МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2022 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Програмна система управління ІТ проєктами"

Виконавець: Шемідько Артем Олександрович

Керівник: д.т.н., професор Воронін Альберт Миколайович

Нормоконтролер: _____ Олександр ШЕВЧЕНКО

Київ – 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Освітній ступінь: "Бакалавр"

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології".

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Аліна САВЧЕНКО

«____»_____2022р.

ЗАВДАННЯ

на виконання дипломного проєкту студента

Шемідько Артема Олександровича

(прізвище, ім'я, по батькові)

1. Тема роботи: «Програмна система управління ІТ проєктами» затверджена наказом ректора від 29.04.2022 за № 454/ст.

2. Термін виконання роботи: з 09.05.2022 по 13.06.2022.

3. Вихідні дані до роботи: принципи та методики (методології) проєктного керування, ключові підходи організації даних проєкту, інтеграція з іншими системами та проєктна аналітика.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, розгляд основних проблем, підходів та практик керування проєктами, огляд загальнодоступних програмних комплексів проєктного менеджменту, дослідження архітекнурних рішень та технологій розробок, створення програмного продукту керування проєктами, завданнями в них, управління власним графіком, аналітика прогресу, оцінка якості технології, висновки.

5. Перелік обов'язкового ілюстративного матеріалу: слайди, презентація.

6. Календарний план-графік

№ п/п	***Завдання***	***Термін виконання***	***Підпис керівника***
1.	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт	09.05.22 – 10.05.22р.	
2.	Огляд та аналіз відомих програмних архітектур і патернів	11.05.22 – 12.05.22р.	
3.	Проведення консультації з науковим керівником щодо плану розділів.	12.05.22 – 13.05.22р.	
4.	Розробка розділу 1	13.05.22 – 17.05.22р.	
5.	Розробка розділу 2	18.05.22 – 23.05.22р.	
6.	Розробка розділу 3	24.05.22 – 29.05.22р.	
7.	Оформлення та друк пояснювальної записки	30.05.22 – 02.06.22р.	
8.	Оформлення супровідних документів. Створення презентації, доповіді та підготовка до захисту дипломної роботи	03.06.22 – 13.06.22р.	

Дата видачі завдання: 09.05.2022р.

Керівник дипломної роботи _____ Альберт ВОРОНІН
(підпис керівника)

Завдання прийняв до виконання _____ Артем ШЕМІДЬКО
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмна система управління ІТ проєктами» складається із вступу, трьох розділів, трьох висновків, списку використаних джерел і містить 64 сторінок тексту та 39 рисунків. Список використаних джерел містить 17 найменувань.

Метою дипломного проекту розробка прототипу програмної системи для управління ІТ проєктами спектру розробки програмного забезпечення.

Об'єктом дослідження є процес керування ІТ проєктами, системи контролю часу, супроводження програмних систем та системи для проєктного менеджменту.

Предметом дослідження є підвищення ефективності роботи проєктного менеджера і команди при розробці проєкту шляхом автоматизації, за допомогою встановлених методологій та вбудованих програмних систем управління.

Результат проєкту: розроблена стільникова програмна система для керування проєктами та власного планування.

PM, WORKFLOW, КЕРУВАННЯ ПРОЕКТАМИ, ЕФЕКТНИВНІСТЬ, АНАЛІТИКА, ПРОГРАМНИЙ КОМПЛЕС, МЕТОДОЛОГІЇ КЕРУВАННЯ, АРХІТЕКТУРА, ШАБЛОНИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ	–	Програмне забезпечення
РМ	–	Проектний менеджер
МП	–	Мова програмування
WATERFALL	–	Філософія ітераційного переходу управління ПЗ
AGILE	–	Філософія гнучкого управління ПЗ
SCRUM	–	Підхід управління проєктами для гнучкої розробки
LEAN	–	Філософія управління з орієнтацією на інтереси і потреби
KANBAN	–	Метод управління РМ за допомогою дошок і карток
ФРЕЙМВОРК	–	Програмний комплекс програмних рішень
QT	–	Кросплатформенний фреймворк розробки ПЗ
QML	–	Декларативна МП, яка базується на JavaScript
SQL	–	Декларативна мова програмування для взаємодії користувача з базами даних
UI	–	Інтерфейс користувача

ВСТУП

В дипломній роботі проведено дослідження методологій та принципів управління проєктами, способів ефективних систем та планування в них та можливостей програмних комплексів PM. Здійснено розробку спрощеної програмної системи базового функціоналу для управління IT проєктами спектру розробки ПЗ.

Кожен проєкт, незалежно від поставлених завдань, його складності та доступних ресурсів, стикається з одними і тими ж проблемами, які полягають у ефективному та покроковому плануванні завдань, зручному керуванні та контролю процесів, отримання принаймні базової аналітики та прогнозів розробки. Існує багато різних методологій управління проєктами, які покликані вирішувати вище поставленні задачі, і всі вони мають як свої переваги, так і недоліки.

Деякі з цих принципів стали адаптуватись до сучасних потреб і інтегруватись в різні програмні комплекси, які наразі кожен може використовувати для вирішення різного спектру задач: починаючи від керування власного розпорядку дня та включно до оцінки та розподілення ресурсів між процесами відділів будь-якої компанії. Такі інструменти управління проєктами допомагають бізнесу розвиватися, правильно організовуючи етапи створення кожного продукту.

У теоретичних дослідженнях роботи розглядаються принципи та методики керування проєктами та можливості програмних комплексів з управління проєктами для досягнення ефективності використання ресурсів, які були використані при розробці подібних програм. Основна мета теоретичного дослідження полягає у визначенні переліку проблем, які програмний комплекс повинен вирішувати та систем, які зменшать навантаження на PM та нададуть можливість аналізувати тренди виконання проєктів, тим самим зменшуючи ризики.

Експериментальні дослідження полягають в огляді існуючих рішень, їх переваг та недоліків, зосереджуючи увагу на відсутності необхідного функціоналу, який покращить досвід як керівника так і розробника в проєкті та перевірці продуктивності, стабільності та інформативності власних розроблених систем, з

урахуванням специфіки діяльності для продукту, шляхом тестування системи, аналізу та відгуку користувачів.

Наукова новизна полягає у створенні повноцінного продукту для вирішення проблем ефективного керування проєктами, зменшуючи при цьому навантаження на керівників та виконавців, надати можливість отримувати аналітичні дані у вигляді графіків для оцінки виконаної роботи та формування прогнозів. Також є важливим інтеграція з іншими системами, які використовуються в IT проєктах, наприклад, інтеграція з GitHub для отримання доступу до кодової бази проєкту.

Практична значимість полягає у розробленні програмного комплексу для управління IT проєктами з урахуванням специфіки індустрії та систем, які використовуються при розробці.

Використання розробленого застосунку дозволить оптимальним чином розподіляти задачі між виконавцями, перерозподіляти навантаження та контролювати загальний та власний прогрес.

Програмний комплекс керування проєктами повинен містити такий базовий функціонал: базу даних користувачів з можливістю їх авторизації та додавання нових, навігаційну систему застосунку, створення, редагування та видалення Workflow (проєктів), створення груп виконавців (запрошення до Workflow), візуалізовану систему створення, редагування, видалення та переміщення завдань (Kanban) та щоденний план. Додатковою функціональністю може стати можливість формування аналітичних даних та перегляд метрик, для швидкого аналізу успішності виконання проєктів, а також інтеграція з іншими системами, необхідними для розробників (виконавців). Обов’язковим є захист даних користувачів, тому для цього повинен створюватись шифрованих канал з’єднання та передачі даних (ssl) та збереження чутливих даних (поштові скриньки, паролі та інші) у зашифрованому вигляді.

В результаті проведеної роботи буде отримано систему, яку можна буде повноцінно використовувати в якості програми для планування власного часу чи часу проектної команди.

РОЗДІЛ 1
ПРИНЦИПИ ТА МЕТОДИКИ КЕРУВАННЯ ПРОЄКТАМИ. МОЖЛИВОСТІ СИСТЕМ УПРАВЛІННЯ

1.1. Ключові аспекти проєктного менеджменту

Проєкт – це тимчасове та унікальне зусилля, призначене для виробництва продукту, послуги або результату з визначеним початком і кінцем (зазвичай обмеженим у часі та часто обмеженим фінансуванням або персоналом), яке здійснюється для досягнення унікальних цілей і завдань, як правило, для досягнення корисних змін або доданої вартості. Тимчасовий характер проєктів відрізняється від звичайного бізнесу (або операцій), що є повторюваною, постійною або напівпостійною функціональною діяльністю з виробництва продуктів або послуг. На практиці управління такими різними виробничими підходами вимагає розвитку окремих технічних навичок і стратегій управління.

Управління проєктом або проєктний менеджмент (англ. project management) — це процес керівництва роботою команди для успішного досягнення всіх цілей проєкту та виконання поставлених завдань в рамках заданих обмежень. Ця інформація зазвичай описується в проєктній документації, створеній на початку процесу розробки. Основними обмеженнями є обсяг завдань, час на виконання і бюджет. Інша проблема, яку вирішує проєктний менеджмент, полягає в оптимізації розподілу необхідних вхідних даних і ефективному застосуванні їх для досягнення заздалегідь визначених цілей.

Метою управління проєктами є повноцінне (завершене) створення певного продукту, який відповідає цілям клієнта. У багатьох випадках метою є також формування або реформування лаконічного опису завдання для реального досягнення

цілей. Після того, як цілі клієнта чітко визначені, вони повинні впливати на рішення, прийняті іншими людьми, які беруть участь у проєкті, наприклад, керівниками проєкту, дизайнерами, підрядниками та субпідрядниками. Неправильно визначені або занадто чітко прописані цілі управління проєктом завдають шкоди прийняттю рішень.

Ключовим фактором, який відрізняє управління проєктами від просто «управління», є те, що воно має кінцевий результат і обмежений період часу, на відміну від управління, яке є постійним процесом. Через це фахівець проєкту потребує широкого спектру навичок; часто технічні навички, і, безумовно, навички управління людьми та хороша ділова обізнаність.

Основними компонентами управління проєктами є:

- визначення причини, чому проєкт необхідний;
- визначення вимог, якості результатів та оцінка ресурсів і термінів
- розробка та впровадження плану управління проєктом;
- управління ризиками, питаннями та змінами в проєкті;
- моніторинг виконання плану;
- управління бюджетом та інші.

Саме від якості формування та дотримання цих компонентів і залежить успішність та якість створення повноцінного продукту.

Загальним серед усіх типів управління проєктами є те, що вони зосереджені на трьох важливих цілях: час, якість та вартість. Успішні завершуються за графіком, у межах бюджету та відповідно до попередньо узгоджених стандартів якості, тобто відповідності трикутнику чи потрійному обмеженню, що визначає успіх чи невдачу проєкту.

Для кожного типу управління проєктами менеджери розробляють і використовують повторювані шаблони, які є специфічними для галузі, з якою вони мають справу. Це дозволяє планам стати дуже ретельними і дуже повторюваними, з особливим наміром підвищити якість, знизити витрати на доставку та скоротити час для досягнення результатів проєкту.

1.2. Методології управління проєктами

Методологія управління проєктами — це система принципів, прийомів і процедур, які використовуються тими, хто працює в певній дисципліні. Найпопулярніші методології відрізняються не тільки структурною організацією, але й вимагають різних результатів, робочих процесів і навіть розробки програмного забезпечення для управління проєктами. По суті, це процеси, які мають на меті допомоги менеджерам проєктів, щоб дати рекомендації в будь-якому випадку та кроках, які необхідно здійснити для виконання завдань. Різні методології мають різні стратегії, які допомагають вирішити проблеми, якщо вони виникнуть під час реалізації проєкту.

Враховуючи різні цілі, ключові показники ефективності та методи виробництва не лише різних типів команд, а й різних типів галузей , має сенс, що не існує універсального підходу до управління проєктом. Те, що найкраще працює для одного типу команд, може стати абсолютним жахливим сценарієм для іншої.

Наприклад, багато розробників програмного забезпечення почали виявляти, що традиційні методи управління заважають, а не допомагають, їх робочим процесам і негативно впливають на їхню продуктивність і результати.

В результаті команди почали розробляти новий тип методології управління проєктами, яка була розроблена для вирішення їхніх конкретних проблем.

Незабаром інші команди та галузі почали адаптувати ці нові методи управління проєктами відповідно до своїх унікальних потреб і проблем. І так далі й далі, коли різні методології управління проєктами переробляються та адаптуються для різних галузей та налаштовуються відповідно до конкретних випадків використання.

До основних типів методологій можна віднести Waterfall, Agile, Scrum, Kanban, Scrumban, eXtreame programming, Lean, Six Sigma, PRINCE2 та інші.

1.2.1. Методологія Waterfall

Це є одна з найбільш простих і лінійних серед усіх методів управління проєктами, а також являє собою найбільш традиційний підхід. Для досягнення такого підходу кожне робоче завдання пов’язано залежністю. Це означає, що у ній кожні завдання та фази виконуються лінійно, і кожен етап проєкту має бути завершеним до початку наступного. Це не лише забезпечення стабільності протягом роботи, але й упродовж усього процесу. За допомогою цього методу окремі команди виконання не зобов’язані постійно спілкуватися і, якщо не потрібна конкретна інтеграція, зазвичай є автономними.

Фази проводять в наступному порядку:

- системні та програмні вимоги: залежить від віри в те, що всі вимоги проєкту можна зібрати та зрозуміти заздалегідь. Керівник проєкту робить все можливе, щоб отримати детальне розуміння вимог спонсора проєкту. Письмові вимоги, які зазвичай містяться в одному документі, використовуються для опису кожного етапу, включаючи витрати , припущення, ризики , залежності, показники успіху та терміни завершення;

- дизайн: розробники програмного забезпечення розробляють технічне рішення проблем, визначених вимогами до продукту, включаючи сценарії, макети та моделі даних. Спочатку створюється проєкт вищого рівня або логічний, який описує мету та обсяг проєкту , загальний потік трафіку кожного компонента та точки інтеграції. Як тільки це завершується, він перетворюється на фізичний дизайн з використанням спеціальних апаратних і програмних технологій;

- реалізація: після завершення проєктування починається технічна реалізація. Це може бути найкоротший етап процесу водоспаду, тому що ретельні дослідження та проєктування вже зроблені. На цьому етапі програмісти кодують програми на основі вимог та специфікацій проєкту, а також проводять деяке тестування та впровадження. Якщо на цьому етапі потрібні значні зміни, це може означати повернення до фази проєктування;

- перевірка чи тестування: перш ніж продукт може бути випущений клієнтам, необхідно провести тестування, щоб переконатися, що продукт не має помилок і виконано всі вимоги, що забезпечує хороший досвід роботи з програмним забезпеченням. Команда тестування звернеться до проєктних документів, персон і сценаріїв прикладів користувачів, наданих менеджером продукту, щоб створити свої тестові приклади;

- розгортання та обслуговування: після того, як програмне забезпечення було розгорнуто на ринку або випущене клієнтам, починається етап обслуговування. У міру виявлення дефектів і надходження запитів на зміни від користувачів буде призначена команда, яка подбає про оновлення та випуск нових версій програмного забезпечення.

До переваг даної методології можна віднести обізнаність кожного учасника системи в тому, що йому потрібно зробити, тим самим він може ефективно планувати свій час, так як вимоги чітко формуються на самому початку; розробники можуть виявити помилки дизайну на етапах аналізу та проєктування, допомагаючи їм уникнути написання несправного коду на етапі впровадження; загальна вартість проєкту може бути точно оцінена, як і терміни, після визначення вимог. Також вагомим є можливість легкої інтеграції додаткових розробників в процесі розробки, тобто увійти в проєкт, що вже триває.

В той же час можна відмітити, що дана методологія має недоліки в тому, що часто, так як вимоги формуються на початку, що робить її не гнучкою, не можуть формувати чітко своїх потреб і може стати необхідним вносити зміни в процесі розробки. Часто це може бути неможливим, або потребувати значних витрат. Також така побудова процесу у випадку запізнення одного з термінів, призводить до зміщення в термінах і всіх інших етапів.

1.2.2. Методологія Agile

Це ітеративний підхід до управління проєктами, який зосереджується на роздроблені великих проєктів на більш керовані завдання, які виконуються за короткі кроки протягом життєвого циклу проєкту . Команди, які використовують цю методологію, можуть виконувати роботу швидше, адаптуватися до мінливих вимог проєкту та оптимізувати свій робочий процес. Дана методологія дозволяє командам переоцінювати роботу, яку вони виконують, і коригувати з певними кроками, щоб переконатися, що в міру того, як змінюється робота та клієнтський ландшафт, змінюється і фокус для команди. Для Agile першочерговим завданням є задовольнити клієнта шляхом своєчасної та безперервної доставки цінного програмного забезпечення. На відміну від попереднього методу, зміни вимог, навіть на пізніх стадіях розробки не є критичними.

До основних етапів відносять:

- планування проєкту: перед початком роботи команда повинна зрозуміти кінцеву мету, цінність для організації чи клієнта та спосіб її досягнення. На цьому етапі можна розробити загальний вигляд проєкту, але обсяг проєкту не повинен розглядатися як незмінний;
- створення дорожньої карти: це поділ функцій, які складатимуть кінцевий продукт. Це важливий компонент на етапі планування, оскільки команда створюватиме ці окремі функції під час кожного спринту;
- планування розгортання: перш ніж розпочати проєкт, складається план випуску функцій високого рівня, а на початку кожного спринту переглядається та переоцінюється план випуску кожної з них;
- планування спринту: перед початком кожного спринту зацікавлені сторони проводять нараду з планування спринту, щоб визначити, що буде досягнуто кожною особою під час цього спринту, як це буде досягнуто, і оцінити навантаження на завдання. Важливим є рівномірне розподілення навантаження між членами команди, щоб вони могли виконувати поставлені завдання під час спринту. Також потрібно буде візуально документувати робочий процес для забезпечення прозорості

команди, спільного розуміння всередині команди, а також виявлення та усунення вузьких місць;

- щоденні стендапи: щоб допомогти команді виконувати свої завдання під час кожного спринту та оцінити, чи потрібно вносити будь-які зміни, проводяться короткі щоденні зустрічі, під час яких кожен член команди коротко розповість про те, чого вони досягли напередодні та над чим працюватимуть цього дня.

- огляд та ретроспектива: після закінчення кожного спринту команда проводить дві зустрічі: спочатку огляд спринту зі зацікавленими сторонами проєкту, щоб показати їм готовий продукт. Це важлива частина відкритого спілкування із зацікавленими сторонами. Потім – проводиться зустріч по ретроспективі спринту зі зацікавленими сторонами, щоб обговорити, що було добре під час спринту, що могло бути краще, чи було навантаження на завдання занадто важким чи занадто легким для кожного учасника, і що було досягнуто під час спринту.

Завдяки цим функціям він зарекомендував себе як методологія, яка робить акцент на співпраці, гнучкості, постійному вдосконаленні та високоякісних результатах.

1.2.3. Методологія Scrum

Scrum — це ітеративний підхід до управління проєктами, що означає, що він призначений для швидкої доставки дрібних речей, які з часом можуть перетворитися на щось більше. Кожна ітерація, відома як спринт, триває всього пару тижнів. Команда, яка відповідає за фактичне виконання роботи, має вирішувати, скільки вона зобов’язується виконати під час спринту, і ніхто не повинен змінювати чи змінювати пріоритети свого робочого навантаження, коли вони взяли це зобов’язання.

Наприкінці кожного спринту команда повинна створити щось, що можна було б випустити для своєї аудиторії, хоча це не обов’язково.

Scrum робить акцент на швидких випусках, щоб дізнатися, що працює. Використання ітерацій через Scrum дозволяє робити це швидше і з меншим ризиком,

ніж традиційні методології управління проєктами, які вимагають величезного попереднього планування.

Концепція, яка лежить в основі Scrum, як і більшості Agile-менеджменту проєктів, проста, але за багато років і незліченних реалізацій вона стала складнішою.

Кожний протокол Scrum відіграє особливу роль, допомагаючи керувати роботою Scrum, і кожен член команди несе відповідальність у чотирьох з них:

- планування спринту;
- щоденний стендап;
- огляд спринту;
- ретроспектива.

Scum унікальний з ряду причин, однією з яких є використання Scrum-майстра. Або, іншими словами, менеджер проєкту, який веде щоденні зустрічі Scrum, демонстрації, спринти та ретроспективи спринту після завершення кожного спринту. Ці зустрічі мають на меті об’єднати зацікавлені сторони проєкту та забезпечити своєчасне виконання завдань.

1.2.4. Методологія Lean

Ця методологія управління проєктами, яка максимізує цінність для клієнтів і мінімізує відходи за рахунок оптимізації потоку матеріалів або завдань через систему в цілому. Керівники та практики Lean-проєктів прагнуть постійного підвищення ефективності шляхом тестування та аналізу різних ідей та модифікацій існуючих практик.

Філософія Lean бере свій початок у виробничій системі Toyota, розробленій після Другої світової війни, яка спонукала співробітників бути частиною рішення для постійного перегляду методів і процесів.

Сьогодні методологія Lean поширилася на всі види бізнесу, а не лише на виробництво. Ідеї, отримані в результаті Lean-процесу, виявилися цінними в стратегіях управління проєктами в багатьох секторах, дозволяючи розширити можливості оптимізації та подолання потенційних підводних каменів .

Lean управління проєктами може допомогти командам досягти кількох цілей одночасно, зокрема:

- зменшення загальних витрат;
- підвищення продуктивності;
- узгодження зусиль із задоволеністю;
- збільшення командної роботи та спільного мислення.

1.3. Можливості програмних комплексів проєктного управління

Існує п’ять основних принципів управління проєктами Lean. Кожен принцип спирається на останній, утворюючи цикл, який дозволяє постійно переглядати та вдосконалювати будь-який процес:

- визначення цінності в очах замовника: цінність може бути визначена тим, за що клієнти готові платити. Але оскільки клієнти часто не можуть сформулювати те, що вони хочуть, чітке визначення цінності не завжди є простим і зрозумілим процесом. Інтерв’ю та опитування можуть допомогти команді скласти більш чітке уявлення про очікування клієнтів щодо послуги чи продукту;
- визначення потоку створення цінності для кожного продукту: у рамках цього процесу у виробничому сценарії можна запитати, як отримати матеріали, куди вони поставляються, які інструменти потрібні для роботи з матеріалами, які етапи виробництва, як підтримується якість, чим результат у процесі відрізняється від замовлення тощо. У розробці програмного забезпечення, процес може визначати етапи проєктування, розробки, тестування та перегляду, а також уважного перегляду, хто виконує кожен крок і які їхні реалізувати;
- визначення цінності як потік і усунення відходів: визначивши всі частини потоку створення цінностей, можна усунути будь-які частини процесу, які зараз вважаються марними, і працювати над створенням більш плавного потоку. Щоб це сталося, може знадобитися розбити та переналаштувати різні кроки, перерозподілити робоче навантаження та перенавчити співробітників;

- примушення клієнта визначати потік: ідея цього принципу полягає в тому, що потреби замовника повинні тягнути потік роботи. Іншими словами, замість того, щоб завчасно купувати ресурси та матеріали або завчасно переробляти продукцію — і те й інше може призвести до додаткових витрат на зберігання чи матеріальних відходів — робочий процес ініціюється та узгоджується безпосередньо з потребами клієнта, при цьому команда замовляє матеріали лише тоді, коли необхідно і продукт генерується вчасно;

- намагання вдосконалення в прагненні до досконалості: співробітники, які беруть участь у всіх етапах процесу, повинні прагнути до постійного вдосконалення у всьому, що вони роблять. Це може включати роздуми над минулими проєктами, виявлення слабких місць та інноваційні методи, які можна спробувати перевірити в майбутньому.

Існує багато компонентів систем управління проєктами, які можуть полегшити керування робочим простором, зокрема:

- інструменти планування та керування завданнями: є багато способів керувати робочим процесом команди, зокрема, за допомогою візуальних інструментів, таких як діаграми Ганта та часові шкали, можна отримати чітке уявлення про те, які завдання та проєкти завершилися, які ще в роботі і загалом, що ще лишилося зробити. Інструменти планування та організації завдань мають бути простими у використанні, оскільки вони є корінням управління проєктами. Вони дозволяють призначати завдання команді та розуміти, що кожен повинен зробити, щоб досягти цілей команди. Реалізація даної функціональності також можлива у формі спільного календаря;

- інструменти командної співпраці: інструменти для спільної роботи в команді особливо корисні, якщо є велика кількість позаштатних або віддалених працівників, оскільки вони дають цифровий простір, де керівник та команда може обмінюватися ідеями та файлами та обговорювати завдання. Інструменти для співпраці можуть включати дошки обговорень, системи обміну файлами та/або функції обміну повідомленнями;

- інтеграції: об’єднання з іншими програмами може спростити робочий процес, тому що це зробить непотрібним стрибати між кількома додатками. Коли все в одному місці, це дає змогу бути організованим та на зв’язку, а також економить час;

- звіти в режимі реального часу: керівник та команда повинні бути в курсі завдань. Завдяки звітності в режимі реального часу можна отримувати миттєві сповіщення про хід проєкту, пропущені терміни або будь-які зміни в поточних проєктах, а також вирішувати проблеми, перш ніж вони переростуть у більш серйозні;

- функції виставлення рахунків: складання бюджету іноді є частиною складання плану проєкту, що набагато простіше, коли функції виставлення рахунків уже вбудовані у програму. Можна використовувати ці інструменти, щоб спілкуватися з клієнтами про ціни та обчислювати додаткові витрати та оплачувані години. Функції виставлення рахунків також можуть надавати доступ до фінансів і звітів про бюджет команди;

- відстеження часу: інструмент контролю часу може швидко надати інформацію, скільки часу команда витрачає на певні завдання та проєкти, що корисно для планування. Відстеження часу є ключовою функцією, яка може допомогти визначити, які члени команди потребують додаткового керівництва з управління часом. Ці інструменти зазвичай мають форму етапних функцій, які висвітлюють певні рівні прогресу проєкту та допомагають точно розрахувати оплачувані години;

- система аналізу даних: система аналізу даних надає інструменти для вивчення робочої етики та процесу команди, щоб можна було побачити, де внести відповідні зміни. Якісна програма може показати прогрес членів команди, а також їхні недоліки, щоб керівники могли збирати інформацію та вносити покращення.

1.4. Порівняння програмних комплексів проєктного управління

Протягом багатьох років Trello (Рис. 1.1) був галузевим стандартом, коли справа доходила до програмного забезпечення для управління проєктами. Він був законодавцем мод з моменту його першого запуску в 2011 році, популяризуючи

підхід Kanban до управління проєктами і став джерелом натхнення для багатьох інструментів, які вийшли на ринок у наступні роки.

Рис. 1.1. Інтерфейс Kanban дошки у Trello

З іншого боку, попри те, що Notion (Рис 1.2) тільки нещодавно у сфері програмного забезпечення для проєктного менеджменту, він вже набрав мільйони користувачів за кілька коротких років.

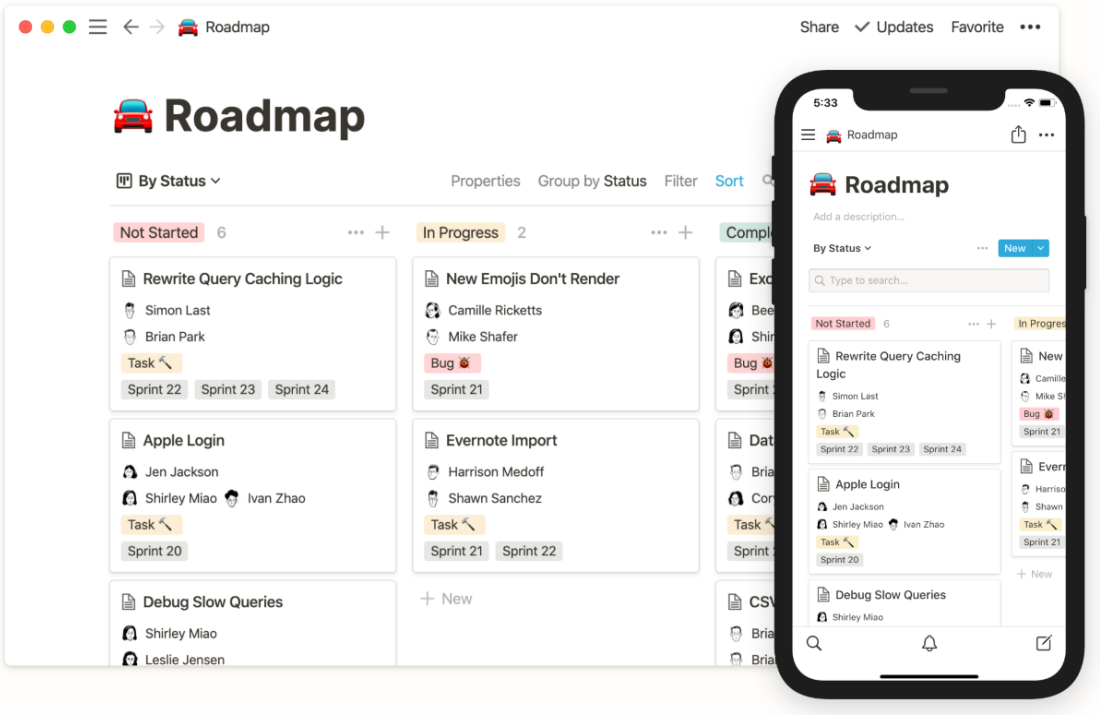

Рис. 1.2. Інтерфейс Kanban дошки у Notion

Notion і Trello мають багато подібностей. Обидва інструменти дозволяють легко керувати своїми завданнями та відстежувати свій прогрес на дошці Kanban. Однак, незважаючи на високу схожість, існує кілька принципових відмінностей між ними.

Головна привабливість Trello полягає в його простоті. Trello — це спеціальний інструмент керування проєктами, який повністю оптимізовано для виконання одного завдання.

На додаток до своєї культової дошки Kanban, Trello нещодавно увімкнув нові способи візуальної організації ваших завдань, зокрема інформаційну панель, шкалу часу, таблицю, календар (Рис. 1.3) і карту. Незважаючи на те, що Trello все ще активно розвивається, він мало змінився з роками. Для деяких його інтерфейс може здатися дещо застарілим у порівнянні з Notion.

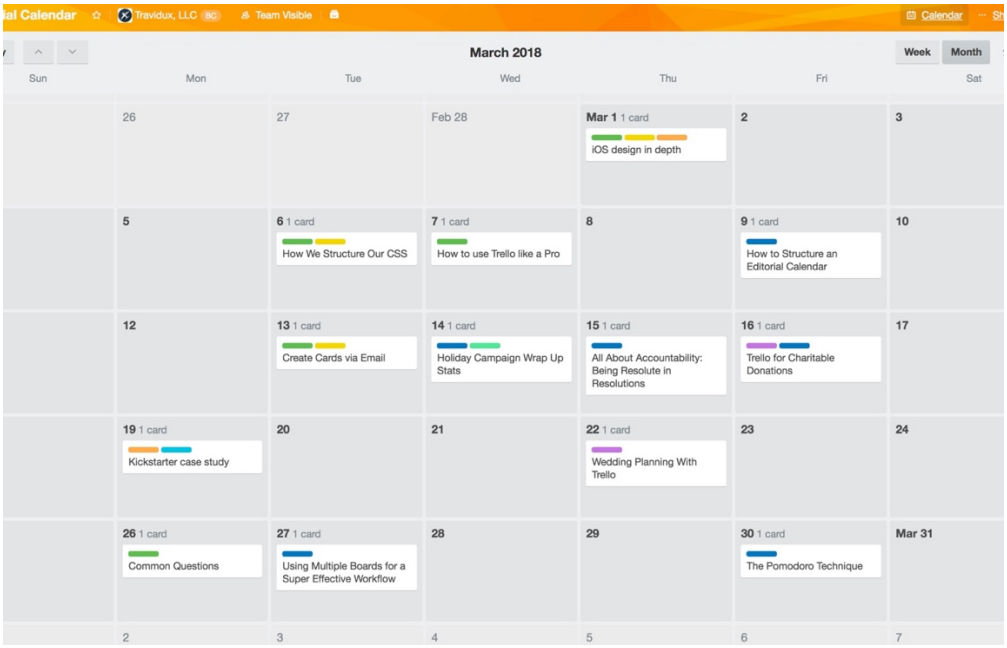

Рис. 1.3. Календар у Trello

Своєю популярністю Notion багато в чому завдячує своїй незрівнянній гнучкості. Як і Trello, у Notion є дошка Kanban, але це лише один із багатьох способів візуально організувати свій вміст у Notion (Рис 1.4). Це пропонує майже безмежні можливості — користувач може створити вікі, задокументувати дорожню карту свого продукту, організувати нотатки зустрічей, публікувати свій блог — і багато іншого.

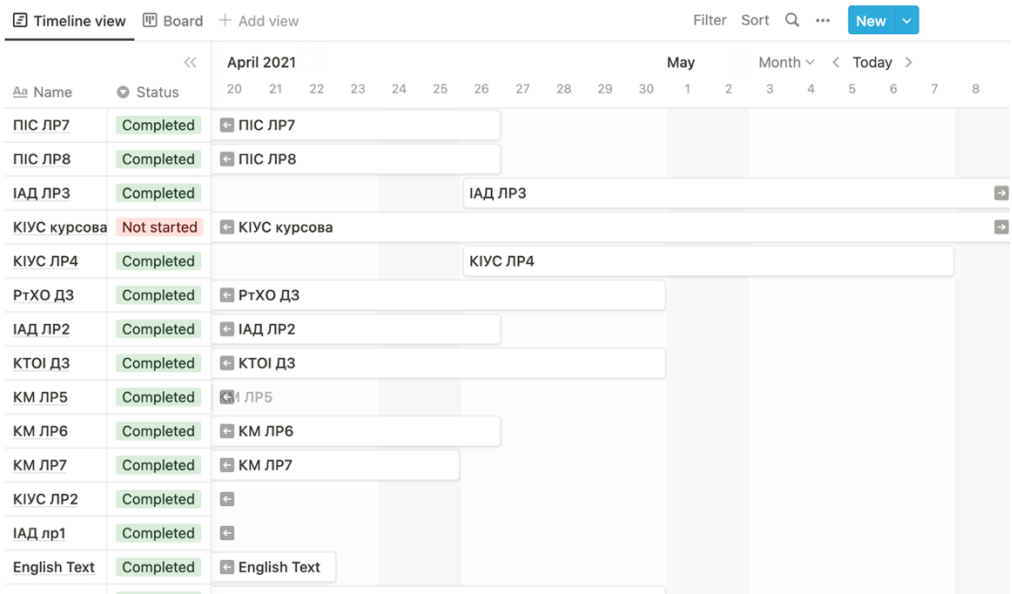

Рис. 1.4. Часова шкала у Notion

Одним з недоліків такого потужного набору функцій є стрімка крива навчання. Нові користувачі можуть відчувати себе перевантаженими всіма меню та параметрами, і вони повинні бути готові приділити достатньо часу для вивчення інструмента.

Використання обох цих систем вимагає дотримання Kanban-системи управління проєктами. Проте, на відміну від Trello, у Notion дошки, як і картки можна безкінечно налаштовувати, розрізняючи завдання за мітками та термінами. Також обидва ці додатки мають інструменти командної співпраці та певні інтеграції з іншими сервісами.

1.5. Постановка задачі

Згідно з дослідженнями, 77% компаній використовують інструменти управління проєктами, а 87% найбільш ефективних компаній використовують програмне забезпечення для управління проєктами. Вони посилюють робочий процес, відстежуючи кожне завдання та зберігаючи часову шкалу. І також виконують такі функції менеджерів проєктів, як: ведення зустрічей, планування та розклад, управління ресурсами та бюджетом, а також аналіз складання звітів за різними показниками. Отже, управління проєктами є актуальним.

Метою дипломного проєкту є розробка прототипу програмної системи для управління IT проєктами, який дозволить оптимальним чином розподіляти задачі між виконавцями, перерозподіляти навантаження та контролювати загальний та власний прогрес.

Відповідно до поставленої мети роботи необхідно вирішити такі завдання:

- провести аналіз предметної області, дослідити методології та принципи управління проєктами, способи ефективного планування;
- визначити методи та програмні засоби для розробки програмної системи;
- спроєктувати базу даних користувачів з можливістю їх авторизації та додавання нових;

- розробити навігаційну систему застосунку для створення проєктів та , груп виконавців;

- розробити візуалізовану систему роботи з завданнями (Kanban) та щоденний план;

- розробити систему захисту даних користувачів;

- реалізувати можливість формування аналітичних даних та перегляд метрик.

В результаті проведеної роботи буде отримано систему, яку можна буде повноцінно використовувати в якості програми для планування власного часу чи часу проєктної команди.

1.6. Висновки до розділу 1

Успіх проєкту – це багатовимірна конструкція, яка може означати різні речі для різних людей. Найкраще це виразити на початку проєкту в термінах ключових і вимірюваних критеріїв, за якими можна судити про відносний успіх або невдачу проєкту. Наприклад: досягнення ключових цілей проєкту, таких як бізнес-цілі організації-спонсора, власника або користувача, викликати задоволеність процесом управління проєктом, тобто результат завершений, відповідає стандартам, вчасно та в межах бюджету, та відображення загального визнання та задоволеність результатами проєкту з боку замовника проєкту та більшості спільноти проєкту в якийсь момент у майбутньому.

Саме менеджер відповідає за досягнення всіх цілей проєкту. Ці цілі завжди слід визначати за допомогою парадигми SMART (конкретні, вимірювані, амбітні, реалістичні, обмежені часом). Маючи туманні цілі, керівник може зіткнутися з щоденними проблемами, щоб все було організовано.

За допомогою різних парадигм структурування, впровадження та підтримки проєктів, які керівник формує у відповідному плані чи спринті, розробник може легко створювати новий продукт, гнучко вносити потрібні зміни від клієнта чи замовника,

використовувати інтегроване тестування та тестування після розгортання та здійснювати подальшу підтримку і інтеграцію нових сервісів.

Кожен інструмент управління проєктами повинен зберігати всі дані кожного проєкту в одному місці, щоб їх можна було порівнювати та аналізувати. На додаток до основних функцій планування, таких як часові шкали діаграми Ганта, може знадобиться хороша система відстеження завдань, яка допоможе призначати завдання та терміни в межах кожного плану проєкту.

За допомогою правильного інструменту керівник може відстежувати поточну ефективність щодо ключових цілей, що робить успіх проєкту набагато більш імовірним, гарантуючи, що нічого не прослизає через мережу непоміченим.

Одними з додатків, які надають таку функціональність можуть бути Trello та Notion. Загалом, Trello не настільки багатофункціональний, як Notion, але він компенсує це більш зручним для користувача. Це перетворює простоту в одну з ключових переваг, що дозволяє будь-кому якомога легше розпочати роботу з інструментом і працювати з ним продуктивно.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Архітектура програмного забезпечення

Архітектуру програмного забезпечення можна визначити як фундаментальну організацію системи, втілену в її компонентах, їх взаємовідносини один з одним і навколишнім середовищем, а також принципи, що керують її проєктуванням і еволюцією. Недоліки будь-якого програмного забезпечення мають значний вплив на бізнес організації. Основною причиною будь-якого збою програмного забезпечення може бути вибір неправильних шаблонів архітектури програмного забезпечення. Отже, архітектура програмного забезпечення – це неоднозначний термін, який стосується не лише дисципліни архітектури програмного забезпечення, а й структури та зв’язків між компонентами.

На сьогоднішній день існує величезна кількість різноманітних програм та їх типів, таких як WEB, Desktop, Mobile і так далі. Кожен тип і конкретна програма вимагають певної архітектури, тому неможливо розділити ту чи іншу реалізацію на хороше чи погане архітектурне рішення, але в усіх випадках можна визнати переваги використання того чи іншого підходу до розробки програмного забезпечення. Такі підходи також називаються архітектурними паттернами або шаблонами, які покликані вирішувати ряд наступних завдань:

- визначення основних характеристик програми – допомагають визначити основні характеристики та поведінку програми, наприклад, деякі шаблони архітектури можна використовувати для високо масштабованих додатків, тоді як інші можна використовувати для гнучких програм.
- зниження витрат – грамотно підібрана архітектура дозволяє розробникам

повторно використовувати існуючі компоненти, краще аналізувати додаток і швидко вносити зміни, тому час на розробку та підтримку скорочується;

- підвищення якості та зниження ризику – це усі види діяльності пов'язані з ризиками. Під час розробки можуть допускатися помилки у вихідному коді, що призводить до некоректної роботи всього застосунку.

- вища продуктивність – визначається продуктивність як людини, яка створює застосунок, так і результуючого додатку. Зазвичай програми працюють швидше і вимагають менше ресурсів, якщо вони розроблені правильно.

- підвищення ефективності управління – архітектурне рішення може мати на меті підвищення ефективності управління, наприклад, шляхом автоматизації документообігу на підприємстві (перехід з паперових документів на електронні з відстеженням історії змін, повідомленнями тощо).

- вища адаптивність – нові функції, такі як інший інтерфейс або додавання правила процесу, легше досягти, оскільки архітектура програмного забезпечення створює чітке розмежування проблем.

Архітектурні шаблони подібні до шаблонів проєктування програмного забезпечення, але мають ширшу область застосування.

В основі застосунку «Програмна система управління ІТ-проєктами» було покладено використання одного з основних архітектурних шаблонів, а саме – шаблон клієнт-серверної архітектури. Він описується як розподілена структура програми, що має два основних компоненти – клієнт і сервер.

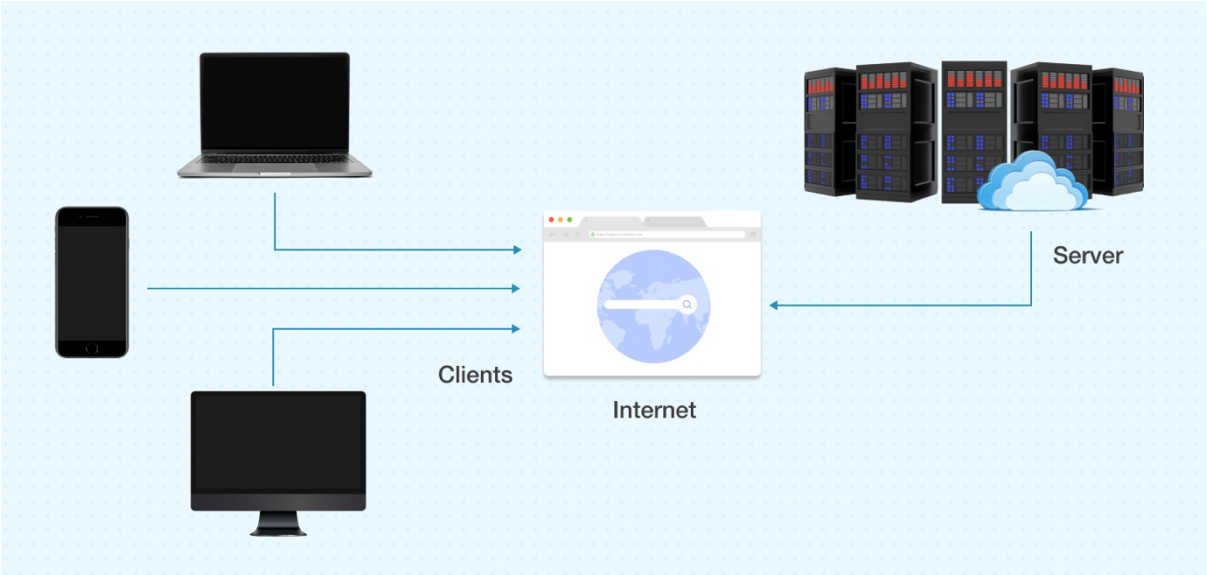

Рис. 2.1. Схема клієнт-серверної архітектури

Ця архітектура полегшує зв'язок між клієнтом і сервером, які можуть бути або не бути в одній мережі. Клієнт запитує певні ресурси для отримання із сервера, які можуть бути у формі даних, вмісту, послуг, файлів тощо (Рис. 2.1). Сервер ідентифікує зроблені запити та відповідає клієнту належним чином, надсилаючи запитувані ресурси.

Функціональні характеристики клієнта і сервера є прикладом програм, які взаємодіють одна з одною в рамках програми. Функціональність цієї архітектури дуже гнучка, оскільки один сервер може обслуговувати кілька клієнтів, або один клієнт може використовувати кілька серверів. Сервери можна класифікувати за послугами або ресурсами, які вони надають, незалежно від того, як вони працюють.

Вибір такого підходу був обумовлений гнучкістю вихідного продукту: реалізація хорошої структури серверної частини зі зручним програмним інтерфейсом застосунку (часто відкритим назагал) дає змогу іншим розробникам, або командам будувати застосунок на різних системах використовуючи одне джерело даних. Реалізовуючи мобільну версію, версію для настільних комп’ютерів (Windows, Linux, Mac та інші) або WEB-версію можна створювати запити на єдиний сервер, який обробить запит і відправить структуровану відповідь певного формату, яку в тому чи

іншому вигляді зможе відобразити розробник відповідної системи. Також такий підхід дає можливість, у випадку відсутності потреби відображення чисельної інформації, накопичення різних модулів і також при обмеженості ресурсів для застосунку, розробник може використати лише необхідні йому дані зі спектру тих, які сервер може насправді надати, тим самим зменшивши навантаження на систему і підвищивши комфорт використання.

В той же час дану архітектуру було поєднано разом зі шаблоном мікросервісів (Рис. 2.2) та шаблоном модель-представлення-контролер (MVC) (Рис. 2.3) при реалізації серверу і клієнту.

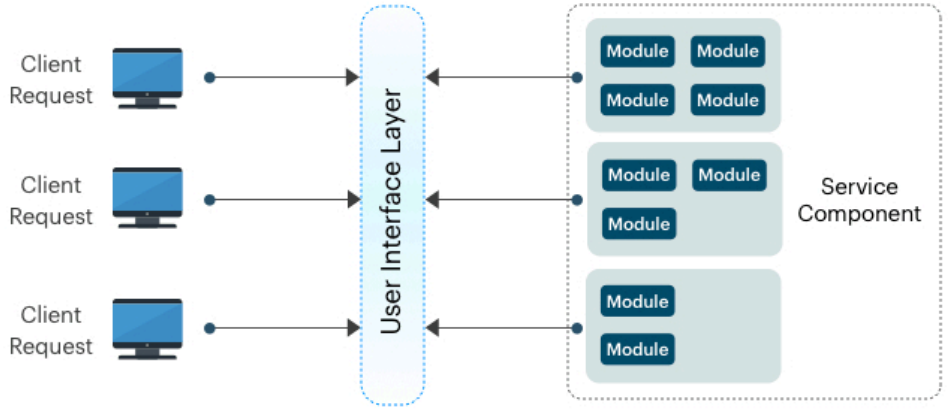

Рис. 2.2. Схема архітектури мікросервісів

Перший розглядається як життєздатна альтернатива монолітним додаткам і сервіс-орієнтованим архітектурам. Компоненти такої системи розгорнуті як окремі блоки за допомогою ефективного, спрощеного способу доставки. Переваги шаблону полягають у підвищеній масштабованості та високому ступеню роз’єднання в програмі.

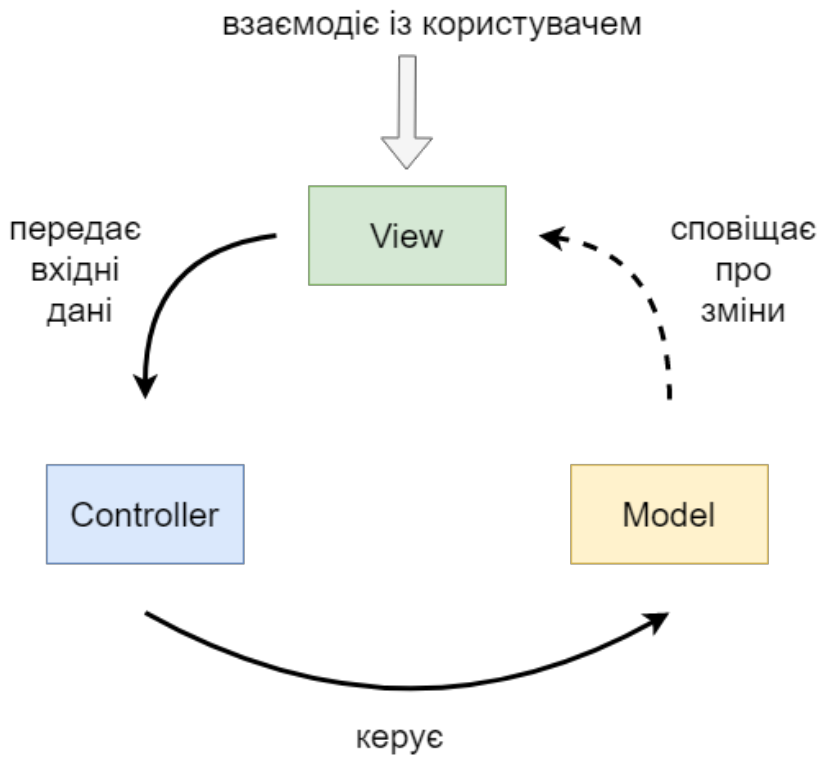

Рис. 2.3. Схема MVC архітектури

Структура Model-View-Controller (MVC), яка є стандартним підходом до розробки програмного забезпечення, що пропонується більшістю популярних фреймворків, явно є багатошаровою архітектурою. Прямо над базою даних розташований рівень моделі, який часто містить бізнес-логіку та інформацію про типи даних у базі даних. Угорі ж знаходиться шар перегляду. Посередині – контролер, який має різні правила та методи для перетворення даних, що переміщуються між представленням і моделлю.

Перевагою багатошарової архітектури є розділення проблем, що означає, що кожен рівень може зосередитися виключно на своїй ролі. Ця архітектура також може містити додаткові відкриті шари, як-от рівень сервісу, який можна використовувати для доступу до спільних служб лише на бізнес-рівні, але також можна обійти для швидкості.

2.2. Мова програмування C++ та Qt framework

При виборі мови для реалізації стійкої і швидкодіючої системи керування ІТ проєктами необхідними аспектами були наявність класичної моделі ООП для можливості реалізації «чистої» архітектури, підтримки кросплатформ розробки, відкритий доступ до керування ресурсами системи, швидкодія та простота розгортання. Саме мова програмування C++ найкраще для цього підійшла, адже вона містить всі вище перераховані особливості та до сьогодні підтримується розробка, покращення та нововведення притаманні іншим сучасним мовам (в той же час похідним), які полегшують і пришвидшують розробку.

Серед основних аспектів мови:

- концепції ООП (абстракція даних, інкапсуляція даних, спадкування, приховування даних та поліморфізм);
- машинна незалежність (або переносимість) відноситься до можливості передавати інструкції з одного операційного середовища в інше (Рис. 2.4). C++ використовує принцип WORA (Write Once, Run Anywhere). Наприклад, якщо ви пишете програму в Microsoft Windows і з якоїсь причини вам доведеться перейти на LINUX, ваш оригінальний програмний код працюватиме таким же чином. Однак C++ не залежить від платформи. Це означає, що компілятор згенерує залежний від ОС файл виконання, який не можна запускати в різних операційних системах;

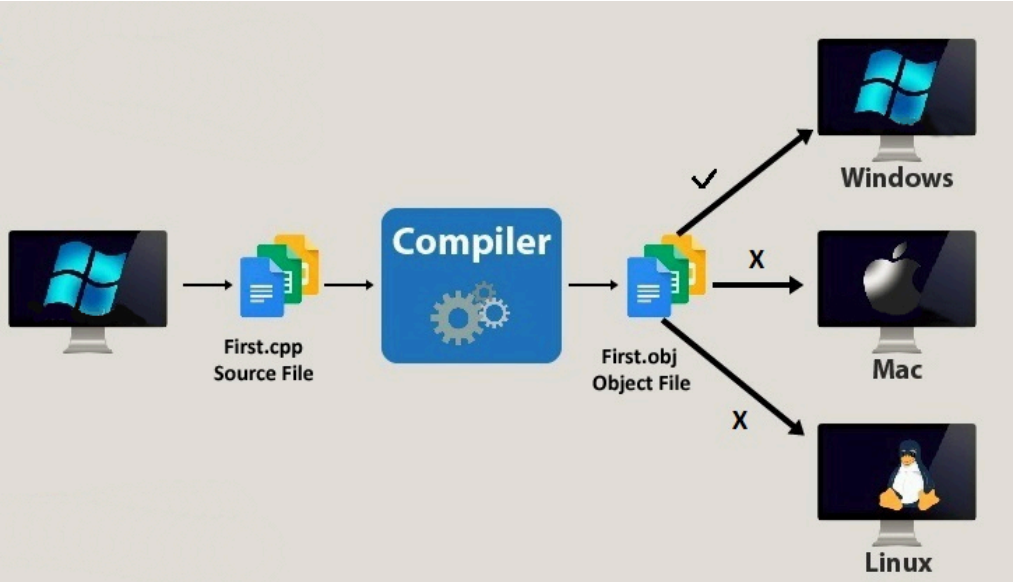

Рис. 2.4. Переносимість C++ програм

- дотримується портативності (Рис. 2.5) – це концепція перенесення інструкцій з однієї системи в іншу. Тобто застосунок можна запустити на іншому ПК при перенесенні скомпільованого файлу між системами одного типу;

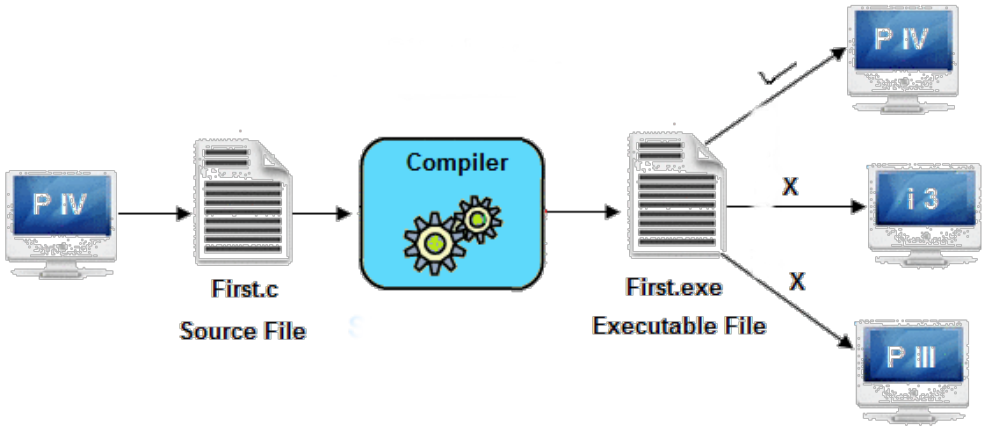

Рис. 2.5. Портативність C++ програм

- код написаний мовою C++ компілюється, отже, він набагато швидший, ніж інші мови програмування, такі як Python і Java, які інтерпретуються;

- пам'ять у C++ можна розділити на дві частини - стек і кучу. Стек відноситься до пам’яті, яка виділяється для змінних, оголошених всередині функції. А, купа відноситься до невикористаної пам’яті, яка може бути виділена динамічно. Наприклад, якщо ви не знаєте про потребу в пам’яті для зберігання інформації у визначеній змінній, розмір пам’яті можна визначити вручну під час виконання. Це досягається за допомогою спеціальних алокаторів або доступу до віртуальної пам’яті. В той же час пам’ять виділена одного разу може бути перевикористана в іншому місці для економії системних ресурсів;

- містить багато інструментів, такі як засоби перевірки стилів і умов кодування, оптимізатори коду, візуалізатори програм і інкрементні компілятори, які покладаються на постійно оновлювану базу даних, яка містить семантичну інформацію, витягнуту з вихідних програм. Крім того, нові методи програмування викликають потребу в розширенні. C++ має потенціал для легкого прийняття та інтеграції нових функцій і отримання знань.

В поєднанні зі вбудованими бібліотеками C++ було використано один з провідних фреймворків для створення користувацьких застосунків Qt framework.

Qt – це кросплатформний фреймворк для розробки додатків для настільних, вбудованих і мобільних пристроїв. Підтримувані платформи включають Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS та інші.

Qt – це фреймворк, написаний на мові C++. Препроцесор, MOC (Meta-Object Compiler), використовується для розширення мови C++ за допомогою таких функцій, як сигнали та слоти. Перед кроком компіляції MOC аналізує вихідні файли, написані на Qt-розширеному C++, і генерує з них стандартні сумісні C++ джерела. Таким чином, сам фреймворк і програми/бібліотеки, які його використовують, можуть бути скомпільовані будь-яким стандартним компілятором C++, таким як Clang, GCC, ICC, MinGW і MSVC.

Qt побудований на таких ключових концепціях:

- повна абстракція графічного інтерфейсу – останні версії фреймворку використовують власний стиль API для різних платформ;

- сигнали та слоти – мовна конструкція, введена в Qt для зв’язку між об’єктами, яка дозволяє легко реалізувати шаблон спостерігача, уникаючи шаблонного коду. Концепція полягає в тому, що віджети GUI можуть надсилати сигнали, що містять інформацію про події, які можуть бути отримані іншими елементами керування за допомогою спеціальних функцій, відомих як слоти;
- компілятор метаоб'єктів (moc) – є інструментом, який запускається на джерелах програми Qt. Він інтерпретує певні макроси з коду C++ як анотації та використовує їх для створення додаткового коду C++ з метаінформацією про класи, які використовуються в програмі;
- мовні прив’язки – Qt можна використовувати в кількох мовах програмування, окрім C++, таких як Python, Javascript, C# та Rust через мовні прив’язки. Саме тісна інтеграція з Javascript, дає змогу будувати власні віджети з гарним інтерфейсом.

Особливою частиною фреймворку є наявність великої кількості модулів високого рівня для таких функцій, як вбудовані віджети криптографія, нетворкінг, для роботи з базами даних та багато інших. Всі вони розділені на окремі групи і незалежні між собою, що дає можливість дестриб’ютити застосунок лише з необхідними частинами фреймворку, при цьому не збільшуючи об’єм пам’яті, який займає програма.

2.3. Спілкування клієнта та сервера. Шифрування

Можливість передачі даних між сервером та клієнтом була реалізована через сокети – програмний інтерфейс для обміну даних між процесами, а саме TCP-сокети – він орієнтований на підключення, який використовує протокол керування передачею (TCP). Для встановлення з’єднання потрібні три пакети: пакет SYN, пакет SYN-ACK і пакет ACK. Такий тип сокету визначається IP-адресою машини та портом, який він використовує та гарантує, що всі дані будуть отримані та підтверджені. Саме через переваги такого типу з’єднання, а саме:

- є надійним: пакети, втрачені в мережі, виявляються і повторно передаються відправником;

- має впорядковану доставку даних: дані зчитуються нашим додатком у тому порядку, в якому вони були записані відправником.

Адже необхідність отримання точної інформації стоїть вище за швидкість, було взято в основу спілкування через TCP-сокети.

В той же час на сервері знаходиться окремий сокет, через який з'єднання спершу реєструється – ssl сокет. Його мета – побудувати шифроване безпечне з'єднання для передачі даних по ньому, яке також називають «TLS-рукостискання».

Перш за все потрібно згенерувати ключі та сертифікати SSL для сервера і клієнта, далі на кожній зі сторін вони зчитуються у внутрішні змінні (Рис. 2.6).

```
30  void Crypto::Initialize(const QString &CryptoPath) {
31      static bool bEvaluated = false;
32      if (bEvaluated)
33      {
34          qDebug() << "SSL key and certificate has already been initialized!";
35          return;
36      }
37      QByteArray key = Helper::LoadFileData(CryptoPath, SSLPrivateKey, InExtension: "key");
38      QByteArray cert = Helper::LoadFileData(CryptoPath, SSLCertificate, InExtension: "pem");
39      QByteArray cacert = Helper::LoadFileData(CryptoPath, SSLCACertificate, InExtension: "pem");
40
41      SSLKey = QSslKey(key, algorithm: QSsl::Rsa, format: QSsl::Pem, type: QSsl::PrivateKey, passPhrase: "localhost");
42      SSLCert = QSslCertificate(cert);
43      SSLCACert = QSslCertificate(cacert);
44
45      qDebug() << "Crypto SSL key and certificate was initialized!";
46
47      bEvaluated = true;
48  }
```

Рис. 2.6. Ініціалізація ключів та сертифікатів SSL

Далі необхідним є для об'єкта SSL сокету (який є дочірнім класом TCP сокету) зареєструвати необхідні ключі і сертифікати. В кінці почати процес шифрування (Рис. 2.7).

```
    QList<QSslCertificate> listCA;
    listCA.append(Crypto::SSLCACert);
    QSslConfiguration conf;
    conf.setPrivateKey(Crypto::SSLKey);
    conf.setLocalCertificate(Crypto::SSLCert);
    conf.setCaCertificates(listCA);

    setSslConfiguration(conf);
    setPeerVerifyMode( mode: QSslSocket::VerifyPeer);

    // Report any SSL errors that occur
    connect( sender: this, SIGNAL( arg: sslErrors(const QList<QSslError> &)), receiver: this, SLOT( arg: OnErrors(const QList<QSslError> &)));
    connect( sender: this, SIGNAL( arg: error(QAbstractSocket::SocketError)), receiver: this, SLOT( arg: OnFatalError()));

    startServerEncryption();
```

Рис. 2.7. Конфігурування та запуск з’єднання шифрування

Після того, як сокет проробить необхідну роботу спілкування з клієнтом і з’єднання встановиться, сокет можна додати у список з’єднань (Рис. 2.8), який можна надалі використовувати в звичайній передачі даних між сервером і клієнтом.

```
void Server::OnHandshake() {
    SSLSocket *Socket = dynamic_cast<SSLSocket *>(sender());

    assert(Socket);

    connect(Socket, SIGNAL( arg: disconnected()), receiver: this, SLOT( arg: OnSocketClosed()));
    connect(Socket, SIGNAL( arg: readyRead()), receiver: this, SLOT( arg: OnMessageReceived()));

    Connections.push_back(Socket);

    qDebug() << "OnHandshake";

}
```

Рис. 2.8. Збереження сокету як з’єднання після успішного SSL шифрування

Після формування з’єднання клієнта та сервера, перший може почати надсилати запити для отримання необхідних даних. Запит і відповідь мають подібні формати: спочатку йде повідомлення з кількістю байт необхідних для того, щоб зчитати змістовні дані в наступному повідомленні по сокету. Дані подаються у форматі json – формат обміну даних, який використовує зрозумілий людині текст для передачі об’єктів даних, що складаються з пар атрибут-значення. Це звичайний формат із різноманітним використанням в електронному обміні даними, включаючи веб-

додатки із серверами. Саме тому стає можливим легке розгортання клієнтської системи на будь-якій платформі.

Застосунок також використовує систему токенів для спілкування з іншими системами, наприклад, Google API або Github API. Через вбудовані інтерфейси цих систем, за рахунок надання клієнтом доступу до його акаунту в них, можна здійснювати додаткове отримання даних: контакти чи календар Google, прив’язку до проєктів в системі Github та інші. Також користувач може здійснити авторизацію в систему через інтерфейс OAuth2.0 використовуючи акаунт в вище згаданих системах.

2.4. База даних

Кожна база даних повинна відповідати трьом звичайним формам. Серверна частина містить окремий модуль бази даних, модель якої відповідає наступним формам:

- усі атрибути прості, усі використовувані домени повинні містять лише скалярні значення, у таблиці немає повторюваних рядків;
- кожен неключовий атрибут незмінно залежить від первинного ключа;
- кожен неключовий атрибут транзитивно не залежить від первинного ключа.

Наступні таблиці (Рис. 2.9) представляють бізнес-модель на стороні бази даних. Вони також відповідають всім 3 правилам нормалізації.

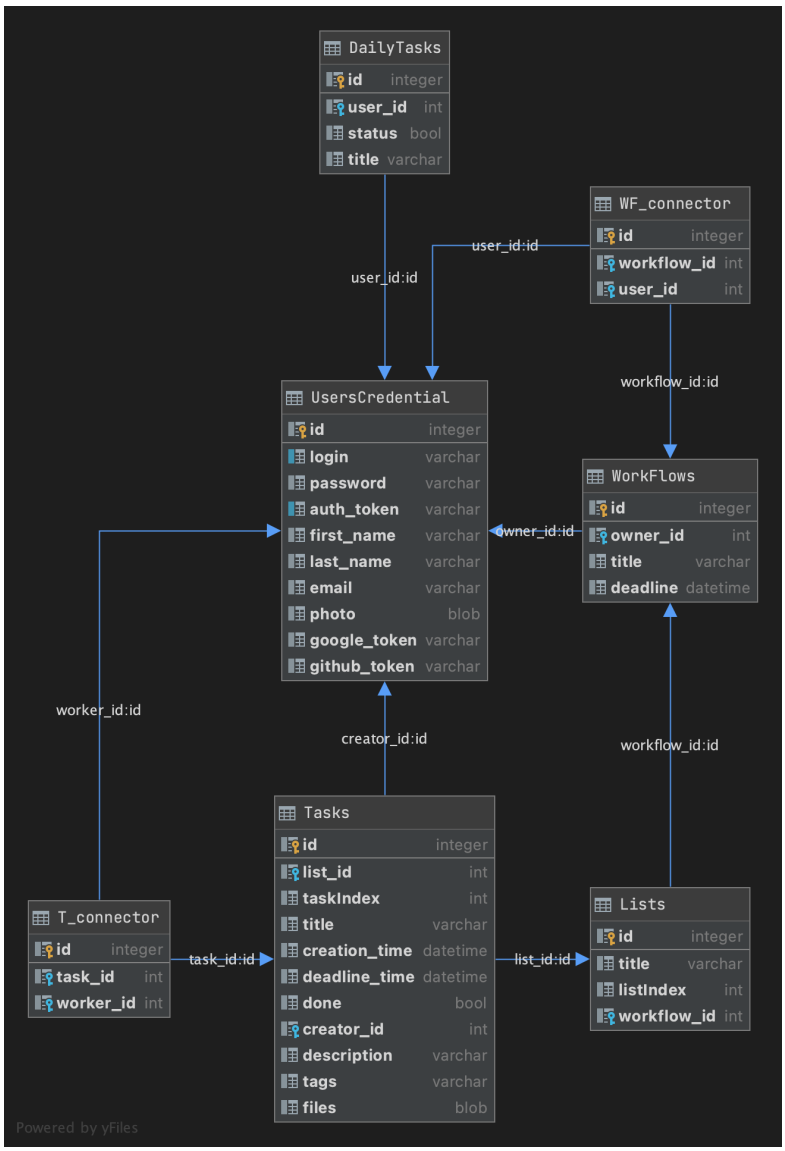

Рис. 2.9. Модель бази даних

У «Програмній системі керування IT проєктами» вибір бази даних впав на реляційну, а саме SQlite базу даних. В Qt є реалізований інтерфейс для створення та керування такою базою даних.

2.5. Шаблони проєктування

У програмній інженерії шаблон проектування є загальним повторюваним рішенням поширеної проблеми в розробці програмного забезпечення. Шаблон дизайну — це не готовий дизайн, який можна перетворити безпосередньо в код. Це певний опис для вирішення проблеми, який можна використовувати в багатьох різних ситуаціях.

Всі шаблони можна поділити на 3 групи: породжувальні, структурні та поведінкові

2.5.1. Одинак

Це шаблон проектування програмного забезпечення, який обмежує створення класу одним «одиничним» екземпляром (Рис. 2.10). Паттерн належить до групи породжувальних. Це корисно, коли для координації дій у системі потрібен саме один об’єкт.

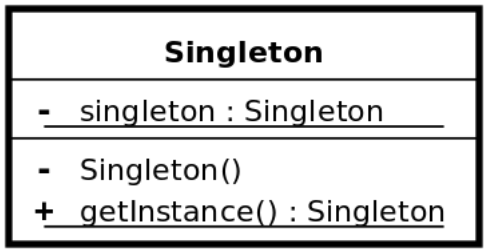

Рис. 2.10. Схема реалізації шаблону «Одинак»

2.5.2. Міст

Міст — це структурний шаблон проектування, який дозволяє відокремити реалізацію від її абстракції (Рис. 2.11), тому реалізацію можна змінити окремо від абстракції, оскільки вона не успадковується від неї безпосередньо.

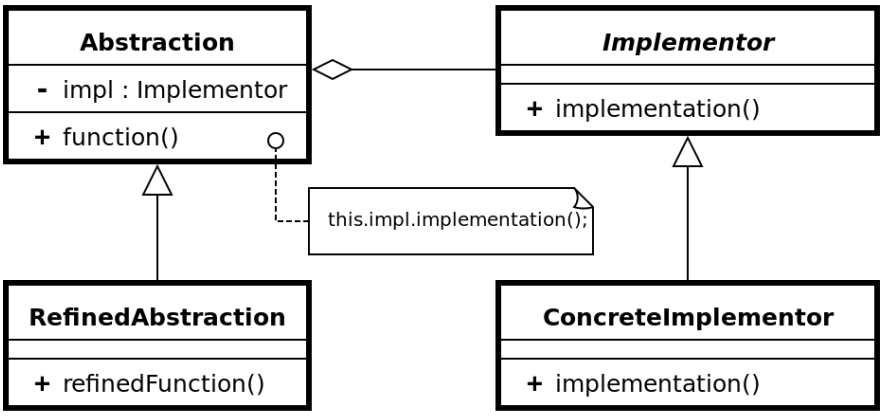

Рис. 2.11. Схема реалізації шаблону «Міст»

Міст використовує інкапсуляцію, агрегацію і може використовувати успадкування для розділення обов'язків на різні класи.

Коли клас часто змінюється, функції об'єктно-орієнтованого програмування стають дуже корисними, оскільки зміни в коді програми можна легко внести з мінімальними попередніми знаннями про програму. Шаблон мосту корисний, коли і клас, і те, що він робить, часто змінюються. Сам клас можна розглядати як абстракцію, а те, що клас може робити як реалізацію. Візерунок-міст також можна розглядати як два шари абстракції.

2.5.3. Шаблонний метод

Метод шаблону — це метод суперкласу, зазвичай абстрактного суперкласу, і визначає скелет операції в термінах ряду кроків високого рівня (Рис. 2.12). Ці кроки самі реалізуються додатковими допоміжними методами в тому ж класі, що й метод шаблону.

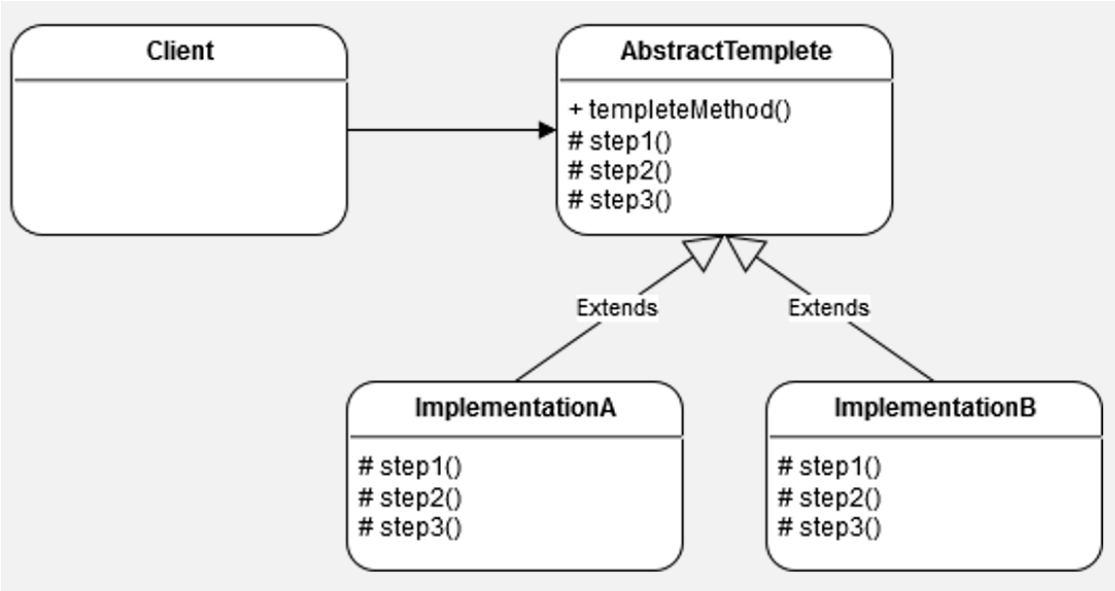

Рис. 2.12. Схема реалізації шаблонного методу

Допоміжними методами можуть бути або абстрактні методи, у цьому випадку підкласи потрібні для надання конкретних реалізацій, або методи гачка, які мають порожні тіла в суперкласі. Підкласи можуть (але не є обов'язковими) налаштовувати операцію, перевизначаючи методи гачка. Мета методу шаблону полягає в тому, щоб визначити загальну структуру операції, дозволяючи підкласам уточнювати або перевизначати певні кроки.

2.6. Висновки до розділу 2

В даному розділі було проведено огляд архітектурних рішень, адже це є однією з найважливіших частин процесу розробки. Успіх проєкту на половину залежить від правильно підібраної архітектури.

Програмний комплекс для керування IT проєктами побудований згідно клієнт-серверної архітектури, що надає можливість створювати мультиплатформенні застосунки, та поєднав її з архітектурою MVC для стільникової клієнтської частини свого проєкту. Весь проєкт побудований за допомогою фреймворку Qt з певним набором його вбудованих модулей, серед яких використав модуль для oAuth

авторизації, ssl шифрування та автентифікація, модуль для керування базою даних, модуль нетворкінгу. Я також приділяв більше уваги бізнес-логіці, оскільки вона є серцем будь-якої програми. Під час реалізації я використовував стандартні шаблони, такі як: одинак, міст, шаблонний метод.

Вибір мови C++ для реалізації клієнтської частини надав змогу зробити програму кросплатформенною: підтримуються Linux, MacOS та Windows.

Для мого проєкту була обрана реляційна база даних, оскільки всі дані, надані користувачами, будуть зберігатися узгоджено та коректно, що надзвичайно важливо в даному випадку.

Усі функції дозволили мені надати зрозуміле, розширюване та гарне масштабоване додаток. Така архітектура серйозно покращила продуктивність, тому користувачу не потрібні дуже високі параметри системи.

РОЗДІЛ 3
РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙС КОРИСТУВАЧА

3.1. Огляд функціональності системи

Серед основних властивостей програмного комплексу управління ІТ-проєктами важливе місце посіла гнучкість та простота використання зі сторони цільового користувача. Досягалися вони швидкою реєстрацією, без надмірних питань, і авторизацією через поштову скриньку, створенням оболонки для ведення проєктів у 2 кліки та визначення завдань у зручному для користувача форматі.

Серед основних вікон користувача можна виділити наступні:

- вікно авторизації;
- вікно реєстрації;
- головне вікно з активними проєктами;
- вікно з дошками та завданнями обраного проєкту;
- вікно створення і налаштування нового проєкту;
- вікно створення і налаштування нового завдання;
- вікно запрошення користувача для спільного керування проєктом;
- вікно з планом на день, яке має властивість щоденне автоочищення зі серверної сторони;
- вікно статистики по проєктам та виконаним завданням користувача.

3.2. Використання MVC в застосунку

Ідеологія інтерфейсу користувача Qt повністю побудована на архітектурі model-view-controller. У фреймворці вже наявні певні стандартні моделі та їх віджети

для відображення даних, а саме списку, таблиці, дерева. Але є і зручний інтерфейс для створення власних, де прикладом може стати модель, яка зберігає та надає дані про щоденний план користувача (Рис. 3.1).

```cpp
QVariant DailyPlanModel::data(const QModelIndex &index, int role) const
{
    if (!index.isValid())
        return QVariant();
    switch(role) {
        case IdRole:
            return m_data[index.row()].id;
        case DescriptionRole:
            return m_data[index.row()].title;
        case StatusRole:
            return m_data[index.row()].status;
    }
    return QVariant();
}

bool DailyPlanModel::setData(const QModelIndex &index, const QVariant &value, int role)
{
    if (data(index, role) != value) {
        emit dataChanged(index, index, roles: QVector<int>() << role);
        return true;
    }
    return false;
}

QHash<int, QByteArray> DailyPlanModel::roleNames() const {
    QHash<int, QByteArray> roles;
    roles[IdRole] = "dailyTaskId";
    roles[DescriptionRole] = "dailyTaskDescription";
    roles[StatusRole] = "dailyTaskStatus";
    return roles;
}
```

Рис. 3.1. Часткова реалізація моделі щоденного плану користувача

Важливим у побудові є створення «ролей» даних (функція roleNames), саме через них (через функцію data) можна отримувати конкретний перелік даних, а не всю модель. В даному випадку створювались лише 3 ролі: id завдання, його опис та статус виконання. Використовуючи setData, можна додавати нові дані до таблиці, або оновлювати наразі вже додані.

В той же час весь інтерфейс будується на унікальній вбудованій мові qml, яка має інтеграцію разом з JavaScript та C++. Її синтаксис дещо схожий на синтаксис

javascript, проте містить певні відмінності. В коді на Рис. 3.2 показано, як можна використати вище описану модель.

```
Repeater {
    id: repeater
    model: DailyPlanModel

    RowLayout {
        Layout.fillWidth: true

        CheckBox {
            id: statusBox
            checked: dailyTaskStatus
            onClicked: {
                client.changeDailyTask(dailyTaskId, dailyTaskDescription, statusBox.checked);
            }
        }
        TextField {
            font.strikeout: statusBox.checked
            Layout.fillWidth: true
            Layout.fillHeight: true
            placeholderText: "Enter description.."
            wrapMode: "WrapAnywhere"
            selectByMouse: true
            text: dailyTaskDescription
            onEditingFinished: client.changeDailyTask(dailyTaskId, text, statusBox.checked);
        }
        RoundButton {
            text: qsTr("×");
            font.pointSize: 16

            Material.background: "transparent"
            onClicked: client.removeDailyTask(dailyTaskId)
        }
    }
}
```

Рис. 3.2. Приклад реалізації UI (з використанням моделі)

Спілкування з іншими об’єктами в даному випадку здійснюється через систему подій. В даному випадку є події натискання на віджети CheckBox та RoundButton, а також подія закінчення вводу тексту опису завдання. Коли ці події спрацьовуються, то використовується клієнт для створення запиту на зміну даних до серверу, адже це дані, які потребують синхронізації разом зі сервером.

Реалізація такого поводження здійснюється через інтерфейс connect (Рис. 3.3), за допомогою якого зв’язуються події, які приходять від об’єкта разом з реакцією, яка

повинна бути здійснена на відповідну подію. Тут показано прив’язка на подію request, яка спрацьовує, коли користувач оновлює якісь дані, які потребують синхронізації з сервером, з реакцією send, яка дані сформує у json формат, та відправить до серверу.

```
64      connect( sender: this, &Client::request,  receiver: this, &Client::send);
```

Рис. 3.3. З’єднання події request зі делегатом send

3.3. Інтерфейс користувача

3.3.1. Авторизація

При першому запуску застосунку користувача вітає вікно авторизації (Рис. 3.4) з двома полями: логін та пароль. Обидва містять перевірку вводу певних правил.

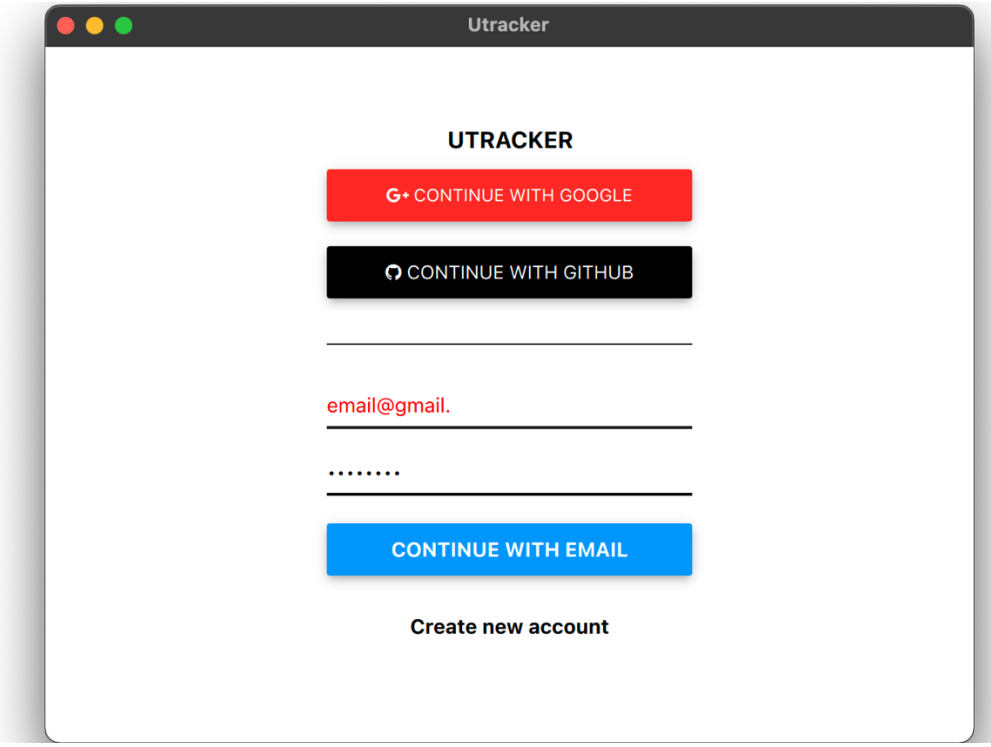

Рис. 3.4. Вікно авторизації

3.3.2. oAuth2.0 авторизація

Також є можливість авторизації через сторонні сервіси: Google або GitHub oAuth, а також можливість створити новий акаунт перейшовши на вікно реєстрації.

В даному випадку прикладом авторизації через Google oAuth може бути код зображений на Рис. 3.5, де спершу визначається Scope, який маємо намір використовувати в своєму застосунку: email, profile та calendar. За допомогою вбудованого інтерфейсу connect у Qt здійснюється прив'язка на подію authorizeWithBrowser, яка після початку авторизації виконає запит з переданим URL.

```
26      google = new QOAuth2AuthorizationCodeFlow( parent: this);
27      google->setScope( scope: "email profile https://www.googleapis.com/auth/calendar");
28
29      connect(this->google, &QOAuth2AuthorizationCodeFlow::authorizeWithBrowser,  slot: [=](QUrl url) {
30          QUrlQuery query(url);
31
32          query.addQueryItem( key: "prompt",  value: "consent");      // Param required to get data everytime
33          query.addQueryItem( key: "access_type",  value: "offline"); // Needed for Refresh Token (as AccessToken expires shortly)
34          url.setQuery(query);
35          QDesktopServices::openUrl(url);
36      });
```

Рис. 3.5. Визначення переліку Scope та створення запиту

Такий тип авторизації також потребує додаткових налаштувань (Рис. 3.6), а саме потрібно вказати приватні ключі, певні токени та посилання, які потрібно створити на сервісі розробника в Google API для власного застосунку.

```
45      google->setAuthorizationUrl(authUri);
46      google->setClientIdentifier(clientId);
47      google->setAccessTokenUrl(tokenUri);
48      google->setClientIdentifierSharedKey(clientSecret);
49      google->setModifyParametersFunction( modifyParametersFunction: [](QAbstractOAuth::Stage stage, QVariantMap* parameters) {
50          // Percent-decode the "code" parameter so Google can match it
51          if (stage == QAbstractOAuth::Stage::RequestingAccessToken) {
52              QByteArray code = parameters->value( akey: "code").toByteArray();
53              (*parameters)["code"] = QUrl::fromPercentEncoding(code);
54          }
55      });
```

Рис.3.6. Ініціалізація інтерфейса oAuth необхідними параметрами

Після налаштування всіх необхідних даних можна здійснювати запит до Google сервера, з метою аутентифікації користувача (Рис. 3.7), та отримання 2 токенів: токену авторизації та токену відновлення. Перший з них використовується для наступних запитів з метою отримання необхідних даних, наприклад, список контактів, доступ до подій календаря та інші, час життя такого токену – 24години, тому для його відновлення саме і потрібен інший з них токен.

```
57      QOAuthHttpServerReplyHandler* replyHandler = new QOAuthHttpServerReplyHandler(port, parent: this);
58      google->setReplyHandler(replyHandler);
59
60      connect(google, &QOAuth2AuthorizationCodeFlow::granted, slot: [=](){
61          const QString token = google->token();
62
63          emit gotToken("accesses_token", token);
64          emit gotToken("refresh_token", google->refreshToken());
65
66  ...
73      });
74      google->grant();
```

Рис. 3.7. Запит на з’єднання oAuth та передача отриманих токенів

Саме запит для відновлення токену авторизації може мати структуру зображену на Рис 3.8.

```
86  void GoogleAuth::refreshToken(const QString &token) {
87      QUrl url( url: QString( ch: "%1?alt=json&grant_type=refresh_token&refresh_token=%2&client_id=%3&client_secret=%4")
88                .arg( a: "https://www.googleapis.com/oauth2/v4/token/")
89                .arg(token)
90                .arg(CLIENT_ID)
91                .arg(CLIENT_SECRET));
92
93      auto reply = google->post(url);
94
95      connect(reply, &QNetworkReply::finished, slot: [=]() {
96          if (reply->error() != QNetworkReply::NoError) {
97              emit gotToken("accesses_token", google->token());
98              emit gotToken("refresh_token", google->refreshToken());
99          }
100         qDebug() << "REQUEST FINISHED. Error? " << (reply->error() != QNetworkReply::NoError);
101         qDebug() << reply->readAll();
102     });
103 }
```

Рис. 3.8. Запит на відновлення токену авторизації oAuth за допомогою токену відновлення

Після переходу на вікно реєстрації (Рис. 3.9) користувач має змогу вернутись назад на вікно авторизації. В даному вікні користувачу необхідно коректно вказати свої дані (поштову скриньку, на яку буде надіслане повідомлення підтвердження користувача, пароль для авторизації та ім’я з прізвищем, яке буде відображатись у дошках завдань проєкту.

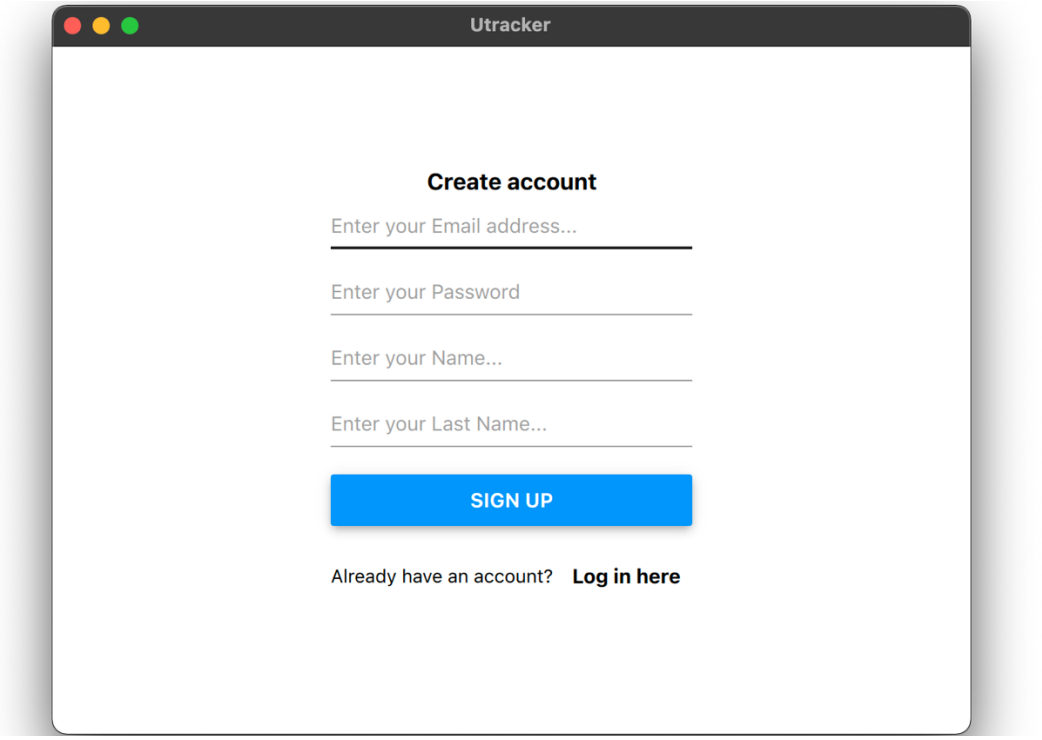

Рис. 3.9. Вікно створення нового аккаунту

У випадку якщо трапляється якась помилка: користувач ввів невірні дані, або користувача не було знайдено в базі даних, додаток відображає помилку внизу екрану (Рис. 3.10).

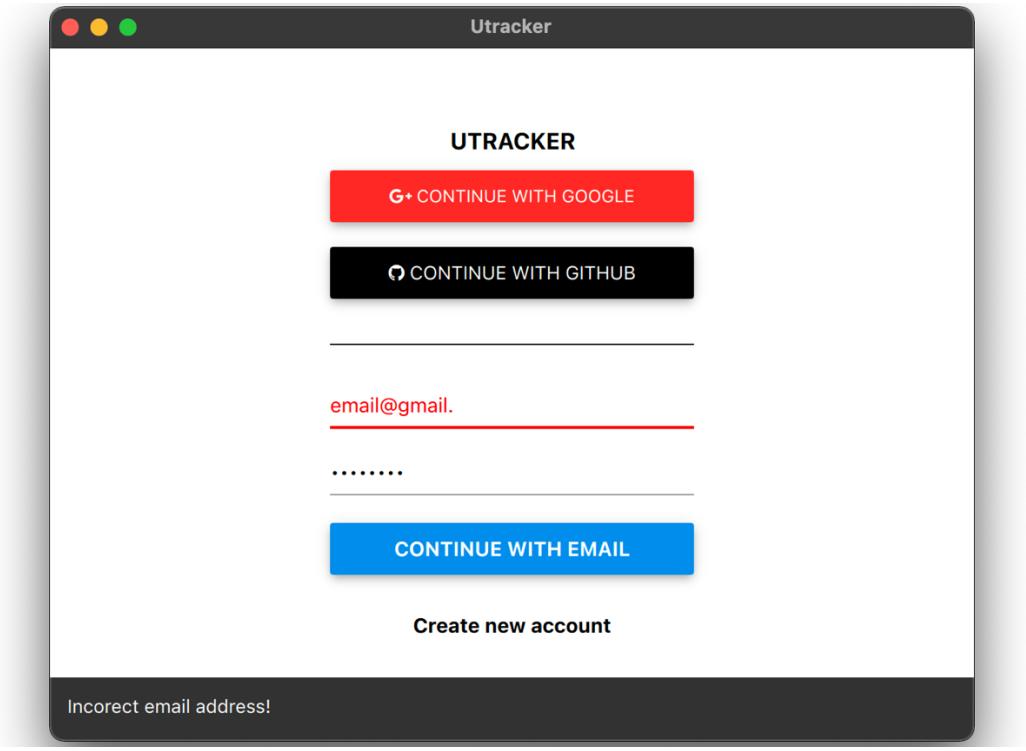

Рис. 3.10 Інформування користувача про невірні дані

Даний спосіб інформування користувача використовується у застосунку всюди, і також передає загальну інформацію, тим самим підвищуючи рівень використання користувачем.

3.3.3. Головне вікно

Після успішної авторизації користувач потрапляє на головне вікно застосунку (Рис. 3.11). На ньому представлені зліва – навігаційне меню по додатку, яке присутнє при користуванні додатком постійно, та справа – в'юпорт, в якому відображається основна частина вибраного розділу з навігаційного меню.

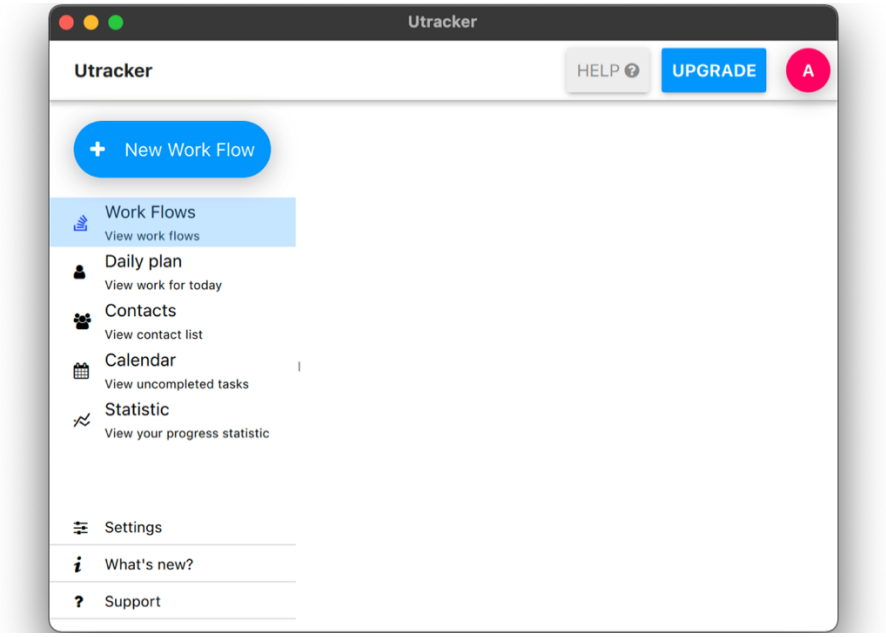

Рис. 3.11. Головне вікно застосунку

Саме тут користувач може почати керування новим проєктом натиснувши на кнопку створення нового проєкту (workflow):

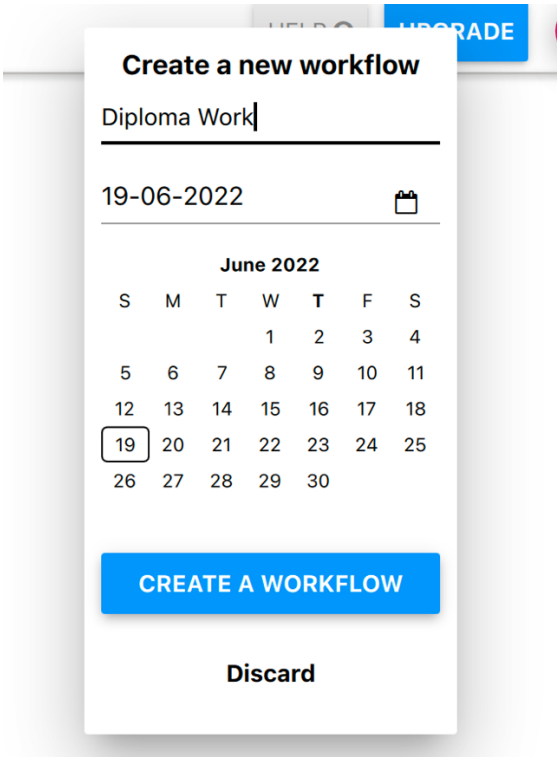

Рис. 3.12. Pop-up вікно створення та налаштування нового проєкту

Після чого користувач вносить дані про проєкт (Рис. 3.12), а саме його назву та дедлайн, які можна змінити після створення. Новостворені проєкти будуть відображатись в сітці у вигляді карток (Рис. 3.13).

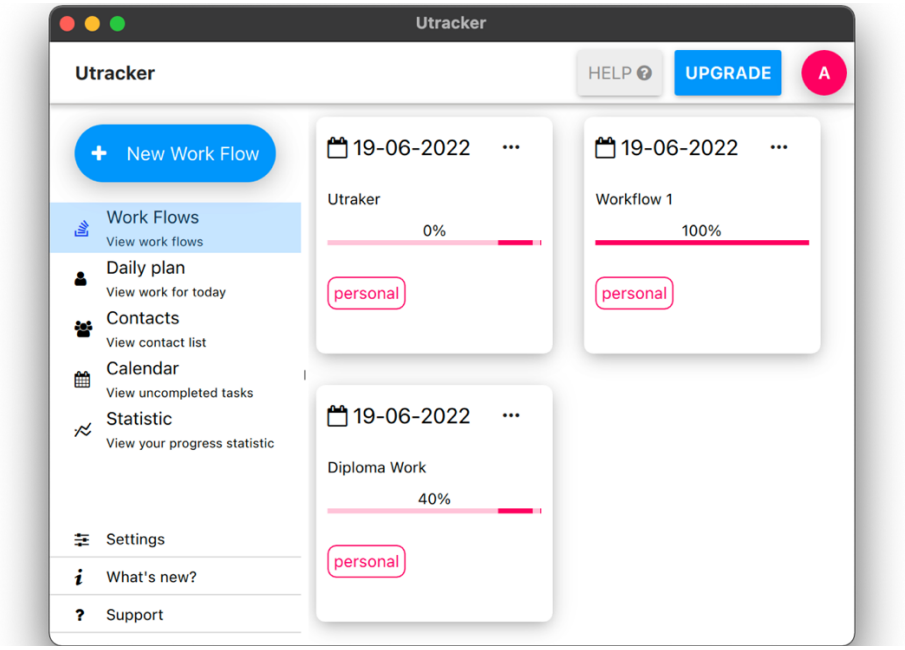

Рис. 3.13. Вигляд вікна зі проєктами в розробці

Викликавши контексне меню проєкту можна виконати певні маніпуляції з ним. Особливим серед них є запрошення користувача (Рис. 3.14) для керування проєктом у парі. У вікні просто потрібно внести поштову скриньку користувача, який є в системі, і з яким ви би хотіли вести проєкт, та натиснути кнопку запрошення.

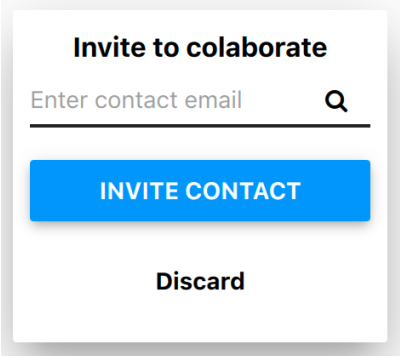

Рис. 3.14. Pop-up вікно для запрошення користувачів у проєкт

3.3.4. Kanban панель

Після створення проєкту у в'юпорті з'явиться нова картка проєкту, перейшовши в яку відкриється класична канбан система (Рис. 3.15): дошки з картками завдань.

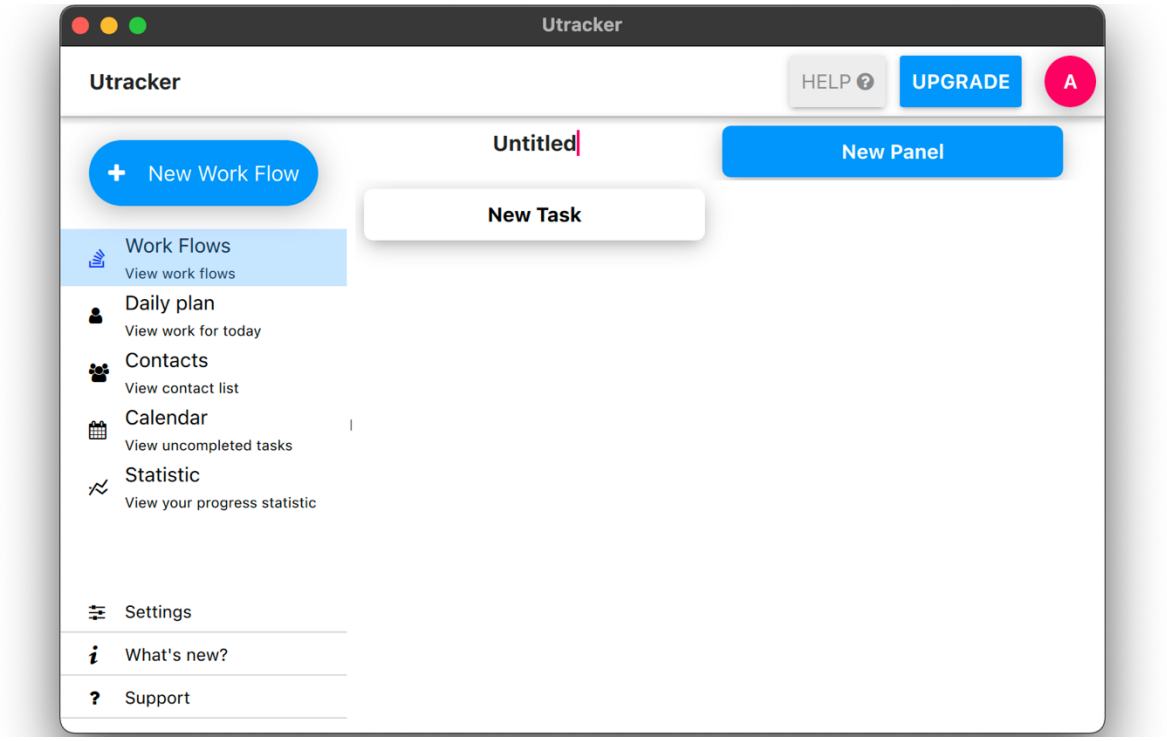

Рис. 3.15. Вікно Kanban панелі

Користувач може створювати нові дошки натискаючи кнопку створення панель. По замовчуванню їх назва «Untitled», але це можна змінити за потребою користувача. Завдання ж додаються за допомогою кнопки створення нових завдань. Після чого з'являється завдання по замовчуванню, яке можна налаштувати (Рис. 3.16) за потребою натиснувши спершу на картку самого завдання.

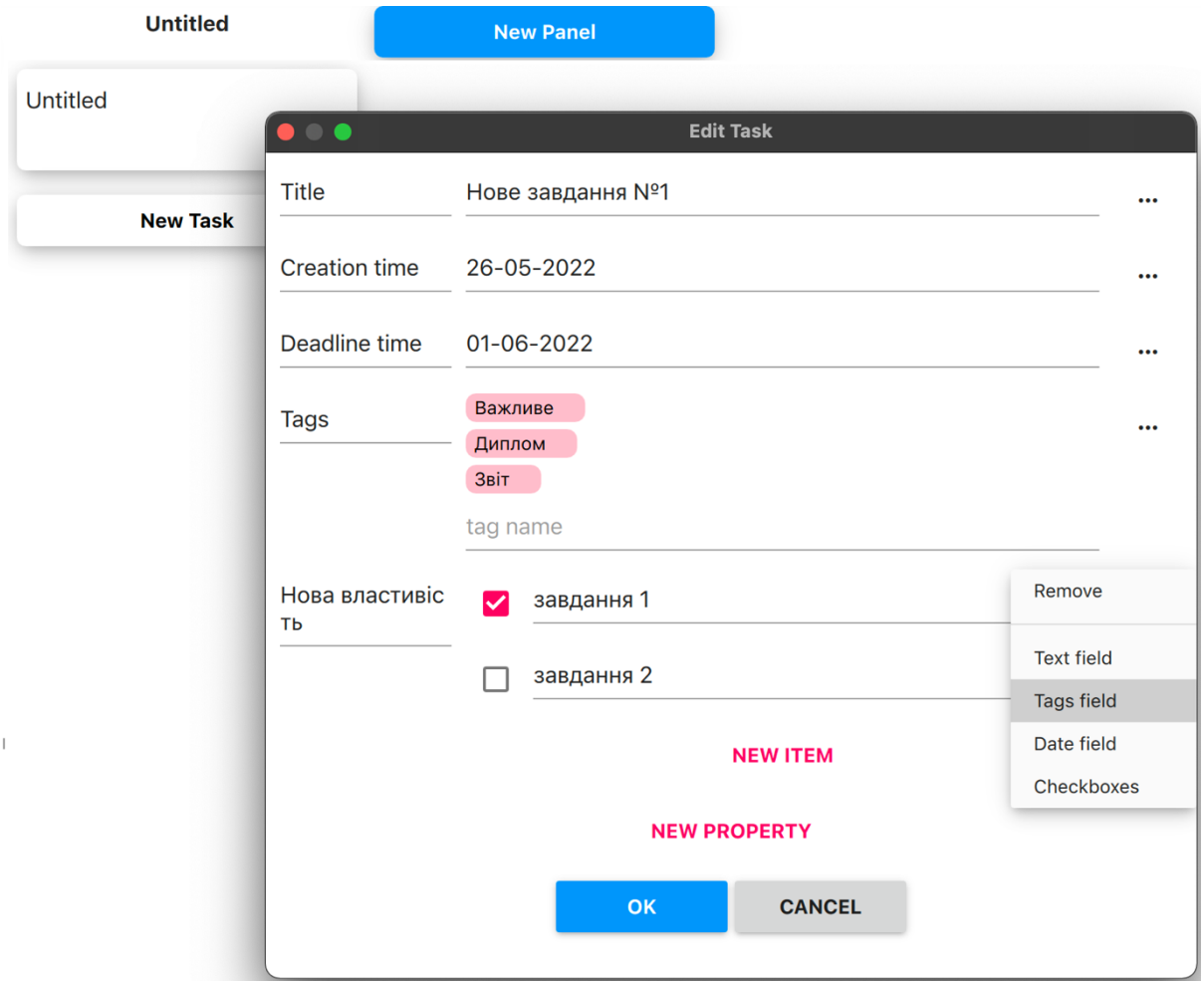

Рис. 3.16. Вікно зі налаштування завдання в Kanban панелі

Саме тут реалізовувалась гнучкість застосунку, яка полягала у вільному налаштуванні полів завдання. Користувач може створювати необмежену кількість певних пунктів, деталей, які необхідні йому для пояснення завдання. Після створення такого завдання зміни одразу ж відобразяться в панелі (Рис. 3.17).

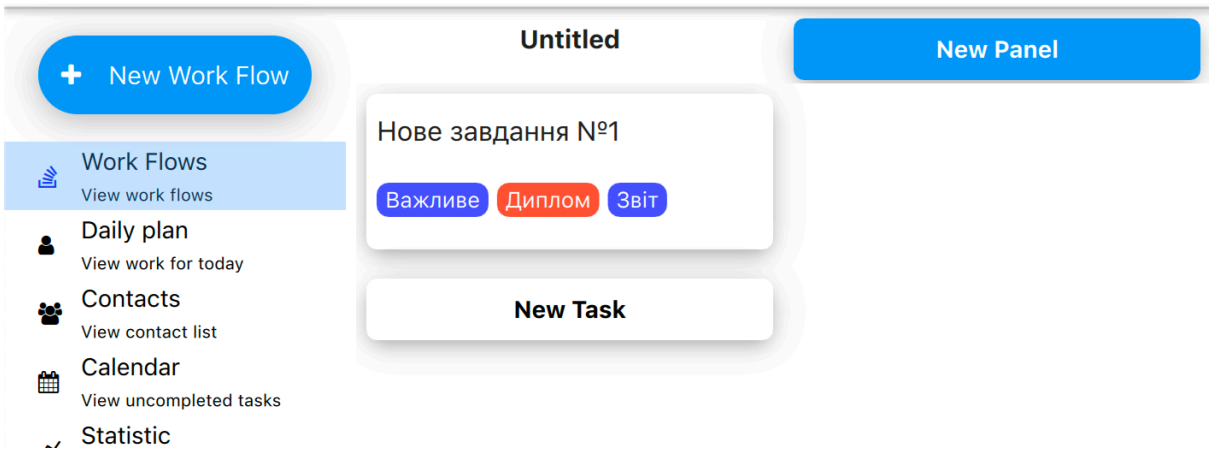

Рис. 3.17. Вигляд панелі після налаштування завдання

В той же час користувач може викликати контексне меню (Рис. 3.18) над завданням і позначити себе як його виконавця, або позначити, що завдання вже повністю виконане або взагалі видалити його.

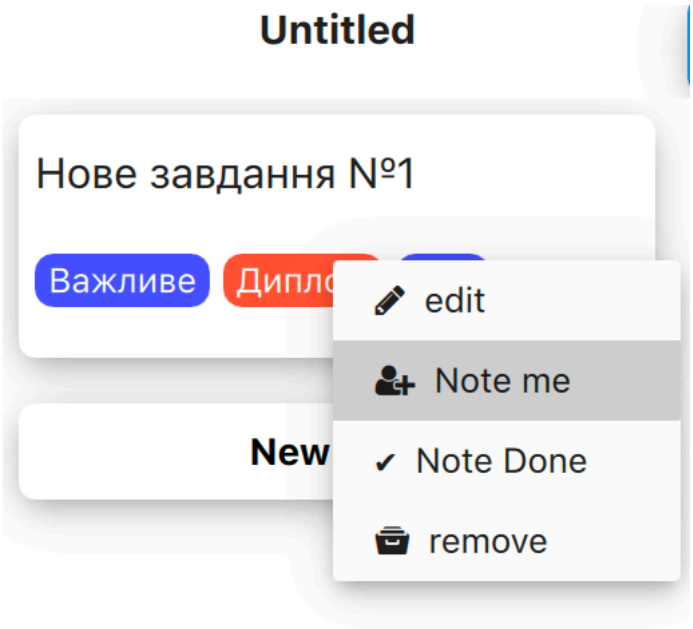

Рис. 3.18. Контексне меню завдання

Після відмітки виконавця з’являється відповідне анімоване кільце в правому верхньому куті, що інформує, що завдання наразі в прогресі або якщо позначити, що завдання виконане, то на тому ж місця буде позначка виконаного завдання (Рис. 3.19).

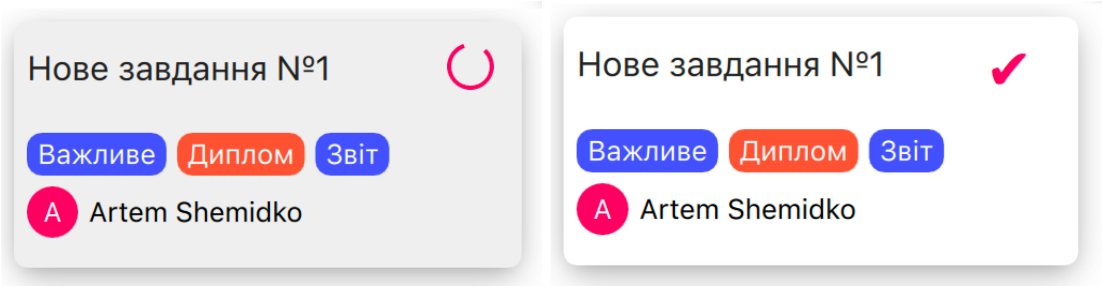

Рис. 3.19. Зміна станів завдання зі стану «в прогресі» на стан «виконано»

Після всіх налаштувань та розподілення завдань між запрошеними виконавцями канбан може мати вигляд показаний на Рис. 3.20.

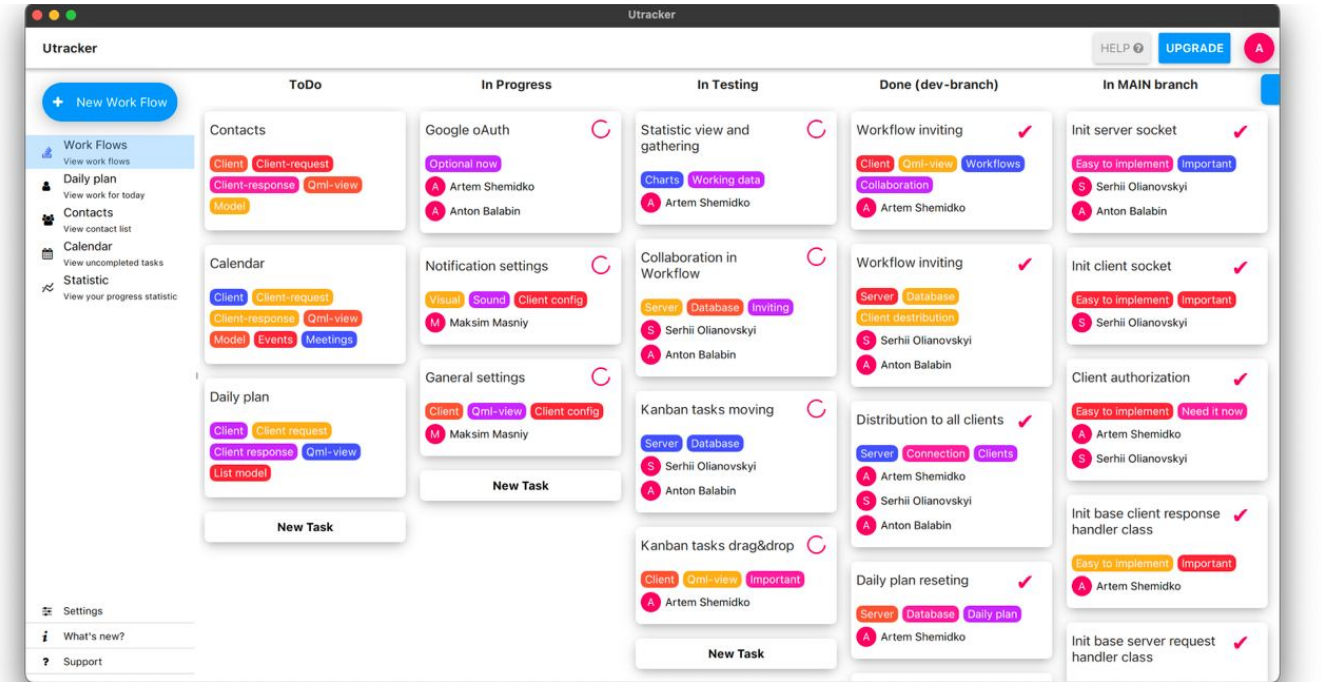

Рис. 3.20. Вигляд Kanban панелі проєкту розробки застосунку

3.3.5. Панель плану на день

Наступним важливим вікном системи керування проєктами є також і особистий план на день (Рис. 3.21). Дане вікно дещо схоже на звичайний записник, або канбан дошка лише з однією панеллю.

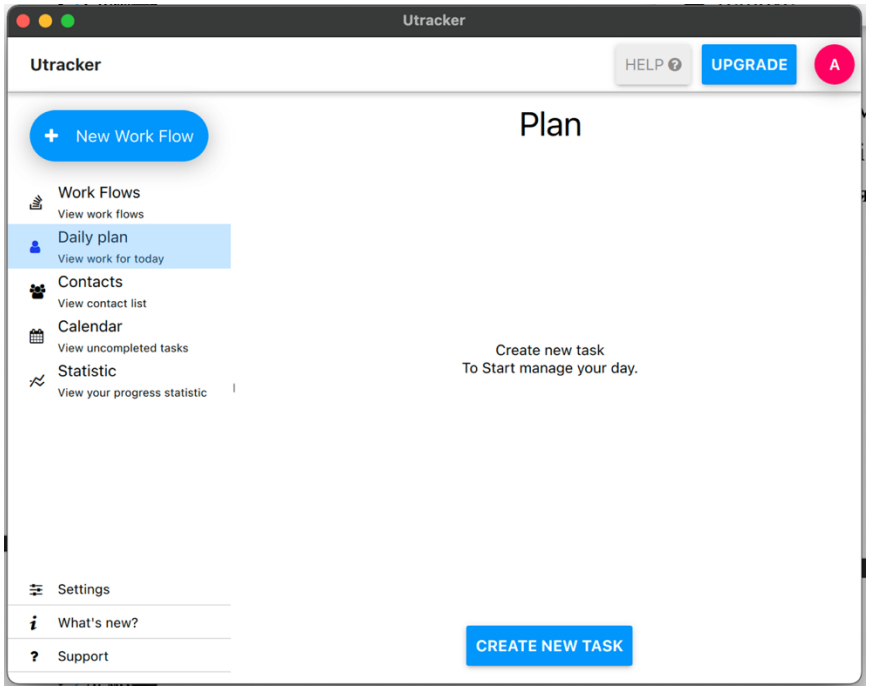

Рис. 3.21. Вікно щоденного плану користувача

Тут користувач може створювати нове завдання, яке якось описує. Після його виконання ставить відповідну відмітку або повністю його (Рис. 3.22). В кінці кожного дня система автоматично очищує цей список.

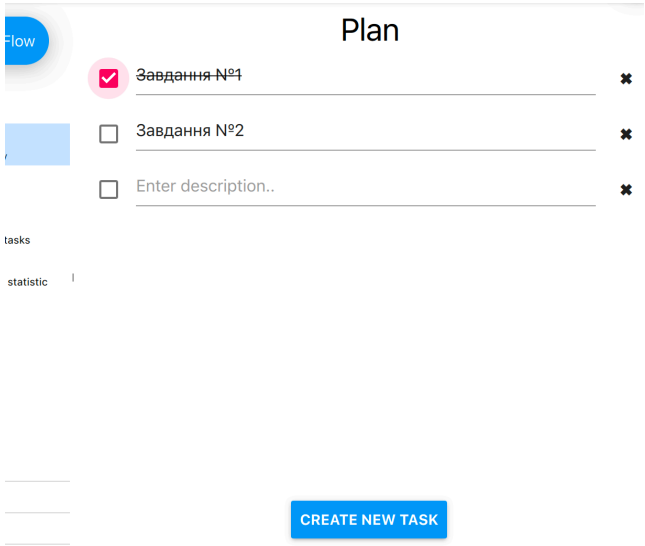

Рис. 3.22. Створення та керування завданнями щоденного плану

3.3.6. Вікно статистики користувача

Не менш важливим вікном є вікно зі загальною статистикою (Рис. 3.23), адже воно надає можливість швидко відстежити слабкі сторони та проаналізувати тенденцію. В даному розділі користувач може переглянути співвідношення виконаних, невиконаних та завдань зроблених невчасно по проєктам, канбанам та особистого щоденного плану.

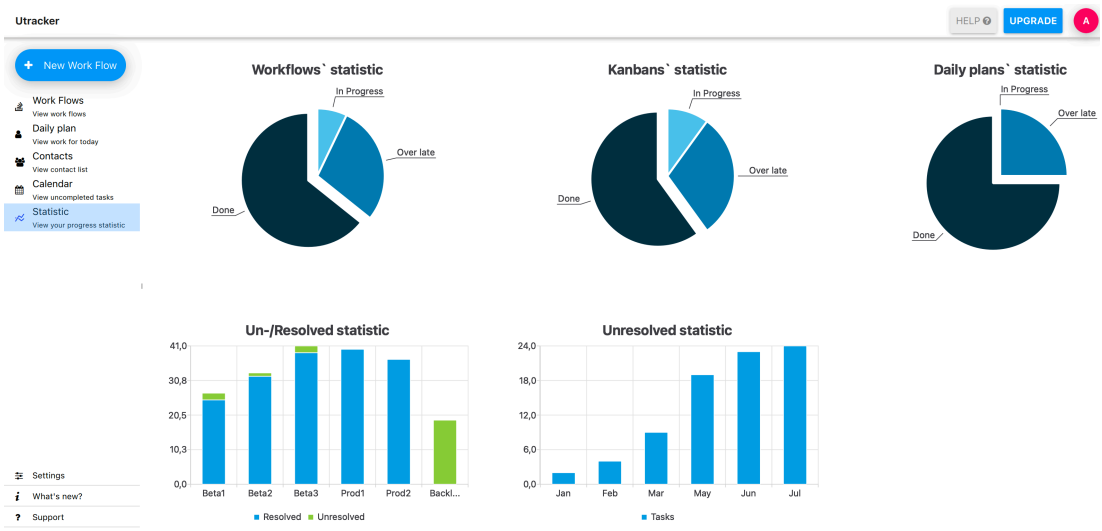

Рис. 3.23. Панель статистики

Тут користувачу також надається можливість відстежувати співвідношення створених, виконаних та невиконаних завдань в часових рамках.

3.4. Висновки до розділу 3

У цьому розділі описано головну функціональність системи управління IT-проєктами та проведено огляд її головних сторінок. З її допомогою користувачі можуть об’єднуватись в групи для реалізації спільних ідей та проєктів. Також є можливість керування власного дня, що доповнює Kanban систему керування.

Застосунок «Програмна система управління IT проєктами» дійсно має простий інтерфейс користувача – не є перевантажена різною функціональністю і є інтуїтивно зрозумілою, більшість дій здійснюються в декілька кліків і не потребують додаткових підтверджень. Серед основної функціональності: створення проєктів та їх налаштування (встановлення назви, дати здачі, запрошення інших користувачів для реалізації проєкту), ведення, керування та відслідковування Kanban системи, гнучке налаштування завдань, щоденний план та отримання аналітичних даних по проєктах у формі статистичних графіків, які допомагають відслідковувати тренд. Функціональна система працює стабільно, без перебоїв та помилок на Unix системах (Linux, Mac) та Windows, та відповідає вимогам систем керування.

ВИСНОВКИ

Під час виконання дипломного проєкту було глибоко досліджено актуальність проблеми ефективного керування проєктами та системи, які покликані мінімізувати навантаження на керівників та виконавців, покращити взаємодію між ними та надати структурований вигляд завдань, їх описів, термінів виконання, тощо. Проаналізувавши недоліки таких програмних комплексів та потреб, з якими стикаються проєктні менеджери та їх команди, було розроблено власну повноцінну програмну систему для ведення, керування та збору і огляду аналітичних даних проєктів, що надає можливість з легкістю контролювати процесом розробки та терміни виконання.

Проєкт був реалізований базуючись першочергово на архітектурі клієнт-сервер, тому що це спрощує етап створення додатку під нові системи, адже частина з логікою керування та зберігання даних може бути універсальною для всіх платформ. Це архітектурне рішення було поєднано разом зі архітектурою мікросервісів та model-view-controller. Обидві частини клієнт та сервер написані на мові C++ з використанням фреймворку Qt з його вбудованими модулями, а серед шаблонів проєктування було використано: «одинак», міст та шаблонний метод. Шифрування сокет з’єднання між одиницями реалізувалось через SSL ін’єкцію.

Отриманий перелік необхідного функціоналу в ході дослідження методологій проєктного менеджменту, запитів клієнтів та власного досвіду користування подібними програмами, в проєкті реалізувався у вигляді необмеженої класичної Kanban панелі з можливість гнучко налаштовувати завдання в ній, що притаманно досліджуваній системі Notion. В той же час користувач не перевантажений функціональністю додатку, який має інтуїтивно зрозумілий інтерфейс. Серед основної функціональності можна виділити: авторизацію користувачів, створення workflow (проєктів), керування ними в Kanban, запрошення користувачів до ведення цих проєктів, ведення власного розпорядку дня, що має властивість щоденного самоочищення та проєктну аналітику. Остання властивість має важливе значення для швидкого аналізу прогалин та тенденцій у веденні проєкту. Саме аналітика даних

надає багато можливостей для підвищення кваліфікації членів команди та оптимізації процесу впровадження управління проєктами. Зрештою, незалежно від кінцевої мети, завжди можна знайти дані, щоб вплинути на результати. Використання даних для аналізу минулої інформації, інформації в реальному часі та майбутньої, здійснюється для того, щоб моделювати ймовірність результатів проєкту та використовують їх для прийняття рішень на основі даних та підвищення ефективності.

Орієнтуючись на ринок IT індустрії, в додаток була додана інтеграція з такими сервісами як Google API та GitHub API. Саме за допомогою них користувачі зможуть з легкістю використовувати контакти Google, додавати синхронізовані події до календаря, підв’язуватись до кодової бази репозиторіїв проєктів на GitHub, тощо. Ця особливість також супроводжувалась реалізацією авторизації oAuth2.0, що дає змогу користувачам приєднуватись до системи за лічені секунди.

Використання цієї системи для ведення проєктів чи власного розпорядку дня може частково вирішити проблеми ефективного використання як часових так і фінансових, що є необхідним сьогодення.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is project management [Електронний ресурс]. Режим доступу: https://www.apm.org.uk/resources/what-is-project-management/ [Дата звернення: 10 травня 2022р.] – Назва з екрану.

2. Project management [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Project_management [Дата звернення: 12 травня 2022р.] – Назва з екрану.

3. Project life cycle [Електронний ресурс]. Режим доступу: https://www.workfront.com/project-management/life-cycle [Дата звернення: 13 травня 2022р.] – Назва з екрану.

4. Why are project management process groups are necessary? [Електронний ресурс]. Режим доступу: https://www.villanovau.com/resources/project-management/pmbok-project-management-process-groups/ [Дата звернення: 15 травня 2022р.] – Назва з екрану.

5. Features to look for in a project management system [Електронний ресурс]. Режим доступу: https://www.businessnewsdaily.com/15911-project-management-software-features.html [Дата звернення: 15 травня 2022р.] – Назва з екрану.

6. Project management methodologies [Електронний ресурс]. Режим доступу: https://asana.com/resources/project-management-methodologies [Дата звернення: 16 травня 2022р.] – Назва з екрану.

7. Top 10 most popular project management methodologies [Електронний ресурс]. Режим доступу: https://www.projectmanager.com/blog/project-management-methodology [Дата звернення: 17 травня 2022р.] – Назва з екрану.

8. Use of data analitycs in project management [Електронний ресурс]. Режим доступу: https://www.linkedin.com/pulse/use-data-analytics-project-management-bohitesh-misra/ [Дата звернення: 17 травня 2022р.] – Назва з екрану.

9. How to choose a right software architecture [Електронний ресурс]. Режим доступу: https://techbeacon.com/app-dev-testing/top-5-software-architecture-patterns-how-make-right-choice [Дата звернення: 20 травня 2022р.] – Назва з екрану.

10. 10 Best software architecture patterns [Електронний ресурс]. Режим доступу: https://www.simform.com/blog/software-architecture-patterns/ [Дата звернення: 21 травня 2022р.] – Назва з екрану.

11. Top 7 most powerful features of C++ [Електронний ресурс]. Режим доступу: https://www.upgrad.com/blog/features-of-c-plus-plus/ [Дата звернення: 26 травня 2022р.] – Назва з екрану.

12. Features of C++ [Електронний ресурс]. Режим доступу: https://data-flair.training/blogs/features-of-c-plus-plus/ [Дата звернення: 27 травня 2022р.] – Назва з екрану.

13. About Qt [Електронний ресурс]. Режим доступу: https://wiki.qt.io/About_Qt/ [Дата звернення: 25 травня 2022р.] – Назва з екрану.

14. Паттерни/шаблони проєктування [Електронний ресурс]. Режим доступу: https://refactoring.guru/uk/design-patterns/ [Дата звернення: 21 травня 2022р.] – Назва з екрану.

15. The C++ resources network [Електронний ресурс]. Режим доступу: https://www.cplusplus.com/ [Дата звернення: 24 травня 2022р.] – Назва з екрану.

16. Cppreference [Електронний ресурс]. Режим доступу: https://en.cppreference.com/w/ [Дата звернення: 24 травня 2022р.] – Назва з екрану.

17. All classes [Електронний ресурс]. Режим доступу: https://doc.qt.io/qt-5.15/classes.html [Дата звернення: 24 травня 2022р.] – Назва з екрану.