МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії
Кафедра комп'ютерних технологій та економічної кібернетики

Допущено до захисту:
Завідувач кафедри
комп'ютерних технологій та
економічної кібернетики
д. е. н., проф. П. М. Грицюк
_____
«____» _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня «магістр»
за освітньо-професійною програмою
«Інформаційні технології в бізнесі»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Інтелектуальна система моніторингу та управління виконанням завдань»

Виконав:
здобувач вищої освіти 2 курсу,
групи ІТБ-61м
Приходчук Василь Олександрович
(прізвище, ім'я, по-батькові)
Керівник:
к. т. н. доцент Барановський В.С.
(науковий ступінь, вчене звання прізвище та ініціали)
Рецензент:
д. е. н. професор Грицюк П.М.
(науковий ступінь, вчене звання прізвище та ініціали)

Рівне – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 57 с., 31 рис., 5 табл., 13 літературних джерел.

Актуальність теми даної магістерської роботи полягає у створенні інтелектуальної системи моніторингу та управління виконанням завдань, яка базується на сучасних інформаційних технологіях, таких як Node.js, MySQL та Telegram API. Ця система спрямована на автоматизацію процесів управління завданнями, аналізу даних та інтеграції з іншими системами та сервісами.

Об’єкт дослідження – процеси управління завданнями в інформаційно-обчислювальному центрі.

Предмет дослідження – методи автоматизації моніторингу, аналізу й управління завданнями за допомогою чат-ботів.

Мета роботи – розробка інтелектуальної системи моніторингу та управління завданнями для оптимізації процесів управління, підвищення ефективності координації та забезпечення інтерактивної взаємодії користувачів із системою.

У магістерській роботі виконано: аналіз існуючих рішень і технологій для автоматизації управління завданнями, проектування архітектури системи на основі Node.js, MySQL та Telegram API, створення математичних і логічних моделей, що забезпечують автоматизацію процесів моніторингу, розробка веб-додатку адміністратора системи, який дозволяє управляти користувачами, завданнями та аналізувати ефективність роботи, впровадження Telegram-бота для інтерактивної взаємодії з користувачами.

КЛЮЧОВІ СЛОВА: ІНТЕЛЕКТУАЛЬНА СИСТЕМА, УПРАВЛІННЯ ЗАВДАННЯМИ, NODE.JS, MYSQL, TELEGRAM API, ЧАТ-БОТ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних.

ІОЦ – Інформаційно-обчислювальний центр.

IDE – Інтегроване середовище розробки.

ПЗ – Програмне забезпечення.

API – Інтерфейс програмування застосунків.

CRUD – Створення, читання, оновлення, видалення (основні операції з даними).

JSON – Текстовий формат обміну даними (JavaScript Object Notation).

Node.js – Середовище виконання JavaScript на сервері.

MySQL – Реляційна система управління базами даних.

Telegram API – Інтерфейс для інтеграції з месенджером Telegram.

ВСТУП

Сучасні організації стикаються з труднощами управління завданнями та моніторингу виконання через зростання обсягів інформації та необхідність координації між різними відділами. В умовах цифрової трансформації бізнесу та освіті ефективне управління завданнями є важливим чинником підвищення продуктивності та конкурентоспроможності. Більшість існуючих інструментів, таких як Jira, Asana та Trello, хоч і в певній мірі забезпечують функціональність для управління завданнями, дуже часто виявляються надто складними для малих і середніх компаній або не враховують специфіку окремих організацій. Це створює потребу у простих та адаптивних рішеннях, які можуть бути інтегровані у повсякденні робочі процеси.

Популярні рішення зазвичай обмежуються статичними системами планування, які недостатньо використовують можливості сучасних технологій, зокрема штучного інтелекту (ШІ). Інтелектуальні системи, що поєднують аналіз даних, автоматизацію процесів та інтерактивні інтерфейси, наприклад чат-боти, здатні значно поліпшити ефективність управління завданнями. Проте у даній сфері досі залишаються нерозв’язані проблеми, такі як забезпечення інтерактивності, адаптивності до потреб користувачів і простоти впровадження.

Актуальність теми кваліфікаційної роботи визначається потребою у створенні інтелектуальної системи моніторингу та управління виконанням завдань, яка поєднує можливості ШІ, інтеграції з популярними платформами управління проектами та зручність використання. Це відповідає сучасним тенденціям цифровізації робочих процесів та сприяє підвищенню їх продуктивності. Робота узгоджується з напрямками центру, спрямованими на дослідження та впровадження інноваційних ІТ-рішень, програмами університету щодо підтримки проектів цифрової трансформації, а також регіональними та державними ініціативами розвитку інформаційних технологій.

Мета роботи – розробка інтелектуальної системи моніторингу та управління виконанням завдань у форматі інтерактивного чат-бота, що забезпечує автоматизацію та оптимізацію робочих процесів.

Завдання дослідження:

1. Проаналізувати існуючі системи управління завданнями та визначити їх недоліки.
2. Розробити архітектуру інтелектуальної системи моніторингу та управління.
3. Реалізувати чат-бот для автоматизації моніторингу, створення та призначення завдань.
4. Інтегрувати розроблену систему з популярними платформами управління проектами (Jira, Asana).
5. Провести тестування системи та оцінити її ефективність.

Об’єкт дослідження – процес управління завданнями у рамках бізнес-процесів організації.

Предмет дослідження – інтелектуальні підходи до автоматизації моніторингу та управління виконанням завдань із використанням чат-ботів.

Методи дослідження:

- Аналіз: для оцінки існуючих рішень та визначення їх недоліків.
- Проектування систем: для створення архітектури чат-бота.
- Програмування: для реалізації функціоналу системи.

Результатом дослідження є інтерактивна система, яка дозволяє оптимізувати робочі процеси завдяки використанню сучасних ІТ-рішень. Практичні результати можуть бути впроваджені в бізнес-середовище, сприяючи зростанню ефективності роботи команд.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1. 1. Аналіз інтернет-комунікаційних технологій

У сьогоднішніх реаліях месенджери, які стали сучасними інструментами комунікацій, стали вільно використовуватися в різних сферах життєдіяльності людини з метою більшої комунікації з іншими користувачами в мережі Інтернет. Сьогодні найбільше значення чат-боти мають у формуванні такої моделі поведінки, яка буде максимально наближена до людської. Ці додатки, розроблені на основі нейромереж та технологій машинного навчання, стали популярнішими, коли почалося їх використання в месенджерах і соціальних мережах (прикладом є Facebook, Viber, Telegram). Завдяки новому інструменту цифрової комунікації з'явилася можливість отримувати посилання на свіжі новини, дізнаватися про спеціальні можливості та пропозиції онлайн та отримувати обслуговування в сфері товарів і послуг, здійснювати складні операції. Ідея створення програм, які імітують людське спілкування, виникла ще в середині XX століття. Основні етапи розвитку чат-ботів включають:

1. **ELIZA (1966) –** перший чат-бот, розроблений Джозефом Вейзенбаумом. ELIZA імітувала розмову психотерапевта, використовуючи прості шаблони для аналізу тексту. Незважаючи на обмеженість, цей проєкт став першим кроком у розвитку технології.
2. **PARRY (1972) –** чат-бот, створений для симуляції пацієнта з шизофренією. PARRY був більш складним, ніж ELIZA, і демонстрував здатність обробляти контекст.
3. **A.L.I.C.E (1995) –** чат-бот на основі AIML (Artificial Intelligence Markup Language), який використовував шаблони для аналізу тексту. A.L.I.C.E став популярним завдяки інтеграції з різними платформами.
4. **Siri, Alexa та Google Assistant (2010-ті роки) –** з розвитком технологій обробки природної мови (NLP) і штучного інтелекту чат-боти еволюціонували в інтерактивних голосових помічників, здатних виконувати складні завдання.

5. **Месенджер-боти (2020-ті роки) –** впровадження ботів у месенджери, такі як Telegram, WhatsApp і Facebook Messenger, зробило технологію доступною для широкого кола користувачів. Боти стали важливим інструментом для бізнесу, освіти та комунікації.

Чат-боти можна описати як спеціальні програми, які здійснюють інтернет спілкування з одним або декількома користувачами. На даний момент сучасні додатки, месенджери та інтернет комунікації здатні виступати в якості віртуального співрозмовника, а також повторювати і відтворювати письмовий набір знаків людини. Можуть надавати запрограмовані відповіді на задані питання, або ж шукати у мережах інформацію яку потребує користувач. Перевагами чат-ботів є їх доступність, масштабованість та інтеграція з різними платформами, включаючи месенджери, веб-сайти та мобільні додатки.
Чат-боти застосовуються у різних сферах, таких як:

- Онлайн підтримка клієнтів: автоматичні відповіді на запитання, консультації.
- Продаж та маркетинг: обробка замовлень, персоналізовані пропозиції.
- Навчання та освіта: інтерактивні курси, підтримка студентів у навчальному процесі.
- Управління завданнями: створення та моніторинг статусів завдань, нагадування.

Словосполучення «чат-бот» йде від двох англійських слів: tochat - невимушена розмова в мережі Інтернет, bot (robot) - скорочено від слова робот, з чого зробимо висновок, що це роботи, призначені для здійснення передачі інформації між користувачами в мережі Інтернет, які виконують дії відповідно до сценарію який в них заклали. Такі програми як чат-боти засновані на сучасних технологіях.

Звернувши увагу на поточну оцінку стану ринку, чат-ботів, особливо для месенджерів – дуже перспективний напрямок у створенні програм, який в даний період стрімко набуває популярності (рис. 1.1).

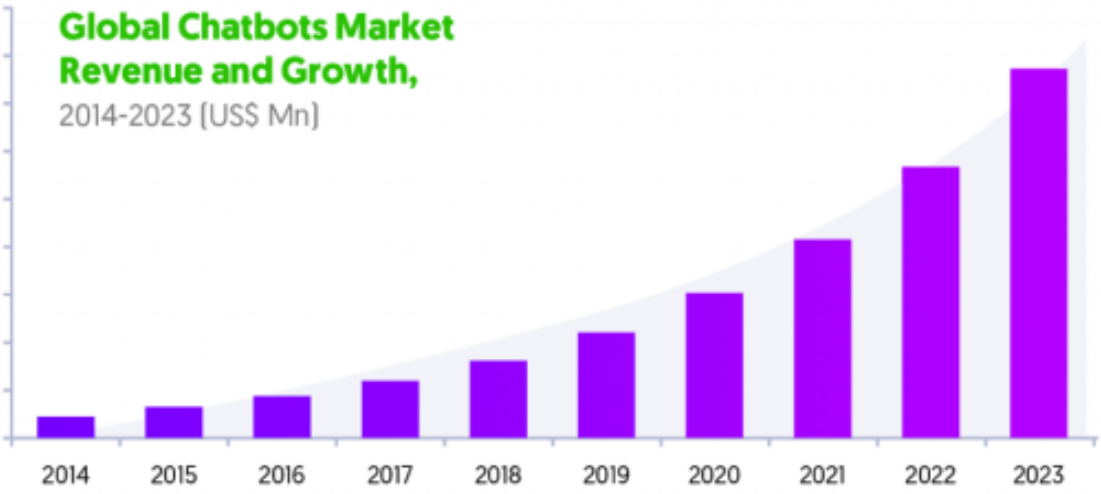

Рис 1.1. Тенденції на ринку чат-ботів

1.2. Порівняльний аналіз месенджерів з платформою чат-ботів

Месенджер – це система, яка призначена для обміну миттєвими повідомленнями (текстовими, мультимедійними), і може включати додаткові сервіси, такі як голосові та відеовиклики, інтеграція з чат-ботами, опитування, магазини покупок, ігри. Месенджери активно використовуються мільйонами користувачів для комунікації, організації завдань та доступу до інформації.

У контексті розробки інтелектуальної системи моніторингу та управління виконанням завдань для інформаційно-обчислювального центру НУВГП доцільно обрати один із сучасних месенджерів із підтримкою платформи чат-ботів. Для аналізу розглянемо такі популярні месенджери: Facebook Messenger, Telegram, Skype та Viber.

Таблиця 1.1. порівняльна характеристика месенджерів для чат-боту

	Facebook	***Telegram***	***Skype***	***Viber***
Форма використання	Умовно безоплатна	Безоплатна	Умовно безоплатна	Умовно безоплатна
Доступи	Доданий до групи, за підпискою, вбудований в діалог	Доданий до групи, за підпискою, вбудований в діалог	Доданий до групи, за підпискою	За підпискою, вбудований в діалог
Інтерфейси користувача	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий	Текстовий, кнопковий, голосовий
Механізми зв'язку	*webhook*	*webhook, polling*	*webhook*	*webhook*
Протоколи передачі даних	*https*			
Формати передачі даних	*JSON, multipart/form-data*	*JSON, URL query, multipart/form-data*	*JSON, multipart/form-data*	*JSON, multipart/form-data*
Функціональні можливості	Листування, опитування, передача файлів, магазин покупок, ігри	Листування, опитування, передача файлів, магазин покупок, ігри	Листування, передача файлів, магазин покупок, ігри	Листування, передача файлів, магазин покупок, ігри

Після аналізу месенджерів, Telegram є найкращою платформою для реалізації інформаційного чат-бота завдяки таким перевагам:

- Відкритість API: дозволяє легко розробляти та інтегрувати бота з іншими сервісами.
- Безпека та швидкість: протокол MTProto забезпечує ефективну передачу даних.
- Функціональні можливості: інтерактивні кнопки, опитування, мультимедіа та зручна інтеграція.
- Доступність: підтримка всіх основних платформ та пристроїв.

Telegram найкраще підходить для розробки чат-бота, спрямованого на автоматизацію моніторингу та управління завданнями, враховуючи вимоги інформаційно-обчислювального центру НУВГП.

1.3. Аналіз сучасних інструментів управління завданнями

Для успішного впровадження інтелектуальної системи моніторингу та управління виконанням завдань важливо звернути увагу на сучасні технології, що вже довели свою ефективність у різних сферах, таких як автоматизація комунікацій. Зокрема, чат-боти є потужним інструментом, здатним забезпечити швидкий доступ до інформації, автоматизацію рутинних операцій і персоналізовану взаємодію з користувачами.

Однак лише використання чат-ботів не вирішує всіх задач, пов’язаних з управлінням завданнями в організаціях. Для створення дійсно ефективної системи потрібно інтегрувати їх з більш широкими платформами, які дозволяють моніторити, аналізувати та оптимізувати процеси. У цьому контексті чат-бот стає лише одним із компонентів загальної системи контролю виконання завдань.

Сучасні інструменти управління завданнями, такі як Jira, Asana чи Trello, демонструють ефективність командної роботи, з іншої сторони вони часто мають обмеження щодо інтеграції з унікальними внутрішніми процесами конкретної організації. Для таких випадків необхідно розробляти кастомізовані рішення, які об’єднують найкращі практики управління завданнями із можливостями інтелектуальних систем.

Таким чином, наступним логічним кроком є розробка інтегрованої системи, яка об’єднує можливості автоматизованого спілкування через чат-бот із потужними інструментами для моніторингу та управління завданнями. Така система дозволить не лише автоматизувати рутинні операції, а й забезпечити прозорість процесів, підвищити ефективність роботи команди та надати аналітичні інструменти для прийняття обґрунтованих рішень.

Інформаційні технології значно трансформували процеси управління завданнями, дозволяючи автоматизувати рутинні операції, підвищувати

продуктивність працівників та покращувати координацію між командами. Етапи розвитку цих технологій можна поділити на кілька основних етапів:

1. Ручне управління та електронні таблиці – на початкових етапах автоматизації використовувалися прості інструменти, такі як електронні таблиці (Microsoft Excel, Google Sheets). Вони забезпечували базову функціональність для управління завданнями, проте не були інтегрованими рішеннями і потребували багато ручної роботи.
2. Системи управління проектами – виникнення спеціалізованих інструментів, таких як Microsoft Project, забезпечило можливість централізованого управління проектами. Ці системи дозволяли планувати ресурси, розподіляти завдання та моніторити їх виконання.
3. Хмарні рішення та інтеграція – подальший розвиток призвів до появи хмарних платформ. Ці інструменти забезпечують доступність з будь-якого пристрою, підтримують інтеграцію з іншими сервісами, такими як електронна пошта або календарі.
4. Інтелектуальні системи моніторингу та управління – сучасні розробки використовують штучний інтелект, інтерактивні інтерфейси (наприклад, чат-боти) та інтеграцію з аналітичними системами. Ці інструменти дозволяють аналізувати дані в реальному часі, прогнозувати результати та автоматично оптимізувати робочі процеси.

Інформаційні системи стали невід'ємною частиною бізнес-процесів, зокрема у сфері управління завданнями. На сучасному етапі розвитку інформаційних технологій є велика кількість інструментів, зосереджених на підвищення ефективновної роботи команд в організація та забезпечення прозорості виконання завдань. Найпоширеніші серед них – Jira, Asana та Trello, а також open-source рішення, такі як Redmine та Taiga.

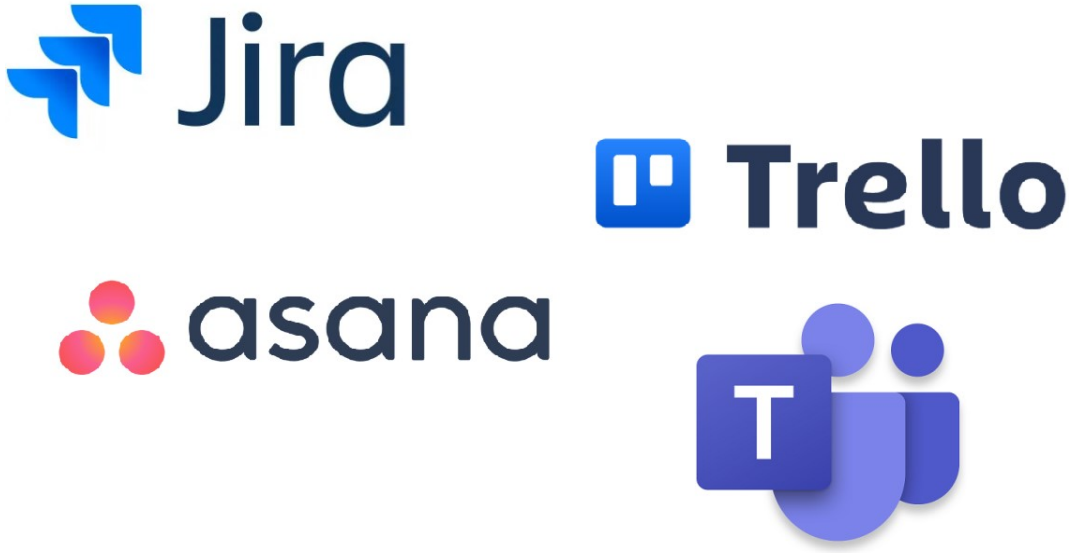

Рис. 1.2. Найпопулярніші системи управління проектами

Попри велике різноманіття функціональності, до основних недоліки цих систем можна віднести:

- Складності інтерфейсу для нових користувачів.
- Відсутності гнучкості для адаптації під потреби малого та середнього бізнесу.
- Обмежених можливостях інтеграції із специфічними системами організації, як-от внутрішні платформи університетів чи дослідницьких центрів.

Підсумувавши що для інформаційно-обчислювального центру НУВГП, де використання таких систем є важливим, ці інструменти не завжди враховують специфічні завдання, пов’язані з технічним обслуговуванням, цифровим документообігом та інноваційними навчальними платформами.

1.4. Потреби організацій у засобах автоматизації управління завданнями

Інтелектуальні системи моніторингу дозволяють автоматизувати контроль виконання завдань, оптимізувати планування ресурсів та забезпечувати гнучке реагування на зміни. Серед сучасних інтелектуальних рішень можна виділити:

- Штучний інтелект (ШІ) для аналізу даних і прогнозування ефективності завдань.
- Чат-боти як інтерактивний інтерфейс для виконання операцій у системі управління.
- Інтеграційні платформи для зв'язку з існуючими інструментами, такими як Slack, Google Workspace, та внутрішніми сервісами.

Основні переваги інтелектуальних систем:

- Постійна доступність інформації в реальному часі.
- Мінімізація ручної роботи через автоматизацію рутинних завдань.
- Гнучкість у адаптації до різних потреб організацій.

Проте аналіз показує, що більшість існуючих рішень недостатньо враховують специфіку роботи освітніх і технічних закладів, зокрема інформаційно-обчислювальних центрів.

Для інформаційно-обчислювального центру НУВГП основними потребами є:

- Автоматизація моніторингу стану серверного та периферійного обладнання.
- Формування звітності виконаних завдань.
- Забезпечення швидкої взаємодії між працівниками через цифрові інструменти.
- Інтеграція існуючих сервісів, таких як мобільні додатки та службова електронна пошта.

Інтелектуальна система, яка буде розроблена у рамках цієї роботи, повинна відповідати таким вимогам:

- Простота використання через чат-бот у популярних месенджерах (Telegram).
- Інтеграція з платформами, що вже впроваджені в університеті.
- Використання автоматизації для рутинних операцій, таких як створення заявок на ремонт або планування завдань.

1.5. Постановка задачі та визначення основних вимог до системи

На основі аналізу об’єкта дослідження можна виділити такі нерозв’язані питання:

- Відсутність єдиної системи для об'єднання інформаційних платформ НУВГП.
- Необхідність автоматизації комунікації між підрозділами через інтерактивні інструменти.
- Відсутність адаптивної системи для моніторингу та управління завданнями.

У рамках роботи пропонується розробка інтелектуальної системи моніторингу та управління виконанням завдань, яка враховує специфіку роботи інформаційно-обчислювального центру. Основними функціями системи мають стати:

1. Моніторинг статусу завдань – бот може надавати інформацію про поточний статус завдань: виконано, в процесі, затримка, очікує підтвердження тощо. Інтеграція з системами управління проектами (Jira, Asana) для отримання актуальних даних.
2. Створення та призначення завдань – користувачі можуть створювати завдання, призначати їх відповідальним особам і задавати терміни виконання. Оповіщення при наближенні терміну або виконанні завдання.

3. Аналітика та звітність – автоматичне генерування звітів про виконання завдань (продуктивність команди, час виконання завдань, проблеми, що виникають). Використання графіків та діаграм для візуалізації результатів.
4. Рекомендації по оптимізації – використання алгоритмів машинного навчання для прогнозування та оптимізації планування завдань. Аналіз історії виконання завдань для надання рекомендацій щодо поліпшення продуктивності.
5. Інтерактивні опитування та перевірка – користувачі можуть взаємодіяти з ботом для підтвердження виконання етапів завдань або проходження міток контролю якості.
6. Нагадування та оповіщення Налаштування персоналізованих нагадувань для виконання завдань або на основі пріоритетів

Таким чином, месенджер Telegram як платформа та розроблений чат-бот стане ефективними інструментом для автоматизації робочих процесів, підвищення продуктивності та покращення внутрішньої комунікації в центрі.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

2.1. Архітектура системи та опис компонентів

Одним з найважливіших етапом проектування ПЗ є вибір архітектури. Архітектура системи – це ключовий елемент проєкту, який визначає спосіб взаємодії компонентів і забезпечує їхню узгоджену роботу. Для реалізації системи моніторингу та управління виконанням завдань за допомогою інтерактивного чат-боту в Telegram використана багаторівнева модульна архітектура. Що дозволяє забезпечити гнучкість, масштабованість та зручність подальшої підтримки системи. Згідно з сформованими функціональними вимогами вирішено розглянути наступні архітектурні шаблони та стилі:

1. Клієнт-серверна архітектура.
2. Front end і back end.
3. Монолітна/мікросервісна архітектура.
4. Модульна структура проєкту.

Серверною стороною є сервери Telegram по відношенню до логіки чат-бота, сервер логіки чат-бота по відношенню до клієнтського веб інтерфейсу, а клієнтами – сервери логіки чат-бота по відношенню до серверів Telegram та клієнтський веб-інтерфейс по відношенню до серверів логіки чат-бота (рис. 2.1).

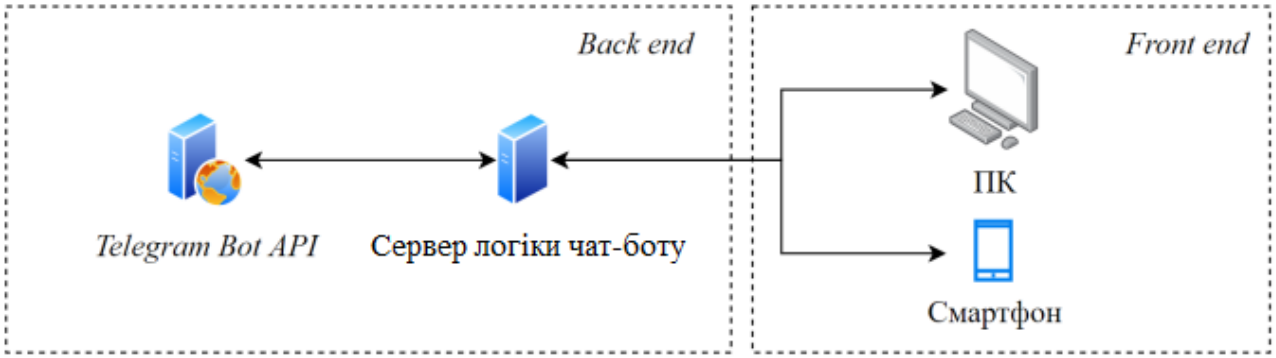

Рис. 2.1. Схема клієнт-серверної архітектуру системи

Представлена схема демонструє архітектуру Telegram-бота, який реалізує функції моніторингу та управління завданнями. Вона складається з двох

основних частин: Back-end і Front-end, а також включає Telegram Bot API як посередник для передачі даних.

Front-end для Telegram-бота виконує роль клієнтської частини системи, через яку користувачі взаємодіють із ботом. У контексті розробки системи моніторингу та управління завданнями, даний інтерфейс для користувача забезпечує доступність функцій бота на різних пристроях, таких як ПК, смартфони та планшети. Telegram як платформа виступає основним засобом взаємодії завдяки інтуїтивному зрозумілому інтерфейсу, широкій підтримці операційних систем і можливостям інтерактивної взаємодії.

Telegram як месенджер забезпечує користувачів засобами для обміну текстовими командами, отримання мультимедійних відповідей, а також доступу до інтерактивних елементів, таких як кнопки меню або опитування. У випадку розробленої системи, через Front-end користувачі мають змогу створювати нові завдання, перевіряти статус існуючих, отримувати аналітичні звіти або автоматичні нагадування. Цей інтерфейс є максимально зручним для кінцевого користувача, тому що всі операції виконуються у знайомому середовищі Telegram, без потреби в окремих програмах чи додатках.

Ключовою перевагою такого Front-end є простота використання та мінімальний навички для входу, що робить систему доступною для будь-якого співробітника від інформаційно-обчислювального центру та за його межами. З технічної точки зору, цей інтерфейс взаємодіє з серверною частиною через Telegram API, що дозволяє швидко передавати запити та отримувати відповіді в реального часу. Це забезпечує високу продуктивність та інтерактивність роботи з системою.

Велике значення у програмуванні бота має API — програмний інтерфейс, за допомогою якого програми, веб-сервіси та додатки обмінюються даними. Ботами в Telegram керують програми, які звертаються до серверів месенджера, щоб отримувати дані (наприклад, про повідомлення, отримане ботом) і надсилати команди (наприклад, відповісти на повідомлення). Зв’язок між програмами, які керують ботами, і серверами здійснюється через API.

Основний API, за допомогою якого відбувається обмін даними — це Telegram API (MTProto API). Він базується на протоколі MTProto, створеному командою Telegram, який є відкритим, щоб розробники могли писати власні програми для мессенджера. Крім нього, існує Bot API — це додатково розроблений API поверх MTProto API, який використовується лише ботами. Його створено для того, щоб програмісти могли писати ботів на стандартних HTML-запитах, не витрачаючи час на ознайомлення з MTProto.

Back-end Telegram-бота є ключовим компонентом системи, що відповідає за обробку запитів користувачів, виконання бізнес-логіки та забезпечення зв'язку з іншими компонентами, такими як база даних та інтеграційні модулі. Це серверна частина, яка обробляє всі запити, надіслані через Telegram API, визначає тип запиту (наприклад, створення нового завдання, моніторинг або аналіз даних) і виконує відповідні дії.

Основні функції Back-end включають розпізнавання та обробку текстових команд, зберігання даних у базі (наприклад, інформації про завдання, користувачів та історію змін), а також інтеграцію з зовнішніми системами. Сервер може взаємодіяти з такими платформами, як Moodle, Jira або Google Calendar, використовуючи REST API, забезпечуючи двосторонній обмін даними. Для досягнення високої продуктивності Back-end побудований на основі асинхронних технологій, таких як Node.js або Python (Flask/Django), що дозволяє одночасно обробляти велику кількість запитів.

Серверна частина також реалізує алгоритми аналітики, які допомагають виявляти критичні завдання, прогнозувати затримки або формувати звіти. Завдяки модульній архітектурі Back-end легко розширюється: можна додавати нові функції, інтегрувати додаткові сервіси або впроваджувати більш складну логіку. Такий підхід забезпечує гнучкість, масштабованість і надійність системи, роблячи її ефективним інструментом для автоматизації робочих процесів в інформаційно-обчислювальному центрі.

Підсумувавши дана система працює наступним чином:

1. Користувач (через ПК або смартфон) надсилає запит боту у Telegram.

2. Telegram Bot API отримує цей запит і пересилає його серверу логіки чат-бота.
3. Сервер обробляє запит:
    - Звертається до бази даних для отримання чи збереження інформації.
    - Виконує необхідну інтеграцію з іншими системами, якщо це потрібно (наприклад, отримання даних з Moodle).
    - Формує відповідь на запит користувача.
4. Telegram Bot API передає сформовану відповідь назад у Telegram.
5. Користувач отримує відповідь через свій інтерфейс (ПК чи смартфон).

Модульна структура – це підхід до організації програмного забезпечення, який розділяє проект на окремі частини (модулі), що мають конкретну відповідальність і можуть працювати незалежно один від одного. Цей підхід забезпечує кращу зрозумілість, підтримуваність та масштабованість проекту.

Основні принципи модульної структури:

1. Розділення обов’язків (Separation of Concerns)

   Кожен модуль відповідає за певну частину функціональності програми (наприклад, робота з базою даних, обробка запитів, бізнес-логіка тощо).
2. Повторне використання (Reusability)

   Код, який виконує одну функцію, може бути легко використаний у різних частинах програми.
3. Чіткі інтерфейси

   Модулі взаємодіють між собою через визначені інтерфейси, що забезпечує низьку зв’язаність (low coupling).
4. Масштабованість

   Додати нову функціональність або змінити існуючу легше, оскільки зміни зазвичай зосереджуються в окремому модулі.
5. Простота тестування

   Модулі можна тестувати окремо, що спрощує виявлення та виправлення помилок.

Структура модульного проекту – модульна структура часто організована так, щоб полегшити підтримку і розробку. Типовий проект може містити такі компоненти:

1. Контролери (controllers/)

   Контролери – це логіка, яка відповідає за обробку запитів від користувача. Вони отримують дані з маршрутизаторів (routes) і викликають відповідні сервіси.
2. Моделі (models/)

   Моделі відповідають за взаємодію з базою даних. Вони визначають структуру даних та операції з ними.
3. Маршрути (routes/)

   Визначають, який контролер викликається у відповідь на певний HTTP-запит або подію (наприклад, запити до API чи повідомлення у Telegram-боту).
4. Сервіси (service/)
5. Сервіси містять бізнес-логіку, яка може використовуватися різними частинами програми. Наприклад, API-запити до зовнішніх систем, алгоритми чи обробка складних даних.
6. Утиліти (utils/)
7. Допоміжні функції або загальні модулі, які можуть бути використані в будь-якому місці проекту. Наприклад, логування, форматування дат тощо.
8. Головний файл (index.js)

   Цей файл ініціалізує додаток: підключає маршрути, налаштовує сервер або Telegram-бот і запускає програму.
9. Файл конфігурації (.env)

   Використовується для збереження секретів та змінних середовища (ключі API, строки підключення до бази даних).

До переваги модульної структури можна віднести:

1. Зрозумілість – коли функціональність розділена на модулі, структура проекту стає зрозуміло для розробників.
2. Масштабованість – Легко додати нові функції без порушення роботи існуючих частин програми.
3. Легкість підтримки – помилки легше локалізувати і виправити, оскільки код розділений на чітко окреслені частини.
4. Командна робота – модульна структура дозволяє різним членам команди працювати над різними модулями паралельно.
5. Мінімізація повторень – завдяки повторному використанню модулів можна уникнути дублювання коду.

```
/project
|
├── controllers/      # Контролери: управління логікою запитів
|   └── userController.js
|
├── models/           # Моделі: управління даними
|   └── userModel.js
|
├── routes/           # Маршрутизація
|   └── userRoutes.js
|
├── service/          # Сервіси: бізнес-логіка
|   └── emailService.js
|
├── utils/            # Утиліти
|   └── logger.js
|
├── .env              # Конфігурація середовища
├── database.js       # Підключення до бази даних
├── index.js          # Точка входу в додаток
├── package.json      # Конфігурація проекту npm
└── README.md         # Документація
```

Рис. 2.2. Приклад модульної структури у Node.js.

Модульна структура є стандартом для сучасних проектів через свою ефективність та зручність у розробці, особливо в багатофункціональних системах, таких як Telegram-боти чи веб-додатки.

2.2. Проектування бази даних для зберігання даних про завдання

Для забезпечення ефективного зберігання, доступу та обробки даних у системі моніторингу та управління завданнями критично важливим є проєктування надійної бази даних. Вона слугує основним сховищем інформації, забезпечуючи організацію роботи з такими елементами, як завдання, користувачі, історія змін і взаємодія з інтегрованими сервісами. Основна мета бази даних полягає у створенні структурованої та логічно зв’язаної системи, яка відповідає вимогам швидкого доступу до даних і забезпечує зручність подальшої роботи з ними.

База даних у системі реалізована за допомогою реляційної моделі з використанням технології MySQL, яка є однією з найпопулярніших систем управління базами даних завдяки своїй надійності, високій продуктивності та здатності працювати з великими обсягами інформації. Вона дозволяє ефективно організовувати дані за допомогою таблиць із чіткими зв’язками між ними. Така організація дозволяє не лише уникати дублювання даних, але й спрощує роботу із запитами для аналізу та формування звітів.

Особливістю спроектованої бази даних є її масштабованість і модульність, що дозволяє легко додавати нові елементи чи інтегруватися з додатковими сервісами, такими як Moodle або Jira. Використання MySQL також забезпечує можливість підтримки високої цілісності даних завдяки чітко визначеним зв’язкам між таблицями, обмеженням цілісності (foreign keys) і можливості автоматизації оновлень через тригери та події. Таким чином, база даних виступає фундаментом усієї системи, забезпечуючи її ефективність, гнучкість і надійність у роботі з даними.

Розробка бази даних для системи моніторингу та управління завданнями вимагає дотримання низки ключових вимог, які забезпечують її ефективність,

надійність та відповідність бізнес-цілям. Дані вимоги враховують необхідність швидкого доступу до інформації, збереження історії змін, підтримки цілісності даних та можливості інтеграції із зовнішніми сервісами.

1. Структурованість та нормалізація – база даних повинна бути структурована таким чином, щоб уникнути надмірного дублювання даних і мінімізувати ризик виникнення логічних помилок. Для цього таблиці мають бути нормалізовані до принаймні третьої нормальної форми (3NF). Це забезпечить:

- Зберігання даних у чітко визначених таблицях (наприклад, окремі таблиці для завдань, користувачів, історії змін).
- Уникнення дублювання записів.
- Полегшення оновлення та зміни даних у системі.

2. Цілісність даних – забезпечення референційної цілісності необхідно використовувати первинні (PRIMARY KEY) та зовнішні ключі (FOREIGN KEY) у таблицях. Це гарантує, що всі записи в базі даних матимуть чіткі зв’язки між собою, наприклад:

- Кожне завдання буде пов’язане з відповідальним користувачем.
- Журнал змін завдання буде посилатися на відповідне завдання та користувача, який вніс зміни.

Це також дозволить уникнути ситуацій, коли дані про завдання чи користувачів втрачають свою узгодженість.

3. Масштабованість – система повинна бути здатною обробляти великий обсяг даних, який з часом збільшуватиметься. Наприклад:

- Зростання кількості завдань у системі.
- Додавання нових функцій, які потребують нових таблиць або змін у вже існуючих структурах. Масштабованість досягається через продуману архітектуру бази даних і оптимізацію запитів.

4. Швидкість доступу – для забезпечення швидкого доступу до даних важливо:

- Використовувати індекси на полях, які найчастіше використовуються в запитах (наприклад, status у таблиці завдань або assigned_to для пошуку завдань конкретного користувача).

- Оптимізувати складні SQL-запити, що включають об’єднання таблиць (JOIN).

5. Зберігання історії змін – оскільки система включає моніторинг завдань, важливо зберігати повну історію їх змін. Це реалізується через таблицю Task_History, яка фіксує:

- Коли і ким було змінено завдання.
- Які зміни були внесені (наприклад, зміна статусу чи термінів виконання). Це дозволяє відстежувати повну історію виконання завдань, що особливо важливо для аналітики та звітності.

6. Інтеграція із зовнішніми сервісами – база даних повинна мати можливість зберігати інформацію для інтеграції з іншими системами, такими як Moodle чи Jira. Наприклад:

- API-ключі та токени для доступу до зовнішніх платформ.
- Лог синхронізації з іншими системами для відстеження оновлень.

7. Безпека – оскільки система зберігає конфіденційну інформацію, база даних повинна бути захищена від несанкціонованого доступу. Основні аспекти безпеки включають:

- Розмежування прав доступу до таблиць та запитів для різних типів користувачів (адміністратор, виконавець).
- Шифрування конфіденційних даних, таких як паролі або API-ключі.
- Регулярне створення резервних копій даних.

8. Гнучкість – система повинна дозволяти легко змінювати структуру бази даних у разі розширення функціональності. Наприклад, можливість додати нові поля до таблиці завдань або створити нові таблиці для зберігання додаткових типів даних.

Дотримання цих вимог забезпечує створення надійної, продуктивної та гнучкої бази даних, яка може обробляти поточні запити системи моніторингу завдань і легко адаптуватися до майбутніх потреб. Ця структура стане основою для ефективного функціонування системи й автоматизації процесів в інформаційно-обчислювальному центрі.

База даних містить кілька ключових таблиць, які забезпечують функціональність системи.

Далі визначимо атрибути для кожної сутності, а також поставимо для всіх сутностей первинні ключі та тип даних кожного поля сутностей. Відомості про таблиці бази даних представлені нижче у вигляді таблиць (табл. 2.1, 2.3).

Таблиця 2.1. Користувачі(Users).

id (PRIMARY KEY)	Унікальний ідентифікатор користувача
name	Ім’я користувача
role	Роль у системі (адміністратор, виконавець, менеджер)
email	Електронна адреса для сповіщень
created_at	Дата реєстрації користувача

Таблиця 2.2. Історія змін завдань(Task_History).

id (PRIMARY KEY)	Унікальний ідентифікатор користувача
title	назва завдання
description	детальний опис
status	статус виконання (нове, у процесі, завершене)
priority	пріоритет (низький, середній, високий).
assigned_to	(FOREIGN KEY) – посилання на користувача, відповідального за
deadline	кінцевий термін виконання
created_at	Дата створення завдання

Таблиця 2.3. Інтеграції(Integrations).

id (PRIMARY KEY)	Унікальний ідентифікатор користувача
service_name	Назва сервісу
api_key	Ключ доступу для інтеграції
last_sync	Дата останньої синхронізації

Зв’язки між таблицями виглядають наступним чином:

- Users ↔ Tasks: Один користувач може бути відповідальним за кілька завдань (1:N).
- Tasks ↔ Task_History: Одне завдання може мати кілька записів у таблиці історії змін (1:N).

На концептуальному рівні структура бази даних представлена через взаємозв’язки між основними таблицями:

- Users: відповідають за створення, виконання та моніторинг завдань.
- Tasks: пов’язують користувачів і фіксують статуси завдань.
- Task_History: забезпечує журнал усіх змін у системі.

На Рис.2.3. зображено модель "Сутність-Зв'язок" для бази даних системи управління завданнями. Основні компоненти:

1. Users (Користувачі): Зберігає інформацію про користувачів системи (адміністраторів, виконавців тощо). Пов’язана з таблицею "Tasks" через поле assigned_to.
2. Tasks (Завдання): Містить основну інформацію про завдання (назва, опис, статус, терміни). Встановлює зв’язки:
    - З таблицею "Users" для призначення відповідального користувача.
    - З таблицею "Task_History" для фіксації змін у статусах завдань.
    - З таблицею "Integrations" для зв’язку з зовнішніми системами (наприклад, Moodle, Jira).
3. Task_History (Історія змін завдань): Відображає журнал змін, вказуючи, хто, коли і як змінив статус або інші параметри завдання.

4. Integrations (Інтеграції): Зберігає інформацію про підключені зовнішні сервіси (API-ключі, назви сервісів, дати синхронізації).

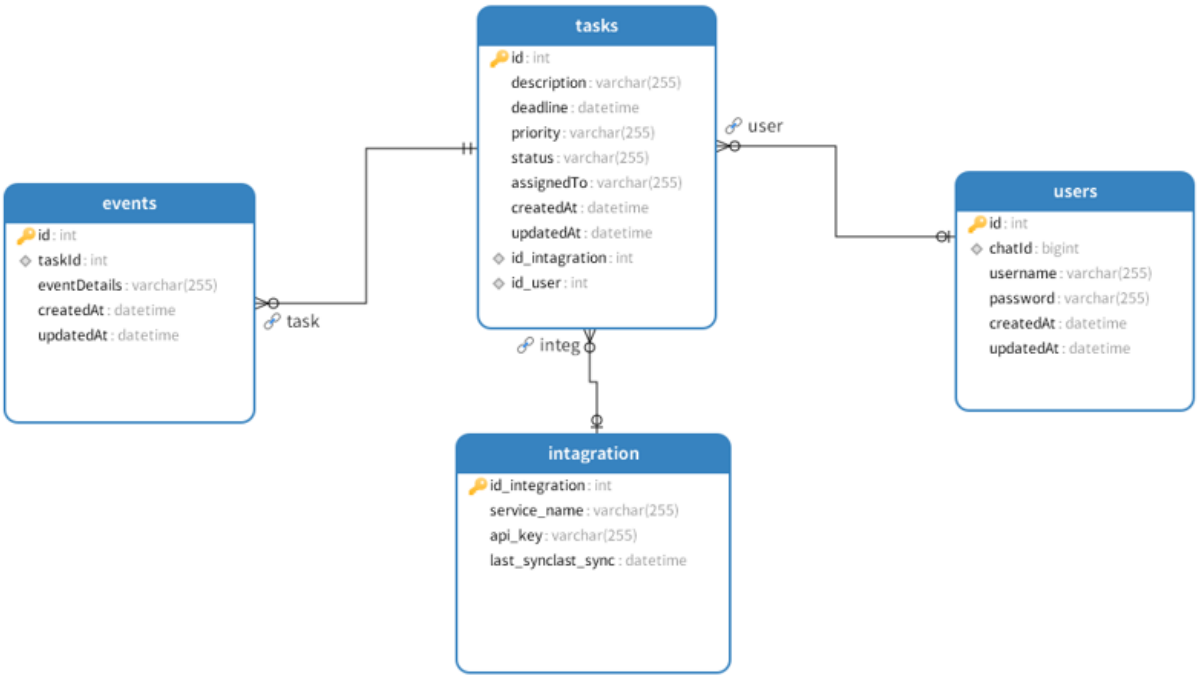

Рис. 2.3. Схема бази-даних

Спроектована база даних забезпечує ефективне управління даними про завдання, користувачів і інтеграції. Структура якої гнучка та масштабована, тому дозволяє додавати новий функціонал, такий як аналітика або інтеграція з додатковими сервісами. Правильна організація бази даних є важливим компонентом успішної реалізації системи управління завданнями.

2.3. Обґрунтування вибору технологій для реалізації системи (Node.js, MySQL, Telegram API)

Для створення ефективної системи моніторингу та управління виконанням завдань важливо обрати сучасні технології, які дозволяють легко масштабувати систему, забезпечують інтеграцію із зовнішніми сервісами та гарантують високу швидкість роботи. Основними компонентами системи є серверна частина, що обробляє запити і реалізує бізнес-логіку, база даних для надійного зберігання інформації та інтерактивний інтерфейс, який забезпечує зручну взаємодію з

користувачами. Ці складові повинні працювати як єдиний механізм для реалізації завдань, таких як створення, моніторинг і аналіз статусів завдань.

У даній системі інтеграція з популярним месенджером Telegram забезпечує можливість швидкої взаємодії між користувачами через знайомий інтерфейс. Це дозволяє обійти необхідність розробки окремого клієнтського додатку, зосереджуючи увагу на серверній частині та базі даних. Зважаючи на сучасні вимоги до систем автоматизації, таких як продуктивність, простота налаштування та інтеграція із зовнішніми платформами (наприклад, Moodle, Jira), вибір оптимальних технологій є ключовим для успішного впровадження.

У цьому розділі обґрунтовано вибір Node.js як платформи для серверної частини, MySQL для організації бази даних і Telegram API як інструменту взаємодії з користувачами. Ці технології були обрані через їхню продуктивність, масштабованість і сумісність, що забезпечує стабільну роботу системи навіть при високих навантаженнях. Кожна з технологій буде детально розглянута з точки зору її особливостей, переваг і ролі в архітектурі системи.

2.3.1. Платформа Node.js

Серверна частина системи відіграє ключову роль у роботі Telegram-бота, оскільки відповідає за обробку запитів, реалізацію бізнес-логіки, інтеграцію з базою даних і зовнішніми сервісами. Сьогодні існує багато платформ і технологій для створення серверної частини веб-додатків, серед яких популярними є Node.js, Python (Flask/Django) та Ruby on Rails і PHP (Laravel). Кожна з них має свої переваги та недоліки, які можуть вплинути на вибір у залежності від специфіки обраної тематики.

Для реалізації нашої системи Node.js було обрано через його асинхронну архітектуру, високу продуктивність та багатий набір бібліотек для роботи з API, включаючи Telegram API. Нижче наведено порівняння Node.js з іншими популярними технологіями, яке обґрунтовує його переваги в контексті реалізації системи моніторингу та керування завданнями.

Табл. 2.4. Порівняння Node.js з іншими технологіями

Критерій	**Node.js**	**Python (Flask/Django)**	**PHP (Laravel)**	**Ruby on Rails**
Модель виконання	Асинхронна (Event-driven)	Синхронна (Flask – мікрофреймворк; Django – повноцінний фреймворк)	Синхронна	Синхронна
Продуктивність	Висока (обробка великої кількості запитів одночасно)	Середня (краще підходить для CPU-інтенсивних завдань)	Середня	Середня
Масштабованість	Відмінна завдяки асинхронності	Помірна (вимагає додаткових інструментів для горизонтального масштабування)	Помірна	Помірна
Підтримка API	Широка, багато бібліотек, включаючи node-telegram-bot-api	Доступно через сторонні бібліотеки	Підтримка через HTTP-запити	Підтримка через HTTP-запити
Переваги	Висока швидкість, асинхронність, легка інтеграція	Зручний для аналітики та машинного навчання	Проста розробка, доступність	Гарний вибір для стартапів
Недоліки	Підходить менше для CPU-інтенсивних завдань	Менша швидкість у порівнянні з Node.js	Менше підходить для високих навантажень	Відносно повільна продуктивність

Рис. 2.4. Логотип Node.js

У таблиці вище наведено порівняння популярних технологій для розробки серверної частини, таких як Node.js, Python (Flask/Django), PHP (Laravel) та Ruby on Rails. Аналіз було проведено за ключовими критеріями: продуктивність, масштабованість, зручність роботи з Telegram API, швидкість розробки та інші параметри, що важливі для реалізації інтерактивної системи моніторингу та управління завданнями.

На основі результатів цього порівняння можна зробити висновок, що Node.js є найбільш підходящим вибором для розробки нашої системи завдяки своїм перевагам у швидкості, асинхронній архітектурі та високій масштабованості. Далі наведено детальне обґрунтування цього вибору:

- Асинхронна архітектура – Node.js використовує модель Event-driven, яка дозволяє ефективно обробляти велику кількість одночасних запитів без блокування основного потоку виконання. Ця особливість критично важлива для чат-ботів, які повинні швидко реагувати на запити користувачів у реальному часі.
- Висока продуктивність – завдяки рушію V8 JavaScript Engine, Node.js виконує JavaScript із високою швидкістю. Це робить його ідеальним для серверних додатків, які вимагають швидкої обробки даних і передачі повідомлень через Telegram API.

- Інтеграція з Telegram API – Node.js має багату екосистему бібліотек, таких як node-telegram-bot-api, які значно спрощують роботу з Telegram API. Це дозволяє швидко розробляти функціонал для надсилання й обробки повідомлень, інтерактивних кнопок і команд.
- Масштабованість – Node.js забезпечує легке горизонтальне масштабування, що дозволяє адаптувати систему до зростання кількості користувачів і обсягу оброблюваних даних. Це особливо важливо для довготривалих проєктів, які мають потенціал розширення.
- Універсальність JavaScript – Використання JavaScript для серверної частини дозволяє зменшити складність проєкту, оскільки одна мова може бути використана для всіх рівнів системи (сервер, клієнт, обробка інтерактивних елементів). Це також скорочує час розробки завдяки широкій доступності JavaScript-розробників.
- Велика кількість бібліотек і модулів – Node.js має багату екосистему пакетів у npm (Node Package Manager), які можна використовувати для швидкої реалізації функцій без необхідності писати їх із нуля. Це прискорює процес розробки та забезпечує стабільність коду.
- Активна підтримка та ком’юніті – Node.js активно підтримується спільнотою розробників і великими компаніями. Це означає регулярні оновлення, розширення функціоналу та доступність документації в навчальних матеріалів.

Node.js є найбільш оптимальним вибором для розробки серверної частини нашої системи. Його особливості забезпечують швидкість роботи, інтерактивність, легкість інтеграції з Telegram API і здатність масштабуватися, що є ключовими вимогами для успішної реалізації системи моніторингу та управління завданнями.

2.3.2. Вибір бази даних. MySql

База даних є критично важливою складовою системи моніторингу та управління завданнями, оскільки вона забезпечує зберігання, доступ і обробку

інформації про завдання, користувачів, історію змін і інтеграції з зовнішніми сервісами. Вибір відповідної технології бази даних залежить від потреб проєкту, таких як структурованість даних, швидкість запитів, цілісність і масштабованість. Сьогодні популярними рішеннями для реалізації реляційних баз даних є MySQL, PostgreSQL, SQLite, а також нереляційні бази даних, такі як MongoDB.

MySQL було обрано для реалізації системи завдяки її високій продуктивності, широкій підтримці в індустрії, багатому функціоналу та простоті інтеграції з серверною частиною на Node.js. MySQL є одним із найпопулярніших і надійних рішень для реляційних баз даних, що забезпечує ефективне управління даними навіть при великих обсягах інформації. Нижче наведено основні аргументи, які обґрунтовують вибір MySQL як базової технології для системи.

2.3.3. Обґрунтування вибору Telegram API

Telegram є одним із найпопулярніших месенджерів, який активно використовується для розробки інтерактивних чат-ботів завдяки його відкритому API, підтримці інтеграцій і багатофункціональному інтерфейсу. Для нашої системи моніторингу та управління завданнями Telegram обрано через зручність його використання як для кінцевих користувачів, так і для розробників. Крім того, Telegram забезпечує високий рівень безпеки, підтримує інтерактивні елементи (кнопки, меню, опитування) та легкий доступ через мобільні пристрої чи ПК.

Щоб обґрунтувати вибір Telegram, наведемо його порівняння з іншими популярними месенджерами: Facebook Messenger, Viber, Skype.

Таблиця 2.5 – Порівняння Telegram з іншими месенджерами

Критерій	**Telegram**	**Facebook Messenger**	**Viber**
Підтримка ботів	Відмінна, через відкритий API	Обмежений функціонал	Задовільна

продовження табл. 2.5.

Інтерактивність	Висока (кнопки, меню, опитування)	Середня	Обмежена
Масштабованість	Відмінна	Середня	Середня
Продуктивність	Швидка передача даних	Задовільна	Задовільна
Відкритість API	Безкоштовний та доступний API	Частково доступний через Meta API	Доступний, але з обмеженнями
Підтримка платформ	Універсальна (Android, iOS, ПК)	Android, iOS, веб	Android, iOS, ПК
Безпека	Висока, наскрізне шифрування	Середня	Висока

2.4. Середовище розробки

Розробка програмного забезпечення вимагає використання інтегрованих середовищ розробки (IDE) та інших інструментів, які забезпечують зручність роботи, підвищують продуктивність і сприяють швидкому вирішенню задач. IDE (Integrated Development Environment) – це програмне забезпечення, яке поєднує в собі текстовий редактор, інструменти для компіляції та налагодження, а також засоби інтеграції із зовнішніми системами. Використання якісного середовища розробки дозволяє мінімізувати кількість помилок у коді, полегшує тестування та забезпечує комфортну співпрацю в команді.

Для реалізації системи моніторингу та управління завданнями обрано два основних інструменти: Visual Studio Code для розробки серверної частини на Node.js та Navicat for MySQL для управління базою даних. Ці інструменти забезпечують інтегроване середовище для створення коду, управління даними,

налагодження та тестування, дозволяючи ефективно впроваджувати функціонал системи.

2.4.1. Visual Studio Code як IDE для Node.js

Для розробки серверної частини системи на платформі Node.js було обрано інтегроване середовище розробки Visual Studio Code (VS Code). Це безкоштовний, кросплатформний редактор коду, розроблений компанією Microsoft, який підтримує широкий спектр мов програмування та має багатий набір функцій для підвищення продуктивності розробки.

Переваги використання Visual Studio Code для Node.js:

1. Розширюваність та підтримка плагінів:
    - VS Code має вбудовану підтримку JavaScript та Node.js.
    - Можливість встановлення розширень, таких як Node.js Extension Pack, ESLint, Prettier, які полегшують написання та форматування коду.
    - Підтримка інтеграції з системами керування версіями (Git, SVN).
2. Інтелектуальні підказки коду (IntelliSense):
    - Автозаповнення коду та надання рекомендацій під час написання.
    - Відображення документації та типів даних для функцій та методів.
3. Вбудований термінал:
    - Можливість запуску команд безпосередньо в IDE.
    - Підтримка кількох терміналів для одночасного виконання різних задач.
4. Налагодження (Debugging):
    - Встановлення точок зупинки, перегляд стеку викликів, моніторинг змінних.
    - Підтримка налагодження як серверної частини Node.js, так і клієнтського JavaScript-коду.
5. Кросплатформність:
    - Доступність для операційних систем Windows, macOS та Linux.

- Можливість синхронізації налаштувань та розширень між різними пристроями.

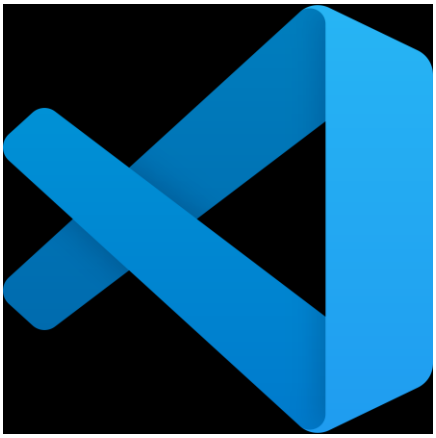

Рис. 2.5. Логотип Visual studio code

2.4.2. Navicat для MySQL

Для управління базою даних MySQL було обрано інструмент Navicat for MySQL. Це потужний та зручний GUI-інструмент для адміністрування, розробки та управління базами даних MySQL та MariaDB

Переваги використання Navicat for MySQL:

1. Інтуїтивно зрозумілий інтерфейс:
    - Зручне візуальне представлення структури бази даних.
    - Можливість швидкого доступу до таблиць, переглядів, процедур та тригерів.
2. Розробка та моделювання:
    - Інструменти для візуального проектування бази даних.
    - Підтримка діаграм ERD (Entity-Relationship Diagram) для моделювання зв’язків між таблицями.
3. Управління даними:
    - Зручний редактор даних з можливістю фільтрації, сортування та групування.
    - Функції імпорту та експорту даних у різних форматах (CSV, Excel, XML, JSON).
4. Виконання та налагодження SQL-запитів:

- Редактор SQL з підсвічуванням синтаксису та автозаповненням.
- Можливість виконання складних запитів та перегляду результатів у реальному часі.

5. Адміністрування серверу:
    - Управління користувачами та правами доступу.
    - Моніторинг продуктивності та стану сервера бази даних.
6. Резервне копіювання та відновлення:
    - Планування автоматичних бекапів.
    - Зручний майстер для відновлення бази даних із резервних копій.

2.4.3. Інші платформи: GitHub та XAMPP

GitHub є однією з найпопулярніших платформ для управління версіями коду, яка дозволяє зберігати, редагувати та відстежувати зміни в проєкті. В основі GitHub лежить система контролю версій Git, яка забезпечує надійний механізм для роботи з кодом у команді, дозволяючи кожному розробнику працювати у своїй гілці (branch) та зливати зміни в основну гілку (merge). Переваги GitHub для проєкту:

- Спільна робота: Розробники можуть легко працювати над спільним кодом, створюючи pull requests для внесення змін.
- Відстеження версій: Збереження історії змін у проєкті дозволяє швидко відновлювати попередні версії коду.
- Інтеграція з іншими інструментами: GitHub легко інтегрується з Visual Studio Code, CI/CD-системами (наприклад, GitHub Actions) і системами управління проєктами (Trello, Jira).
- Хмарне сховище: Забезпечує централізоване зберігання проєкту, доступне з будь-якого місця.

GitHub у даному проєкті використовується для управління кодом серверної частини на Node.js, контролю змін у структурі бази даних.

Рис. 2.6. Логотоп GitHub

XAMPP – це кросплатформний локальний сервер, який дозволяє створювати та тестувати веб-додатки на локальному комп'ютері. До складу XAMPP входять Apache (веб-сервер), MySQL (або MariaDB – база даних), і підтримка мов програмування PHP і Perl. Це популярний інструмент для розробників, які працюють із веб-додатками, базами даних і серверними середовищами. До переваги XAMPP для даного проекту можна віднести:

- Локальне тестування: Дає змогу запускати та тестувати серверну частину без необхідності використання зовнішнього хостингу.
- Легка конфігурація: Простий процес налаштування локального середовища для бази даних MySQL.
- Доступ до MySQL: XAMPP включає в себе MySQL, що спрощує управління базою даних через вбудований інструмент phpMyAdmin.
- Кросплатформність: Підтримує операційні системи Windows, macOS і Linux.

XAMPP у цьому проєкті використовується для локального розгортання бази даних MySQL та тестування інтеграції серверної частини з базою даних до розгортання у продакшн-середовищі.

Рис. 2.7. Логотип Xampp

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЧАТ-БОТА

3.1. Розробка основної логіки чат-бота

Основна логіка чат-бота розроблена з урахуванням потреб користувачів та інтеграції з іншими сервісами. Система чат-боту спрямована на обробку запитів користувачів, взаємодію з базою даних і надсилання повідомлень через API Telegram. Деякі завдання чат-бота включають інформацію та реагування на команди користувача, а також інтерактивні функції для взаємодії з користувачем.

Логіка чат-бота розроблена за допомогою технології Node.js та бібліотеки node-telegram-bot-api, яка дозволяє інтегрувати чат-бота з Telegram. Вона охоплює кілька основних компонентів: файл index.js, маршрути в botRoutes.js, контролери для управління завданнями та іншими функціональностями, а також конфігураційний файл .env. У файлі index.js виконується ініціалізується чат-бот і встановлення необхідного підключення до Telegram API. Додатково в даному файлі налаштовується сервер для прослуховування команд та повідомлень від користувачів.

```
const TelegramBot = require('node-telegram-bot-api');
const token = process.env.BOT_TOKEN;  // Токен, збережений у файлі .env
const bot = new TelegramBot(token, { polling: true });
```

Рис. 3.1. Ініціалізація чат-бота

Також в цьому файлі реєструються маршрути, які будуть обробляти вхідні запити від користувачів через команди, які надсилаються через Telegram.

Файл botRoutes.js виконує функцію реєстрації команд та взаємодію з користувачем через клавіатуру в Telegram. Він містить маршрути для різних команд, таких як "Створити завдання", "Переглянути завдання", "Оновити статус" тощо. Кожен маршрут викликає відповідну функцію в контролері для виконання необхідної операції.

На рис. 3.2 зображено фрагмент коду чат-бот реагує на команду "Створити завдання", змінює стан користувача на "creating_task" і запитує опис завдання.

```
bot.onText(/Створити завдання/, (msg) => {
    const chatId = msg.chat.id;
    userStates[chatId] = 'creating_task'; // Змінюємо стан користувача
    bot.sendMessage(chatId, 'Будь ласка, введіть опис завдання');
});
```

Рис. 3.2. Приклад обробки команди "Створити завдання"

Основний функціонал описани в файлах контролери які основною частиною логіки чат-бота. Вони відповідають за обробку запитів від користувачів, а також за взаємодію з базою даних і зовнішніми сервісами. Контролери поділяються за функціональністю, наприклад:

- taskController.js — відповідає за обробку завдань. Тут розташовані функції для створення, перегляду, оновлення та видалення завдань.
- reminderService.js — відповідає за реалізацію функції нагадувань про дедлайни завдань, відправляючи сповіщення користувачам перед закінченням терміну виконання.

Для зберігання конфігураційних даних є файл .env, в ньому зберігаються такі дані як токени доступу, дані для підключення до бази даних та інші налаштування. Це дозволяє зберігати чутливу інформацію в окремому файлі, щоб уникнути її публікації в репозиторії.

```
TELEGRAM_TOKEN=
MYSQL_HOST=localhost
MYSQL_USER=root
MYSQL_PASSWORD=
MYSQL_DATABASE=chat_db
```

Рис. 3.3. Конфігураційний файл .env

Отже принцип роботи виглядає наступним чином – користувач взаємодіє з чат-ботом, надсилаючи йому повідомлення або натискаючи кнопки, які відображаються на клавіатурі. Бот обробляє ці запити за допомогою вищезгаданих маршрутів, викликає відповідні функції в контролерах, які

взаємодіють з базою даних для виконання необхідних операцій (створення завдань, перегляд завдань, оновлення статусів тощо). Використовуючи дану архітектуру з контролерами та маршрутами, чат-бот легко розширюється новим функціоналом та підтримує гнучку обробку команд від користувачів.

Після запуску чат-бота перед користувачу виводиться головне меню з кнопками, та користувач може почати роботу з системою.

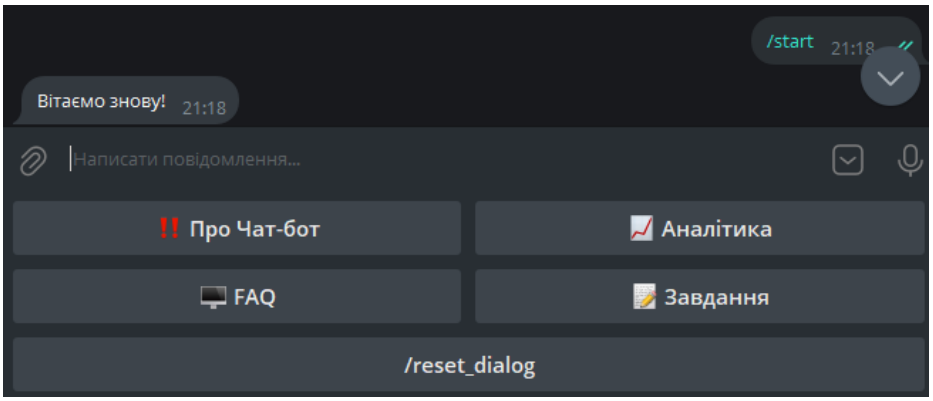

Рис. 3.4. Головне меню чат-боту

3.2. Реалізація функціоналу керування завданнями

Керування завданнями є одним із основних елементів чат-бота, тому що дозволяє користувачам створювати, переглядати, оновлювати та організовувати свої завдання безпосередньо через Telegram. Функції керування завданнями розроблено у вигляді кількох взаємопов'язаних компонентів, що відповідають за обробку команд користувача, роботу з базою даних і надання зручного інтерфейсу для взаємодії.

Основна логіка для керування завданнями реалізована в контролері завдань (taskController.js). Тут знаходяться функції, які обробляють запити на створення, перегляд, оновлення статусу та видалення завдань. Всі ці операції здійснюються шляхом взаємодії з базою даних за допомогою Sequelize.

Для створення завдання через чат-бота, користувач натискає на кнопку створити, після чого користувач вводить боту наступну інформацію: опис завдання, дедлайн, пріоритет і відповідальну особу. Коли користувач вводить ці дані, вони зберігаються до бази даних.

```
exports.createTask = async (msg, bot, description, deadline, priority, assignedTo) => {
    const chatId = msg.chat.id;

    // Перевірка на наявність всіх полів
    if (!description || !deadline || !priority || !assignedTo) {
        return bot.sendMessage(chatId, 'Будь ласка, введіть всі дані для створення завдання.');
    }
    // Форматування дедлайну
    const formattedDeadline = new Date(deadline);
    // Створення нового завдання
    try {
        const task = await Task.create({
            description,
            deadline: formattedDeadline,
            priority,
            assignedTo: chatId
        });
        // Підтвердження створення завдання
        bot.sendMessage(chatId, `Завдання успішно створено!\nОпис: ${description}\nДедлайн:
            ${formattedDeadline}\nПріоритет: ${priority}\nПризначено: ${assignedTo}`);
    } catch (error) {
        bot.sendMessage(chatId, 'Сталася помилка при створенні завдання. Спробуйте ще раз.');
        console.error(error);
    }
};
```

Рис. 3.5. Фрагмент коду для створення завдання

Для перегляду список своїх завдань користувач натискає на кнопку “Перегляд завдань”, після чого бот виводить інформацію про кожне завдання, включаючи опис, статус, дедлайн та пріоритет.

```
exports.viewTasks = async (msg, bot) => { // Перегляд завдань
    const chatId = msg.chat.id;
    console.log(`Chat ID для перевірки: ${chatId}`); // Логування chatId
    const tasks = await Task.findAll({ where: { assignedTo: chatId, status: { [Op.ne]: 'Завершено' } } });
    if (tasks.length === 0) {
        return bot.sendMessage(chatId, 'У вас немає завдань, які не завершено.');
    }
    // Виведення кожного завдання в окремому повідомленні
    tasks.forEach(task => {
        // Форматування дати дедлайну
        const deadline = new Date(task.deadline);
        const formattedDeadline = `${deadline.getDate()} ${deadline.toLocaleString('uk-UA', { month: 'long' })}
        ${deadline.getFullYear()} р. ${deadline.getHours().toString().padStart(2, '0')}:${deadline.getMinutes().toString().padStart(2, '0')}:
        ${deadline.getSeconds().toString().padStart(2, '0')}`;
        const response =
            `Завдання №${task.id}\n` +
            `Опис: ${task.description}\n` +
            `Дедлайн: ${formattedDeadline}\n` +  // Виведення форматованої дати
            `Пріоритет: ${task.priority}\n` +
            `Статус: ${task.status}\n` +
            `Призначено: ${task.assignedTo}\n`;
        // Відправляємо кожне завдання окремо
        bot.sendMessage(chatId, response);
    });
};
```

Рис. 3.6. Фрагмент коду для перегляду завдань

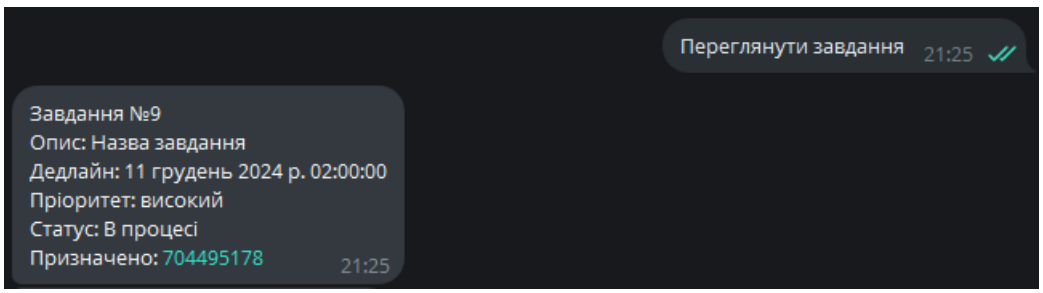

Рис. 3.7. Приклад виведення завдання

Функції оновлення статусу дозволяє користувачу змінювати статус завдання, наприклад, “В процесі”, “Завершено” або “Затримка”. Після вибору завдання натиснувши на кнопку “оновити статус” завдання, бот пропонує вибрати новий статус для цього завдання.

```
exports.updateTaskStatus = async (chatId, taskId, newStatus, bot) => {
    try {
        const task = await Task.findByPk(taskId);
        if (!task) {
            return bot.sendMessage(chatId, `Завдання з ID ${taskId} не знайдено.`);
        }
        // Оновлюємо статус завдання
        task.status = newStatus;
        await task.save(); // Зберігаємо зміни в базі даних
        bot.sendMessage(chatId, `Статус завдання №${taskId} оновлено на: ${newStatus}`);
        // Після оновлення статусу скидаємо стан
        userStatess[chatId] = undefined;
    } catch (error) {
        console.error('Помилка при оновленні статусу завдання:', error);
    }
};
```

Рис. 3.8. Фрагмент коду для оновлення статусу завдання

3.3. Розробка функцій аналітики

Аналітики в чат-боті є важливою складовою для моніторингу виконання завдань та продуктивності команди. Він включає функції для обчислення та генерації звітів щодо виконання завдань, їх статусів, а також для створення візуалізацій на основі зібраних даних. Крім того, до функцій аналітики додано можливість надсилання нагадувань користувачам щодо важливих дедлайнів завдань. Однією з основних функцій аналітики є обчислення продуктивності команди чи користувача на основі середнього часу виконання завдань. Для цього використано модель Task, яка містить інформацію про створення та завершення кожного завдання.

Функція calculatePerformance обчислює середній час виконання завдання для тих, що вже завершені, і повертає результат у хвилинах.

```
exports.calculatePerformance = async () => {
    const tasks = await Task.findAll({
        where: {
            status: 'Завершено',
        }
    });
    if (tasks.length === 0) {
        return 'Немає завершених завдань для обчислення продуктивності.';
    }
    let totalTime = 0;
    tasks.forEach(task => {
        const startTime = new Date(task.createdAt);
        const endTime = new Date(task.updatedAt);
        totalTime += (endTime - startTime); // Час в мілісекундах
    });
    const averageTime = totalTime / tasks.length / 1000 / 60; // Середній час у хвилинах
    return `Середній час виконання завдання: ${averageTime.toFixed(2)} хвилин`;
};
```

Рис. 3.9. Фрагмент коду обчислення продуктивності команди

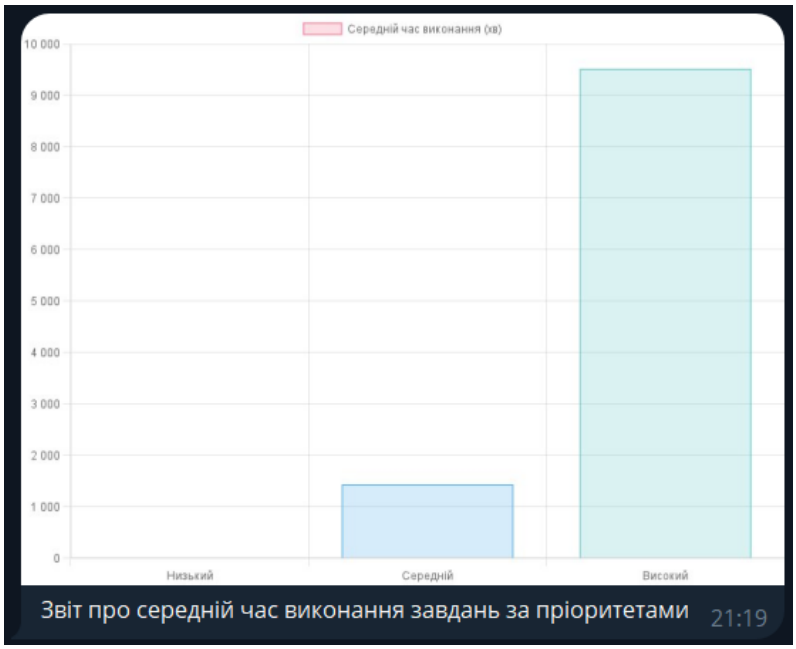

Рис. 3.10. Звіт за середній час виконання завдання

Для збору аналітичних даних до завдань реалізовано наступний функції для генерації звітів по статусах завдань та проблемах. Звіт по статусах завдань підраховує кількість завдань для кожного з трьох основних статусів: "В процесі", "Завершено", "Затримка"

```
exports.generateTaskStatusReport = async () => {
    const taskStatuses = ['В процесі', 'Завершено', 'Затримка'];
    let report = 'Звіт по статусах завдань:\n';
    for (const status of taskStatuses) {
        const count = await Task.count({
            where: {
                status: status,
            },
        });
        report += `${status}: ${count} завдань\n`;
    }
    return report;
};
```

Рис. 3.11. Фрагмент коду для створення звіту по статусах завдань

Звіт по проблемах визначає кількість завдань з проблемами на основі наявності приміток (поле notes), що є індикатором наявності проблем.

```
exports.generateProblemReport = async () => {
    const tasks = await Task.findAll({
        where: {
            notes: { [Op.ne]: null },
        }
    });
    const problemTasks = tasks.filter(task => task.notes && task.notes !== '');
    return `Кількість завдань з проблемами: ${problemTasks.length}`;
};
```

Рис. 3.12. Фрагмент коду для створення звіту по проблемних завданнях

Для створення графіків використовується бібліотека Chart.js в поєднанні з chartjs-node-canvas для Node.js, яка дозволяє генерувати графіки на сервері. Ця бібліотека дозволяє створювати різноманітні типи графіків, такі як стовпчикові , кругові та лінійні , використовуючи дані, отримані з бази даних. Тому заради цієї бібліотеки чат-бот може виводити інформації по завданнях не лише в текстовому вигляді, а також в графічному.

Функціонал аналітики базується на генерації графіків для візуалізації даних. Наприклад, функція generateChart створює графік на основі даних по статусах завдань.

```
exports.generateChart = async (data, chartType) => {
    const width = 400;
    const height = 400;
    const chartJSNodeCanvas = new ChartJSNodeCanvas({ width, height });
    const configuration = {
        type: chartType,
        data: {
            labels: data.labels,
            datasets: [{
                label: 'Статуси завдань',
                data: data.values,
                backgroundColor: ['#FF0000', '#00FF00', '#0000FF'],
                borderColor: ['#FF0000', '#00FF00', '#0000FF'],
                borderWidth: 1,
            }],
        },
        options: {
            responsive: true,
            plugins: {
                legend: {
                    position: 'top',
                },
                tooltip: {
                    callbacks: {
                        label: (tooltipItem) => {
                            return `${tooltipItem.label}: ${tooltipItem.raw} завдань`;
                        }
                    }
                }
            },
            scales: {
                y: {
                    beginAtZero: true,
                }
            }
        }
    };
```

Рис. 3.13 Фрагмент коду для генерації графіків

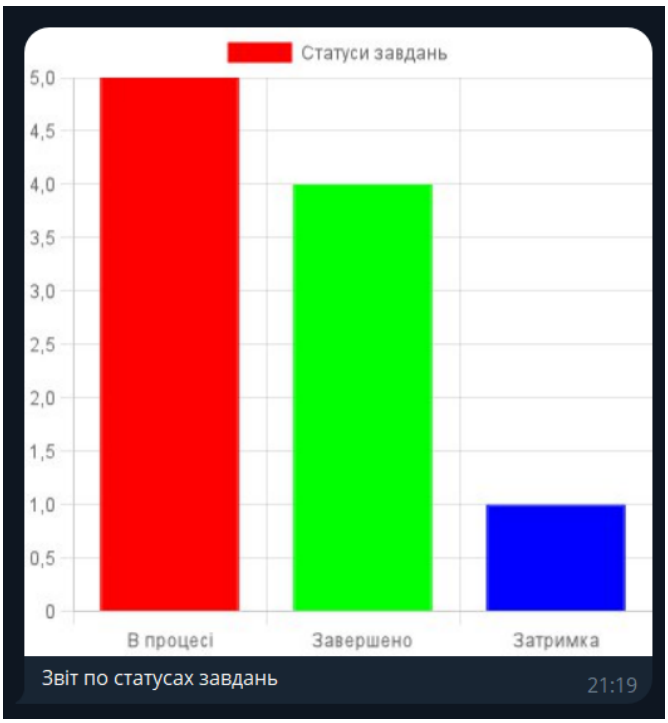

Рис. 3.14. Приклад згенерованого графіку

Для відслідковування вчасного виконання завдань реалізовано функцію нагадування, яка є важливою частиною аналітики, оскільки вона дозволяє

користувачам отримувати сповіщення про наближення дедлайнів або запізнення виконання завдань. Для створення нагадування використано бібліотеку cron, яка дозволяє налаштувати регулярні завдання для надсилання нагадувань. налаштувати регулярні завдання для надсилання нагадувань.

```
cron.schedule('0 9 * * *', async () => {
    console.log('Генерація звітів...');

    // Генерація продуктивності
    const performanceReport = await exports.calculatePerformance();
    console.log(performanceReport);

    // Генерація звіту по статусах
    const statusReport = await exports.generateTaskStatusReport();
    console.log(statusReport);

    // Генерація звіту по проблемах
    const problemReport = await exports.generateProblemReport();
    console.log(problemReport);
});
```

Рис. 3.15. Фрагмент коду для нагадування

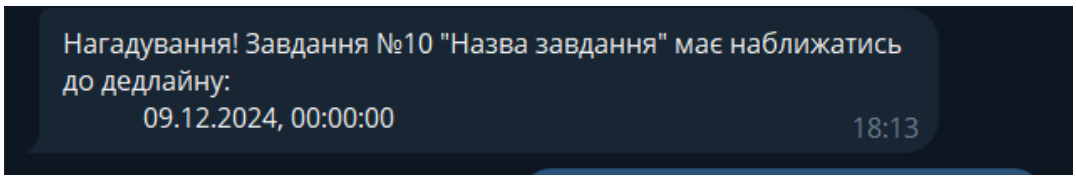

Рис. 3.16. Приклад нагадування про наближення

Цей планувальник генерує звіти кожного дня о 9:00. Отже, для користувачів будуть автоматично надсилатися актуальні дані про виконання завдань, їх статуси, а також кількість завдань з проблемами.

3.4. Реалізація функціоналу для інтеграції (API)

Для інтеграції чат-бота з іншими сервісами та платформами реалізовано API за допомогою бібліотеки Express.js для Node.js, що дозволяє створювати сервери та обробляти HTTP запити. В основі інтеграції лежить сервер на Express.js. Він обробляє запити, що надходять від клієнта, і надає доступ до даних завдань, що зберігаються в базі даних.

```
const express = require('express');
const app = express();
const port = 3000;
app.use(express.json());
```

Рис. 3.17. Створення сервера за допомогою Express

Для зручності використано Swagger для автоматичного створення документації до API. Це дозволяє легко переглядати доступні маршрути, параметри та відповіді API.

```
const swaggerUi = require('swagger-ui-express');
const YAML = require('yamljs');
const swaggerDocument = YAML.load('./swagger.yaml');
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```

Рис. 3.18. Підключення Swagger

Завдяки Swagger, користувачі можуть зайти за посиланням http://localhost:3000/api-docs та переглянути повну документацію API, що допомагає при тестуванні та інтеграції з іншими системами.

Для отримання всіх завдання використано метод GET, даний маршрут дозволяє отримати список всіх завдань із бази даних, який використовує Sequelize для взаємодії з базою даних.

```
app.get('/tasks', async (req, res) => {
    try {
        const tasks = await Task.findAll();
        res.json(tasks);
    } catch (error) {
        res.status(500).send('Помилка при отриманні завдань');
    }
});
```

Рис. 3.18. Фрагмент коду для отримання всіх завдань

Метод POST для створення нових завдання. Цей маршрут дозволяє користувачеві створити нове завдання. Дані завдання, такі як опис, дедлайн, пріоритет, передаються через тіло запиту.

```
app.post('/tasks', async (req, res) => {
    const { description, deadline, priority } = req.body;

    if (!description || !deadline || !priority) {
        return res.status(400).send('Будь ласка, надайте всі дані для створення завдання');
    }
    try {
        const task = await Task.create({
            description,
            deadline,
            priority,
            status: 'В процесі',
        });
        res.status(201).json(task);
    } catch (error) {
        res.status(500).send('Помилка при створенні завдання');
    }
});
```

Рис. 3.18. Фрагмент коду створення нового завдання

Щоб отримати завдання конкретне завдання по його ID використано метод GET. Якщо завдання з таким ID не знайдено, сервер повертає відповідь 404.

```
app.get('/tasks/:id', async (req, res) => {
    const taskId = req.params.id;
    try {
        const task = await Task.findByPk(taskId);

        if (!task) {
            return res.status(404).send('Завдання не знайдено');
        }
        res.json(task);
    } catch (error) {
        res.status(500).send('Помилка при отриманні завдання');
    }
});
```

Рис. 3.19. Фрагмент коду для отримання за його ID

Response body


```
{
  "id": 16,
  "description": "Налаштування БФП",
  "deadline": "2024-12-18T00:00:00.000Z",
  "priority": "середній",
  "status": "В процесі",
  "assignedTo": "1914394021",
  "createdAt": "2024-12-17T19:35:59.000Z",
  "updatedAt": "2024-12-17T19:35:59.000Z"
}
```


Рис. 3.19. API запит на отримання завдання по ID

Розроблене API дозволяє інтегрувати чат-бот з іншими платформами та сервісами, а також забезпечує функціональність для керування завданнями, таких як створення, оновлення, перегляд і видалення завдань. Завдяки інтеграції Swagger документація забезпечує зручний інтерфейс для розробників для тестування та інтеграції з іншими системами.

ВИСНОВКИ

У ході виконання магістерської роботи на тему «Інтелектуальна система моніторингу та управління виконанням завдань» досягнуто основної мети дослідження – створено інформаційну систему для автоматизації управління завданнями, який забезпечує ефективний моніторинг, аналіз і обробку інформації. Розроблена система поєднує сучасні технології Node.js, MySQL та Telegram API, що дозволило забезпечити гнучкість, продуктивність і зручність використання для кінцевих користувачів.

Після проведеного аналізу було визначено, що існуючі інформаційні системи не в повній мірі відповідають вимогам автоматизації процесів управління завданнями через обмежену інтеграцію та відсутність засобів аналітики. Запропонована система усуває ці недоліки завдяки інтерактивному Telegram-боту, який дозволяє користувачам у зручному форматі реєструвати та контролювати завдання, а адміністраторам – отримувати аналітичні дані для оцінки ефективності виконання робіт.

До створюваної системи керування чат-ботом поставлені чіткі функціональні вимоги, які були повністю реалізовані за результатами дипломної роботи:

1. Проаналізовано сучасні інструменти та технології для управління завданнями, визначено їх переваги та недоліки.
2. Розроблено архітектуру системи, що включає серверну частину на базі Node.js, базу даних MySQL та інтерфейс взаємодії через Telegram API.
3. Створено функціональний Telegram-бот, що підтримує інтерактивну роботу з користувачами та надає можливість автоматизованого моніторингу виконання завдань та аналітику.
4. Реалізовано моделі для обробки інформації та аналізу даних, що дозволяє отримувати статистику та аналітику у реальному часі.

Практична цінність роботи полягає у створенні інструменту, що може бути адаптований для використання в різних організаціях, забезпечуючи автоматизацію рутинних завдань і підвищення продуктивності праці.

Запропонована система є прикладом ефективного використання сучасних технологій для оптимізації робочих процесів, а також покзує перспективи подальшого розвитку в напрямі інтеграції з іншими інформаційними системами та розширення функціоналу.

Таким чином, результати магістерської роботи підтверджують актуальність дослідження, практичну значущість запропонованої системи та можливість її застосування для покращення процесів управління завданнями в інформаційно-обчислювальному центрі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot API [Інтернет-ресурс]. – Режим доступу: https://core.telegram.org/bots/api вільний.
2. Eleni Adamopoulou. Chatbots: History, technology, and applications / Eleni Adamopoulou, Lefteris Moussiades // Machine Learning with Applications. – 2020.
3. Ajay Sudhir Bale. Chatbots: Cross-Domain Engineering Applications / Ajay Sudhir Bale, Subhashish Tiwari, B. C. Hemapriya, Baby Chithra // Turkish Online Journal of Qualitative Inquiry. – 2021. – №12(6). – 8416 p.
4. Guendalina Caldarini. A Literature Survey of Recent Advances in Chatbots / Guendalina Caldarini, Sardar Jaf, Kenneth McGarry // Natural Language Interface for Smart Systems. – 2022. – №13(1). – pp 7-11.
5. How do Chatbots Work? A Guide to Chatbot Architecture [Електронний ресурс]. – Режим доступу: https://marutitech.com/chatbots-work-guide-chatbot-architecture/.
6. Adnan Rehan. 10 Best Chatbot Development Frameworks to Build Powerful Bots [Електронний ресурс]. – 2020. – Режим доступу: https://geekflare.com/chatbot-development-frameworks/.
7. Rakesh Patel. 10 Best AI Chatbot Framework Comparison [Електронний ресурс]. – 2021. – Режим доступу: https://www.spaceo.ca/blog/top-ai-chatbot-frameworks/.
8. Микола Петров. Використання Telegram-ботів для автоматизації задач бізнес-процесів [Електронний ресурс] / М. Петров. – 2021. – Режим доступу: https://itc.ua/articles/telegram-bot/.
9. Іван Кравчук. Чат-боти в системах управління завданнями: перспективи та можливості / І. Кравчук // Інформаційні технології та системи. – 2022. – №2. – С. 45-49.
10. Сергій Мельник. Аналіз технологій розробки чат-ботів для інтеграції у бізнес-процеси [Електронний ресурс] / С. Мельник. – 2021. – Режим доступу: https://www.bizinform.com.ua/articles/chatbot-analysis.

11. Олег Зубко. Використання Node.js у розробці інтегрованих інформаційних систем [Електронний ресурс] / О. Зубко. – 2022. – Режим доступу: https://dev.ua/articles/nodejs-integration.
12. Офіційна документація Node.js [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/.
13. Офіційна документація MySQL [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/.

ДОДАТОК А

Конфігураційний файл package.json проєкту

```
{
  "name": "chat_bot",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "canvas": "^3.0.0-rc3",
    "chart.js": "^3.9.1",
    "chartjs-node-canvas": "^4.1.6",
    "dotenv": "^16.4.5",
    "express": "^4.21.2",
    "mysql2": "^3.11.4",
    "node-cron": "^3.0.3",
    "node-telegram-bot-api": "^0.66.0",
    "sequelize": "^6.37.5",
    "swagger-jsdoc": "^6.2.8",
    "swagger-ui-express": "^5.0.1",
    "telegraf": "^4.16.3",
    "yamljs": "^0.3.0"
  }
}
```