МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА»
навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
кафедра інформаційних систем в економіці

Освітньо-професійна програма
«Інформаційні управляючі системи та технологі»

Галузь знань 12 Інформаційні технології
Спеціальність 122 Комп'ютерні науки

форма навчання: денна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ПРОЕКТАМИ

здобувача Когут Діани Леонідівни

Науковий керівник: д.е.н., професор

_____Устенко С.В.

Кваліфікаційна магістерська робота
допущена до захисту в
Екзаменаційній комісії з атестації
здобувачів вищої освіти
в.о.завідувач кафедри: к.е.н., доцент

_____ Тішков Б.О.

Київ 2022

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Київський національний економічний університет

імені Вадима Гетьмана»
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

галузь знань	12 «Інформаційні технології»
спеціальність	122 «Комп'ютерні науки»
ОПП	«Інформаційні управляючі системи та технології»

Затверджую:

В.о. завідувача кафедри

_____ Тішков Б.О.

"____" _____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Когут Діана Леонідівна
денної форми навчання
на підготовку кваліфікаційної магістерської роботи
на тему: *«Інформаційна система з управління проектами»*
Тему затверджено наказом ректора від «__» березня 2022 р. №_____.

Кваліфікаційна магістерська робота виконується на матеріалах

Кафедри інформаційних систем в економіці та науково-дослідної роботи та практики

План кваліфікаційної магістерської роботи

Розділ І ТЕОРЕТИЧНИЙ РОЗДІЛ. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

Розділ ІІ АНАЛІТИЧНИЙ РОЗДІЛ. СТВОРЕННЯ МОДЕЛЕЙ, РОЗГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ УПРАВЛІННЯ ПРОЕКТАМИ

Розділ ІІІ КОНСТРУКТИВНИЙ РОЗДІЛ. РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ

Об’єкт дослідження процес управління проектами, необхідні інструменти для членів проектної команди та менеджера

Предмет дослідження - це управління проектами та інформаційні системи управління проектами

Мета кваліфікаційної магістерської роботи є систематизація, закріплення та розширення теоретичних знань, їх застосування для побудови інформаційної системи управління проектами

Конкретні завдання, які студент повинен виконати для досягнення поставленої мети:

У розділі І: Дослідження предметної області (управління проектами), аналіз існуючих інформаційних систем з управління проектами та обгрунтування обраних технологій

У розділі ІІ: Побудова діаграми IDEF0, що відображатиме процеси системи, її декомпозиція та опис використовуваних для розробки технологій. Розгляд основних методів візуалізації в системі управління проектами

У розділі ІІІ: Практична реалізація системи управління проектами, створення бази даних, опис структури розробленої системи, детальний розгляд функціональних можливостей та процесу розробки

Завдання підготував
науковий керівник_____ **Устенко Станіслав Веніамінович**

“_____”_____ 2022 р.

Завдання одержав
студент **Когут Діана Леонідівна**

(підпис)

“_____”_____ 2022 р

АНОТАЦІЯ

кваліфікаційної магістерської роботи студентки 6 курсу ННІ «Інститут інформаційних технологій в економіці» Когут Діани Леонідівни
виконану на тему:
«Інформаційна система з управління проектами»
Київ: кафедра інформаційних систем в економіці, 2022 р.

Магістерська робота присвячена актуальній проблемі розробки системи управління проектами, по даній темі було опубліковано наукову роботу, зокрема **тези на І Всеукраїнській студентській науковій конференції «НАУКОВИЙ ПРОСТІР: АНАЛІЗ, СУЧАСНИЙ СТАН, ТРЕНДИ ТА ПЕРСПЕКТИВИ»,** текст тез наведено в додатку.

У роботі розглянуто методи полегшення командної роботи, створення інструментів планування, керування задачами та управління ресурсами: бюджетом, визначеними часовими рамками та людськими ресурсами за допомогою використання інформаційної управляючої системи, яка має вирішити проблеми пов’язані з керуванням проектами.

Дана робота складається з трьох розділів. Перший розділ являється теоретичним. У ньому наведено основні теоретичні дані, аналіз задачі, що вирішується та викладено основні вимоги та функції пов’язані із сферою управління проектами, зроблено аналіз існуючих систем. Також обгрунтовано вибір розробки веб-додатку.

Другий розділ є проектним і присвячений створенню IDEF0 моделі. Зроблено опис та аналіз використаних для розробки технологій.

Третій розділ конструктивний містить в собі інформацію про створення бази даних, результати розробки серверної та інтерфейсної частини додатку.

Висновки містять рекомендації щодо доцільності розробки та впровадження системи управління проектами та переваги, які можливо з цього отримати.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Інформаційна система управління проектами» складається із переліку умовних позначень, вступу, трьох розділів, загальних висновків, списку використаних джерел і містить 80 сторінок тексту, 36 рисунків. Список використаних джерел містить 39 найменувань.

Об'єкт дослідження процес управління проектами, необхідні інструменти для членів проектної команди та менеджера

Предмет дослідження - управління проектами та інформаційні системи управління проектами

Мета кваліфікаційної магістерської роботи - систематизація, закріплення та розширення теоретичних знань, їх застосування для побудови інформаційної системи управління проектами

Ключові слова: веб-додаток, проект, система управління, база даних

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

БД – База даних

REST - Representational state transfer

IT – Інформаційні технології

MEAN - абревіатура від MongoDB, Express.js, Angular.js, Node.js

ІУС – інформаційна управляюча система

HTML - HyperText Markup Language

ORM - об'єктно-реляційне відображення

HTTP - HyperText Transfer Protocol

XML - eXtensible Markup Language

SQL - Structured query language

ВСТУП

У сучасному світі інформаційних технологій комп'ютери, IT-системи та Інтернет стали необхідними для багатьох повсякденних завдань. Таким чином, наявність веб-доступу до систем та додатків є обов'язковою для збереження конкурентоспроможності в бізнесі. Найважливішими та мотивуючими факторами для використання програмного забезпечення для бізнесу є підвищення продуктивності та прибутку.

Програмні продукти відіграють значну роль для успіху та розвитку бізнесу, за допомогою таких систем є можливість автоматизувати декілька процесів, які в іншому випадку є складними та тривалими для виконання вручну. Ці програми економлять багато часу та грошей, виконуючи різні завдання за лічені хвилини. В іншому випадку підприємствам доведеться наймати працівників для виконання цих завдань. Вибір правильного програмного забезпечення для бізнесу може допомогти компанії швидко розвиватися.

Зараз існує велике різноманіття програм для різних цілей, наприклад, програмне забезпечення для малого бізнесу може виконувати функції обробки текстів, нарахування заробітної плати, контролю запасів та обліку та багато іншого. Програмне забезпечення в основному відповідає за управління завданнями всієї організації.

Використання програмного забезпечення важливе для звітності про прогрес або відставання в діяльності організації. Це значно підвищує ефективність та темпи розвитку компанії. Програмне забезпечення зменшує навантаження та автоматизує діяльність. Це також допомагає усунути людські помилки, в результаті покращує ефективність роботи.

Існують різні типи програмного забезпечення для бізнесу, які зазвичай використовують компанії. Програмне забезпечення відповідає за виконання різних

завдань. Одним з найважливіших бізнес-програм для будь-якої організації є система нарахування заробітної плати. Це програмне забезпечення важливе для управління платежами в організації для співробітників і підрядників. Це дозволяє розрахувати оплату та податкові відрахування для кожного працівника в організації.

Іншим важливим програмним забезпеченням для компаній є програмне забезпечення для виставлення рахунків. Система повністю автоматизує процес, потрібно лише ввести дані клієнта.

Для компаній, які надають свої послуги клієнтам важливою та необхідною є система управління проектами, оскільки, управління проектами є надзвичайно зручним та корисним інструментом при розробці проектів, а за даними консалтингової компанії Bain & Company, до 2027 року більшість робіт буде виконуватись саме на основі проектів.[2]

Дійсно, досить велика частина сучасного бізнесу в Україні, Європі та в усьому світі проектно-орієнтована. В Україні ця частка наближається до позначки в 50%. Це пов'язано з тим, що все більше компаній орієнтуються на створення принципово нових продуктів або послуг, на досягнення нових результатів у відомих сферах.

Частіше всього менеджер відповідає не за один проект, а постійно керує кількома проектами. Задля успіху проекту потрібно обробляти та керувати великою кількістю інформації, людей, задач, вимог клієнтів. Оскільки існує велика кількість змінних, що впливають на результат та успішність проекту процес управління проектами може бути неорганізованим та хаотичним.

Якщо для розробки проекту не використовується система для управління проектами, є велика ймовірність, що час витрачається на дрібниці. Важко відстежити, хто над чим працює. Співпраця між командами неефективна і поширюється на ланцюжки електронної пошти та програмне забезпечення для чату.

У результаті на один проект використовується більше часу, робота ведеться неефективно, що зменшує конкурентоспроможність компаній.

Програмне забезпечення для управління проектами використовується для планування, організації та розподілу ресурсів між командами. Це допомагає спеціалістам співпрацювати та стежити за ходом проекту, чітко визначаючи завдання та відповідальність, контролювати витрати та час і забезпечує безперебійну співпрацю між зацікавленими сторонами. Програмне забезпечення для управління проектами включає широкий спектр інструментів, які служать різним людям для різних цілей.

Отже, саме від уміння реалізувати проект в компанії залежить успіх всього підприємства. У зв'язку з цим, управління проектами стає актуальною і важливою темою для менеджерів будь-якої ланки.

Виходячи з цього використання системи підтримки прийняття рішень в сфері управління проектами є надзвичайно важливим та зможе полегшити процес прийняття рішень, збільшити ефективність, швидкість та раціональність прийнятих рішень.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ РОЗДІЛ. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

1.1 Поняття управління проектами

Управління проектом передбачає досягнення конкретних попередньо узгоджених цілей в поставлених межах (наприклад час, бюджет) проекту за допомогою різних методів, навичок, досвіду.

Проект – обмежена в часі, ресурсах та вимогах якості унікальна сукупність процесів, направлена на досягнення унікальних цілей та завдань для створення нової цінності (продукту або послуги). [3, 34]

Управління проектом, по суті, спрямоване на створення кінцевого продукту, який внесе певні зміни на користь організації, яка ініціювала проект. Під управлінням мається на увазі ініціювання, планування та контроль ряду завдань, необхідних для доставки цього кінцевого продукту. Результатом проекту може бути фізичний продукт, або ж це може бути нове програмне забезпечення чи наприклад методологія.

Ключовим фактором, який відрізняє управління проектами від простого управління, є те, що воно має кінцевий результат і обмежений період часу, на відміну від управління, яке є постійним процесом. Через це менеджеру потрібен широкий спектр навичок; часто технічні навички, безумовно, навички управління людьми та хороша ділова обізнаність.

Управління проектами важливе, оскільки воно забезпечує конкурентоспроможність компанії, мотивацію команди та дозволяє усунути певні перешкоди, що можуть виникнути без ефективного управління, що допомагає впроваджувати нові продукти чи послуги швидше та якісніше, збільшувати дохід та досягати інших цілей компанії.

Згідно з дослідженням Інституту управління проектами у 2020 році 11,4% доларів, інвестованих в проекти, було витрачено даремно через погану систему управління [29].

Використовуючи ефективні методи управління проектами, організації можуть активно й постійно покращувати свої робочі процеси, щоб уникнути перевитрат і помилок.

Іноді для розроблення, завершення проекту необхідно, щоб команда людей тимчасово працювала разом, щоб зосередитися на конкретних цілях проекту. У результаті ефективна командна робота є основою успішних проектів.

Інститут управління проектами (PMI) розбиває процес управління проектами на п'ять етапів, що показані на схемі рисунку 1.1[29, 35]:

Рис. 1.1 Етапи управління проектом

Керівник проекту, застосовуючи сучасні методології, має створити план розробки та створення проекту, що враховуватиме обмеження проекту: це може

бути обмеження у часі, бюджеті та обсязі, які застосовуються до всіх проектів, проте також варто брати до уваги вимоги замовника (клієнта) та зацікавлених осіб, адже цілі та завдання проекту визначають саме вони.

Для швидшого, простішого управління проектами та для ефективного балансування обмежень, вимог та графіку виконання проектів менеджери часто використовують програмне забезпечення, що допомагає у виконанні проектів. Таке програмне забезпечення називають ***системами управління проектами***, воно допомагає підтримувати, керувати проектами та збільшувати продуктивність команди, що працює над даним проектом.

За допомогою такої системи можливо створювати графік виконання проекту, відслідковування виконання задач, навантаження на працівників, управління ризиками та бюджетом проекту.

1.2 Основні переваги та задачі систем управління проектами, вимоги що до них ставляться

Системи управління проектами виконують функції, необхідні для ефективного планування проектів, управління доступними ресурсами, реагування на проблеми та підтримки всіх зацікавлених сторін проекту. Незалежно від послуг, які пропонує бізнес, вибір системи управління проектами може мати великий вплив на подальший розвиток проекту.

Розглянемо основні переваги систем управління проектами:

1. Планування та створення графіку проекту є надзвичайно важливим аспектом управління проектом незалежно від застосовуваної методології. Завдяки системам можна легко отримати доступ до інформації з минулих проектів, що може бути корисною в поточному проекті.

План проекту, також відомий як ***план управління проектом***, — це документ, який описує, як проект буде виконуватися, відстежуватися,

контролюватися та закриватися. У ньому окреслюються цілі та масштаби проекту, він служить офіційним орієнтиром для команди проекту та зацікавлених сторін.

Він створюється на етапі планування проекту і є компіляцією кількох інших документів. Це більше, ніж просто розклад чи список завдань, хоча він містить і цю інформацію. План управління проектом офіційно затверджується на початку проекту, а потім поступово оновлюється протягом усього часу.

Графік проекту передбачає створення документа, в останній час, як правило, цифрового, який містить детальну інформацію про терміни проекту та організаційні ресурси, необхідні для виконання кожного завдання. [34, 35]

Розклад проекту має бути доступним для кожного члена команди. Його мета — передати важливу інформацію команді, тому вона має бути вичерпною та легкою для розуміння.

Менеджери проекту можуть зручно створити послідовний план управління та розставити пріоритети завдань для успіху проекту. За допомогою програмного забезпечення для управління проектами такі завдання, як розподіл ресурсів, визначення залежностей, встановлення термінів і створення результатів проекту, можна легко виконати в найкоротші терміни.

Більшість систем управління проектами мають функції, які допомагають упорядкувати планування проекту та процес створення графіку проекту.

2. Краща командна робота

Проектні групи іноді складаються з представників різних відділів. Із великою кількістю людей в команді іноді може бути важко зібрати всіх на зустріч або доносити необхідну інформацію до кожного учасника.

Величезною перевагою системи управління проектами є те, що вона робить ефективну співпрацю команди проекту надзвичайно простою. Система зберігає всі комунікації в єдиному місці. До таких даних, як графік проекту та оновлення статусу, можна легко отримати доступ одним клацанням миші, а важливі сповіщення можуть автоматично надсилатися відповідним сторонам.

3. Віддалена робота

Фізичне розташування офісу чи відділу ніяк не обмежують проект. Зараз робітники можуть знаходитись в різних офісах, містах чи навіть країнах, і керувати ними може бути досить проблематично. Завдяки програмному забезпеченню для управління проектами керувати віддаленими проектами стає легко, оскільки ці програми допомагають менеджерам ефективно керувати своїми командами незалежно від їх розташування.

4. Ефективне делегування завдань

Іноді у великому проекті може бути не очевидно хто з членів команди виконує більше/менше завдань, управління великою кількістю людей може бути важким. За допомогою системи управління проектами менеджери можуть легко делегувати завдання проекту членам команди або ж співробітники можуть навіть самостійно забирати наступну задачу з огляду на пріоритети.

Це також простіший варіант для членів команди, оскільки в управлінні завданнями єдине, що їм потрібно зробити, це зайти в систему, переглянути визначені задачі та почати працювати над своїми завданнями відповідно до їх пріоритетів. У більшості випадків ці програми також надсилають автоматичні нагадування про закінчення кінцевих термінів, щоб забезпечити безперебійну та ефективну роботу.

5. Простіший доступ до файлів і спільний доступ

Безпечний доступ до документів і обмін ними дуже важливий. Деякі системи управління проектами пропонують сховище для зберігання даних, де можливо вносити зміни в файли, залишати відгуки та коментувати. Ці програми також ведуть журнал змін, щоб забезпечити прозорість проекту в команді.

6. Простіша інтеграція нових членів

Проекти носять динамічний характер, і іноді стає необхідним залучення нових членів до команди проекту. Проте проекту можуть бути громіздкими, виконуватись протягом довгого часу та містити великий об’єм інформації, тому

додавання нових членів команди, їх навчання може бути складним та довгим процесом. Цей процес можна полегшити за допомогою будь-якого ефективного програмного забезпечення для управління проектами завдяки журналам введення проекту та засобами візуалізації.

7. Ефективне управління ризиками

Системи управління проектами допомагають менеджерам в зменшенні ризиків, оцінюванні інформації.

8. Управління бюджетом

Основна перевага використання програмного забезпечення для управління проектами полягає в тому, що воно дозволяє користувачам ефективно керувати кожним доступним ресурсом. У ході будь-якого проекту шанси на розширення масштабів досить великі.

Система управління проектами може допомогти менеджерам запобігти цьому, оскільки дозволяє їм відслідковувати все за допомогою інструменту звітності проекту в режимі реального часу та запобігти різкому зростанню вартості проекту.

9. Підвищення продуктивності

Ще одна велика перевага системи управління проектами полягає в тому, що вона оптимізує процес прийняття рішень і підвищує продуктивність на роботі, оскільки всі важливі деталі можна знайти в одному місці.

Отже, можна зробити висновок що сновними функціями та задачами, такої системи є:

— Планування проекту
— Колективна співпраця
— Відстеження часу
— Звітність
— Бюджетування проекту

— Надання інформації

Вимоги можуть відрізнятися в залежності від того, як використовується інструмент. Найбільш поширеними є:

— планування різних подій, що залежать одна від одної;

— планування розкладу роботи співробітників і керування ресурсами;

— сортування завдань залежно від термінів їх завершення;

— керування декількома проектами одночасно.

1.3 Огляд існуючих систем управління проектами

Системи управління задач можуть виконувати одне кокретне завдання, так і низку задач із управління проектами. Нині на ринку є багато прикладів таких систем, розглянемо переваги та недоліки декількох із них:

1. ***Wrike*** — це комплексне рішення для управління проектами. Він створений для забезпечення повної видимості та прозорості проектів і завдань, покращення співпраці у команді.

Wrike має можливість додавати декілька канбан дошок, інтерактивні діаграми Ганта, містить список задач, графік роботи та можливість преглянути робоче навантаження на працівника. Надає можливість поширення та завантаження файлів та можливість для спілкування між членами команди в реальному часі.

Переваги:

Аудит та робота з вимогами, відстежування робочих процесів та управління навантаженням на працівників: в реальному часі модна відстежувати, на якому етапі знаходяться задачі та хто з працівників їх виконує. У системі такоє є можливість друкування звітів по окремим працівників із інформацією про витрачений час на завдання.

Є потужні інструменти звітності даних та можливість додавати функції для бізнес-аналітики та більш складної звітності.

Є інтеграція із Outlook і Microsoft Teams показано на рисунку 1.2

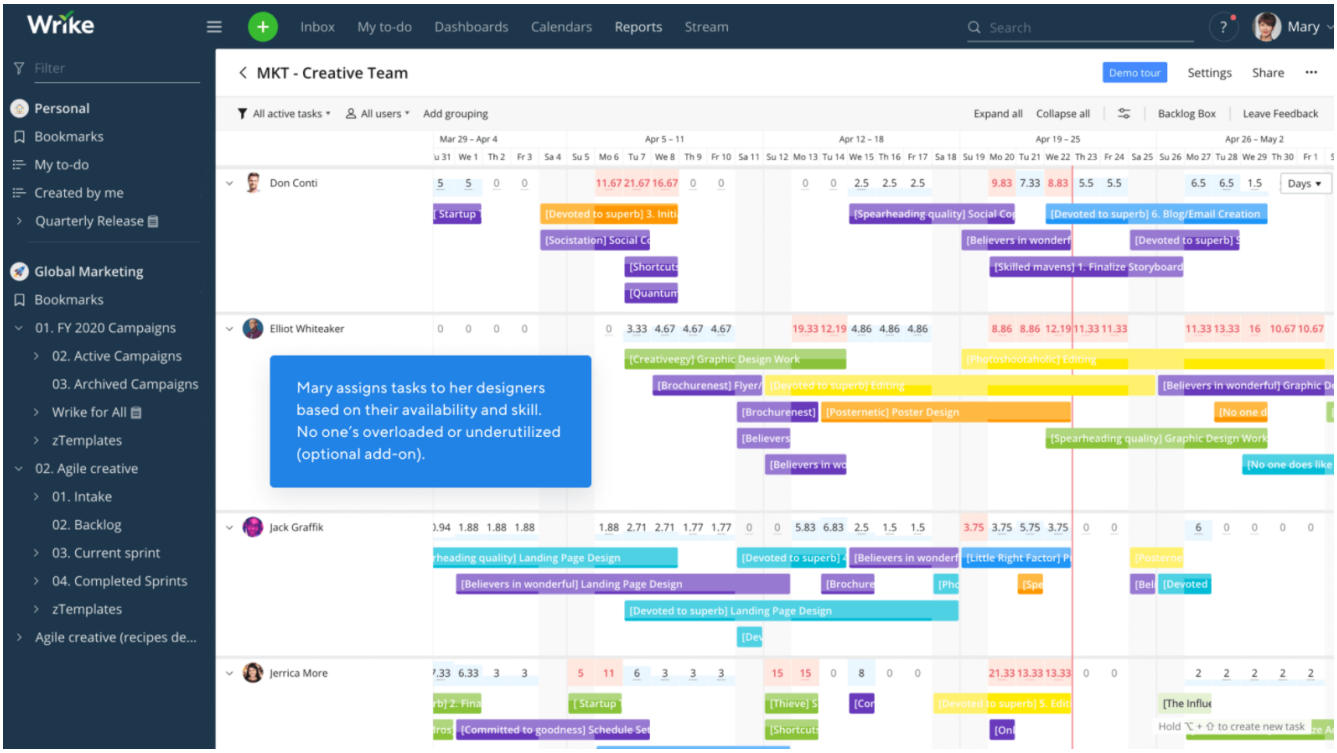


Рис. 1.2 Інтерфейс Wrike

Недоліки:

Проте інтерфейс користувача не такий інтуїтивно зрозумілий, як міг би бути, а отже потребує часу на навчання працівників роботі із системою. Налаштування програми також досить складне.

Ціна:

— Доступний безкоштовний план для 5 користувачів.

— Професійний тарифний план для 5, 10 та 15 користувачів становить 9,80 доларів США за користувача на місяць, оплачується щорічно.

— Бізнес-план для 5-200 користувачів становить 24,80 дол. США за користувача на місяць, оплата щорічно.

— План Enterprise для 5-Unlimited користувачів вимагає зв'язку з командою продажів Wrike та уточнення ціни.

— Для планів Professional, Business і Enterprise доступна 14-денна безкоштовна пробна версія.

2. ***LiquidPlanner*** — одна з провідних програм для управління проектами онлайн на ринку. Він використовує інтелектуальну технологію планування, щоб допомогти командам краще виконувати свою роботу. LiquidPlanner допомагає командам складати графік роботи, відстежувати хід роботи, приймати кращі рішення та забезпечувати своєчасне виконання проектів.

Це програмне забезпечення ідеально підходить для середніх і великих організацій, які хочуть, щоб їхні команди працювали ефективніше, рисунок 1.3.

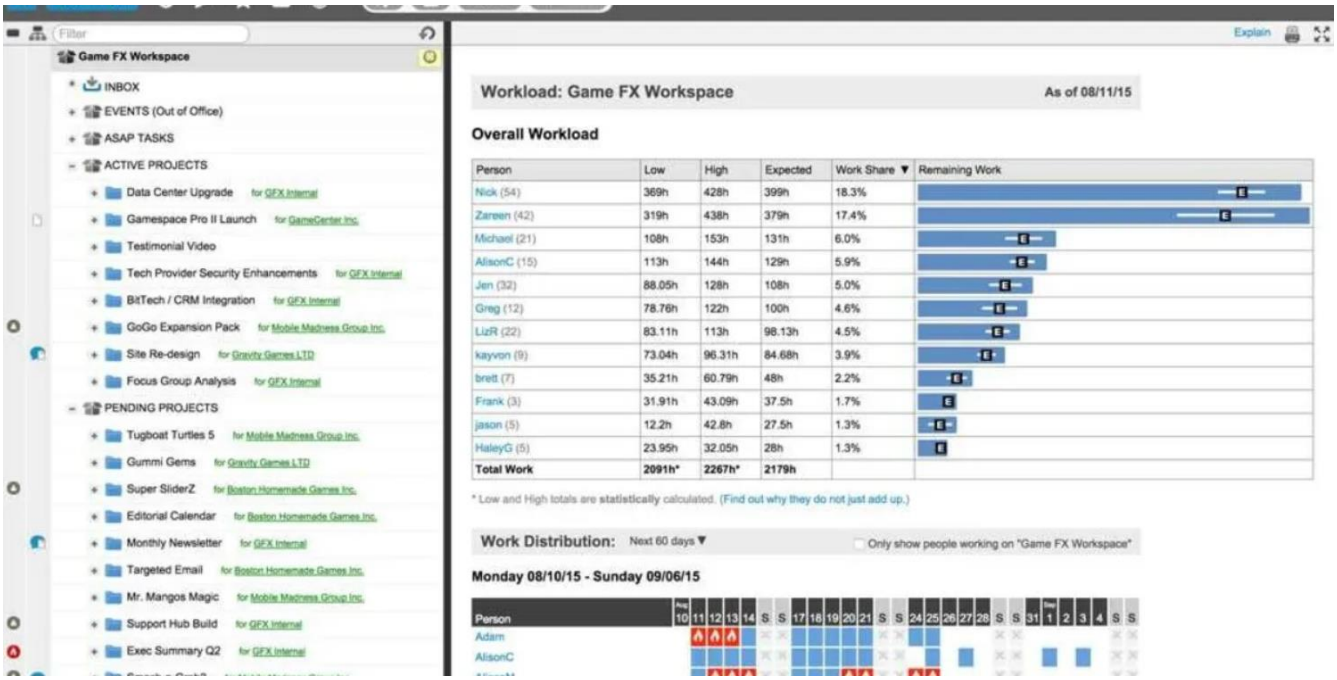


Рис. 1.3 Інтерфейс LiquidPlanner

Переваги:

При додаванні нового користувача є можливість вказати рівень доступу до даних, команду тощо або відправити запит на приєднання до команди.

Програма пропонує різні варіанти візуалізації проекту: канбан дошка або діаграма Ганта.

Учасникам команди є можливість додавати та відстежувати свою відпустку, лікарняні чи вихідні, а менеджер може оцінити як це вплине на робочий процес та відповідно перепризначити завдання.

Також є функція відстеження часу: в браузері запускається таймер та автоматично записує витрачений час на роботу. Це може бути корисним як для тайм-менеджменту, так і для виставлення рахунків клієнту. Також цей витрачений час можна вводити вручну, якщо він не відстежувався автоматично за допомогою інструменту.

Недоліки:

Потребує певного часу на навчання перед початком роботи із системою, інтерфейс не інтуїтивно зрозумілий.

Ціна:

Професійний тарифний план становить 45 доларів США за користувача на місяць, оплачується щорічно. Для тарифного плану Professional доступна 14-денна безкоштовна пробна версія.

3. ***Trello*** — побудований на основі канбан-дошок, є зручним програмним забезпеченням для візуалізації проекту, простий у використанні, але орієнтований лише на керування завданнями. Інтерфейс можна побачити на рисунку 1.4.

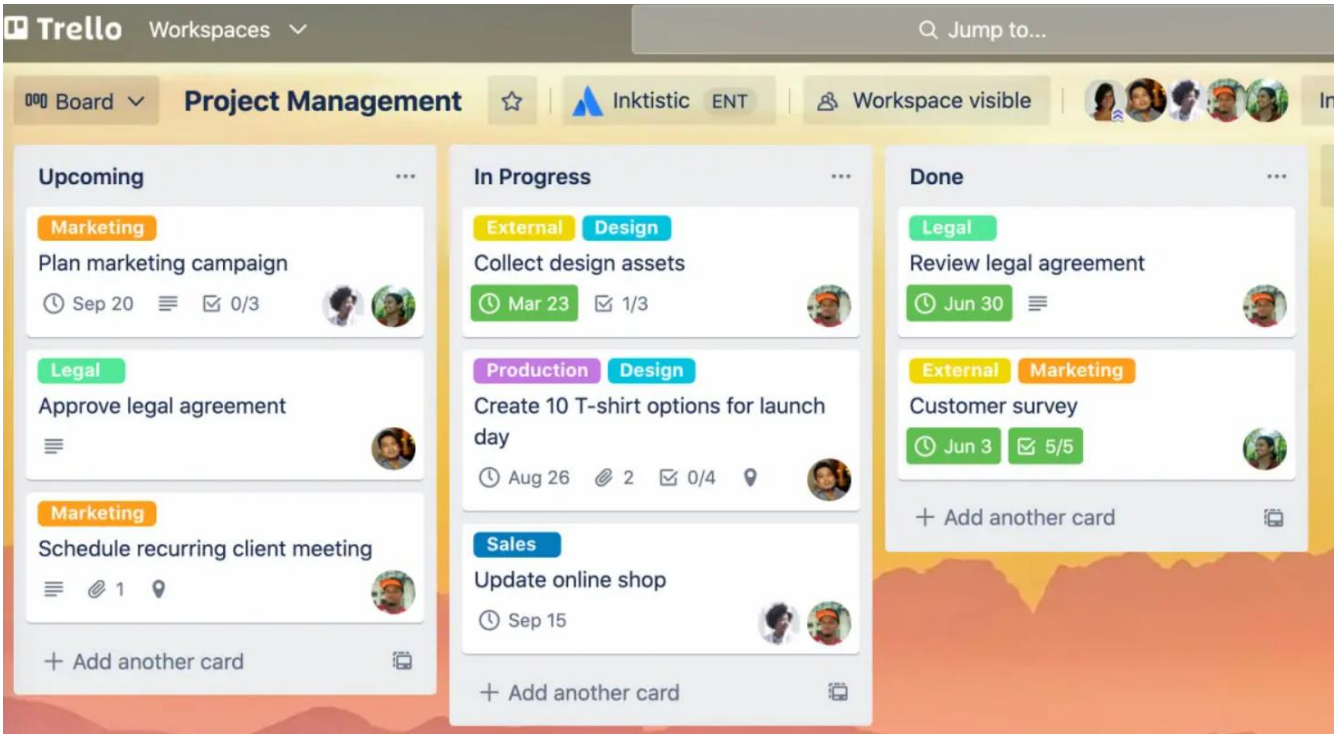


Рис. 1.4 Інтерфейс Trello

Переваги:

Картки із задачми містять в собі всю необхідну інформацію без потреби відкривати їх, є можливість додавати коментарі, встановлювати терміни виконання на кожне завдання, прикріплювати файли та створювати власні колонки.

Trello інтегрується зі Slack, Evernote, Dropbox і Google Drive та іншими відомими програмами. Це дозволяє синхронізуватись з програмами, які вже використовуються на проекті.

Додаток є популярним серед фрілансерів, нових стартапів та рекламується як система для планування сімейного відпочинку – тобто вона дуже проста у використанні.

Недоліки:

Trello не створений для складних проектів, з великою кількістю процесів, потреб в автоматизації, аналітиці та великою командою. Якщо проект громіздкий, то Trello скоріше всього не матиме потрібних інструментів.

Ціна:

Доступний безкоштовний тарифний план з необмеженою кількістю карток і обмеженням 10 МБ на вкладення.

Business та Enterprise план оплачуються за одного користувача на місяць.

Для бізнес-плану доступна безкоштовна 14-денна пробна версія.

4. ***monday.com*** найкраще підходить для організації робочих процесів між командою. Використовується такими компаніями як BBC Studios, L'Oréal Paris, Adobe та Deezer, рисунок 1.5.

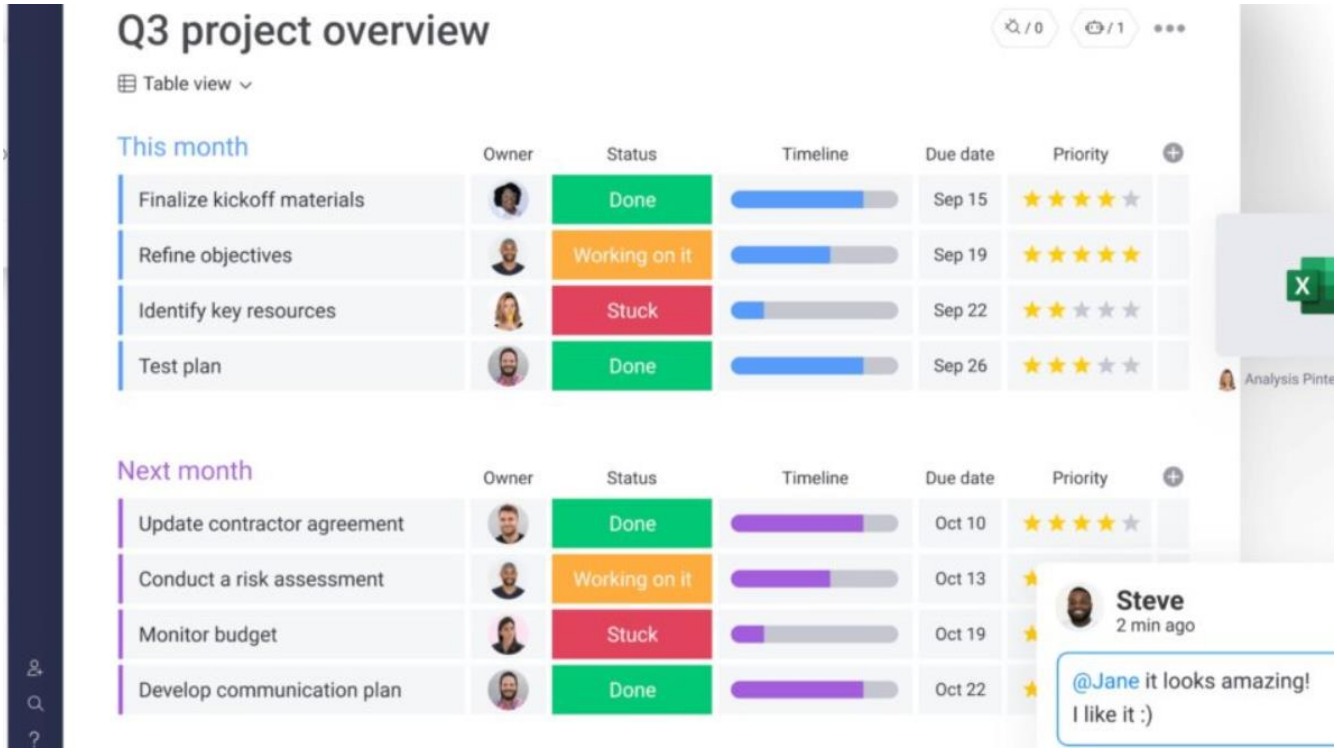


Рис. 1.5 Інтерфейс monday

Переваги:

Система містить наступні функції:

— управління ресурсами

— аналіз витраченого на завдання часу

— функції звітності

— чат у реальному часі, для ефективнішої командної роботи

Наприклад користувач може завантажувати, прикріплювати файли до картки із завданням, додавати коментарі. Система також пропонує панель звітів по проектам, що автоматично збирає та аналізує інформацію із дошок.

Найкраще підходить саме для відстеження робочого процесу та організації командної роботи, також підходить для стартапів і власників малого бізнесу. А також для індивідуального використання, наприклад фрілансерам.

Інтеграції monday.com включають програми для керування проектами, такі як Slack, Google Drive, Gmail, Google Calendar, Jira, GitHub, Trello, Dropbox та інші. Безкоштовна для 2 користувачів.

Недоліки:

Не пропонує набору інструментів для аналітики, управління бюджетом проекту, мобільна версія не зручна для використання.

Ціна:

Базовий план становить 25 доларів США за користувача на місяць, рахунок виставляється щорічно.

Стандартний план становить 39 доларів США за користувача на місяць, рахунок виставляється щорічно.

Професійний план становить 59 доларів США за користувача на місяць, рахунки виставляються щорічно.

План Enterprise вимагає зв’язку з Monday.com для уточнення.

Без кредитної картки доступна 14-денна безкоштовна пробна версія з необмеженою кількістю картор та користувачів.

5. ***Zoho Projects*** – хмарний інструмент для керування проектами. Допомагає планувати роботу, відстежувати її та співпрацювати на відстані, дошку проекту можна побачити на рисунку 1.6.

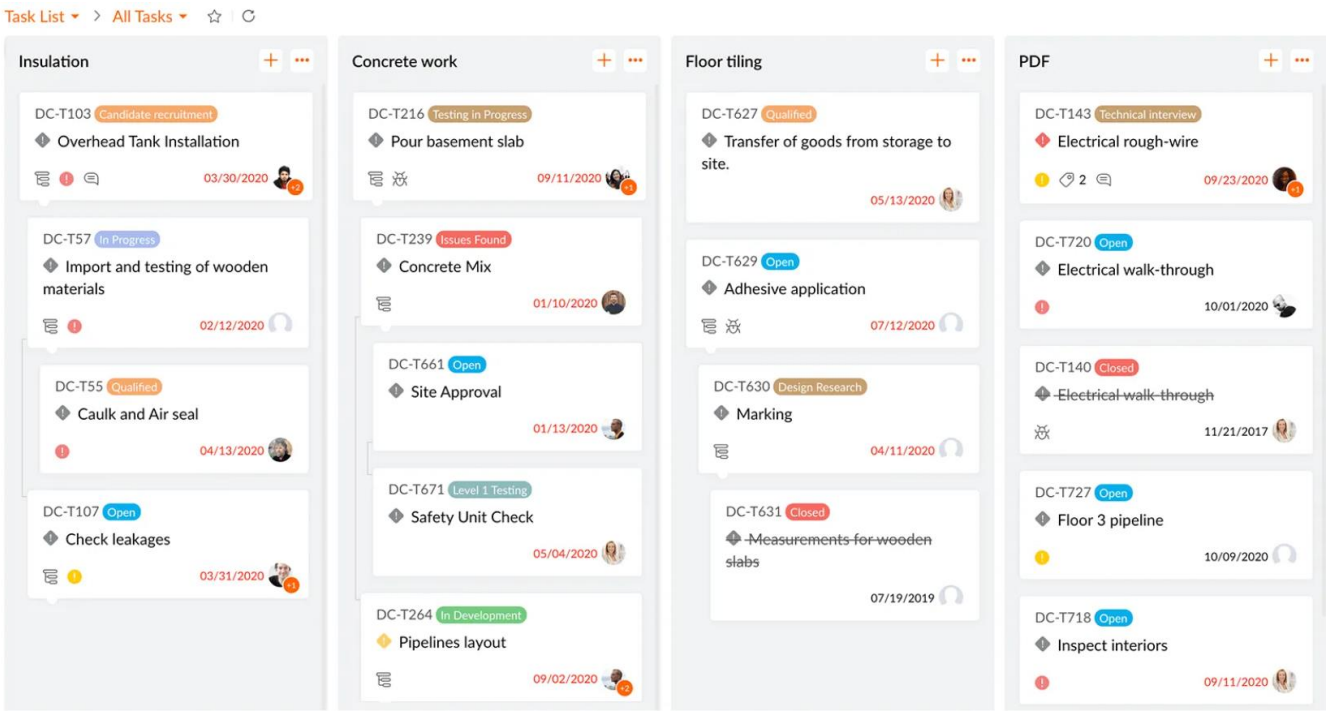


Рис. 1.6 Інтерфейс Zoho Projects

Переваги:

За допомогою діаграм Ганта будується план проекту та відстежується виконання завдань та прогрес. Статистика, що надає інформацію про заплановані терміни виконання та реальний прогрес.

Є можливість відстежувати робочі години по задачам: оплачувані та неоплачувані години, що є корисним для виставлення рахунку замовнику проекта. Також існує пряма інтеграція з Zoho Invoice, яка автоматично створюватиме рахунки з табелів обліку робочого часу.

Є інтеграція із Excel і Slack.

Ціна:

Доступний безкоштовний план для 3 користувачів і 2 проектів.

Стандартний тарифний план для 6-10 користувачів становить 2,50 дол. США за користувача на місяць, що оплачується щомісяця, і 3 долари США за користувача на місяць.

Також є платні експрес, преміум та Enterprise плани, оплачуються також кожного місяця.

Для всіх платних планів доступна 10-денна безкоштовна пробна версія.

6. ***ClickUp*** розроблений для управління проектами як із маленькою командою, так і з великою кількістю співробітників.

ClickUp — це програмний інструмент для управління проектами з потужними функціями для керування та виконання проектів на одній платформі.[8]

Система пропонує наступний функціонал:

— планування проектів

— відстежування прогресу виконання по завданням

— управління ресурсами

— організація командної роботи

— списки задач, підзадачі, шаблони задач

— фільтрація, сортування, пошук задач

Є можливість переглядати завдання за допомогою діаграми Ганта, календаря, часової шкали.

ClickUp також включає функції для створення, спільного доступу та спільного редагування Wiki сторінок і документів. Користувачі мають можливість залишати коментарі під документами, задачами, спілкуватись із іншими членами команди у реальному часі. Функції звітності включають можливість створювати спеціальні інформаційні панелі, а також шість вбудованих типів звітів.

ClickUp пропонує нативну інтеграцію зі Slack, G Suite, Dropbox та багатьма іншими інструментами, а також понад 1000 інтеграцій через Zapier, рисунок 1.7.

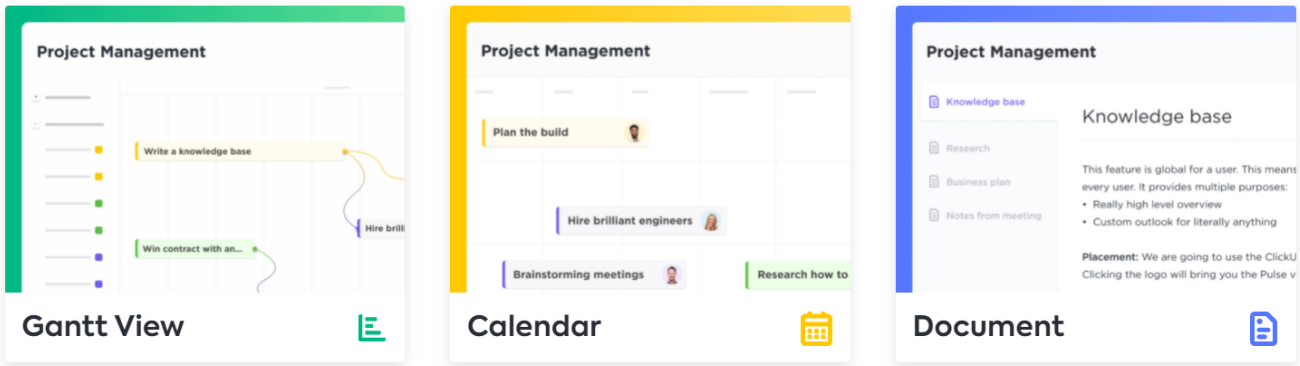


Рис. 1.7 Інтерфейс ClickUp

Переваги:

— Безкоштовний план надає можливість додати необмежену кількість учасників

— Немає обмежень на збереження файлів, незалежно від плану

Недоліки:

— Для користування додатком необхідно буде оформити платну версію, оскільки права гостя дуже обмежені

— Набір звітів залежить від плану, не всі доступні навіть на платних версіях

— Складний процес налаштування додатку

Найпопулярнішими системами в Україні є:

Worksection – система розроблена українською командою, в якій представлені такі функції: планування проектів, інструменти для спілкування з клієнтами та фрілансерами, управління командою, а саме можливість використовувати часову ставку та відстеження часу, що витрачається на окремі задачі[8]

Atlassian JIRA – система, що використовується для відстеження всіх задач та помилок у проекті, призначення конкретних завдань на учасника команди, організація спілкування з користувачами. [8].

Basecamp – система розроблена одноіменною американською компанією для організації командної роботи та планування, управління задачами.

1.4 Обгрунтування вибору веб-додатку, поняття та переваги веб-рішення

Системи значно трансформувалися за останні роки, і завдяки значним поліпшенням безпеки та технологій системи все частіше і частіше створюються саме як веб додатки, рідше є можливість використовувати як мобільний-додаток, так і десктопне програмне забезпечення.

Веб-додаток - це програма клієнт-сервер. Це означає, що він має клієнтську і серверну сторону. Термін "клієнт" відноситься до програми, яку користувач використовує для запуску та користування програмою – інтерфейс (фронт-енд). Це частина клієнт-серверного середовища, де багато комп'ютерів обмінюються інформацією. Наприклад, у випадку з базою даних клієнтом є програма, за допомогою якої користувач вводить дані. Сервер - це програма, яка зберігає інформацію.

Користувачі можуть отримати доступ до програми з будь-якого комп'ютера, підключеного до Інтернету або Інтранету, замість того, щоб використовувати програму, встановлену на окремому комп'ютері. Веб-пошта, така як Outlook, є одним із найкращих прикладів популярної веб-програми, яка виконує ті ж функції, що й традиційна настільна програма.

У порівнянні з настільними додатками, веб-додатки надають цілий ряд переваг для бізнесу. По-перше це можливість доступу до програми будь та будь-коли без прив'язки до одного комп'ютера чи іншого мобільного пристрою. Використання веб-програм зазвичай називають програмним забезпеченням як послугою (SaaS), де програми працюють у віртуальному хмарному середовищі; наприклад, впровадження хмарних платформ електронної пошти дозволило

користувачам отримувати доступ до електронної пошти в дорозі, не встановлюючи клієнта електронної пошти для комп’ютера. Додатки SaaS дають відчутні переваги для бізнесу в порівнянні з локальним програмним забезпеченням.

Розробники кодують веб-додатки двома мовами. Веб-додаток зазвичай використовує для роботи комбінацію сценаріїв на стороні сервера та клієнта. Сценарій на стороні сервера займається зберіганням та отриманням інформації та пишеться, наприклад, на Python або Java. Розробники програмують на стороні сервера для створення, зберігання та обробки даних, які використовуватиме веб-додаток та з якими працюватиме клієнт на інтерфейс-частині додатку. Для клієнтської програми потрібні такі мови, як JavaScript, каскадні таблиці стилів (CSS) і HTML5. Для виконання програми використовується браузер. Програма на стороні клієнта займається представленням інформації користувачеві.

Розглянемо переваги більш детально:

— *Міжплатформна сумісність*. Більшість веб-додатків набагато більш сумісні між різними платформами, ніж традиційне встановлене програмне забезпечення. Зазвичай мінімальною вимогою є веб-браузер, яких в даний час існує великий вибір для клієнтів. (Internet Explorer, Firefox, Chrome, Safari – це лише деякі з них). Тож, можна з точністю сказати, що проблем із доступом не виникне незалежно від операційної системи. Це може бути Windows, Linux чи Mac OS, всеодно є можливість запустити веб-додаток.

— *Більш керовані.* Системи розробки веб-додатків необхідно встановлювати лише на сервері, що пред'являє мінімальні вимоги до робочої станції кінцевого користувача. Обслуговувати та оновлювати систему набагато простіше, будь-які оновлення клієнта можна легко розгорнути через веб-сервер (при цьому від самого клієнта нічого не потребується).

— *Більше можливостей для розгортання*. Завдяки керованості та міжплатформній підтримці розгортання веб-додатків для будь-якої платформи в будь-якому типі робочого середовища є простішим. Все що потрібно для надання

доступу клієнту це надіслати користувачеві адресу веб-сайту для входу та впевнитись в його доступі до Інтернету. Таким чином, це дозволяє розширити доступ до різних систем, оптимізувати процеси та покращити спілкування та отримання інформації, надаючи доступ до клієнтів, постачальників та будь-яких третіх сторін.

— *Захищені дані в реальному часі*. Більші і складні системи несуть більше даних і мають окремі системи та джерела даних. У веб-системах ці системи та процеси часто можна консолідувати, зменшуючи потребу в окремих системах. Веб-додатки забезпечують додатковий рівень безпеки, позбавляючи доступу до даних і внутрішніх серверів.

— *Менші витрати*. За допомогою веб-додатків можна значно знизити витрати завдяки зменшенню витрат на підтримки та обслуговування, меншим вимогам до системи кінцевого користувача та спрощеній архітектурі. Подальша оптимізація бізнес-операцій за рахунок використання веб-додатків часто дозволяє отримати додатковий дохід.

Отже, веб-додатки можуть запропонувати конкурентні переваги в порівнянні з традиційними програмними системами, дозволяючи підприємствам оптимізувати інформацію та процеси зі зниженими витратами.

РОЗДІЛ 2 АНАЛІТИЧНИЙ РОЗДІЛ. СТВОРЕННЯ МОДЕЛЕЙ, РОЗГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ УПРАВЛІННЯ ПРОЕКТАМИ

2.1 Побудова контекстної IDEF0-діаграми

У систему управління проектами входить ряд послідовних дій:

— визначення і формування вимог до проекту;

— формування максимально чітких і зрозумілих цілей;

— встановлення і реалізація комунікації між задіяними в проекті сторонами;

— створення та дотримання графіку виконання задач та цілей проекту

— врегулювання проектних обмежень: зокрема бюджету, ресурсів, ризиків, дедлайнів, якості;

— спілкування з командою, врахування їх потреб/побажань/очікувань і корекція існуючих планів відповідно до отриманих матеріалів.

Отже, система управління проектами має допомагати як менеджерам проекту так і членам команди в ініціації проекту, плануванні та створенні графіку роботи по проекту, виконанні задач та контролі за виконанням, управління ресурсами: сюди може входити бюджет, виділений час, людські ресурси та інше та звершення проекту, створення звітності.

Саме ретельне планування, організація завдань і проектних складових, забезпечення необхідними ресурсами і контроль дієвості обраної стратегії — це і є основна мета управління проектами, що виконується з перспективою досягнення поставленої цілі проекту і кожний етап породжує низку рішень, які необхідно прийняти для успішного виконання та завершення проекту в зазначених межах.

IDEF0, складена абревіатура ("Icam DEFinition for Function Modeling", де ICAM є абревіатурою від "Integrated Computer Aided Manufacturing"), є методологією моделювання систем/процесів для опису функціональних можливостей, яка пропонує мову функціонального моделювання для аналізу, розробки, реінжинірингу та інтеграції інформаційних систем, бізнес-процесів або аналіз програмної інженерії. [30]

Моделювання є центральною частиною всіх видів діяльності, які ведуть до розгортання якісного програмного забезпечення, оскільки моделювання дає графічне представлення системи, яку потрібно побудувати; сприяє успішній організації програмного забезпечення; є перевіреною та загальноприйнятою технікою.

Модель може бути структурною, підкреслюючи організацію системи, або може бути поведінковою, підкреслюючи динаміку системи.

За допомогою моделювання можливо краще оцінити досліджуваний процес та отримати розуміння того, як він відбувається, Виходячи з цього побудуємо контекстну IDEF0-діаграму та розглянемо два рівня декомпозиції.

Життєвий цикл проекту є концепцією, що розглядає проект як послідовність фаз, подій та етапів, кожна з яких має свої часові межі. При розробці проекту розглядають 6 етапів:

1. *Ініціювання* (старт проекту) – на цьому етапі формується команда, визначається суть та цілі проекту, його обмеження: бюджет, часові рамки виконання, ризики.

2. *Планування* — на цьому етапі необхідно створити план та графік проекту, визначити критерії при виконанні яких проект вважатиметься успішно виконаним, є одним із найважливіших етапів з огляду на організацію роботи.

Для створення плану проект розбивають на частини, кожна з яких розбивається на задачі, а ті в свою чергу на підзадачі, для кожного зазначається термін виконання та необхідні ресурси. Панування також включатиме в себе

внесення поправок, які можуть виникати в процесу роботи, створення додаткових підзадач.

3. *Розробка*. Цей етап передбачає безспосереднє виконання задач з у рахуванням всіх обмежень та ресурсів. У системі управління проектами використовуються багато інструментів для ефективнішої роботи із задачами, наприклад для делегації задач, тайм-менеджмент та спілкування членів команди у реальному часі. Саме на цьому етапі може виникати потреба в створенні додаткових підзадач.

4. *Оцінка і аналіз результатів* — на цьому етапі частіше всього створюють звіти по виконаній роботі, проводиться тестування, наскільки в результаті виконання проекту було досягнено всіх зазначених цілей. Тобто на цьому етапі робиться контрольна перевірка виконаної роботи і обов'язково зберігаються вихідні дані, задіяні інструкції і регламенти. Це потрібно для команди, яка буде підтримувати створений проект.

5. *Введення в експлуатацію* – на даному етапі проект поширюється для користувачів для використовування. На перших етапах збираються відгуки.

6. *Підтримка проекту* – в разі виникнення будь-яких проблем чи запитань з боку клієнта чи користувачів або помилкової роботи проекту. Також з часом додання нової функціональності.

Саме ці етапи розглянемо, використовуючи моделювання.

Опис вхідних та вихідних даних

Вхідними даними для розроблюваної системи будемо вважати вимоги до проекту, цілі проекту, що необхідно досягти, обмеження проекту та ресурси.

Вихідними даними є сам проект та проектна документація.

Побудова діаграми потоків даних

Проект має розроблятись на основі поставлених до нього цілей та проблем, які він повинен буде вирішувати. Зазвичай управлінням проекту займається безспосередньо менеджер проекту, виконує роботу - проектна команда. За

допомогою системи стає можливим ефективне використання ресурсів: як людських, так і виділених матеріалів (якщо такі є), бюджет, обмеження проекту та технології.

Елементи моделі:

— База даних

— Вимоги та цілі проекту

— Команда та менеджер проекту

— Обмеження проекту

— Ресурси

На виході отримаємо сам проект та проектну документацію. Розроблену контекстну діаграму зображено на рисунку 2.1.

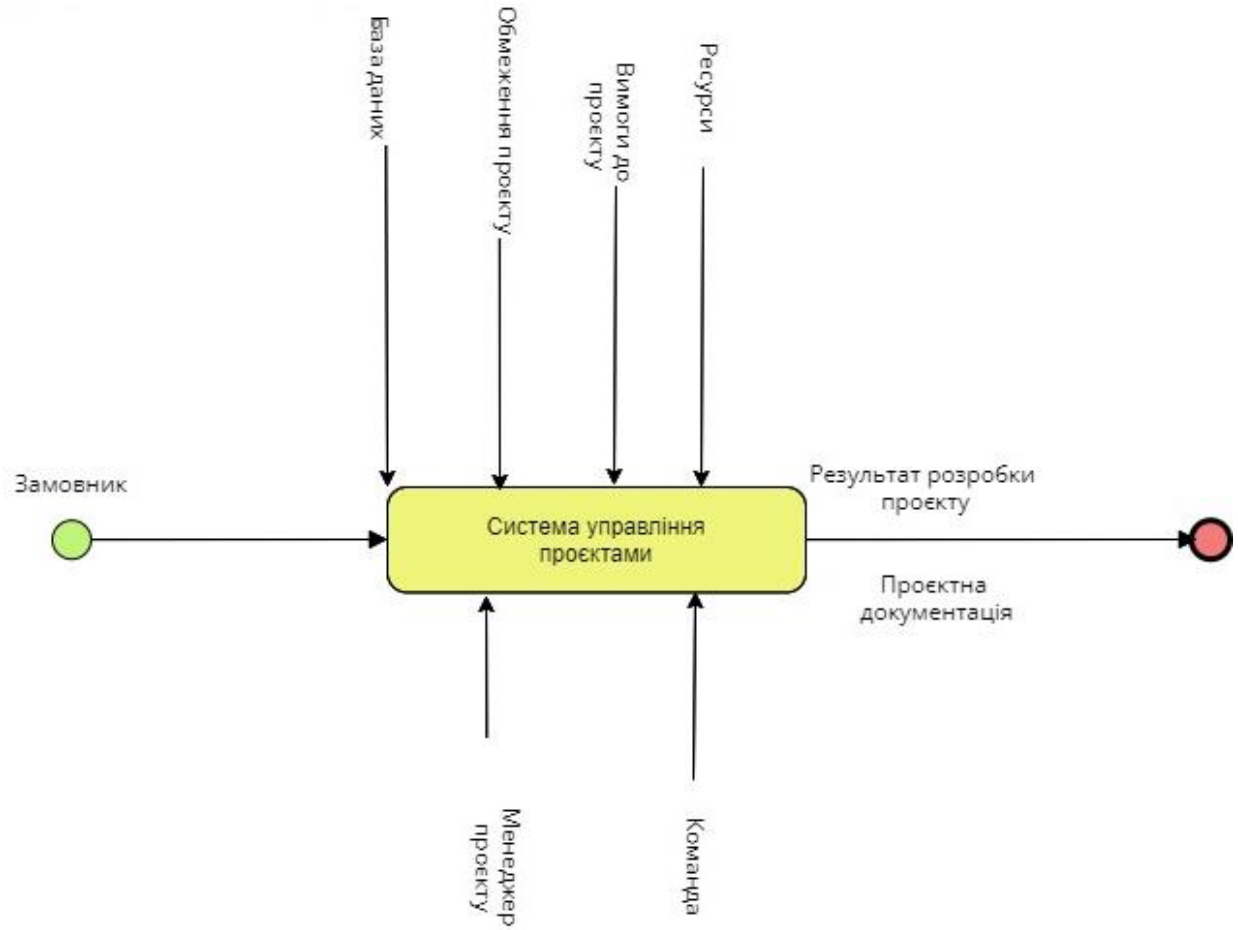


Рис. 2.1 Контекстна діаграма

На даній контекстній діаграмі відображені основні вхідні та вихідні дані. Для більш детального розгляду системи створимо діаграму потоків даних.

Діаграма потоків даних складається з наступних компонентів:

1. Зовнішня сутність - матеріальний предмет або фізична особа, що представляє собою джерело або приймач інформації. Для даної системи розглянемо базу даних, як приймач інформації. Замовник – як особа, що ініціювала проект, менджер та команда – планування та кионання проекту, контролем займається менеджер проекту.

2. Процес - перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму. Виділимо наступні процеси: ***ініціювання проекту, планування проекту, розробка, контроль та інтеграція проекту.***

3. Накопичувач даних - абстрактний пристрій для зберігання інформації, яку можна в будь-який момент помістити в накопичувач і через деякий час витягнути. Виділимо ***беклог***. Діаграма представлена на рисунку 2.2.

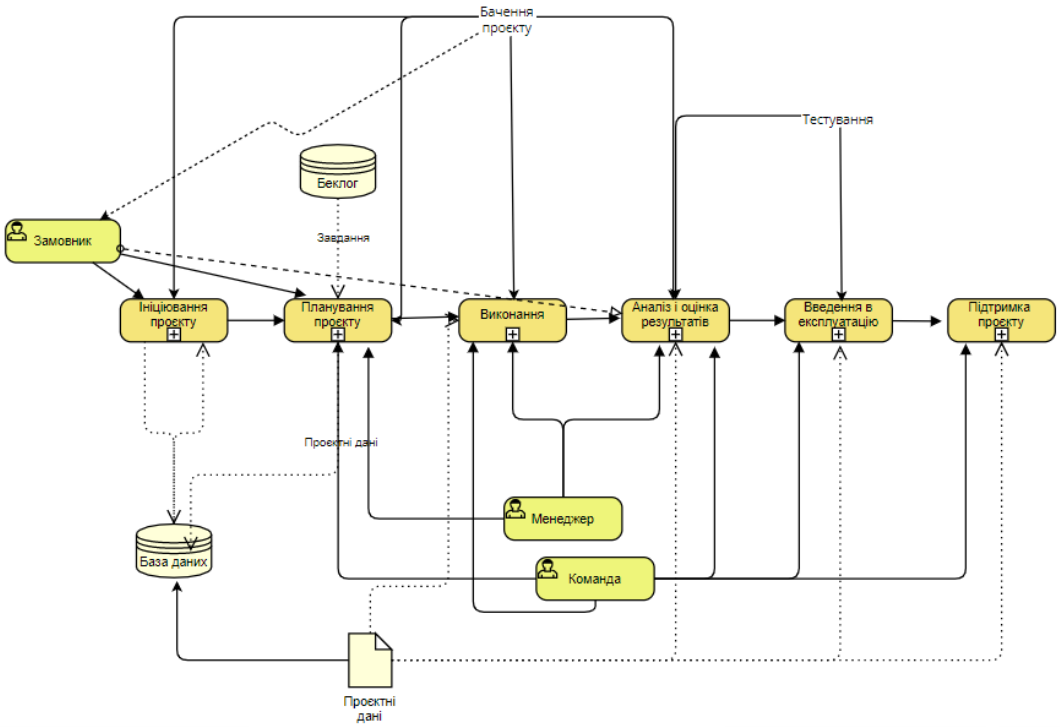


Рис. 2.2 Декомпозиція контекстної діаграми

Опишемо дану діаграму для системи управління проектами більш детально.

Першим кроком є ініціювання проекту замовником: визначається суть, обмеження, вимоги та цілі проекту на основі яких буде відбуватись планування проекту, складання графіку, формується проектна команда, назначається відповідальна особа (менеджер). На цьому етапі характерні наступні рішення, прийняття яких може бути ефективнішим із використанням інформаційної системи:

— про ініціювання проекту;

— визначення цілей проекту при досягненні яких проект можна буде вважати успішним;

— визначення необхідних ресурсів;

— визначення цільової аудиторії;

— визначення обмежень проекту;

— оцінка ризиків проекту;

— про технічну здійсненність проекту;

— створення бізнес плану.

Наступним кроком є планування проекту (див. рисунок 2.3). На даному етапі необхідно розглянути можливі варіанти досягнення цілей та визначити накращий із запропонованих, розробити план та графік проекту, визначитись із часовими та бюджетними рамками, розподілити ресурси ефективно на протязі всього виконання проекту.

Етапи на етапі планування проекту можуть включати наступне:

— Створення плану проекту: визначення часових рамок, включаючи етапи, завдання, які необхідно виконати, та можливі обмеження

— Розрахунок бюджету та створення фінансового плану. Оцінка витрат, щоб визначити, скільки витратити на проект, щоб отримати максимальну віддачу від інвестицій

— Управління ресурсами, створення проектної команди внутрішніх і зовнішніх кадрів, забезпечення необхідними інструментами (програмне забезпечення, апаратне забезпечення тощо) для виконання своїх завдань

— Оцінити можливі ризики та перешкоди для завершення проекту: визначення можливих проблем та розроблення плану їх запобігання або пом’якшення

Наступним етапом є безпосереднє виконання проекту. На етапі виконання проекту можуть включати наступне:

— Створення завдань та організація робочих процесів, розподіл завдань по членам команди

— Інструктаж членів команди щодо завдань: обговорення та пояснення завдань із вказівками як саме вони мають бути виконані, в разі необхідності організація навчання

— Спілкування з членами команди, клієнтами та вищим керівництвом

— Контроль якості роботи

— Управління бюджетом, відстеження витрат

Наступний етап – контроль якості виконання та оцінка прогресу. На цьому етапі найважливішим є перевірка відповідності проекту поставленим вимогам.

Далі йде введення проекту в експлуатацію, що передбачає поширення проекту між користувачами, фінальне тестування (регресія додатку) та збір відгуків від користувачів.

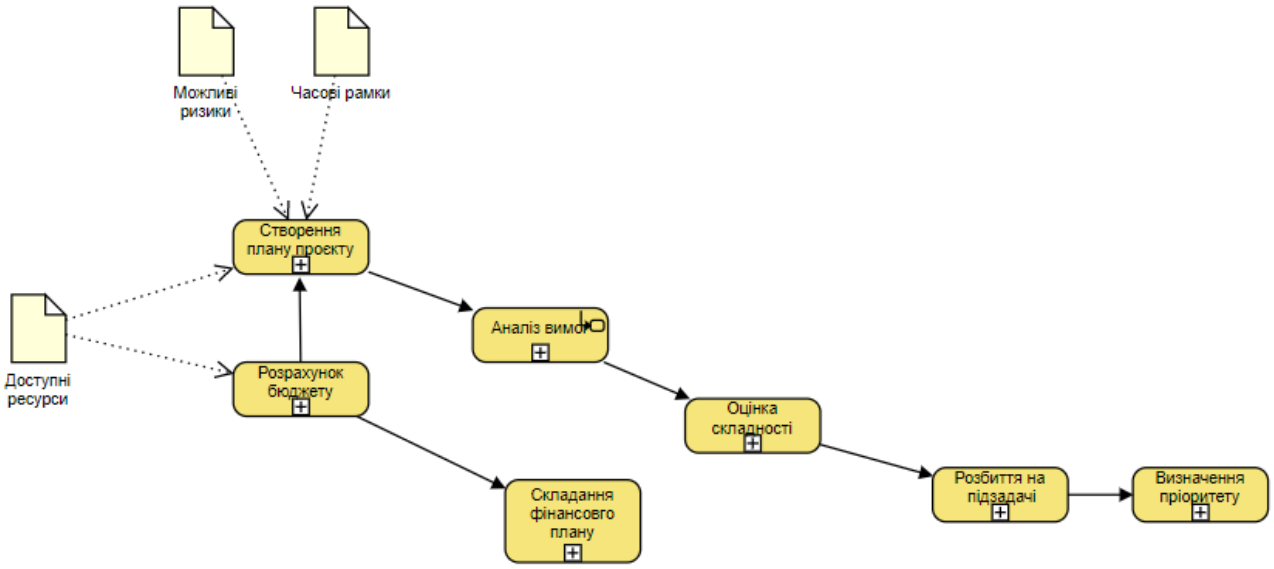


Рис. 2.3. Декомпозиція кроку «Планування»

На цьому етапі (рисунок 2.3) результатом має бути графік проекту, мають бути визначені пріоритету задач, проаналізовані вимоги щодо їх виконання та необхідні ресурси. Визначення складності задачі в якомусь визначеному форматі (може бути що завгодно: бали, розмір одягу, кількість чашок кави). Розглянемо також діаграму IDEF3, рисунок 2.4.

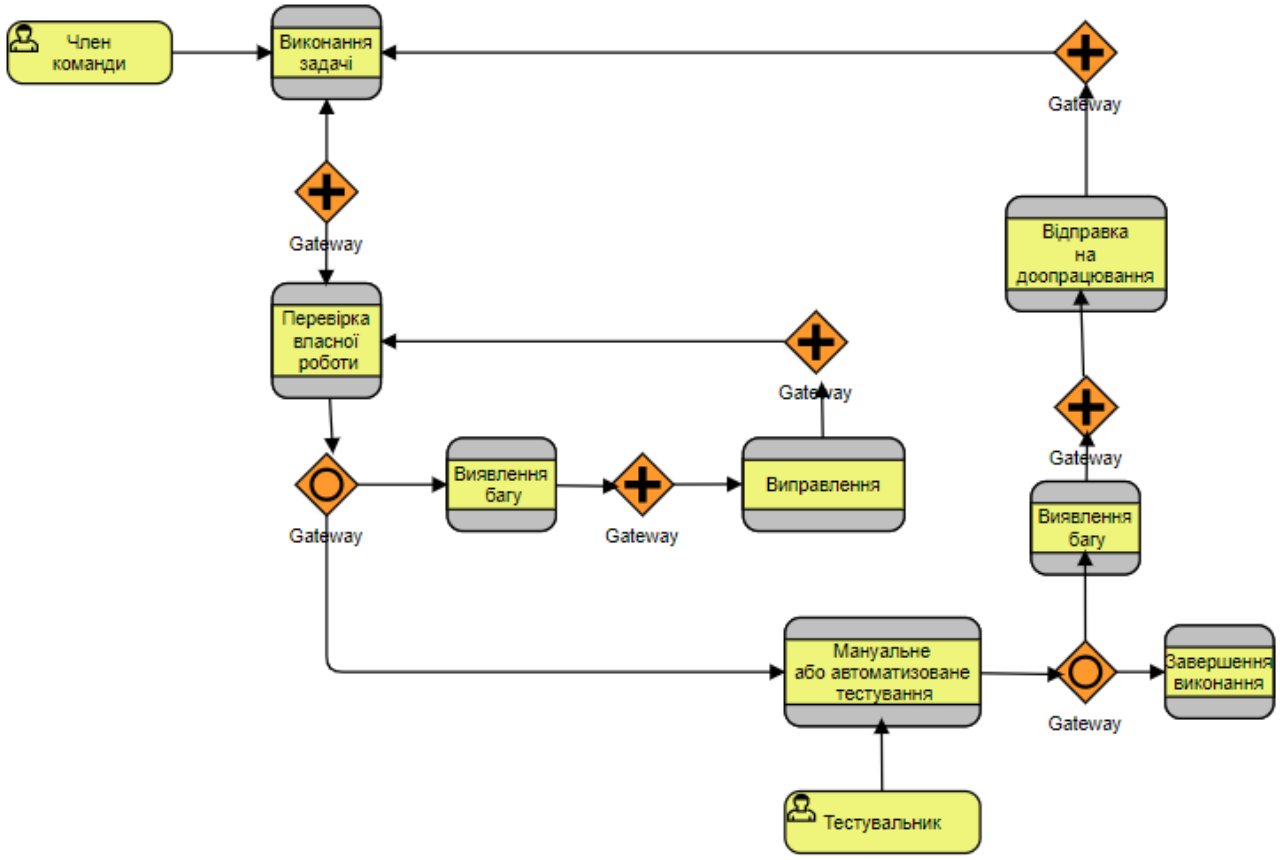


Рис. 2.4. Діаграма IDEF3 для кроку «Розробка»

Дана діаграма зображує процес розробки проекту в контексті однієї задачі.

Член команди бере задачу на розробку, виконує її та самостійно перевіряє результат своєї роботи. На даному кроці є два варіанти розвитку подій: або було знайдено помилку в роботі і в такому разі розробник має її виправити та знову перевірити правильність виконання задачі, або ж помилоку не було знайдено.

Лише в разі, якщо задача виконана повністю із поставленими вимогами вона переходить на наступний етап. Далі відбувається тестуввання виконаної задачі, на даному етапі є два можливих варіанта: виявлення багу і в такому разі повернення задачі розробнику для виконання всіх попередніх дій або ж завершення даної задачі.

2.2 Опис використовуваних технологій

Для розробки системи управління проектами використаємо наступні технології:

Інтерфейс-частина додатку:

— Angular

— NGXS стейт менеджмент

Серверна частина додатку:

— Node js (express.js)

— Sequelize ORM

— база даних PostgreSQL

Система працюватиме за наступною моделлю, зображеною на рисунку 2.5:

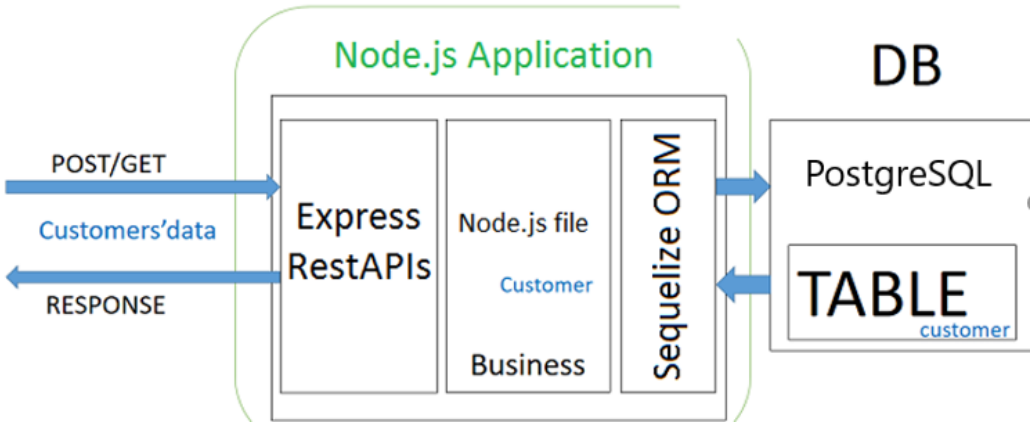


Рис. 2.5 Схема роботи системи

Розглянемо використовувані технології детальніше.

2.2.1 Angular

Angular — це платформа розробки, побудована на TypeScript, фреймворк Javascript створений ще в 2009 році компанією Google для ефективнішої та

простішої веб-розробки. TypeScript — мова програмування, представлена Microsoft восени 2012; позиціонується як засіб розробки клієнтської частини веб-додатків. [14] Ангуляр дуже популярний фреймворк веб-розробки, за даними Stack Overflow Survey 2021, це четвертий за частотою використання інтерфейсних веб-фреймворків [31].

Використовуючи Angular CLI є можливість автоматично створювати структуру компонентів. сервісів, модулів, додавати тести та розгортати проект.

Архітектура програми Angular спирається на певні фундаментальні концепції. Основними будівельними блоками фреймворка Angular є *компоненти*, які згруповані в NgModules (модулях). Додаток завжди має принаймні кореневий *модуль*, необхідний для розгортання додатку, проте зазвичай додатки мають складнішу структуру та декілька модулів.

Компоненти — це будівельні блоки, з яких складається додаток. Компонент включає в себе клас TypeScript з декоратором ***@Component(),*** шаблон HTML і стилі. Декоратор @Component() вказує специфічну для Angular інформацію [38]:

— Селектор, який визначає, як компонент використовується в шаблоні. Елементи HTML у вашому шаблоні, які відповідають цьому селектору, стають екземплярами компонента.

— Шаблон HTML, який вказує Angular, як відобразити компонент.

— Додатковий набір стилів CSS, які визначають зовнішній вигляд HTML-елементів шаблону.

Приклад базового елементу в Ангуляр виглядатиме так [38]:

```
import { Component } from '@angular/core';
@Component({
 selector: 'hello-world',
 template: `
  <h2>Hello World</h2>
  <p>This is my component!</p>
```

```
  `
})
export class HelloWorldComponent {}
```

Щоб використати цей компонент, слід в темплейті написати наступне:

```
<hello-world></hello-world>
```

Компоненти відповідають за те, що додаток відображає, набір елементів та даних якими є можливість маніпулювати та відображати відповідно до логіки додатку.

Компоненти використовують *сервіси*, які зазвичай займаються отриманням, обробкою, зберіганням та передачою даних, проте ніяк не пов'язані з відображенням.

Сервіси додають до компонентів чи інших сервісів за допомогою впровадження залежностей (ін'єкції залежностей - dependency injection), що робить код модульним, його можна використовувати багаторазово, що пришвидшує розробку та полегшує пітримку додатку в майбутньому:

```
@Component({
  selector: 'pm-auth-page',
  templateUrl: './auth-page.component.html',
  styleUrls: ['./auth-page.component.scss']
})
export class AuthPageComponent {
  constructor(
     private readonly store: Store,
  ) {

   }
}
```

У даному прикладі в компонент додана залежність private readonly store: Store.

Модулі, компоненти та сервіси — це класи, які використовують декоратори. Ці декоратори позначають тип і надають метадані, які повідомляють Angular, як їх використовувати.

Метадані компоненту пов'язують його із шаблоном, що є звичайним HTML у поєднанні з директивами та прив'язкою даних Angular, що дозволяють Angular змінювати HTML у відповідності з написаною логікою перед його відображенням користувачеві.

Метадані сервісу надають інформацію, необхідну Angular для ін'єкції залежностей (DI).

Екосистема Angular допомагає, наприклад, скористатися миттєвим знаходженням помилок, інтелектуальним завершенням коду та деякими іншими функціями в інтегрованому середовищі розробки (IDE) та редакторах. Також варто зазначити, що є багато безкоштовних середовищ розробки, що підтримують Ангуляр проекти.

Як платформа, Angular включає:

— Компонентний фреймворк для створення масштабованих веб-додатків

— Колекції добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, клієнт-серверний зв’язок тощо

— Набір інструментів для розробників, які допоможуть розробляти, створювати, тестувати та оновлювати код

Переваги використання Ангуляру:

— Можливість багаторазового використання коду. Компоненти в ангуляр проекті добре інкапсульовані, іншими словами, самодостатні. Тобто їх можна повторно використовувати у різних частинах програми. Це особливо корисно в корпоративних програмах, де різні системи відрізняються за основним

функціоналом, але можуть мати багато подібних елементів, як-от вікна пошуку, засоби вибору дати, списки сортування тощо.

— Читабельність. Інкапсуляція також гарантує, що нові розробники, які нещодавно приєдналися до проекту, можуть швидше та простіше зрозуміти код і в кінцевому підсумку швидше вивчити проект для продуктивної роботи.

— Зручний для модульного тестування. Незалежний характер компонентів спрощує написання тестів.

— Підтримка проекту. Компоненти, які є незалежними одиницями можна легко замінити в будь-який момент в разі необхідності, що пришвидшує розширення, оновлення та підтримку проекту.

2.2.2 Node.js та Express.js

Офіційна документація надає таке визначення Node.js:

Node.js – це платформа, побудована на середовищі виконання Chrome JavaScript для легкого створення швидких і масштабованих мережевих програм. Node.js використовує керовану подіями, неблокуючу модель вводу-виводу, що робить його легким та ефективним, ідеальним для роботи додатків у реальному часі з інтенсивним використанням даних, які працюють на розподілених пристроях. [32]

Розглянемо архітектуру NodeJs детальніше [32].

У той час як більшість альтернативних середовищ виконання використовує багатопотокові моделі обробки, Node.js є однопотоковим.

У налаштуваннях багатопотокової обробки кожен сервер має обмежений пул потоків, до якого він може отримати доступ. Тому щоразу, коли сервер отримує запит, він витягує потік з пулу і призначає його цьому запиту. У цьому випадку обробка є синхронною та послідовною, що означає, що виконується одна операція за раз.

При багатопоточній обробці потік вибирається щоразу, коли надходить запит, доки не будуть використані всі обмежені потоки – в такому разі сервер буде чекати поки не звільниться потік. Це може призводити до повільного отримання/надсилання даних, що негативно впливатиме на досвід клієнтів. Особливо це може стати проблемою, якщо ваша програма має працювати з великою кількістю одночасних запитів клієнта.

Node.js використовує однопотокову обробку. Однопотокові архітектури обробляють кожен запит за допомогою одного основного потоку, використовуючи цикли подій для виконання блокуючих операцій введення/виводу неблокуючим способом.

Однопотокова архітектура, зображена на рисунку 2.6, теоретично може працювати та масштабуватися набагато швидше й ефективніше, ніж багатопотокові налаштування. Це те, що Райан Дал мав на увазі, коли вперше написав Node.js, і є важливою частиною того, чому він настільки популярний серед розробників веб-додатків.

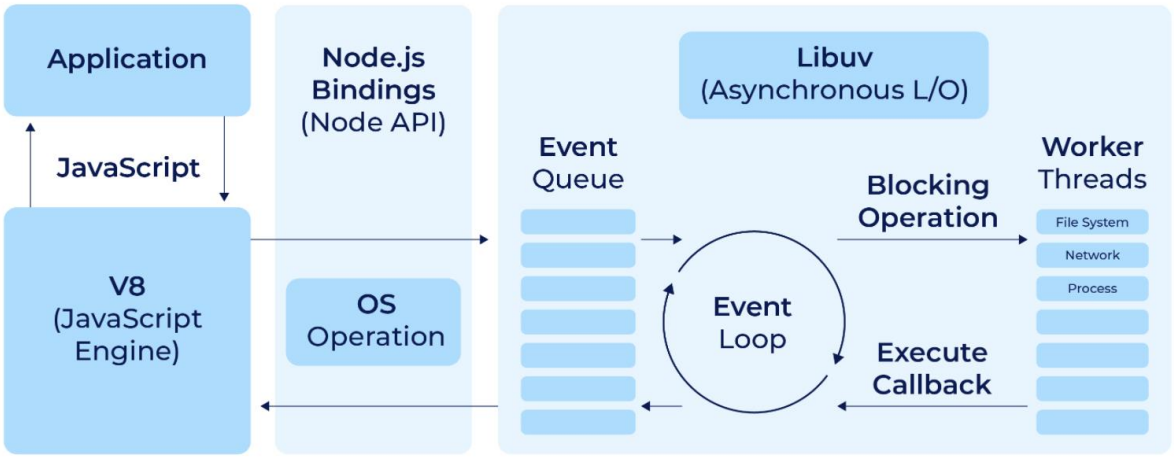


Рис. 2.6 Архітектура Node js

JavaScript, який лежить в основі Node.js, вже деякий час є однією з найпопулярніших мов програмування. Хоча він частіше використовується для веб-

розробки інтерфейсу (фронтенд частини додатку), він також завоював популярність у різних сферах застосування та на окремих платформах, таких як Node.js.

Незважаючи на те, що Node.js був спочатку написаний у 2009 році, поширення він отримав не так давно. Наразі численні успішні бренди використовують Node.js для розробки додатків, включаючи Walmart, Netflix, Medium, LinkedIn або Groupon.

Розглянемо переваги та недоліки Node.js більш детально.

— Використання Node.js для IoT (Internet of Things) — це мережа пристроїв, таких як датчики, маяки, приводи та будь-які інші елементи, вбудовані з електронікою, що дозволяє їм надсилати та обмінюватися даними. IoT може складатися з тисяч таких пристроїв, що ускладнює керування запитами та потоками даних від пристроїв і між ними. З 2012 року Node.js став одним із кращих рішень для підприємств та організацій, які прагнуть розробити системи IoT, оскільки він може обробляти кілька одночасних запитів і подій, які надсилаються тисячами чи навіть мільйонами пристроїв у мережі без зниження швидкості та продуктивності.

Лавина запитів і даних, що надходять від пристроїв IoT, не блокує сервери Node.js завдяки його архітектурі, керованій подіями, та асинхронній обробці, що підходить для операцій із введенням-виводом у мережі IoT.

— Ще одним прикладом, де Node.js є одним із найкращих рішень є чат у режимі реального часу. Node.js надає всі основні функції для створення чатів у реальному часі будь-якої складності. Зокрема, реалізувати push-повідомлення, які широко використовуються в обміні миттєвими повідомленнями та інших програмах реального часу.

— Односторінкові програми (SPA) – це популярний підхід до веб-розробки, за якого ціла програма розміщується на одній сторінці з метою забезпечення всебічного користувальницького досвіду, подібно до настільної програми. Класичним прикладом SPA є Gmail з його цілісною презентацією та безперебійним оновленням нових вхідних повідомлень. Node.js чудово підходить

для SPA завдяки його здатності обробляти асинхронні виклики та навантаження з великими обсягами даних, характерні для цих програм. Також, Node.js чудово підходить для SPA, оскільки він написаний тією ж мовою (JavaScript), що й багато популярних фреймворків JavaScript (Ember, Meteor, React, Angular), які використовуються для створення SPA.

— Спільнота розробників Node.js – це активна та активна група розробників, які сприяють постійному вдосконаленню Node.js. Завдяки співпраці програмістів JavaScript та їх вкладу до спільноти є можливість отримати доступ до вже готових рішень, знайти відповіді на питання, які виникають.

Незважаючи на всі переваги, перераховані вище, Node.js має деякі «мінуси»:

— Нестабільний API. Одним з найбільших недоліків Node.js є несумісність нових версій із новими. API Node.js часто змінюється і якщо нова версія містить оновлення, що є несумісними із попередньою версію, то потрібно вносити зміни в код, що потребує часу.

— Більше часу на розробку. Відсутність автоматизації, початкового створення проекту та вказівок, з Node.js в основному потрібно писати все з нуля. Це забезпечує гнучкість проекту, але також прихводить до зниження продуктивності та уповільнення роботи.

— Не підходить для важких комп’ютерних програм. Node.js не підтримує багатопотокове програмування. У той час як він може працювати з великим обсягом даних, він не підходить для виконання тривалих обчислень. Довгі та громіздкі обчислення блокують вхідні запити, що може призвести до зниження продуктивності.

— Відсутність стандартних рішень. Хоча деякі рішення є досить популярними, вони не є стандартизованими. Інструменти встановлюються за допомогою менеджера пакетів npm, що є відкритим для публікування, тому деякі пакети можуть не відповідати стандартам кодування.

Для створення API серверної частини використаємо фреймворк Express.js.

Express.js — це безкоштовна платформа веб-додатків з відкритим вихідним кодом для Node.js. Він використовується для швидкого та легкого проектування та створення веб-додатків. Оскільки для Express.js потрібен лише javascript, програмістам і розробникам стає легше створювати веб-додатки та API без будь-яких зусиль.

Express.js є легким і допомагає організувати веб-додатки на стороні сервера в більш організовану архітектуру MVC. Express є бекендом для програмного стека MEAN, разом з базою даних MongoDB і каркасом Angular для фронтенду, проте, як стандартна технологія для Node.js, часто використовується незалежно від бази даних чи фреймоврку клієнтської частини.

Під API розуміється в даному впадку REST API — підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP.

Ось як зазвичай реалізується служба REST:

Формат обміну даними: тут немає жодних обмежень. JSON — дуже популярний формат, хоча можна використовувати інші, наприклад XML

Транспортний протокол: завжди HTTP(s). REST повністю побудований на основі HTTP.

Опис сервісу: немає стандарту для цього, REST є доволі гнучким. Однак широко використовуються такі мови визначення веб-програм, як WADL (Web Application Definition Language) і Swagger. Принцип роботи РЕСТ АПІ можна побачити на рисунку 2.7

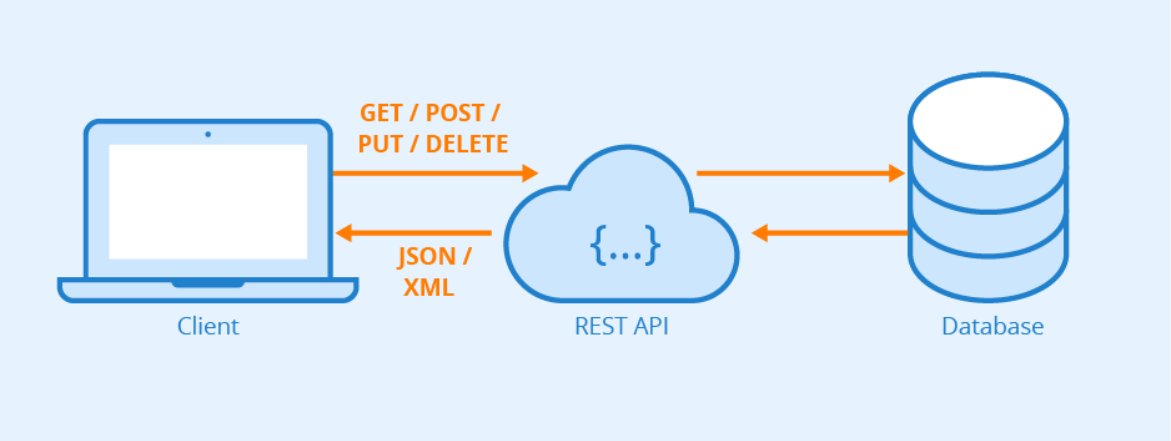


Рис. 2.7 Принцип роботи РЕСТ АПІ

Дані можуть передаватись у стандартних форматах: HTML, XML, JSON, протокод предачі даних (наприклад HTTP) має підтримувати кешування та не повинен зберігати дані про спілкування клієнта та сервера.

2.2.3 NPM

Для керування модулями використовується пакетний менеджер *npm (node package manager*).[9]

Менеджер пакетів (або скорочено «npm») виконує кілька речей; по-перше, він діє як онлайн-сховище для публікації проектів Node.js з відкритим кодом. По-друге, він використовується як засіб командного рядка для взаємодії з цим репозиторієм, допомагаючи в установці пакетів, керуванні версіями та залежностями.

Найчастіше він використовується для публікації, виявлення, встановлення та розробки програм Node. По суті, це допомагає розробникам використовувати сторонні інструменти та пакети Node.js у своєму проекті.

2.2.4 PostgreSQL

PostgreSQL – це потужна система об’єктно-реляційних баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями. Витоки PostgreSQL сягають 1986 року в рамках проекту POSTGRES в Каліфорнійському університеті в Берклі і має понад 30 років активної розробки на базовій платформі.

PostgreSQL заслужив популярність серед розробників завдяки своїй перевіреній архітектурі, надійності, цілісності даних, великому набору функцій та розширюваності. PostgreSQL працює на всіх основних операційних системах, має можливість встановити доповнення, такі як популярний розширювач геопросторової бази даних PostGIS.

У PostgreSQL є багато функцій, які допомагають розробникам створювати програми, адміністраторам захищати цілісність даних і створювати відмовостійкі середовища, а також допомагають керувати даними незалежно від того, наскільки великий чи малий набір даних. Наприклад, є можливість зазначати власний тип даних, створювати власні функції, писати код з різних мов програмування без перекомпіляції бази даних.

PostgreSQL намагається відповідати стандарту SQL, якщо така відповідність не суперечить традиційним функціям або може призвести до неправильних архітектурних рішень. Багато функцій, необхідних стандартом SQL, підтримуються, хоча іноді вони мають дещо відмінний синтаксис або функції.

2.2.5 NGXS стейт

NGXS — це стейт менеджмент паттерн + бібліотека для Angular. Він діє як єдине джерело істини для даних (стану) додатку, забезпечуючи прості правила для передбачуваних мутацій стану.

У великих програмах керувати даними дуже складно. У Angular кожен компонент має свій власний стан, щоб поділитися даними між компонентами, зазвичай використовуються декоратори @Inpput і @ Output, але коли програма стає більшою, підтримувати узгодженість даних важко, оскільки за допомогою цих декораторів дані передаються лише у визначеному напрямку:

— @Input – від батьківського компонента до наступного

— @ Output – від вкладеного компонента на верх до батьківського

Тому, якщо дані потрібно передати в межах двох чи більше рівнів це може бути складним. Можливо також використовувати сервіси, але існує можливість того, що дані можуть відрізнятись на різних рівнях, адже на будь-якому з них вони можуть бути змінені.

Для вирішення цієї проблеми було введено redux. Він забезпечує центральне сховище даних, яке зберігає всі дані вашої програми. Кожен компонент може отримати доступ до збереженого стану, не надсилаючи його від одного компонента до іншого.

NGXS використовує функції TypeScript, такі як класи та декоратори, що полегшує роботу з ним.

NGXS дуже простий у використанні в порівнянні з іншими моделями управління станом, такими як redux і Akita. NGXS використовує всі переваги angular і typescript над шаблоном redux. NGXS має чотри концепції:

1. *Store* - це глобальний контейнер, який і зберігає в собі всі дані. Для зміни даних використовується спеціальна функція dispatch, що викликає екшіон (action).

Наприклад екшіон для логіну в жодаток за допомогою різних платформ

```
  public login(loginPlatform: LoginPlatform): void {
    this.store.dispatch(new AuthActions.LoginViaExternalProvider({ loginPlatform }));
  }
```

2. *Action* - це тип команди, яку слід викликати, коли потрібно змінити дані в стейті або потрібно запустити цепочку подій, як от вхід у додаток.

```
export class LoginViaExternalProvider extends Initializable<LoginViaExternalProvider> {
public static readonly type: string = `${STATE_LABEL} login via external provider`;

public readonly loginPlatform: LoginPlatform;
}
  @@Action(AuthStateActions.LoginViaExternalProvider)
public loginViaExternalProvider(ctx: StateContext<AuthStateModel>,
  { loginPlatform }: AuthStateActions.LoginViaExternalProvider,
) {

  ctx.patchState({
    isLoading: true,
  });

  return this.authService.loginUserViaExternalProvider(loginPlatform).subscribe(
    data => ctx.dispatch(new AuthStateActions.LoginViaExternalProviderSuccess({ userInfo: data?.additionalUserInfo?.profile })),
    error => ctx.dispatch(new AuthStateActions.LoginViaExternalProviderError({ error }))
  );
}
```

3. *Стейт* - це класи разом із декораторами для опису метаданих та відображення дій.

```
@State<AuthStateModel>({
  name: 'authState',
  defaults: new AuthStateModel(),
})
@Injectable()
export class AuthState { }
```

4. *Селектор* – це функції, які вирізають певну частину стану з контейнера глобального стану, тобто отримують дані з стейту, тоді, коли це потрібно.

```
@Selector()
public static userInfo(state: AuthStateModel): UserInfo {
  return state.userInfo;
}

@Select(AuthState.userInfo)
public userInfo: Observable<UserInfo>;
```

Селектор отримує дані із стейту у вигляді Observable.

2.2.6 Взаємодія клієнта та сервера

Існує багато різних способів структурувати веб-додаток. Інтерфейс може приймати різні форми та взаємодіє з сервером (бек-енд).

Інтерфейс, який також називають «клієнтським» програмуванням, — це те, що відбувається у браузері. Це все, що користувач бачить і з чим взаємодіє.

Бекенд, також званий «серверним» програмуванням, відбувається на сервері та базі даних. Це механізм, який працює за лаштунками, щоб забезпечити користувача даними, з якими користувачі взаємодіють на стороні клієнта.

Основні основи веб-розробки включають такі мови, як Java, Ruby, Python, PHP, .Net тощо. Найпоширенішими мовами інтерфейсу є взаємодія HTML, CSS та JavaScript.

Процес отримання даних:

1. Користувач створює запит до веб-сервера через інтерфейс користувача програми.
2. Веб-сервер надсилає цей запит на сервер веб-додатків.
3. Сервер веб-додатків виконує запитане завдання, а потім генерує результати необхідних даних.
4. Сервер веб-додатків надсилає ці результати назад на веб-сервер (запитувана інформація або оброблені дані).
5. Веб-сервер передає запитувану інформацію клієнту (планшет, мобільний пристрій або комп’ютер).
6. Запитувана інформація з’являється на дисплеї користувача.

Наприклад, коли клієнт банку отримує доступ до послуг онлайн-банкінгу за допомогою веб-браузера (клієнта), клієнт ініціює запит на веб-сервер банку.

Облікові дані клієнта можуть зберігатися в базі даних, а веб-сервер отримує доступ до сервера бази даних як клієнт. Сервер додатків інтерпретує повернуті дані, застосовуючи бізнес-логіку банку, і надає вихід на веб-сервер. Нарешті, веб-сервер повертає результат клієнтському веб-браузеру для відображення.

На кожному кроці цієї послідовності обміну повідомленнями клієнт-сервер комп’ютер обробляє запит і повертає дані. Це шаблон обміну повідомленнями запит-відповідь. Коли всі запити задоволені, послідовність завершується, і веб-браузер представляє дані клієнту.

Переваги моделі клієнт-сервер:

— Централізована система з усіма даними в одному місці.

— Вимагає менших витрат на обслуговування, а відновлення даних можливе.

— Частини інтерфейсу і сервера можна змінювати окремо.

Недоліки моделі клієнт-сервер:

— Клієнти схильні до вірусів, троянів і хробаків, якщо вони присутні на Сервері або завантажені на Сервер.

— Сервер схильний до атак відмови в обслуговуванні (DOS).

— Пакети даних можуть бути підроблені або змінені під час передачі.

— Фішинг або захоплення облікових даних для входу або іншої корисної інформації користувача є поширеними, а атаки MITM (Людина посередині) є поширеними.

2.3 Інструменти візуалізаціїї в системах управління проектами

Новим трендом у розробці систем по управлінню проектами є інструменти візуалізації процесів.

Ці інструменти дозволяють PM та членам команди візуалізувати хід своїх проектів. Візуалізація допомагає контролювати процеси, аналізувати їх ефективність. Візуальні інструменти управління проектами також є засобом виявлення проблем у процесі та внесення поправок там, де це необхідно.

Деякі з найбільш широко використовуваних способів візуалізації включають:

— Канбан дошки

— Структура декомпозиції робіт (WBS),

— Діаграма Ганта

— Календар

Канбан-дошка — це інструмент, який забезпечує візуальну систему для команд задля управління завданнями проекту, робочими процесами та спілкуванням. Використання таких дошок може допомогти спростити назначення задач на членів команди, менеджери проектів можуть швидко вказати, на якому саме кроці знаходиться кожна задача.

Канбан-картки – це візуальне представлення завдань. Кожна картка містить інформацію про завдання та його статус, наприклад, термін виконання, на кого задача назначена, опис тощо.

Стовпці Kanban – кожен стовпець на дошці представляє етап робочого процесу. Картки мають відображати робочий процес від початку та до повного завершення, що можна побачити на рисунку 2.8.

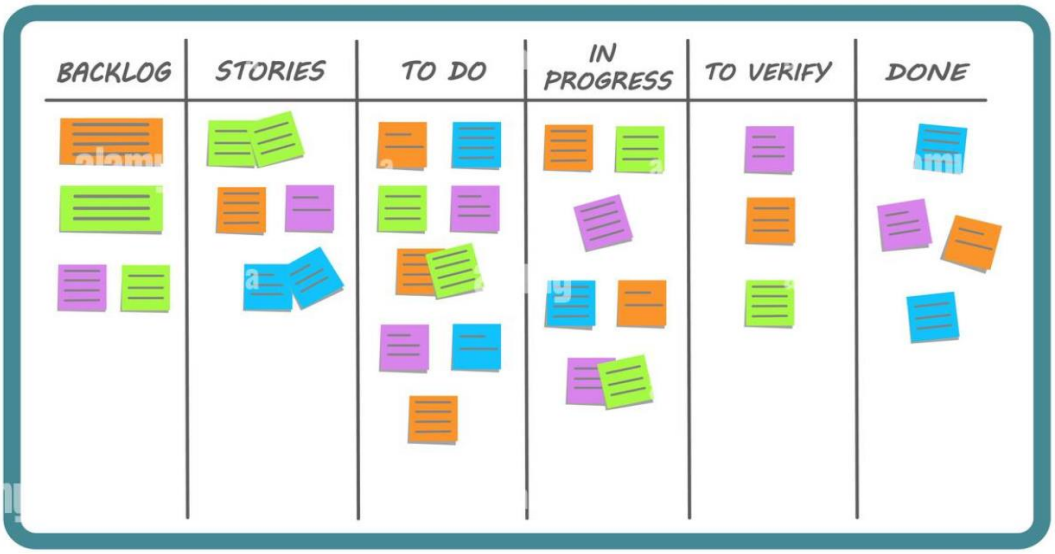


Рис 2.8 Приклад канбан-дошки

Структура декомпозиції робіт (WBS) — це візуальна, ієрархічна та орієнтована на виконання деконструкція проекту. Це корисна діаграма для менеджерів проектів, оскільки вона дозволяє їм розбити масштаби свого проекту та візуалізувати всі завдання, необхідні для завершення проектів.

Усі етапи роботи над проектом описані в структурній діаграмі, що робить її важливим інструментом планування проекту. Остаточний результат проекту, а також пов'язані з ним завдання та робочі пакети знаходяться на верхній частині діаграми WBS (див. рисунок 2.9), а рівні WBS нижче відображають завдання, результати та робочі пакети, які необхідні для завершення проекту з початку до кінця.

Створення WBS є першим кроком у розробці графіка проекту. Він визначає всю роботу, яку необхідно виконати (і в якому порядку) для досягнення цілей і завдань проекту. Візуалізуючи свій проект таким чином, ви можете зрозуміти масштаб свого проекту та виділити ресурси для всіх завдань проекту.

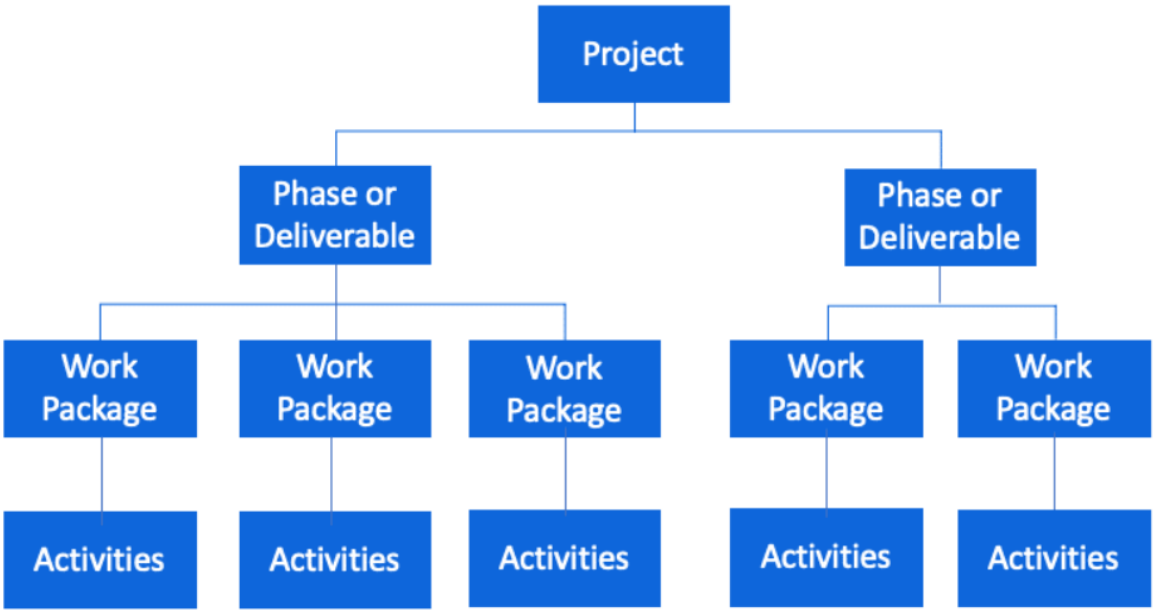


Рис 2.9 Приклад декомпозиції робіт

Діаграма Ганта — це стовпчаста діаграма, яка надає візуальне уявлення про завдання проекту, заплановані протягом певного часу. Діаграма Ганта використовується для планування проекту: зазвичай використовується для планування задач на декілька днів, рисунок 2.10.

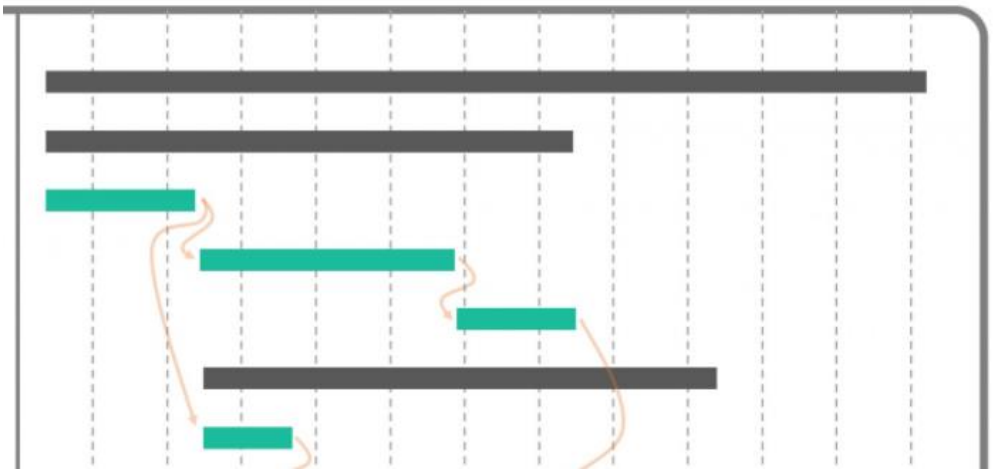


Рис. 2.10 Приклад Діаграми Ганта

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Календар проекту або календар планування проекту— це інструмент, який допоможе організувати графік проекту.

Менеджери проектів використовують календар для ілюстрації графіків у легкому для засвоювання стилі, часто в поєднанні з діаграмою Ганта, щоб відображати інформацію якомога детальніше.

РОЗДІЛ 3 КОНСТРУКТИВНИЙ РОЗДІЛ. РОЗРОБЛЕННЯ ПРОЕКТНИХ РІШЕНЬ ТА ЇХ РЕАЛІЗАЦІЯ

3.1 Проектування бази даних

3.1.1 Створення ERD діаграми бази даних

Як базу даних для додатку використаємо PostgreSQL. Спочатку створимо діаграму зв’язків (див. рисунок 3.1) між сутностями (Entity Relationship Diagram), щоб дослідити структуру бази даних.

БД міститиме наступні таблиці

- Проект (Project): первинний ключ id,
- Користувач (User) – первинний ключ uid,
- Команда (Project_Team) – первинний ключ ProjectID
- Задача (Task) – первинний ключ id
- Назначена задача (Assigned_Task)
- Бюджет (Budget)
- Валюта (Currency)
- Статус завдання (Status)
- Дошка (Board)
- Дошки проекту (Project_Boards)
- Колонка дошки (Board Column)

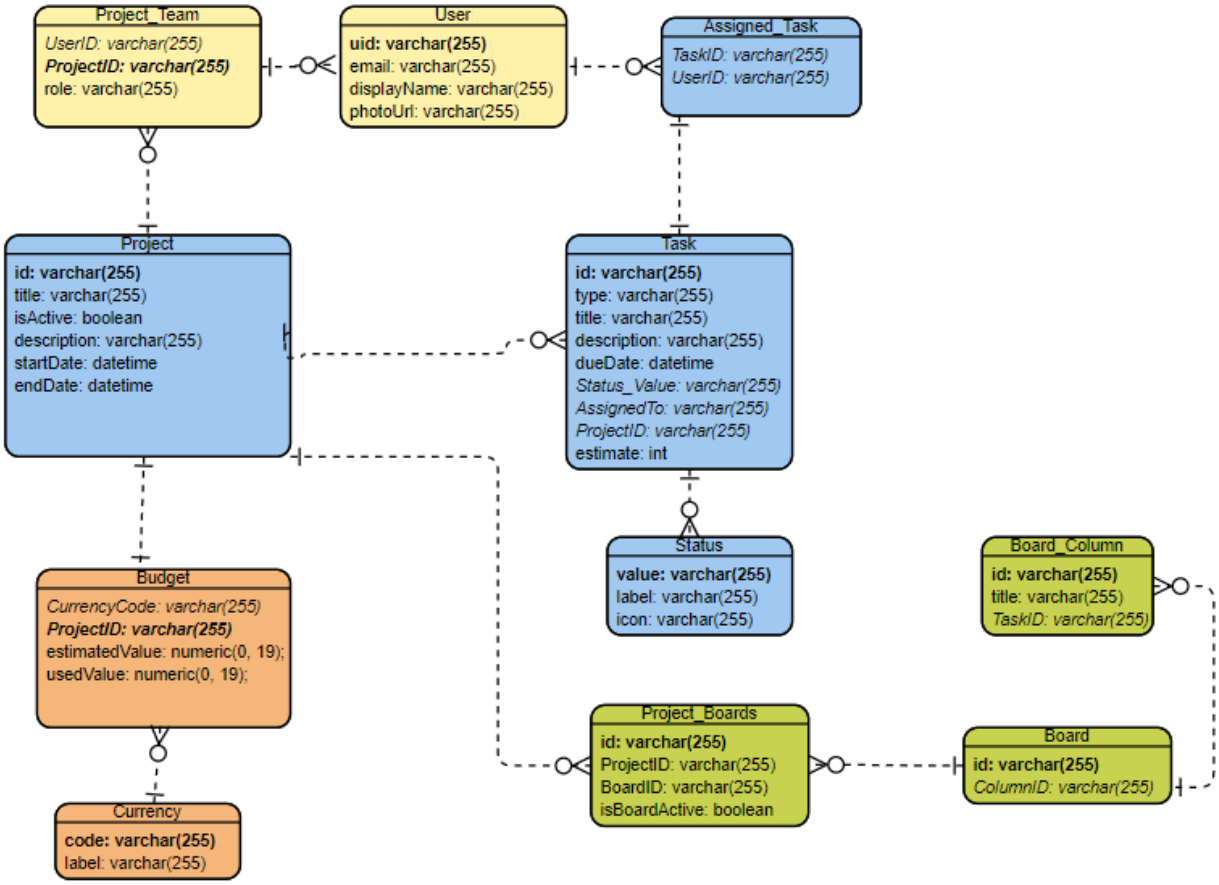


Рис. 3.1 ERD діаграма

3.1.2 Створення та підключення бази даних до проекту

Як хостинговий сервіс для бази даних використаємо ElephantSQL — це безкоштовний сервіс хостингу баз даних PostgreSQL у хмарному середовищі.

Для додавання нової бази даних необхідно увійти в систему, після чого відкриється вікно із вже доданими БД, рисунок 3.2.

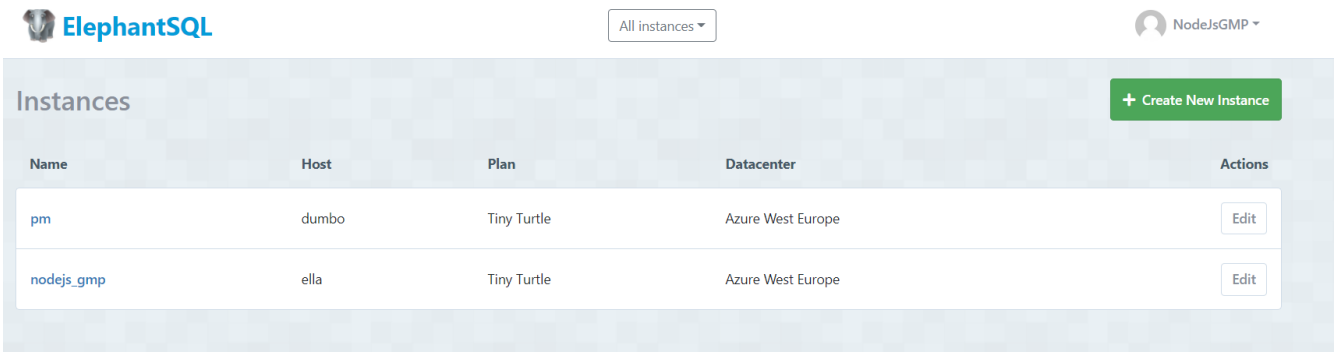


Рис. 3.2 Інтерфейс ElephantSQL

Для створення бази даних натискаємо “Create new instance” та вводимо необхідну інформацію: назву проекту, рисунок, тарифний план та за бажанням можна додати теги.

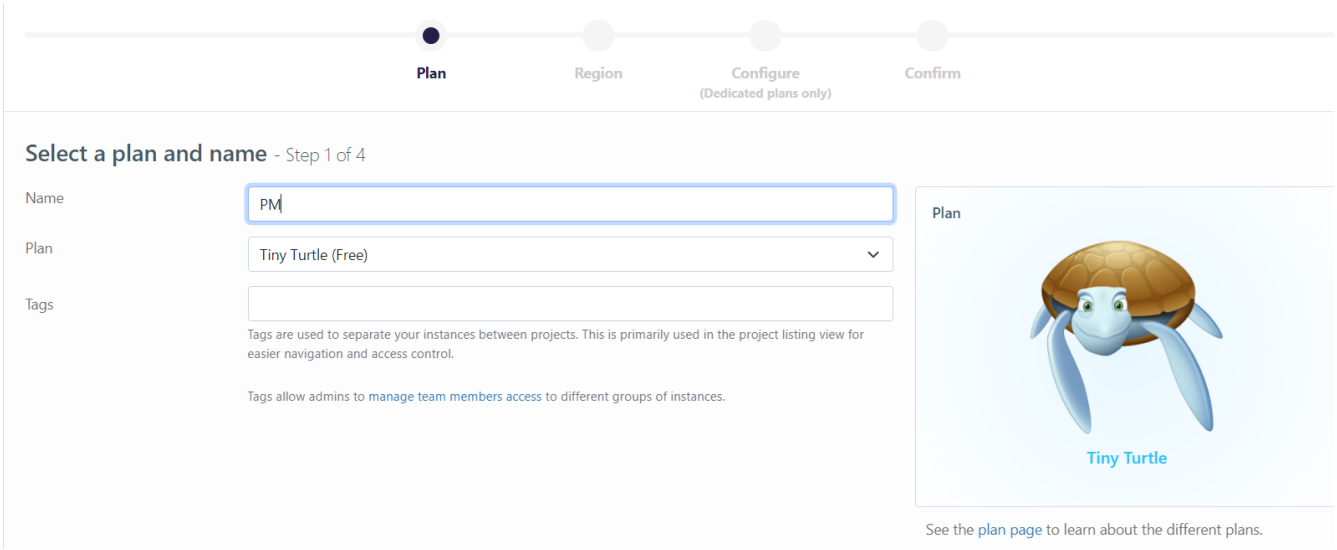
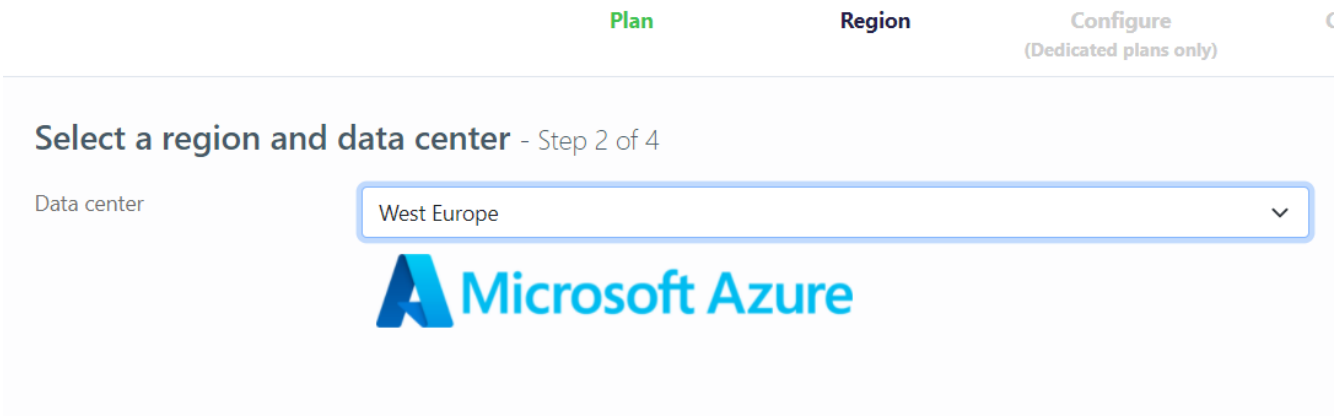


Рис. 3.3 Діалогове вікно створення нової бази даних в ElephantSQL

Після чого буде запропоновано обрати регіон і центр обробки даних для екземпляра бази даних, в цьому випадку обираємо West Europe та Microsoft Azure (рисунок 3.4). ElephantSQL, як правило, сам обирає центр обробки даних/регіон, який є найближчим до вас за замовчуванням.

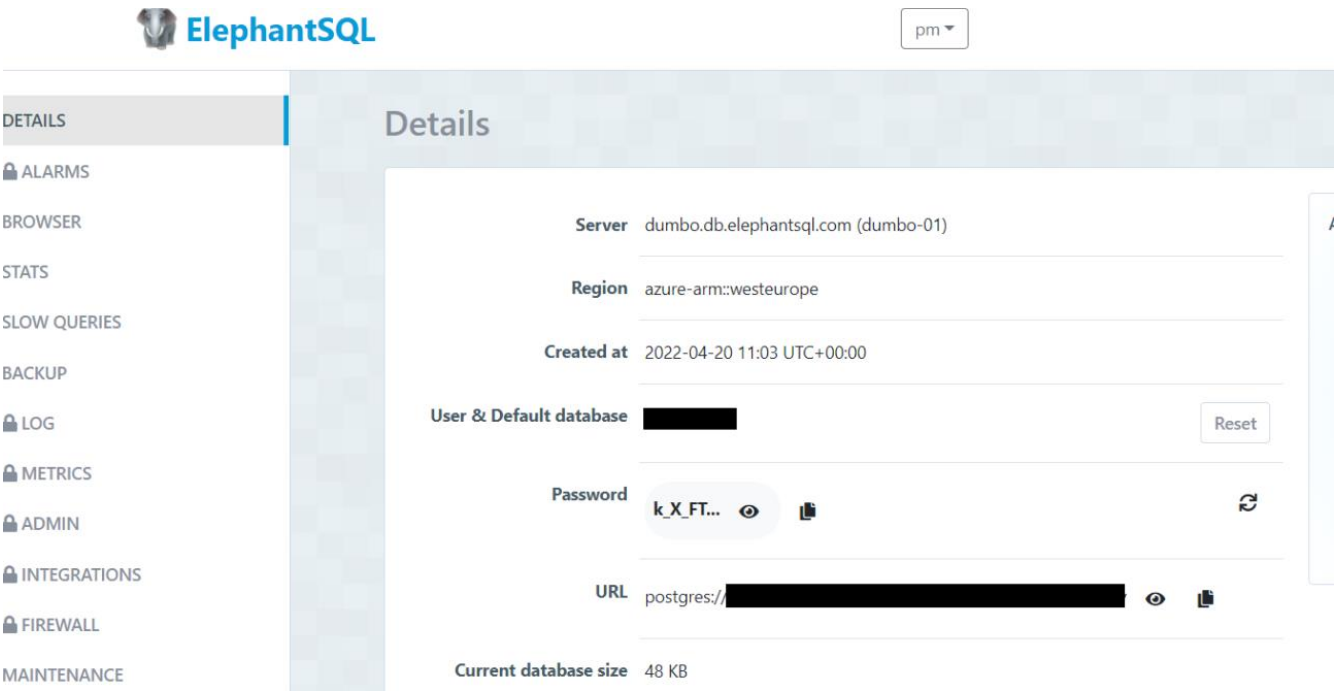


3.4 Вибір датацентру, де зберігатиметься база дних

Наступним кроком є перевірка даних та підтвердження створення екземпляру.

Підключимо створену базу даних до серверної частини додатку.

Інформацію, необхідну для підключення, отримаємо на сторінці деталі, рисунок 3.5:



3.5 Інформація по створеній БД

Оскільки для роботи з базою даних будемо працювати за допомогою Sequelize, то встановимо цей пакет за допомогою node package manager команди [39]:

```
npm install --save sequelize
```

Також необхідно встановити драйвер для PostgreSQL бази даних за допомогою команди:

```
npm install --save pg pg-hstore
```

Щоб підключитися до бази даних, необхідно створити екземпляр Sequelize. Це можна зробити, передавши параметри підключення окремо до конструктора Sequelize або використовуючи URI підключення [39]:

```
export const POSTGRE_DB_URL = `postgres://${process.env.PG_USER}:${process.env.PG_TOKEN}@${process.env.PG_HOST}/${process.env.PG_USER}`;
const sequelize = new Sequelize(POSTGRE_DB_URL, {
     dialect: 'postgres',
});
```

Для перевірки з’єднання використовуємо команду sequelize.authenticate():

```
export const sequelizeAuthenticateLoader = async () => {
  try {
     await sequelize.authenticate();
     sequelize.sync();
  } catch (err) {
     console.error(`Unable to connect to the database: ${err}`);
  }};
```

У результаті успішного підключення після запуску проекту отримаємо наступну інформацію в консолі, зображену на рисунку 3.6:

```
> pm-backend@1.0.0 start
> nodemon --exec npx ts-node ./src/index.ts

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: ts,js
[nodemon] starting `npx ts-node ./src/index.ts`
Executing (default): SELECT 1+1 AS result
Executing (default): CREATE TABLE IF NOT EXISTS "projects" ("id" UUID NOT NULL , "title" VARCHAR(255) NOT NULL, "description" VARCHAR(255), "projectManager" VARCHAR(255)[] NOT N
ULL, "accessType" VARCHAR(255) DEFAULT 'public', "team" VARCHAR(255)[] DEFAULT ARRAY[]::VARCHAR(255)[], "startDate" TIMESTAMP WITH TIME ZONE, "endDate" TIMESTAMP WITH TIME ZONE,
 "isActive" BOOLEAN DEFAULT true, "estimatedBudget" VARCHAR(255), "usedBudget" VARCHAR(255), "createdAt" TIMESTAMP WITH TIME ZONE NOT NULL, "updatedAt" TIMESTAMP WITH TIME ZONE
NOT NULL, PRIMARY KEY ("id"));
Executing (default): SELECT 1+1 AS result
Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array_agg(a.attnum) as column_indexes, array_agg(a.attn
ame) AS column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM pg_class t, pg_class i, pg_index ix, pg_attribute a WHERE t.oid = ix.indrelid AND i.oid = ix.indexrelid A
ND a.attrelid = t.oid AND t.relkind = 'r' and t.relname = 'projects' GROUP BY i.relname, ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY i.relname;
Server is running on port 8080...
```

Рис. 3.6 Вивід в консоль при успішному підключенні до бази даних

Для управління базою даних використаємо PgAdmin - це інструмент керування для PostgreSQL та похідних реляційних баз даних.

Для підключення, після загрузки та встановлення програми, додамо новий сервіс, натиснувши Create Server, відкриється діалогове вікно, зображене на рисунку 3.7.

У вкладці Connection необхідно ввести дані, що надаються в ElephantSQL на сторінці Details:

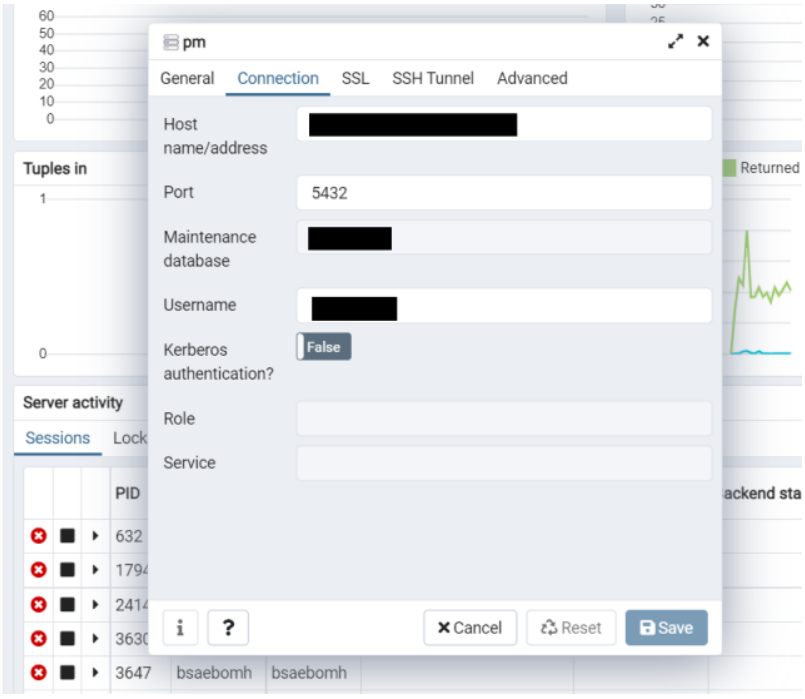


Рис. 3.7 Інтерфейс підключення до інструменту управління базою даних PgAdmin

У разі успішного підключення отримаємо інформацію по базі даних, рисунок 3.8:

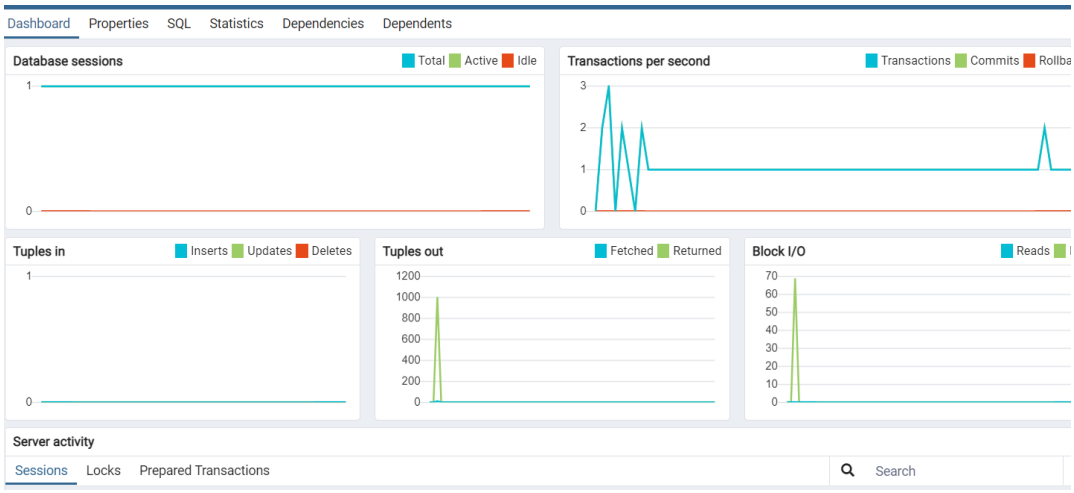


Рис. 3.8 Інтерфейс PgAdmin

3.2 Програмне забезпечення

3.2.1 Ініціалізація інтерфейс-частини проекту

Для ініціалізації Ангуляр-проекту необхідно вставновити Angular CLI. Для цього відкриваємо командний рядок та запускаємо відповідну команду:

npm install -g @angular/cli

Ангуляр-додатки розробляються в контексті робочої області (Angular workspace). Для цього в папці, де потрібно створити проект вводимо команду:

ng new <назва проекту>

Ця команда запропонує надати інформацію про проект:

Зокрема, потрібно вказати чи буде використовуватись роутинг в додатку та обрати формат стилів, що будуть використовуватись в додатку, рисунок 3.9.

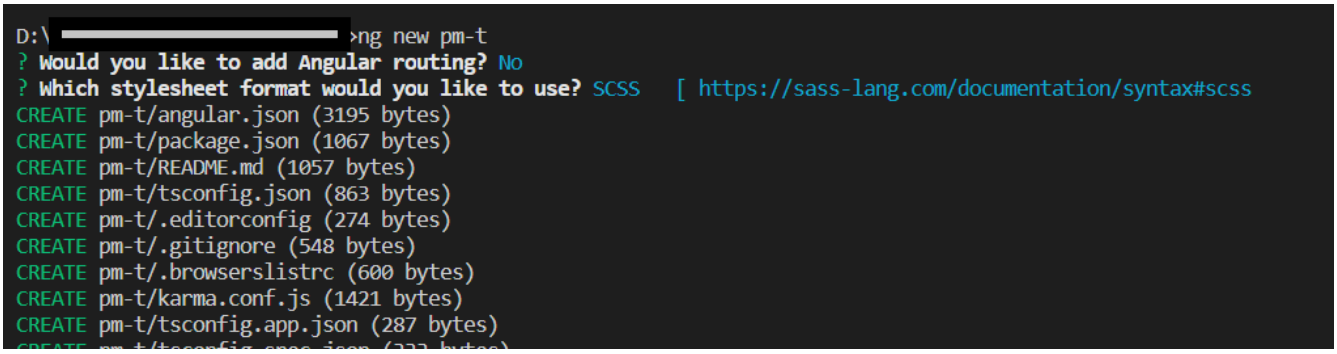


Рис. 3.9 Результат команди ng new в консолі

У даному проекті, так як роутинг буде необхідний, обираємо так. Для формату стилів обрано SCSS (Syntactically Awesome Style Sheet) – надбудова над звичайним CSS, пропонує спосіб писати стилі для веб-сайтів із покращеним синтаксисом CSS, використовуючи функції, міксіни, вкладеність, використання змінних та інше.

Загалом, браузери не знають, як обробляти функції SCSS тому для їх запуску вони конвертуються у звичайні файли CSS.

У результаті команди отримуємо початково сконфігурований проект, рисунок 3.10, що має наступну структуру:

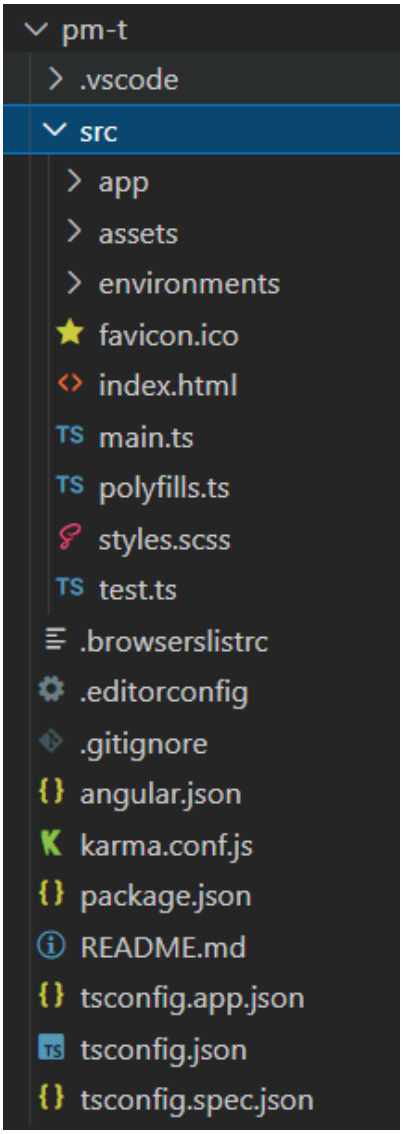


Рис. 3.10 Згенерований проект

Верхній рівень робочої області містить файли конфігурації всього робочого простору, файли конфігурації для програми кореневого рівня та підпапки для вихідних і файлів із тестами програми кореневого рівня.

— .editorconfig - конфігурація для редакторів коду

— .gitignore – в даному файлі вказують папки чи файли, які будуть ігноруватись при заливанні та відстежуванні змін у файлах системою контролю версій (git)

— README.md – містить інформацію про інші файли в каталозі або архіві комп’ютерного програмного забезпечення. Форма документації, зазвичай це простий текстовий файл, якщо для створення проекту використовувався Angular CLI, то README файл буде містити автоматично згенеровану інформацію про ангуляр-проект.

— angular.json - надає конфігурації за замовчуванням для всього робочого простору та конкретного проектудля інструментів збірки та розробки, наданих Angular CLI

— package.json - містить важливі метадані про проект, які потрібні перед публікацією в NPM, а також визначає функціональні атрибути проекту, які npm використовує для встановлення залежностей (рисунок 3.11), запуску сценаріїв та визначення точки входу в наш пакет. . Містить в собі також список всіх залежностей проекту

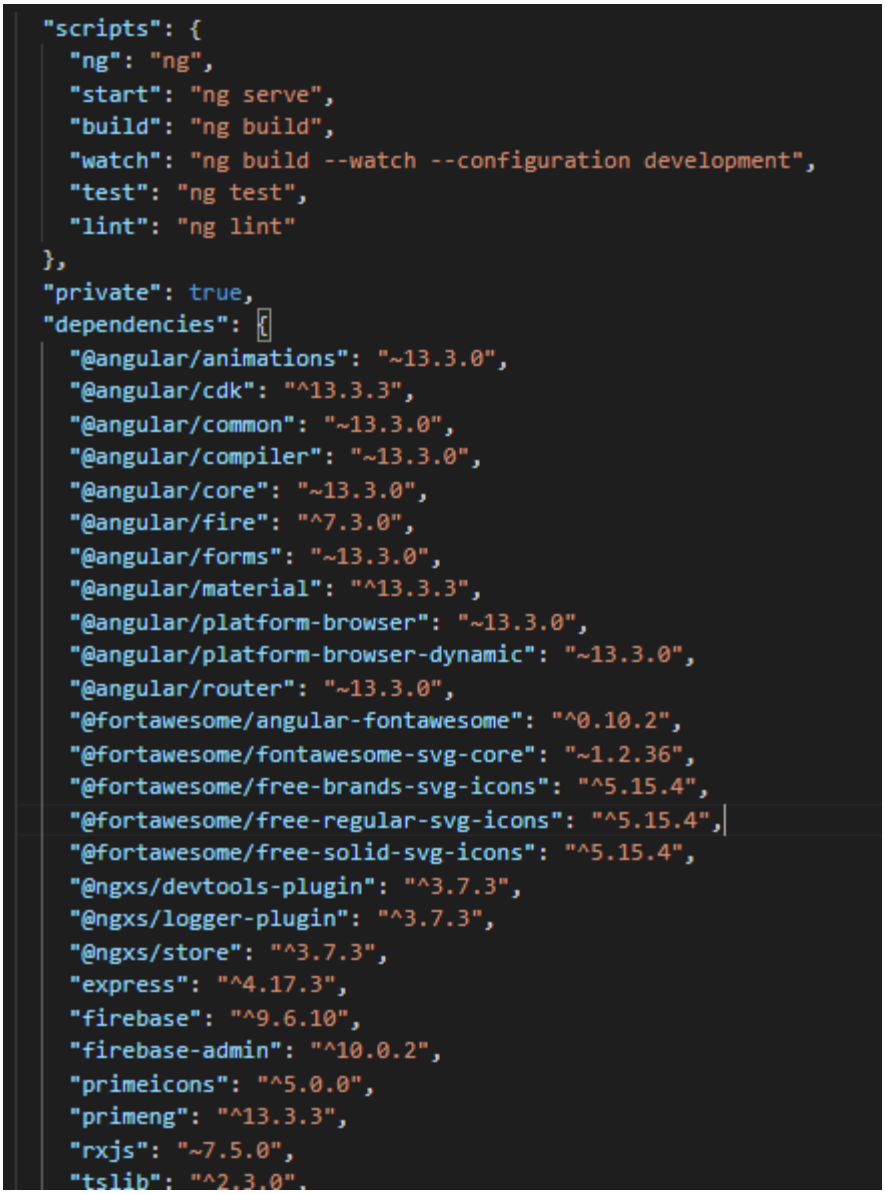


Рис. 3.11 Встановлені залежності

— package-lock.json - надає інформацію про версію для всіх пакетів, встановлених у node_modules клієнтом npm

— src – коренева папка для проекту

— node_modules – містить в собі всі локально завантажені залежності для проекту

— tsconfig.json - основна конфігурація TypeScript для проектів у робочій області. Усі інші файли конфігурації успадковуються від цього базового файлу

Розглянемо також основні файли в папці src:

— polyfills.ts - містить поліфіл-скрипти для підтримки в браузері

— index.html - головна сторінка HTML, яка відкривається, коли хтось відвідує додаток. CLI автоматично додає всі файли JavaScript і CSS під час створення програми

— styles.scss - файли CSS, що містяться в проекті

Команди, що використовуються для збірки проекту описуються в package.json. Для старту проекту, рисунок 3.12, в режимі розробки використаємо команду:

npm start

```
> pm@0.0.0 start
> ng serve

✓ Browser application bundle generation complete.

Initial Chunk Files   | Names         |  Raw Size
vendor.js             | vendor        |   7.51 MB |
styles.css, styles.js | styles        | 513.76 kB |
polyfills.js          | polyfills     | 339.37 kB |
main.js               | main          | 304.81 kB |
runtime.js            | runtime       |   7.12 kB |

                      | Initial Total |   8.65 MB

Build at: 2022-05-02T09:42:13.814Z - Hash: 27c134d7d4a7fd78 - Time: 17987ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
```

Рис. 3.12 Результат команди в консолі

3.2.2 Ініціалізація серверної частини проекту

У кореневому рівні проекту створимо папку src – коренева папка проекту- та додамо файли конфігурації.

Структура проекту виглядатиме наступним чином:

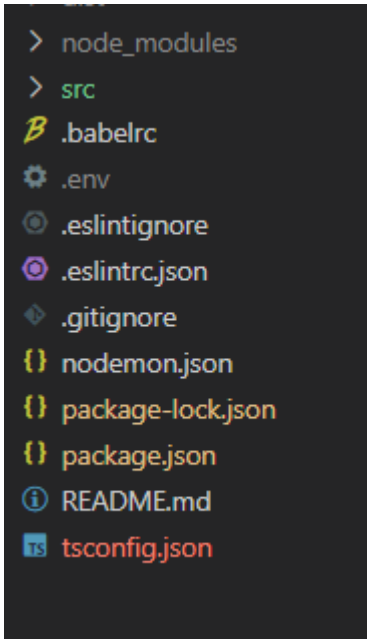


Рис. 3.13 Структура серверної частини додатку

— .babelrc – конфігураційний файл для Babel, інструменту, який в основному використовується для перетворення коду ECMAScript 2015+ у зворотно-сумісну версію JavaScript у поточних і старих браузерах чи середовищах

— .env – файл, що містить в соббі змінні середовища (environment variables)

— .gitignore – в даному файлі вказують папки чи файли, які будуть ігноруватись при заливанні та відстежуванні змін у файлах системою контролю версій (git)

— package.json - містить важливі метадані про проект

— node_modules – містить в собі всі локально завантажені залежності для проекту

— tsconfig.json - основна конфігурація TypeScript для проектів у робочій області. Усі інші файли конфігурації успадковуються від цього базового файлу

— README.md – містить інформацію про інші файли в каталозі або архіві комп'ютерного програмного забезпечення. Форма документації, зазвичай це простий текстовий файл

— .eslintrc.json – файл із конфігураційями для eslint - інструмент статичного аналізу коду для виявлення проблемних шаблонів, виявлених у коді JavaScript

— .eslintignore – файли та папки, які будуть ігноруватись при перевірці коду за допомогою лінтера

— nodemon.json – файл конфігурації nodemon

Рис. 3.14 Файл конфігурації nodemon

У Node.js для внесення змін в проект необхідно перезапустити процес, зупинивши попередній та ввести команду запуску знову. Можна автоматизувати цей крок за допомогою утиліти nodemon. Вона спостерігає за файловою системою та автоматично перезапускає процес.

3.3 Модулі та функціонал розробленої системи

Існує можливість зареєструватись в даний додаток, рисунок 3.15, використавши один із відомих провайдерів або створивши нового користувача.

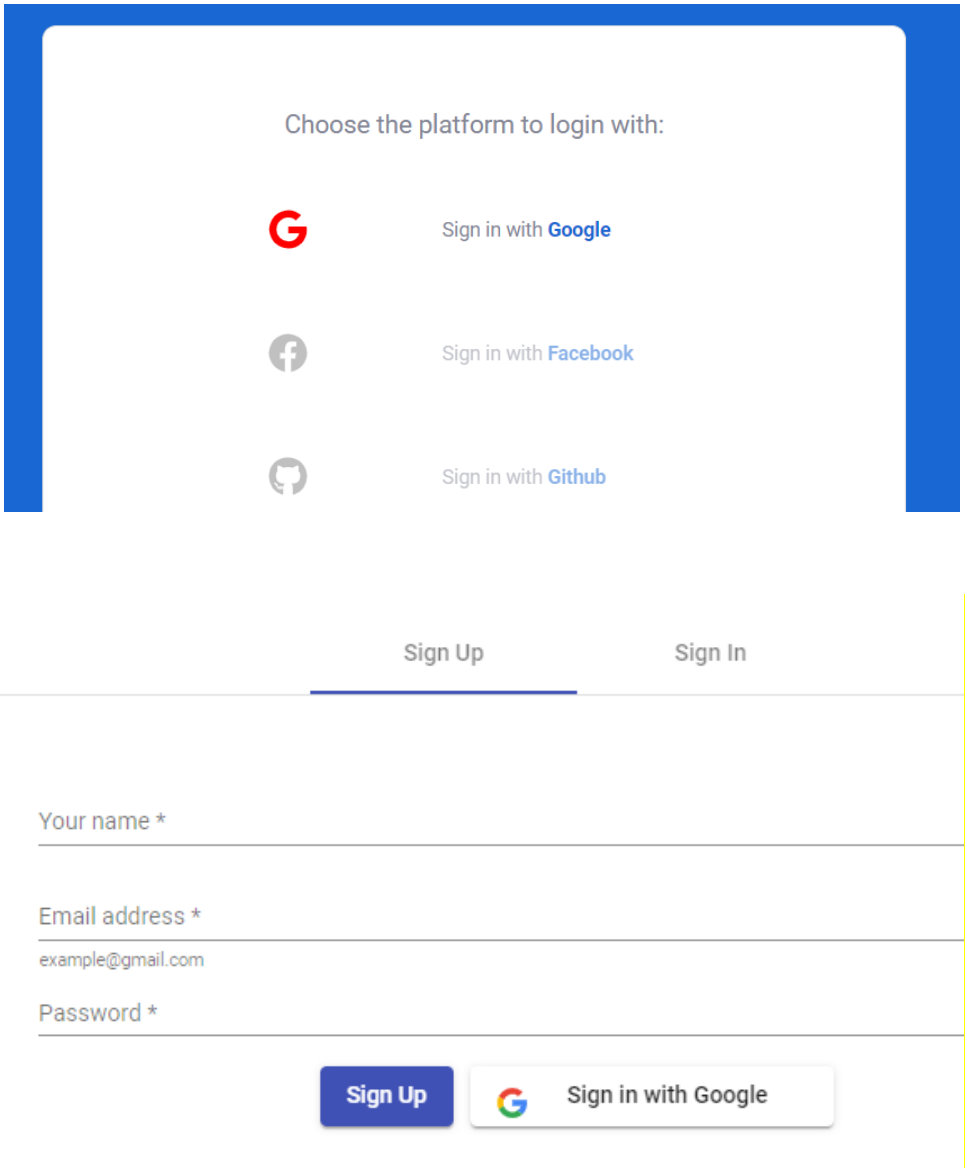


Рис. 3.15 Сторінка авторизації

Приклад авторизації за допомогою Google, рисунок 3.16.

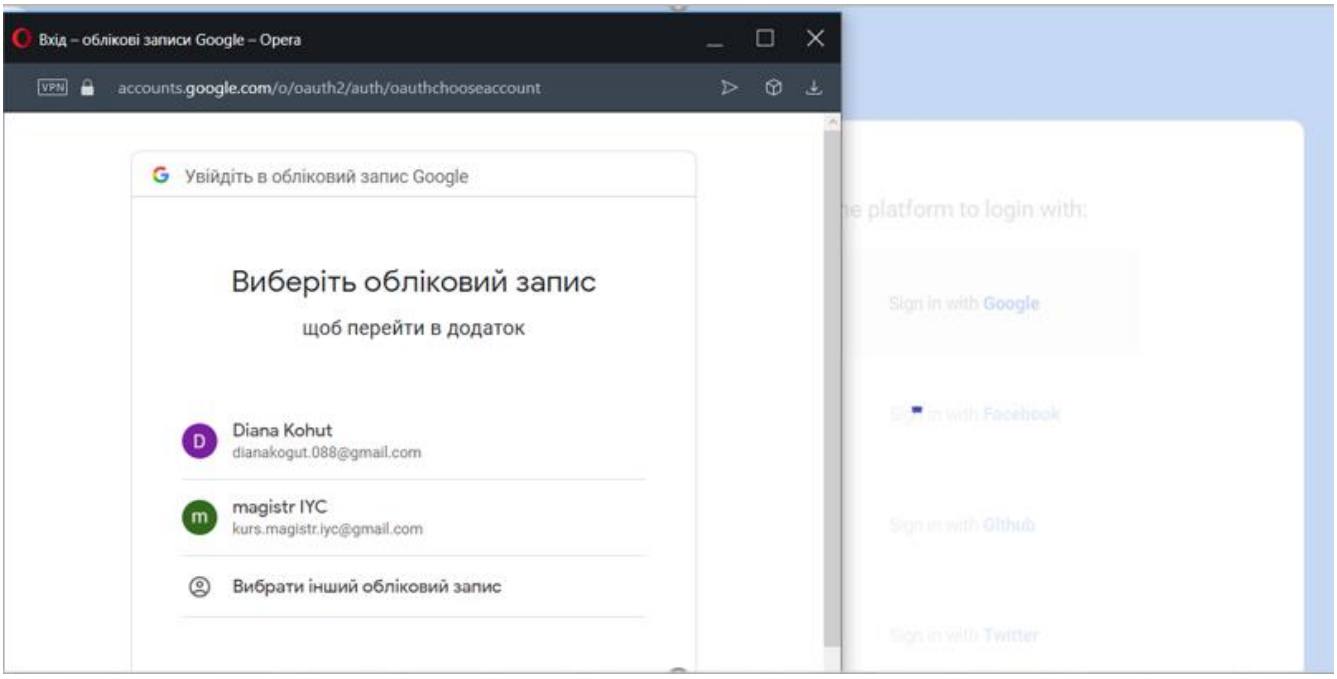


Рис. 3.16 Авторизація за допомогою Google акаунту

Після успішного входу в систему юзер бачить дашборд-сторінку додатку, рисунок 3.17, є можливість відкрити сторінку створення нового проекту:

Рис. 3.17 Головна сторінка додатку

На даній сторінці додатку можна побачити інформацію по прогнозованій та зробленій роботі, а також подивитись зведену інформацію по всіх проектах.

Для планування задачами в системі створено канбан дошку, де є можливість перегляд всіх задач проекту у вигляді з наступних списків та можливістю створити всласний список (рисунок 3.18):

1. Backlog - міститиме в собі ті задачі, які планується виконати в майбутньому проте вони не увійдуть в наступний стек задач.
2. To do – задачі, які можна брати до виконання учасникам проекту, ці задачі ще не взяті ніким із учасників до виконання
3. In progress – ті задачі, над якими працює команда в даний момент.
4. Ready for testing – виконані задачі, які передаються далі для тестування, для перевірки правильності виконання.
5. Done – повністю завершені задачі.

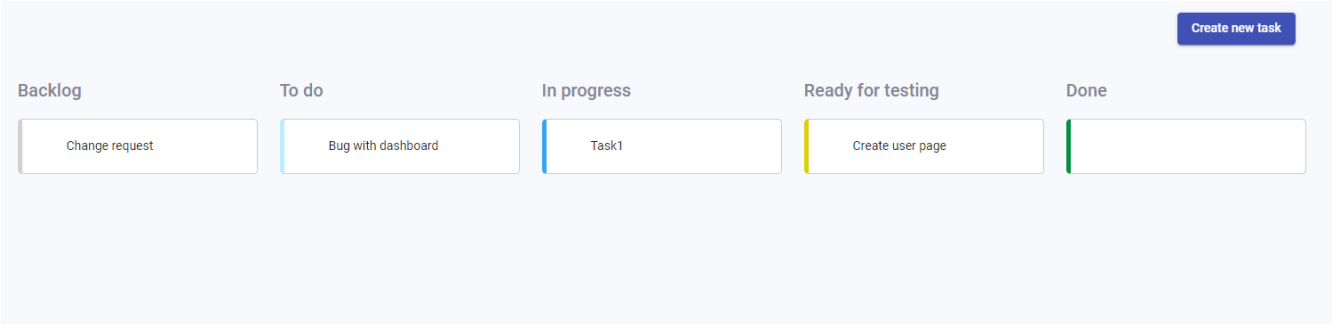


Рис. 3.18 Канбан дошка

Також є можливість створити новий проект, переглянути список проектів, де користувач є членом команди або власником проекту, створити задачу, назначити користувача на виконання певної задачі.

Розглянемо роботу системи та взаємодію серверної та клієнтської частини на прикладі створення проекту.

1. Для початку користувач має ввести всі необхідні дані на сторінці створення проекту, рисунок 3.19.

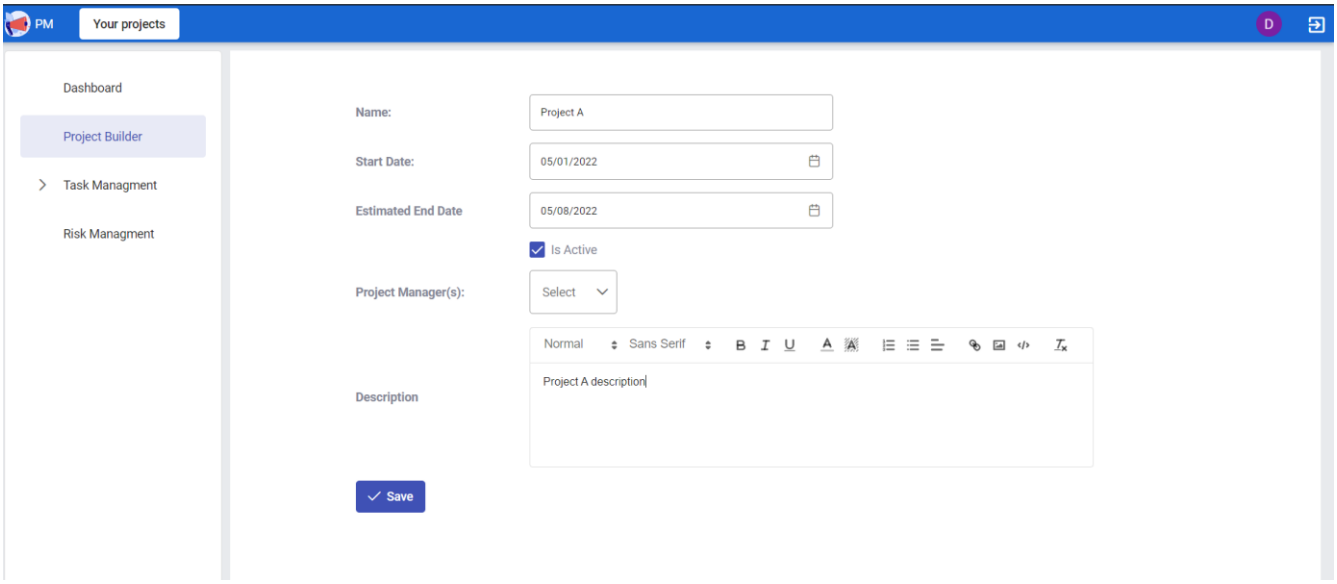


Рис. 3.19 Інтерфейс створення додатку

2. Функція saveProject() створить новий інстанс класу Project та викличе action стейту:

```
public saveProject(): void {
    const projectManagers = this.projectManagers.map(manager => manager.uid);
    this.store.dispatch(new SessionActions.SaveProject({
        project: new Project({
            title: this.title,
            description: this.editorText,
            isActive: !!this.isActive?.length,
            estimatedEndDate: this.estimatedEndDate,
            startDate: this.startDate,
            projectManagers,
        })
```

```
            }));
        }
```

3. Той в свою чергу звернеться до сервіса, що відповідає за відправлення запиту до сервера:

```
    @Action(SessionStateActions.SaveProject)
    public saveProject(
        ctx: StateContext<SessionStateModel>,
        { project }: SessionStateActions.SaveProject,
    ): void {
        ctx.patchState({
            isLoading: true,
        });
        this.projectService.saveProject(project)
            .subscribe(
                () => ctx.dispatch(new SessionStateActions.SaveProjectSuccess()),
                (error) => ctx.dispatch(new SessionStateActions.SaveProjectError({ error })))
    }
```

4. Функція в сервісі виглядає наступний чином:

```
    public saveProject(project: Project): Observable<any> {
        return this.http.post(`${environment.apiUrl}/project`, project)
            .pipe(
                map((response: any) => response.project),
            );
    }
```

HttpModule вбудований ангуляр модуль. Більшості інтерфейсних програм потрібно зв’язуватися з сервером за протоколом HTTP, щоб отримувати або зберігати ти дані та отримувати доступ до інших внутрішніх служб. Angular надає

клієнтський HTTP API для додатків Angular, клас HttpClient в @angular/common/http.

Даний сервіс HTTP пропонує наступні основні функції.

— Можливість надсилати запити

— Спрощена обробка помилок

— Тестування,

— Перехоплення запитів і відповідей

Можемо перевірити правильність надсилання даних за допомогою DevTools вкладки Network, зображеній на рисунку 3.20.

Chrome DevTools – це набір інструментів для веб-розробників, вбудованих безпосередньо у браузер Google Chromeю Такі інструменти наразі присутні у більшості популярних браузерів та в основному маюьт однаковий інструментарій для тестувальників та розробників. DevTools може допомогти редагувати сторінки «на льоту» та швидко діагностувати проблеми, переглядувати запити, аналізувати швидкість роботи додатку та багато іншого.

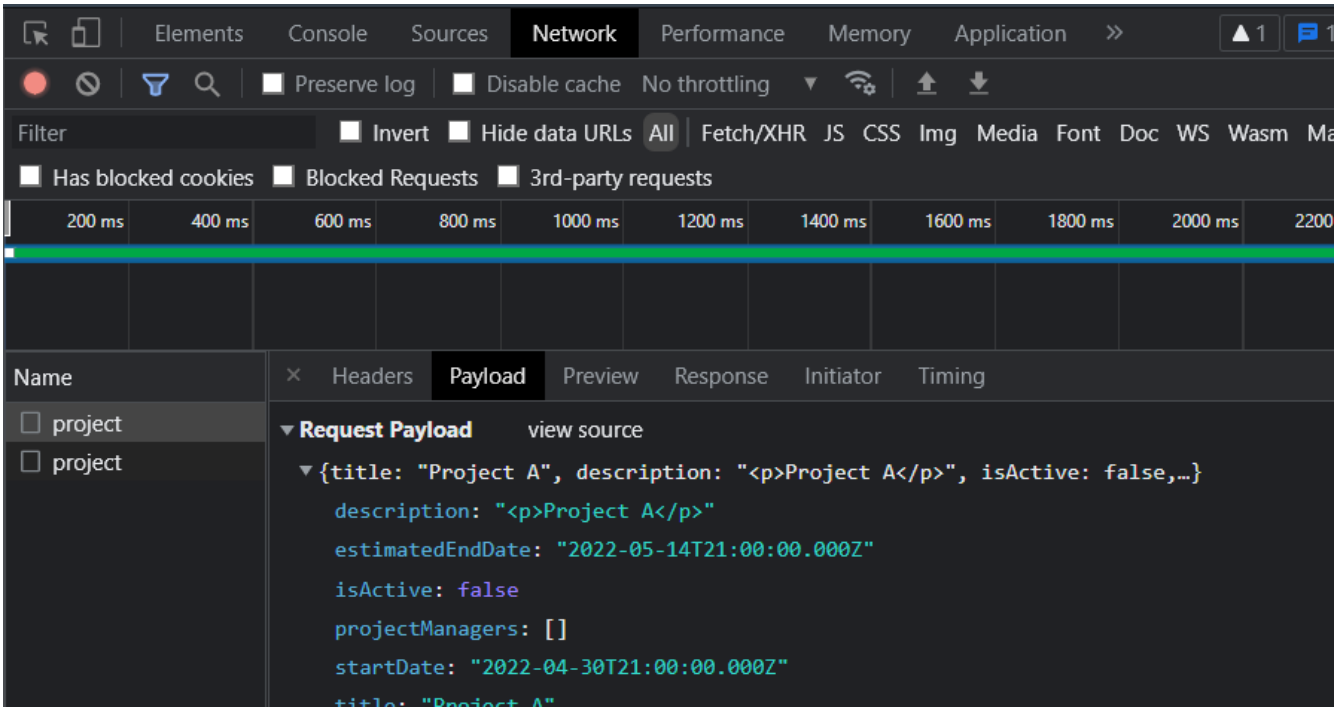


Рис. 3.20 Вкладка Network

5. У сервері на відповідний роутинг /project налаштовано оброблення GET запиту.

```
projectRouter.route('/project')
        .post(authMiddleware, ProjectController.createNewProject)
        .get(ProjectController.getProjectsList)
```

6. Контролер викликає функцію сервісу, що безпосередньо зберігає дані в базі даних.

```
export const createNewProject = async (req: Request, res: Response, next:
NextFunction) => {
        try {
                const body: IProject = req.body;
                const projectServiceInstance = Container.get(ProjectService);
                const savedProject = await
projectServiceInstance.saveProjectToDb(body);

                await projectServiceInstance.addUsersToProject(savedProject,
body.projectManagers, MemberRole.Manager);

                return res.status(HTTP_CODES.CREATED).json({
                        message: GeneralResponseMessage.Success,
                        project: savedProject,
                });
        } catch (err) {
                return next(err);
        }
};
```

7. Останнім кроком у даній взаємодіїї є збереження даних у базі даних.

```
        public saveProjectToDb(project: any) {
```

```
        return this.projectModel.create(project);
    }
```

3.4 Технічне забезпечення

Щоб отримати доступ до додатку та ***користуватись ним*** необхідно мати комп’ютер, ноутбук чи інший пристрій із виходом в Інтернет.

Розглянемо мінімальні та рекомендовані вимоги до обладнання:

Таблиця 3.1

	Мінімальні параметри	Рекомендовані параметри
Процесор - основний компонент пристрою, що здатний виконувати логічні чи арифметичні операції та управляти іншими компонентами	1,9 ГГЦ та два чи більше ядра (ядро – найголовніший елемент центрального процесора. Є частиною процесора, здатне виконувати один потік команд)	3,3 (ГГц) або 64-розрядний двоядерний процесор чи більше
Оперативна пам’ять – зберігає інформацію на короткий час у процесі її обробки	2ГБ	4 ГБ або більше
Дисплей	Super VGA (відеоадаптер та стандарт дисплеїв) з роздільною здатністю 1024 x 768	

У разі якщо вимоги не відповідатимуть мінімальним, сторінка браузера може працювати повільно або ж взагалі не відповідати.

Також обов’язковою вимогою для доступу до додатку є Інтернет-з’єднання та встановлений браузер.

Розглянемо браузери, що підтримуються та їх версії:

Таблиця 3.2

IE	Edge	Firefox	Safari	Opera
не підтримується	12 - 101 101	33-104	7.1 - 15.4, TP	20-87

Мобільна версія додатку не підтримується, тому неможливо користуватись ним із мобільних браузерів: Opera Mini, Safari Mobile та інші.

Також підтримка Internet Explorer і Microsoft Edge Legacy незабаром буде припинена компанією Microsoft, тому також немає підтримки для цих браузерів.

Для пристрою, що буде використовуватись як ***Node js сервер*** необхідними є наступні вимоги до обладнання:

— Принаймні 2 ГБ пам’яті RAM

— Принаймні 4 ГБ оперативної пам’яті

— Принаймні 25 ГБ пам’яті на диску

3.5 Результати реалізації інформаційної системи

У першому розділі роботи було досліджено предметну область, а саме поняття управління проектами, основні вимоги та задачі, які має виконувати така система. Розглянуто представлені на ринку програмні продукти та проаналізовано їх переваги та недоліки, основний функціонал.

З даного аналізу можна зробити висновок, що більшість існуючих програмних продуктів не мають повного інструментарію, часто зосереджені лише

на плануванні або тільки на управлінні задачами. Тож система має об’єднати в собі весь необхідний функціонал для управління проектами в одній системі: планування, управління ризиками, організація командної роботи, управління ресурсами проекту та задачами.

У другому розділім було побудовано контекстну IDEF0-діаграми та зроблено її декомпозицію для кращого вивчення та аналізу роботи системи, що розроблюється. Також було описано всі технології, що використовувались для розробки, навведено основні методи візуалізації у таких системах.

Третій розділ містить в собі інформацію про розробку даної системи та проектування, підключення її до бази даних. Розроблено модель ERD – візуальне представлення бази даних, що довзволяє дослідити зв’язок між таблицями.

Детально описано взаємодія інтерфейс-частини додатку та серверної частини додатку на прикладі створення проекту, розказано про доступний функціонал

У результаті дана система може використовуватись менеджерами, фрілансерами або командою осіб, що працюють над одним проектом для:

— Реєстрації користувачів та збір всієї команди в одному додатку
— Створення власних проектів
— Управління задачами
— Командна робота
— Швидкий доступ до інформації

Це дозволить набагато ефективніше, швидше виконувати задачі, досягати всіх цілей та працювати у команді задля успішного завершення проекту.

Планується подальша розробка та покращення даної системи, додавання нового функціоналу.

ВИСНОВКИ

Отже, оскільки більшість компаній ведуть свою роботу саме на основі проектів та кількість таких компаній в майбутньому тільки буде зростати система розробка системи управління проектами є дуже актуальною темою.

Нині використання інформаційних систем у бізнесі є необхідністю задля збереження конкурентоспроможності компанії, залучення нових клієнтів та ефективнішої і простішої роботи для працівників, що дозволить зосередити увагу на пошук та безпосередньо розробку нових переваг на компанію та збереже кошти, що в інакшому випадку витрачаються на рутинну роботу.

Використовуючи системи управління проектами можливо розробити проект та досягти всіх необхідних цілей для його успішного завершення в коротші строки та набагато ефективніше.

У даній роботі розроблено систему управління проектами: інтерфейс та серверну частину додатку. Інтерфейс частина написана за допомогою Angular – джаваскріпт фреймворку. Система надає можливість:

— зареєструвати користувача, використовуючи відомих провайдерів або ж за допомогою форми реєстрації
— створити новий проект
— переглянути канбан-дошку проекту
— створити нову задачу, при створенні можна вказати статус задачі
— назначити задачу на члена команди
— переглянути інформацію по задачі
— додати чи видалити члена команди, якщо у вас є права «власника»
— видалити або відредагувати задачу
— переглянути інформацію по всім проектам на головній сторінці

Досліджено приклади систем управління проектами, що представлені зараз на ринку, зроблено їх опис, визначення недоліків та переваг кожної із них.

Також було обгрунтовано вибір веб-додатку у наступній розробці. Описано основні технології, що будуть використовуватись. Для розробки системи управління проектами планується використати наступні технології:

— Angular
— Node js (express.js)
— Sequelize ORM
— база даних PostgreSQL

У ході написання кваліфікаційної магістерської роботи була досягнута поставлена мета, а саме створена система управління проектами, що можу використовуватись менеджерами, окремими командами чи фрілансерами для ефективнішого виконання задач із планування та організації командної роботи, управління ризиками та ресурсами.

Отже, системи управлінням проектами здатні систематизувати процеси управління, полегшити роботу та комунікацію команди, мінімізувати витрати та зусилля, що витрачаються. Передбачається, що в найближчі роки більшість сфер будуть працювати на основі проекту, тому ця тема є надзвичайно актуальною. У майбутньому планується подальша розробка додатку, додавання нового функціоналу.

СПИСОК ЛІТЕРАТУРИ

1. І Всеукраїнська студентська науково - технічна конференція "ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"УДК 004.414 Гуйда О. - ст. гр. СІм-51

2. The Firm of the Future, By James Allen, James Root and Andrew Schwedel April 12, 2017 URL: https://www.bain.com/insights/firm-of-the-future/

3. Проект // «Словники України» online https://lcorp.ulif.org.ua/dictua/

4. PERT, CPM, and Gantt. San Ramon, CA: Center for Project Management, 1990

5. Старченко Г. В. Управління проектами: теорія та практика : навч. посіб. / Г. В. Старченко. – Чернігів : видавець Брагинець О. В., 2018. – 306 с.

6. PMI (2010). A Guide to the Project Management Body of Knowledge p.27-35

7. Вільямс, П. Р. (2015). Візуальне управління проектами. Доповідь, представлена на PMI® Global Congress 2015 — EMEA, Лондон, Англія. Ньютаун-сквер, Пенсильванія: Інститут управління проектами.Карта розуму. (канбан, гант чарт, калнедар, діаграми та інше)

8. Top 20 Best Project Management Software in 2021: An Overview Erin Gilliam Haije URL: https://mopinion.com/top-20-best-project-management-software-an-overview

9. Бехтерев С. Майнд-менеджмент: Решение бизнес-задач с помощью интеллект-карт / Сергей Бехтерев; Под ред. Глеба Архангельского. — М.: Альпина Паблишерз, 2009. — 308 с.

10. Структурно-логічні схеми. Таблиці. Опорні конспекти. Есе. Навчальні презентації: рекомендації до складання : метод. посіб. для студ. / уклад. : Л. Л. Бутенко, О. Г. Ігнатович, В. М. Швирка. – Старобільськ, 2015. – 112 с.

11. «Тони Бьюзен. Интеллект-карты. Полное руководство по мощному инструменту мышления»: Манн, Иванов и Фербер; Москва; 2019

12. Стэнли Э. Портни. Управление проектами для "чайников" = Project Management For Dummies. — М.: «Диалектика», 2006. — С. 368.

13. Лапыгин Ю. Н. Управление проектами: от планирования до оценки эффективности. — М.: Омега-Л, 2008. — С. 252.

14. What is TypeScript? URL: http://www.typescriptlang.org/

15. About Node.js, and why you should add Node.js to your skill set? Training.com. Training.com . URL: http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html

16. MariaDB. URL: https://uk.wikipedia.org/wiki/MariaDB

17. Іванюта, П. В. Управління ресурсами та витратами: навч. посіб. / П. В. Іванюта, О. П. Лугівська ; за ред. С. М. Іванюти. – К. : ЦУЛ, 2009. – 320 с.

18. Ковшун, Н. Е. Аналіз та планування проектів: навч. посібник / Н. Е. Ковшун. – К.: Центр учбової літератури, 2008. – 344 с

19. Петруня Ю.Є. Прийняття управлінських рішень : навчальний посібник / [Ю. Є. Петруня, Б. В. Літовченко, Т. О. Пасічник та ін.] ; за ред. Ю. Є. Петруні. - [3-тє вид., переробл. і доп.]. - Дніпропетровськ: Університет митної справи та фінансів, 2015. - 209 с.

20. Бабаєв, В. М. Управління проектами: навч. пос. / В. М. Бабаєв. – Харків : ХНАМГ, 2006. – 244 с.

21. Калач, Г. М. Управління проектами: навч. посіб. / Г. М. Калач. – Ірпінь : Національний університет ДПС України, 2010. – 334 с.

22. Логачова, Л. М. Управління проектами: навч. посіб. / Л. М. Логачова, О. В. Логачова. – Суми : Університетська книга, 2011. – 208 с.

23. Ноздріна, Л. В. Управління проектами: підручник / Л. В. Ноздріна, В. І. Ящук, О. І. Полотай. – К. : Центр учбової літератури, 2010. – 432 с.

24. Поцелуев, Павел (04 березня 2015). "Огляд сервісів по управлінню проектами: Basecamp, Trello, Slack, Asana, Worksection". ain.ua (російською).

25. Яровая, Майя (13 листопада 2014). "Made in Ukraine: 5 українських таск-менеджерів для ефективного управління проектами"

26. "Тайм-трекінг. Зручний інструмент або каторга для співробітників?". http://delo.ua/ (російською).

27. Нікель, Ніа "Методики та інструменти управління командами в період кризи". URL: http://community.com.ua/

28. Restful API. URL: https://restfulapi.net

29. Project Management Institute. Pulse of the Profession 2020. URL: https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2020

30. Systems Engineering Fundamentals. URL: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf

31. 2021 Developer Survey. URL: https://insights.stackoverflow.com/survey/2021?utm_source=Google&utm_medium=Search

32. Node JS. URL: https://nodejs.org/en/about/

33. Проектний аналіз. Відп. ред. Москвін С. О. – К.: Лібра, 1999. – 368 с.

34. Тарасюк Г. М. Управління проектами: Навчальний посібник для студентів вищих навчальних закладів. – К.:Каравела, 2004. – 344 с.

35. Тян Р. Б. Управління проектами: Підручник / Р. Б. Тян, Б. І. Холод, В. А. Ткаченко. – Київ: Центр навч. літератури, 2003. – 224 с.

36. Батенко Л. П. Управління проектами: Навч. посібн./ Л. П. Батенко, О. А. Загородніх, В. В. Міщинська. – К.: КНЕУ, 2003. –231 с.

37. Збірник бізнес-планів з коментарями і рекомендаціями /за ред. Попова В. М. Видання четверте, перероб. і доп. – К.: ЦУЛ, КноРус, 2003. – 382 с.

38. Офіційна документація Ангуляр. URL: https://angular.io/docs

39. Офіційна документація Sequelize. URL: https://sequelize.org/docs/v6/getting-started/

ДОДАТОК А

Лістинг програми

Серверна частина

src\api\controllers\project.controller.ts

```
import { NextFunction, Request, Response } from "express";
import Container from "typedi";
import { HTTP_CODES } from "../../config/http-model";
import { IProject } from "../../models/project.model";
import { GeneralResponseMessage } from "../../models/response-messages";
import { MemberRole } from "../../models/team-members.model";
import { ProjectService } from "../../services/project.service";

export const createNewProject = async (req: Request, res: Response, next:
NextFunction) => {
  try {
      const body: IProject = req.body;
      const projectServiceInstance = Container.get(ProjectService);
      const savedProject = await projectServiceInstance.saveProjectToDb(body);

      await projectServiceInstance.addUsersToProject(savedProject,
body.projectManagers, MemberRole.Manager);

      return res.status(HTTP_CODES.CREATED).json({
          message: GeneralResponseMessage.Success,
          project: savedProject,
      });
```

```
    } catch (err) {
        return next(err);
    }
};

export const updateProject = async (req: Request, res: Response, next: NextFunction) => {
    try {
        const body: IProject = req.body;
        const projectId = req.params.id;
        const projectServiceInstance = Container.get(ProjectService);
        const project = await projectServiceInstance.updateProject(projectId, body);

        if (!project) {
            throw new Error(GeneralResponseMessage.NotFound);
        }

        return res.status(HTTP_CODES.OK).json({
            message: GeneralResponseMessage.Success,
            project,
        });
    } catch (err) {
        return next(err);
    }
};
```

```
export const getProjectById = async (req: Request, res: Response, next:
NextFunction) => {
  try {
    const projectId = req.params.id;
    const projectServiceInstance = Container.get(ProjectService);
    const project = await projectServiceInstance.findProjectById(projectId);

    if (!project) {
      throw new Error(GeneralResponseMessage.NotFound);
    }

    return res.status(HTTP_CODES.OK).json({
      project,
    });
  } catch (err) {
    return next(err);
  }
};

export const deleteProject = async (req: Request, res: Response, next:
NextFunction) => {
  try {
    const projectId = req.params.id;
    const projectServiceInstance = Container.get(ProjectService);

    await projectServiceInstance.deleteProject(projectId);

    return res.status(HTTP_CODES.OK).json({
```

```
                message: GeneralResponseMessage.Success,
            });
        } catch (err) {
            return next(err);
        }
    };

    export const getProjectsList = async (req: Request, res: Response, next:
NextFunction) => {
        try {
            const projectServiceInstance = Container.get(ProjectService);
            const projects = await projectServiceInstance.getProjectsList();

            return res.status(HTTP_CODES.OK).json({
                message: GeneralResponseMessage.Success,
                projects,
            });
        } catch (err) {
            return next(err);
        }
    };

    export const getAutoSuggestProjecst = async (req: Request, res: Response, next:
NextFunction) => {
        try {
            const { searchValue, limit } = req.body;
            const projectServiceInstance = Container.get(ProjectService);
```

```
            const projects = await
projectServiceInstance.getProjectsAutoSuggestList(searchValue, limit);

            return res.status(HTTP_CODES.OK).json({
                message: GeneralResponseMessage.Success,
                projects,
            });
        } catch (err) {
            return next(err);
        }
    };
```

src\api\controllers\users.controller.ts

```
    import { Request, Response, NextFunction } from 'express';
    import { UserInfo } from 'firebase-admin/auth';
    import Container from 'typedi';
    import { HTTP_CODES } from '../../config/http-model';
    import { GeneralResponseMessage } from '../../models/response-messages';
    import { UserService } from '../../services/user.service';

    export const createNewUser = async (req: Request, res: Response, next:
NextFunction) => {
        try {
            const body: UserInfo = req.body;

            if (!body) {
                throw new Error(GeneralResponseMessage.BadRequest);
            }
```

```
            const userServiceInstance = Container.get(UserService);
            const savedUser = await userServiceInstance.saveUserToDb(body);

            return res.status(HTTP_CODES.CREATED).json({
                message: GeneralResponseMessage.Success,
                user: savedUser,
            });
        } catch (err) {
            return next(err);
        }
    };

    export const listAllUsers = async (req: Request, res: Response, next: NextFunction)
=> {
        try {
            const userServiceInstance = Container.get(UserService);
            const response = await userServiceInstance.listAllUsers();
            return res.status(HTTP_CODES.OK).json({
                message: GeneralResponseMessage.Success,
                users: response.users,
            });
        } catch (err) {
            return next(err);
        }
    }
```

src\app\components\session\services\project.service.ts

```
import { NextFunction, Request, Response } from 'express';
import { HTTP_CODES } from '../../config/http-model';
import { GeneralResponseMessage } from '../../models/response-messages';

export const errorHandler = (err: any, req: Request, res: Response, _next:
NextFunction) => {
  const status = err.message === GeneralResponseMessage.NotFound ?
HTTP_CODES.NOT_FOUND : HTTP_CODES.INTERNAL_SERVER_ERROR;

  return res.status(status).json({
      message: err.message,
      status,
  });
};
```

src\api\middlewares\request-validation.middleware.ts

```
import { NextFunction, Request, Response } from 'express';
import { Schema } from 'joi';
import { HTTP_CODES } from '../../config/http-model';

export const validateRequest = (schema: Schema) => {
  return (req: Request, res: Response, next: NextFunction) => {
      const { error } = schema.validate(req.body);

      if (error) {
          const message = error?.details?.map((detail) =>
detail.message).join(',');
```

```
                return res.status(HTTP_CODES.BAD_REQUEST).json({
                    error: message,
                });
            }

            return next();
        };
    };
```

src\app\components\session\services\project.service.ts

```
import express from 'express';
import { createProjectSchema, updateProjectSchema } from
'../../validators/project.validator';
import * as ProjectController from '../controllers/project.controller';
import { validateRequest } from '../middlewares/request-validation.middleware';

export const projectRouter = express.Router();

projectRouter.route('/project/:id')
  .get(ProjectController.getProjectById)
  .patch(validateRequest(updateProjectSchema), ProjectController.updateProject)
  .delete(ProjectController.deleteProject);

projectRouter.route('/project')
  .post(ProjectController.createNewProject)
  .get(ProjectController.getProjectsList)
```

src\api\routers\user.router.ts

```
import express from 'express';
import * as UserController from '../controllers/user.controller';
import { validateRequest } from '../middlewares/request-validation.middleware';

export const userRouter = express.Router();

userRouter.route('/user')
   .get(UserController.listAllUsers)
   .post(UserController.createNewUser)
```

src\config\app.ts

```
import { config } from 'dotenv';

config();

export const POSTGRE_DB_URL =
`postgres://${process.env.PG_USER}:${process.env.PG_TOKEN}@${process.env.PG
_HOST}/${process.env.PG_USER}`;
export const SECRET = process.env.SECRET;
export const PORT = process.env.PORT;
```

src\loaders\app.loader.ts

```
import express from 'express';
import expressLoader from './express.loader';
import { dependencyInjectorLoader } from './dependency-injector.loader';
import { PORT } from '../config/app';
import { ProjectTable } from '../models/project.model';
import { UserTable } from '../models/user.model';
```

```
import { TeamMembersTable } from '../models/team-members.model';

export const appLoader = async () => {
  const app = express();

  expressLoader({ app });
  dependencyInjectorLoader();

  await ProjectTable.sync();
  await UserTable.sync();
  await TeamMembersTable.sync();

  app.listen(PORT, () => console.log(`Server is running on port ${PORT}...`));
};
```

src\loaders\dependency-injector.loader.ts

```
import { Container } from 'typedi';
import { ProjectTable } from '../models/project.model';
import { UserTable } from '../models/user.model';

export const dependencyInjectorLoader = async () => {
  try {
        Container.set('projectModel', ProjectTable);
        Container.set('userModel', UserTable);
  } catch (e) {
        console.error('Error on dependency injector loader: ', e);
  }
};
```

src\loaders\express.loader.ts

```
import express, { json } from 'express';
import cors from 'cors';

import { projectRouter, userRouter } from '../api/routers';
import { errorHandler } from '../api/middlewares/error-handler.middleware';

export default ({ app }: { app: express.Application }) => {
  app.use(cors());
  app.use(json());
  app.use('/api', projectRouter);
  app.use('/api', userRouter);

    app.use(errorHandler);
};
```

src\loaders\sequelize.loader.ts

```
import { Sequelize } from 'sequelize';
import { POSTGRE_DB_URL } from '../config/app';
import { ProjectTable } from '../models/project.model';
import { TeamMembersTable } from '../models/team-members.model';
import { UserTable } from '../models/user.model';

const sequelize = new Sequelize(POSTGRE_DB_URL, {
  dialect: 'postgres',
});
```

```
export const sequelizeAuthenticateLoader = async () => {
  try {
        await sequelize.authenticate();

        UserTable.belongsToMany(ProjectTable, { through: TeamMembersTable, foreignKey: 'projectId' });
        ProjectTable.belongsToMany(UserTable, { through: TeamMembersTable, foreignKey: 'userUid' });

        sequelize.sync();
  } catch (err) {
        console.error(`Unable to connect to the database: ${err}`);
  }
};
```

src\models\project.model.ts

```
import { DataTypes } from 'sequelize';
import { Sequelize } from 'sequelize';
import { POSTGRE_DB_URL } from '../config/app';

const sequelize = new Sequelize(POSTGRE_DB_URL, {
   dialect: 'postgres',
});

export enum AccessType {
   Public = 'public',
   Private = 'private',
}
```

```
export interface IProject {
    id: string;
    title: string;
    isActive: boolean;
    projectManagers: string[];
    accessType: AccessType;
    team: string[];

    description?: string;
    startDate?: Date;
    estimatedEndDate: Date;
    endDate?: Date;
    estimatedBudget?: string;
    usedBudget?: string;
}

export const ProjectTable = sequelize?.define('project', {
    id: {
        type: DataTypes.UUID,
        allowNull: false,
        primaryKey: true,
        defaultValue: DataTypes.UUIDV4,
    },
    title: {
        type: DataTypes.STRING,
        allowNull: false,
    },
```

```
description: {
   type: DataTypes.STRING,
},
projectManagers: {
   type: DataTypes.ARRAY(DataTypes.STRING),
   allowNull: false,
   defaultValue: [],
},
accessType: {
   type: DataTypes.STRING,
   defaultValue: AccessType.Public,
},
team: {
   type: DataTypes.ARRAY(DataTypes.STRING),
   defaultValue: [],
},
startDate: {
   type: DataTypes.DATE,
   defaultValue: new Date(),
},
estimatedEndDate: {
   type: DataTypes.DATE,
},
endDate: {
   type: DataTypes.DATE,
},
isActive: {
   type: DataTypes.BOOLEAN,
```

```
        defaultValue: true,
    },
    estimatedBudget: {
        type: DataTypes.STRING,
    },
    usedBudget: {
        type: DataTypes.STRING,
    },
});
```

src\models\team-members.model.ts

```
import { DataTypes } from 'sequelize';
import { Sequelize } from 'sequelize';
import { POSTGRE_DB_URL } from '../config/app';

const sequelize = new Sequelize(POSTGRE_DB_URL, {
    dialect: 'postgres',
});

export enum MemberRole {
    Manager = 'manager',
    TeamMember = 'teamMember',
}

export const TeamMembersTable = sequelize.define('teamMembers', {
    role: DataTypes.STRING,
    projectId: DataTypes.STRING,
    userUid: DataTypes.STRING,
```

```
});
```

src\services\project.service.ts

```
import { Op } from 'sequelize';
import { Inject, Service } from 'typedi';
import { IProject, ProjectTable } from '../models/project.model';
import { MemberRole } from '../models/team-members.model';

@Service()
export class ProjectService {
  constructor(
        @Inject('projectModel') private projectModel: typeof ProjectTable,
  ) { }

  public saveProjectToDb(project: any) {
        return this.projectModel.create(project);
  }

  public updateProject(projectId: string, project: Partial<IProject>) {
        return this.projectModel.update(project, { where: { id: projectId } });
  }

  public findProjectById(projectId: string) {
        return this.projectModel.findOne({ where: { id: projectId } });
  }

  public deleteProject(projectId: string) {
```

```
        return this.projectModel.update({ isDeleted: true }, { where: { id: projectId
} });
    }

    public getProjectsList() {
        return this.projectModel.findAll();
    }

    public getProjectsAutoSuggestList(searchValue: string, limit = 10) {
        return this.projectModel.findAll({
            where: {
                login: {
                    [Op.iLike]: `%${searchValue}%`,
                },
            },
            limit,
        });
    }

    public addUsersToProject(project: any, usersUids: string[], role: MemberRole) {
        return project?.addUsers(usersUids, { through: { role } });
    }
}
```

Інтерфейс частина додатку

src\app\app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
```

```
import { NgxsModule } from '@ngxs/store';
import { provideDatabase, getDatabase } from '@angular/fire/database';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { environment } from 'src/environments/environment';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { HttpClientModule } from '@angular/common/http';

import { AngularFireDatabaseModule } from '@angular/fire/compat/database';
import { CreateTaskFormComponent } from './components/create-task/create-task.component';
import { DragAndDropListsComponent } from './components/drag-and-drop-lists/drag-and-drop-lists.component';
import { TaskDescriptionComponent } from './components/task-description/task-description.component';
import { CreateProjectComponent } from './components/create-project/create-project.component';
import { ProjectsListComponent } from './components/projects-list/projects-list.component';
import { TeamPanelComponent } from './components/team-panel/team-panel.component';
import { NgxsLoggerPluginModule } from '@ngxs/logger-plugin';
import { SharedModule } from './shared/shared.module';
import { AngularFireModule } from '@angular/fire/compat';
import { NgxsReduxDevtoolsPluginModule } from '@ngxs/devtools-plugin';
import { initializeApp, provideFirebaseApp } from '@angular/fire/app';
```

```
import { provideAuth, getAuth } from '@angular/fire/auth';
import { AuthModule } from './components/auth/auth.module';
import { FontAwesomeModule } from '@fortawesome/angular-fontawesome';
import { SessionModule } from './components/session/session.module';

@NgModule({
  declarations: [
        AppComponent,
        CreateTaskFormComponent,
        DragAndDropListsComponent,
        TaskDescriptionComponent,
        CreateProjectComponent,
        ProjectsListComponent,
        TeamPanelComponent,
  ],
  imports: [
        BrowserModule,
        AppRoutingModule,
        AngularFireModule.initializeApp(environment.firebase),
        AngularFireDatabaseModule,
        AuthModule,
        NgxsReduxDevtoolsPluginModule.forRoot({
              disabled: environment.production,
        }),
        NgxsModule.forRoot([], {
              developmentMode: !environment.production
        }),
        NgxsLoggerPluginModule.forRoot({
```

```
            disabled: environment.production,
        }),
        FormsModule,
        ReactiveFormsModule,
        BrowserAnimationsModule,
        HttpClientModule,
        SharedModule,
        FontAwesomeModule,
        SessionModule,
        provideFirebaseApp(() => initializeApp(environment.firebase)),
        provideAuth(() => getAuth()),
        provideDatabase(() => getDatabase()),
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

src\app\components\auth\components\login-method\login-method.component.html

```
<div
```

```
        [classList]="!!loginMethod.disabled? 'pm-login-method pm-login-method--disabled':'pm-login-method'"
        (click)="loginMethodSelected.emit(loginMethod.value)"
      >
        <fa-icon
          [icon]="loginMethod.icon"
          [style.color]="loginMethod.disabled ? 'grey': 'red'">
        </fa-icon>

        <p class="pm-login-method__text">
          Sign in with <span class="pm-login-method__text--highlighted">{{ loginMethod.label }}</span>
        </p>
      </div>
```

src\app\components\auth\components\login-method\login-method.component.scss

```
      .pm-login-method {
        display: flex;
        align-items: center;
        width: 60%;
        margin: 0 auto;
        padding: 30px;

        &--disabled {
              opacity: 0.5;
              color: inherit;
              pointer-events: none;
```

```
  }

  &:hover {
        background-color: #f7f7f7;
        cursor: pointer;
  }

  fa-icon {
        font-size: 2rem;
        width: 100px;
  }

  &__text {
        display: inline-block;
        vertical-align: middle;
        padding: 0 24px;
        font-size: 14px;

        &--highlighted {
              font-weight: bold;
              color: #1967d2;
        }
  }
}
```

src\app\components\auth\components\root\auth-page.component.ts

```
import { Component } from '@angular/core';
import { Select, Store } from '@ngxs/store';
```

```
import * as AuthActions from '../../store/auth.actions';
import { AuthState } from '../../store/auth.state';
import { Observable } from 'rxjs';
import { LoginMethod, LoginPlatform } from '../../store/auth.state.models';

@Component({
  selector: 'pm-auth-page',
  templateUrl: './auth-page.component.html',
  styleUrls: ['./auth-page.component.scss']
})
export class AuthPageComponent {
  @Select(AuthState.loginMethods)
  public loginMethods$: Observable<LoginMethod[]>;

  @Select(AuthState.isLoading)
  public isLoading$: Observable<boolean>;

  constructor(
        private readonly store: Store,
  ) {
  }

  public login(loginPlatform: LoginPlatform): void {
        this.store.dispatch(new AuthActions.LoginViaExternalProvider({
loginPlatform }));
  }
}
```

```
export const LoginServiceToProviderMap: LoginServiceToProviderType = {
  [LoginPlatform.Google]: GoogleAuthProvider,
  [LoginPlatform.Github]: GithubAuthProvider,
  [LoginPlatform.Facebook]: FacebookAuthProvider,
  [LoginPlatform.Twitter]: TwitterAuthProvider,
};
```

src\app\components\auth\services\auth.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, from, of, catchError, map } from 'rxjs';
import { IUser } from '../../../interfaces';
import { AngularFireDatabase, AngularFireList } from
'@angular/fire/compat/database';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import firebase from 'firebase/compat/app';
import { LoginPlatform } from '../store/auth.state.models';
import { LoginServiceToProviderMap } from './auth.service.models';
import { HttpClient } from '@angular/common/http';
import { environment } from 'src/environments/environment';
import { UserInfo } from 'firebase/auth';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  // user: Observable<firebase.User> = this.angularFireAuth.authState;
  listOfUsers: AngularFireList<any> = this.db.list('users');

  constructor(
```

```
        private readonly angularFireAuth: AngularFireAuth,
        private readonly db: AngularFireDatabase,
        private readonly http: HttpClient,
    ) {
    }

    public loginUserViaExternalProvider(loginPlatform: LoginPlatform):
Observable<any> {
        const provider = LoginServiceToProviderMap[loginPlatform];
        return from(this.angularFireAuth.signInWithPopup(new provider()))
            .pipe(
                catchError(error => of(error))
            );
    }

    public logout(): Observable<any> {
        return from(this.angularFireAuth.signOut());
    }

    public getAllUsers(): Observable<any> {
        return this.http.get(`${environment.apiUrl}/user`)
            .pipe(
                map((response: any) => response.users),
            );
    }

    public saveUserToDb(user: UserInfo): Observable<unknown> {
        return this.http.post(`${environment.apiUrl}/user`, user)
```

```
                .pipe(
                    map((response: any) => console.log('[SAVE USER
RESPONSE]', response)),
                    catchError(error => {
                        console.log('[SAVE USER Error]', error)
                        return of(null);
                    }),
                );

  }
}
```

src\app\components\auth\store\auth.state.ts

```
import { Injectable } from '@angular/core';
import { Action, Selector, State, StateContext, Store } from '@ngxs/store';
import { AuthStateModel, LoginMethod } from './auth.state.models';
import * as AuthStateActions from './auth.actions';
import { SNACKBAR_MESSAGES } from 'src/app/shared/default-data';
import { AuthService } from 'src/app/components/auth/services/auth.service';
import { NotificationService } from 'src/app/services/notification.service';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { Router } from '@angular/router';
import { UserInfo } from 'firebase/auth';
import { from, Observable, tap } from 'rxjs';

@State<AuthStateModel>({
  name: 'authState',
  defaults: new AuthStateModel(),
```

```
})
@Injectable()
export class AuthState {
  constructor(
        private readonly store: Store,
        private readonly router: Router,
        private readonly angularFireAuth: AngularFireAuth,
        private readonly authService: AuthService,
        private readonly notificationService: NotificationService,
  ) {

  }

  @Selector()
  public static loginMethods(state: AuthStateModel): LoginMethod[] {
        return state.loginMethods;
  }

  @Selector()
  public static userInfo(state: AuthStateModel): UserInfo {
        return state.userInfo;
  }

  @Selector()
  public static usersList(state: AuthStateModel): UserInfo[] {
        return state.usersList;
  }
```

```
	@Selector()
	public static isLoading(state: AuthStateModel): boolean {
		return !!state.isLoading;
	}

	@Action(AuthStateActions.LoginViaExternalProvider)
	public loginViaExternalProvider(ctx: StateContext<AuthStateModel>,
		{ loginPlatform }: AuthStateActions.LoginViaExternalProvider,
	) { // TODO: check return type

		ctx.patchState({
			isLoading: true,
		});

		return
this.authService.loginUserViaExternalProvider(loginPlatform).subscribe(
			data => ctx.dispatch(new
AuthStateActions.LoginViaExternalProviderSuccess({ userInfo:
data?.additionalUserInfo?.profile })),
			error => ctx.dispatch(new
AuthStateActions.LoginViaExternalProviderError({ error }))
		);
	}

	@Action(AuthStateActions.LoginViaExternalProviderSuccess)
	public loginViaExternalProviderSuccess(ctx: StateContext<AuthStateModel>,
		{ userInfo }: AuthStateActions.LoginViaExternalProviderSuccess,
	): Observable<unknown> {
```

```
        this.notificationService.showMessage(SNACKBAR_MESSAGES.login_success)
;
            return this.authService.saveUserToDb(userInfo).pipe(
                tap(() => {
                    ctx.patchState({
                        userInfo,
                        isLoading: false,
                    });
                })
            )
        }

        @Action(AuthStateActions.LoginViaExternalProviderError)
        public loginViaExternalProviderError(ctx: StateContext<AuthStateModel>,
            { error }: AuthStateActions.LoginViaExternalProviderError,
        ): void {
            this.notificationService.showMessage(SNACKBAR_MESSAGES.error);

            ctx.patchState({
                error,
                isLoading: false
            });
        }

        @Action(AuthStateActions.CheckIfUserAuthorized)
        public checkIfUserAuthorized(ctx: StateContext<AuthStateModel>) {
            return this.angularFireAuth.authState
```

```
                .subscribe(user => {
                        const userInfo = JSON.parse(JSON.stringify(user));

                        if (userInfo?.email) {
                                ctx.dispatch(new AuthStateActions.SetUserInfo({
userInfo }))
                        } else {
                                this.router.navigate(['/auth']);
                        }
                });
        }

        @Action(AuthStateActions.SetUserInfo)
        public setUserInfo(ctx: StateContext<AuthStateModel>, { userInfo }:
AuthStateActions.SetUserInfo): void {
                ctx.patchState({
                        userInfo,
                });
                ctx.dispatch(new AuthStateActions.LoadUsersList());

                this.router.navigate(['/session/dashboard']);
        }

        @Action(AuthStateActions.Logout)
        public logout(ctx: StateContext<AuthStateModel>): void {
                ctx.patchState({
                        isLoading: true,
                });
```

```
        this.authService.logout().subscribe(
            data => ctx.dispatch(new AuthStateActions.LogoutSuccess()),
            error => ctx.dispatch(new AuthStateActions.LogoutError({ error }))
        )
    }

    @Action(AuthStateActions.LogoutSuccess)
    public logoutSuccess(ctx: StateContext<AuthStateModel>): void {
        ctx.patchState({
            isLoading: false,
            userInfo: null,
        });

        this.router.navigate(['/auth']);
    }

    @Action(AuthStateActions.LogoutError)
    public logoutError(ctx: StateContext<AuthStateModel>, { error }:
AuthStateActions.LogoutError): void {
        ctx.patchState({
            isLoading: false,
        });

        this.notificationService.showMessage(error);
    }

    @Action(AuthStateActions.LoadUsersList)
```

```
    public loadUsersList(ctx: StateContext<AuthStateModel>) {
        ctx.patchState({
            isLoading: true,
        });

        return this.authService.getAllUsers()
            .subscribe(
                usersList => ctx.dispatch(new
AuthStateActions.LoadUsersListSuccess({ usersList })),
                error => ctx.dispatch(new
AuthStateActions.LoadUsersListError({ error }))
            );
    }


    @Action(AuthStateActions.LoadUsersListSuccess)
    public loadUsersListSuccess(ctx: StateContext<AuthStateModel>, { usersList }:
AuthStateActions.LoadUsersListSuccess) {
        ctx.patchState({
            isLoading: false,
            usersList,
        });
    }


    @Action(AuthStateActions.LoadUsersListError)
    public loadUsersListError(
        ctx: StateContext<AuthStateModel>,
```

```
        { error }: AuthStateActions.LoadUsersListError,
    ): void {
        ctx.patchState({
            isLoading: false,
        });

        this.notificationService.showMessage(error);
    }
}
```

ДОДАТОК Б

МОЖЛИВОСТІ ТА ПЕРЕВАГИ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ІЗ УПРАВЛІННЯ ПРОЕКТАМИ

Когут Діана Леонідівна
здобувач вищої освіти Інституту інформаційних технологій в економіці
Київського економічного університету імені Вадима Гетьмана
Науковий керівник: Устенко Станіслав Веніамінович
доктор економ. наук, проф. кафедри інформаційних систем в економіці
Державний вищий навчальний заклад «Київський національний економічний університет імені Вадима Гетьмана»
Україна

У нинішній час управління проектами є надзвичайно зручним та корисним інструментом при розробці проектів, а за даними консалтингової компанії Bain & Company, до 2027 року більшість робіт буде виконуватись саме на основі проектів.[1]

Традиційно управління проектами – це організаційний інструмент, який використовується для досягнення окремих цілей, які можуть включати запуск окремого продукту чи послуги або досягнення певного результату.

Проте в останні роки роль управління проектами в багатьох організаціях почала розширюватися. Управління проектами – це більше, ніж просто інструмент для досягнення окремих цілей, ця методика також застосовується в контексті стратегії та ініціатив компанії.

Основними компонентами управління проектами є:

— визначення причини необхідності проекту;

— визначення вимог до проекту, визначення якості результатів, оцінка ресурсів і термінів;

— підготовка бізнес-кейсу для обґрунтування інвестицій;

— розробка та впровадження плану управління проектом;

— управління та мотивація команди, що працює над проектом;

— управління ризиками, моніторинг виконання плану;

— управління бюджетом проекту;

— підтримка комунікацій із зацікавленими сторонами та проектною організацією;

Для полегшення та автоматизації роботи над управлінням проектами можливо використовувати *системи з управління проектами*. У цей час на ринку представлено багато програм для управління проектами, кожна з яких може вирішувати як низку задач так і бути розробленою для конкретного етапу в управлінні.

Використання таких систем допоможе у налагодженні комунікації, адже вся переписка та обговорення щодо проектів ведеться у одній системі.

До переваг такого програмного забезпечення відноситься також візуалізація робочих процесів за допомогою таблиць, графіків, календарів, списків та інше. В останній час популярними також є канбан дошки, які зазвичай містять в собі такі категорії як «Потрібно зробити», «У процесі», «Зроблено», але можуть доповнюватись різними списками.

При використанні системи управління проектами легше систематизувати та проаналізувати інформацію про час, що витрачається на задачі, існує чітке розуміння дедлайнів як для планування проекту для проектних менеджерів так і для команди в цілому.

Також до переваг таких систем відносять управління бюджетом проекту, можливість планувати задачі наперед, систематизація всіх робочих проектів, створення чітких та зрозумілих звітів, графіків, таблиць.

В Україні найкраще себе зарекомендували такі сервіси для управління проектами:

Worksection – відмінно підходить для компаній, що прагнуть систематизувати задачі з управління проектами. Представлені такі функції: планування проектів та відслідковування процесу, робота з клієнтом та фрілансерами, контроль співробітників, а саме можливість використовувати часову ставку та слідкувати на які задачі та скільки часу співробітник витратив. [2]

Atlassian JIRA – система, що використовується для відстеження всіх задач та помилок у проекті, призначення конкретних завдань на учасника команди, організація спілкування з користувачами. Наразі є три підсистеми: JIRA Software для розробників, JIRA Service Desk - для підтримки проектів та JIRA Core (для безпосереднього управління проектами) [2].

Wrike – онлайн сервіс для організації роботи команди, дозволяє планувати проекти, виставляти пріоритетність задачам та контролювати процес їх виконання.

Basecamp – дуже легка в освоєнні система, що має гарний і швидкий веб-інтерфейс, загалом використовується невеликими компаніями, адже в ній відсутня функція баг-трекінгу, немає можливості розробляти звіти, спілкування відбувається лише у вигляді блогу, тому бракує функціоналу для використання у великих проектах.

Висновок: отже, системи управлінням проектами здатні систематизувати процеси управління, полегшити роботу та комунікацію команди, мінімізувати витрати та зусилля, що витрачаються. Передбачається, що в найближчі роки більшість сфер будуть працювати на основі проекту, тому ця тема є надзвичайно актуальною.