Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка модуля обліку робочого часу для CRM-системи»

Виконав: студент групи К23-2М

Спеціальність 122 Комп'ютерні науки

Гетьман О.Г.
(прізвище та ініціали)

Керівник д.т.н., проф. Яковенко В.О.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Дніпровський державний технічний університет
(місце роботи)

Доцент кафедри математичного моделювання та системного аналізу
(посада)

к.т.н., доц. Волосова Н.М.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Гетьман О.Г. Розробка модуля обліку робочого часу для CRM-системи.

Дипломна робота на здобуття освітнього ступеня магістра за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єктом дослідження є процес обліку робочого часу в організаціях.

Предмет дослідження – розробка програмного модуля обліку робочого часу для CRM-системи.

Метою роботи є створення модуля обліку робочого часу, який забезпечує автоматизацію процесів управління робочим часом та інтеграцію з іншими компонентами CRM-системи.

Дана робота присвячена розробці модуля обліку робочого часу для CRM-системи. У процесі виконання роботи проведено аналіз сучасних систем обліку робочого часу, визначено їх переваги та недоліки, а також обґрунтовано вибір технологій для реалізації. Основу модуля складає мікросервісна архітектура, яка забезпечує гнучкість, масштабованість та простоту інтеграції.

Розроблений модуль реалізує ключові функції, такі як автоматизований облік часу, генерація звітів, налаштування робочих графіків і забезпечення контролю трудової дисципліни. У клієнтській частині застосовано бібліотеку React.js, у серверній – платформу ASP.NET Core. Проведено тестування модуля, яке підтвердило його відповідність функціональним і нефункціональним вимогам.

Практична цінність роботи полягає у створенні універсального рішення, яке може бути використане в компаніях для оптимізації бізнес-процесів, підвищення продуктивності працівників та прозорості в управлінні робочим часом.

Ключові слова: CRM-система, облік робочого часу, мікросервісна архітектура, React.js, ASP.NET Core.

ABSTRACT

Hetman O.H. Development of a working time tracking module for a CRM system.

Diploma thesis (project) for obtaining a master's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2025.

The object of research is the process of working time tracking in organizations.

The subject of research is the development of a working time tracking module for a CRM system.

The aim of the study is to create a working time tracking module that automates time management processes and integrates with other components of the CRM system.

This thesis is devoted to the development of a working time tracking module for a CRM system. During the research, an analysis of modern time tracking systems was carried out, their advantages and disadvantages were identified, and the choice of technologies for implementation was justified. The module is based on a microservices architecture that ensures flexibility, scalability, and ease of integration.

The developed module implements key features such as automated time tracking, report generation, work schedule customization, and enforcement of labor discipline. The client side is implemented using React.js, while the server side uses the ASP.NET Core platform. Testing of the module confirmed its compliance with functional and non-functional requirements.

The practical value of the work lies in the creation of a universal solution that can be used by organizations to optimize business processes, improve employee productivity, and enhance transparency in time management.

Keywords: CRM system, time tracking, microservices architecture, automation, React.js, ASP.NET Core.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах стрімкого розвитку інформаційних технологій автоматизація бізнес-процесів є важливою складовою успішної діяльності компаній. CRM-системи відіграють ключову роль у забезпеченні ефективного управління взаємодією з клієнтами, автоматизації робочих процесів та аналізу даних. Одним із важливих компонентів таких систем є модуль обліку робочого часу, який сприяє підвищенню продуктивності, забезпеченню прозорості процесів і покращенню управління людськими ресурсами.

Інноваційність даної роботи полягає у створенні модуля обліку робочого часу для CRM-системи, який інтегрується з іншими функціональними модулями, забезпечує автоматизацію процесів та відповідає сучасним технологічним стандартам. Використання мікросервісної архітектури дозволяє досягти високої гнучкості, масштабованості та ефективності системи.

Мета роботи – розробка модуля обліку робочого часу для CRM-системи.

Методи дослідження – методи проектування, розробки програмного забезпечення, метод теорії інформації, обробка та аналіз інформації.

Завдання роботи:

1. Провести аналіз існуючих рішень та підходів до обліку робочого часу.
2. Визначити вимоги до програмного забезпечення та обґрунтувати вибір технологічного стеку.
3. Розробити архітектуру модуля обліку робочого часу.
4. Реалізувати модуль із використанням сучасних інструментів програмування.
5. Перевірити відповідність модуля функціональним та нефункціональним вимогам.

Об'єкт дослідження – процеси обліку робочого часу у CRM-системах.

Предмет дослідження – програмний модуль для автоматизації обліку робочого часу у CRM-системі.

Практичне значення результатів роботи полягає у створенні ефективного інструменту, який може бути використаний у компаніях для автоматизації процесів управління робочим часом, підвищення дисципліни та оптимізації трудових ресурсів.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Обсяг роботи становить 60 сторінок основного тексту, 38 рисунків та 1 таблиці.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз публікацій щодо розробки CRM-систем

CRM або управління відносинами з клієнтами - це не просто модне словосполучення, а ціла філософія ведення бізнесу, спрямована на побудову міцних та взаємовигідних стосунків з кожним клієнтом. Ця філософія реалізується через комплекс стратегій та технологій, що дозволяють компаніям ефективно взаємодіяти з покупцями на всіх етапах: від першого знайомства до післяпродажного обслуговування. [1] Серцем CRM є збір, зберігання та скрупульозний аналіз інформації не тільки про клієнтів, а й про постачальників, партнерів та внутрішні бізнес-процеси.

Найбільш затребувані CRM-рішення у сферах, де панує висока конкуренція і кожен клієнт на вагу золота. Це, перш за все, фінансовий сектор (банки, страхові компанії), телекомунікації, оптова та роздрібна торгівля, дистрибуція, а також сфера високих технологій. Крім того, CRM успішно застосовують компанії, що спеціалізуються на сервісному обслуговуванні (наприклад, ремонт побутової техніки чи автомобілів), рекламні агентства, фармацевтичні компанії, виробники та постачальники комп'ютерного обладнання та програмного забезпечення, а також провайдери послуг зв'язку, туристичні фірми та транспортні компанії.

Подібно до досвідченого диригента, CRM-система керує численними інформаційними потоками, швидко обробляючи їх та дозволяючи компанії миттєво реагувати на найменші коливання ринку. Принципи роботи CRM-системи можна зобразити схематично (див. рис. 1.1).

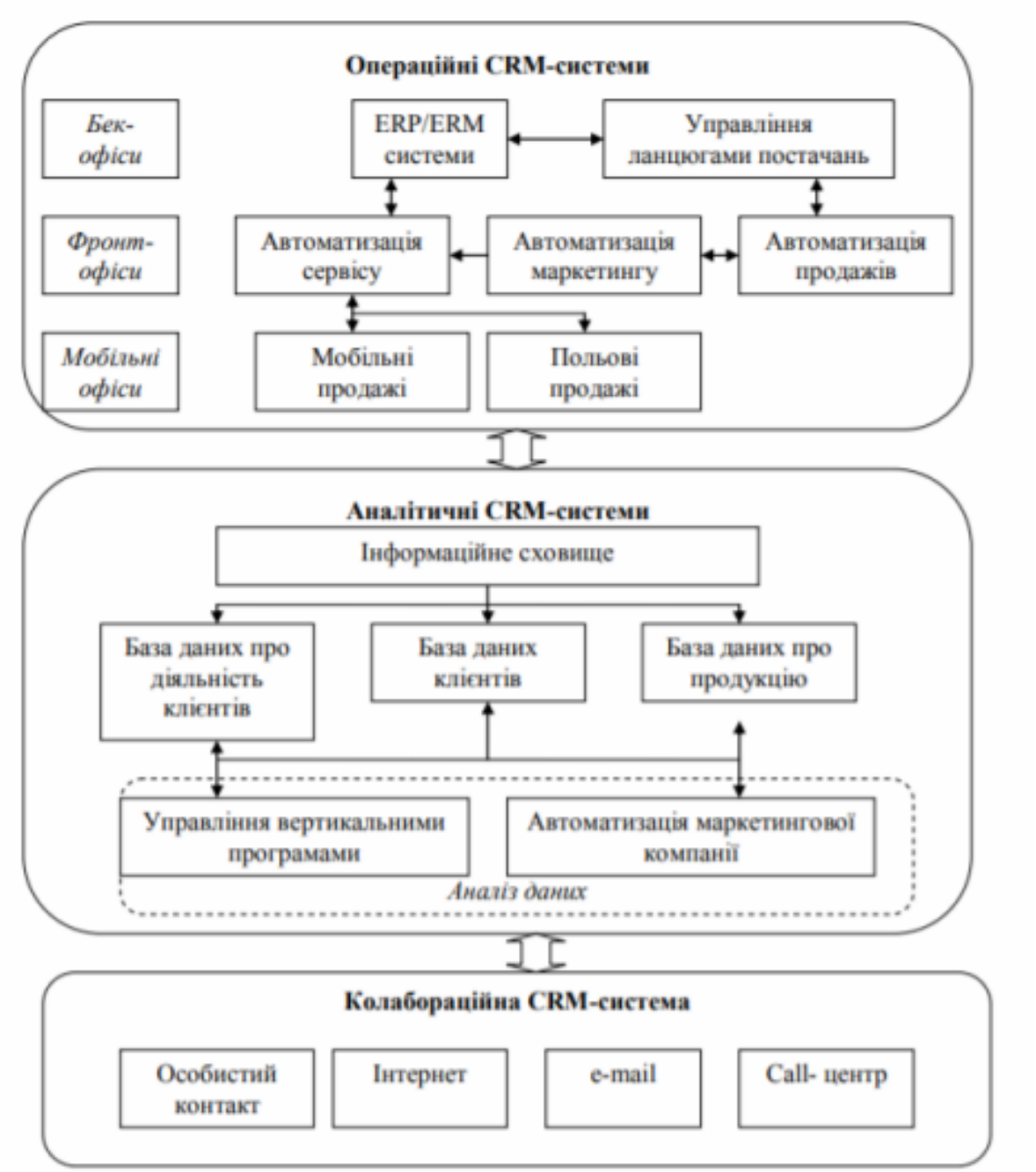

Рисунок 1.1 – Організація і функціонування CRM-систем

Залежно від рівня обробки інформації та розв'язуваних завдань, CRM-системи поділяються на: операційні, аналітичні та колабораційні [2].

- Операційні CRM-системи полегшують взаємодію з клієнтами, систематизують дані про заявки та угоди, виставляють рахунки, нагадують про необхідність дзвінка, можуть автоматично надсилати SMS-повідомлення,

записують телефонні розмови тощо. Їх головна мета - підвищення лояльності клієнта під час безпосереднього контакту.

- Аналітичні CRM-системи об'єднують розрізнені масиви даних та проводять їх спільний аналіз для розробки ефективних стратегій маркетингу, продажів та обслуговування клієнтів. Вони потребують значного обсягу накопичених статистичних даних. Спочатку більшість CRM-систем були операційними, але сучасні рішення все частіше поєднують функції всіх трьох типів.
- Колаборaційні CRM-системи (CRM взаємодії) забезпечують взаємодію компанії з клієнтами через електронну пошту, чати, форуми, call-центри та ін. Це дозволяє клієнтам впливати на розробку продукту, виробництво, сервісне обслуговування, а також висловлювати свої пропозиції та зауваження. Сучасні колаборaційні CRM базуються на інтернет-технологіях (e-CRM), інтегруючись з системами електронної комерції та іншими додатками, що підтримують роботу з клієнтами через Інтернет. Наприклад, e-CRM дозволяє приймати замовлення на сайті, відстежувати доставку, розсилати маркетингові матеріали електронною поштою.

На українському ринку представлені такі CRM-системи, як Microsoft CRM, 1С: Управління торгівлею 8.0, Siebel, Oracle CRM, E-Business Suite, Terrasoft CRM, WinPeak CRM, Парус-Менеджмент і Маркетинг, Облік CRM, Sales Expert.

Багато українських підприємств вже використовують CRM-системи. Лідерами ринку є Бітрікс24 (26% компаній), Terrasoft (15%) та AmoCRM (9%). Далі в рейтингу йдуть Salesforce, Zoho та OneBox [3].

Значна частина компаній є посередниками для вендорів всесвітньо відомих CRM-систем, таких як SAP SRM (SAP SE, Німеччина), Microsoft Dynamics CRM (Microsoft Corporation, США), Bitrix24.CRM (Bitrix Inc., США), Oracle CRM on demand та Oracle Siebel CRM (Oracle Corporation, США) [4].

1.2 Аналіз систем обліку робочого часу

Облік робочого часу являє собою структурований процес контролю за діяльністю працівників, який дозволяє оцінити відповідність їхньої роботи встановленому графіку в межах організації. Головною метою такого моніторингу є забезпечення трудової дисципліни, яка є важливим чинником ефективної роботи підприємства. Відсутність чіткої системи обліку може призводити до зловживання довірою роботодавця та зниження загальної продуктивності.

Методи обліку робочого часу є різноманітними, що дозволяє організаціям адаптувати їх до власних потреб та специфіки діяльності:

1. Реєстрація часу прибуття та залишення роботи відповідальною особою. Цей підхід передбачає ручний запис часу охоронцем або іншим уповноваженим співробітником у спеціальний журнал. Такий метод є простим у впровадженні, проте вимагає регулярного аналізу даних для отримання корисної інформації.

2. Спостереження за робочим процесом спеціально призначеним працівником. Даний метод передбачає наявність співробітника, який безпосередньо контролює діяльність інших у робочому приміщенні. Це дозволяє оперативно реагувати на відхилення від трудової дисципліни, зменшуючи кількість несанкціонованих перерв.

3. Самозвітність працівників. Такий підхід базується на самостійному внесенні працівниками даних про виконану роботу в звіти, які потім аналізуються керівництвом. Крім фіксації часу, цей метод сприяє підвищенню відповідальності співробітників, стимулюючи їх до виконання завдань у встановлені терміни.

4. Контрольований доступ до робочих місць. Використання електронних карток або біометричної ідентифікації дозволяє автоматизувати процес реєстрації присутності працівників. Ця технологія забезпечує точність даних і спрощує процес аналізу, зводячи людський фактор до мінімуму.

5. Відеоспостереження. Це один із найбільш надійних методів контролю, який дозволяє не лише фіксувати присутність, але й аналізувати поведінку працівників на робочому місці. Попри ефективність, цей метод є затратним і може викликати психологічний дискомфорт, що обмежує його застосування до критичних ділянок, таких як склади або магазини.

6. Спеціалізоване програмне забезпечення. Даний інструмент дозволяє фіксувати всі дії співробітника, виконані на робочому місці, що особливо актуально для офісних працівників. Таке ПЗ автоматично генерує звіти, аналізуючи не лише час роботи, а й продуктивність.

Для порівняння та виявлення сильних і слабких сторін існуючих систем обліку робочого часу було проаналізовано кілька популярних рішень із подібним функціоналом. Опис систем та результати їх аналізу наведено нижче.

Toggl Track є популярним інструментом для обліку робочого часу, що дозволяє відстежувати витрати часу на виконання завдань і проєктів. Заснована у 2006 році, система вже понад 18 років допомагає мільйонам користувачів у всьому світі. Вона розпочала свою діяльність як стартап в Естонії й нині обслуговує понад 5 мільйонів користувачів, серед яких фрилансери, консультанти та працівники з погодинною оплатою. Завдяки інтуїтивному інтерфейсу та інтеграціям з популярними інструментами Toggl Track є одним із лідерів у цій галузі [11].

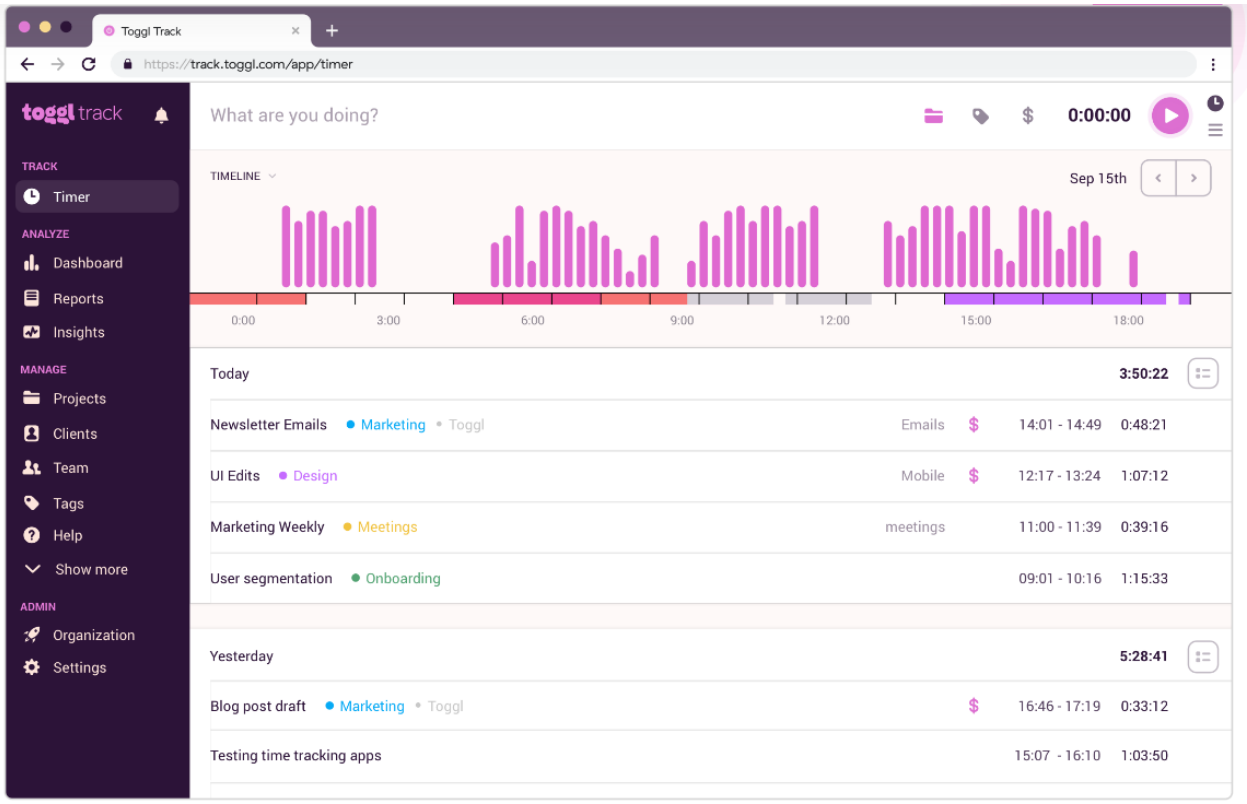

Рисунок 1.1 – Toggl Track

Застосунок пропонує широкий спектр функцій, які задовольняють потреби як індивідуальних користувачів, так і команд. Основні можливості системи включають:

1. Облік робочого часу:

- Система дозволяє користувачам відстежувати час вручну або автоматично за допомогою вбудованого таймера.
- Є можливість класифікувати витрачений час за проєктами, завданнями або клієнтами.

2. Генерація звітів:

- Toggl Track надає інструменти для створення звітів, які можна сортувати за різними критеріями, включаючи проєкти, завдання чи часові рамки.
- Дані можна експортувати у форматах PDF, Excel або CSV для подальшої обробки.

3. Інтеграція з іншими платформами:

- Система підтримує інтеграції з такими популярними інструментами, як Asana, Slack, Trello, Jira, і багатьма іншими.
- Це забезпечує автоматичний облік часу, синхронізований із завданнями, створеними на цих платформах.

4. Мобільна та настільна доступність:

- Toggl Track доступний на мобільних пристроях (Android, iOS), настільних комп'ютерах (Windows, Mac, Linux), а також у вигляді веб-додатка.

5. Аналітика та планування:

- Користувачі можуть отримувати аналітичні дані про розподіл часу, що допомагає ідентифікувати неефективні процеси.
- Є можливість встановлювати бюджети для проєктів і відстежувати їхнє виконання.

Переваги використання Toggl Track:

1. Простота використання:

- Інтерфейс системи створений таким чином, щоб бути інтуїтивно зрозумілим навіть для новачків. Відсутність складних налаштувань дозволяє швидко почати роботу.

2. Гнучкість інтеграцій:

- Завдяки підтримці інтеграцій із багатьма популярними інструментами для управління проєктами та комунікації, Toggl Track дозволяє спростити процес обліку часу.

3. Деталізовані звіти:

- Звіти, створені у Toggl Track, дозволяють аналізувати, як витрачається час, що є важливим для прийняття управлінських рішень.

4. Мультиплатформність:

- Доступність системи на різних платформах забезпечує зручність використання незалежно від місця перебування користувача.

5. Гнучка модель ціноутворення:

- Toggl Track пропонує безкоштовний тарифний план із базовим функціоналом, що робить його доступним для індивідуальних користувачів.

Попри значні переваги, Toggl Track має і певні недоліки, які слід враховувати:

1. Обмежений функціонал для фінансового обліку:

- Система орієнтована переважно на облік часу, а її можливості у сфері фінансового обліку є досить обмеженими. Це може бути проблемою для користувачів, які потребують повноцінного інструменту для управління фінансами.

2. Висока вартість розширених функцій:

- Платна версія Toggl Track із доступом до додаткових функцій, таких як розширені звіти та інтеграції, може бути дорогою для індивідуальних користувачів або малих команд.

3. Відсутність підтримки розширеного налаштування інтерфейсу:

- Інтерфейс системи, хоч і зручний, не дозволяє користувачам адаптувати його під власні потреби, що може створювати певні обмеження.

Практичне застосування Toggl Track Toggl Track є ідеальним рішенням для професіоналів, яким важливо відстежувати робочий час для підвищення продуктивності або звітування перед клієнтами. Фрилансери та консалтингові компанії можуть використовувати систему для обчислення погодинної оплати та створення детальних звітів про виконані завдання. Завдяки інтеграціям із проєктними менеджерами, такими як Asana та Trello, Toggl Track підходить також для командної роботи, забезпечуючи синхронізацію завдань і часу, витраченого на їх виконання.

Clockify – це популярний інструмент для обліку робочого часу, який поєднує простоту використання та широкий набір функціональних можливостей. Заснована у 2017 році, ця система швидко здобула популярність і нині налічує понад 4 мільйони користувачів. Clockify є повністю безкоштовним інструментом, що вирізняє його серед конкурентів. Система забезпечує підтримку як індивідуальних користувачів, так і команд, пропонуючи інструменти для відстеження робочого часу, фінансового обліку та управління проєктами. Ця універсальність робить Clockify привабливим

вибором для професіоналів, які шукають доступне рішення для тайм-менеджменту [12].

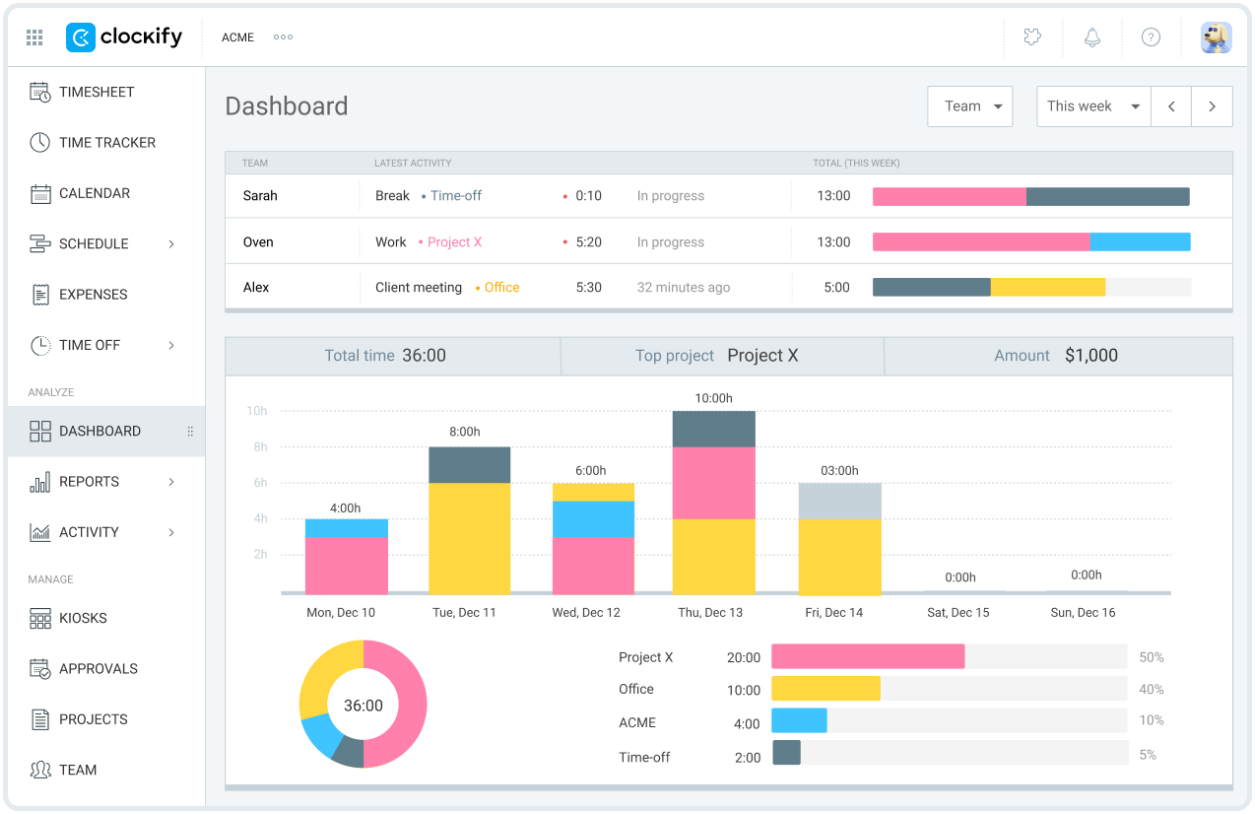

Рисунок 1.2 – Clockify

Функціональні можливості Clockify надає користувачам низку функцій, спрямованих на підвищення ефективності роботи та спрощення обліку часу. Основні можливості системи включають:

1. Облік робочого часу:

- Користувачі можуть відстежувати час вручну або за допомогою автоматичного таймера.
- Система дозволяє організовувати час за проєктами, завданнями та клієнтами, що забезпечує чітку структуру даних.

2. Управління фінансами:

- Clockify підтримує базовий фінансовий облік, включаючи підрахунок заробітків на основі відпрацьованого часу.

- Можливість встановлення погодинних ставок для проєктів або клієнтів.

3. Інтеграція з іншими інструментами:

- Система інтегрується з популярними платформами, такими як Trello, Asana, Slack, Jira, що дозволяє синхронізувати облік часу з управлінням завданнями.

4. Звіти та аналітика:

- Clockify генерує детальні звіти, які можна сортувати за проєктами, завданнями або користувачами.
- Звіти доступні у форматах PDF, Excel і CSV для подальшого аналізу.

5. Платформна доступність:

- Clockify доступний у веб-версії, як мобільний додаток (iOS, Android) і настільна програма (Windows, macOS, Linux).

Переваги використання Clockify:

1. Безкоштовність:

- Основний функціонал системи доступний безкоштовно, що робить її привабливою для індивідуальних користувачів і малих команд.

2. Гнучкість у використанні:

- Завдяки інтеграціям та підтримці багатьох платформ Clockify забезпечує гнучкість у роботі та доступ до системи з будь-якого пристрою.

3. Деталізація обліку:

- Можливість відстежувати час на рівні завдань і проєктів дозволяє отримувати повну картину використання часу.

4. Зручність звітності:

- Інструмент надає користувачам гнучкі інструменти для формування звітів, що спрощує аналіз ефективності роботи.

Попри значні переваги, Clockify має кілька недоліків, які можуть обмежувати його використання в деяких випадках:

1. Базовий фінансовий облік:

- Фінансовий функціонал системи є обмеженим і може не задовольнити потреби користувачів, які потребують комплексного управління фінансами.

2. Обмеження налаштувань інтерфейсу:

- Інтерфейс системи не дозволяє значною мірою налаштовувати робочий простір під індивідуальні потреби користувачів.

3. Відсутність розширеного функціоналу в базовій версії:

- Деякі розширені функції, такі як автоматизоване планування або прогнозування, відсутні, що може знижувати ефективність роботи для великих команд.

Clockify ідеально підходить для професіоналів, які потребують простого і безкоштовного інструменту для відстеження часу. Він є чудовим рішенням для фрилансерів, які працюють на погодинній оплаті, та малих команд, яким необхідно контролювати виконання проєктів. Завдяки своїй інтеграції з іншими популярними платформами, Clockify також може бути корисним для командного управління завданнями.

FreshBooks — це сучасна система управління фінансами, що поєднує функціонал для обліку робочого часу та ведення фінансового обліку. Заснована у 2003 році в Канаді, вона стала вибором понад 30 мільйонів користувачів у більш ніж 160 країнах. Система спочатку була створена як інструмент для виставлення рахунків, але поступово розвинулась у комплексну платформу для малого бізнесу та фрилансерів. FreshBooks пропонує зручні інструменти для управління фінансами, генерування звітів і відстеження робочого часу, що робить її універсальним рішенням для управління проєктами й фінансовими потоками [13].

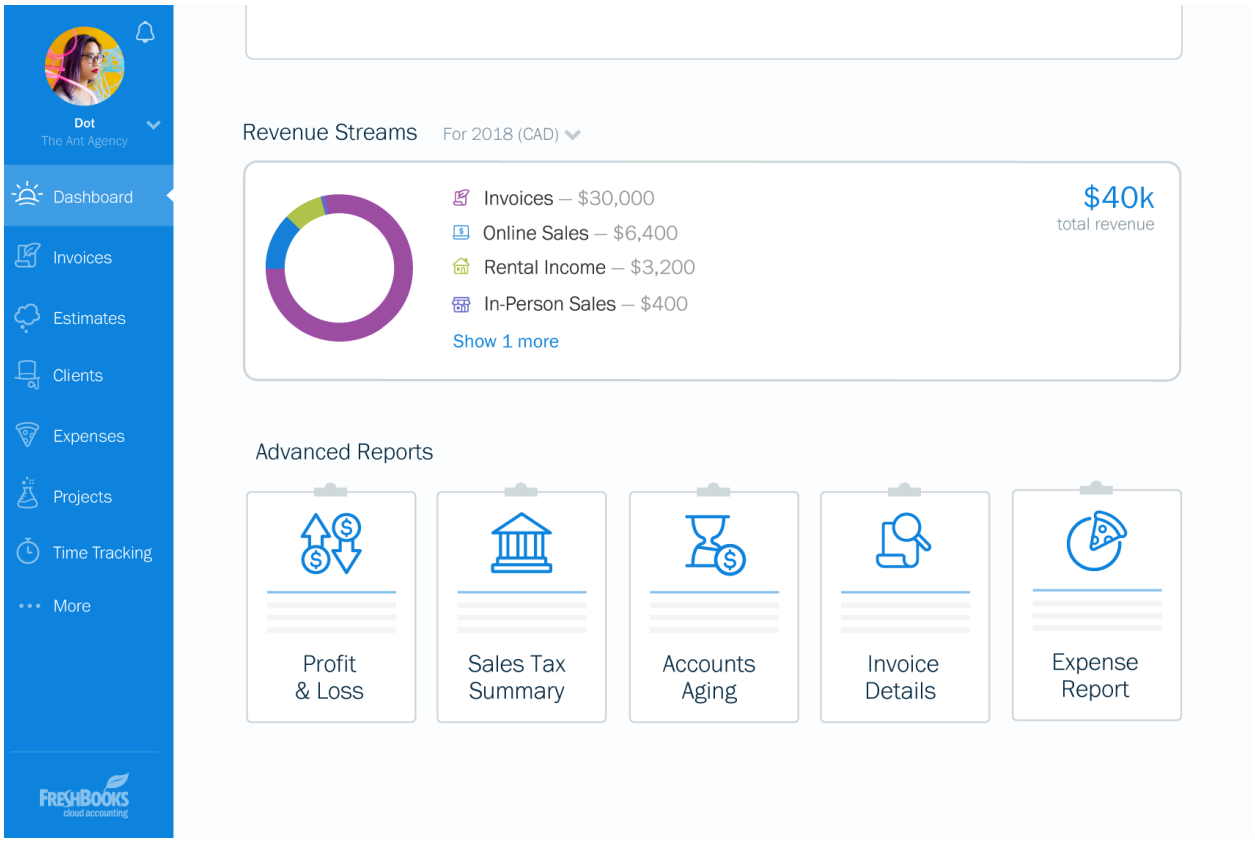

Рисунок 1.3 – FreshBooks

FreshBooks забезпечує користувачів багатофункціональними інструментами для фінансового обліку та управління робочими процесами. Основні можливості системи включають:

1. Управління фінансами:
   - Інструменти для обліку доходів і витрат.
   - Підключення до банківських рахунків для автоматичного завантаження транзакцій.
   - Генерація та відправлення професійних рахунків-фактур клієнтам.
2. Облік робочого часу:
   - Відстеження часу, витраченого на виконання завдань, із можливістю прив’язки до конкретних проєктів.
   - Автоматичне перетворення відпрацьованого часу у рахунки для клієнтів.
3. Звіти та аналітика:

- Детальні звіти про доходи, витрати, касові потоки та продуктивність.
- Можливість налаштування звітів для отримання потрібної інформації.

4. Інтеграція з іншими платформами:
   - Система інтегрується з популярними інструментами, такими як G Suite, Trello, Asana, Stripe, PayPal та іншими.
   - Підтримка синхронізації з бухгалтерським програмним забезпеченням.
5. Мобільна доступність:
   - FreshBooks доступний як мобільний додаток для iOS і Android, що дозволяє користувачам управляти своїми фінансами з будь-якого місця.

Переваги використання:

1. Повнофункціональний фінансовий облік:
   - Система дозволяє вести повний облік доходів, витрат і касових потоків, що є критично важливим для підприємців.
2. Зручний інтерфейс:
   - Інтуїтивно зрозумілий дизайн спрощує використання системи навіть для користувачів без досвіду в бухгалтерії.
3. Автоматизація процесів:
   - Автоматичне створення рахунків, нагадування про платежі та синхронізація з банківськими рахунками значно економлять час користувачів.
4. Можливості масштабування:
   - FreshBooks пропонує різні тарифні плани, що дозволяє користувачам вибирати функціонал залежно від їхніх потреб.

Попри численні переваги, FreshBooks має кілька недоліків, які варто враховувати при виборі системи:

1. Висока вартість:
   - Підписка на FreshBooks може бути занадто дорогою для індивідуальних фахівців або малого бізнесу на початкових етапах.

2. Складність для невеликих проєктів:

- Розширений функціонал може бути надмірним для користувачів, які потребують лише базових функцій для обліку робочого часу.

3. Обмеження інтерфейсу для обліку часу:

- Користувачам, які зосереджуються переважно на тайм-менеджменті, може бути незручно використовувати систему через її орієнтованість на фінансовий облік.

FreshBooks є ідеальним рішенням для малого бізнесу, фрилансерів і консалтингових компаній. Завдяки поєднанню фінансового обліку та управління часом, система дозволяє підприємцям оптимізувати свої робочі процеси, зосереджуючись на основних завданнях. FreshBooks особливо корисний для тих, хто часто працює з клієнтами, потребує створення рахунків та управління фінансовими потоками.

Toggl Track, Clockify та FreshBooks — це три популярні платформи, які пропонують різноманітні можливості для підвищення продуктивності та оптимізації робочих процесів. Кожна з них має унікальні функції, які можуть задовольнити потреби різних типів користувачів, включаючи фрилансерів, малі бізнеси та команди.

Таблиця 1.1 – Порівняльний аналіз систем Toggl Track, Clockify та FreshBook

Характеристика	Toggl Track	Clockify	FreshBooks
Рік заснування	2006	2017	2003
Основний функціонал	Облік робочого часу	Облік часу з базовим фінансовим обліком	Управління фінансами з обліком часу
Цільова аудиторія	Фрилансери, консультанти, невеликі команди	Фрилансери, невеликі команди	Малий бізнес, фрилансери, консультанти
Відстеження часу	Так	Так	Так
Фінансовий облік	Ні	Базовий	Повний

Інтеграції	Asana, Slack, Trello, Jira	Trello, Asana, Slack, Jira	Stripe, PayPal, G Suite, Trello
Платформи	Веб, ПК (Windows, Mac, Linux), мобільні (iOS, Android)	Веб, ПК (Windows, Mac, Linux), мобільні (iOS, Android)	Веб, мобільні (iOS, Android)
Модель ціноутворення	Безкоштовний базовий план, платні розширені функції	Безкоштовний для необмеженої кількості користувачів, платні плани	Платні плани з декількома рівнями
Переваги	Інтуїтивний інтерфейс, детальні звіти, сильні інтеграції	Безкоштовний доступ, гнучкі інтеграції	Повний фінансовий функціонал, автоматизація рахунків
Недоліки	Обмежений фінансовий функціонал, висока ціна платної версії	Обмежений фінансовий облік, базові налаштування інтерфейсу	Висока вартість, складність для базового обліку часу

Сучасна система обліку робочого часу є багатофункціональним інструментом, що виходить за межі простого моніторингу присутності. Вона забезпечує комплексний контроль за місцезнаходженням співробітників, дозволяє виявляти прогалини у використанні робочого часу та формувати звіти, які сприяють оптимізації робочих процесів. Завдяки впровадженню таких систем організації можуть підвищувати рівень дисципліни, продуктивності та прозорості роботи.

1.3 Висновок до першого розділу

У даному розділі було проведено дослідження методів обліку робочого часу, а також аналіз сучасних рішень для інтеграції цих функцій у CRM-системи. Було проаналізовано основні підходи до обліку робочого часу, такі як ручний, автоматизований і комбінований. Визначено, що кожен із цих

методів має свої переваги та недоліки, залежно від специфіки бізнес-процесів і потреб організації.

Особливу увагу приділено існуючим інструментам і платформам, що забезпечують функціонал обліку робочого часу, включаючи Toggl Track, Clockify і FreshBooks. Розглянуто переваги цих рішень, зокрема автоматизацію процесів, зручність для користувачів та можливість інтеграції з іншими системами, а також їхні обмеження, серед яких відсутність універсального функціоналу для специфічних завдань.

Метою роботи є розробка модуля обліку робочого часу для CRM-системи, який автоматизує процеси управління робочим часом та забезпечує інтеграцію з іншими компонентами системи.

Для досягнення поставленої мети у кваліфікаційній роботі було визначено та виконано такі завдання:

1. Проведено аналіз сучасних інструментів і технологій, що використовуються для обліку робочого часу і визначено їхні ключові характеристики.
2. Сформульовано вимоги до програмного забезпечення, включаючи функціональні і нефункціональні аспекти.
3. Обґрунтовано вибір мікросервісної архітектури, яка забезпечує незалежне масштабування компонентів та гнучкість у виборі технологій майбутньої системи.
4. Визначено технологічний стек для реалізації модуля.
5. Реалізовано модуль обліку робочого часу для CRM-системи
6. Проведено тестування модулю.

Вхідними даними для модуля є робочі графіки, які вводяться користувачем через інтерфейс. Ці дані обробляються серверною частиною та зберігаються у форматі JSON, що забезпечує простоту інтеграції з іншими модулями CRM-системи.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ МОДУЛЮ ОБЛІКУ РОБОЧОГО ЧАСУ

2.1 Вимоги до системи

Система модуль обліку робочого часу є важливим інструментом для автоматизації управління кадровими ресурсами, забезпечуючи точність обліку, зручність у плануванні та оптимізацію трудових процесів. Дозволяє мінімізувати ручну працю, скоротити витрати часу на адміністрування та підвищити продуктивність завдяки інтеграції сучасних технологій. Були сформовані ключові функціональні, нефункціональні та технічні вимоги до системи, спрямовані на забезпечення її надійності, ефективності та відповідності актуальним стандартам програмного забезпечення.

Функціональні вимоги:

1. Система авторизації:
   - Реалізація механізму авторизації користувачів через введення ідентифікатора та пароля.
   - Відображення інформаційних повідомлень про помилки у разі неправильного введення облікових даних.
2. Управління робочими графіками:
   - Надання можливості пошуку робочих графіків працівників за критеріями, що включають табельний номер, підрозділ, посаду та часовий період.
   - Реалізація функції редагування робочих графіків із можливістю внесення змін до тривалості робочого дня, додавання днів відпусток та лікарняних.
   - Впровадження інструменту автоматизованого табулювання для оброблення даних за певними параметрами.
3. Управління режимами роботи:

- Розробка функціоналу для створення та налаштування робочих режимів із зазначенням робочих та вихідних днів.
- Забезпечення підтримки циклічності режимів та обліку специфічних умов роботи у святкові дні.

4. Облік та формування звітів:

- Генерація аналітичних звітів за певний період із деталізацією параметрів робочого часу працівників.
- Надання можливості одночасного відображення та аналізу декількох графіків.

Нефункціональні вимоги:

1. Продуктивність системи
2. Безпека даних:
3. Сумісність:
4. Масштабованість

2.2 Вибір програмних засобів для реалізації системи

Розробка веб-застосунків являє собою сферу, що охоплює процеси – проектування, створення, тестування та підтримки програмного забезпечення. Функціонування якого здійснюється через глобальну мережу Інтернет з використанням протоколу HTTP. Даний вид програмного забезпечення, на відміну від традиційних інсталяційних програм, функціонує на віддалених серверах, а взаємодія з кінцевим користувачем відбувається за посередництвом веб-браузера [8].

2.2.1 Монолітна та Мікросервісна архітектури

В сучасній розробці програмного забезпечення, дедалі складніші вимоги диктують необхідність адаптивних та масштабованих рішень, вибір архітектурного підходу набуває критичного значення. Серед розмаїття доступних варіантів виділяють: монолітну та мікросервісну архітектури.

Монолітна архітектура, традиційний підхід, характеризується побудовою застосунку як єдиного, неподільного блоку. Всі компоненти системи тісно пов'язані, розгорнуті та функціонують як єдине ціле в рамках спільної кодової бази та середовища виконання. Цей підхід, широко розповсюджений у минулому та досі актуальний для певних сценаріїв, не позбавлений, однак, недоліків, які стають особливо помітними в контексті сучасних вимог до гнучкості та масштабованості.

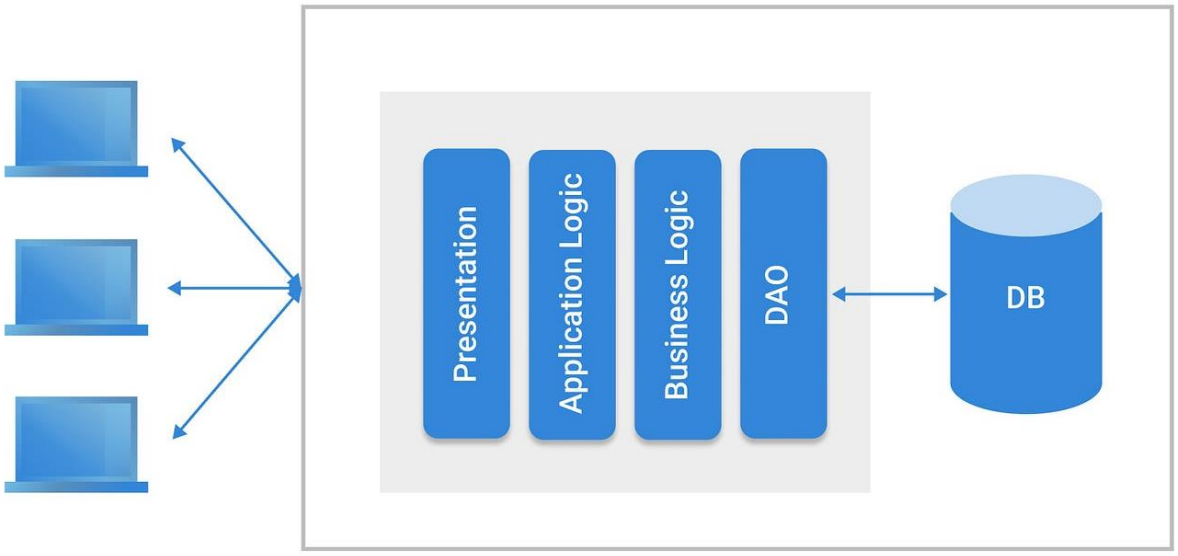

Рисунок 2.1 – Монолітна архітектура

Проблеми, пов'язані з монолітною архітектурою:

- Складнощі масштабування: Масштабування монолітного застосунку часто вимагає дублювання всього застосунку, а не лише тих компонентів, які потребують додаткових ресурсів, що призводить до неефективного використання ресурсів.
- Обмежений технологічний стек: Моноліти, як правило, побудовані на єдиному технологічному стеку, що ускладнює інтеграцію нових технологій або мов програмування, потенційно обмежуючи інновації та оптимізацію [15].

- Уповільнений цикл розробки: Внесення змін у монолітний застосунок, навіть незначних, часто вимагає повного перерозгортання, що значно уповільнює цикл розробки та випуску оновлень.
- Ускладнене обслуговування: Зі зростанням розміру та складності кодової бази моноліту, його підтримка та рефакторинг стають дедалі складнішими. Взаємозалежність компонентів може призвести до каскадних помилок та ускладнити процес ізоляції та виправлення дефектів.

На відміну від монолітного підходу, мікросервісна архітектура пропонує принципово інший шлях, розглядаючи застосунок як сукупність незалежних, слабозв'язаних сервісів, кожен з яких відповідає за окрему бізнес-функціональність. Така декомпозиція забезпечує ряд переваг:

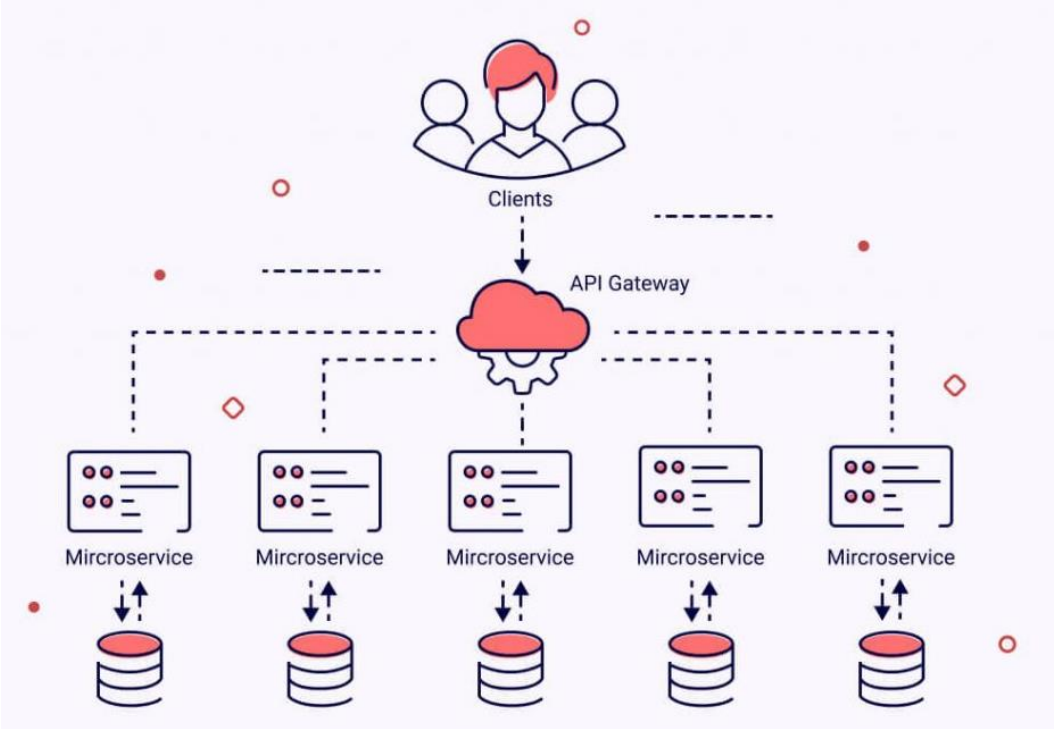

Рисунок 2.2 – Структура архітектури мікросервісів

- Незалежне масштабування: Кожен мікросервіс може масштабуватися незалежно, дозволяючи оптимально розподіляти ресурси відповідно до поточного навантаження.

- Технологічна різноманітність: Кожен мікросервіс може бути розроблений з використанням оптимального для його функціональності технологічного стеку.
- Пришвидшений цикл розробки: Незалежність мікросервісів дозволяє командам розробників працювати автономно, прискорюючи процес розробки та випуску оновлень.
- Підвищена стійкість до збоїв: Вихід з ладу одного мікросервісу не обов'язково призводить до зупинки всього застосунку [14].

Щоб наглядно побачити відмінності між цими підходами нижче представлена порівняльна таблиця 2.1, яка висвітлює ключові характеристики кожної архітектури.

Таблиця 2.1 – Порівняльна таблиця

Критерій	Монолітна архітектура	Мікросервісна архітектура
Структура	Єдиний, неподільний блок	Набір незалежних, слабозв'язаних сервісів
Розгортання	Єдиний модуль, повне перерозгортання	Незалежне розгортання кожного сервісу
Масштабування	Масштабується весь застосунок	Масштабуються окремі сервіси
Технологічний стек	Як правило, єдиний стек	Можливість використовувати різні стеки для кожного сервісу
Розробка	Повільніший цикл розробки, тісна залежність команд	Швидший цикл розробки, автономні команди
Обслуговування	Складніше з ростом кодової бази	Легше, завдяки ізольованості сервісів
Стійкість до збоїв	Збій одного компонента може вивести з ладу весь застосунок	Збій одного сервісу не обов'язково впливає на інші
Складність	Відносно проста на початкових етапах	Складніша в управлінні та інфраструктурі

Комунікація	Внутрішньопроцесова комунікація	Міжсервісна комунікація (наприклад, через REST API, gRPC)
Розподіл даних	Зазвичай єдина база даних	Кожен сервіс може мати власну базу даних
Транзакції	Прості транзакції в межах єдиної бази даних	Складніші розподілені транзакції
Приклади	Традиційні веб-додатки, CRM-системи на ранніх етапах	Netflix, Amazon, Uber, eBay

В контексті даного проекту, після ретельного аналізу потенційних архітектурних рішень, було прийнято рішення на користь мікросервісної архітектури. Цей вибір обумовлений, в першу чергу, необхідністю забезпечення високої масштабованості та гнучкості системи. Очікується, що проект буде розвиватися та розширюватися в майбутньому, і мікросервісний підхід дозволить ефективно реагувати на ці зміни, незалежно масштабуючи окремі компоненти та впроваджуючи нові функції без впливу на всю систему в цілому. Крім того, можливість використання різних технологічних стеків для кожного окремого сервісу відкриває шлях до оптимізації та інновацій, дозволяючи використовувати найбільш підходящі інструменти для вирішення конкретних завдань.

Вибір на користь мікросервісів також продиктований прагненням до прискорення циклу розробки та підвищення стійкості системи до збоїв. Незалежність мікросервісів дозволяє командам працювати автономно, паралельно розробляючи та тестуючи різні частини системи. Це, в свою чергу, веде до швидшого випуску оновлень та оперативного реагування на потреби користувачів. Більше того, ізольованість сервісів гарантує, що збій в одному з них не призведе до зупинки всього застосунку, забезпечуючи безперервність роботи та високий рівень доступності. Таким чином, мікросервісна архітектура є стратегічним вибором, спрямованим на створення надійної,

гнучкої та масштабованої системи, здатної ефективно розвиватися в довгостроковій перспективі.

2.2.2 WEB-сервіси, SOA, API

Веб-сервіси, сервісно-орієнтована архітектура (SOA) та програмні інтерфейси додатків (API) є фундаментальними концепціями сучасної програмної інженерії, що забезпечують інтеграцію, взаємодію та автоматизацію між програмними системами. У контексті веб-розробки вони відіграють ключову роль у створенні масштабованих та адаптивних програмних рішень [5].

Веб-сервіси визначаються як програмні компоненти, що забезпечують обмін даними між додатками через інтернет за допомогою стандартних протоколів, таких як HTTP або HTTPS. Основними характеристиками веб-сервісів є їхня модульність, платформонезалежність та стандартизований обмін даними. Для реалізації взаємодії використовуються формати, що забезпечують сумісність, зокрема XML (eXtensible Markup Language) та JSON (JavaScript Object Notation).

Виділяють два основних підходи до реалізації веб-сервісів:

1. SOAP (Simple Object Access Protocol): Стандартизований протокол, що використовує XML для обміну повідомленнями. Він відзначається високим рівнем формалізації та безпеки, що робить його популярним у корпоративних середовищах.

2. REST (Representational State Transfer): Полегшений підхід, що використовує стандартні методи HTTP (GET, POST, PUT, DELETE). REST забезпечує простоту реалізації, швидкість роботи та легкість інтеграції.

Застосування веб-сервісів охоплює широкий спектр завдань, включаючи обробку платежів, інтеграцію геолокаційних сервісів та автоматизацію бізнес-процесів.

Сервісно-орієнтована архітектура (SOA) являє собою підхід до розробки програмного забезпечення, заснований на використанні незалежних сервісів, які виконують чітко визначені функції. Ключовим принципом SOA є слабке зв’язування компонентів, яке досягається завдяки стандартизованим інтерфейсам [9].

Основні принципи SOA:

1. Повторне використання сервісів: Сервіси розробляються таким чином, щоб їх можна було інтегрувати до різних програм.

2. Слабке зв’язування: Мінімізація залежностей між сервісами для підвищення гнучкості.

3. Інтероперабельність: Забезпечення сумісності між сервісами незалежно від технологічних платформ.

SOA широко застосовується у великих корпоративних середовищах для інтеграції ERP, CRM та інших бізнес-систем. Її основна перевага полягає у можливості децентралізованого управління, що сприяє підвищенню ефективності.

Програмний інтерфейс додатків (API) — це набір правил і протоколів, які визначають спосіб взаємодії між програмними компонентами. API може бути реалізований як частина веб-сервісу або використовуватися для інтеграції різних систем [6].

Типи API:

1. Відкриті API (Public API): Призначені для широкого використання сторонніми розробниками (наприклад, Google Maps API).

2. Приватні API: Використовуються всередині організації для забезпечення інтеграції її систем.

3. Партнерські API: Доступні лише для обраних партнерів, з метою забезпечення безпеки та контролю.

4. Веб-API: Розраховані на використання в інтернет-застосунках через стандартні протоколи.

Формати передачі даних:

- JSON: Легкий текстовий формат, що використовується для обміну даними.
- XML: Структурований формат, що забезпечує складну обробку даних.

API значно спрощує процес розробки завдяки можливості повторного використання функціональності, а також сприяє інтеграції зовнішніх сервісів.

Для кращого розуміння ключових характеристик та відмінностей між веб-сервісами, SOA та API доцільно представити їх у вигляді порівняльної таблиці 2.2.

Таблиця 2.2 – Порівняння підходів до створення Веб-застосунків

Критерій	Веб-сервіси	SOA	API
Призначення	Обмін даними між додатками через інтернет	Архітектурний підхід для інтеграції систем	Інтерфейс для взаємодії між компонентами програм
Архітектурний підхід	Ні	Так	Ні
Протоколи	HTTP, HTTPS, SOAP, REST	Використовує веб-сервіси чи інші API	HTTP, HTTPS, WebSocket
Масштабованість	Висока	Дуже висока	Залежить від реалізації
Складність реалізації	Середня	Висока	Низька або середня
Типи даних	XML, JSON	Залежить від використовуваних веб-сервісів	JSON, XML, інші
Сфери застосування	Інтеграція веб-додатків	Корпоративні системи, ERP, CRM	Мобільні додатки, інтеграція сторонніх сервісів

SOA було обрано як архітектурний підхід завдяки його здатності забезпечувати гнучкість, масштабованість і повторне використання сервісів. Це дозволяє створювати складні корпоративні системи, такі як ERP і CRM, де

сервіси можуть працювати незалежно один від одного, зберігаючи при цьому здатність до інтеграції. Завдяки слабкому зв’язуванню компонентів SOA підтримує адаптацію до змінних вимог бізнесу та технологій, що робить його ідеальним вибором для великих і децентралізованих систе

2.3 Засоби розробки клієнтської частини

Розробка клієнтської частини веб-застосунків займає центральне місце у створенні інтерактивних інтерфейсів, що забезпечують зручну взаємодію користувачів із системою. Важливу роль у цьому процесі відіграють фреймворки — програмні бібліотеки, які спрощують та пришвидшують розробку за рахунок надання готових інструментів та компонентів. Серед них можна виділити такі популярні рішення: React.js, Angular, Vue.js, Svelte та інші. Кожен із цих фреймворків має свої унікальні особливості, переваги та обмеження, що робить їх підходящими для різних завдань [10].

React.js — це бібліотека JavaScript, створена компанією Facebook (тепер Meta), яка дозволяє створювати динамічні та ефективні користувацькі інтерфейси. React фокусується на компонентному підході, який сприяє багаторазовому використанню коду та структуризації проекту [4].

Основні характеристики:

1. Компонентний підхід: React базується на використанні компонентів, які можна легко створювати, повторно використовувати та комбінувати.

2. Virtual DOM: Замість роботи безпосередньо з реальним DOM, React використовує віртуальну модель DOM, що значно підвищує продуктивність завдяки мінімізації змін у реальному DOM.

3. Односторонній потік даних: React використовує односторонню прив'язку даних, що спрощує відстеження стану застосунку та зменшує ризик помилок.

Переваги:

- Продуктивність: Завдяки Virtual DOM React швидко оновлює інтерфейси.
- Гнучкість: Бібліотека легко інтегрується з іншими інструментами та фреймворками.
- Активна спільнота: Велика кількість готових бібліотек, документації та прикладів коду.

Недоліки:

- Висока вхідна планка: Для новачків може бути складно зрозуміти концепції JSX, Virtual DOM та керування станом.
- Вибір інструментів: React — це лише бібліотека, тому для повноцінного проєкту розробнику потрібно обирати додаткові інструменти, наприклад, Redux або MobX для управління станом.

Сфери застосування:

React ідеально підходить для створення односторінкових застосунків (SPA), таких як соціальні мережі, панелі адміністрування та інтернет-магазини. Його гнучкість також дозволяє використовувати бібліотеку в розробці мобільних застосунків через React Native.

React.js є одним із найпопулярніших інструментів серед розробників завдяки своїй потужності, простоті інтеграції та підтримці з боку спільноти та корпорацій.

Angular — це повноцінний фреймворк для розробки веб-застосунків, створений Google. Angular є потужним інструментом для побудови як односторінкових, так і багатосторінкових додатків, забезпечуючи все необхідне для розробки в одному пакеті.

Основні характеристики:

1. Двостороння прив'язка даних: Angular автоматично синхронізує дані між моделлю та представленням, що спрощує управління станом.
2. Модульна структура: Код організовано у вигляді модулів, що полегшує підтримку та масштабування проекту.

3. TypeScript: Angular використовує мову програмування TypeScript, яка додає типізацію та підвищує надійність коду.

4. DI (Dependency Injection): Angular має вбудований механізм управління залежностями, що полегшує повторне використання компонентів.

Переваги:

- Повноцінність: Angular включає все необхідне для розробки, від шаблонів до управління станом.
- Швидкість: Завдяки використанню компіляції AOT (Ahead-Of-Time) додатки завантажуються швидше.
- Підтримка Google: Постійне оновлення та вдосконалення фреймворку.

Недоліки:

- Висока складність: Angular має круту криву навчання через велику кількість концепцій та складний синтаксис.
- Великі обсяги коду: Для реалізації навіть простих функцій часто потрібні значні обсяги коду.

Сфери застосування:

Angular ідеально підходить для створення складних корпоративних систем, таких як CRM, ERP, та інших багатофункціональних веб-додатків. Завдяки підтримці масштабування фреймворк використовується у великих проектах з великою кількістю інтеграцій.

Angular є вибором для команд, які потребують універсального інструменту для розробки та готові витратити час на навчання для опанування його потужних можливостей.

Svelte — це сучасний фреймворк для розробки веб-застосунків, який відрізняється від традиційних інструментів тим, що виконує більшість роботи під час етапу компіляції. Створений Річем Харрісом у 2016 році, Svelte швидко здобув популярність серед розробників завдяки своїй простоті та високій продуктивності.

Основні характеристики:

1. Компіляція на етапі збірки: Замість того щоб використовувати віртуальний DOM, Svelte компілює код у чистий JavaScript на етапі збірки, що зменшує розмір і складність виконуваного коду.

2. Реактивність: Реактивність досягається без необхідності використання додаткових бібліотек чи спеціальних методів.

3. Простота синтаксису: Код у Svelte виглядає чисто та інтуїтивно, що знижує поріг входження для новачків.

Переваги:

- Висока продуктивність: Відсутність віртуального DOM зменшує затримки та підвищує швидкість роботи.
- Менший розмір застосунків: Завдяки компіляції на етапі збірки.
- Легкість навчання: Інтуїтивно зрозумілий підхід та проста структура проектів.

Недоліки:

- Менша спільнота: Порівняно з React, Angular чи Vue, Svelte має меншу кількість розробників та доступних ресурсів.
- Інтеграція: Через унікальний підхід Svelte може бути складніше інтегрувати у великі існуючі проекти.

Svelte ідеально підходить для створення невеликих і середніх за складністю проектів, де важлива продуктивність та простота розробки. Він часто використовується для створення односторінкових додатків, прототипів та швидкого тестування ідей.

Svelte привертає увагу як інноваційний інструмент, який поєднує у собі простоту та ефективність, роблячи його перспективним вибором для сучасних веб-розробників.

Ці інструменти займають ключові позиції у сфері веб-розробки, кожен з них має власні унікальні особливості, що робить їх оптимальними для різних типів проектів. Нижче представлена таблиця, яка дає змогу швидко оцінити їх основні риси та допомагає вибрати найбільш підходящий фреймворк залежно від потреб проекту.

Таблиця 2.3 – Порівняння фреймворків

Критерій	React.js	Angular	Svelte
Тип	Бібліотека JavaScript	Повноцінний фреймворк	Фреймворк
Основна концепція	Компонентний підхід, Virtual DOM	MVC (Model-View-Controller), двостороння прив'язка даних	Компіляція під час збірки
Мова програмування	JavaScript із JSX	TypeScript	JavaScript
Продуктивність	Висока, завдяки Virtual DOM	Середня, залежить від обсягу проекту	Дуже висока, завдяки компіляції коду
Складність навчання	Середня (необхідно зрозуміти JSX та концепції React)	Висока (багато концепцій, використання TypeScript)	Низька (проста структура та інтуїтивний синтаксис)
Гнучкість	Висока, інтегрується з іншими бібліотеками	Висока, але залежить від використання фреймворку	Висока, завдяки мінімалістичному підходу
Сфери застосування	Односторінкові додатки (SPA), мобільні додатки	Великі корпоративні системи (ERP, CRM), багатосторінкові проекти	Невеликі та середні проекти, SPA
Підтримка	Facebook (Meta) та велика спільнота	Google та активна спільнота	Незалежна, невелика спільнота
Особливості	Гнучкість і велика екосистема інструментів	Універсальність, повний набір функцій для розробки	Менший розмір коду, висока продуктивність
Недоліки	Необхідність додаткових бібліотек для управління станом	Висока складність, великий розмір коду	Невелика спільнота, обмежені ресурси

React.js було обрано як основний інструмент для розробки клієнтської частини завдяки його гнучкості, продуктивності та широким можливостям інтеграції. Компонентний підхід у React дозволяє розробникам створювати багаторазово використовувані та незалежні елементи, що значно полегшує підтримку та масштабування проєктів. Завдяки Virtual DOM бібліотека забезпечує високу продуктивність і швидку реакцію інтерфейсу на зміну даних, що особливо важливо для сучасних веб-додатків із великим обсягом інтерактивності. Окрім того, активна спільнота розробників та підтримка корпорації Meta гарантують постійний розвиток бібліотеки, доступ до великої кількості ресурсів і бібліотек, а також швидке вирішення проблем.

2.4 Засоби розробки серверної частини

Розробка серверної частини веб-застосунків є ключовим етапом створення сучасних програмних систем. Ефективність, надійність та масштабованість бекенду багато в чому визначається вибором відповідних інструментів та технологій. Одним із найважливіших аспектів цього вибору є використання фреймворків, які надають розробникам готові рішення для типових завдань, спрощуючи процес розробки та підвищуючи якість кінцевого продукту. У цьому рефераті будуть розглянуті основні засоби розробки серверної частини з акцентом на огляд популярних фреймворків, включаючи ASP.NET, та аналіз їх особливостей, переваг та недоліків. Метою роботи є надання комплексного огляду сучасного інструментарію для розробки бекенду, що дозволить розробникам здійснювати обґрунтований вибір технологій в залежності від вимог конкретного проекту [5].

Сучасний ландшафт розробки серверної частини характеризується широким спектром фреймворків, що відрізняються мовами програмування, архітектурними підходами та функціональними можливостями. Фреймворк, в контексті бекенд-розробки, являє собою програмний каркас, що надає набір бібліотек, інструментів та шаблонів проектування для створення веб-

застосунків. Використання фреймворків дозволяє значно прискорити процес розробки, стандартизувати структуру коду, підвищити його надійність та спростити подальшу підтримку та розширення функціональності.

Одним з ключових критеріїв класифікації фреймворків є мова програмування, що використовується для розробки. Популярними мовами для створення бекенду є Python, Java, Node.js (JavaScript), Ruby, PHP та C#. Кожна з цих мов має свої особливості, переваги та недоліки, а також екосистему фреймворків, що сформувалася навколо неї.

Python славиться своєю простотою, читабельністю коду та потужними бібліотеками для обробки даних та машинного навчання. Серед найбільш відомих Python-фреймворків можна виділити Django та Flask. Django є високорівневим фреймворком, що реалізує архітектурний шаблон Model-View-Template (MVT) та надає широкий спектр вбудованих функцій, включаючи ORM (Object-Relational Mapper), систему автентифікації та адміністративний інтерфейс. Flask, у свою чергу, є мікрофреймворком, що пропонує мінімальний набір необхідних інструментів, залишаючи розробнику більшу свободу у виборі додаткових компонентів та організації структури застосунку.

Java, відома своєю надійністю, продуктивністю та крос-платформеністю, широко використовується для розробки корпоративних застосунків. Одним із найпопулярніших Java-фреймворків є Spring. Spring являє собою потужну екосистему, що надає рішення для різних аспектів розробки, включаючи впровадження залежностей (Dependency Injection), аспектно-орієнтоване програмування (Aspect-Oriented Programming), роботу з базами даних та веб-розробку (Spring MVC).

Node.js, платформа, що дозволяє виконувати JavaScript на сервері, завоювала популярність завдяки своїй асинхронній, неблокуючій моделі вводу-виводу, що забезпечує високу продуктивність при обробці великої кількості одночасних запитів. Express.js є найбільш поширеним фреймворком

для Node.js, що надає мінімалістичний та гнучкий інструментарій для створення веб-серверів та API.

Ruby on Rails, фреймворк, написаний на мові Ruby, відомий своєю швидкістю розробки та конвенціями, що спрощують конфігурацію. Rails реалізує архітектурний шаблон Model-View-Controller (MVC) та надає розробникам готові рішення для багатьох типових завдань, дозволяючи зосередитись на логіці застосунку.

PHP, мова, що історично використовувалась для веб-розробки, також має ряд популярних фреймворків, серед яких можна виділити Laravel та Symfony. Laravel, відомий своєю елегантністю та простотою, пропонує широкий спектр функцій, включаючи маршрутизацію, шаблонізатор Blade, ORM Eloquent та систему автентифікації. Symfony, у свою чергу, є більш низькорівневим фреймворком, що надає набір компонентів, які можна використовувати як окремо, так і в рамках повноцінного Symfony-додатку.

Вибір конкретного фреймворку залежить від безлічі факторів, включаючи вимоги до продуктивності, масштабованості, складності проекту, досвід команди розробників та екосистему доступних інструментів та бібліотек. Розглянемо більш детально Django та ASP.NET.

1. Фреймворк Django

Django, будучи одним з найпопулярніших Python-фреймворків, заслуговує на більш детальний розгляд. Його високорівнева природа, реалізація шаблону Model-View-Template (MVT) та філософія "batteries included" ("батарейки в комплекті") роблять його потужним інструментом для швидкої розробки веб-застосунків будь-якої складності.

Django використовує шаблон MVT, який є варіацією класичного MVC. У даному контексті, Model відповідає за взаємодію з базою даних, View обробляє HTTP-запити та формує відповіді, а Template відповідає за візуальне представлення даних. Такий поділ логіки забезпечує чітку структуру коду, спрощує його розуміння та підтримку.

Django постачається з широким набором вбудованих функцій, що охоплюють більшість потреб веб-розробки. До них відносяться:

- ORM (Object-Relational Mapper): Дозволяє взаємодіяти з базою даних, використовуючи об'єкти Python замість написання SQL-запитів. Це спрощує роботу з даними та робить код більш читабельним.
- Система автентифікації та авторизації: Надає готові рішення для управління користувачами, групами та правами доступу.
- Адміністративний інтерфейс: Автоматично генерує адміністративний інтерфейс для управління даними, що значно економить час на розробку.
- Система шаблонів: Дозволяє створювати динамічні HTML-сторінки, використовуючи простий та інтуїтивно зрозумілий синтаксис.
- Робота з формами: Спрощує обробку форм, валідацію даних та захист від CSRF-атак.
- Система маршрутизації (URL dispatcher): Дозволяє гнучко налаштовувати відповідність між URL-адресами та функціями обробки запитів (view).

Переваги Django:

- Швидкість розробки: Завдяки високорівневій природі та багатому функціоналу, Django дозволяє швидко створювати прототипи та розробляти повноцінні веб-застосунки.
- Безпека: Django має вбудовані засоби захисту від поширених веб-атак, таких як SQL-ін'єкції, XSS та CSRF.
- Масштабованість: Django підходить для розробки як невеликих, так і високонавантажених проектів.
- Спільнота: Django має велику та активну спільноту розробників, що забезпечує доступ до документації, навчальних матеріалів та підтримки.

Недоліки Django:

- Монолітність: Django є відносно монолітним фреймворком, що може ускладнювати його використання у мікросервісних архітектурах.

- Крута крива навчання: Для ефективного використання Django необхідно вивчити його внутрішню структуру та принципи роботи.

Django широко використовується для розробки різноманітних веб-застосунків, включаючи соціальні мережі, контент-менеджмент системи (CMS), електронні комерційні платформи, новинні портали та інші.

2. Фреймворк ASP.NET

ASP.NET, розроблений компанією Microsoft, є потужним фреймворком для створення веб-застосунків та веб-сервісів на платформі .NET. Він надає розробникам широкий спектр інструментів та бібліотек, що дозволяють створювати динамічні, масштабовані та безпечні веб-рішення. ASP.NET підтримує декілька мов програмування, включаючи C#, F# та Visual Basic .NET, але найпоширенішою мовою для розробки ASP.NET застосунків є C#.

Еволюція ASP.NET: Історія ASP.NET налічує декілька поколінь. Спочатку з'явився класичний ASP.NET, що базувався на моделі Web Forms. Потім з'явився ASP.NET MVC, що реалізував шаблон Model-View-Controller. Сучасним втіленням ASP.NET є ASP.NET Core - крос-платформенний, високопродуктивний фреймворк з відкритим вихідним кодом, що підтримує розробку як традиційних веб-застосунків, так і хмарних сервісів та мікросервісів.

Ключові особливості ASP.NET Core:

- Крос-платформеність: ASP.NET Core може працювати на різних операційних системах, включаючи Windows, macOS та Linux. Це забезпечує гнучкість у виборі платформи для розгортання та дозволяє створювати веб-застосунки, що можуть функціонувати в різних середовищах.
- Модульність: ASP.NET Core побудований на основі модульної архітектури. Розробники можуть вибирати та використовувати лише ті компоненти, які необхідні для конкретного проекту, що зменшує розмір застосунку та підвищує його продуктивність.

- Впровадження залежностей (Dependency Injection): ASP.NET Core має вбудовану систему впровадження залежностей, що спрощує керування залежностями між компонентами та полегшує тестування коду.
- Висока продуктивність: ASP.NET Core є одним з найпродуктивніших веб-фреймворків, завдяки оптимізованому конвеєру обробки запитів та підтримці асинхронного програмування.
- Підтримка мікросервісів: ASP.NET Core ідеально підходить для розробки мікросервісних архітектур, завдяки своїй модульності, легкості та підтримці контейнеризації (Docker).
- Інтеграція з іншими сервісами Microsoft: ASP.NET Core легко інтегрується з іншими сервісами Microsoft, такими як Azure, SQL Server та Active Directory.

Інструменти розробки:

Основним інструментом для розробки ASP.NET Core застосунків є Visual Studio - потужне інтегроване середовище розробки (IDE) від Microsoft. Visual Studio надає широкий спектр функцій, включаючи редактор коду з підтримкою IntelliSense, налагоджувач, інструменти для роботи з базами даних та хмарними сервісами. Також можна використовувати Visual Studio Code - легкий крос-платформенний редактор коду з підтримкою розширень, що дозволяє налаштувати його для розробки ASP.NET Core.

Переваги ASP.NET Core:

- Продуктивність та масштабованість: ASP.NET Core забезпечує високу продуктивність та можливість масштабування для високонавантажених застосунків.
- Безпека: ASP.NET Core має вбудовані засоби захисту від поширених веб-атак та інтегрується з системами автентифікації та авторизації.
- Спільнота та підтримка: ASP.NET Core підтримується корпорацією Microsoft та має велику спільноту розробників.

Недоліки ASP.NET Core:

- Відносна складність: Для ефективного використання ASP.NET Core потрібен певний рівень знань платформи .NET та мови C#.
- Залежність від екосистеми Microsoft: Хоча ASP.NET Core є кросплатформенним, він тісно інтегрований з екосистемою Microsoft [7].

ASP.NET Core широко використовується для розробки корпоративних веб-застосунків, веб-сервісів, інтернет-магазинів, порталів та інших веб-рішень, де важливі продуктивність, безпека та масштабованість.

Розглянуті фреймворки, Django, ASP.NET Core, являють собою потужні інструменти для розробки серверної частини веб-застосунків. Кожен з них має свої особливості, переваги та недоліки, що робить їх придатними для різних типів проектів та вимог. Наглядно можна побачити в таблиці 2.4.

Таблиця 2.4 – Порівння засобів реалізації серверної частини

Критерій	Django (Python)	ASP.NET Core (C#)
Мова програмування	Python	C#
Архітектура	MVT	MVC, Razor Pages
Рівень абстракції	Високий	Середній - Високий
Продуктивність	Середня	Висока
Масштабованість	Добра	Відмінна
Безпека	Добра	Відмінна
Спільнота та підтримка	Велика та активна	Велика, підтримка Microsoft
Порог входу	Низький	Середній

В рамках даного проекту вибір пав на ASP.NET Core в якості основного фреймворку для розробки серверної частини. Це рішення було прийнято після ретельного аналізу вимог до продуктивності, масштабованості та безпеки, а також з урахуванням досвіду команди розробників. Одним з ключових факторів стало те, що ASP.NET Core є високоефективною платформою, здатною обробляти велику кількість запитів з мінімальними затримками. Це критично важливо для забезпечення швидкого відгуку програми та

комфортного користувацького досвіду, особливо в умовах очікуваного зростання навантаження. Крім того, ASP.NET Core пропонує надійні вбудовані механізми забезпечення безпеки, включаючи захист від поширених веб-атак, що дозволяє створити надійне та захищене рішення.

2.5 Висновок до розділу два

У другому розділі було здійснено детальний аналіз засобів та інструментів для створення модуля обліку робочого часу для CRM-системи. Основну увагу приділено обґрунтуванню вибору технологічного стеку, здатного забезпечити стабільну роботу системи, її продуктивність, масштабованість та відповідність сучасним вимогам безпеки й інтеграції з іншими модулями CRM.

У процесі дослідження було розглянуто сучасні архітектурні підходи, зокрема монолітну та мікросервісну архітектури. Проведений аналіз продемонстрував, що мікросервісна архітектура є оптимальним вибором для даного проєкту, оскільки вона забезпечує незалежне масштабування компонентів, стійкість до збоїв та можливість використання різноманітних технологій для різних модулів системи.

Важливим етапом роботи стало формування вимог до програмного забезпечення, серед яких:

- Функціональні вимоги: управління робочими графіками працівників, автоматизація обліку часу, формування звітів про відпрацьований час, інтеграція з іншими модулями CRM.
- Нефункціональні вимоги: висока продуктивність, надійність зберігання даних, безпека доступу до інформації, зручність використання для кінцевих користувачів.

Обґрунтовано архітектурний підхід до розробки модуля. Клієнтська частина реалізована з використанням бібліотеки React.js, що забезпечує інтерактивний інтерфейс та компонентну структуру, яка сприяє повторному

використанню елементів. Серверна частина розроблена за допомогою ASP.NET Core, який гарантує високу продуктивність, безпеку та можливість інтеграції з існуючими CRM-системами. Для зберігання даних було запропоновано використання реляційної бази даних, яка забезпечує ефективне управління інформацією про працівників та їх графіки.

Таким чином, у другому розділі було обґрунтовано вибір сучасного технологічного стеку для реалізації модуля обліку робочого часу, визначено функціональні та нефункціональні вимоги до системи, а також описано архітектурний підхід, який забезпечує ефективну роботу програмного продукту та створює основу для його подальшої реалізації.

РОЗДІЛ 3 РОЗРОБКА МОДУЛЮ ОБЛІКУ РОБОЧОГО ЧАСУ ДЛЯ CRM-СИСТЕМИ

3.1 Опис програмної реалізації

В сучасному бізнес-середовищі використання програмного забезпечення для обліку робочого часу стає необхідним елементом ефективного управління кадровими ресурсами. Розробка таких систем допомагає автоматизувати управління командою забезпечивши точність та швидкість роботи.

Модуль складається з двох частин:

1) Системи обліку
2) Системи звітів

Далі розглядатимуться класи модулю «Commands» (рис. 3.1).

Рисунок 3.1 – структура модулю «Commands»

Ці класи представляють собою спеціалізовані структури, призначені для ефективного зберігання даних і параметрів. Вони забезпечують зручний механізм для запису інформації, отриманої з JSON файлу, а також формування гнучких колекцій. Окремої уваги заслуговує клас PeriodCalendarCommands, до складу якого входять унікальні структури PeriodCalendarSearchCommand та PeriodCalendarMassCalculateCommand. (рис. 3.2).

```
4 references
public class PeriodCalendarSearchCommand : BaseCommand
{
    1 reference
    public int PeriodFrom { get; set; }
    2 references
    public int? PeriodTo { get; set; }
    3 references
    public string? EmployeeNumber { get; set; }
    1 reference
    public int OrganizationId { get; set; }
    2 references
    public int? RegimeId { get; set; }
}

3 references
public class PeriodCalendarMassCalculateCommand : BaseCommand
{
    5 references
    public int PeriodFrom { get; set; }
    4 references
    public int? PeriodTo { get; set; }
    1 reference
    public int OrganizationId { get; set; }
    8 references
    public int[]? EmployeeIds { get; set; }
}
```

Рисунок 3.2 – Клас PeriodCalendarCommands

Клас PeriodCalendarSearchCommand є ключовим інструментом для гнучкої фільтрації періодів у календарі. Його функціонал дозволяє налаштувати пошукові параметри, такі як діапазон дат, кількість днів відпустки чи лікарняних, персональний номер працівника, ідентифікатор

організації та специфічний режим роботи. Це забезпечує точність і релевантність отриманих результатів, спрощуючи аналіз даних.

Клас PeriodCalendarMassCalculateCommand розроблений для розрахунку періодів у календарі. Завдяки цьому класу можна задати не лише діапазон періодів і ідентифікатор організації, а й передати масив ідентифікаторів різних режимів роботи для автоматизованого виконання розрахунків. Це значно прискорює процеси, пов'язані з плануванням і управлінням.

У загальному контексті ці класи не лише забезпечують передачу та обробку даних календаря періодів, а й виступають інструментами для оптимізації управлінських процесів. Вони дозволяють підвищити ефективність роботи з графіками співробітників, інтегруючись у сучасні системи управління та автоматизації, що робить їх незамінною частиною комплексного підходу до управління персоналом.

Наступним класом є RegimeCommands (рис. 3.3).

```
public class RegimeCreateCommand : BaseCommand
{
    0 references
    public int Code { get; set; }

    1 reference
    public string Name { get; set; }

    0 references
    public bool IsCircle { get; set; }

    0 references
    public int DaysCount { get; set; }

    1 reference
    public IEnumerable<WorkDayDetailDto> WorkDayDetails { get; set; }

    2 references
    public IEnumerable<int> RestDayDetails { get; set; }

    0 references
    public int RestDayCount => RestDayDetails.Count();

    1 reference
    public DateTime StartDateInCurrentYear { get; set; }

    0 references
    public DateTime? StartDateInPreviousYear { get; set; }

    1 reference
    public DateTime? StartDateInNextYear { get; set; }

    0 references
    public int ShiftsCount { get; set; }
}

4 references
public class RegimeUpdateCommand : RegimeCreateCommand
{
    4 references
    public int Id { get; set; }
}
```

Рисунок 3.3 – Клас RegimeCommands

Класи RegimeCreateCommand та RegimeUpdateCommand виступають як основні інструменти для управління режимами роботи, створюючи гнучкі та адаптовані об'єкти-команди, які дозволяють ефективно налаштовувати графіки праці співробітників.

RegimeCreateCommand надає можливість створення нових режимів роботи, інтегруючи всі необхідні параметри. Цей клас містить унікальні властивості, такі як код режиму, його назва, позначення циклічності, кількість днів у режимі, детальний опис робочих та вихідних днів, а також початкові дати для поточного, попереднього та наступного років. Властивість RestDayCount, яка автоматично обчислює кількість вихідних днів на основі введених даних, виділяє цей клас серед аналогічних структур, роблячи його більш універсальним і зручним.

RegimeUpdateCommand розширює функціонал RegimeCreateCommand, додаючи унікальний ідентифікатор (Id), що забезпечує можливість адресного оновлення існуючих режимів роботи. Завдяки цьому класу можна не тільки модифікувати параметри, але й інтегрувати додаткові дані для точного налаштування графіків, що робить процес оновлення максимально ефективним.

Обидва класи відіграють центральну роль у процесах автоматизації управління графіками роботи, дозволяючи з легкістю передавати й обробляти необхідні дані.

Для управління та збереження інформації про робочі дні використовуються класи WorkDayCommands та WorkDayRegimeCommands. Зокрема, WorkDayCommands має багаторівневу структуру, яка включає спеціалізовані елементи для глибокого налаштування графіків. Завдяки своїй модульній конструкції ці класи дозволяють створювати високоефективні рішення для управління робочим часом і забезпечують легку інтеграцію в більш широкі системи управління персоналом.

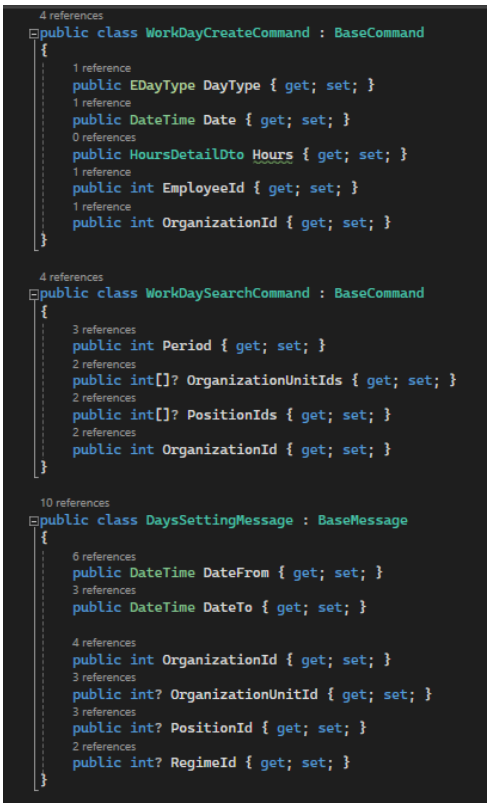

Рисунок 3.4 – Клас WorkDayCommands

Класи WorkDayCreateCommand, WorkDaySearchCommand та DaysSettingMessage є спеціалізованими інструментами для управління робочими днями та їх налаштуванням, створюючи потужний функціонал для оптимізації графіків роботи співробітників у сучасних організаціях.

WorkDayCreateCommand слугує для створення нових робочих днів, забезпечуючи точність і повноту даних. Цей клас інтегрує важливі параметри, такі як тип дня (DayType), дата (Date), розклад годин роботи (Hours), ідентифікатор працівника (EmployeeId) та організації (OrganizationId). Його гнучкість дозволяє налаштовувати навіть найдрібніші деталі робочих днів, що робить його ідеальним для складних організаційних процесів.

WorkDaySearchCommand надає розширені можливості для пошуку та фільтрації робочих днів за заданими критеріями. Завдяки властивостям, таким як період (Period), ідентифікатори підрозділів (OrganizationUnitIds) і посад (PositionIds), а також організаційний ідентифікатор (OrganizationId), цей клас дозволяє легко знаходити релевантні дані для аналізу. Він сприяє не лише

пошуку, але й глибокому аналізу, зокрема для стратегічного планування та управління персоналом.

DaysSettingMessage виділяється своєю функціональністю для конфігурації робочих днів у певному часовому інтервалі. Цей клас, успадкований від BaseMessage, включає властивості, що визначають період налаштування (DateFrom та DateTo), ідентифікатори організації (OrganizationId), відділу (OrganizationUnitId), посади (PositionId) і режиму роботи (RegimeId). Унікальність цього класу полягає у його здатності враховувати багаторівневі організаційні структури та специфічні режимні налаштування, що дозволяє створювати кастомізовані графіки, адаптовані під унікальні вимоги організації.

Завершує модуль Commands клас WorkDayRegimeCommands, який забезпечує інтеграцію даних про режими роботи з управлінням робочими днями. Цей клас завершує комплексний підхід до налаштування робочих днів, дозволяючи організаціям ефективно керувати графіками, оптимізувати процеси та підвищувати продуктивність.

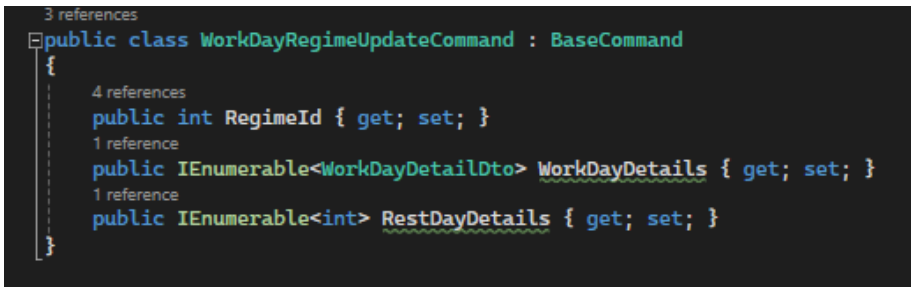

Рисунок 3.5 – Клас WorkDayRegimeCommands

Клас WorkDayRegimeUpdateCommand є унікальним інструментом для оновлення робочих днів у межах визначеного робочого режиму, успадкованим від базового класу BaseCommand. Він надає широкі можливості для кастомізації графіків роботи завдяки своїй багатофункціональній структурі, що включає:

- RegimeId — ідентифікатор робочого режиму, який виступає ключовим параметром для ідентифікації контексту оновлення. Це дозволяє чітко визначати межі застосування команди.
- WorkDayDetails — колекція, що забезпечує деталізований опис кожного робочого дня. Вона дозволяє задавати такі параметри, як тип дня, дату та деталі робочих годин, надаючи можливість для гнучкого налаштування та оптимізації графіків.
- RestDayDetails — набір ідентифікаторів днів, які мають бути визначені як вихідні. Завдяки цій властивості можна швидко і зручно виділяти дні для відпочинку, забезпечуючи баланс між робочими навантаженнями та відпочинком.

Цей клас створений для інтеграції в складні системи управління графіками, де потрібна висока точність і адаптивність у налаштуванні робочих днів.

Клас HoursDetails додає унікальність у представлення даних про робочий час, забезпечуючи структурований підхід до врахування особливостей роботи в різні періоди. Він включає:

- Summary — загальна тривалість робочого часу, яка дозволяє швидко оцінити навантаження працівника;
- Day, Night, Evening — деталізація роботи за денними, нічними та вечірніми годинами, що сприяє кращому розумінню специфіки робочого графіка;
- HolidaySummary, HolidayDay, HolidayNight, HolidayEvening — аналогічні параметри для святкових днів, що дозволяють враховувати додаткові аспекти планування, пов’язані з неробочими днями.

Особливістю HoursDetails є його здатність враховувати специфіку роботи в святкові та неробочі дні, що робить його незамінним для точного планування робочого часу та аналізу навантаження співробітників.

Разом ці класи створюють потужний набір інструментів для управління робочими режимами та графіками, забезпечуючи високу деталізацію, ефективність і адаптивність до специфічних потреб організацій.

```
6 references
public class HoursDetails
{
    4 references
    public decimal Summary { get; set; }
    4 references
    public decimal Day { get; set; }
    4 references
    public decimal Night { get; set; }
    4 references
    public decimal Evening { get; set; }
    4 references
    public decimal HolidaySummary { get; set; }
    4 references
    public decimal HolidayDay { get; set; }
    4 references
    public decimal HolidayNight { get; set; }
    4 references
    public decimal HolidayEvening { get; set; }
}
```

Рисунок 3.6 – модель HoursDetails

Наступний модуль «Dto» (рис. 3.7):

Рисунок 3.7 – Структура модулю «Dto»

Розглянемо класи цього модуля. Клас EmpDayDto (рис. 4.13) є об'єктом передачі даних (DTO), що дозволяє представляти комплексну інформацію про робочий день працівника. Він включає ключові властивості: ідентифікатор працівника, дату робочого дня, кількість годин, проведених на роботі протягом дня, ввечері та вночі. Унікальною особливістю класу є властивість Summary, яка надає можливість включати підсумкову інформацію або додаткові примітки про робочий день. Завдяки цьому клас забезпечує детальне і структуроване представлення робочих даних, що дозволяє використовувати його як базу для складних аналітичних завдань або звітності.

Клас EmpDayShortDto (рис. 3.8) виступає спрощеною альтернативою EmpDayDto, створеною для передачі лише основної інформації. Він містить такі властивості, як дата робочого дня та робочий статус, що дозволяє отримати швидкий огляд без деталізації. Цей клас оптимізований для використання у сценаріях, де необхідна швидкість обробки даних і зосередження на базових аспектах графіка роботи.

Особливістю цих класів є їх орієнтованість на різні рівні деталізації: EmpDayDto забезпечує повний набір даних для аналітики та деталізованого моніторингу, тоді як EmpDayShortDto фокусується на забезпеченні зручного огляду основної інформації. Разом вони створюють ефективну систему для обробки, передачі й зберігання даних про робочі дні працівників, що робить їх незамінними в автоматизованих системах обліку та управління робочим часом.

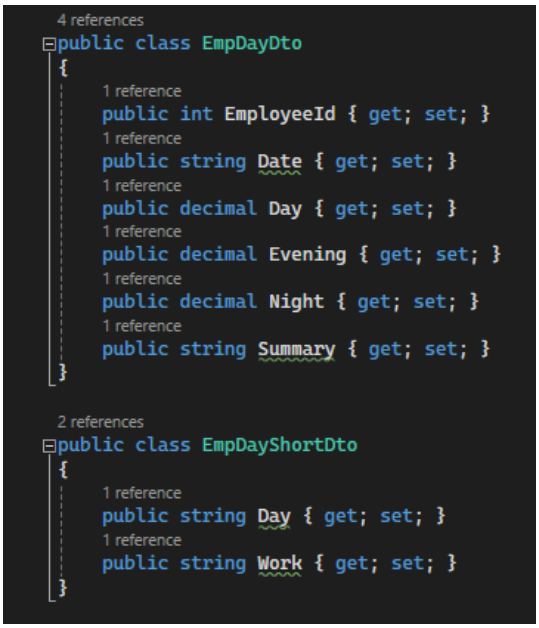

Рисунок 3.8 – Класи EmpDayDto та EmpDayShortDto

Клас HoursDetailDto (рис. 3.9) є універсальним об'єктом передачі даних (DTO), що забезпечує детальне представлення робочого часу для кожного конкретного робочого дня. Його властивості включають:

- Summary — загальний підсумок годин роботи за день, що дозволяє швидко оцінити загальний обсяг робочого часу.
- Day, Night, Evening — детальний розподіл годин за періодами доби, що допомагає аналізувати специфіку робочого графіка.
- Holiday — унікальна властивість, яка вказує, чи є цей день святковим, що додає гнучкість і точність до обліку особливих робочих днів.

Цей клас ідеально підходить для задач, де потрібна деталізація та контекстна інформація про робочі дні працівників.

Клас HoursDetailsDto (рис. 3.9) розширює можливості HoursDetailDto, додаючи унікальний функціонал для обліку святкових годин. До властивостей базового класу він додає:

- HolidaySummary — загальний підсумок годин, відпрацьованих у святкові дні.

- HolidayDay, HolidayNight, HolidayEvening — розподіл святкових годин за періодами доби, що дозволяє враховувати специфіку роботи в святкові дні.

Ця розширена функціональність робить клас HoursDetailsDto незамінним для більш складних аналітичних завдань, пов'язаних із плануванням та оптимізацією робочого часу в умовах нерегулярного графіка.

Обидва класи створені для забезпечення структурованого та гнучкого представлення даних про робочий час. Вони дозволяють не лише точно зберігати інформацію, але й здійснювати її глибокий аналіз з урахуванням різних періодів доби та статусу дня. Завдяки своїй багатофункціональності, ці класи є ключовими компонентами систем управління робочим часом, забезпечуючи інтеграцію з широким спектром бізнес-процесів, від формування графіків до стратегічного планування ресурсів.

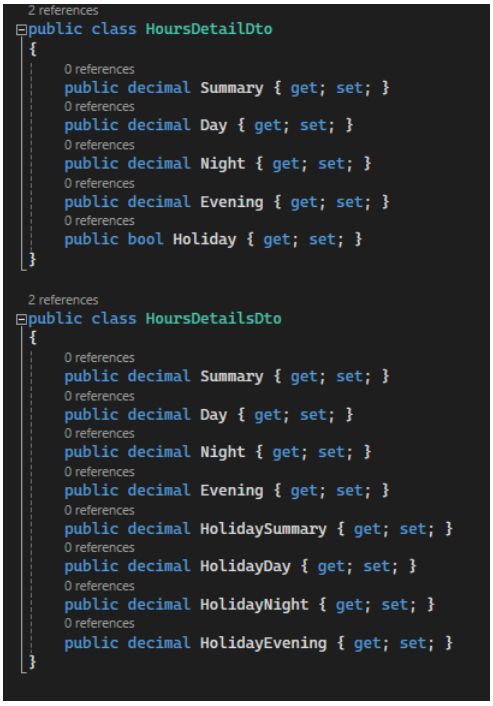

Рисунок 3.9 – Класи HoursDetailDto, HoursDetailsDto

Клас PeriodCalendarDto (DTO), використовується для зберігання інформації про календарний період, графік роботи та обліку робочого часу працівників (рис. 3.10).

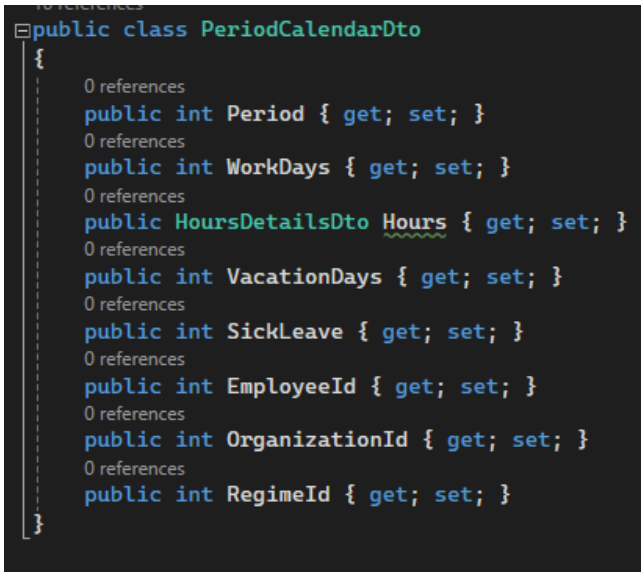

Рисунок 3.10 – Клас PeriodCalendarDto

Класи RegimeDto та CalculationRegimeDto є високоефективними об'єктами передачі даних (DTO), спеціально розробленими для управління та аналізу режимів роботи співробітників. Їх використання дозволяє забезпечити не лише зручну передачу даних, а й інтеграцію в складні системи планування та обліку робочого часу.

Клас RegimeDto надає всебічну інформацію про робочий режим, зокрема:

- Код і назва режиму, які унікально ідентифікують кожен робочий графік.
- Позначення циклічності, що дозволяє визначити, чи повторюється режим із певною періодичністю.
- Кількість днів у режимі, що дає змогу оцінити тривалість графіка.
- Деталі робочих і вихідних днів, які забезпечують повне розуміння структури режиму.

Унікальністю RegimeDto є його здатність детально моделювати режими роботи, включаючи складні графіки з різноманітними вихідними та робочими періодами, що особливо важливо для підприємств зі змінним або нестандартним графіком роботи.

Клас CalculationRegimeDto доповнює RegimeDto, зосереджуючись на аспектах, пов’язаних із розрахунками, такими як:

- Інтеграція з розрахунковими алгоритмами для оптимізації режимів.
- Забезпечення точності у плануванні навантаження та оцінці витрат робочого часу.
- Можливість аналізувати режими роботи в контексті їхньої ефективності.

Обидва класи не лише забезпечують структуроване представлення даних, а й сприяють оптимізації процесів управління графіками роботи. Вони є невід'ємними інструментами для організацій, які прагнуть автоматизувати складні процеси планування та обліку робочого часу, забезпечуючи гнучкість, точність і надійність у роботі з режимами праці.

```
public class RegimeDto
{
    0 references
    public int Code { get; set; }

    0 references
    public string Name { get; set; }

    0 references
    public bool IsCircle { get; set; }

    0 references
    public int DaysCount { get; set; }

    0 references
    public IEnumerable<WorkDayDetailDto> WorkDayDetails { get; set; }

    1 reference
    public IEnumerable<int> RestDayDetails { get; set; }

    0 references
    public int RestDayCount => RestDayDetails.Count();

    0 references
    public DateTime StartDateInCurrentYear { get; set; }

    0 references
    public DateTime? StartDateInPreviousYear { get; set; }

    0 references
    public DateTime? StartDateInNextYear { get; set; }
}
```

Рисунок 3.11 – Клас RegimeDto

Клас CalculationRegime (рис. 3.12) призначений для представлення інформації про робочий режим, необхідної для розрахунку графіка. Він містить такі ключові параметри: ідентифікатор режиму, тривалість у днях, деталі робочих і вихідних днів, позначення циклічності, а також дату початку дії режиму.

```
11 references
public class CalculationRegime
{
    3 references
    public int RegimeId { get; set; }
    7 references
    public int DaysCount => RestDays.Count() + WorkDayDetails.Sum(x => x.DaysOfWeek.Count());
    8 references
    public IEnumerable<WorkDayDetail> WorkDayDetails { get; set; } //пн-пт / 1-2
    4 references
    public IEnumerable<int> RestDays { get; set; } // сб-нд / 3-4
    2 references
    public bool IsCircle { get; set; }
    4 references
    public DateTime StartDateInCurrentYear { get; set; }
    4 references
    public DateTime? StartDateInPreviousYear { get; set; }
    4 references
    public DateTime? StartDateInNextYear { get; set; }
    5 references
    public int ShiftsCount { get; set; }
}
```

Рисунок 3.12 – Клас CalculationRegime

Клас WorkDayDetailDto (рис. 3.13) є високофункціональним об'єктом передачі даних (DTO), спеціально розробленим для точного представлення параметрів робочого дня в контексті різноманітних режимів роботи. Його основне призначення — надавати всебічну інформацію про структуру та умови робочого дня, що є критично важливим для ефективного планування графіків роботи та розрахунку робочого часу.

Ключові властивості класу забезпечують унікальні можливості:

- DaysOfWeek — перелік днів тижня, коли працівник виконує свої обов’язки.
- StartTime і EndTime — час початку та завершення робочого дня в стандартних умовах.
- IsEndTimeNextDay — позначає, чи закінчується робочий день наступного календарного дня.
- IsHolidayWork — вказує, чи передбачена робота у святкові дні.
- IsHolidayShort — визначає, чи є робочий день скороченим під час святкових днів.
- Параметри оплати обідньої перерви:
    - IsLaunchPaid — чи оплачується перерва
    - LaunchTime — тривалість обідньої перерви.

Цей клас вирізняється своєю багатофункціональністю та здатністю враховувати широкий спектр параметрів робочого дня, включаючи нестандартні сценарії. Завдяки своїй адаптивності WorkDayDetailDto є незамінним інструментом для сучасних систем управління графіками роботи, забезпечуючи баланс між деталізацією даних та зручністю їх обробки.

```
public class WorkDayDetailDto
{
    0 references
    public IEnumerable<int> DaysOfWeek { get; set; }

    0 references
    public TimeDto StartTime { get; set; }

    0 references
    public TimeDto EndTime { get; set; }

    0 references
    public bool IsEndTimeNextDay { get; set; }

    0 references
    public bool IsHolidayWork { get; set; }

    0 references
    public bool? IsHolidayShort { get; set; }

    0 references
    public TimeDto? StartTimeInHoliday { get; set; }

    0 references
    public TimeDto? EndTimeInHoliday { get; set; }

    0 references
    public bool? IsEndTimeInHolidayNextDay { get; set; }

    0 references
    public bool IsLaunchPaid { get; set; }

    0 references
    public int? LaunchTime { get; set; }
}
```

Рисунок 3.13 – Клас WorkDayDetailDto

3.2 Тестування системи

Розроблений вище функціонал, потрібно перевірити на його сумісність та коректність роботи, для цього потрібно провести тестування сервісу.

Розглянемо тестування частини програми пов’язаним з авторизацією користувача. Спочатку потрібно запустити веб-додаток та ввести власні дані для авторизації (рис. 3.14).

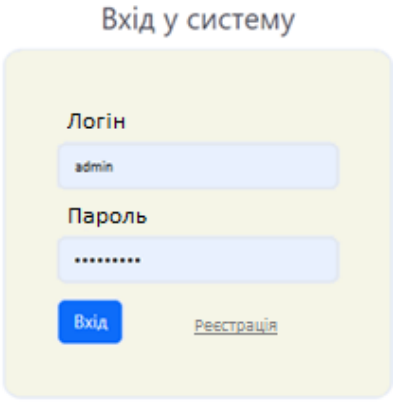

Рисунок 3.14 – Вікно логіну

Далі система повинна перевірити правильність введених даних, щоб надати доступ до основного функціоналу (рис. 3.15).

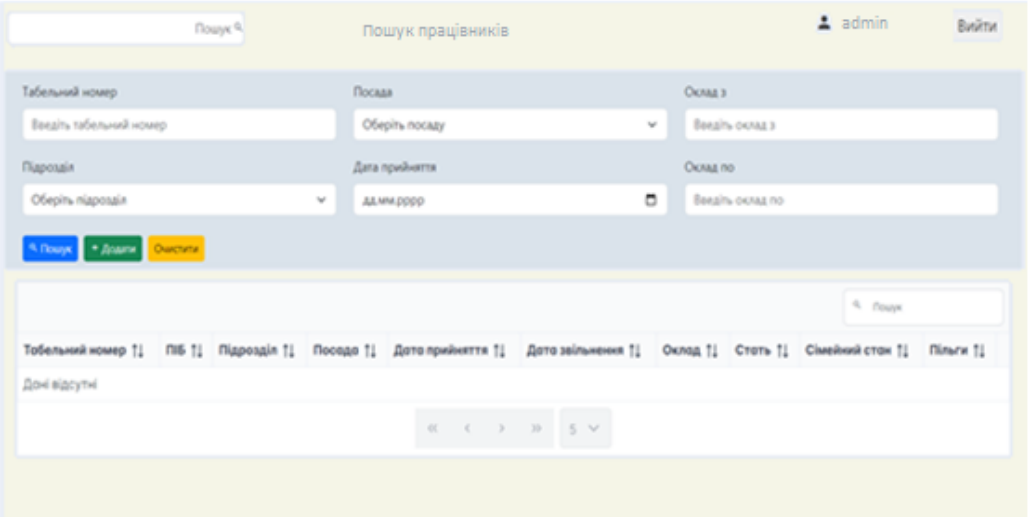

Рисунок 3.15 – Вікно програми

В іншому випадку потрібно буде повідомлено що данні введено не вірно (рис. 3.16).

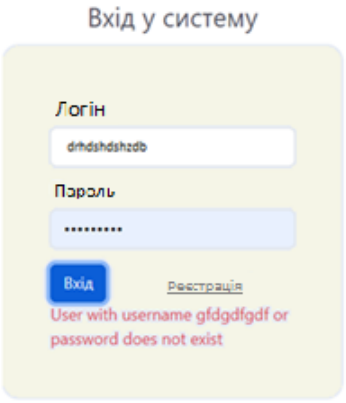

Рисунок 3.16 – Обробка не вірно введених даних

Система згідно очікувань вірно реагує на невірно введені данні супроводжуючи це повідомлення червоним кольором та не дозволяє отримати доступ до системи.

Наступний функціонал який потрібно перевірити це пошук та налаштування графіку працівників компанії. Для цього слугує наступне вікно, рис. 3.17

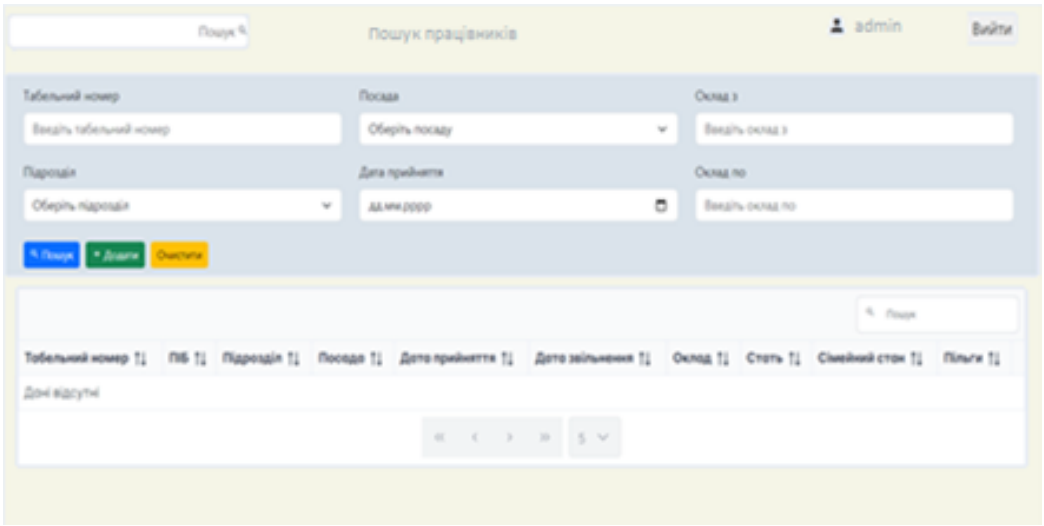

Рисунок 3.17 – Сторінку налаштування робочого графіку

Для перевірки роботи системи, потрібно ввести дані в наступні поля (рис. 3.18): табельний номер, підрозділ до якого належить працівник, календарний період, посада працівника.

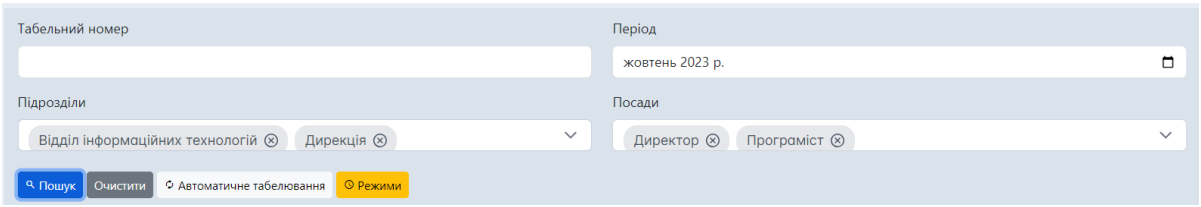

Рисунок 3.18 – Введення даних для пошуку

Якщо дані вірні програма повинна або показати знайти графіки або позначити, що такого графіку не знайдено. Після введення даних (рис 3.19), потрібно натиснути на «Пошук», щоб знайти відповідний робочий графік. В результаті програма знайшла графік на ім’я «Іванов Іван Іванович» (рис. 3.19).

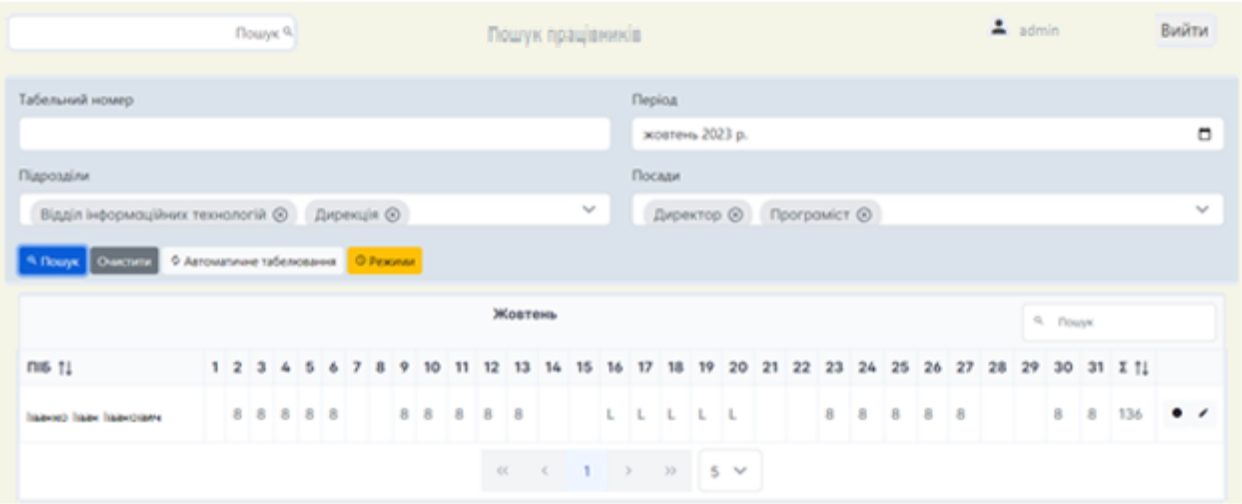

Рисунок 3.19 – робочий графік за введеними даними

В результат запиту отримаємо графік роботи людини за місяць чи інший вказаний термін. На прикладі обирався робочий місяць (грудень 2024). Де можна побачити дні роботи, відпусток, лікарняні (рис. 3.19).

Далі потрібно розглянути спосіб розрахунку відразу декілька робітників, для цього наступна форма – «Автоматичне табулювання». Для прикладу виведемо дані за період з 1 грудня по 31 за 2024, всі інші поля залишимо пустими (рис. 3.20).

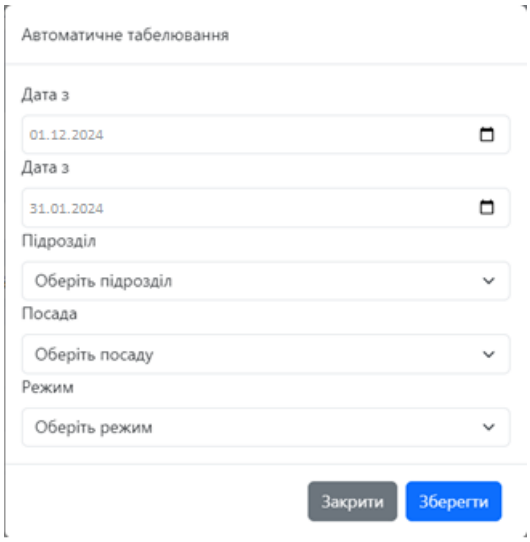

Рисунок 3.20 – вікно «Автоматичне табулювання»

В результаті система знайде відповідні графіки за вище зазначеними критеріями, щодо розпорядку в конкретно заданий період для декількох робітників (рис. 3.21).

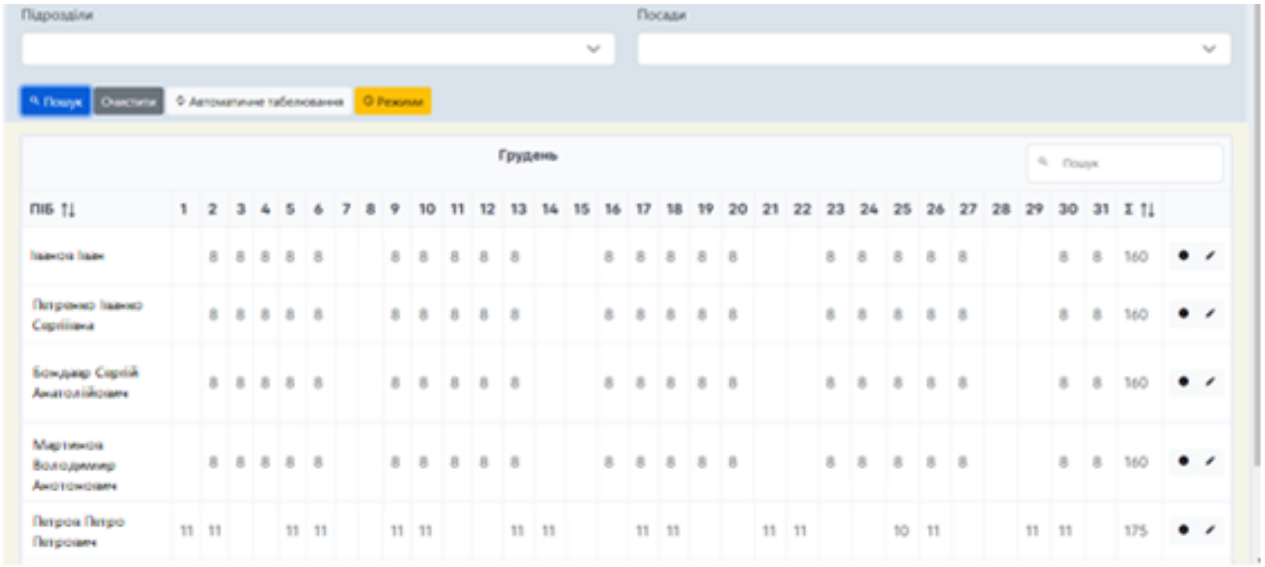

Рисунок 3.21 – виведення декількох графіків

Відповідно, якщо користувач введе не вірно дані то сиситема відреагує повідомленням про помилку (рис. 3.32).

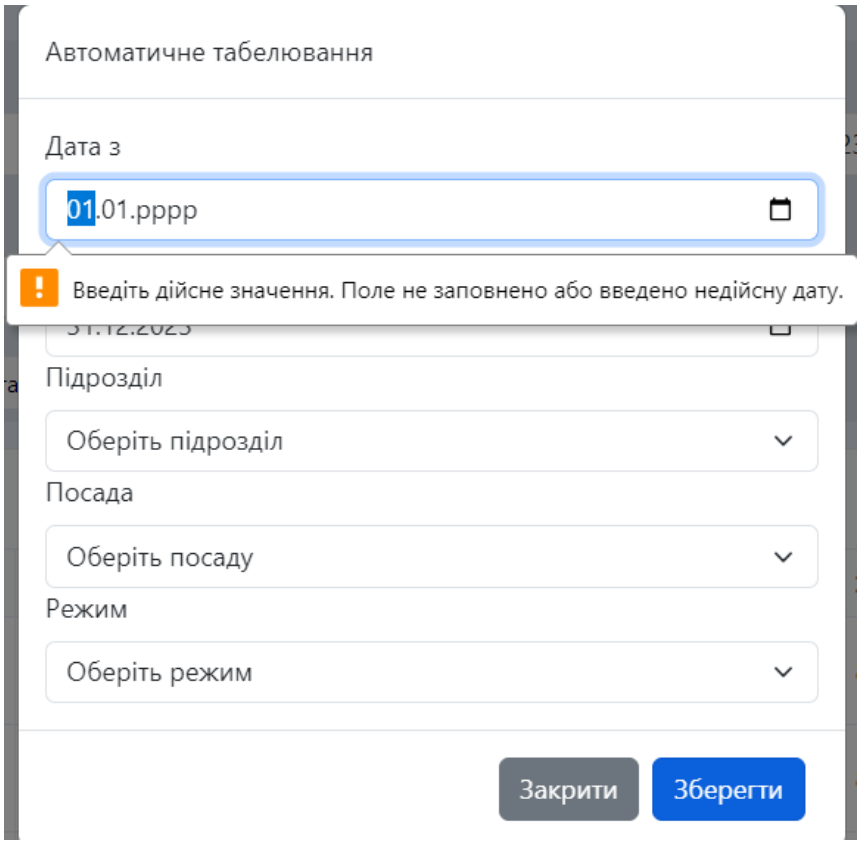

Рисунок 3.32 – виведення повідомлення про помилку

Наступним важливим аспектом системи це редагування робочого графіку. Було реалізовано можливість корегування даних так як не завжди можна внести їх коректно чи в певний період часу людина змінила свій графік роботи. (3.33).

Рисунок 3.33 – Редагування робочого графіку працівника

Добавимо до графіку відпустку. Вона позначається – V. В результаті працівник матиме робочий графік з днями відпустки (рис. 3.34).

Рисунок 3.34 – відредагований новий графік

Система також має можливість формування звітів для цього користувачу потрібно натиснути на кнопку «Розрахувати період» (рис. 3.35).

Рисунок 3.35 – Кнопка Розрахувати період

Після цього система розрахує параметри та створить звіт за наявними даними (рис. 3.36).

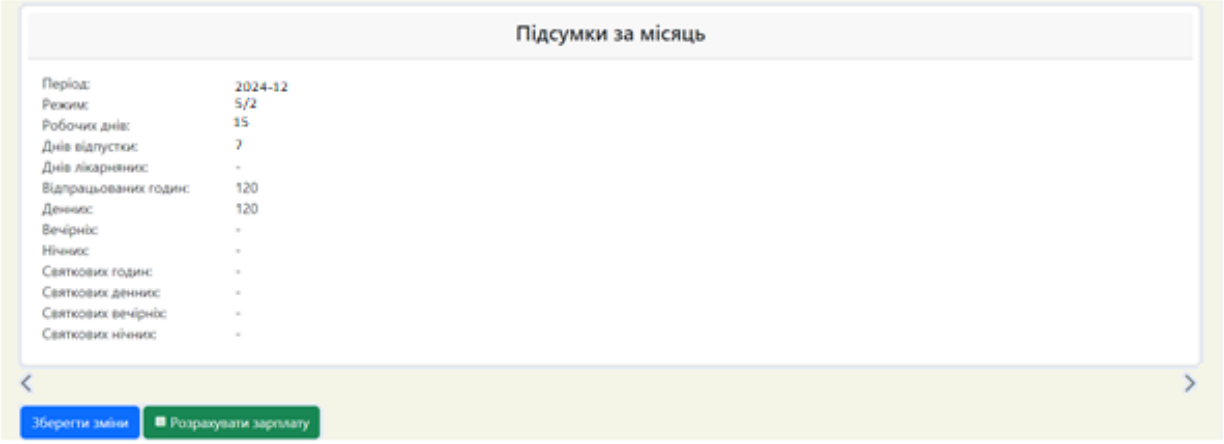

Рисунок 3.36 – Вигляд звіту за календарний місяць

3.3 Висновок до третього розділу

У третьому розділі було виконано практичну реалізацію модуля обліку робочого часу для CRM-системи. Основну увагу приділено розробці клієнтської та серверної частин, їх інтеграції, а також тестуванню функціональних можливостей програмного продукту.

Клієнтська частина розроблена з використанням бібліотеки React.js, що дозволило створити інтуїтивно зрозумілий інтерфейс користувача з можливістю динамічного оновлення даних. Серверна частина реалізована на платформі ASP.NET Core, яка забезпечує обробку HTTP-запитів, управління бізнес-логікою та взаємодію з базою даних. У процесі реалізації було інтегровано функції створення, редагування та перегляду робочих графіків, а також автоматизовано формування звітів про облік робочого часу.

Для зберігання даних використано JSON як формат файлової системи, що забезпечує простоту інтеграції з функціями модуля та мінімізує затрати на адміністрування. Такий підхід сприяє зручності обміну даними між клієнтською та серверною частинами, а також забезпечує гнучкість у роботі з різними структурами даних.

Особлива увага приділена тестуванню модуля, зокрема функціональному, навантажувальному та інтеграційному тестуванню.

Результати тестування підтвердили стабільність роботи системи, відповідність її функціональних можливостей вимогам, а також здатність ефективно обробляти великі обсяги даних.

Таким чином, у третьому розділі було реалізовано та протестовано модуль обліку робочого часу, який повністю відповідає поставленим вимогам та демонструє високу ефективність, надійність і готовність до інтеграції у CRM-систему.

ВИСНОВОК

У процесі розробки модуля обліку робочого часу для CRM-системи було виконано комплексну роботу, що включала створення архітектури проєкту, реалізацію клієнтської та серверної частин, інтеграцію з іншими модулями CRM, а також тестування функціональності програмного продукту. Модуль орієнтований на автоматизацію процесів управління робочим часом, забезпечуючи високу продуктивність і зручність для кінцевих користувачів.

Основними завданнями, вирішеними під час роботи, стали:

1. Проведення аналізу сучасних підходів до автоматизації обліку робочого часу та їх використання в CRM-системах.
2. Обґрунтування вибору технологічного стеку для реалізації проєкту, що включає React.js, ASP.NET Core і JSON.
3. Розробка вимог до програмного забезпечення:
   - Функціональні: автоматизація створення, редагування та перегляду робочих графіків, формування звітів, інтеграція з іншими модулями CRM.
   - Нефункціональні: забезпечення продуктивності, надійності, безпеки даних та зручності використання.
4. Розробка клієнтської частини із використанням бібліотеки React.js, що забезпечила компонентний підхід і динамічний користувацький інтерфейс.
5. Реалізація серверної частини на базі ASP.NET Core, яка забезпечує стабільну обробку запитів, управління бізнес-логікою та збереження даних.
6. Використання JSON як формату зберігання даних, що сприяє простій інтеграції та ефективному управлінню структурою даних.
7. Тестування програмного продукту на відповідність функціональним вимогам і стабільності роботи в умовах навантаження.

Було здійснено ґрунтовний аналіз сучасних архітектурних підходів, серед яких монолітна та мікросервісна архітектури. Обрано мікросервісну

архітектуру, яка забезпечує незалежне масштабування, гнучкість у використанні технологій та стійкість до збоїв.

Тестування підтвердило стабільну роботу модуля, відповідність функціональності заявленим вимогам, а також готовність до інтеграції в CRM-систему.

Таким чином, розроблений модуль обліку робочого часу відповідає сучасним вимогам щодо продуктивності, зручності та надійності. Проєкт повністю досяг поставленої мети, забезпечуючи автоматизацію процесів обліку часу та створюючи основу для подальшого вдосконалення і розширення функціональності.

СПИСВОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chen I. J., Popovich K. Understanding customer relationship management (CRM) People, process and technology // Business process management journal. – 2003. – Т. 9. – №. 5. – С. 672 – 688.

2. Юрчук Н. П. CRM-системи: особливості функціонування та аналіз українського ринку / Н. П. Юрчук. – Київ: Університет сучасних знань, 2018. 187 с.

3. Гевко В. Класифікація інформаційних систем управління взаємовідносинами з клієнтами / В. Гевко // Соціально-економічні проблеми і держава. 2013. Вип. 2 (9). С. 44–57

4. Legacy Reactjs. – [Електронний ресурс] – Режим доступу: https://legacy.reactjs.org.

5. Web Application Architecture in 2024: Moving in the Right Direction. – [Електронний ресурс] – Режим доступу: https://mobidev.biz/blog/web-application-architecture-types.

6. What is Client-Server Architecture? Explained in Detail. – [Електронний ресурс] – Режим доступу: https://www.theknowledgeacademy.com/blog/client-server-architecture/.

7. ASP.NET documentation. Microsoft. – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/overview

8. What is Client-Server Architecture? Explained in Detail. – [Електронний ресурс] – Режим доступу: https://www.theknowledgeacademy.com/blog/client-server-architecture/

9. Creatio CRM. Architecture and principles. – [Електронний ресурс] – Режим доступу: https://www.creatio.com/our-technologies/architecture-and-principles p.

10. What is CRM system? Full Guide: Definition & Benefits/ Creatio – [Електронний ресурс] – Режим доступу: https://www.creatio.com/page/what-is-crm

11. Toggl Track – [Електронний ресурс] – Режим доступу: https://toggl.com

12. Clockify – [Електронний ресурс] – Режим доступу: https://clockify.me

13. FreshBooks – [Електронний ресурс] – Режим доступу: https://www.freshbooks.com

14. What is Microservices Architecture? –[Електронний ресурс] – Режим доступу: https://cloud.google.com/learn/what-is-microservices-architecture

15. What is a monolithic architecture? –[Електронний ресурс] – Режим доступу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a%20singular%2C%20large%20computing%20network%20with,of%20the%20service-side%20interface.