнятті обґрунтованих управлінських рішень на основі кількісних показників.

1.3 Огляд існуючих інженерних рішень КС в галузі та визначення можливих напрямків рішення поставлених завдань

Локальний сервер для розробки та тестування.

Новий сервер для застосунку підвищення ефективності. Враховуючи завдання нового сервера – координацію роботи розробників – оптимальним вибором буде менш потужний, але високонадійний сервер з достатнім рівнем

обробки для управлінських завдань при збереженні енергоефективності та нижчих витрат на експлуатацію.

Модернізація мережевої інфраструктури.

Технології:

– комутатори Cisco і маршрутизатори: Надійність і передові технології забезпечення безпеки, включаючи підтримку новітніх мережевих протоколів;

– безпека: Cisco пропонує більш розгалужену систему безпеки і керування доступом;

– конфігурація: Cisco зазвичай легше налаштовується і інтегрується у багатофункціональні мережі завдяки своїм інтуїтивним інтерфейсам управління.

Системи резервного копіювання та відновлення.

Технології:

– Veeam Backup & Replication: забезпечує швидке та надійне резервне копіювання з можливістю миттєвого відновлення;

– Acronis True Image: відомий своїми потужними функціями для створення образів диска та гнучкістю в управлінні.

Порівняння:

– швидкість резервного копіювання: Veeam забезпечує швидше резервне копіювання завдяки оптимізації даних;

– гнучкість: Acronis пропонує більш гнучкі опції для резервного копіювання на різні носії і платформи.

Цей огляд дозволить більш точно вибрати необхідні компоненти і технології для створення і підтримки додаткового застосунку, який допоможе підвищити ефективність командних розробок у компанії TraderEvolution.

1.4 Розробка схеми організаційної структури підприємства

Центральний офіс компанії знаходиться на третьому поверсі семиповерхової будівлі на проспекті Богдана Хмельницького, буд. 108. Керівний офіс розташований у центрі міста на вулиці Князя Владимира Великого буд. 20, на п'ятому поверсі п'ятиповерхової будівлі. Відстань між центральним та

керівним офісами складає приблизно 7,31 км. Зв'язок між мережами офісів буде здійснюватися за допомогою зашифрованих VPN-каналів, що буде забезпечувати безпечний обмін даними та високий рівень конфіденційності.

Топологічна схема розміщення структурних підрозділів центрального та керівного офісів компанії зображена на рисунку 1.1.

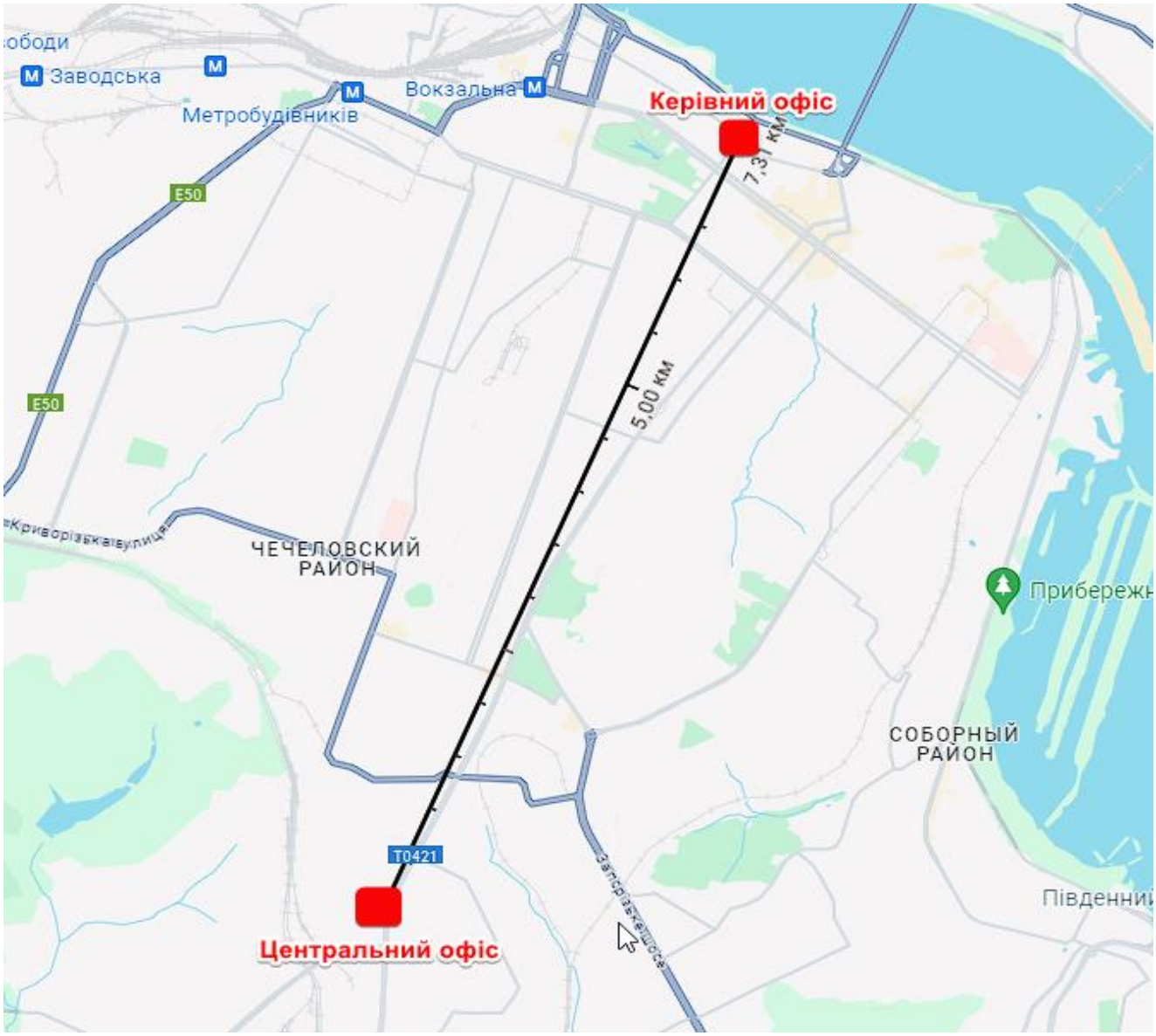

Рисунок 1.1 – Топологічна схема розміщення структурних підрозділів компанії

Топологічні схеми розміщення структурних підрозділів центрального та керівного офісів зображена на рисунках 1.2 – 1.3.

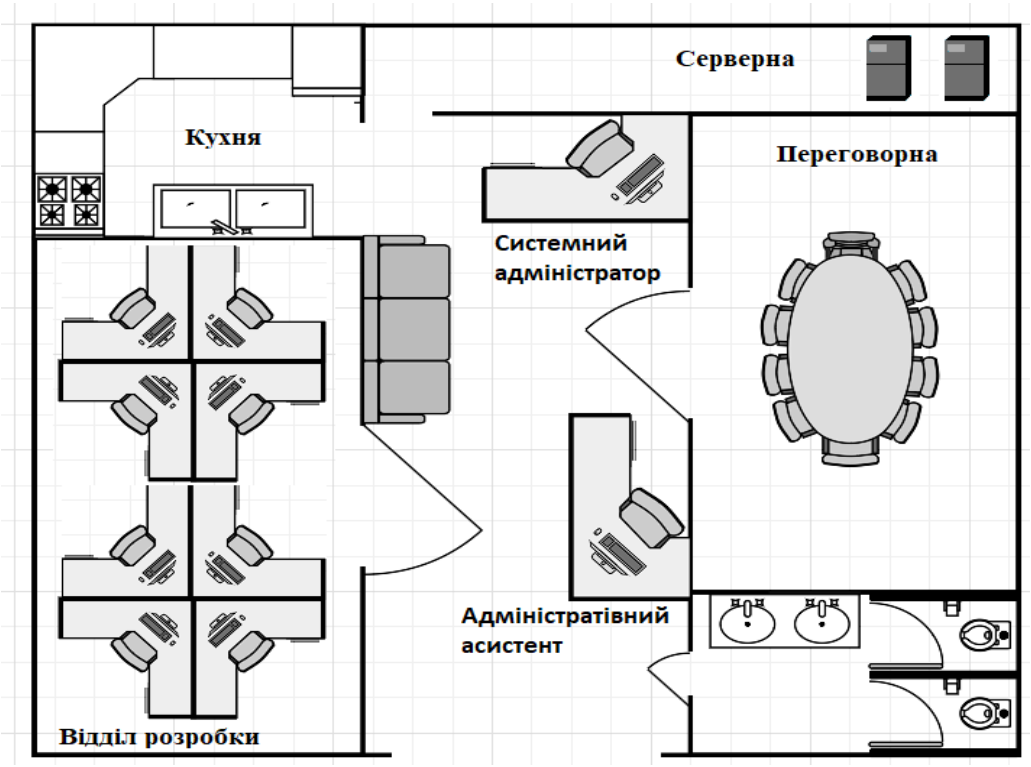

Рисунок 1.2 – Топологічна схема розміщення структурних підрозділів центрального офісу (пр. Богдана Хмельницького, буд. 108)

Рисунок 1.3 – Топологічна схема розміщення структурних підрозділів керівного офісу (вул. Князя Владимира Великого, буд. 20)

На рисунку 1.4 представлена схема організаційної структури підприємства.

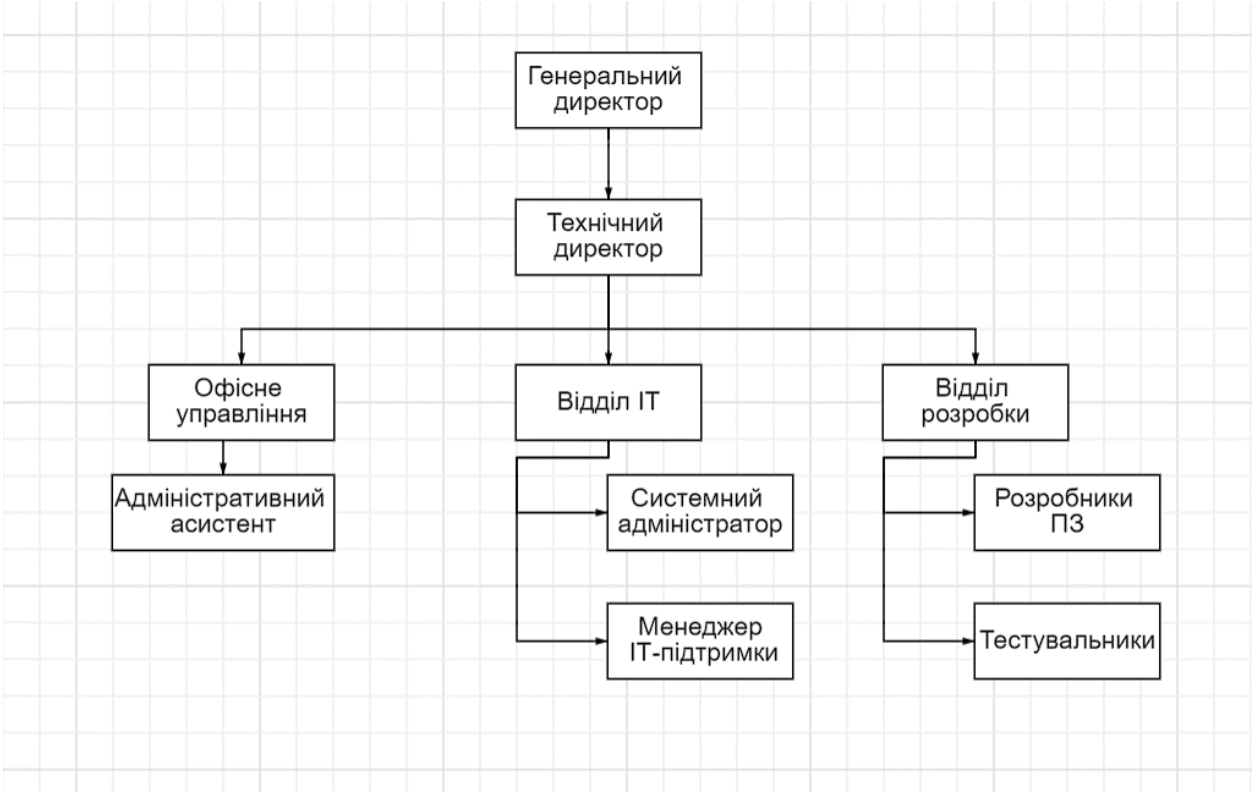

Рисунок 1.4 – Схема організаційної структури підприємства

Компанія «TraderEvolution» використовує структуру, яка дозволяє ефективно координувати діяльність між різними відділами та забезпечувати високу продуктивність їх роботи. Структура компанії складається з двох основних локацій: центрального офісу в Дніпрі (пр. Богдана Хмельницького) та нового керівного офісу, на вул. Князя Владимира Великого.

Структура центрального офісу (пр. Богдана Хмельницького, Дніпро) та основні обов’язки відділів.

Відділ IT.

Системний адміністратор керує всіма аспектами мережевої інфраструктури та забезпечує підтримку серверів і робочих станцій. Менеджер IT-підтримки відповідає за технічну підтримку співробітників і забезпечення безперебійної роботи систем.

Відділ розробки.

Розробники ПЗ займаються проектуванням і реалізацією програмних рішень. Тестувальники ПЗ забезпечують контроль якості продукції, виявлення та усунення помилок.

Офісне управління.

Адміністративний асистент відповідає за адміністративні та організаційні задачі в офісі.

Структура керівного офісу (вул. Князя Володимира Великого, Дніпро) та основні обов’язки відділів.

Генеральний директор (CEO) відповідає за загальне управління компанією. Технічний директор (CTO) забезпечує технічне лідерство у розробці нових продуктів. Менеджер IT координує всю IT-інфраструктуру та стратегії розвитку. Асистент виконує адміністративні функції та забезпечує підтримку керівництва.

Сильні сторони нової організаційної структури:

– концентрація керівництва – зосередження керівництва в окремому офісі сприяє кращій стратегічній фокусації та швидкому прийняттю рішень;

– спеціалізовані відділи – чітке розмежування обов'язків між відділами підвищує ефективність роботи та сприяє спеціалізації.

Слабкі сторони нової організаційної структури:

– обмежена взаємодія між локаціями – відсутність інтегрованих систем для забезпечення плавної взаємодії між центральним та керівним офісами може сповільнити комунікацію;

– пропозиції щодо удосконалення структури;

– впровадження інтегрованих комунікаційних систем – застосування єдиної системи управління.

1.5 Постановка завдання

Завдання: Розширення корпоративної мережі ТОВ «TraderEvolution» для інтеграції нового керівного офісу та розробка нового програмного додатку для підвищення ефективності командної розробки.

Мета: Забезпечення стабільної роботи корпоративної мережі з високою продуктивністю та безпекою, а також оптимізація внутрішніх бізнес-процесів через впровадження нових технологічних рішень.

Очікувані результати:

– безперебійна інтеграція нового керівного офісу в існуючу мережеву інфраструктуру;

– розробка та впровадження програмного додатку для управління проектами, що дозволить підвищити координацію між командами і забезпечити ефективне управління ресурсами;

– впровадження мережевих заходів безпеки для захисту корпоративних даних, зокрема через використання VPN-з'єднань і шифрування даних.

Для досягнення поставлених завдань планується використання наступних технологічних рішень:

– мережеве обладнання Cisco: для забезпечення надійності, масштабованості та безпеки корпоративної мережі, включаючи комутатори і маршрутизатори;

– резервне копіювання та відновлення даних: настройка систем veeam backup & replication та acronis true image на серверах, включаючи новий сервер у центральному офісі для підтримки розробки програмного додатку;

– VPN-з'єднання: для безпечного зв'язку між офісами в Дніпрі, а також для інтеграції віддалених працівників.

Вибір стратегії та технологій ґрунтується на критеріях надійності, масштабованості та безпеки, що є критично важливим для фінтех-компанії з міжнародною присутністю. Використання обладнання Cisco та системи шифрування та VPN дозволить компанії «TraderEvolution» захистити передачу даних та забезпечити безперервну роботу системи в будь-яких умовах.

2 РОЗРОБКА АПАРАТНОЇ ЧАСТИНИ КОМП'ЮТЕРНОЇ СИСТЕМИ ПІДПРИЄМСТВА «TRADEREVOLUTION»

2.1 Технічні вимоги до комп'ютерної системи підприємства

2.1.1 Вимоги до системи в цілому

2.1.1.1 Вимоги до структури і функціонуванню комп'ютерної системи підприємства «TraderEvolution»

Комп'ютерна система ТОВ «TraderEvolution» складається з наступних підсистем:

– LAN1 (46 вузлів) – основна мережа для розробників програмного забезпечення і тестувальників. Призначена для забезпечення безперебійної роботи внутрішніх IT-сервісів та розробки програмних продуктів;

– LAN2 (5 вузлів) – мережа для керівного складу;

– LAN3 (28 вузлів) – мережа для адміністративного персоналу та технічної підтримки;

– LAN4 (50 вузлів) – резервна мережа для управління і адміністративних задач. Призначена для забезпечення безпеки та резервування критичних бізнес-процесів.

Серверна інфраструктура включає:

– сервери для розробки та тестування.;

– сервери баз даних;

– сервер для централізованого управління проектами;

– сервер для резервного копіювання.

Система має дворівневу ієрархію:

а) перший рівень: центральний офіс і керівний офіс, кожен з яких має власні локальні мережі (LAN1 і LAN3 у центральному офісі, LAN2 і LAN4 у керівному офісі);

б) другий рівень: взаємодія між офісами через VPN, що забезпечує централізацію і безпеку передачі даних.

2.1.1.2 Вимоги до способів і засобів зв'язку для інформаційного обміну між компонентами системи

Інформаційний обмін між компонентами системи забезпечується за допомогою наступних засобів зв'язку:

– Ethernet – використовується для підключення пристроїв у локальних мережах кожного офісу;

– VPN – використовується для захищеного з'єднання між центральним і керівним офісами, а також для підключення віддалених працівників;

Вимоги до способів обміну інформацією:

– автоматичний обмін інформацією: всі внутрішні системи повинні автоматично обмінюватися даними через захищені канали;

– документообіг: внутрішній документообіг здійснюється через корпоративні сервери з шифруванням даних;

– телефонний зв'язок: використовується для координації дій між підрозділами у разі необхідності оперативного втручання.

2.1.1.3 Вимоги до характеристик взаємозв'язків створюваної системи із суміжними системами, вимоги до її сумісності

Система повинна забезпечувати інтеграцію з іншими суміжними системами компанії TraderEvolution Global, а саме:

– торговельні платформи – інтеграція з основними торговими платформами компанії для забезпечення синхронізації даних та обміну інформацією в реальному часі;

– обчислювальні кластери – інтеграція для обробки великих обсягів даних та виконання фінансових розрахунків;

– системи безпеки – інтеграція з системами кібербезпеки для забезпечення надійного захисту даних.

Способи обміну інформацією:

– автоматично: інтеграція через API та інші стандартні протоколи;

– пересиланням документів: через захищені канали передачі даних;

– телефоном: у разі необхідності оперативного втручання.

2.1.1.4 Вимоги до режимів функціонування системи

Режими функціонування системи повинні забезпечувати:

– неперервний доступ: система повинна бути доступною 24/7 для користувачів;

– ефективне відновлення: у разі виникнення збоїв система повинна мати можливість швидко відновлюватися до нормального режиму роботи;

– гнучкість: система повинна бути здатною до адаптації у разі змін у вимогах та потребах користувачів;

– безпека: система повинна забезпечувати високий рівень захисту від несанкціонованого доступу та інших загроз безпеці.

2.1.1.5 Вимоги до діагностування системи

Діагностування системи повинно включати:

– виявлення проблем: автоматичне виявлення і сповіщення про можливі проблеми та помилки в роботі системи;

– регулярний моніторинг: постійний моніторинг стану компонентів та процесів системи;

– інструменти аналізу: використання інструментів для аналізу виниклих помилок та аномалій;

– логування подій: ведення детального журналу подій для подальшого аналізу та діагностики.

2.1.1.6 Перспективи розвитку, модернізації системи

Система повинна мати можливість для подальшого розвитку та модернізації, включаючи:

– масштабованість: легке додавання нових підсистем та розширення існуючих;

– інтеграція нових технологій: впровадження нових технологій для підвищення продуктивності та безпеки системи;

– підвищення надійності: впровадження додаткових заходів для забезпечення безперервної роботи та захисту даних.

Показники призначення.

Для комп’ютерної системи підприємства «TraderEvolution» встановлюються наступні показники призначення, які визначають основні характеристики та вимоги до надійності та ефективності мережевої інфраструктури:

– категорія кабелювання – усі кабельні системи повинні відповідати категорії не нижче CAT 6A, забезпечуючи підтримку швидкостей передачі даних до 10 Gbps, що критично важливо для підтримки високошвидкісних операцій торгівлі та обміну даними;

– стандарти кабельної інфраструктури – інформаційна кабельна система має бути побудована відповідно до міжнародного стандарту ISO/IEC 11801, клас EA, забезпечуючи високу надійність та відмінну продуктивність для всіх видів медіа-з’єднань у компанії.

– якість компонентів – використання лише високоякісних мережевих компонентів, які мають сертифікацію ISO 9001, гарантує надійність та довговічність мережевої інфраструктури. Всі компоненти системи, включаючи кабелі, розетки, комутаційні панелі та з'єднувальні шнури, мають проходити стовідсоткове тестування на відповідність цим стандартам;

– відповідність стандартам безпеки – мережева інфраструктура має бути виконана з урахуванням вимог міжнародних стандартів безпеки та захисту даних, включаючи захищені шифруванням VPN-канали для взаємодії між основними та регіональними офісами.

Ці показники встановлюють мінімальні вимоги, яким має відповідати комп’ютерна система компанії для забезпечення її стабільної та безпечної роботи в умовах високих навантажень і забезпечення вимог сучасного бізнес-середовища.

2.1.1.7 Вимоги до експлуатації, технічного обслуговування, ремонту і збереженню

Для забезпечення безперервної та ефективної роботи комп'ютерної системи компанії «TraderEvolution», встановлюються наступні вимоги до експлуатації, технічного обслуговування, ремонту і збереження:

– всі компоненти системи мають підлягати плановим технічним оглядам і діагностиці не рідше ніж раз на шість місяців для вчасного виявлення та усунення можливих несправностей та зношувань;

– під час встановлення та монтажу системи мають бути дотримані всі норми і правила техніки безпеки, з метою захисту персоналу і уникнення пошкоджень обладнання;

– кабелі повинні бути ретельно організовані, марковані для легкого ідентифікування, та мати достатній запас довжини для забезпечення гнучкості підключення, але без надмірного натягу;

– на підприємстві має бути постійний запас необхідних компонентів, включаючи запасні кабелі, роз'єми, комутатори (мінімум 2 штуки), маршрутизатори (мінімум 3 штуки), а також інші витратні матеріали, які можуть знадобитися для швидкого ремонту чи заміни;

– повинна бути ведена ретельна документація всіх проведених технічних оглядів, ремонтних робіт та замін компонентів, що допоможе планувати майбутнє обслуговування та вести історію стану обладнання.

Ці вимоги дозволяють забезпечити стабільність роботи інформаційної системи підприємства та мінімізувати ризик виникнення виробничих збоїв, які можуть негативно вплинути на бізнес-операції компанії.

2.1.1.8 Вимоги до патентної чистоти

Для забезпечення юридичної бездоганності та уникнення порушення авторських прав, всі технології, програмне забезпечення та обладнання, що використовуються в рамках комп'ютерної системи ТОВ «TraderEvolution», повинні відповідати наступним критеріям:

– все програмне забезпечення, що входить до складу комп’ютерної системи, повинно бути ліцензійним. компанія має забезпечити наявність відповідних ліцензій на використання програмного забезпечення, що підтверджує право на його використання та розповсюдження;

– все обладнання має відповідати міжнародним і національним стандартам та нормам. важливо, що всі компоненти системи пройшли необхідне сертифікування та випробування в акредитованих установах;

– перед впровадженням будь-яких нових технологій або компонентів у систему, необхідно провести дослідження на предмет існуючих патентних зобов'язань чи спорів, щоб запобігти можливим юридичним проблемам з іншими компаніями або особами;

– для забезпечення патентної чистоти необхідно регулярно консультуватись з юридичними радниками, які спеціалізуються на правах інтелектуальної власності.

Ці заходи є критично важливими для забезпечення юридичної безпеки компанії та уникнення фінансових та репутаційних ризиків, пов’язаних із порушенням патентних прав та авторських обмежень.

2.1.2 Вимоги до функцій, які виконує КС

2.1.2.1 Підсистема IT-інфраструктури

Підсистема IT-інфраструктури відповідає за забезпечення надійного та ефективного функціонування всіх IT-компонентів компанії. Вона включає в себе як фізичні, так і логічні аспекти інфраструктури, які підтримують роботу компанії на всіх рівнях.

Забезпечення зв'язку між підсистемами:

а) функції:

1) забезпечення високошвидкісного з'єднання між робочими станціями;
2) забезпечення високошвидкісного з'єднання з серверами та периферійними пристроями.

б) технічні вимоги:

1) використання комутаторів Cisco Catalyst 2960X-24TS-L для забезпечення швидкості передачі даних до 1 Гбіт/с.;
2) підключення серверів через прямі кабелі Cat6a для мінімізації затримок;
3) налаштування VLAN для сегментації мережі та підвищення безпеки;
4) використання протоколу RADIUS для централізованої аутентифікації користувачів з резервуванням у локальній базі даних;
5) використання SSL/TLS для шифрування даних на веб-серверах;

Моніторинг та звітність:

а) функції:

1) відстеження стану мережі, генерація звітів про продуктивність та безпеку.

б) технічні вимоги:

1) використання системи моніторингу Nagios для реального часу аналізу стану мережі;
2) налаштування регулярного збору та аналізу логів за допомогою Elasticsearch;
3) генерація звітів про продуктивність мережі щотижня.

Резервне копіювання та відновлення:

а) функції:

1) автоматизоване резервне копіювання даних і швидке відновлення в разі збою;

б) технічні вимоги:

1) використання Veeam Backup & Replication для резервного копіювання конфігурацій мережевих пристроїв та даних;
2) налаштування щоденного резервного копіювання;
3) забезпечення можливості відновлення даних не більше ніж за 1 годину.

2.1.2.2 Підсистема розробки та тестування

Підсистема розробки та тестування забезпечує середовище для створення, тестування та впровадження програмних рішень. Вона охоплює всі етапи життєвого циклу програмного забезпечення, від початкового проектування до кінцевого тестування і випуску.

Забезпечення робочого середовища:

а) функції:

1) забезпечення середовища для розробки та тестування програмного забезпечення.

б) технічні вимоги:

1) використання віртуальних машин на базі VMware для розробки та тестування ПЗ;
2) налаштування автоматизованого тестування за допомогою Jenkins або аналогічної системи CI/CD;
3) підключення розробників через швидкісні комутатори з підтримкою QoS для пріоритезації трафіку.

Контроль якості:

а) функції:

1) автоматизоване тестування розробленого ПЗ для виявлення помилок.

б) технічні вимоги:

1) використання інструментів для моніторингу та аналізу результатів тестування, таких як SonarQube;
2) налаштування тестових середовищ для автоматизованого тестування нових функцій і оновлень.

Впровадження оновлень:

а) функції:

1) розгортання нових версій ПЗ на тестових та продуктивних серверах.

б) технічні вимоги:

1) розгортання оновлень на тестові сервери щотижня;
2) використання Git для управління версіями коду та контролю змін;

3) забезпечення мінімального часу простою під час оновлення – не більше 30 хвилин.

2.1.2.3 Підсистема адміністративна

Підсистема адміністративна відповідає за підтримку адміністративних та організаційних процесів у компанії. Вона включає в себе управління офісними ресурсами, координацію робочих процесів та забезпечення загальної ефективності діяльності компанії.

Управління персоналом:

а) функції:

1) забезпечення зберігання та управління даними про співробітників.

б) технічні вимоги:

1) використання програмного забезпечення для управління людськими ресурсами, наприклад, BambooHR;
2) захищене зберігання даних про співробітників у базах даних;
3) оновлення даних про співробітників щодня.

Фінансове управління:

а) функції:

1) облік та звітність фінансових операцій компанії.

б) технічні вимоги:

1) впровадження системи обліку фінансових операцій, наприклад, QuickBooks або SAP;
2) налаштування автоматизованого створення фінансових звітів з перевіркою на помилки;
3) підготовка фінансових звітів щоквартально.

Адміністративна підтримка:

а) функції:

1) автоматизація рутинних адміністративних задач;

б) технічні вимоги:

1) використання CRM-системи для управління взаємодією з клієнтами та партнерами, наприклад, Salesforce;
2) налаштування автоматизованого виконання рутинних задач за допомогою RPA (роботизованої автоматизації процесів).

2.1.2.4 Підсистема безпеки

Підсистема безпеки охоплює всі аспекти захисту інформаційних активів компанії. Вона включає в себе як фізичні, так і логічні засоби безпеки, що забезпечують захист даних та мереж від загроз та несанкціонованого доступу.

Контроль доступу:

а) функції:

1) забезпечення аутентифікації та авторизації користувачів;

б) технічні вимоги:

1) впровадження багатофакторної аутентифікації (MFA) для всіх критичних систем;
2) налаштування RADIUS для централізованої аутентифікації користувачів.

Шифрування даних:

а) функції:

1) захист даних під час передачі та зберігання;

б) технічні вимоги:

1) використання шифрування AES-256 для захисту даних під час передачі та зберігання;
2) налаштування SSL/TLS для всіх веб-сервісів та внутрішніх порталів.

Моніторинг безпеки:

а) функції:

1) відстеження інцидентів безпеки та реагування на них.

б) технічні вимоги:

1) використання SIEM-систем (наприклад, Splunk) для моніторингу інцидентів безпеки в реальному часі;

2) налаштування автоматизованих оповіщень про підозрілу активність та аномалії в мережевому трафіку.

2.1.2.5 Підсистема комунікацій

Підсистема комунікацій забезпечує ефективний обмін інформацією як всередині компанії, так і з зовнішніми партнерами та клієнтами. Вона охоплює всі засоби зв'язку, включаючи електронну пошту, телефонію та інші засоби корпоративної комунікації.

Внутрішня комунікація:

а) функції:

1) забезпечення зв'язку між співробітниками компанії;

б) технічні вимоги:

1) використання корпоративних месенджерів, таких як Microsoft Teams або Slack, для внутрішнього зв'язку;
2) забезпечення безперебійного доступу до електронної пошти через Microsoft Exchange або Google Workspace.

Зовнішня комунікація:

а) функції:

1) забезпечення зв'язку з клієнтами та партнерами;

б) технічні вимоги:

1) впровадження системи підтримки клієнтів, наприклад, Zendesk, для ефективного управління зверненнями;
2) налаштування VoIP для телефонних дзвінків через інтернет з використанням Cisco Unified Communications Manager.

Відеоконференції:

а) функції:

1) організація відеоконференцій для дистанційної роботи та зустрічей.

б) технічні вимоги:

1) використання Zoom або Microsoft Teams для проведення відеоконференцій;

2) забезпечення якості відео та аудіо на рівні HD з мінімальними затримками.

2.1.2.5 Загальні вимоги до системи

Загальні вимоги до системи визначають основні характеристики, які повинна мати система для забезпечення надійної та ефективної роботи всієї IT-інфраструктури компанії:

– безперебійність роботи – система повинна працювати без простоїв з доступністю не менше 99,9%;

– висока продуктивність – всі операції повинні виконуватися з мінімальними затримками, забезпечуючи швидкий відгук користувачів;

– безпека – всі дані повинні бути захищені за допомогою сучасних методів шифрування та багаторівневої аутентифікації;

– масштабованість – система повинна мати можливість легкого додавання нових вузлів та компонентів без значних змін в архітектурі;

– інтеграція – система повинна інтегруватися з існуючими корпоративними системами та підтримувати стандарти сумісності.

Ці вимоги забезпечують ефективну роботу системи, високу якість послуг, що надаються, та безпеку корпоративної мережі ТОВ «TraderEvolution».

2.1.3 Вимоги до видів забезпечення комп'ютерної системи ТОВ «TraderEvolution»

2.1.3.1 Інформаційне забезпечення

Інформаційне забезпечення включає всі процеси, пов'язані зі зберіганням, обробкою та обміном даними. Воно забезпечує надійну роботу баз даних, логування подій та інтеграцію з іншими системами для підтримки бізнес-процесів компанії.

а) склад та структура даних:

1) зберігання даних про мережеві пристрої їх конфігурації з'єднання та стан системи в базах даних;

2) логи трафіку, події безпеки та системні журнали для аудиту та аналізу;
3) бази даних користувачів, включаючи аутентифікаційні та авторизаційні дані;
4) інформаційне забезпечення підтримується серверами баз даних;

б) Способи організації даних:

1) використання реляційних баз даних (наприклад, PostgreSQL) для зберігання конфігураційних даних та логів;
2) структурування даних у форматах JSON або XML для легкого обміну між компонентами системи;

в) інформаційний обмін між компонентами системи:

1) використання REST API для взаємодії між різними компонентами системи;
2) застосування протоколу MQTT для обміну даними в реальному часі між пристроями;

г) інформаційна сумісність із суміжними системами:

1) підтримка стандартів JSON, XML та протоколів HTTPS для сумісності з іншими системами;
2) використання стандартних протоколів обміну даними для забезпечення інтеграції з існуючими корпоративними системами;

д) системи керування базами даних:

1) використання PostgreSQL для централізованого зберігання даних конфігурацій та логів;
2) впровадження Elasticsearch для зберігання та швидкого пошуку логів подій безпеки;

е) структура процесу збору, обробки, передачі даних:

1) автоматизований збір даних про стан мережі та пристроїв через SNMP;
2) обробка даних за допомогою сервісів обробки подій (наприклад, Logstash) ;

3) передача даних у центральний репозиторій для подальшого аналізу;

ж) контроль, збереження і відновлення даних:

1) регулярне резервне копіювання конфігурацій мережевих пристроїв;
2) використання систем резервного копіювання та відновлення (наприклад, Veeam Backup & Replication) для забезпечення збереження даних.

2.1.3.2 Лінгвістичне забезпечення

Лінгвістичне забезпечення охоплює використання мов програмування та методи взаємодії користувачів із системою. Воно спрямоване на забезпечення ефективної комунікації та інтерфейсів, що підтримують багатомовність і стандарти кодування даних:

а) застосування мов програмування:

1) програмування: використання мов програмування високого рівня (Python, JavaScript) для розробки скриптів та автоматизації задач;
2) бази даних: застосування мови SQL для взаємодії з базами даних;

б) мови взаємодії користувачів і технічних засобів:

1) основна мова: використання англійської мови як основної мови інтерфейсів користувача та технічної документації;
2) багатомовність: підтримка багатомовних інтерфейсів для міжнародних користувачів;

в) кодування і декодування даних:

1) стандарти: використання UTF-8 для кодування текстових даних;
2) сумісність: застосування стандартів кодування для забезпечення сумісності з іншими системами;

г) організація діалогу:

1) інтерфейси: використання інтерфейсів командного рядка (CLI) та веб-інтерфейсів для адміністрування системи;
2) документація: забезпечення інтерактивних підказок та документації для користувачів.

2.1.3.3 Технічне забезпечення

Технічне забезпечення визначає види технічних засобів, необхідних для функціонування системи, а також налаштування мережевої інфраструктури та серверного обладнання.:

а) види технічних засобів:

1) комутатори:

– використання комутаторів Cisco Catalyst 2960X-24TS-L для забезпечення мережевого з'єднання;

2) маршрутизатори:

– використання маршрутизаторів Cisco для забезпечення надійного з'єднання та високої продуктивності;

3) сервери:

– використання фізичних серверів для різних потреб, включаючи сервери для розробки та тестування, сервери баз даних, сервер для централізованого управління проектами та сервер для резервного копіювання.

б) налаштування VLAN:

1) для забезпечення сегментації мережі та підвищення рівня безпеки використовуються віртуальні локальні мережі (VLAN).

2) Сегментація мережі:

– LAN1 (46 вузлів) – основна робоча мережа для розробників та тестувальників:

– VLAN 15 – розробка: включає 15 портів на комутаторах Cisco Catalyst 2960X-24TS-L (1 комутатор);

– VLAN 25 – тестування: включає 14 портів на комутаторах Cisco Catalyst 2960X-24TS-L. (1 комутатор);

– VLAN 99 – управління: цей VLAN використовується для адміністративного доступу та управління мережевим обладнанням.

Загалом, 2 комутатори:

– LAN2 (5 вузлів) – мережа для керівного складу включає 5 портів на комутаторах Cisco Catalyst 2960X-24TS-L. (1 комутатор);

– LAN3 (28 вузлів) – мережа для адміністративного персоналу та технічної підтримки включає 28 портів на комутаторах Cisco Catalyst 2960X-24TS-L. (1 комутатор);

– LAN4 (50 вузлів) – резервна мережа для управління і адміністративних задач: включає 50 портів на комутаторах Cisco Catalyst 2960X-24TS-L. (3 комутатори);

2.1.3.4 Організаційне забезпечення

Структура і функції підрозділів:

– відділ IT – відповідальний за підтримку та моніторинг мережевої інфраструктури;

– відділ розробки – займається розробкою та тестуванням програмного забезпечення;

– адміністративний відділ – забезпечує підтримку користувачів та управління персоналом;

Організація функціонування системи:

– зустрічі – регулярні зустрічі команд для обговорення стану системи та планування оновлень;

– стандарти – впровадження стандартів ITIL для управління IT-послугами;

Захист від помилкових дій персоналу:

– контроль доступу – використання системи контролю доступу для обмеження прав користувачів;

– тренінги – проведення регулярних тренінгів з інформаційної безпеки для персоналу.

2.1.3.5 Методичне забезпечення

Нормативно-технічна документація:

– документація – підготовка документації для всіх компонентів системи, включаючи інструкції з налаштування та експлуатації;

– стандарти – використання стандартів ISO/IEC 27001 для забезпечення інформаційної безпеки;

– політики – розробка внутрішніх політик та процедур для управління IT-ресурсами.

Ці вимоги забезпечують ефективне функціонування комп'ютерної системи ТОВ «TraderEvolution», забезпечуючи високу продуктивність, безпеку та надійність.

2.2 Розробка апаратної частини системи

2.2.1 Розробка структурної схеми комплексу технічних засобів

З’єднання між маршрутизаторами виконується кабелями Serial DCE або крос-кабелі, маршрутизатори з комутаторами поєднуються між собою прямим кабелем так само як і комп’ютери до комутаторів. Для з’єднання комутаторів використовується крос-кабель.

На основі загальної характеристики компанії, її архітектури, кількості підмереж та завдання, розроблено структурну схему комплексу технічних засобів комп’ютерної системи компанії, яка показана на рисунку 2.1.

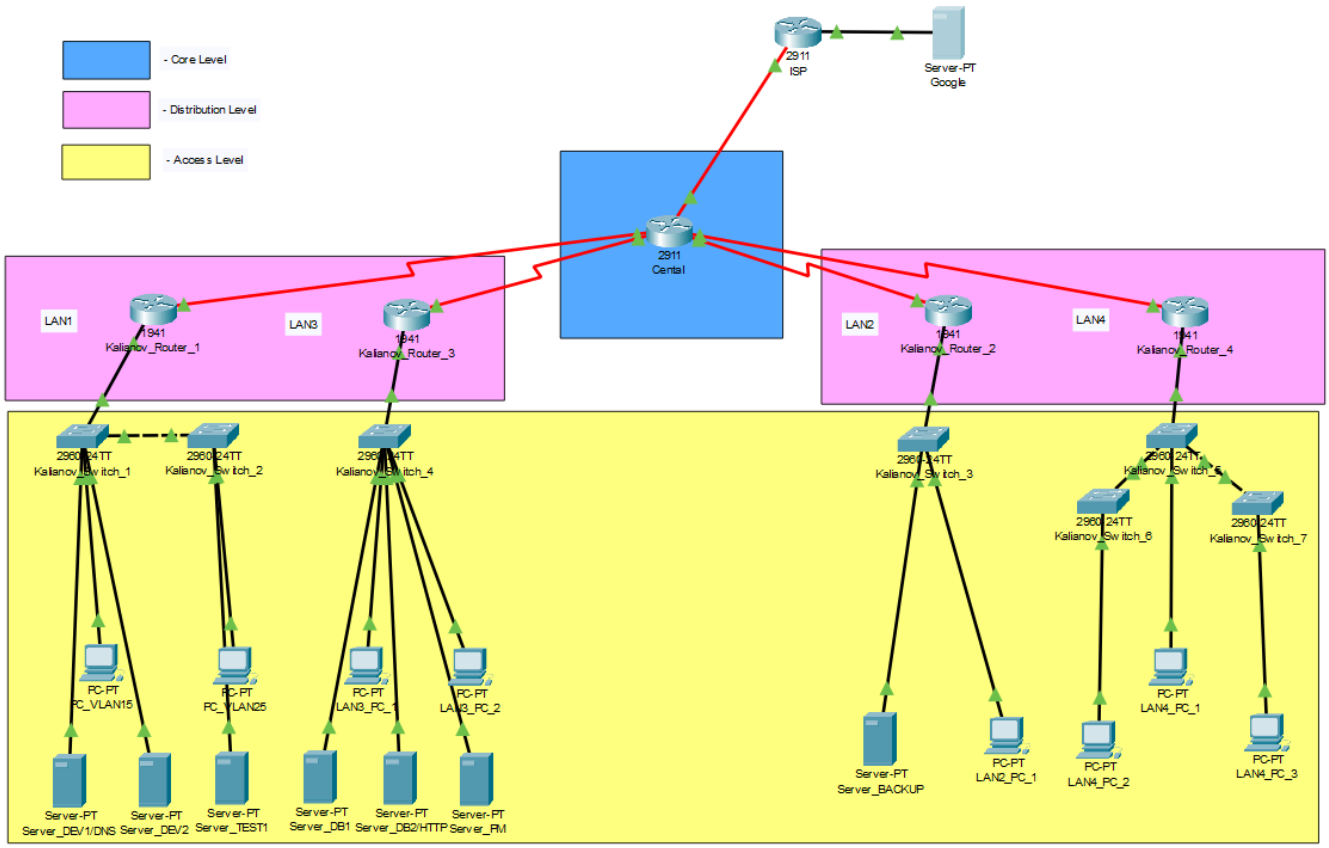

Рисунок 2.1 – Структурна схема комплексу технічних засобів комп'ютерної системи

2.2.2 Розробка специфікації та опису апаратних засобів комп'ютерної системи

Вибір апаратної частини корпоративної мережі є важливим кроком для забезпечення надійності та продуктивності системи. Для досягнення цих цілей було обрано найсучасніші та ефективні маршрутизатори, комутатори, сервери, робочі станції та кабельні системи. Кожен з них має певні технічні характеристики і призначення, що дозволяють оптимізувати роботу корпоративної мережі та забезпечити її стабільне функціонування.

Для мережі LAN1, яка має 46 вузлів, необхідний один роутер, два комутатори та три сервери. Це означає, що залишилося 40 вузлів для ПК розробників та тестувальників. Мережа LAN2, яка має 5 вузлів, потребує один роутер, один комутатор та один сервер, що залишає 2 вузли для ПК адміністративного персоналу. LAN3 має 28 вузлів, що включає один роутер,

один комутатор та три сервери, залишаючи 23 вузли для інших ПК. У LAN4 з 50 вузлами необхідний один роутер та три комутатори, залишаючи 46 вузлів для ПК. LAN1 і LAN3 знаходяться в одній будівлі, а LAN2 і LAN4 – в іншій.

Маршрутизатор Cisco ASR1001-X обраний завдяки його здатності підтримувати до 20 Гбіт/с пропускної здатності, що дозволяє ефективно обробляти великий обсяг трафіку. Вбудовані функції безпеки, такі як шифрування та підтримка VPN, забезпечують захист даних від несанкціонованого доступу.

Комутатор Cisco Catalyst 2960X-24TS-L підтримує швидкість комутації до 216 Гбіт/с і забезпечує швидке та стабільне підключення робочих станцій до мережі. Він має 24 порти 10/100/1000 Ethernet, що дозволяє підключити велику кількість пристроїв. Підтримка VLAN дозволяє сегментувати мережу, що підвищує безпеку та ефективність роботи.

Сервер для резервного копіювання HP ProLiant DL380 Gen10 оснащений двома процесорами Intel Xeon Scalable CPUs, підтримує до 3TB оперативної пам’яті RAM та до 24 NVMe SSD. Використання програмного забезпечення Veeam Backup & Replication забезпечує надійне резервування та відновлення даних, що критично важливо для збереження інформації.

Сервер Dell PowerEdge R740 підтримує віртуалізацію на базі VMware, оснащений двома процесорами Intel Xeon Silver 4110 (2.1GHz, 8 Core), 128GB оперативної пам'яті DDR4 RAM та чотирма SSD-дисками по 2TB кожен. Це забезпечує високу продуктивність та надійність при розробці та тестуванні програмного забезпечення.

Сервери Oracle Server X8-2 мають дві процесори Intel Xeon Gold 6130, 384GB оперативної пам'яті RAM та вісім SSD-дисків по 2.4TB кожен. Використання Oracle Linux забезпечує високу продуктивність та надійність для централізованого зберігання та управління даними.

Сервер Fujitsu PRIMERGY RX2540 M5 має дві процесори Intel Xeon Silver 4210, 256GB оперативної пам'яті DDR4 RAM та чотири SSD-диски по 1.2TB кожен. Цей сервер забезпечує платформу для централізованого управління

проектами, що дозволяє ефективно планувати, координувати та контролювати виконання завдань.

Робочі станції Dell Precision 5820 Tower оснащені процесором Intel Xeon W-2133, оперативною пам’яттю об’ємом 32GB DDR4 2666MHz та графічною картою NVIDIA Quadro P2000. Це забезпечує високопродуктивну обробку графіки та швидку обробку інформації, що є критично важливим для розробки та тестування програмного забезпечення. Жорсткий диск 1TB NVMe SSD гарантує швидке завантаження операційної системи та програмного забезпечення.

Робочі станції HP EliteDesk 800 G5 оснащені процесором Intel Core i5-9500, оперативною пам’яттю об’ємом 16GB DDR4 2400MHz та жорстким диском об’ємом 512GB SSD. Це дозволяє ефективно працювати з офісними додатками та базами даних, забезпечуючи швидке завантаження операційної системи та додатків.

Для забезпечення стабільного та високошвидкісного зв'язку між хостами в підрозділі IT-інфраструктури обирається структурована кабельна система категорії 6A (Cat 6A), яка підтримує пропускну здатність до 10 Гбіт/с. Використання таких кабелів гарантує високу швидкість передачі даних, що є критично важливим для обробки великих обсягів інформації в процесі розробки та тестування програмного забезпечення. Підключення робочих станцій до мережевих комутаторів забезпечується комутаторами Cisco Catalyst 2960X-24TS-L, які мають 24 порти 10/100/1000 Ethernet і підтримують VLAN для сегментації мережі, що підвищує безпеку та ефективність роботи мережі.

Для забезпечення безперебійної роботи мережевого обладнання та серверів використовуються джерела безперебійного живлення (UPS). Враховуючи розподіл обладнання та потужність серверів, комутаторів і роутерів, знадобиться 5 UPS. Потужність середнього сервера становить приблизно 300-500 Вт. Комутатор споживає близько 100-200 Вт, а роутер - 200-300 Вт. Використання APC Smart-UPS, які забезпечують 1500 ВА потужності, дозволяє

покрити потреби в живленні для приблизно 3 – 4 серверів або 5 – 7 комутаторів/маршрутизаторів.

Сучасні сервери та робочі станції мають вбудовані мережеві адаптери, які відповідають вимогам мережі (швидкість до 1 Гбіт/с). Додаткові мережеві карти (Intel Ethernet Server Adapter I350-T4) будуть використовуватися лише у випадках, якщо виникне потреба у додаткових портах або більш високій пропускній здатності.

Для побудови мережевої інфраструктури використовуються кабелі Cat 6A, які підтримують пропускну здатність до 10 Гбіт/с. Враховуючи загальну кількість вузлів та пристроїв, знадобиться приблизно 1000 метрів кабелів Cat 6A. Також використовуються патч-корди (30 штук різної довжини), патч-панелі (20 штук по 24 порти), органайзери кабелів (10 штук) та розетки RJ45 (50 штук).

Для розміщення обладнання в кожній будівлі знадобляться серверні шафи. LAN1 та LAN3, які знаходяться в одному будинку і мають 6 серверів та інше обладнання, потребують 2 серверні шафи. Для LAN2 та LAN4, які знаходяться в іншому будинку і мають тільки 1 сервер і інше обладнання, потрібна 1 серверна шафа.

У таблиці 2.1 представлена специфікація серверів, маршрутизаторів та комутаторів корпоративної мережі "TraderEvolution".

Таблиця 2.1 – Специфікація обладнання.

Найменування і технічна характеристика	Тип, марка, позначення документа	Одиниці виміру	Кількість	Примітки
1	2	3	4	5
Маршрутизатор Cisco ASR1001-X	Cisco	Шт.	4	Головний маршрутизатор
Комутатор Cisco Catalyst 2960X-24TS-L	Cisco	Шт.	7	Для сегментації VLAN і локальних мереж

Продовження таблиці 2.1

1	2	3	4	5
Сервер для резервного копіювання	HP ProLiant DL380 Gen10	Шт.	1	З встановленим ПЗ Veeam Backup & Replication
Сервер для розробки та тестування	Dell PowerEdge R740	Шт.	3	Для розробки та тестування ПЗ
Сервер баз даних	Oracle Server X8-2	Шт.	2	Для централізованого зберігання даних
Сервер для централізованого управління проектами	Fujitsu PRIMERGY RX2540 M5	Шт.	1	Для управління проектами
Робоча станція для розробників та тестувальників	Dell Precision 5820 Tower	Шт.	40	Для розробки та тестування
Робоча станція для адміністративного персоналу	HP EliteDesk 800 G5	Шт.	25	Для офісних задач та комунікацій
Робоча станція для інших ПК	Dell Precision 5820 Tower	Шт.	46	Для роботи інших ПК
Кабелі Cat 6A	-		1000	Для підключення робочих станцій
Патч-корди Cat 6A	-	Шт.	30	Для підключення пристроїв до патч-панелей та комутаторів
Патч-панелі Keystone	-	Шт.	20	Для організації кабельної інфраструктури у серверних стійках
Органайзери кабелів	-	Шт.	10	Для підтримки порядку у серверних стійках

Продовження таблиці 2.1

1	2	3	4	5
Розетки RJ45	-	Шт.	50	Для підключення кабелів до пристроїв у різних кімнатах
Джерело безперебійного живлення (UPS)	APC Smart-UPS	Шт.	5	Для серверів та ключових мережевих пристроїв
Серверна шафа	-	Шт.	3	42U серверна шафа для розміщення обладнання

2.2.3 Розрахунок інтенсивності вихідного трафіку найбільшої локальної мережі підприємства

Для обчислення інтенсивності вихідного трафіку у найбільшій локальній мережі (LAN4) компанії TraderEvolution необхідно визначити основні параметри мережі. В даному випадку пропускна здатність лінії вихідного каналу дорівнює 1000 Мбіт/с.

Середня інтенсивність трафіку становить 154 кадри/с, а середня довжина повідомлення – 650 байт. Ці параметри дозволяють розрахувати пропускну здатність та інші показники продуктивності мережі.

Припустимо, що всі користувачі одночасно використовують послуги. Для розрахунку пропускної здатності LAN4, яка складається з 50 вузлів, використовуємо формулу:

$$P = \mu * L * N * 8 \qquad (2.1)$$

де $\mu = 154$ – середня інтенсивність трафіку, кадрів/с;

$L = 650$ – середня довжина повідомлення, байт;

$N = 50$ од. – кількість вузлів у мережі;

8 – коефіцієнт для перетворення байтів у біти.

$$P = 154 * 650 * 50 * 8 = 40{,}04 \text{ Мбіт/с}$$

Отримані результати не перевищують заданих параметрів мережі по вихідному каналу (1000 Мбіт/с), тому перевантаження не очікується. Комутатор рівня доступу передає трафік до маршрутизатора через вихідний порт зі швидкістю передачі даних 1000 Мбіт/с. Для визначення загального навантаження на комутатор, використовуємо формулу:

$$\mu_{вих} = \frac{1000000000}{L * 8} \tag{2.2}$$

$$\mu_{вих} = \frac{1000000000}{650 * 8} = 192307{,}69 \text{ пакетів/с}$$

Кількість підключення до комутатора.

Оскільки кожне джерело в середньому виробляє 154 пакети/с, кількість приєднань, якими обмежений комутатор рівня доступу, розраховується наступним чином:

$$N = \frac{\mu_{вих}}{\mu} \tag{2.3}$$

$$N = \frac{192307{,}69}{154} = 1248{,}75 \text{ джерел}$$

Інтенсивність вихідного трафіку для LAN4, яка складається з 50 вузлів, розраховується за формулою:

$$\lambda = N * \mu \tag{2.4}$$

$$\lambda = 50 \times 154 = 7700 \text{ пакетів/с}$$

Коефіцієнт затримки для комутатора рівня доступу визначається як:

$$\rho = \frac{\lambda}{\mu_{вих}} \tag{2.5}$$

$$\rho = \frac{7700}{192307{,}69} = 0{,}04004$$

Коефіцієнт зайнятості комутатора рівня доступу розраховується як:

$$\rho_{зан} = \frac{\rho}{1 - \rho} \tag{2.6}$$

$$\rho_{зан} = \frac{0{,}04004}{1 - 0{,}04004} = 0{,}04171$$

Середня затримка кадру, пов’язана з чергою M/M/1, визначається за формулою:

$$T = \frac{1}{\mu_{вих} - \lambda} \tag{2.7}$$

$$T = \frac{1}{192307{,}69 - 77001} = 5{,}42 \text{ мкс}$$

Середня довжина черги розраховується як:

$$L_{чер} = \frac{\rho^2}{1 - \rho} \tag{2.8}$$

$$L_{чер} = \frac{0{,}04004^2}{1 - 0{,}04004} = 0{,}00167$$

Середній час перебування пакета у черзі визначається за формулою:

$$T_{оч} = \frac{L_{чер}}{\lambda} \tag{2.9}$$

$$T_{оч} = \frac{0{,}00167}{7700} = 0{,}217 \text{ мкс}$$

Ці значення свідчать про те, що мережа працюватиме стабільно та відповідатиме вимогам щодо затримки передачі даних.

3 РОЗРОБКА КОРПОРАТИВНОЇ МЕРЕЖІ

3.1 Розрахунок схеми адресації корпоративної мережі

Розрахунок схеми адресації корпоративної мережі є важливим етапом при плануванні та організації комп'ютерної мережі підприємства. VLSM (Variable Length Subnet Mask) – це метод адресації, який дозволяє розділяти мережу на підмережі різного розміру, що надає гнучкість у плануванні та використанні IP-адресного простору.

При розрахунку схеми адресації за допомогою методу VLSM необхідна ретельна планування та аналіз мережі. Першим кроком є визначення числа відділів та підрозділів у мережі, а також чисельності комп'ютерів у кожному з них. За цією інформацією будується ієрархічна структура мережі, яка допомагає визначити оптимальне число підмереж.

Далі, за допомогою математичних розрахунків визначаються маски підмереж, які забезпечують ефективне використання IP-адресного простору. Оскільки маски підмереж можуть мати різну довжину, то це дозволяє створити підмережі з різним числом хостів, що зменшує кількість не використаних IP-адрес.

В результаті розрахунку схеми адресації корпоративної мережі методом VLSM отримуємо зручну для керування та конфігурації мережею, яка дозволяє ефективно використати IP-адресний простір. Крім того, метод VLSM допомагає в боротьбі з надмірним використанням IP-адрес, що є важливим з погляду безпеки та стабільності мережі.

Виконаємо розрахунки відповідно до вимог в розділі 2.

LAN4 (50 вузлів).

LAN4 є резервною мережею для управління та адміністративних завдань і містить 50 вузлів. Для підмережі LAN4 необхідно виділити 64 адреси, оскільки це найближче значення більше 50 (2^6= 64).

Адреса мережі: 192.168.15.0/26

Маска підмережі: 255.255.255.192 (/26)

Бінарний вигляд адреси мережі: 192.168.15.00000000

Широкомовна адреса: 192.168.15.63 (192.168.15.00111111)

Діапазон допустимих адрес: 192.168.15.1 - 192.168.15.62

LAN1 (46 вузлів).

LAN1 є основною робочою мережею, яка обслуговує 46 вузлів, включаючи розробників та тестувальників. Для підмережі LAN1 необхідно виділити 64 адреси, оскільки це найближче значення більше 46 (2^6= 64).

Адреса мережі: 192.168.15.64/26

Маска підмережі: 255.255.255.192 (/26)

Бінарний вигляд адреси мережі: 192.168.15.01000000

Широкомовна адреса: 192.168.15.127 (192.168.15.01111111)

Діапазон допустимих адрес: 192.168.15.65 - 192.168.15.126

VLAN 15 – розробка (15 вузлів).

Для підмережі VLAN 15 необхідно виділити 32 адреси, оскільки це найближче значення більше 15 (2^5= 32).

Адреса мережі: 192.168.15.64/27

Маска підмережі: 255.255.255.224 (/27)

Бінарний вигляд адреси мережі: 192.168.15.01000000

Широкомовна адреса: 192.168.15.95 (192.168.15.01011111)

Діапазон допустимих адрес: 192.168.15.65 - 192.168.15.94

VLAN 25 – тестування (14 вузлів).

Для підмережі VLAN 25 необхідно виділити 16 адрес, оскільки це найближче значення більше 14 (2^4 = 16).

Адреса мережі: 192.168.15.96/28

Маска підмережі: 255.255.255.240 (/28)

Бінарний вигляд адреси мережі: 192.168.15.01100000

Широкомовна адреса: 192.168.15.111 (192.168.15.01101111)

Діапазон допустимих адрес: 192.168.15.97 - 192.168.15.110

VLAN 99 (14 вузлів).

Для підмережі VLAN 99 необхідно виділити 16 адрес, оскільки це найближче значення більше 14 (2^4= 16).

Адреса мережі: 192.168.15.112/28

Маска підмережі: 255.255.255.240 (/28)

Бінарний вигляд адреси мережі: 192.168.15.01110000

Широкомовна адреса: 192.168.15.127 (192.168.15.01111111)

Діапазон допустимих адрес: 192.168.15.113 - 192.168.15.126

LAN3 (28 вузлів).

LAN3 призначена для адміністративного персоналу та технічної підтримки і містить 28 вузлів. Для підмережі LAN3 необхідно виділити 32 адреси, оскільки це найближче значення більше 28 (2^5= 32).

Адреса мережі: 192.168.15.128/27

Маска підмережі: 255.255.255.224 (/27)

Бінарний вигляд адреси мережі: 192.168.15.10000000

Широкомовна адреса: 192.168.15.159 (192.168.15.10111111)

Діапазон допустимих адрес: 192.168.15.129 - 192.168.15.158

LAN2 (5 вузлів).

LAN2 обслуговує управлінський склад підприємства і містить 5 вузлів. Для підмережі LAN2 необхідно виділити 8 адрес, оскільки це найближче значення більше 5 (2^3 = 8).

Адреса мережі: 192.168.15.160/29

Маска підмережі: 255.255.255.248 (/29)

Бінарний вигляд адреси мережі: 192.168.15.10100000

Широкомовна адреса: 192.168.15.167 (192.168.15.10101111)

Діапазон допустимих адрес: 192.168.15.161 - 192.168.15.166

Адресації підмереж представлені у таблиці 3.1

Таблиця 3.1 – Адресації підмереж

Назва мережі	Кіль. вузлів	Номер мережі	Маска мережі	Початкове значення діапазону можливих адрес вузлів у підмережі	Кінцеве значення діапазону можливих адрес вузлів у підмережі
LAN4	50	192.168.15.0	255.255.255.192	192.168.15.1	192.168.15.62
LAN 1	46	192.168.15.64	255.255.255.192	192.168.15.65	192.168.15.126
VLAN 15	15	192.168.15.64	255.255.255.224	192.168.15.65	192.168.14.94
VLAN 25	14	192.168.15.96	255.255.255.240	192.168.15.97	192.168.15.110
VLAN 99	14	192.168.15.112	255.255.255.240	192.168.15.113	192.168.15.126
LAN3	28	192.168.15.128	255.255.255.224	192.168.15.129	192.168.15.158
LAN2	5	192.168.15.160	255.255.255.248	192.168.15.161	192.168.15.166

Для забезпечення з'єднання між двома офісами компанії використовується технологія VPN. Це дозволяє забезпечити безпечне і надійне з'єднання між віддаленими офісами, що знаходяться на великій відстані один від одного. Для VPN-з'єднання виділяється окремий блок IP-адрес, які використовуються для інтерфейсів маршрутизаторів з обох сторін з'єднання. Виділені адреси представлені в таблиці 3.1.

3.2 Розрахунок схеми адресації пристроїв

Призначення IP-адрес інтерфейсам та підінтерфейсам маршрутизаторів

Перші можливі для використання IP-адреси в кожній підмережі були призначені інтерфейсам і підінтерфейсам маршрутизаторів. Це дозволяє легко ідентифікувати маршрутизатори в кожній підмережі та забезпечує їхню коректну взаємодію з іншими мережевими пристроями.

Другі можливі IP-адреси у кожній підмережі були призначені комутаторам. Це забезпечує централізоване управління і спрощує діагностику мережевих проблем.

Для серверів використовувалися IP-адреси за правилом: перший можливий адрес у мережі плюс 9 та номер варіанту студента (5). Це забезпечує унікальність і легкість управління IP-адресами серверів.

Останні з використовуваних IP-адрес у кожній підмережі були призначені кінцевим вузлам. Це дозволяє максимально ефективно використовувати адресний простір і забезпечує чітку структуру адресації.

Для кінцевих пристроїв у VLAN використовується протокол DHCP для динамічного призначення IP-адрес. Це забезпечує гнучкість у управлінні мережею і спрощує додавання нових пристроїв. DHCP дозволяє автоматично призначати IP-адреси кінцевим пристроям з заданого пулу, зменшуючи ризик виникнення конфліктів адрес.

Таблиця 3.2 – Схема адресації пристроїв.

Пристрій	Інтерфейс	IP-адреса	Маска	Шлюз	VLAN
Central	Se0/2/0	172.16.0.9	/30	-	-
	Se0/2/1	172.16.0.13	/30	-	-
	Se0/3/0	172.16.0.1	/30	-	-
	Se0/3/1	172.16.0.5	/30	-	-
	Gig0/1/0	209.165.202.2	/30	-	-
ISP	Gig0/3/0	209.165.202.1	/30	-	-
	Gig0/0	8.8.8.1	/8		
LAN1					
Kalianov_Router_1	Se0/1/0	172.16.0.2	/30	-	-
	Gig0/1.15	192.168.15.65	/27	-	15
	Gig0/1.25	192.168.15.97	/28	-	25
	Gig0/1.99	192.168.15.113	/28	-	99
Kalianov_Switch_1	Vlan99	192.168.15.66	/27	192.168.15.112	99
Kalianov_Switch_2		192.168.15.98	/28		99
PC_VLAN15	F0/0	192.168.15.75	/27	192.168.15.65	15
PC_VLAN25		192.168.15.107	/28	192.168.15.97	25
Server_DEV1/DNS		192.168.15.79	/27	192.168.15.97	15
Server_DEV2		192.168.15.80	/27	192.168.15.97	15
Server_TEST1		192.168.15.111	/27	192.168.15.94	25
LAN2					
Kalianov_Router_2	Se0/1/0	172.16.0.10	/30	-	-
	Gig0/1	192.168.15.161	/29	-	-
Kalianov_Switch_3	Vlan1	192.168.15.162	/29	192.168.15.161	1
LAN2_PC_1	F0/0	192.168.15.166	/29	192.168.15.161	-

Продовження таблиці 3.2

Server_BACKUP		192.168.15.163	/29	192.168.15.161	-
LAN3					
Kalianov_Router_3	Se0/1/0	172.16.0.6	/30	-	-
	Gig0/1	192.168.15.129	/27	-	-
Kalianov_Switch_4	Vlan1	192.168.15.130	/27	192.168.15.129	1
LAN3_PC_1	F0/0	192.168.15.158	/27	192.168.15.129	-
LAN3_PC_2		192.168.15.157	/27	192.168.15.129	-
Server_DB1		192.168.15.143	/27	192.168.15.129	-
Server_DB2		192.168.15.144	/27	192.168.15.129	-
Server_PM		192.168.15.145	/27	192.168.15.129	-
LAN4					
Kalianov_Router_4	Se0/1/0	172.16.0.14	/30	-	-
	Gig0/1	192.168.15.1	/26	-	-
Kalianov_Switch_5	Vlan1	192.168.15.2	/26	192.168.15.1	1
Kalianov_Switch_6		192.168.15.3	/26	192.168.15.1	1
Kalianov_Switch_7		192.168.15.4	/26	192.168.15.1	1
LAN4_PC_1	F0/0	192.168.15.62	/26	192.168.15.1	-
LAN4_PC_2		192.168.15.61	/26	192.168.15.1	-
LAN4_PC_3		192.168.15.60	/26	192.168.15.1	-

3.3 Розробка логічної схеми корпоративної мережі

Корпоративна мережа компанії «TraderEvolution» побудована на основі топології "зірка", що забезпечує ефективне управління ресурсами мережі та високу продуктивність. Основні підмережі (LAN1, LAN2, LAN3, LAN4) з'єднані між собою та з хмарним середовищем через захищені VPN-канали, що гарантує безпечний обмін даними між офісами. Використання VLAN дозволяє сегментувати мережу, підвищуючи рівень безпеки та керованості.

Корпоративна мережа побудована з урахуванням вимог до безпеки, надійності та масштабованості. Основні підмережі (LAN1, LAN2, LAN3, LAN4) з'єднані між собою та з хмарою через захищені VPN-канали, забезпечуючи безпечний обмін даними між офісами. Використання VLAN дозволяє сегментувати мережу, підвищуючи рівень безпеки та керованості.

Така топологічна схема дозволяє ефективно керувати ресурсами мережі, забезпечуючи високу продуктивність та безперебійність роботи.

На рисунку 3.1 представлена топологічна схема корпоративної мережі.

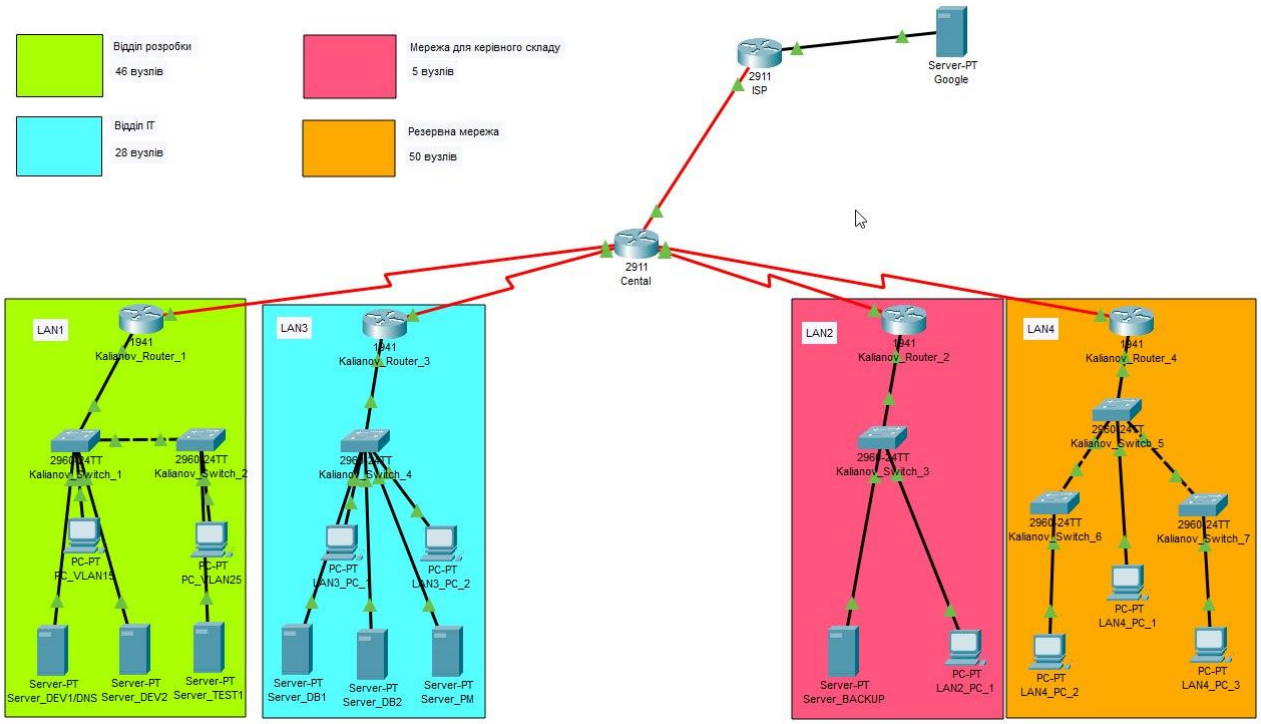

Рисунок 3.1 – Топологічна схема корпоративної мережі

3.4 Налаштування та перевірка роботи комп’ютерної системи

3.4.1 Базове налаштування конфігурації пристроїв

Для виконання базового налаштування конфігурації пристроїв у корпоративній мережі було виконано декілька важливих кроків. Спочатку всі пристрої отримали унікальні назви відповідно до правила: прізвище студента, тип пристрою, номер пристрою. Наприклад, для маршрутизатора використовувалась назва Kalianov_Router_1, а для комутатора – Kalianov_Switch_1. Це правило допомагає у подальшому управлінні обладнанням та орієнтуванні в мережі, особливо у випадку її розширення.

На всіх пристроях були встановлені паролі для консолі та віртуальних терміналів (vty) з використанням пароля cisco, а для привілейованого режиму – class. Для забезпечення інформаційної безпеки, усі паролі, що зберігаються у відкритому вигляді, були зашифровані.

Для віддаленого управління мережевими пристроями був налаштований протокол SSH. Це дозволяє забезпечити захищене підключення до пристроїв, шифруючи всі дані, які передаються між клієнтом та сервером. Для цього було

створено RSA-ключ довжиною 1024 біт. На всіх пристроях був призначений користувач за правилом: група та прізвище, наприклад 12320z1_Kalianov, з паролем admincisco.

На всіх пристроях було встановлено банер повідомлення дня (MOTD) із текстом: Welcome to Kalianov Router/Switch. Ці налаштування забезпечують базову безпеку і зручність управління пристроями в корпоративній мережі.

Для прикладу наведена конфігурація маршрутизатора Kalianov_Router_1.

```
enable
configure terminal
line console 0
password cisco
login
exit
line vty 0 4
password cisco
login
exit
enable secret class
service password-encryption
banner motd #Welcome to Kalianov Router#
username 12320z1_Kalianov password admincisco
ip domain-name Kalianov_Router_3
crypto key generate rsa
modulus 1024
line vty 0 4
transport input ssh
login local
exit
aaa new-model
aaa authentication login default local
```

3.4.2 Налаштування маршрутизаторів корпоративної мережі

Для забезпечення ефективної маршрутизації в корпоративній мережі був використаний протокол OSPF, який підтримує множинні шляхи, має швидкий час збіжності та створює мінімальний службовий трафік. Відповідно до розрахунків, представлених у таблиці 3.2, на маршрутизаторі були виконані наступні дії.

Спочатку були оголошені безпосередньо підключені мережі, що дозволило іншим маршрутизаторам знати про існування цих мереж. Це було досягнуто за допомогою команд:

```
router ospf 5
log-adjacency-changes
network 192.168.15.64 0.0.0.63 area 0
network 172.16.0.0 0.0.0.255 area 0
```

Щоб зменшити службовий трафік і підвищити безпеку, ми відключили поширення оновлень маршрутизації на інтерфейси, підключені до локальних мереж. Виконання цієї задачі включало введення команд:

```
router ospf 5
passive-interface GigabitEthernet0/1
```

Далі, змінили еталонну пропускну спроможність для обчислення вартості маршрутів через інтерфейси Gigabit Ethernet на значення 1000, що дозволило оптимізувати маршрутизацію в мережі:

```
router ospf 5
auto-cost reference-bandwidth 1000
```

Також, задали пропускну спроможність на серійних інтерфейсах на рівні 128 Кб/с і встановили вартість метрики 7500. Це забезпечило точнішу оцінку маршрутів через серійні інтерфейси:

```
interface Serial0/1/0
bandwidth 128
ip ospf cost 7500
```

Для підвищення безпеки доступу до маршрутизаторів була налаштована служба AAA з використанням RADIUS-сервера, що дозволяє централізовано керувати обліковими записами користувачів і підвищити безпеку мережі.

Налаштування включали команди:

```
aaa new-model
aaa authentication login default group radius local
aaa authentication login vty local
aaa authorization exec default local
aaa accounting exec default start-stop group radius
```

```
radius-server host 192.168.15.145 auth-port 1645
radius-server key radius123
line console 0
login authentication default
exit
line vty 0 15
login authentication default
exit
```

Також були налаштовані підінтерфейси для VLAN. Це дозволяє маршрутизатору обробляти трафік з різних VLAN, що забезпечує ефективне розділення мережевого трафіку.

```
interface GigabitEthernet0/1.15
encapsulation dot1Q 15
ip address 192.168.15.65 255.255.255.224
interface GigabitEthernet0/1.25
encapsulation dot1Q 25
ip address 192.168.15.97 255.255.255.240
interface GigabitEthernet0/1.99
encapsulation dot1Q 99
ip address 192.168.15.113 255.255.255.240
```

Ці налаштування дозволяють створити окремі логічні мережі для різних відділів організації, забезпечуючи підвищену безпеку та ефективність управління мережею.

3.4.3 Налаштування доступу в Інтернет

Для забезпечення стабільного і надійного доступу до Інтернету в корпоративній мережі була виконана конфігурація пограничного маршрутизатора. Ця конфігурація включає налаштування NAT (Network Address Translation), що дозволяє внутрішнім IP-адресам отримувати доступ до зовнішніх мереж, а також налаштування сервера HTTP для відображення веб-сайту з відомостями про кваліфікаційну роботу.

NAT використовується для трансляції приватних IP-адрес у публічні, що дозволяє пристроям у внутрішній мережі отримувати доступ до Інтернету, використовуючи спільний публічний IP-адрес. Це не тільки забезпечує безпеку, приховуючи внутрішні структури мережі, але й дозволяє ефективно використовувати обмежену кількість публічних IP-адрес.

Спочатку ми створили розширений список контролю доступу (ACL), який дозволяє трафіку з внутрішньої мережі 192.168.15.0/24 отримувати доступ до будь-яких зовнішніх ресурсів:

```
ip access-list extended _5
permit ip 192.168.15.0 0.0.0.255 any
```

Після цього ми налаштували пул публічних IP-адрес, які будуть використовуватися для трансляції:

```
ip nat pool Internet 209.165.200.5 209.165.200.30 netmask 255.255.255.224
```

Далі, ми вказали маршрутизатору використовувати цей пул для трансляції внутрішніх адрес, які відповідають списку доступу, і дозволили перевантаження (overload), щоб всі внутрішні адреси могли використовувати одну або кілька публічних IP-адрес одночасно:

```
ip nat inside source list 5 pool Internet overload
```

Також була налаштована статична трансляція для сервера, щоб він завжди використовував один і той самий публічний IP-адрес:

```
ip nat inside source static 192.168.15.144 209.165.200.4
```

На інтерфейсах маршрутизатора були налаштовані внутрішні і зовнішні зони NAT. Інтерфейс Serial0/1/0 був налаштований як зовнішній:

```
interface Gig0/1/0
ip nat outside
```

А інтерфейс GigabitEthernet0/1 був налаштований як внутрішній:

```
interface Se0/3/0
ip nat inside
interface Se0/3/1
ip nat inside
interface Se0/2/0
ip nat inside
interface Se0/2/1
ip nat inside
```

Завершивши налаштування, конфігурація була збережена:

```
exit
write memory
```

Налаштування HTTP сервера.

Для відображення веб-сайту з інформацією про тему та завдання кваліфікаційної роботи студента був налаштований сервер HTTP. Цей сервер був налаштований таким чином, щоб при вводі в браузері адреси http://123.dnipro.ua або http://209.165.200.4 відображалася відповідна веб-сторінка.

Налаштування сервера включали:

– встановлення відповідного програмного забезпечення для веб-сервера;

– розміщення веб-сторінки з необхідною інформацією на сервері;

– налаштування доменного імені (http://123.dnipro.ua) для відображення публічного IP-адресу (http://209.165.200.4).

Це дозволило забезпечити доступ до необхідної інформації з будь-якого пристрою в мережі, підвищуючи зручність та ефективність використання мережевих ресурсів.

3.4.4 Перевірка роботи комп’ютерної системи

На рисунку 3.2 представлений результат перевірки базових налаштувань пристроїв на прикладі Kalianov_Router_1.

Після налаштування протоколу OSPF на маршрутизаторах була проведена перевірка роботи маршрутизації. За допомогою команди show ip route були отримані результати, що відображають правильність налаштувань маршрутизації (рис.3.3).

Для перевірки налаштувань DNS і HTTP серверів було відкрито сторінку http://123.dnipro.ua. Це дозволило перевірити коректність роботи DNS-сервера, який повинен перетворити доменне ім'я в IP-адресу, а також коректність роботи HTTP-сервера, який повинен обробити запит і відобразити веб-сторінку.

```
Kalianov_Router_1#show running-config | include hostname
hostname Kalianov_Router_1
Kalianov_Router_1#show running-config | include line con|line vty|passw
service password-encryption
username 12320zl_Kalianov password 7 082048430017061E010803
line con 0
 password 7 0822455D0A16
line vty 0 4
line vty 5 15
Kalianov_Router_1#show running-config | include enable secret
enable secret 5 $1$mERr$9cTjUIEqNGurQiFU.ZeCi1
Kalianov_Router_1#show running-config | include service password-encryp
service password-encryption
Kalianov_Router_1#show running-config | include banner motd
banner motd ^C Unauthorized access is prohibited. ^C
Kalianov_Router_1#show running-config | include transport input ssh
 transport input ssh
Kalianov_Router_1#show running-config | include username
username 12320zl_Kalianov password 7 082048430017061E010803
Kalianov_Router_1#show running-config | include ip domain-name
ip domain-name Kalianov_Router_1
```

Рисунок 3.2 – Результат перевірки базових налаштувань

```
     172.16.0.0/16 is variably subnetted, 8 subnets, 2 masks
C       172.16.0.0/30 is directly connected, Serial0/3/0
L       172.16.0.1/32 is directly connected, Serial0/3/0
C       172.16.0.4/30 is directly connected, Serial0/3/1
L       172.16.0.5/32 is directly connected, Serial0/3/1
C       172.16.0.8/30 is directly connected, Serial0/2/0
L       172.16.0.9/32 is directly connected, Serial0/2/0
C       172.16.0.12/30 is directly connected, Serial0/2/1
L       172.16.0.13/32 is directly connected, Serial0/2/1
     192.168.15.0/24 is variably subnetted, 6 subnets, 4 masks
O       192.168.15.0/26 [110/65] via 172.16.0.14, 01:14:03, Serial0/2/1
O       192.168.15.64/27 [110/65] via 172.16.0.2, 01:14:03, Serial0/3/0
O       192.168.15.96/28 [110/65] via 172.16.0.2, 01:14:03, Serial0/3/0
O       192.168.15.112/28 [110/65] via 172.16.0.2, 01:14:03, Serial0/3/0
O       192.168.15.128/27 [110/65] via 172.16.0.6, 01:14:03, Serial0/3/1
O       192.168.15.160/29 [110/65] via 172.16.0.10, 01:14:03, Serial0/2/0
     209.165.202.0/24 is variably subnetted, 2 subnets, 2 masks
C       209.165.202.0/30 is directly connected, GigabitEthernet0/1/0
L       209.165.202.2/32 is directly connected, GigabitEthernet0/1/0
S*   0.0.0.0/0 [1/0] via 209.165.202.1
```

Рисунок 3.3 – Таблиця маршрутизації

Як видно на скріншоті, сторінка відкривається коректно, що підтверджує правильність налаштувань обох серверів. Це означає, що як DNS-сервер, так і HTTP-сервер функціонують належним чином і здатні обслуговувати запити користувачів (рис.3.4).

Перегляд конфігурації пулу DHCP і призначених адрес допомагає впевнитися в правильності роботи DHCP сервера. Ці команди дозволяють адміністратору побачити, які адреси були виділені клієнтам, а також загальну

інформацію про налаштовані пули, включаючи діапазони адрес і кількість використаних адрес (рис.3.5).

Рисунок 3.4 – Перевірка налаштувань DNS та HTTP серверів через відкриття сторінки 123.dnipro.ua

```
Kalianov_Router_1#show ip dhcp pool

Pool pollvlan15 :
 Utilization mark (high/low)     : 100 / 0
 Subnet size (first/next)        : 0 / 0
 Total addresses                 : 30
 Leased addresses                : 1
 Excluded addresses              : 5
 Pending event                   : none

 1 subnet is currently in the pool
 Current index        IP address range                    Leased/Excluded/Total
 192.168.15.65        192.168.15.65    - 192.168.15.94     1    / 5     / 30

Pool pollvlan25 :
 Utilization mark (high/low)     : 100 / 0
 Subnet size (first/next)        : 0 / 0
 Total addresses                 : 14
 Leased addresses                : 1
 Excluded addresses              : 5
 Pending event                   : none

 1 subnet is currently in the pool
 Current index        IP address range                    Leased/Excluded/Total
 192.168.15.97        192.168.15.97    - 192.168.15.110    1    / 5     / 14
Kalianov_Router_1#show ip dhcp binding
IP address       Client-ID/              Lease expiration        Type
                 Hardware address
192.168.15.75    0006.2ADB.561C           --                     Automatic
192.168.15.107   00D0.58B6.D59E           --                     Automatic
Kalianov_Router_1#
```

Рисунок 3.5 – Перевірка налаштувань DHCP пулу та призначених адрес

Для забезпечення коректної роботи VLAN та транкових портів на комутаторі було виконано перевірку за допомогою команд show vlan та show

interfaces trunk. Ці команди дозволяють отримати інформацію про налаштування VLAN, а також про стан і налаштування транкових інтерфейсів.

Команда show vlan використовується для перегляду поточних налаштувань VLAN на комутаторі. Вона відображає інформацію про всі VLAN, їх номери, імена та порти, що належать кожному VLAN.

Команда show interfaces trunk використовується для перегляду стану транкових портів на комутаторі. Вона показує інформацію про всі транкові порти, включаючи дозволені VLAN, рідний VLAN і режими транків.

```
Kalianov_Switch_1#show vlan

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Gig0/2
15   Development                       active    Fa0/2, Fa0/3, Fa0/4, Fa0/5
                                                 Fa0/6, Fa0/7, Fa0/8, Fa0/9
                                                 Fa0/10, Fa0/11, Fa0/12, Fa0/13
                                                 Fa0/14, Fa0/15, Fa0/16, Fa0/17
                                                 Fa0/18, Fa0/19, Fa0/20, Fa0/21
                                                 Fa0/22, Fa0/23, Fa0/24
25   Testing                           active
99   Management                        active
100  Native                            active
1002 fddi-default                      active
1003 token-ring-default                active
1004 fddinet-default                   active
1005 trnet-default                     active

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------
1    enet  100001     1500  -      -      -        -    -        0      0
15   enet  100015     1500  -      -      -        -    -        0      0
25   enet  100025     1500  -      -      -        -    -        0      0
99   enet  100099     1500  -      -      -        -    -        0      0
100  enet  100100     1500  -      -      -        -    -        0      0
1002 fddi  101002     1500  -      -      -        -    -        0      0
1003 tr    101003     1500  -      -      -        -    -        0      0
1004 fdnet 101004     1500  -      -      -        ieee -        0      0
1005 trnet 101005     1500  -      -      -        ibm  -        0      0

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------

Remote SPAN VLANs
------------------------------------------------------------------------------

Primary Secondary Type              Ports
------- --------- ----------------- ------------------------------------------
Kalianov_Switch_1#show interfaces trunk
Port        Mode         Encapsulation  Status        Native vlan
Fa0/1       on           802.1q         trunking      100
Gig0/1      on           802.1q         trunking      100

Port        Vlans allowed on trunk
Fa0/1       15,25,99-100
Gig0/1      15,25,99-100

Port        Vlans allowed and active in management domain
Fa0/1       15,25,99,100
Gig0/1      15,25,99,100

Port        Vlans in spanning tree forwarding state and not pruned
Fa0/1       15,25,99,100
Gig0/1      15,25,99,100
```

Рисунок 3.6 – Перевірка налаштувань VLAN та транкових інтерфейсів за допомогою команд

3.5 Захист інформації в комп’ютерній системі від несанкціонованого доступу

Для забезпечення захисту інформації в комп'ютерній системі від несанкціонованого доступу були розроблені та реалізовані відповідні методи. Основним завданням було налаштування мереж VLAN та маршрутизації між ними, а також забезпечення безпеки мережевих пристроїв.

Для налаштування мереж VLAN і маршрутизації між ними було виконано наступне:

– створення VLAN: На основі схеми топології та з використанням таблиці 3.3 були створені необхідні мережі VLAN та присвоєні їм відповідні імена;

– налаштування транкових і портів доступу: Було налаштовано транкові порти для забезпечення комунікації між комутаторами, а також порти доступу для підключення кінцевих пристроїв. Усі невикористовувані фізичні порти комутаторів були вимкнені для підвищення рівня безпеки.

Ці кроки дозволили значно підвищити рівень захисту інформації в мережі, забезпечуючи правильну сегментацію трафіку та зменшуючи ризики несанкціонованого доступу. Далі будуть представлені більш детальні описи налаштувань.

3.5.1 Налаштування мереж VLAN

Для захисту інформації в комп'ютерній системі від несанкціонованого доступу було налаштовано мережі VLAN, відповідно до схеми топології та писку мереж VLAN представлених у таблиці 3.3.

Підмережа LAN1 була розділена на кілька VLAN для різних функціональних груп: Default (VLAN 1), Development (VLAN 15), Testing (VLAN 25), Management (VLAN 99) і Native (VLAN 100).

Таблиця 3.3 – Список мереж VLAN

Номер VLAN	Ім’я VLAN	Примітка
1	default	Не використовується
15	Development	Для розробки
25	Testing	Для тестування
99	Management	Для управління пристроями
100	Native	Власна мережа

Налаштування VLAN на прикладі Switch 1.

```
interface f0/1
 switchport mode trunk
 switchport trunk native vlan 100
 switchport trunk allowed vlan 15,25,99,100
 exit
interface gig0/1
 switchport mode trunk
 switchport trunk native vlan 100
 switchport trunk allowed vlan 15,25,99,100
 exit
interface vlan 99
 ip address 192.168.15.113 255.255.255.240
 exit
ip default-gateway 192.168.15.112
 exit
copy running-config startup-config
```

Ці налаштування дозволяють створити логічні сегменти в мережі, що підвищує рівень безпеки та керованість трафіком. Розподіл підмережі LAN1 на кілька VLAN допомагає ізолювати різні типи трафіку та забезпечує більш гнучке управління мережевими ресурсами.

3.5.2 Налаштування параметрів безпеки комутаторів та адресації ПК в мережах VLAN

Для забезпечення коректної роботи мереж VLAN та ефективного управління IP-адресами було налаштовано маршрутизатор як сервер DHCP. Це дозволяє автоматично призначати IP-адреси пристроям в різних VLAN, спрощуючи управління мережею та забезпечуючи гнучкість конфігурації.

Налаштування DHCP на маршрутизаторі:

– налаштування маршрутизатора як DHCP сервер: маршрутизатор було налаштовано для здійснення маршрутизації між VLAN та функціонування як DHCP сервер для кожної VLAN. Це забезпечує централізоване управління IP-адресами в мережі;

– створення DHCP пулів: були створені пули DHCP під назвою pollvlan№, де № – номер VLAN. Це дозволяє автоматично призначати IP-адреси пристроям у відповідних VLAN;

– виключення перших 10 адрес з пулу: для кожного пулу DHCP були виключені перші 10 адрес, щоб уникнути конфліктів з вручну призначеними IP-адресами, такими як адреси серверів та інших критичних пристроїв. Також для кожного пулу були вказані адреса DNS-сервера і шлюз за умовчанням.

Команди для налаштування DHCP.

```
ip dhcp excluded-address 192.168.15.65 192.168.15.74
ip dhcp excluded-address 192.168.15.97 192.168.15.106
ip dhcp excluded-address 192.168.15.79
ip dhcp excluded-address 192.168.15.80
ip dhcp excluded-address 192.168.15.111
ip dhcp pool pollvlan15
 network 192.168.15.64 255.255.255.224
 default-router 192.168.15.65
 dns-server 192.168.15.79
 exit
ip dhcp pool pollvlan25
 network 192.168.15.96 255.255.255.240
 default-router 192.168.15.97
 dns-server 192.168.15.79
 exit
exit
copy running-config startup-config
```

Пояснення налаштувань (виключення адрес).

```
ip dhcp excluded-address 192.168.15.65 192.168.15.74
ip dhcp excluded-address 192.168.15.97 192.168.15.106
ip dhcp excluded-address 192.168.15.79
ip dhcp excluded-address 192.168.15.80
ip dhcp excluded-address 192.168.15.111
```

Ці команди виключають з пулу перші 10 адрес для кожного VLAN, а також адреси серверів у мережі, щоб уникнути конфліктів і забезпечити наявність фіксованих адрес для критичних пристроїв.

```
ip dhcp pool pollvlan15
network 192.168.15.64 255.255.255.224
default-router 192.168.15.65
dns-server 192.168.15.79
exit
```

Ця команда створює пул DHCP для VLAN 15, встановлюючи діапазон адрес, шлюз за замовчуванням і DNS-сервер.

Команди для створення пулу DHCP для VLAN 25**.**

```
ip dhcp pool pollvlan25
network 192.168.15.96 255.255.255.240
default-router 192.168.15.97
dns-server 192.168.15.79
exit
```

Ця команда створює пул DHCP для VLAN 25 з відповідними налаштуваннями.

Ці налаштування дозволяють забезпечити автоматичний розподіл IP-адрес серед пристроїв у різних VLAN, зменшуючи ризики конфліктів адрес та забезпечуючи ефективне використання мережевих ресурсів.

4 РОЗРОБКА КОМПОНЕНТА СИСТЕМИ

4.1 Мета та завдання роботи

Метою даного розділу є створення настільного застосунку для управління проектами, задачами, термінами та ресурсами в середовищі розробки програмного забезпечення. Це дозволить підвищити ефективність та організацію командної роботи, що є ключовим фактором для успішного виконання проектів.

Для досягнення цієї мети необхідно виконати кілька важливих завдань. Насамперед, необхідно провести дослідження існуючих рішень в області систем управління проектами, щоб виявити їх ключові функції та обмеження. Це дозволить зрозуміти, які аспекти потребують покращення або доповнення. Важливо також проаналізувати вимоги до системи управління проектами, особливо для розробників програмного забезпечення, включаючи підтримку Agile методологій та управління ресурсами. Це допоможе визначити необхідні функціональні можливості для ефективної роботи команди.

Проектування архітектури застосунку повинно враховувати вимоги до модульності, розширюваності та користувацького інтерфейсу, що забезпечить зручність використання і легкість у подальшій розробці та підтримці. Наступним кроком є розробка настільного застосунку з використанням технологій WPF для Windows. Важливо також інтегрувати функції для підтримки Agile методологій, таких як дошки Kanban та підтримка Scrum-процесів, що допоможе ефективніше організувати роботу команди.

Після завершення розробки, необхідно провести ретельне тестування системи, включаючи функціональне тестування та тестування користувацького інтерфейсу. Це дозволить виявити можливі помилки та недоліки, а також оцінити зручність використання системи. Останнім етапом є аналіз отриманих результатів, що включає оцінку зручності використання та ефективності системи управління проектами, порівняння з існуючими рішеннями, а також виявлення переваг та недоліків розробленого застосунку.

Завдяки виконанню зазначених завдань очікується створення застосунку, що дозволить командам розробників ефективно управляти проектами та задачами. Це сприятиме покращенню організації робочого процесу та підвищенню продуктивності команди завдяки впровадженню Agile практик. Також очікується підвищення прозорості проектних процесів та спрощення звітності по проектам і ресурсам, що забезпечить більш ефективне управління та контроль за виконанням задач.

4.2 Огляд існуючих систем управління проектами

В рамках цього підрозділу буде проведено огляд існуючих систем управління проектами, таких як TargetProcess, JIRA та Trello. Мета огляду полягає у визначенні їхніх переваг та недоліків, що дозволить краще зрозуміти, які аспекти цих систем можна використати або уникнути при розробці власного застосунку.

TargetProcess:

Однією з головних переваг TargetProcess є його широкий функціонал, який включає підтримку різних методологій управління проектами, зокрема Agile. Система також пропонує інтеграцію з іншими інструментами, що дозволяє легко налаштувати робочий процес відповідно до потреб команди. Однак, варто зазначити, що складний інтерфейс та надмірна функціональність можуть стати перешкодою для користувачів, особливо для тих, хто не має достатнього досвіду роботи з такими системами. Це ускладнює використання TargetProcess і може призвести до зниження ефективності роботи команди.

JIRA:

JIRA є одним з найпопулярніших інструментів для управління проектами, і це не випадково. Вона пропонує високу гнучкість налаштувань, підтримку численних плагінів, що дозволяє розширити функціональність системи відповідно до потреб конкретного проекту. Крім того, JIRA має велику спільноту користувачів, що сприяє обміну досвідом та підтримці. Проте, варто врахувати, що висока вартість ліцензії та складність налаштування і використання можуть

стати значними бар'єрами для її впровадження, особливо для невеликих команд та стартапів.

Trello:

Trello вирізняється своєю простотою у використанні та візуальним представленням задач у вигляді карток на дошках, що робить його привабливим для багатьох користувачів. Цей інструмент дозволяє швидко організувати робочий процес і інтегрується з іншими сервісами, що додає зручності в роботі. Однак, Trello має обмежений функціонал для великих команд та відсутність деяких професійних інструментів для управління проектами, що може обмежити його застосування в складніших проектах.

На основі проведеного аналізу існуючих рішень можна зробити висновок, що більшість систем мають широкий функціонал, але часто є надто складними у використанні для середніх і малих команд розробників. Це призводить до того, що розробники змушені створювати додаткові інструменти, наприклад, текстові файли для відстеження задач, що знижує загальну ефективність роботи команди. Важливо врахувати ці аспекти при розробці власного застосунку, щоб забезпечити його простоту у використанні та водночас функціональність, необхідну для ефективного управління проектами.

4.3 Обґрунтування технічних характеристик застосунку

Застосунок, який ми розробляємо, повинен відповідати ряду ключових вимог, які забезпечать його ефективність та зручність у використанні для команд розробників програмного забезпечення. Ці вимоги включають підтримку Agile методологій, можливість управління ресурсами та задачами, інтуїтивно зрозумілий користувацький інтерфейс, а також підтримку реальних повідомлень про зміни в задачах.

Підтримка Agile методологій.

Застосунок повинен підтримувати методології Agile, зокрема Kanban та Scrum. Це забезпечить гнучкість управління проектами та дозволить командам адаптуватися до змін. Kanban дозволяє візуально відстежувати прогрес задач за

допомогою дошок, де кожна задача представлено у вигляді картки, яка переміщується між колонками, що відображають стадії виконання. Scrum, з іншого боку, підтримує організацію роботи в коротких ітераціях (спринтах) та включає регулярні зустрічі для обговорення прогресу і планування подальших дій.

Можливість управління ресурсами та задачами.

Ефективне управління ресурсами є критично важливим для успішного виконання проектів. Застосунок повинен дозволяти менеджерам проектів розподіляти задачі серед членів команди, відстежувати виконання цих задач та управляти доступними ресурсами. Це включає можливість призначення задач конкретним користувачам, встановлення пріоритетів, а також відстеження статусу виконання задач.

Інтуїтивно зрозумілий користувацький інтерфейс.

Однією з головних вимог до застосунку є створення інтуїтивно зрозумілого користувацького інтерфейсу, який буде простим у використанні для всіх членів команди, незалежно від їхнього рівня технічної підготовки. Інтерфейс повинен бути логічно структурованим і забезпечувати швидкий доступ до основних функцій системи. Використання сучасних принципів дизайну та UX/UI рішень сприятиме підвищенню зручності та ефективності роботи з застосунком.

Підтримка реальних повідомлень про зміни в задачах.

Для забезпечення оперативного інформування користувачів про зміни у задачах, застосунок повинен підтримувати функцію реальних повідомлень. Це дозволить командам бути в курсі актуального стану проектів та оперативно реагувати на будь-які зміни. Взаємодія між компонентами системи буде побудована за допомогою технології WebSocket, що забезпечить миттєве передавання повідомлень.

Обґрунтування вибору технологій.

Для розробки настільного застосунку було обрано технологію WPF (Windows Presentation Foundation) для клієнтської частини. WPF забезпечує багатий набір інструментів для створення сучасного та функціонального

користувацького інтерфейсу. Основними перевагами WPF є підтримка високоякісної графіки, гнучкість у налаштуванні вигляду інтерфейсу та можливість використання XAML для опису інтерфейсу користувача.

Серверна частина застосунку розроблена з використанням ASP.NET Core, що дозволяє створювати високопродуктивні та масштабовані веб-додатки. ASP.NET Core підтримує мультиплатформність та дозволяє розгортати застосунок на різних операційних системах, що є важливим для забезпечення гнучкості та масштабованості системи.

Для зберігання даних використовується Entity Framework Core, який забезпечує зручну роботу з базами даних на основі об'єктно-реляційного відображення (ORM). Це дозволяє швидко розробляти та підтримувати базу даних, а також забезпечує інтеграцію з різними системами управління базами даних (СУБД).

Технологія WebSocket була обрана для забезпечення реальних повідомлень між клієнтською та серверною частинами. WebSocket дозволяє встановлювати постійне з'єднання між клієнтом і сервером, що забезпечує миттєву передачу даних у двох напрямках. Це є критично важливим для забезпечення оперативного інформування користувачів про зміни у задачах та інші важливі події.

Архітектура застосунку.

Архітектура застосунку включає в себе модульність, що дозволяє легко додавати нові функції та розширюваність для підтримки майбутніх змін. Модульність досягається за рахунок чіткої структури проекту, де кожен компонент виконує окрему функцію та взаємодіє з іншими компонентами через визначені інтерфейси. Це забезпечує легкість у підтримці та оновленні системи.

Взаємодія між компонентами побудована за допомогою технології WebSocket, що забезпечує реальні повідомлення про зміни у задачах. Це дозволяє користувачам оперативно отримувати інформацію про зміни у проекті, що сприяє ефективнішій організації роботи команди та знижує ризики виникнення помилок через несвоєчасне інформування.

У результаті, застосунок, розроблений з урахуванням усіх вищезазначених вимог та обраних технологій, забезпечить ефективне управління задачами, підвищуючи продуктивність та організацію командної роботи.

Загальна функціональна структура.

Загальна структура застосунку складається з трьох основних частин: клієнтської частини, серверної частини та бази даних. Кожна з цих частин виконує свої специфічні функції, забезпечуючи інтегровану та ефективну роботу системи в цілому.

Клієнтська частина і користувацький інтерфейс.

Клієнтська частина включає користувацький інтерфейс та логіку роботи з задачами. Використання WPF для Windows дозволяє створити багатий і зручний інтерфейс, який легко налаштовується під потреби користувачів. Інтерфейс розроблений з урахуванням сучасних принципів UX/UI дизайну, що робить його зручним та легким у використанні. Користувачі можуть створювати нові задачі, редагувати існуючі, призначати їх іншим членам команди та змінювати статус задач. Візуально привабливий та функціональний інтерфейс підтримує різноманітні інтерактивні елементи, забезпечуючи інтуїтивно зрозумілу взаємодію користувачів із системою.

Серверна частина.

Серверна частина відповідає за обробку запитів від клієнтської частини, управління користувачами, задачами та повідомленнями. Використання ASP.NET Core забезпечує високу продуктивність та масштабованість системи. Серверна частина обробляє запити на створення, редагування та видалення задач, керує пріоритетами задач і надсилає повідомлення про зміни у реальному часі за допомогою технології WebSocket. Це дозволяє користувачам оперативно отримувати інформацію про зміни у задачах. Надійність та безпека сервера, а також підтримка масштабованості системи, забезпечуються за рахунок використання сучасних технологій та методів проектування.

База даних і взаємодія з базою даних.

База даних зберігає всю необхідну інформацію про користувачів та задачі. Використання Entity Framework Core дозволяє зручно працювати з базою даних, виконуючи запити на збереження, оновлення та видалення даних. Це забезпечує швидкий доступ до даних та їх цілісність. Взаємодія з базою даних відбувається через чітко визначені інтерфейси, що дозволяє легко інтегрувати базу даних з іншими компонентами системи. Надійне зберігання даних та їх цілісність забезпечуються за рахунок використання перевірених технологій і підходів.

Функціональна структура застосунку забезпечує ефективну роботу всіх компонентів, що сприяє підвищенню продуктивності команди розробників та покращенню організації робочого процесу. Використання сучасних технологій та принципів проектування дозволяє створити надійну та масштабовану систему, яка відповідає вимогам сучасного ринку програмного забезпечення.

4.5 Схема функціональної структури

На рис. 4.1 представлена схема функціональної структури застосунку. Вона складається з кількох ключових етапів, кожен з яких виконує важливу функцію у забезпеченні ефективного управління задачами та взаємодії користувачів із системою. Далі наведено детальний опис кожного етапу.

Логін користувача та перевірка автентичності.

Для входу в систему користувач вводить свої логін і пароль, які відправляються на сервер. Сервер, отримавши ці дані, передає їх контролеру користувачів, який перевіряє валідність введених даних. Якщо дані коректні, сервер повертає клієнту токен, що підтверджує успішний логін. Клієнтська частина, отримавши токен, обробляє його та відкриває вікно зі списком задач.

Отримання списку задач для конкретного користувача.

Після успішного логіну клієнтська частина відправляє запит на сервер для отримання списку задач. Цей запит включає токен і userID. Сервер перевіряє валідність токену і запитує список задач у бази даних. База даних зберігає та повертає необхідні дані серверу, який відправляє їх клієнту. Клієнтська частина отримує список задач і відображає їх у відповідному інтерфейсі.

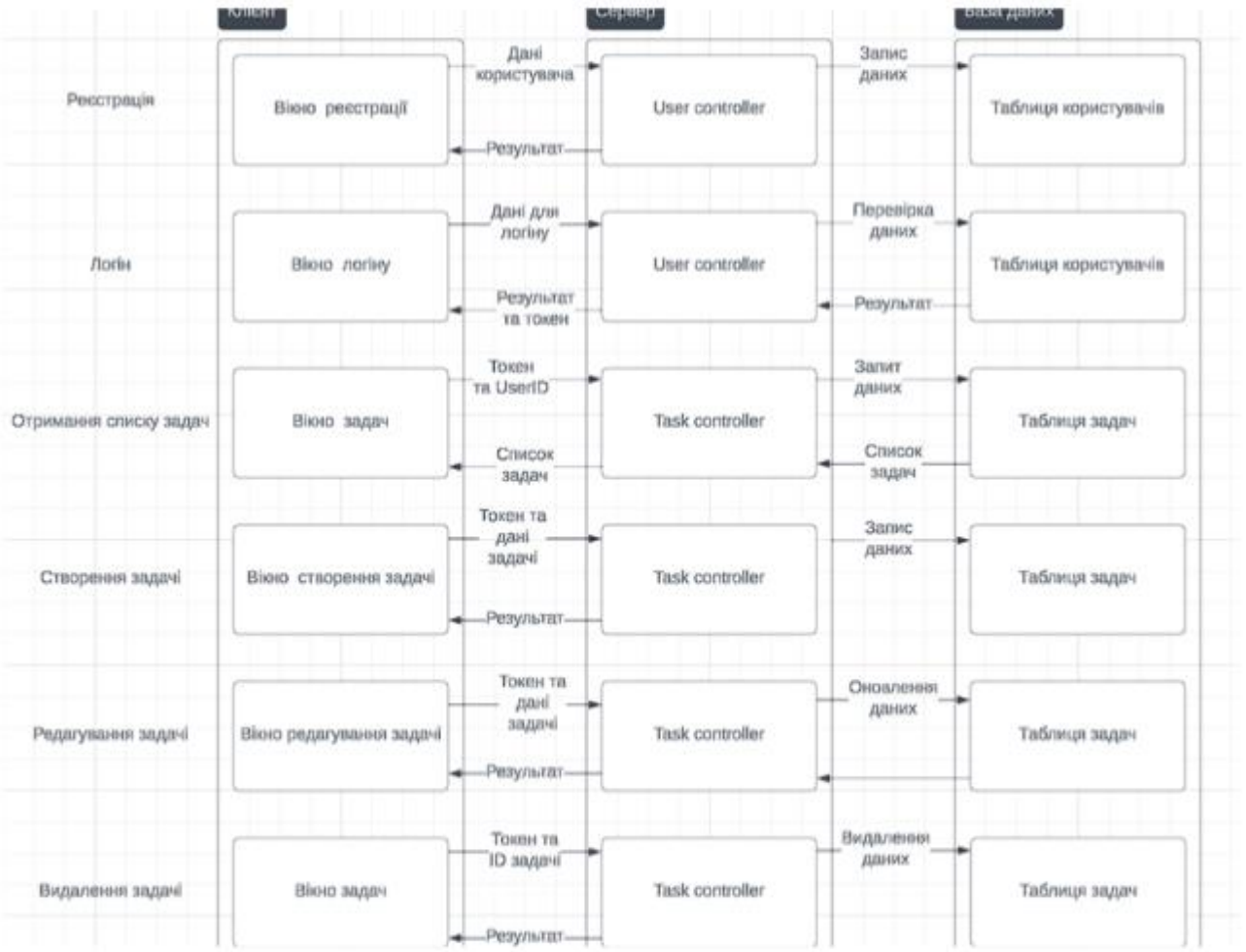

Рисунок 4.1 – Схема функціональної структури

Створення нової задачі користувачем.

Користувач вводить дані для нової задачі у відповідному полі на клієнтській частині. Ці дані відправляються на сервер, де їх обробляє контролер задач. Сервер зберігає нову задачу у базі даних і надсилає повідомлення про створення нової задачі. Після цього клієнтська частина отримує підтвердження та оновлює інтерфейс відповідно.

Редагування задачі.

Користувач може редагувати задачу, змінюючи його статус, пріоритет або інші дані на клієнтській частині. Ці зміни відправляються на сервер, де контролер задач оновлює відповідні дані у базі даних. Сервер також відправляє повідомлення про зміни, і клієнтська частина отримує підтвердження, відображаючи актуальну інформацію у користувацькому інтерфейсі.

Видалення задачі з системи.

Процес видалення задачі починається з того, що користувач ініціює видалення на клієнтській частині. Запит на видалення відправляється на сервер, де контролер задач видаляє відповідну задачу з бази даних. Сервер відправляє підтвердження клієнту, який обробляє відповідь і оновлює інтерфейс.

Ці етапи роботи компонента управління задачами та взаємодії з користувачами забезпечують ефективне та зручне використання системи, сприяючи підвищенню продуктивності команди та оптимізації робочих процесів.

4.5 Демонстрація роботи застосунку

Користувацький інтерфейс.

Інтерфейс користувача нашого застосунку забезпечує інтуїтивно зрозумілу взаємодію з системою, що дозволяє користувачам ефективно управляти своїми задачами. Далі наведено опис основного функціоналу користувацького інтерфейсу.

Логін користувача.

Екран логіну складається з текстового поля для введення імені користувача та поля для введення пароля, яке підтримує водяні знаки для покращення візуального сприйняття (рис. 4.2). Під цими полями знаходиться кнопка "Login", натискання на яку ініціює процес логіну. Після введення коректних даних і успішного входу, користувача перенаправляють на вікно зі списком задач. Внизу екрану розташовано посилання для переходу на екран реєстрації, що дозволяє новим користувачам швидко зареєструватися у системі.

Реєстрація користувача.

Екран реєстрації містить текстове поле для введення імені користувача та два поля для введення пароля (рис. 4.3). Після заповнення цих полів користувач натискає кнопку підтвердження реєстрації, розташовану нижче. Якщо реєстрація проходить успішно, користувач автоматично перенаправляється на екран логіну для входу в систему.

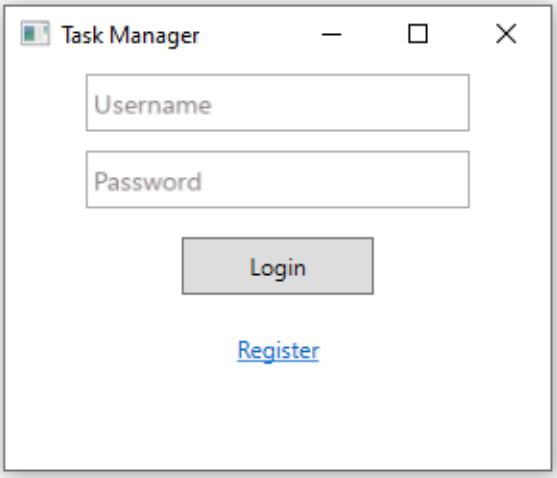

Рисунок 4.2 – Вікно логіну

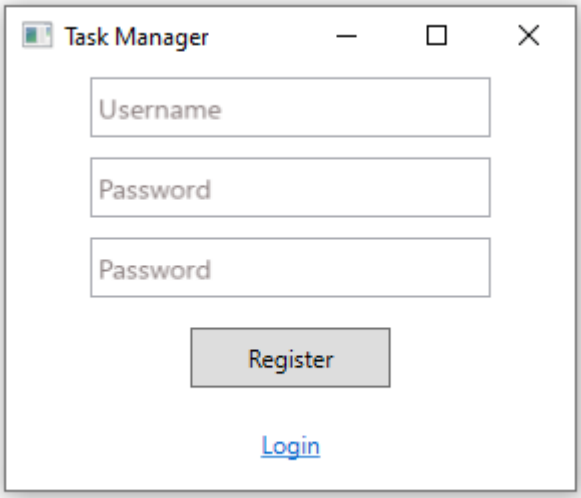

Рисунок 4.3 – Вікно реєстрації

Список задач.

Екран списку задач організовано таким чином, щоб забезпечити зручний доступ до задач користувача та його підлеглих. Ліворуч розташований вертикальний список користувачів, де першим відображається поточний користувач, а нижче - його підлеглі. За замовчуванням обраний поточний користувач, і для нього завантажуються та відображаються задачі в основній частині вікна.

Основна частина вікна розділена на дві частини: більшу частину займає список задач, представлених у вигляді карток, а внизу по всій ширині розташована кнопка для додавання нової задачі (рис. 4.4). Користувач може взаємодіяти із задачами різними способами. При натисканні правої кнопки миші на картку задачі відкривається контекстне меню з можливістю видалити задачу або змінити його статус (Delete, Change status). Подвійне натискання на картку відкриває вікно редагування задачі (рис. 4.5). Крім того, задачі можна перетягувати для зміни їх пріоритету, при цьому на сервер відправляється запит на редагування задачі з новим пріоритетом.

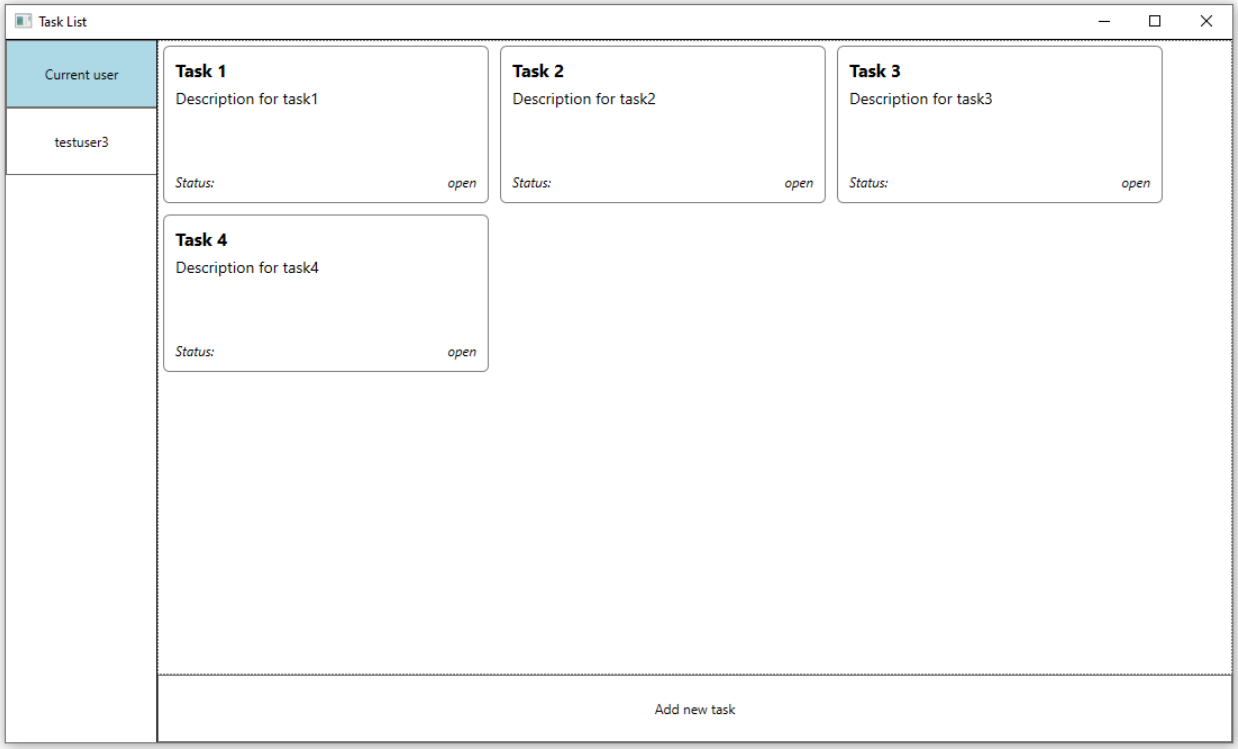

Рисунок 4.4 – Вікно списку задач

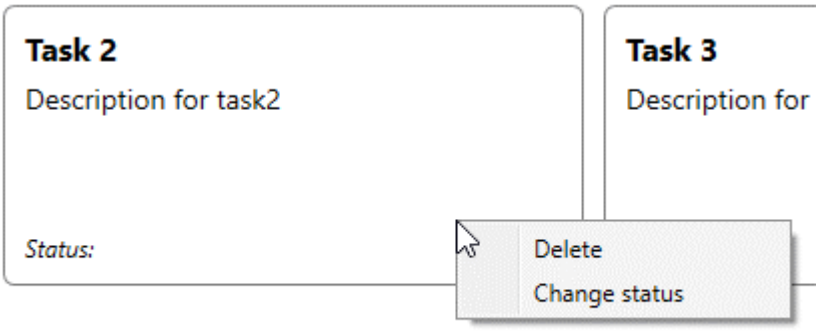

Рисунок 4.5 – Контекстне меню картки

Створення та редагування задачі.

Вікно створення та редагування задачі має аналогічний інтерфейс для забезпечення узгодженого користувацького досвіду (рис. 4.6). У верхній частині вікна розташоване текстове поле з водяним знаком для введення заголовка задачі. Справа від цього поля знаходиться кнопка "Submit", яка в режимі редагування залишається неактивною, доки не будуть внесені зміни. Натискання на цю кнопку відправляє дані на сервер для створення або оновлення задачі, залежно від режиму відкриття вікна.

Основну частину вікна займає текстове поле з водяним знаком для введення опису задач. Це забезпечує зручне та зрозуміле введення інформації, необхідної для повного опису задачі.

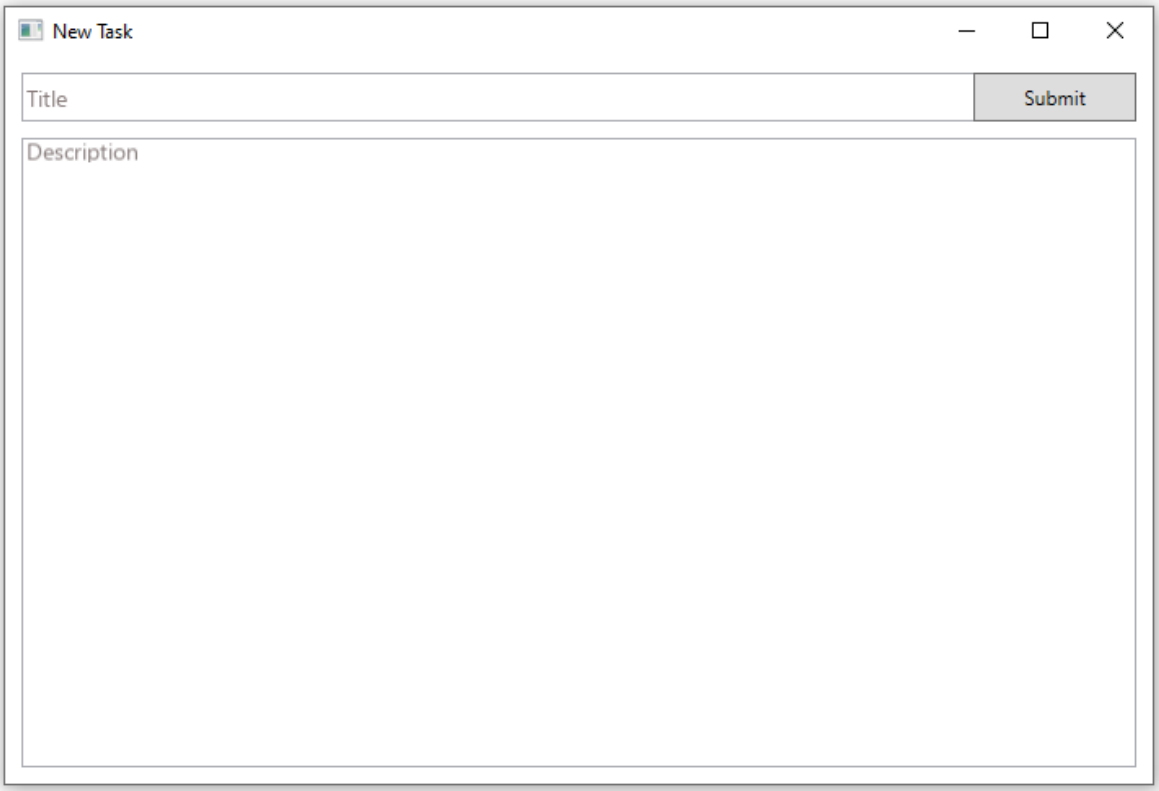

Рисунок 4.6 – Вікно створення та редагування задачі

Процес управління задачами.

Процес управління задачами включає кілька ключових етапів, які забезпечують їх ефективне виконання та контроль. Основні функції управління

охоплюють створення, редагування, видалення задач, зміну їх статусу та пріоритету, а також отримання повідомлень про зміни у задачах.

Коли в системі відбуваються будь-які зміни, пов'язані з задачіми, відповідні користувачі отримують повідомлення через вебсокет. Це включає створення нової задачі, зміну статусу або пріоритету задач, а також їх видалення. Такі повідомлення інформують користувачів про всі актуальні зміни, що відбуваються у їхньому списку задач, і служать індикатором для оновлення цього списку.

Отримані повідомлення дозволяють користувачам швидко дізнаватися про зміни у стані задач, що сприяє більш ефективному управлінню проектами. Усі повідомлення, отримані через вебсокет, є сигналом для оновлення списку задач у застосунку. Це забезпечує актуальну інформацію та підвищує ефективність роботи команди.

4.5 Тестування системи

Функціональне тестування.

Функціональне тестування є важливим етапом розробки застосунку, який забезпечує перевірку правильності роботи основних функцій системи. Під час тестування проводиться детальний аналіз роботи всіх компонентів застосунку для забезпечення їхньої коректної роботи. Нижче описані процеси тестування основних функцій системи.

Тестування створення задачі.

Процес тестування створення задачі включає перевірку коректності заповнення форми створення нової задачі, відправки даних на сервер та відображення нової задачі у списку задач користувача. На першому етапі перевіряється, чи може користувач без проблем ввести назву та опис задачі, а також чи правильно обробляються ці дані на клієнтській частині. Далі перевіряється відправка даних на сервер, де здійснюється створення нової задачі у базі даних. Після успішного створення задачі перевіряється, чи відображається нова задача у списку задач користувача у клієнтському інтерфейсі.

Тестування редагування задачі.

Тестування редагування задачі включає перевірку можливості коректного внесення змін до назви, опису, статусу та пріоритету задачі. Користувач заповнює форму редагування, яка повинна відображати поточні дані задачі, вносить необхідні зміни та відправляє їх на сервер. Сервер обробляє запит і оновлює відповідні записи у базі даних.

Тестування видалення задачі.

Процес тестування видалення задачі включає перевірку можливості видалення задачі користувачем, відправки запиту на сервер та оновлення списку задач у клієнтському інтерфейсі. Перевіряється, чи зникає задача зі списку після успішного видалення та чи оновлюються пріоритети інших задач у списку.

Тестування користувацького інтерфейсу.

Тестування інтерфейсу включає перевірку всіх інтерактивних елементів, таких як кнопки, текстові поля, меню та інші компоненти. Перевіряється, чи правильно відображаються та функціонують всі елементи інтерфейсу, включаючи повідомлення про помилки, підказки та індикатори стану.

Функціональне тестування дозволило переконатися у стабільності та надійності роботи системи. Цей етап є критично важливим для забезпечення високої якості кінцевого продукту.

ВИСНОВКИ

Проведено аналіз існуючих методів та інструментів управління проектами, що дозволило визначити основні вимоги та завдання для розробки власного рішення. Було сформульовано чіткі цілі та завдання проекту, що охоплюють усі ключові аспекти системи управління проектами.

Розроблено апаратну частину системи, що включає вибір обладнання, розробку структурної схеми та обґрунтування вибору компонентів. Було розроблено детальний план впровадження апаратної частини, що забезпечує надійну роботу системи та високу ефективність.

Створено модель корпоративної мережі, яка забезпечує стабільну та безпечну комунікацію між усіма компонентами системи. Модель мережі була реалізована та протестована в симуляційному середовищі Cisco Packet Tracer, що підтвердило її функціональність та відповідність вимогам проекту.

Було розроблено програмний компонент, призначений для підвищення ефективності командної розробки програмного забезпечення. Компонент забезпечує зручне управління завданнями, моніторинг прогресу та координацію роботи команди. Реалізовано інтуїтивно зрозумілий інтерфейс, що дозволяє легко створювати та призначати завдання, а також отримувати повідомлення про оновлення в режимі реального часу.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цвіркун Л.І. Комп’ютерні мережі. Методичні рекомендації до виконання лабораторних робіт студентами галузі знань 12 Інформаційні технології спеціальності 123 Комп’ютерна інженерія. Ч. 1 / Цвіркун, Л. І. Панферова, Я. В. М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро: НТУ «ДП», 2018. - 59 с.

2. Дипломування. Методичні вказівки для бакалаврів галузі знань 12, Інформаційні технології спеціальності 123 Комп’ютерна інженерія / Л.І. Цвіркун, С.М. Ткаченко, Я.В. Панферова ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро: НТУ «ДП», 2024. – 56 с.

3. Computer networking: a top-down approach / James F. Kurose, University of Massachusetts, Amherst, Keith W. Ross, NYU and NYU Shanghai, 2017. – 856 с.

4. WPF4.5Unleashed 1st Edition / Adam Nathan; Sams Publishing, 2013.-799 с.

5. The Art of Project Management / Scott Berkun; O'Reilly, 2005. - 392 с.

6. Cisco Systems. Cisco Enterprise Networks: Product Listing. - Cisco Systems, 2024. – Режим доступу: https://www.cisco.com/c/en/us/solutions/enterprise-networks/product-listing.html

7. Stack Overflow. Stack Overflow: Q&A for Professional and Enthusiast Programmers. - Stack Exchange Inc., 2024. – Режим доступу : https://stackoverflow.com/

8. Lucidchart. Lucidchart: Online Diagram Software & Visual Solution. - Lucid Software Inc., 2024. – Режим доступу :https://www.lucidchart.com/

Додаток А

Текст програми застосунку для підвищення ефективності командних розробок ПЗ

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ
ЗАСТОСУНОК ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ КОМАНДНИХ РОЗРОБОК ПЗ

Текст програми

804.02070743.24005-01 12 01

Листів 15

2024

АНОТАЦІЯ

Дана програма містить в собі програмний код настільного застосунку Task Manager для управління проектами та завданнями в команді розробників програмного забезпечення. Програма призначена для забезпечення ефективного моніторингу та контролю за виконанням завдань, управління пріоритетами задач, а також інформування користувачів про зміни в реальному часі. Програма написана мовою C# з використанням технологій WPF для клієнтської частини та ASP.NET Core для серверної частини, відлагоджена із застосуванням середовища Visual Studio.

ЗМІСТ

Сервер. Код класу Program.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Diagnostics;
using System.Net.WebSockets;
using System.Security.Claims;
using System.Text;
using TaskManagerApi;
using TaskManagerApi.Data;

var builder = WebApplication.CreateBuilder(args);

// Додаємо сервіси в контейнер
builder.Services.AddSingleton<WebSocketConnectionManager>();
builder.Services.AddControllers();
builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

// Налаштовуємо ключ для JWT
var key = Encoding.UTF8.GetBytes("task_manager_key");

// Налаштовуємо аутентифікацію
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.RequireHttpsMetadata = false;
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidIssuer = "TaskManagerService",
        ValidAudience = "TaskManagerApp",
    };
    options.Events = new JwtBearerEvents
    {
        OnMessageReceived = context =>
        {
            var accessToken = context.Request.Headers["X-Authorization"];
            if (!string.IsNullOrEmpty(accessToken))
            {
                context.Token = accessToken;
            }
            return Task.CompletedTask;
        },
        OnAuthenticationFailed = context =>
        {
            context.Response.Headers.Add("Token-Validation-Failed",
context.Exception.Message);
            return Task.CompletedTask;
        },
        OnTokenValidated = context =>
        {
            Debug.WriteLine("Token Validated Successfully");
```

```
            return Task.CompletedTask;
        }
    };
});

// Налаштовуємо авторизацію
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("UserPolicy", policy =>
    {
        policy.RequireClaim(ClaimTypes.NameIdentifier);
    });
});

var app = builder.Build();

// Налаштовуємо HTTP pipeline
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}

app.UseHttpsRedirection();
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.UseWebSockets();

app.Use(async (context, next) =>
{
    if (context.Request.Path == "/ws")
    {
        if (context.WebSockets.IsWebSocketRequest)
        {
            var webSocket = await context.WebSockets.AcceptWebSocketAsync();
            var userIdClaim = context.User.Claims.FirstOrDefault(c => c.Type ==
ClaimTypes.NameIdentifier);
            if (userIdClaim != null)
            {
                var userId = int.Parse(userIdClaim.Value);
                var webSocketManager =
context.RequestServices.GetService<WebSocketConnectionManager>();
                webSocketManager.AddSocket(userId.ToString(), webSocket);

                await WebSocketHandler.HandleWebSocketAsync(webSocket);
            }
            else
            {
                await webSocket.CloseAsync(WebSocketCloseStatus.PolicyViolation,
"Unauthorized", CancellationToken.None);
            }
        }
        else
        {
            context.Response.StatusCode = 400;
        }
    }
    else
    {
        await next();
    }
});

app.Run();
```

Код класу ProjectTaskController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using TaskManagerApi.Data;
using TaskManagerApi.Models;

[Authorize]
[Route("api/[controller]")]
[ApiController]
public class ProjectTasksController : ControllerBase
{
    private readonly ApplicationDbContext _context;
    private readonly WebSocketConnectionManager _webSocketConnectionManager;

    // Конструктор класу, ініціалізує контекст бази даних та менеджер WebSocket
з'єднань
    public ProjectTasksController(ApplicationDbContext context,
WebSocketConnectionManager webSocketConnectionManager)
    {
        _context = context;
        _webSocketConnectionManager = webSocketConnectionManager;
    }

    // Метод для коригування пріоритетів задач
    private async Task AdjustTaskPriorities(int assigneeId, int newPriority, int?
excludeTaskId = null)
    {
        // Отримуємо список задач, які потребують оновлення пріоритету
        var tasksToUpdate = _context.ProjectTasks
                                    .Where(t => t.AssigneeId == assigneeId &&
t.Priority >= newPriority)
                                    .OrderBy(t => t.Priority)
                                    .ToList();

        // Оновлюємо пріоритет кожної задачі
        foreach (var t in tasksToUpdate)
        {
            if (excludeTaskId.HasValue && t.Id == excludeTaskId.Value)
                continue;

            t.Priority++;
        }

        // Зберігаємо зміни в базі даних
        await _context.SaveChangesAsync();
    }

    // Метод для зменшення пріоритетів задач після видалення задачі
    private async Task DecreaseTaskPriorities(int assigneeId, int removedPriority)
    {
        // Отримуємо список задач, які потребують оновлення пріоритету
        var tasksToUpdate = _context.ProjectTasks
                                    .Where(t => t.AssigneeId == assigneeId &&
t.Priority > removedPriority)
                                    .OrderBy(t => t.Priority)
                                    .ToList();

        // Зменшуємо пріоритет кожної задачі
        foreach (var t in tasksToUpdate)
        {
            t.Priority--;
```

```
        }

        // Зберігаємо зміни в базі даних
        await _context.SaveChangesAsync();
    }

    // Метод для отримання всіх задач поточного користувача
    [HttpGet]
    public IActionResult GetTasks()
    {
        // Отримуємо ідентифікатор поточного користувача
        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
        var user = _context.Users.SingleOrDefault(u => u.Id == userId);

        // Отримуємо задачі, які призначені користувачеві або його підлеглим
        var tasks = _context.ProjectTasks
                            .Where(t => t.AssigneeId == userId || _context.Users.Any(u
=> u.ManagerId == userId && u.Id == t.AssigneeId))
                            .OrderBy(t => t.Priority)
                            .ToList();

        // Повертаємо список задач
        return Ok(tasks);
    }

    // Метод для отримання задач конкретного користувача
    [HttpGet("user/{reqUserId}")]
    public IActionResult GetTasks(int reqUserId)
    {
        // Отримуємо ідентифікатор поточного користувача
        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
        if (reqUserId != userId)
        {
            // Перевіряємо, чи є запитуваний користувач підлеглим поточного
            var isSubordinate = _context.Users.Any(u => u.Id == reqUserId &&
u.ManagerId == userId);
            if (!isSubordinate)
            {
                return NotFound();
            }
        }

        // Отримуємо задачі запитуваного користувача
        var tasks = _context.ProjectTasks
            .Where(t => t.AssigneeId == reqUserId)
            .OrderBy(t => t.Priority)
            .ToList();

        if (tasks == null || !tasks.Any())
        {
            return NotFound();
        }

        // Повертаємо список задач
        return Ok(tasks);
    }

    // Метод для отримання задачі за її ідентифікатором
    [HttpGet("{id}")]
    public IActionResult GetTask(int id)
    {
        // Отримуємо ідентифікатор поточного користувача
        var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier).Value);
        var user = _context.Users.SingleOrDefault(u => u.Id == userId);
```

```
        // Отримуємо задачу, якщо вона призначена користувачеві або його підлеглим
        var task = _context.ProjectTasks.SingleOrDefault(t => t.Id == id &&
                                                              (t.AssigneeId == userId
|| _context.Users.Any(u => u.ManagerId == userId && u.Id == t.AssigneeId)));
        if (task == null)
        {
            return NotFound();
        }

        // Повертаємо задачу
        return Ok(task);
    }

    // Метод для створення нової задачі
    [HttpPost]
    public async Task<IActionResult> CreateTask([FromBody] ProjectTask task)
    {
        // Коригуємо пріоритети інших задач
        await AdjustTaskPriorities(task.AssigneeId, task.Priority);

        // Додаємо нову задачу в базу даних
        _context.ProjectTasks.Add(task);
        await _context.SaveChangesAsync();

        // Отримуємо ідентифікатор виконавця та його менеджера
        var assigneeId = task.AssigneeId.ToString();
        var assignee = _context.Users.SingleOrDefault(u => u.Id == task.AssigneeId);
        var managerId = assignee?.ManagerId?.ToString();

        // Відправляємо повідомлення виконавцеві та його менеджеру
        await _webSocketConnectionManager.SendMessageToUser(assigneeId, $"Task
'{task.Title}' was created for you.");
        if (managerId != null)
        {
            await _webSocketConnectionManager.SendMessageToUser(managerId, $"Task
'{task.Title}' was created for your subordinate.");
        }

        // Повертаємо створену задачу
        return CreatedAtAction(nameof(GetTask), new { id = task.Id }, task);
    }

    // Метод для оновлення задачі
    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateTask(int id, [FromBody] ProjectTask
updatedTask)
    {
        // Знаходимо існуючу задачу за ідентифікатором
        var task = _context.ProjectTasks.SingleOrDefault(t => t.Id == id);
        if (task == null)
        {
            return NotFound();
        }

        // Коригуємо пріоритети інших задач, якщо пріоритет змінився
        if (task.Priority != updatedTask.Priority)
        {
            await AdjustTaskPriorities(updatedTask.AssigneeId, updatedTask.Priority,
id);
        }

        // Оновлюємо поля задачі
        task.Title = updatedTask.Title;
        task.Description = updatedTask.Description;
        task.Status = updatedTask.Status;
```

```
        task.Priority = updatedTask.Priority;
        task.AssigneeId = updatedTask.AssigneeId;

        // Оновлюємо задачу в базі даних
        _context.ProjectTasks.Update(task);
        await _context.SaveChangesAsync();

        // Отримуємо ідентифікатор виконавця та його менеджера
        var assigneeId = task.AssigneeId.ToString();
        var assignee = _context.Users.SingleOrDefault(u => u.Id == task.AssigneeId);
        var managerId = assignee?.ManagerId?.ToString();

        // Відправляємо повідомлення виконавцеві та його менеджеру
        await _webSocketConnectionManager.SendMessageToUser(assigneeId, $"Task
'{task.Title}' was updated.");
        if (managerId != null)
        {
            await _webSocketConnectionManager.SendMessageToUser(managerId, $"Task
'{task.Title}' assigned to your subordinate was updated.");
        }

        // Повертаємо результат
        return NoContent();
    }

    // Метод для видалення задачі
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteTask(int id)
    {
        // Знаходимо задачу за ідентифікатором
        var task = _context.ProjectTasks.SingleOrDefault(t => t.Id == id);
        if (task == null)
        {
            return NotFound();
        }

        // Зберігаємо пріоритет видаленої задачі
        var removedPriority = task.Priority;

        // Видаляємо задачу з бази даних
        _context.ProjectTasks.Remove(task);
        await _context.SaveChangesAsync();

        // Коригуємо пріоритети інших задач
        await DecreaseTaskPriorities(task.AssigneeId, removedPriority);

        // Отримуємо ідентифікатор виконавця та його менеджера
        var assigneeId = task.AssigneeId.ToString();
        var assignee = _context.Users.SingleOrDefault(u => u.Id == task.AssigneeId);
        var managerId = assignee?.ManagerId?.ToString();

        // Відправляємо повідомлення виконавцеві та його менеджеру
        await _webSocketConnectionManager.SendMessageToUser(assigneeId, $"Task
'{task.Title}' was deleted.");
        if (managerId != null)
        {
            await _webSocketConnectionManager.SendMessageToUser(managerId, $"Task
'{task.Title}' assigned to your subordinate was deleted.");
        }

        // Повертаємо результат
        return NoContent();
    }
}
```

Код класу UserController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using TaskManagerApi.Data;
using TaskManagerApi.Models;

[Route("api/[controller]")]
[ApiController]
public class UsersController : ControllerBase
{
    private readonly ApplicationDbContext _context;

    // Конструктор класу, ініціалізує контекст бази даних
    public UsersController(ApplicationDbContext context)
    {
        _context = context;
    }

    // Метод для реєстрації нового користувача
    [HttpPost("register")]
    public IActionResult Register([FromBody] User user)
    {
        // Хешуємо пароль користувача
        user.PasswordHash = HashPassword(user.PasswordHash);

        // Додаємо користувача в базу даних
        _context.Users.Add(user);
        _context.SaveChanges();

        // Повертаємо успішний результат
        return Ok(new { message = "User registered successfully" });
    }

    // Метод для входу користувача
    [HttpPost("login")]
    public IActionResult Login([FromBody] LoginModel loginModel)
    {
        // Перевіряємо, чи існує користувач з вказаним ім'ям
        var existingUser = _context.Users.SingleOrDefault(u => u.Username ==
loginModel.Username);

        // Перевіряємо правильність паролю
        if (existingUser != null && VerifyPassword(loginModel.PasswordHash,
existingUser.PasswordHash))
        {
            // Генеруємо JWT токен
            var token = GenerateJwtToken(existingUser);
            return Ok(new { token });
        }

        // Повертаємо результат неавторизованого доступу
        return Unauthorized(new { message = "Invalid username or password" });
    }

    // Метод для отримання інформації про поточного користувача
    [HttpGet("me")]
```

```
    public async Task<IActionResult> GetCurrentUser()
    {
        // Отримуємо ідентифікатор поточного користувача з клеймів
        var userIdClaim = User.Claims.FirstOrDefault(c => c.Type ==
ClaimTypes.NameIdentifier);
        if (userIdClaim == null)
        {
            Console.WriteLine("Unauthorized: No userId claim found");
            return Unauthorized();
        }

        var userId = int.Parse(userIdClaim.Value);

        // Отримуємо користувача з бази даних
        var user = await _context.Users
            .AsNoTracking()
            .Select(u => new { u.Id, u.Username, u.Role, u.ManagerId })
            .SingleOrDefaultAsync(u => u.Id == userId);

        if (user == null)
        {
            Console.WriteLine($"User not found for ID: {userId}");
            return NotFound();
        }

        Console.WriteLine($"User found: {user.Username}");
        return Ok(user);
    }

    // Метод для отримання списку підлеглих користувача
    [HttpGet("subordinates/{managerId}")]
    public async Task<ActionResult<IEnumerable<User>>> GetSubordinates(int managerId)
    {
        // Отримуємо список підлеглих з бази даних
        var subordinates = await _context.Users.Where(u => u.ManagerId ==
managerId).ToListAsync();

        if (subordinates == null || subordinates.Count == 0)
        {
            return NotFound("No subordinates found");
        }

        return Ok(subordinates);
    }

    // Метод для хешування паролю
    private string HashPassword(string password)
    {
        using (var sha256 = SHA256.Create())
        {
            var hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
            return BitConverter.ToString(hash).Replace("-", "").ToLower();
        }
    }

    // Метод для перевірки паролю
    private bool VerifyPassword(string password, string storedHash)
    {
        using (var sha256 = SHA256.Create())
        {
            var hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
            var hashString = BitConverter.ToString(hash).Replace("-", "").ToLower();
            return hashString == storedHash;
        }
    }
```

```
    // Метод для генерації JWT токену
    private string GenerateJwtToken(User user)
    {
        var tokenHandler = new
System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();
        var key = Encoding.UTF8.GetBytes("task_manager_key");
        var tokenDescriptor = new SecurityTokenDescriptor
        {
            Subject = new ClaimsIdentity(new[]
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
                new Claim(ClaimTypes.Name, user.Username)
            }),
            Expires = DateTime.UtcNow.AddDays(7),
            SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature),
            Issuer = "TaskManagerService",
            Audience = "TaskManagerApp"
        };
        var token = tokenHandler.CreateToken(tokenDescriptor);
        return tokenHandler.WriteToken(token);
    }
}
```

Клієнт. Код класу ApiClient.cs

```
using System;
using System.Data;
using System.Diagnostics;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using System.Threading.Tasks;
using Newtonsoft.Json;
using TaskManagerApp.Models;

namespace TaskManagerApp.Services
{
    public class ApiClient
    {
        private readonly HttpClient _httpClient;
        public string Token { get; set; }

        // Конструктор класу, який ініціалізує HttpClient з базовою адресою API
        public ApiClient()
        {
            _httpClient = new HttpClient();
            _httpClient.BaseAddress = new Uri("http://localhost:5052/");
        }

        // Метод для входу користувача та отримання JWT токену
        public async Task<string> LoginAsync(string username, string password)
        {
            var loginModel = new LoginModel
            {
                Username = username,
                PasswordHash = password
            };

            var json = JsonConvert.SerializeObject(loginModel);
            var content = new StringContent(json, Encoding.UTF8, "application/json");

            // Відправляємо POST запит на API для входу користувача
```

```
            var response = await _httpClient.PostAsync("api/users/login", content);

            if (!response.IsSuccessStatusCode)
                return string.Empty;

            var responseString = await response.Content.ReadAsStringAsync();
            var tokenResponse =
JsonConvert.DeserializeObject<TokenResponse>(responseString);

            // Зберігаємо отриманий токен
            Token = tokenResponse.Token;
            SetAuthorizationHeader();

            Log($"Token received: {Token}");

            return Token;
        }

        // Метод для реєстрації нового користувача
        public async Task<bool> RegisterAsync(string username, string password, string
role)
        {
            var registerModel = new { Username = username, PasswordHash = password,
Role = role };
            var content = new
StringContent(JsonConvert.SerializeObject(registerModel), Encoding.UTF8,
"application/json");

            // Відправляємо POST запит на API для реєстрації користувача
            var response = await _httpClient.PostAsync("api/users/register", content);
            return response.IsSuccessStatusCode;
        }

        // Метод для отримання задач користувача за його ідентифікатором
        public async Task<ProjectTask[]> GetTasksAsync(int userId)
        {
            Log($"Getting tasks for user: {userId}");

            // Відправляємо GET запит на API для отримання задач користувача
            var response = await
_httpClient.GetAsync($"api/projecttasks/user/{userId}");
            if (!response.IsSuccessStatusCode)
            {
                Log($"Failed to get tasks: {response.ReasonPhrase}");
            }
            response.EnsureSuccessStatusCode();

            var responseString = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<ProjectTask[]>(responseString);
        }

        // Метод для отримання інформації про поточного користувача
        public async Task<User> GetCurrentUserAsync()
        {
            Log("Getting current user");

            // Відправляємо GET запит на API для отримання поточного користувача
            var response = await _httpClient.GetAsync("api/users/me");
            if (!response.IsSuccessStatusCode)
            {
                Log($"Failed to get current user: {response.ReasonPhrase}");
            }
            response.EnsureSuccessStatusCode();

            var responseString = await response.Content.ReadAsStringAsync();
```

```
            return JsonConvert.DeserializeObject<User>(responseString);
        }

        // Метод для отримання списку підлеглих користувача
        public async Task<User[]> GetSubordinatesAsync(int managerId)
        {
            Log($"Getting subordinates for manager: {managerId}");

            // Відправляємо GET запит на API для отримання підлеглих користувача
            var response = await
_httpClient.GetAsync($"api/users/subordinates/{managerId}");
            if (response.StatusCode == System.Net.HttpStatusCode.NotFound)
                return new User[0];
            if (!response.IsSuccessStatusCode)
            {
                Log($"Failed to get subordinates: {response.ReasonPhrase}");
            }
            response.EnsureSuccessStatusCode();

            var responseString = await response.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<User[]>(responseString);
        }

        // Метод для створення нової задачі для користувача
        internal async Task<bool> CreateNewTaskAsync(int id, string taskTitle, string
taskDescription)
        {
            Log($"Creating new task for user: {id}");

            var taskModel = new TaskModel
            {
                Title = taskTitle,
                Description = taskDescription,
                Status = "open",
                Priority = 1,
                AssigneeId = id
            };

            var content = new StringContent(JsonConvert.SerializeObject(taskModel),
Encoding.UTF8, "application/json");

            // Відправляємо POST запит на API для створення нової задачі
            var response = await _httpClient.PostAsync("api/projecttasks", content);
            return response.IsSuccessStatusCode;
        }

        // Метод для оновлення існуючої задачі
        public async Task UpdateTaskAsync(int taskId, TaskModel taskModel)
        {
            Log($"Updating task: {taskId}");

            var content = new StringContent(JsonConvert.SerializeObject(taskModel),
Encoding.UTF8, "application/json");

            // Відправляємо PUT запит на API для оновлення задачі
            var response = await _httpClient.PutAsync($"api/projecttasks/{taskId}",
content);
            response.EnsureSuccessStatusCode();
        }

        // Метод для видалення задачі за її ідентифікатором
        public async Task<bool> DeleteTaskAsync(int taskId)
        {
            // Відправляємо DELETE запит на API для видалення задачі
```

```
            var response = await
_httpClient.DeleteAsync($"api/projecttasks/{taskId}");
            return response.IsSuccessStatusCode;
        }

        // Метод для встановлення заголовку авторизації з JWT токеном
        private void SetAuthorizationHeader()
        {
            if (!string.IsNullOrEmpty(Token))
            {
                _httpClient.DefaultRequestHeaders.Remove("X-Authorization");
                _httpClient.DefaultRequestHeaders.Add("X-Authorization", Token);
            }
        }

        // Метод для логування повідомлень
        private void Log(string message)
        {
            Debug.WriteLine(message);
        }
    }
}
```

Код класу TaskListViewModel.cs

```
using System.Collections.ObjectModel;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using TaskManagerApp.Models;
using TaskManagerApp.Services;
using TaskManagerApp.Views;

namespace TaskManagerApp.ViewModels
{
    public class TaskListViewModel : BaseViewModel
    {
        private readonly ApiClient _apiClient;
        private readonly User _currentUser;
        private readonly NotificationService _notificationService;
        private ObservableCollection<TaskModel> _tasks = new
ObservableCollection<TaskModel>();
        private ObservableCollection<User> _subordinates = new
ObservableCollection<User>();
        private User _selectedUser;

        // Конструктор класу, ініціалізує ApiClient, поточного користувача та службу
сповіщень
        public TaskListViewModel(User currentUser, ApiClient apiClient, string
serverUrl)
        {
            _apiClient = apiClient;
            _currentUser = currentUser;
            _notificationService = new NotificationService(serverUrl);

            // Підписуємося на події отримання сповіщень
            _notificationService.OnNotificationReceived += async (message) =>
            {
                await LoadTasksAsync(_selectedUser.Id);
            };

            Tasks = new ObservableCollection<TaskModel>();
```

```
            Subordinates = new ObservableCollection<User>();

            // Ініціалізуємо команди
            LoadCurrentUserTasksCommand = new RelayCommand(async () =>
            {
                _selectedUser = _currentUser;
                await LoadTasksAsync(_currentUser.Id);
            });

            LoadUserTasksCommand = new RelayCommand<User>(async (user) =>
            {
                _selectedUser = user;
                await LoadTasksAsync(user.Id);
            });

            LoadSubordinatesCommand = new RelayCommand(async () => await
LoadSubordinatesAsync());
            OpenNewTaskWindowCommand = new RelayCommand(async () => await
OpenNewTaskWindow());

            // Завантажуємо задачі поточного користувача та підлеглих
            LoadCurrentUserTasksCommand.Execute(null);
            LoadSubordinatesCommand.Execute(null);

            StartNotificationService();
        }

        // Властивість для задач
        public ObservableCollection<TaskModel> Tasks
        {
            get => _tasks;
            set => SetProperty(ref _tasks, value);
        }

        // Властивість для підлеглих
        public ObservableCollection<User> Subordinates
        {
            get => _subordinates;
            set => SetProperty(ref _subordinates, value);
        }

        // Команди для завантаження задач та підлеглих, відкриття вікна нової задачі
        public ICommand LoadCurrentUserTasksCommand { get; }
        public ICommand LoadUserTasksCommand { get; }
        public ICommand LoadSubordinatesCommand { get; }
        public ICommand OpenNewTaskWindowCommand { get; }

        // Метод для завантаження задач користувача
        private async Task LoadTasksAsync(int userId)
        {
            var tasks = await _apiClient.GetTasksAsync(userId);
            Tasks.Clear();
            foreach (var task in tasks)
            {
                Tasks.Add(new TaskModel
                {
                    Id = task.Id,
                    Title = task.Title,
                    Description = task.Description,
                    Status = task.Status,
                    Priority = task.Priority,
                    AssigneeId = task.AssigneeId
                });
            }
            OnPropertyChanged(nameof(Tasks));
```

```
        }

        // Метод для завантаження підлеглих
        private async Task LoadSubordinatesAsync()
        {
            var subordinates = await _apiClient.GetSubordinatesAsync(_currentUser.Id);
            Subordinates.Clear();
            foreach (var subordinate in subordinates)
            {
                Subordinates.Add(subordinate);
            }
        }

        // Метод для відкриття вікна створення нової задачі
        private async Task OpenNewTaskWindow()
        {
            var newTaskWindow = new NewTaskWindow(_apiClient);
            newTaskWindow.ShowDialog();

            if (newTaskWindow.DialogResult != true)
                return;

            string taskTitle = newTaskWindow.TaskTitle;
            string taskDescription = newTaskWindow.TaskDescription;

            var isCreated = await _apiClient.CreateNewTaskAsync(_selectedUser.Id,
taskTitle, taskDescription);
            if (!isCreated)
                MessageBox.Show("Task creation failed");
            else
                await LoadTasksAsync(_selectedUser.Id);
        }

        // Метод для переміщення задачі
        public void MoveTask(TaskModel source, TaskModel target)
        {
            int oldIndex = Tasks.IndexOf(source);
            int newIndex = Tasks.IndexOf(target);

            if (oldIndex < newIndex)
            {
                newIndex--;
            }

            Tasks.Move(oldIndex, newIndex);
            UpdateTaskPriorities();
        }

        // Метод для оновлення пріоритетів задач
        private async void UpdateTaskPriorities()
        {
            for (int i = 0; i < Tasks.Count; i++)
            {
                Tasks[i].Priority = i + 1;
                await _apiClient.UpdateTaskAsync(Tasks[i].Id, Tasks[i]);
            }
        }

        // Метод для відкриття вікна редагування задачі
        public async Task OpenEditTaskWindow(TaskModel task)
        {
            var editTaskWindow = new NewTaskWindow(_apiClient, task);
            editTaskWindow.ShowDialog();

            if (editTaskWindow.DialogResult != true)
```

```
                return;

            task.Title = editTaskWindow.TaskTitle;
            task.Description = editTaskWindow.TaskDescription;

            await _apiClient.UpdateTaskAsync(task.Id, task);
            await LoadTasksAsync(_selectedUser.Id);
        }

        // Метод для видалення задачі
        public async Task DeleteTaskAsync(TaskModel task)
        {
            var isDeleted = await _apiClient.DeleteTaskAsync(task.Id);
            if (isDeleted)
            {
                await LoadTasksAsync(_selectedUser.Id);
            }
            else
            {
                MessageBox.Show("Task deletion failed");
            }
        }

        // Метод для зміни статусу задачі
        public async Task ChangeTaskStatusAsync(TaskModel task)
        {
            task.Status = task.Status == "open" ? "closed" : "open";
            await _apiClient.UpdateTaskAsync(task.Id, task);
            await LoadTasksAsync(_selectedUser.Id);
        }

        // Метод для запуску служби сповіщень
        private async void StartNotificationService()
        {
            await _notificationService.StartAsync(_apiClient.Token);
        }

        // Метод для зупинки служби сповіщень
        public async void StopNotificationService()
        {
            await _notificationService.StopAsync();
        }
    }
}
```