МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Олексій Горський
"____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: "Методика та програмний застосунок для управління pre-sales етапом розробки ПЗ"

Виконавець: Штиба Ярослав Генадійович

Керівник: к.т.н., доцент Ходаков Данііл Вікторович

Нормоконтролер: к.ф.-м.н., ст.викл. Гололобов Дмитро Олександрович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра Інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Програмне забезпечення систем»

ЗАТВЕРДЖУЮ
Олексій Горський
"____"_____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Штиби Ярослави Генадійовичи

1. Тема проекту: Методика та програмний застосунок для управління pre-sales етапом розробки ПЗ.

затверджена наказом ректора від 10.09.2023 р. № 2582/ст

2. Термін виконання проекту: з 10.10.2023 р. до 31.12.2023 р.

3. Вихідні данні до проекту : програмний продукт розробити за допомогою програмного середовища WebStorm, та з використанням технологій: JS, React, HTML, CSS/SASS, Node.JS.

4. Зміст пояснювальної записки:

1. Pre-sales етап розробки ПЗ.

2. Вимоги до засобу управління pre-sales етапом розробки ПЗ.

3. Архітектура та проєктування засобу управління pre-sales етапом розробки ПЗ.

5. Перелік обов'язкового графічного матеріалу:

1. Інтерфейс користувача.

2. Діаграма класів.

3. Діаграма прецендентів.

4. Скріншоти роботи програмного засобу.

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Ознайомлення з постановкою задачі та вивчення літератури Написання 1 розділу, представлення керівнику		
2.	Попередній друк 1 розділу та допоміжних сторінок (черновик) - титульної, завдання, графіка, реферат, список скорочень, зміст, вступ, список джерел. 1-ий нормо-контроль.		
3.	Написання 2 розділу, представлення керівнику		
4.	Написання 3 розділу, представлення керівнику		
5.	Загальне редагування та друк пояснювальної записки, графічного матеріалу		
6.	Проходження нормо-контролю, перепліт пояснювальної записки.		
7.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації		
8.	Отримання відгуку керівника, рецензії.		
9.	Підготовка матеріалів для передачі		

	секретарю ДЕК (ПЗ, ГМ, CD-R з електронними копіями ПЗ, ГМ, презентації, відгук керівника, рецензія, довідка про успішність, 1 папка, 1 конверт)		

7. Дата видачі завдання 10.09.2023

Керівник: _____ к.т.н., доцент Ходаков Д.В.

Завдання прийняв до виконання: _____ Штиба Я.Г.

Дата

РЕФЕРАТ

Пояснювальна записка до дипломного проекту "Методика та програмний застосунок для управління pre-sales етапом розробки ПЗ": 78 с., 33 рис., 5 таб., 12 інформаційних джерел.

Об'єкт розробки – застосунок для управління pre-sales етапом розробки ПЗ.

Мета проекту – розробити зручний застосунок для управління pre-sales етапом розробки ПЗ, використовуючи найсучасніші технології веб розробки.

ABSTRACT

Explanatory note to the diploma project "Methodology and software application for managing the pre-sales stage of software development": 78 pages, 33 figures, 5 tables, 12 information sources.

The **object** of development is an application for managing the pre-sales stage of software development.

The **goal** of the project is to develop a convenient application for managing the pre-sales stage of software development, using the most modern web development technologies.

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ПЗ – програмний засіб;

CLI – Common Language Infrastructure;

API – application programming interface;

COM – Component Object Model.

JS – JavaScript

ІС – інформаційні системи

DB - database

ВСТУП

У будь-якій сфері, кожній особі важливо майстерно управляти своїм часом та організовувати робочий процес. Недавно поняття "проект" було пов'язане головним чином із технічними галузями, такими як будівництво (проект будинку, заводу, вокзалу). Проте сучасна наука управління визнає його ключовим елементом, що відкриває значні можливості для реалізації ідей у різних сферах, включаючи культуру, науку, техніку, а також соціальні та медичні сфери. У зв'язку з цим, управління проектами стає невід'ємною частиною сучасного менеджменту.

Керування людьми - завдання непросте, а ефективне керування надзвичайно складне. Тому виникає необхідність поєднувати різні практики, алгоритми та технології контролю, оцінки результатів праці. В управлінні переходять від застарілих методик до новітніх, намагаючись врахувати всі аспекти та ризики, і витрачають ресурси на планування замість стандартного виконання завдань.

Проектний менеджмент (PM) стає надзвичайно популярним у всіх галузях, оскільки багато компаній переходять від класичних (бюрократичних) схем управління до проектних моделей, де кожна задача розглядається окремо, і відповідальність делегується. У галузі управління IT-проектами, PM стає дисципліною, яка об'єднує процедури, принципи та стратегії ведення бізнесу, керуючи проектом від концепції до завершення.

З іншого боку, кожен проект проект проходить етап Project Discovery та Pre-sales. Чому ці фази такі важливі? Тому, що успішне завершення фази discovery дає змогу правильно визначити бізнес-потреби, проблеми та можливості, що тим самим означає, що ваші рішення, швидше за все, відповідатимуть вимогам бізнесу.

Однак це не єдина перевага, яку ваша команда може отримати, проводячи discovery на початковому етапі проекту. Це також допомагає:

• Зменшити ризики неправильного розуміння вимог, через що команда розробників надає неточні оцінки часу та бюджету.

• Надавати чітку дорожню карту з проміжними цілями, результатами та кінцевими термінами, що зменшують перепланування або можливі зміни після початку проекту.

• Формує довіру. Регулярне спілкування та обговорення вашого проекту з командою може допомогти встановити двосторонню довіру.

РОЗДІЛ 1
PRE-SALES ЕТАП РОЗРОБКИ ПЗ

1.1. Короткий опис фаз життєвого циклу проекту

Життєвий цикл розробки програмного забезпечення (ЖЦ) — це комбінація фаз, які має пройти проект від початку до завершення. Типовими етапами життєвого циклу розробки програмного забезпечення є ініціація, розробка концепції, планування, визначення вимог, дизайн інтерфейсу користувача, розробка, інтеграція, тестування, розгортання, гіпердогляд, технічне обслуговування, передача, закриття та підтримка.

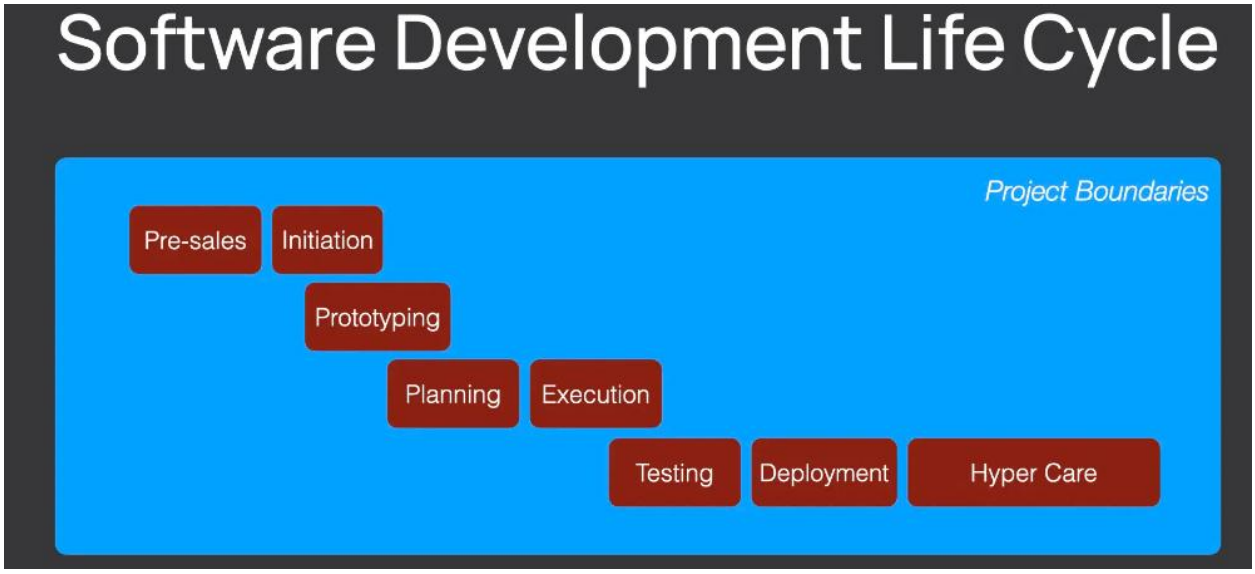


Рис. 1.1. Фази ЖЦ проекту

Фаза проекту – це набір логічно пов’язаних проектних дій, які завершуються виконанням одного або кількох результатів.

Проект не обов’язково повинен включати всі етапи. Ви, як керівник проекту, повинні вирішити, як це розбити.

Табл. 1.1.

Види фаз ЖЦ, їх опис фаз та результати

Фаза	**Опис**	**Результат**
Pre-sales Phase	Переговори з клієнтами, відділом продажів, малим і середнім бізнесом для залучення нових клієнтів.	– Підписаний договір
Project Initiation Phase	Менеджер проекту допомагає оцінити здійсненність і надати оцінку високого рівня.	– Проектна пропозиція
Concept Development Phase	Керівник проекту архівує всю проектну документацію, збирає отримані уроки тощо.	- Статут проекту
Prototyping Phase (Usually, it's a Phase Gate to ensure the feasibility of the project.)	Команда проекту забезпечує передачу знань і створює необхідну документацію та процеси для команди підтримки для підтримки продукту.	– Інформація про проект високого рівня
Planning Phase	Менеджер проекту працює з SME для створення документа, який описує загальний підхід до досягнення мети проекту.	– Доопрацьована та підтверджена концептуальна документація
Requirements Definition Phase	Концепція описує, що робити в загальних рисах, але може не визначати, як це робити.	– Концептуальні проекти
UI Design Phase	Команда проекту працює над невеликим проектом для створення робочої моделі майбутнього продукту.	– Прототип кінцевого продукту або послуги.
Development Phase	Мета полягає в тому, щоб довести припущення, яке критично впливає на здійсненність.	Рішення про перехід до наступного етапу.
Integration	Керівник проекту обирає підхід до управління проектом. Разом з командою вони створюють план управління проектом.	– Затверджений план управління проектом

Фаза	**Опис**	**Результат**
Testing Phase	Зміст плану може відрізнятися залежно від середовища.	– Матриця відстеження вимог
Deployment Phase	Команда проекту збирає вимоги зацікавлених сторін.	– Вимоги до документації
Hyper Care Phase	Менеджер проектів сприяє роботі бізнес-аналітиків і предметних експертів.	– Макети
Maintenance Phase	Керівник проекту працює з дизайнерами та художниками-оформлювачами, щоб створювати проекти інтерфейсу користувача, які відповідають зібраним вимогам.	– Каркаси
Hand-off, Closure, Support Phase(s)	Команда проекту дотримується Плану управління проектом. Керівник проекту виконує щоденні дії, щоб контролювати роботу, усунути перешкоди та зменшити ризики.	– Графічні ресурси

Знову ж таки, НЕ зосереджуйтеся надто на заголовках. Тут важлива робота та її характер.

Майте на увазі, що перед початком етапів управління проектами відбувається цілий процес пошуку клієнтів і продажу їм ваших послуг.

У великих корпоративних компаніях процес не такий вже й далекий від відносин клієнт-постачальник. Це лише внутрішній процес вибору, в який проект інвестувати гроші.

1. Фаза ініціації проекту

Після підписання контракту ви переходите на етап ініціації.

Тут вам потрібно визначити мету проекту та критерії його успіху.

Навіть якщо ви укладаєте контракт на терміни та матеріали, життєво важливо пройти цей етап.

Тому необхідно створити Статут проекту. Принаймні вам потрібно визначити ключову інформацію, яку він має.

Другим важливим кроком є визначення ключових зацікавлених сторін.

Вам знадобиться багато інформації щодо багатьох аспектів розробки програмного забезпечення:

- Інтерфейс користувача
- Досвід користувача
- Технічні обмеження
- Апаратна специфіка
- Середовище розробки
- Архітектура програмного забезпечення

Тому не розраховуйте отримати всі подробиці від своїх клієнтів і клієнтів. Вам потрібно буде включити внутрішню експертизу. У тому числі поза вашою командою.

Ознайомившись із проектом і його цілями, ви можете перейти до наступного етапу.

2. Етап розробки концепції

Існує багато різних способів розробки програмних додатків.

Я віддаю перевагу підходу «перш за все дизайн».

Отже, як це працює?

Перш за все, вам потрібно розробити концептуальний дизайн і каркаси для майбутніх застосувань.

Це не повинно бути повністю деталізованим. Але він повинен забезпечити структуру для майбутніх заходів із визначення вимог.

Однак це дає відчутні результати вашим зацікавленим сторонам.

Вони будуть більше залучені до всього проекту, коли складний процес розробки програмного забезпечення буде спрощено до образів майбутнього продукту.

Іншим аспектом етапу розробки концепції є технічний.

Іноді необхідний продукт абсолютно унікальний. Або він має специфічні вимоги, аналогів яких немає. І жодне доступне рішення не забезпечить бажаного результату.

Отже, вам потрібно розробити абсолютно нову технологічну концепцію.

Вам потрібно буде підібрати найкращих експертів у своїй галузі, щоб генерувати ідеї та рішення.

Тому в більшості випадків це фаза шлюзу.

Якщо у вас є прийнятне рішення, ви продовжите проект. Якщо рішення потребує ресурсів і часу понад обмеження, можливо, доведеться скасувати проект.

3. Фаза планування

У великих IT-проектах вам потрібно буде створити повний план управління проектом.

Однак на початку ви працюватимете над меншими проектами. Там планування буде простим.

Ось правда:

Багато постачальників програмного забезпечення розробляють індивідуальний підхід до управління проектами.

Він природним чином формується в процесі розвитку компанії.

Отже, вам потрібно буде дотримуватися або абсолютно спеціального підходу, або варіації Scrum або Kanban.

У цьому випадку планування проекту зводиться до:

1. Визначення обсягу проекту

2. Оцінки часу та витрат

3. Встановлення віх

Іноді вам потрібно буде визначити необхідні ресурси та досвід.

Усі інші аспекти управління проектом будуть визначені звичаями цієї організації.

Тому наприкінці цього етапу у вас буде або план управління проектом.

Або ви візьмете на себе зобов’язання надати певний обсяг до визначеного терміну. А решта – внутрішні проблеми.

4. Етап визначення вимог

Після того як ви сплануєте, як керувати проектом, вам потрібно з’ясувати, що вам потрібно розробити.

На цьому етапі ви можете використовувати всі доступні параметри для визначення вимог:

• Мозковий штурм

• Інтерв'ю

• Фокус-групи

• Анкети та опитування

• Аналіз документів

• Mind Mapping

• Каркаси

• Історії користувачів

Спеціального підходу до збору вимог немає.

Зрештою, вам потрібні вимоги, які зрозуміють інженери програмного забезпечення.

Іноді прості зміни у вимогах можуть спричинити значну зміну необхідних зусиль. Напевно, ви хочете стежити за такими можливостями, щоб спростити майбутній продукт.

Після цього ви можете включити сюди аналіз здійсненності високого рівня.

Ось чому ви часто можете бачити матрицю відстеження вимог у проектах програмного забезпечення. Навіть у гнучкому середовищі.

5. Фаза проектування

Знову ж таки, тут є щонайменше два аспекти:

1. Технічна архітектура

2. UI/UX дизайн

На цьому етапі вам потрібно проаналізувати зібрані вимоги. Після цього розробіть архітектуру, яка підтримуватиме їх.

Крім того, вам потрібен інтерфейс користувача, який зробить програму чи службу придатною для використання.

Результати цього етапу:

• Макети

• Каркаси

• Діаграми робочого процесу

• Документація опису архітектури

• Список технологій, фреймворків і бібліотек

Після того, як ви все зібрали, ви можете почати писати код для реалізації вимог.

6. Фаза розробки

На цьому етапі ви починаєте повсякденне виконання плану проекту.

Розробники програмного забезпечення налаштують робоче середовище та почнуть писати код.

Тут ви виконуватимете фактичну роботу, дотримуючись одного з вибраних фреймворків.

Заради кращої якості та залучення зацікавлених сторін ви можете робити це в ітераційний та поетапний спосіб.

Що це означає?

Ви розробите робочу частину програми. Ви покажете це зацікавленим сторонам.

Після цього зацікавлені сторони нададуть відгук. Ви внесете зміни в проект.

Цикл повторюється.

Однак є підступ:

Такий підхід не звільняє вас від виконання проекту вчасно та в рамках бюджету.

Таким чином, ваші клієнти все ще можуть мати жорсткі терміни та обмеження.

7. Фаза інтеграції

Сьогодні ви рідко побачите програму, яка не інтегрується з іншими службами чи програмами.

У корпоративних середовищах ви часто побачите, що дані та облікові дані співробітників зберігаються в окремій службі. Вам може знадобитися інтеграція з ним, щоб отримати доступ до бази даних співробітників.

Місце для зберігання часто передається стороннім постачальникам, таким як Amazon, Dropbox тощо. Отже, вам потрібно з ним інтегруватися.

Загалом, вам потрібно інтегрувати програмне забезпечення з бізнес-процесами компанії чи ринку, на якому ви продаєте додаток.

Цей етап зазвичай вимагає багато співпраці. Це може зайняти багато часу.

Тому вам краще спланувати це заздалегідь. Зверніть увагу на можливі ризики та вимоги до інтеграції інших служб.

Досить часто інтеграція є частиною розвитку.

Однак, якщо зусиль буде багато, краще це зробити окремим етапом.

8. Етап тестування

Ця фаза також називається приймальним тестуванням. Або фінальне тестування.

Ви повинні розуміти, що ви повинні постійно тестувати свою програму з самого початку. Крім того, потрібно не допускати дефектів у процесі розробки.

В ідеалі ваша програма має бути стабільною та без серйозних дефектів весь час.

Отже, це питання забезпечення якості, яке ви планували та виконували на всіх попередніх етапах.

Тут вам потрібно підтвердити, що версія програми, яку ви тестуєте, має необхідну якість.

Це не означає, що додаток не має недоліків. Вони присутні, але не заважають людям користуватися додатком.

Вам потрібно надати список усіх відомих дефектів у заявці як частину сертифіката якості.

9. Фаза розгортання

Програмне забезпечення не працює без обладнання. Будь то сервер, ваш ПК чи Mac або мобільний пристрій, вам потрібно доставити туди свою програму чи послугу.

Крім того, розгортання великої та складної програми може бути проектом саме по собі.

Вам потрібно налаштувати сервери, завантажити свою програму та підключити її до всіх інших служб і серверів.

Для мобільних пристроїв і настільних комп'ютерів вам потрібно створити інсталятор програми. Вам також може знадобитися надіслати цей інсталятор на ринок (App Store, Google Play тощо)

Загалом, результатом цього етапу є ваша програма або послуга, доступна для кінцевих користувачів.

10. Фаза технічного обслуговування

Додаток обслуговується та підтримується протягом усього терміну служби. Це частина життєвого циклу продукту, яка виходить за межі життєвого циклу проекту.

Однак досить часто ви можете побачити відносно короткий етап обслуговування.

У цей період частина команди проекту продовжує працювати з додатком після того, як він потрапить до користувачів.

Вони тут, щоб усунути будь-які критичні дефекти, які були упущені.

Через кілька місяців, коли програму використовують і «тестують» тисячі користувачів, команда проекту звільняється, а продукт передається окремій групі підтримки.

11. Фази передачі, закриття, підтримки

Передача також може вимагати багато зусиль.

Вам потрібно зібрати всі отримані знання про програму та її особливості. Після цього потрібно передати ці знання команді підтримки.

Крім того, вам може знадобитися створити документацію, спеціальні можливості для адміністрування служби тощо.

Якщо це так, вам потрібно визначити такі вимоги на ранній стадії.

Також подумайте про процес передачі на початку проекту.

1.2. Приклади життєвих циклів розробки програмного забезпечення

Життєвий цикл розробки програмного забезпечення великого проекту

Великі проекти можуть включати етапи, які є більшими за проекти, якими ви зараз керуєте.

Тому, якщо ви вважаєте, що такі підходи застаріли, не будьте занадто обмеженими.

Майте на увазі, що підхід до водоспаду все ще має місце. Це має свої плюси і мінуси.

Але досить часто ви помічаєте, що навіть керований планом життєвий цикл матиме ітерації та збільшення.

Навіть у великих проектах потрібен швидкий зворотний зв’язок між різними етапами.

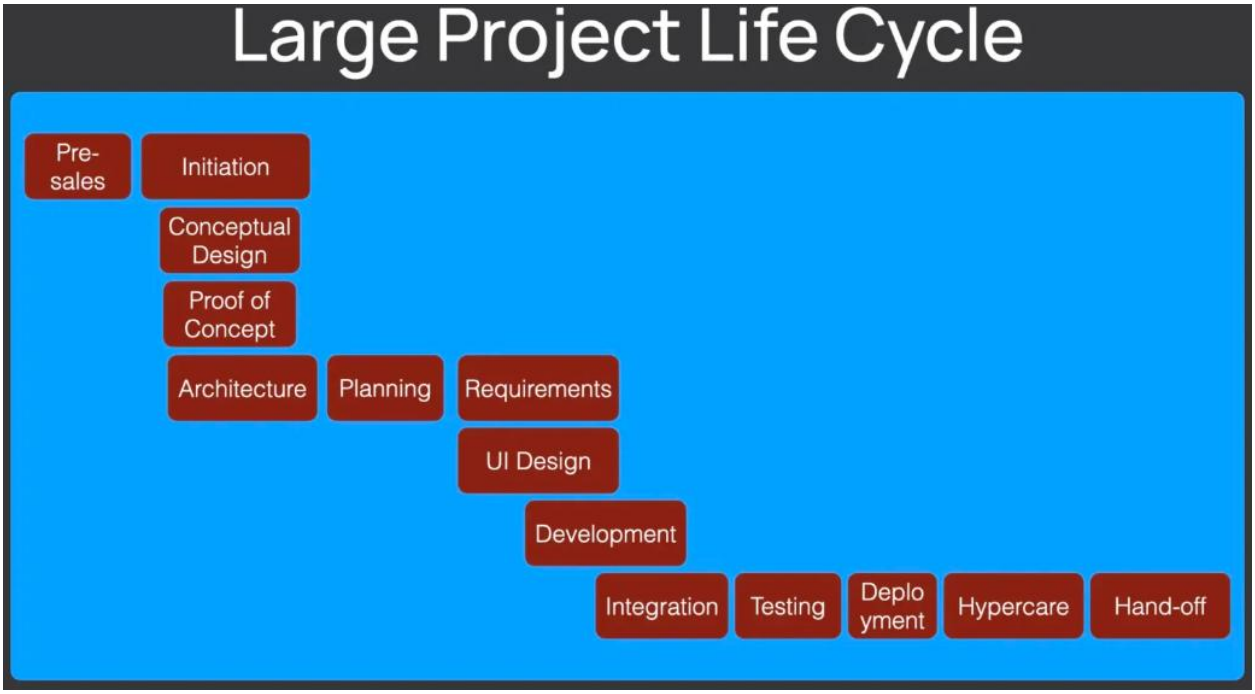


Рис. 1.3. Фази життєвого циклу розробки програмного забезпечення великого проекту

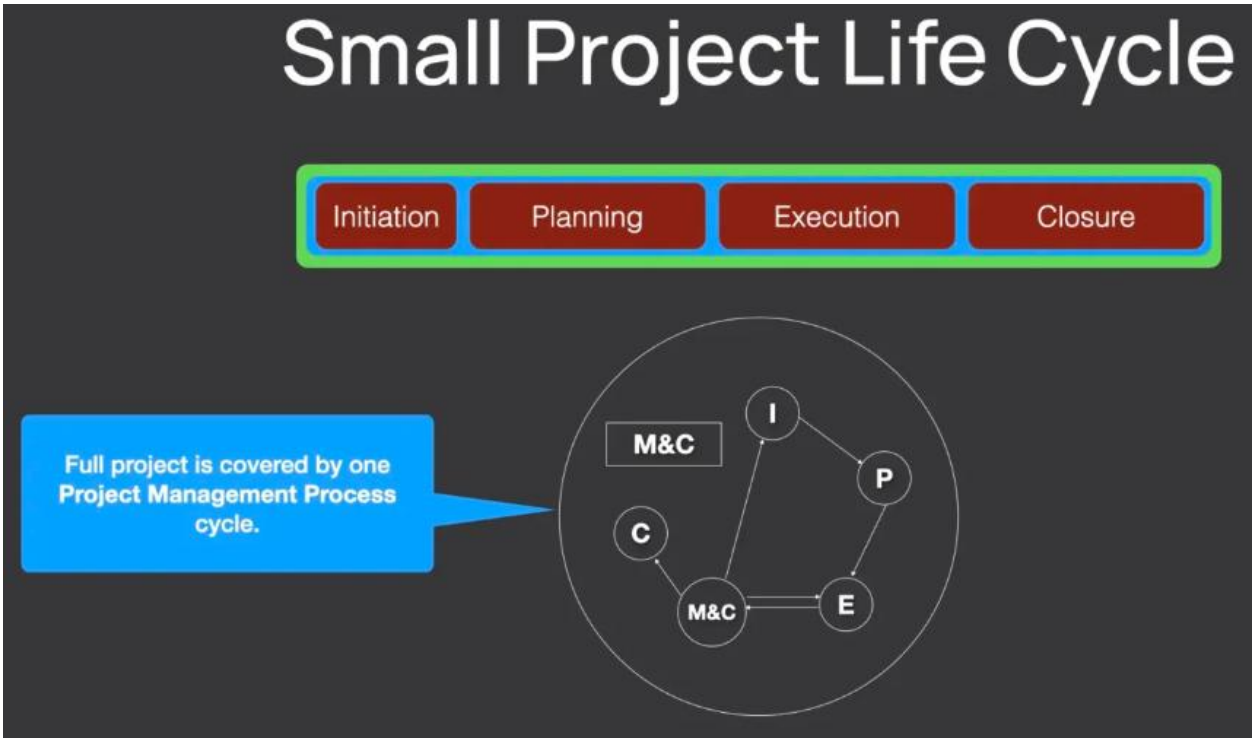
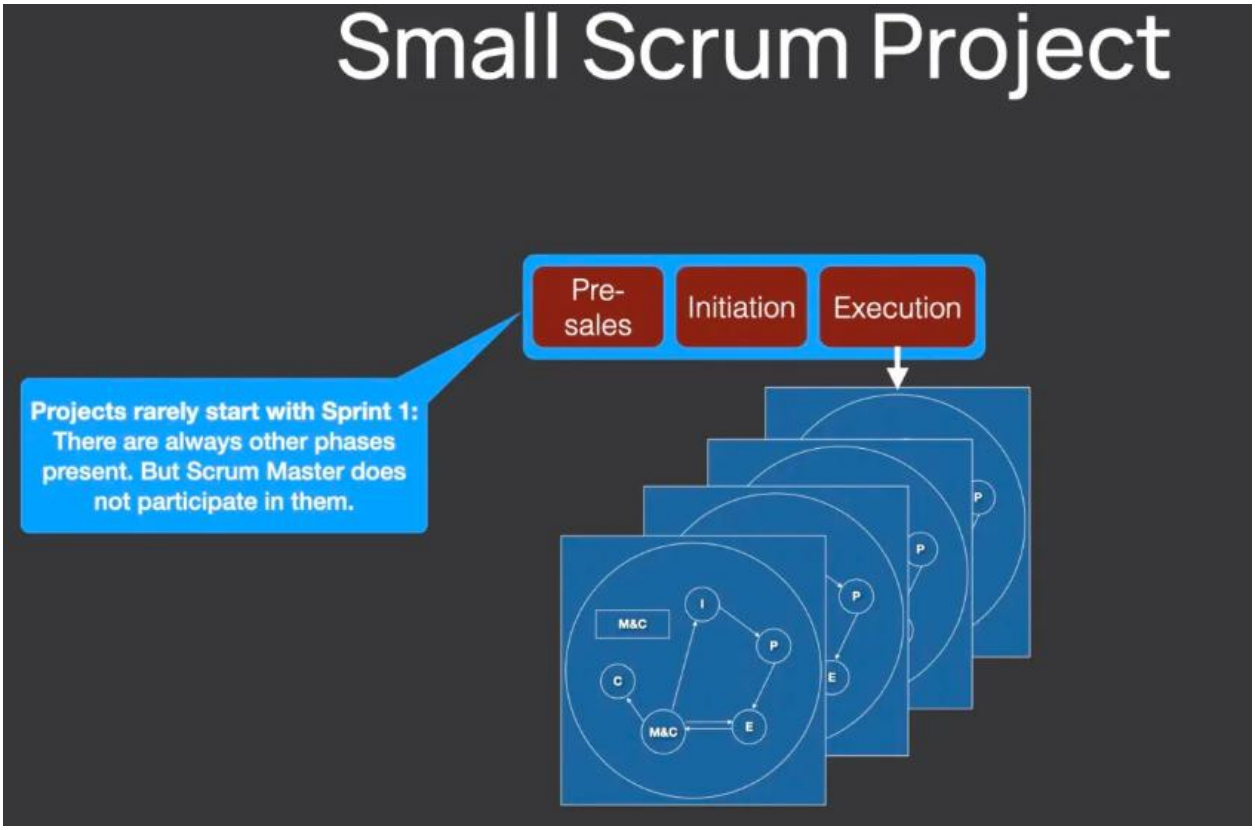


Рис. 1.4. Фази життєвого циклу розробки програмного забезпечення малого проекту

Невеликі проекти важче розділити на фази, оскільки вони перетинаються з процесом управління проектом.



Для гнучких проектів все ще є місце для етапів, на яких слід використовувати підходи, керовані планом.

Кожна фаза, якщо вона достатньо велика, може проходити через усі ці групи процесів.

Однак, якщо проект невеликий, групи процесів можуть збігатися з фазами.

Або технологічна група охоплюватиме декілька фаз одночасно.

Подібним чином у невеликих проектах буде лише одна фаза.

Отже, навіщо вам розуміти SDLC?

Розуміння життєвого циклу розробки програмного забезпечення робить процес управління проектом ефективним.

Але тут є справжня проблема:

З такою кількістю різних структур і підходів до управління проектами життєві цикли заплутані.

Тому тут ви отримаєте більше ясності щодо цієї теми, а нижче ви знайдете приклад SDLC.

Почнемо з визначення.

Визначення SDLC в управлінні проектами

Коли я загуглив «визначення життєвого циклу розробки програмного забезпечення», я знайшов різні варіанти.

Деякі кажуть, що це каркас. Деякі кажуть, що це процес або підхід.

Загалом, це життєвий цикл:

«Життєвий цикл проекту — це серія фаз, які проходить проект від його початку до завершення. Фаза проекту – це сукупність логічно пов’язаних проектних дій, кульмінацією яких є виконання одного або кількох результатів».

– Посібник PMBOK®

Отже, це лише кілька етапів, які вам потрібно пройти.

На цьому рівні ми не говоримо про те, як організувати фази та працювати всередині них. Це стосується підходу до управління проектами.

Навіщо потрібен життєвий цикл проекту?

Я хочу відзначити чотири основні переваги організації проекту на етапи:

1. Зробити проект керованим.

2. Відокремити роботу, яка є окремою та не пов’язаною з іншими фазами природи.

3. Зробити проект більш передбачуваним.

4. Створити контрольні точки проекту.

Отже, якщо ви хочете отримати додаткові відомості, у мене є окрема стаття тут:

3 корисні застосування знань про життєвий цикл проекту

Загальні етапи розробки програмного забезпечення

Ось кілька типових етапів розробки програмного забезпечення:

1. Ініціація проекту

2. Розробка концепції

3. Планування

4. Визначення вимог

5. Дизайн

6. Розвиток

7. Інтеграція

8. Тестування

9. Розгортання

10. Технічне обслуговування

11. Передача, закриття, підтримка

І ось хитрість:

Кожна організація розробляє життєвий цикл розробки програмного забезпечення, який відповідає її потребам.

Тому ви можете знайти різні назви в різних компаніях.

Фази SDLC проекту та процес керування проектом

Тут невелика плутанина.

Розумієте, процес управління проектом описується п’ятьма групами процесів:

1. Ініціація

2. Планування

3. Виконання

4. Моніторинг і контролінг

5. Закриття

Дійте так, ніби у вас є всі фази SDLC

Деякі з вас можуть подумати, що у вас немає всіх цих фаз.

Тому складається враження, що у вас лише одна відмінна фаза.

Туди вписується весь проект.

Або ви можете працювати в гнучкому середовищі, і вам здається, що ваш проект починається зі Sprint 1.

Однак ось що я пропоную:

Спробуйте уявити, що у вас є всі ці фази.

І кожна фаза має свої критерії виходу. Наприклад, щось, що вам потрібно створити або досягти, щоб рухатися вперед.

Більшість етапів – ви можете негайно закрити, оскільки у вас є попередньо визначені процеси та інструменти.

Деякі фази будуть накладатися одна на одну, і ви проходитимете їх у ітераціях.

І будуть фази, де ви не берете участь.

Але вони існують.

Отже, люди там впливають на ваш проект.

Життєвий цикл розробки програмного забезпечення та підходи до управління проектами

Життєвий цикл розробки програмного забезпечення в управлінні проектами — це структура проекту.

SDLC НЕ є фреймворком.

SDLC не дорівнює Scrum або Kanban.

Підхід до управління проектом може відрізнятися на різних етапах життєвого циклу.

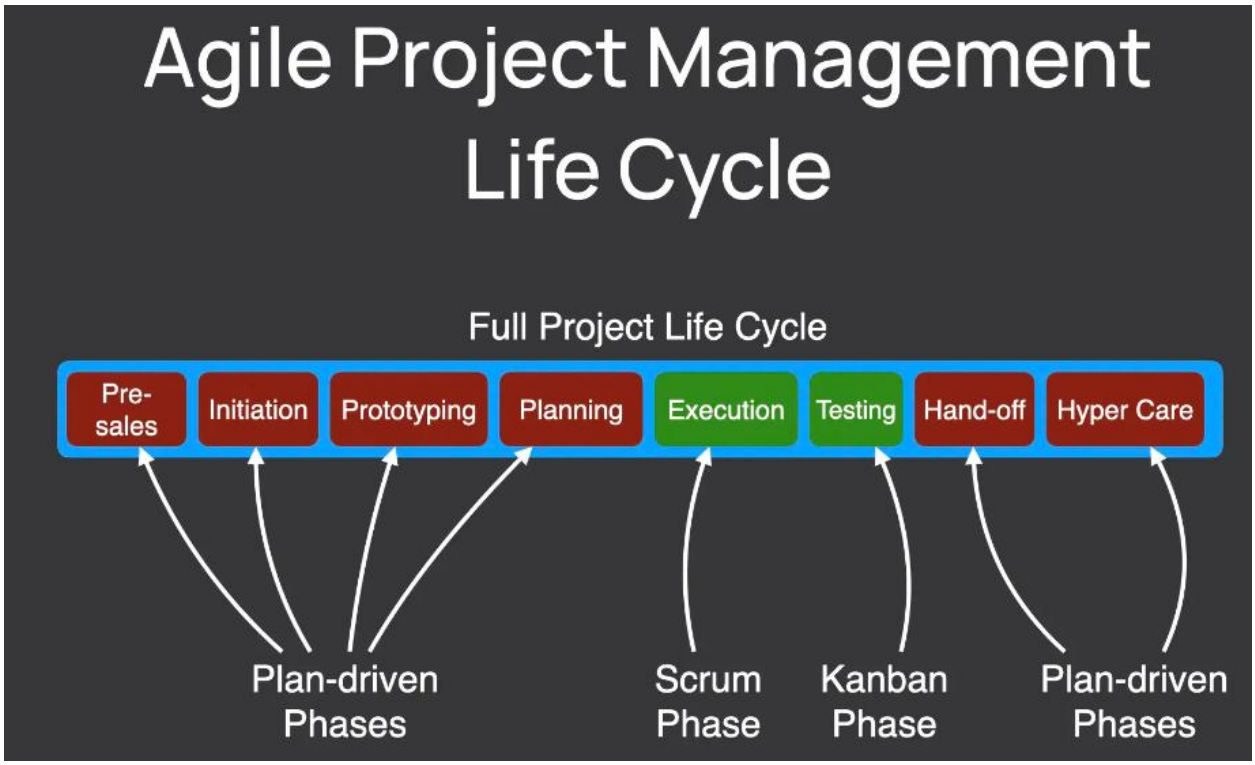


Рис. 1.5. Фреймворки та підходи на різних етапах ЖЦ проекту

1.2. Роль фази discovery

На початку будь-якого нового проекту аналітики часто можуть опинитися в ситуації, коли вони навіть не знають, з чого почати, маючи більше питань, ніж відповідей. Щоб успішно перейти від цього неоднозначного етапу до роботи над рішенням, аналітики BI проводять discovery проекту.

Discovery — це повторюваний процес, який зосереджується на зменшенні невизначеності навколо проблеми чи ідеї, щоб гарантувати, що правильний продукт буде створено для потрібної аудиторії. Таким чином, Discovery є одним із компонентів більш комплексного проекту або плану програми. discovery має цілісно охоплювати всі аспекти аналітики, пов’язані з проблемою чи ідеєю. Цього можна досягти, побудувавши «стек виявлення» з такими компонентами, як показано на слайді. Ця діаграма показує, що нам потрібно виконати на кожному етапі та які результати слід підготувати.

Загалом усі дії аналітиків можна згрупувати в п’ять етапів. Діаграма нижче показує, що потрібно виконати на кожному етапі та які результати слід підготувати.

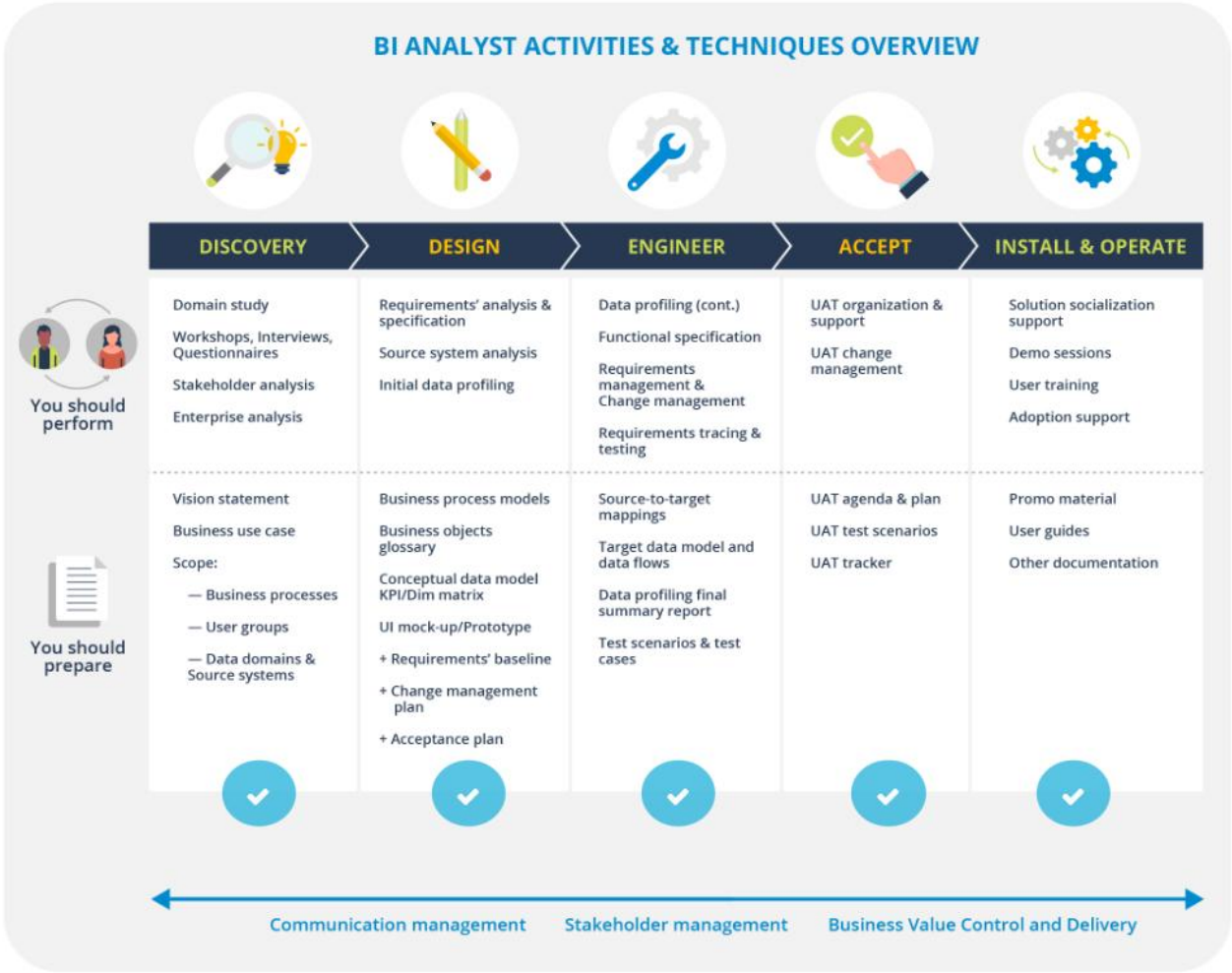


Рис. 1.7. Задачі та техніки аналітиків

Discovery

Початкове дослідження зазвичай відбувається перед початком розробки проекту чи продукту. Він зосереджений на визначенні зацікавлених сторін, їхніх проблемах, потребах і вимогах і дозволяє глибоко зрозуміти цілі, масштаби та обмеження проекту.

Дизайн (проєктування)

На цьому етапі ви повинні поставити собі такі запитання: звідки надходять дані? Які джерела даних? Чи потрібно буде використовувати неструктуровані джерела даних? Які дані, необхідні для KPI, наразі недоступні

та їх потрібно отримати? Відповіді дадуть вам чітке уявлення про цільовий інтерфейс користувача та модель даних, що стоїть за ним. Потім ви створюєте макет із віджетами та візуалізаціями даних, які ви захочете застосувати, щоб найкраще представити дані та KPI.

Інжинірінг

На даному етапі аналітики документують функціональні вимоги, цільову модель даних і створює відображення джерело-ціль на основі отриманих і проаналізованих на попередніх етапах. Ця фаза присвячена розробці, і команда розробників повинна знати, що робити

Тестування прийнятності користувачем (UAT, User acceptance testing, определить степень удобства, простоты и интуитивности интерфейса, а также степень удовлетворенности пользователя)

Є ключовим процесом у будь-якому бізнес-проекті, оскільки це етап, на якому кінцеві користувачі бачать, яку інформацію або користь вони можуть отримати від рішення. Зауважте, що вимірювання, представлене користувачеві, має бути значущим і технічно правильним.

Install&Operate

Коли продукт готовий, робота аналітика BI ще не завершена. Вам потрібно допомогти компанії клієнта з впровадженням продукту шляхом проведення демонстраційних сесій, тренінгів, навчальних матеріалів тощо.

Як бачите, перш ніж поставити завдання для команди розробників на етапі розробки, необхідно:

• визначити тип проекту та перелік заходів, рівень вимог до збору та глибину аналізу

• створити план discovery

• проводити відкриття, щоб виявити вимоги та/або бізнес-потреби

• виконувати аналіз вимог, включаючи встановлення пріоритетів та документування.

1.3. Аналіз існуючих програмних засобів управління проектами

Існує велика кількість інструментів для управління програмним проектом, які можна використовувати на різних етапах розробки програмного забезпечення [4]. Деякі з них наведено нижче:

Trello

Trello надає вам можливість спільно працювати над одним проектом, і користувач може залишатися на зв'язку незалежно від пристрою, яким він користується. Дошка Trello складається з картки, яка може представляти певне завдання або навіть набір процесів. Це дозволяє Trello бути чудовим інструментом управління для тих проектів, які використовують у своїй роботі методології scrum або kanban. Рис. 1.7. показує інтерфейс Trello.

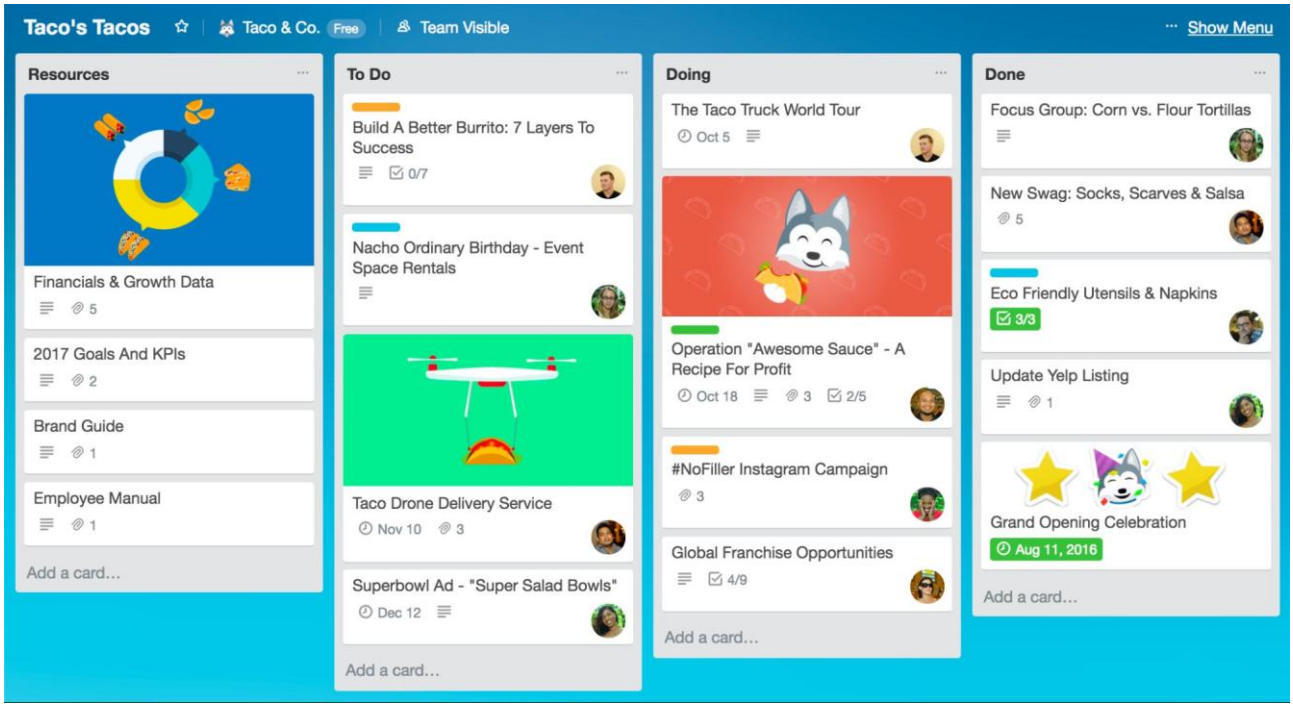


Рис. 1.7. Інтерфейс Trello

Переваги:

• Легке виконання завдань;

• Простий пошук завдання в проекті;

• Можливість розділити роботу на окремі завдання за допомогою «карток»;

• Можливість призначення завдання конкретному користувачеві;

• Можливість бути приписаним до кількох проектів;

• Можливість налаштувати багатопроектне середовище.

Недоліки:

• З першого погляду нелегко зрозуміти, як видалити завдання;

• Календар некоректно працює при видаленні «карт»;

• Відсутня можливість перегляду розкладу завдань для окремого користувача.

Jira

Jira — це потужний інструмент для управління проектами, який представляє собою купу підінструментів, які створюють цілу екосистему для управління проектами. Jira дозволяє групувати проблеми за набором критеріїв. Це може бути група за бізнес-підрозділом, командою або продуктом. Він також може підтримувати декілька проектів одночасно та керувати ними окремо або всією системою. Кожна проблема представлена карткою завдання з унікальним ідентифікатором, який складається з ключа проблеми та номера проблеми. Інтерфейс Jira показано на рис. 1.8.

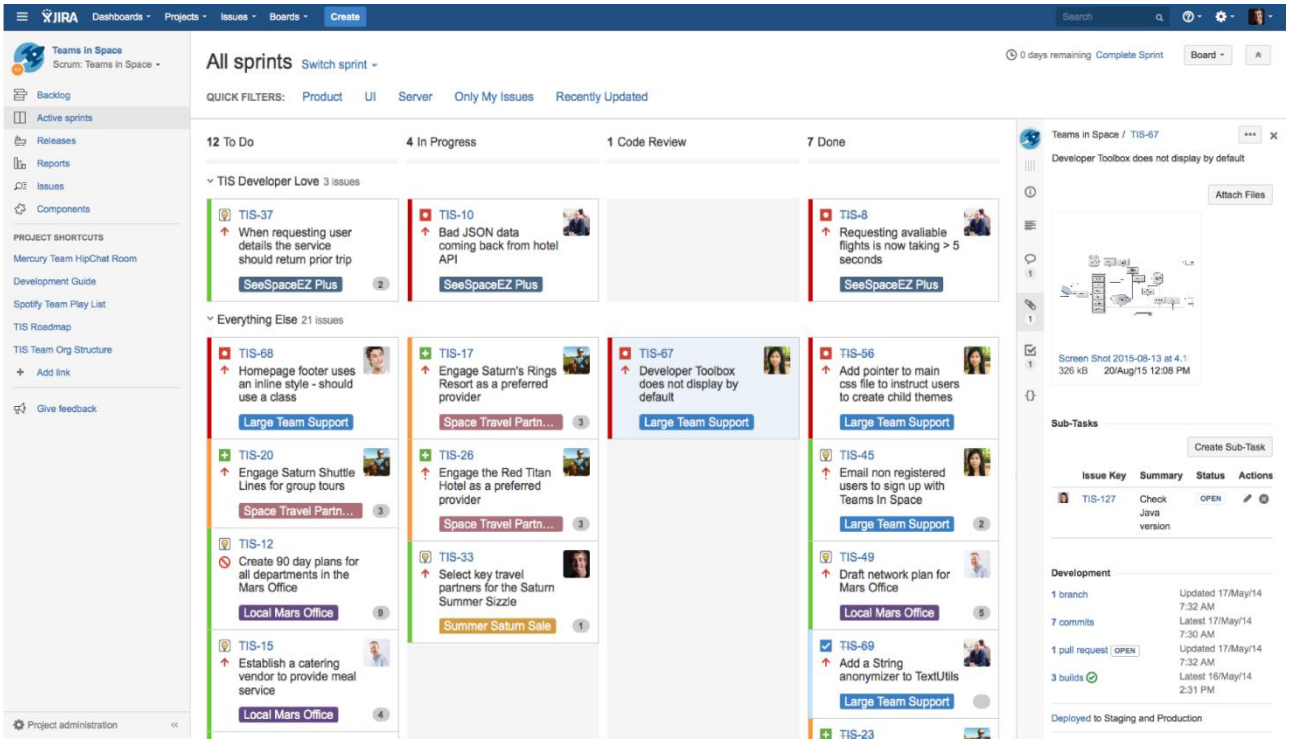


Рис. 1.8. Інтерфейс Jira

Переваги:

• Легке включення в робочий процес;

• Забезпечити шляхи співпраці;

• Отримайте можливість побудувати спринт-звіт і діаграму згорання;

• Надати купу фільтрів для організації робочого простору;

• Охопіть набір гаджетів для перевірки стану.

Недоліки:

• Проблеми з роботою при відкритті;

• Можна надати історію спаду поля пошуку;

• Складне середовище налаштування.

1.2.3. Microsoft Project

Microsoft Project — це програмний продукт для керування проектами, розроблений і проданий Microsoft. Він розроблений, щоб допомогти керівнику проекту розробити графік, призначити ресурси для завдань, відстежувати прогрес, керувати бюджетом і аналізувати навантаження. Інтерфейс Microsoft Project, показаний на рисунку 1.9.

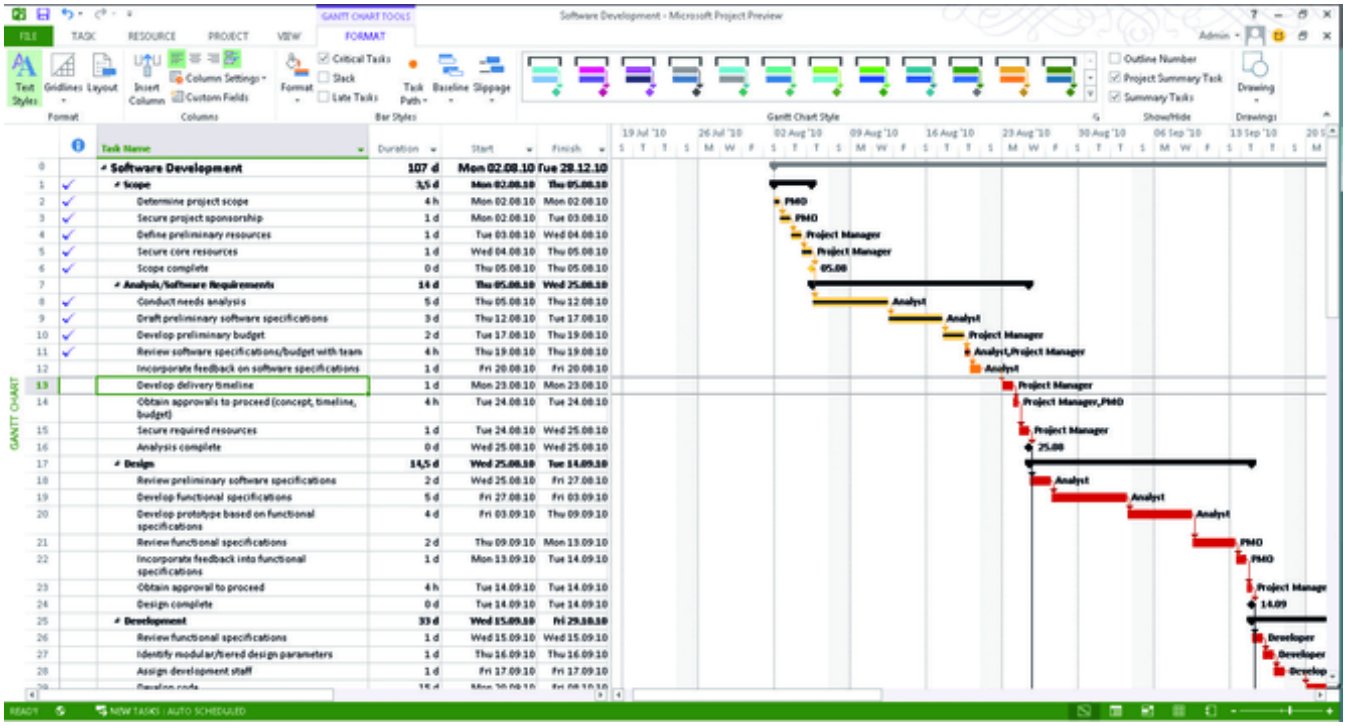


Рис. 1.9. Інтерфейс Microsoft Project

Microsoft Project має наступні плюси:

• Він гнучкий. Він містить багато функцій, які допомагають правильно налаштувати проект;

• Ви можете інтегрувати інші інструменти Microsoft. Наприклад, Microsoft пропонує різноманітні інструменти, такі як Word, Power Point і Visio, які можна інтегрувати в одну систему;

• Пропонує багато каналів для спілкування. Це означає, що ви матимете легкий доступ до таких інструментів, як Outlook, Yammer і Skype.

Microsoft Project має наступні мінуси:

• Занадто просунутий для початківців. Велика кількість різноманітних функцій також означає, що вам потрібно отримати певний досвід, перш ніж почати використовувати Microsoft Project;

• Не підходить для співпраці та обміну даними. Наприклад, Microsoft Project є автономною програмою, це означає, що ви не можете використовувати її разом з іншими без налаштування системи фахівцями;

• Дорого. Microsoft Project є окремим продуктом Microsoft і коштує він досить дорого.

Платформа eXo

Платформа eXo — це програмне забезпечення з відкритим кодом, яке здебільшого зосереджено на соціальній співпраці. Це підходить для вашого проекту, якщо він вимагає тісної співпраці між членами всередині команди та зовнішніми учасниками, такими як власник продукту та клієнт. Також дозволяє підключатися до соціальних мереж, таких як LinkedIn, Google+ і Facebook. Інтерфейс програмного забезпечення показано на рисунку 1.10.

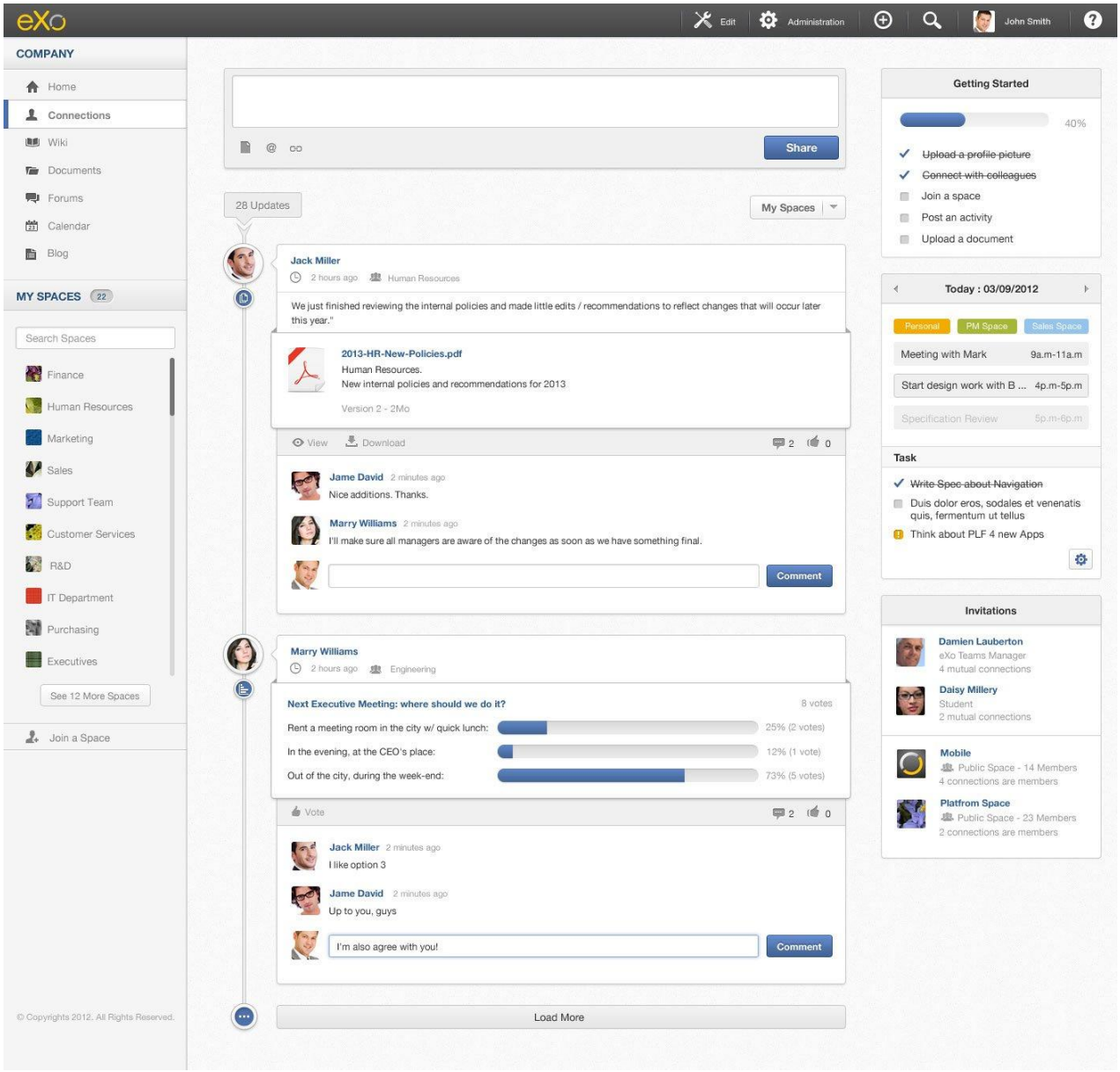


Fig. 1.10. eXo Platform interface

Основними недоліками цього програмного забезпечення є те, що ви не можете організувати відстеження завдань і те, що за це потрібно платити.

Висновки по розділу

Управління pre-sales етапом розробки ПЗ є складним і важливим фактором для отримання якісного продукту. Існує велика кількість методологій, які допомагають організувати роботу та дозволяють краще спілкуватися всередині проекту. Така ж ситуація з інструментами для pre-sales етапом розробки ПЗ, але незважаючи на те, що всі вони використовуються в

проекті, тому що немає двох однакових проектів і для вирішення конкретних завдань завжди потрібні конкретні інструменти. Крім того, звичайним є використання набору інструментів, які доповнюють один одного. Це означає, що досі немає універсального інструменту, який міг би задовольнити всі потреби та забезпечити найбільш ефективний розподіл роботи. Таким чином, це робить pre-sales етапом розробки ПЗ, а також управління проектами та інструменти розробки для них актуальними сьогодні.

РОЗДІЛ 2
ВИЗНАЧЕННЯ ОСНОВНИХ ВИМОГ

2.1. Встановлення вимог вцілому

• Планування: цей початковий етап передбачає окреслення обсягу та цілей проекту, закладаючи основу для всього процесу розробки.

• Аналіз: Етап аналізу глибоко заглиблюється в розуміння цільової аудиторії, виявлення проблемних точок і пропонування рішень. Йдеться про перевірку бізнес-ідеї та її потенційного ринкового попиту.

• Дизайн: після того як вимоги зібрано, команда дизайнерів UX/UI створює інформаційну архітектуру, створюючи каркаси та прототипи, які візуалізують функціональність і потік майбутнього продукту.

• Розробка: тут продукт набуває форми, коли розробники пишуть код відповідно до визначених вимог і інструкцій щодо дизайну. Розробники бекенда та інтерфейсу спільно створюють компоненти на стороні сервера та клієнта відповідно.

• Тестування програмного забезпечення: інженери із забезпечення якості проводять ручні та автоматизовані тести, щоб переконатися, що функції продукту працюють належним чином, гарантуючи, що він відповідає всім технічним вимогам.

• Розгортання: програмне забезпечення підготовлене до випуску у виробництво, а інженери із забезпечення якості забезпечують плавний процес розгортання, перевіряючи, чи рішення відповідає заданим вимогам.

• Технічне обслуговування та підтримка після запуску: навіть після розгортання процес розробки програмного забезпечення продовжується, оскільки розробники адаптують продукт до нових потреб користувачів, надають оновлення та вирішують будь-які проблеми, про які повідомляється.

Бізнес-аналіз: pre-sales

Перш ніж зануритися у повноцінну розробку програмного забезпечення, дуже важливо підтвердити бізнес-ідею. Це передбачає звуження

цільової аудиторії, визначення її проблемних точок і оцінку попиту ринку на запропоноване вами рішення. Бізнес-аналіз включає два етапи: передпродажний і аналіз конкретного проекту. Під час попереднього продажу початкові вимоги до проєкту визначаються та перетворюються на історії користувачів, що дозволяє групі розробників надавати попередні оцінки.

Ключові ролі в команді веб-розробників:

• Бізнес-аналітик: Оцінка бізнес-ідеї та основних вимог для створення точних технічних специфікацій для команди розробників.

• Керівник групи розробки програмного забезпечення: відповідає за технічні рішення та комунікацію з керівниками проектів і бізнес-аналітиками, щоб команда розробників залишалася на шляху.

• Дизайнери UX/UI: створюйте інформаційну архітектуру, макети, каркаси та прототипи, що забезпечує ясність зовнішнього вигляду продукту.

• Back-End розробники: зосередьтеся на серверній частині продукту, обробці аналітики, баз даних, інтеграції та кешування.

• Front-End розробники: Забезпечте зручні для користувача інтерфейси та вирішуйте складні передові завдання.

• Інженери із забезпечення якості: проводять тестування, щоб перевірити функціональність програмного забезпечення та переконатися, що воно відповідає технічним вимогам.

• Керівники проектів: контролюють весь процес розробки, встановлюють терміни, керують обов’язками та підтримують зв’язок із клієнтами та зацікавленими сторонами.

Бізнес-аналіз проекту

Після того, як початкові вимоги зібрані, бізнес-аналітик створює каркаси та детальний документ вимог. Інженери із забезпечення якості співпрацюють із бізнес-аналітиками, щоб забезпечити повне розуміння функціональних можливостей програмного забезпечення. Цей спільний підхід мінімізує зміни та збої після того, як розробка вже почалася. Завершені вимоги

перетворюються на епопеї та історії користувачів, керуючи завданнями розробки в спринтах.

Документи, підготовлені під час бізнес-аналізу:

• Каркаси

• Документ вимог

• Специфікації вимог до програмного забезпечення

• Епоси та історії користувачів

• Плани тестування та документація

Фаза проектування

Фаза проектування починається з аналізу вимог проекту для створення інформаційної архітектури. Каркаси, представлення програмного забезпечення з низькою точністю, створені для окреслення функціональності. Ці каркаси перетворюються на інтерактивні прототипи, пропонуючи детальне розуміння дизайну та взаємодії з користувачем. Після схвалення дизайну інтерфейсу користувача команда розробників бере на себе роботу.

Побудова основної функціональності

На цьому етапі розробники пишуть код на основі специфікацій і вимог, дотримуючись інструкцій із кодування та найкращих практик. Вибір технологічного стеку має вирішальне значення, оскільки він впливає на масштабованість і майбутні оновлення. Зрілі технології, такі як Ruby, Python або PHP, надають перевагу для забезпечення стабільності та мінімізації вразливостей.

Побудова основної функціональності

На цьому етапі розробники пишуть код на основі специфікацій і вимог, дотримуючись інструкцій із кодування та найкращих практик. Вибір технологічного стеку має вирішальне значення, оскільки він впливає на масштабованість і майбутні оновлення. Зрілі технології, такі як Ruby, Python або PHP, надають перевагу для забезпечення стабільності та мінімізації вразливостей.

Перевірка якості

Інженери із забезпечення якості завчасно приєднуються до проекту, щоб співпрацювати з бізнес-аналітиками та визначати вимоги до проекту. Вони перевіряють архітектуру програми, щоб зрозуміти, як взаємодіють різні модулі та бази даних. Готується документація щодо тестування, включаючи стратегію тестування, плани та контрольні списки. Ручні та автоматичні тести проводяться, щоб виявити помилки та переконатися, що програмне забезпечення працює правильно.

Запуск проекту

Розгортання є останнім етапом, який гарантує, що програмне забезпечення правильно розгорнуто та відповідає всім визначеним вимогам. Інженери із забезпечення якості виконують остаточні перевірки перед тим, як продукт запрацює. Примітки до випуску створено, щоб інформувати користувачів про технічні деталі та оновлення.

Технічне обслуговування

Життєвий цикл розробки програмного забезпечення продовжується після запуску, оскільки розробники вирішують проблеми, про які повідомляють, впроваджують оновлення, оптимізують продуктивність і адаптуються до мінливих потреб бізнесу. Можна запропонувати чотири типи технічного обслуговування: коригувальний, профілактичний, досконалий і адаптивний.

Популярні моделі розробки програмного забезпечення

Кілька моделей розробки програмного забезпечення відповідають потребам різних проектів. Ось деякі з найпопулярніших:

• Гнучкість: гнучкий підхід, зосереджений на ітераційній розробці, співпраці та швидкій адаптації до змін.

• Scrum: підмножина Agile з короткими циклами розробки (спринти) і фокусом на пріоритетних завданнях.

• Kanban: Візуалізація робочого процесу за допомогою панелей завдань, ідеально підходить для проектів із різноманітними завданнями та пріоритетами.

• Водоспад: лінійний підхід із послідовними етапами, підходить для проектів зі стабільними вимогами.

• V-подібна модель: наголошується на валідації та верифікації, забезпечуючи сильний акцент на якості.

• Спіральна модель: поєднує ітераційну розробку з аналізом ризиків, що робить її придатною для складних проектів.

• Ітеративна інкрементальна модель: ділить проект на ітерації, надаючи робочі функції на ранній стадії.

• RAD (швидка розробка додатків): надає пріоритет швидкому створенню прототипів і швидкому постачанню продукту.

Кожна модель має унікальні переваги та недоліки, що робить їх придатними для конкретних вимог проекту. Вибір залежить від таких факторів, як розмір проекту, складність і потреби в гнучкості.

Підсумовуючи, розуміння процесу розробки програмного забезпечення та вибір правильної моделі є критично важливими кроками на шляху до досягнення вашої мети – надання високоякісного програмного рішення, яке відповідає вимогам ринку. Avena готова надати спеціалізовані групи розробників програмного забезпечення для інноваційних і революційних проектів, проводячи вас на кожному етапі вашого шляху розробки програмного забезпечення.

Усі проекти даних можна розділити на три основні типи проектів залежно від рівня невизначеності та складності ландшафту (від низького до високого).

Сценарій 1: Standard data delivery

У цьому випадку бізнес-потреба проста і зрозуміла.

Standard data delivery вимагає точних бізнес-потреб і зрозумілих, послідовних вимог користувача. Залишилося задокументувати ці вимоги та впровадити необхідні зміни.

Зазвичай такі проекти є короткостроковими.

Сценарій 2: бізнес-аналіз

У цьому випадку бізнес-потреба відома, але складніша порівняно зі standard data delivery.

Рівень невизначеності середній: зміни та потреби бізнесу чітко визначені та забезпечують передбачувані результати.

Бізнес-аналіз — це процес дослідження, який відбувається після завершення аналізу підприємства. Його слід виконувати лише після визначення ділової потреби. На цьому етапі BI Analyst взаємодіє із зацікавленими сторонами, щоб:

• Виявлення, аналіз і управління вимогами

• Розробити та визначити рішення для негайної бізнес-проблеми.

Рішення може бути у формі продукту, системи, процесу або частини програмного забезпечення. Крім того, бізнес-аналіз може передбачати тестування та документування запропонованого рішення.

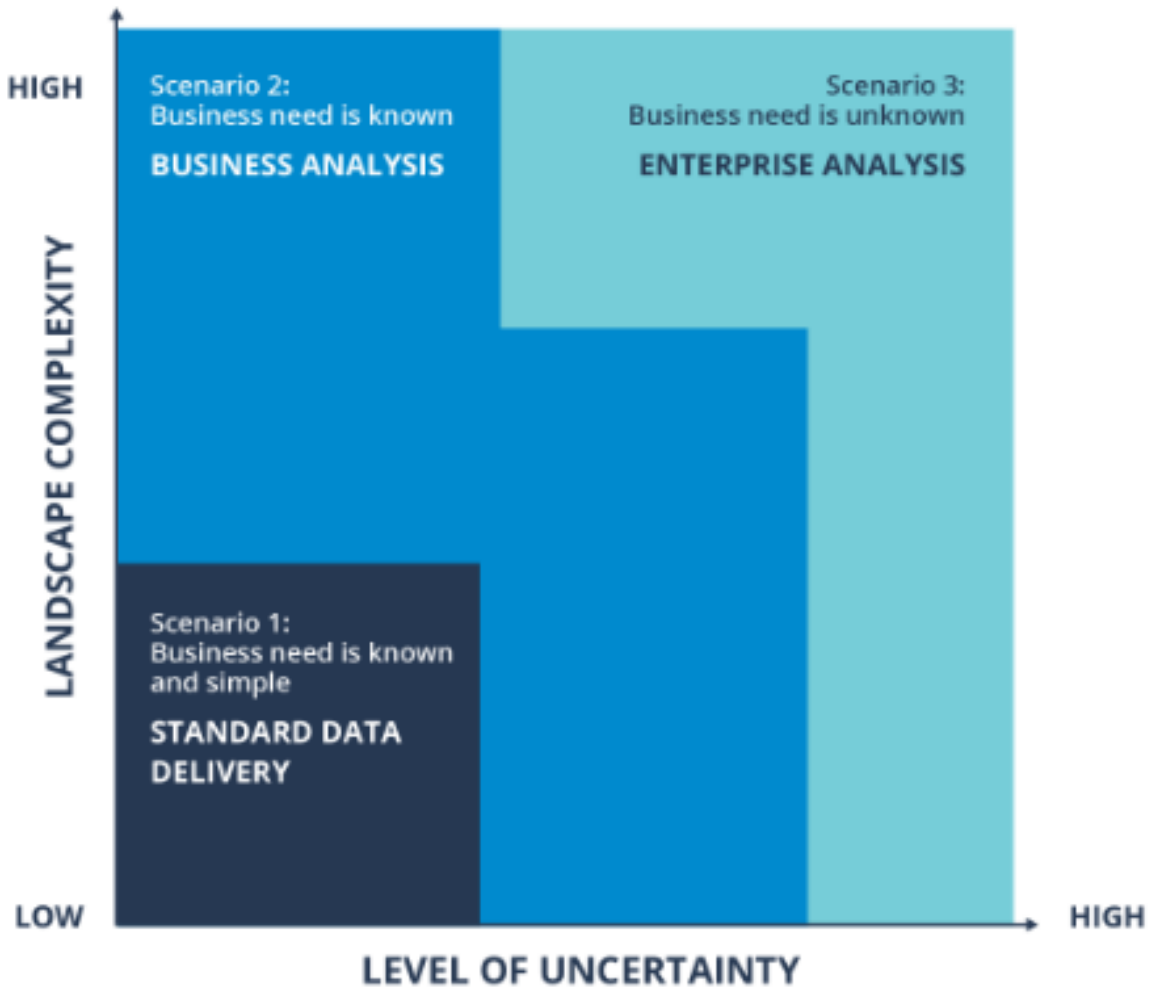


Рис. 2.3. Типи проектів

Сценарій 3: Аналіз підприємства

Тут бізнес-потреба невідома, що спричиняє високу невизначеність і низьку керованість, високу складність або високу причинність.

Якщо і бізнес-потреби, і складність проекту неясні, почніть з Enterprise Analysis.

Аналіз підприємства – це діяльність на рівні керівника/підприємства, мета якої:

• Вивчити довгострокові потреби бізнесу

• Визначте стратегічний напрямок

• Вивчіть можливі відповіді та рішення, визначте та оцініть їх на предмет масштабу, ефективності, здійсненності та ризику.

Enterprise Analysis допомагає визначити та розставити пріоритети щодо розробки та впровадження на багаторічний горизонт.

Аналіз підприємства та бізнес-аналіз вимагають процесу discovery, щоб виявити, уточнити та підтвердити вимоги бізнесу та користувачів.

У таблиці нижче ви можете побачити порівняльний опис трьох видів аналізу:

Зауважте, що аналітики зазвичай працюють разом із консультантом та/або провідним аналітиком у проекті, де потрібен аналіз підприємства.

Давайте детальніше розглянемо Value Creation Flow (потік створення цінності), який узагальнює завдання та дії, за які відповідає аналітик на етапах виявлення та проектування.

Як бачите, первинні результати етапу discovery є основоположними для всіх наступних дій. Обробляючи вимоги, зібрані на етапі discovery, аналітик BI може підготувати чернетки, які пізніше будуть використані в процесі проектування, наприклад концептуальні моделі даних, глосарії тощо.

Недооцінка важливості фази discovery на початку проекту може призвести до розповзання scope, перевитрати бюджету та порушення термінів. Відкриття слід проводити як частину аналізу як підприємства, так і бізнесу, оскільки воно відіграє важливу роль у створенні кінцевої цінності для бізнесу.

Discovery дозволяє охопити складний scope, що охоплює кілька бізнес-функцій, узгодити запити кількох бізнес-груп і оцінити різні джерела даних і аналітичні програми за відносно короткий час.

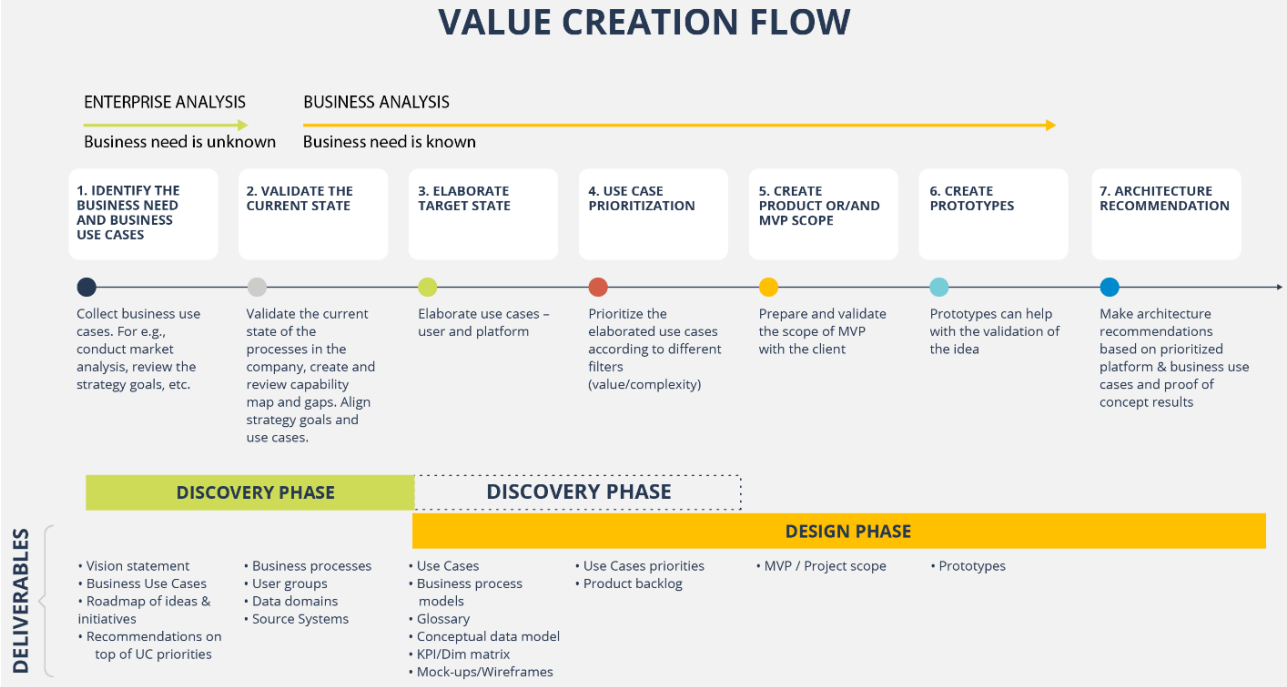


Рис. 2.4. Value Creation Flow

2.2. Ціль розробки та область використання

Метою цієї розробки програмного забезпечення є розробка прототипу управління завданнями в проектах розробки програмного забезпечення

Цільовою групою цього прототипу системи є аналітики, розробники програмного забезпечення та експерти з управління.

Сфера використання

Проекти є організаційними рамками для безперервного, систематичного отримання ідей, розуміння та результатів на основі методологічних принципів. Засоби управління проектами зараз використовуються для вирішення складних, комплексних і простих завдань. Управління проектами в компаніях полягає в реалізації різноманітних завдань і вирішенні проблем як всередині компанії, так і між різними компаніями. При цьому не повинно бути досягнуто негативного ефекту на виробничо-організаційних аспектах діяльності підприємства. Сфера

застосування методів управління проектами дуже різноманітна, хоча інструмент управління проектами був розроблений в США для організації суднобудівних і авіабудівних компаній. В даний час управління проектами стає все більш популярним і використовується в різних компаніях, незалежно від розміру, виду діяльності, поставлених цілей і завдань.

.

2.3. Специфікація вимог до програмного забезпечення

Створення специфікації вимог до програмного забезпечення (SRC) або специфікації вимог до програмного забезпечення є важливою частиною розробки, так як добре створені вимоги гарантують, що продукт, створений на основі цих вимог, матиме хорошу якість і буде отриманий клієнтом [5]. SRC складається з двох основних частин:

• Функціональні вимоги (FR)

• Нефункціональні вимоги (NFR)

Поряд з цими вимогами можуть бути створені додаткові вимоги до більш повного розуміння аспектів розвитку та управління.

Крім вимог, SRC може мати купу діаграм випадків використання, які дозволяють описати, як користувач буде взаємодіяти з програмним забезпеченням.

Функціональні вимоги показують, які дії та операції має виконувати розроблене програмне забезпечення [6].

FR повинна дати відповідь на наступні питання:

• Які дані вводяться;

• Яку операцію слід виконати;

• Який результат має бути відображено;

• Хто може вводити дані.

Нефункціональні вимоги або NFR показують нам вимоги, які не виконують жодних дій у системі, але, тим не менш, важливі, наприклад,

визначають якості майбутнього продукту та описують, як система, як очікується, працюватиме.

Функціональні вимоги (FR)

Таблиця 2.1.

Функціональні вимоги (FR)

№	Опис
FR 1	Веб-додаток повинен мати заголовок
FR 1.1	Веб-додаток повинен мати блок з навігацією в ньому
FR 1.2	Веб-додаток повинен мати підменю
FR 2	Веб-додаток повинен мати слайдер
FR 3	Веб-додаток повинен мати блок контенту
FR 3.1	Блок вмісту повинен мати прокрутку, яка надходить до необхідного місця
FR 4	Веб-додаток повинен мати галерею картинок
FR 5	Веб-додаток повинен мати контактний розділ
FR 6	Веб-додаток повинен мати дошку для завдань
FR 6.1	Веб-додаток має надавати можливість додавати колонки на дошку
FR 6.2	Веб-додаток має надавати можливість видаляти колонки з дошки
FR 6.3	Веб-додаток має надавати можливість додавати завдання
FR 6.4	Веб-додаток має надавати можливість видаляти завдання
FR 6.5	Веб-додаток має забезпечувати можливість переміщення завдання з однієї колонки в іншу
FR 6.6	Веб-додаток має надавати можливість сортувати завдання
FR 6.7	Веб-додаток має надавати можливість фільтрації завдань
FR 6.8	Веб-додаток має надавати можливість пошуку завдань
FR 7	Веб-додаток має надавати можливість авторизуватися
FR 7.1	Веб-додаток має надавати можливість переглядати поставлені завдання
FR 7.2	Веб-додаток має надавати можливість переглядати розклад
FR 8	Веб-додаток має надавати можливість виходу з системи

Нефункціональні вимоги (NFR)

2.3.1. Вміст платформи

Веб-додаток має коректно відображатися у всіх сучасних браузерах і займати для нього всю ширину.

2.3.2. Встановлюваність

Система повинна працювати в браузері і не потребує встановлення**.**

2.3.3. Доступність

Система повинна бути доступна через Інтернет і цілодобово.

2.3.4. Надійність

Система повинна реагувати на будь-яку помилку, що виникла в ній, і виконувати відповідні дії.

Якщо сталася помилка через неправильне введення, користувач повинен бути проінформований і мати можливість ввести інше значення.

Якщо сталася помилка через збій усередині системи, її слід спробувати виправити, а якщо ні, повідомити користувача, відкривши відповідне вікно.

2.3.5. Довговічність

Система повинна працювати коректно і забезпечувати захист даних користувача до моменту його звільнення.

2.3.6. Масштабованість

Система повинна правильно відображатися на всіх можливих розмірах екрана та забезпечувати спеціальні версії для трьох основних платформ: телефону, планшета та комп’ютера.

2.3.7. Юзабіліті

Система повинна бути:

• Легко почати використовувати;

• Інтуїтивно зрозумілою;

• Зручною у використанні.

2.3.8. Безпека

Система повинна забезпечувати захист зібраних персональних даних і дозволяти ділитися лише псевдонімом користувача.

2.3.9. Продуктивність

Відгук інтерфейсу не повинен перевищувати 500 мс.

2.3.10. Розширюваність

Система має забезпечувати розширення лише розробником через повне оновлення системи.

2.3.11. Локалізація

Інтерфейс системи повинен використовувати українську та англійську мови.

Назву колонки та завдання можна ввести будь-якою мовою.

Взаємодія з користувачем

Взаємодія користувача з дошкою завдань описана на діаграмі варіантів використання (рис. 2.1.).

Рис. 2.6. Діаграма прецендентів

Інтерфейс програми повинен бути зручним і відповідати стандартам дизайну інтерфейсу.

Інтерфейс користувача складається з таких вікон:

- вікно авторизації;
- Реєстрація;
- Вікно вибору активних проектів або додавання нових;

— Вікно додавання/перегляду завдань для проекту.

Організація роботи з інтерфейсом програми має виглядати так:

- Запустити програму;
- Переглянути головне меню програми:

"Реєстрація"

«Дозволи»

«Створити новий проект»

«Створення завдань»;

У вікні «Реєстрація» користувач повинен ввести свої дані:

- Прізвище;
- Пошта;
- Пароль;
- Підтвердьте пароль

Особливості системи:

- Додаток дозволяє людині вводити кілька облікових записів з проектами
- Надати користувачам доступ до облікового запису
- Можливість додавати завдання для індивідуальних проектів.
- Додайте дошки для кожного проекту окремо
- Вступайте в багато проектів одночасно
- Системний код повинен відповідати стандартам кодування мови програмування JS.

Перспективи продукту

Ця система суттєво відрізняється від інших систем функціональною частиною та інтерфейсом в цілому. Система створена за методологією Kanban, тому ми не накладаємо жодних обмежень на користувачів

2.4. Технічні вимоги до платформи, на якій повинна працювати система

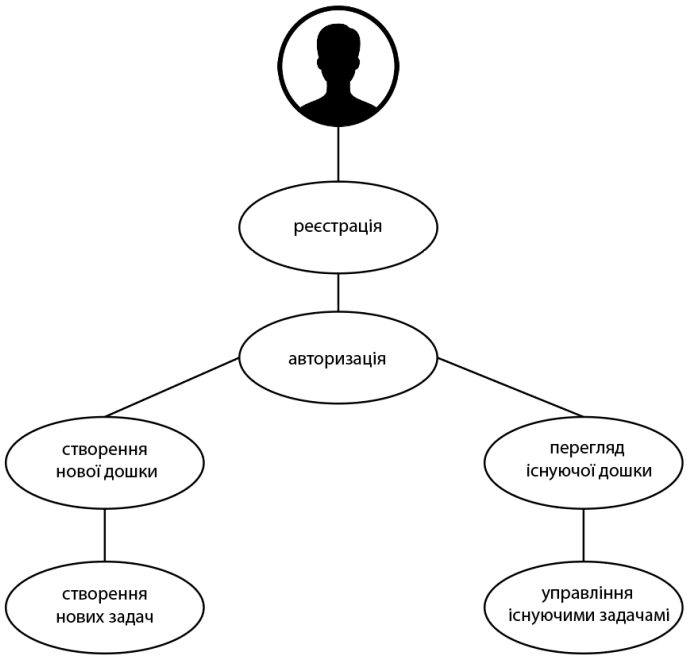


Рис. 2.8. Функціональні вимоги програмного засобу

Апаратні інтерфейси

Система працює на персональних комп'ютерах (ПК) з такими мінімальними вимогами:

- 512 Мб оперативної пам'яті;

- процесор Pentium 4 (або аналоги інших виробників) з тактовою частотою 2,0 ГГц і більше;

Програмні інтерфейси

- Операційна система: Windows, Mac OS;

- Microsoft .NET Framework 4;

- Вузол JS (npm)

- Код WebStorm/VS

2.5. Бізнес вимоги

Користувач системи повинен мати базові знання операційної системи Windows або Mac OS, а також знання користування браузером, оскільки це веб-додаток.

Досвідчений користувач повинен мати знання з управління проектами, оскільки він створює завдання та контролює терміни, які впливають на результат проекту.

Висновки по розділу

Даний розділ присвячено опису вимог до розроблюваного програмного засобу. Функціональні та нефункціональні вимоги дійсно важливі при розробці програмного забезпечення, оскільки вони дозволяють сформувати бачення майбутнього продукту навіть без його фактичної розробки. Ще один прибуток – це можливість передбачити ймовірні помилки, з’явитися та попередити їх. Різні види схем дозволяють отримати більш повне бачення продукту. Діаграма варіантів використання показує, яку функцію повинен створити розробник і який результат він повинен отримати. Потік даних показує, якими даними ми повинні мати можливість працювати та яким чином.

Варто зазначити, що даний програмний засіб не вимогливий до апаратної частини так як використовує браузер для запуску програми. Має весь необхідний функціонал для управління проектами для розробки програмного забезпечення і не тільки. Цю програму можна використовувати в повсякденному житті з огляду на дуже простий інтерфейс який дозволяє прописувати будь-які завдання і відстежувати їх виконання.

Відсутність додаткового функціоналу наприклад таймінг або календар що часто використовують в подібних програмах це дозволить користувачеві зосередиться на своїй основній роботі. Програма відповідає вимогам сучасних інтерфейсів і вимогам користувача.

РОЗДІЛ 3
АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ЗАСОБУ УПРАВЛІННЯ PRE-SALES ЕТАПОМ РОЗРОБКИ ПЗ

3.1 Користувацький інтерфейс(керівництво для користувача системи)

Інтерфейс користувача (UI) — це засіб зручної взаємодії користувача з інформаційною системою. Комплекс засобів обробки та відображення інформації, максимально адаптованих до зручності користувача; У графічних системах інтерфейс користувача втілюється в багатовіконному режимі, змінюється колір, розмір, видимість (прозорість, напівпрозорість, невидимість) вікон, їх положення, сортування елементів вікон, гнучкі налаштування як самих вікон, так і їх окремі елементи (файли, папки, ярлики, шрифти тощо), наявність багатокористувацьких налаштувань.

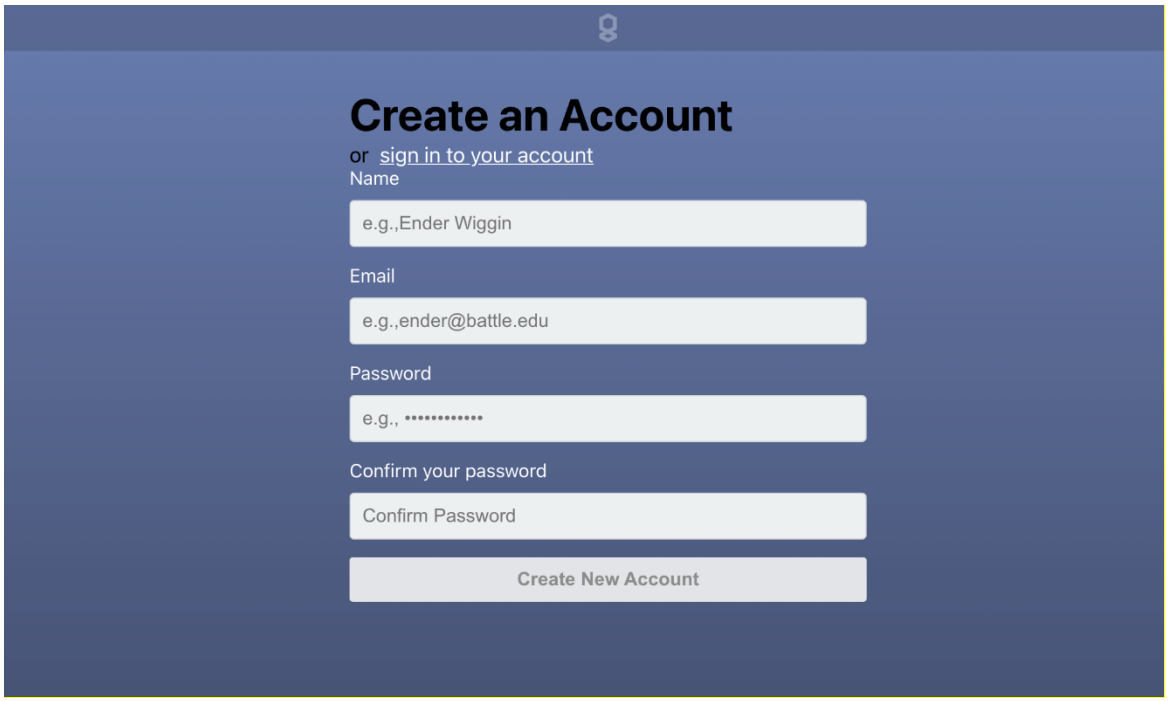


Рис. 3.1 Реєстрація користувача в системі

У вікні " Авторизації " користувач повинен ввести дані аккаунту

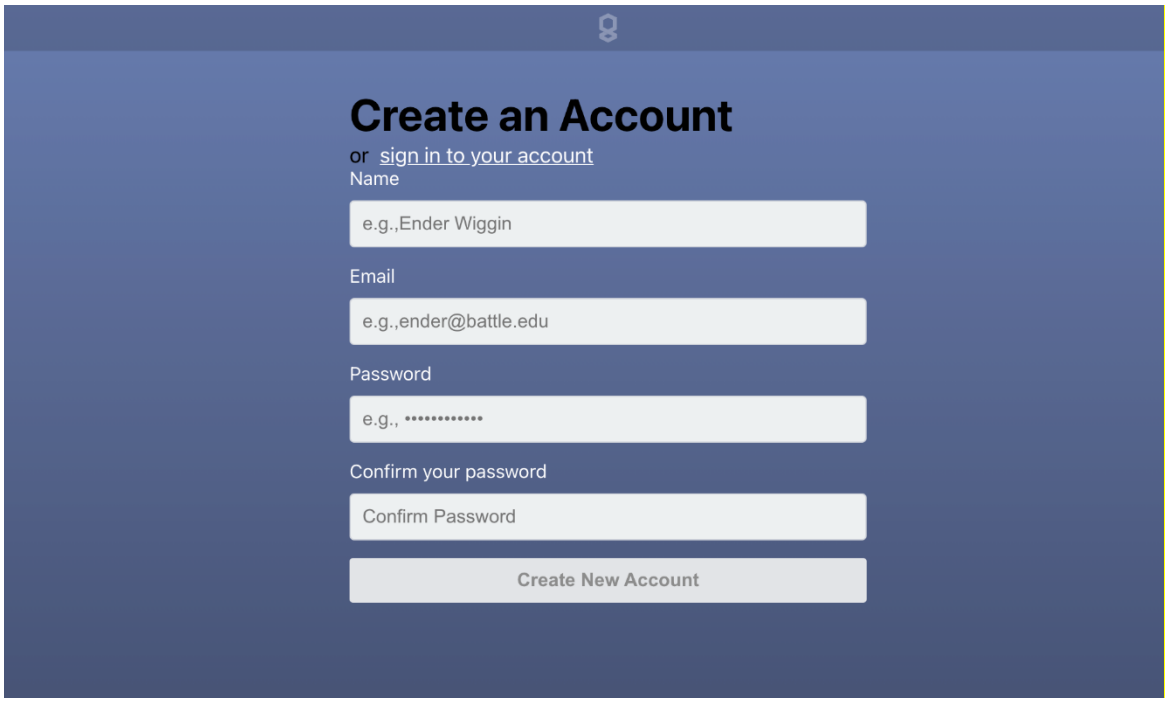


Рис.3.2 Авторизації користувача в системі

Далі ми потрапляємо до вікна створення проекту

Для того щоб створити новий проект потрібно натиснути кнопку «Add a board», або вибрати вже існуючий.

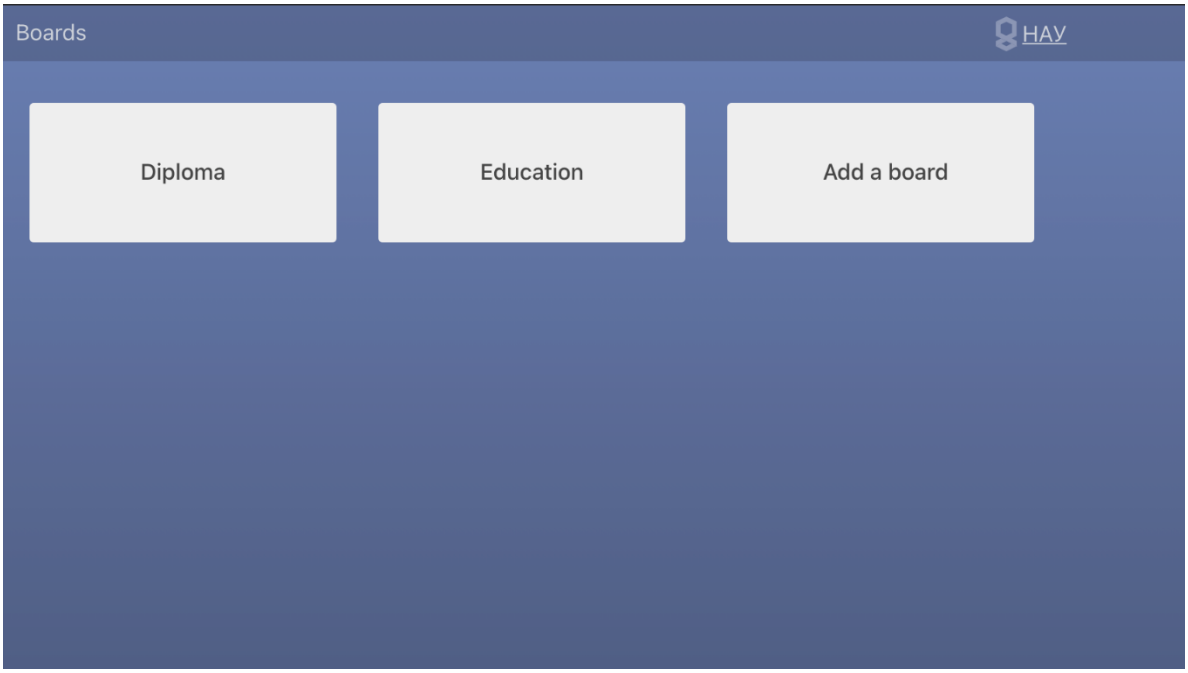


Рис.3.3 Створення нового проекту

У вікні " Створення задач " користувач має можливисть створити нові задачі

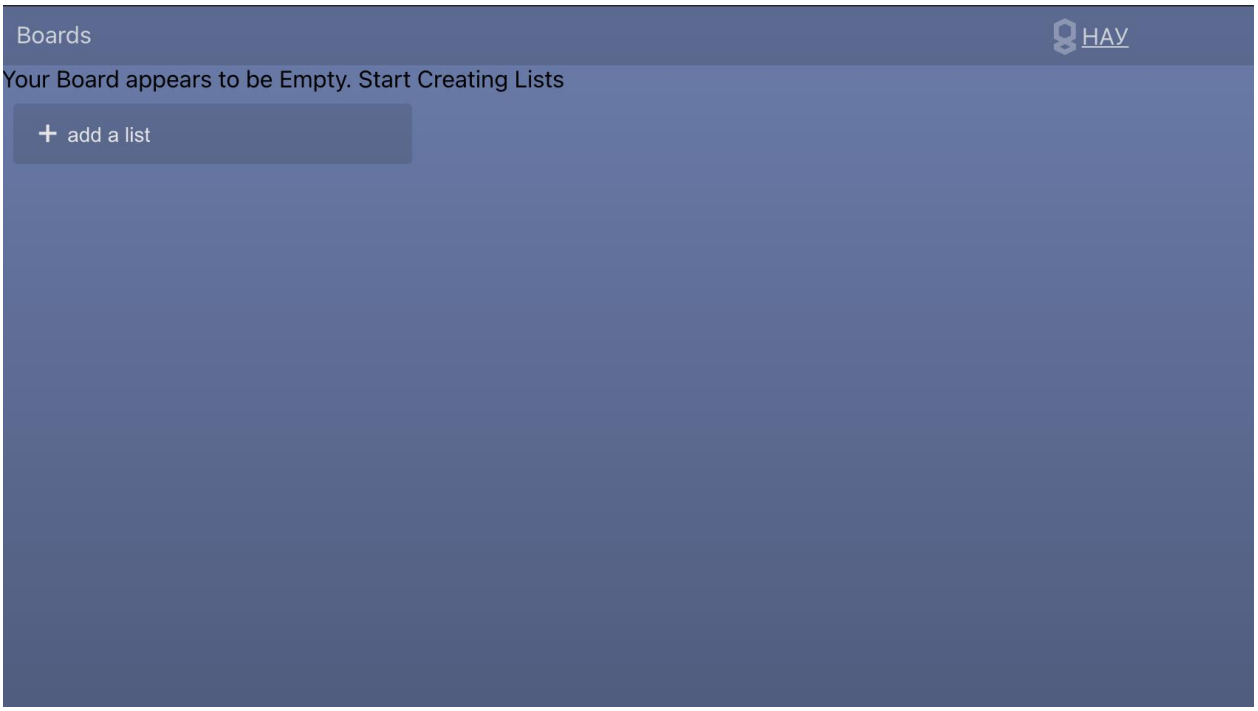


Рис. 3.4 Створення задач

Функція додавання нової дошки (add a list)

Ця функція дозволяє ввести назву назву робочої дошки та створити декілька нових дошек на приклад To Do, Doing, Done

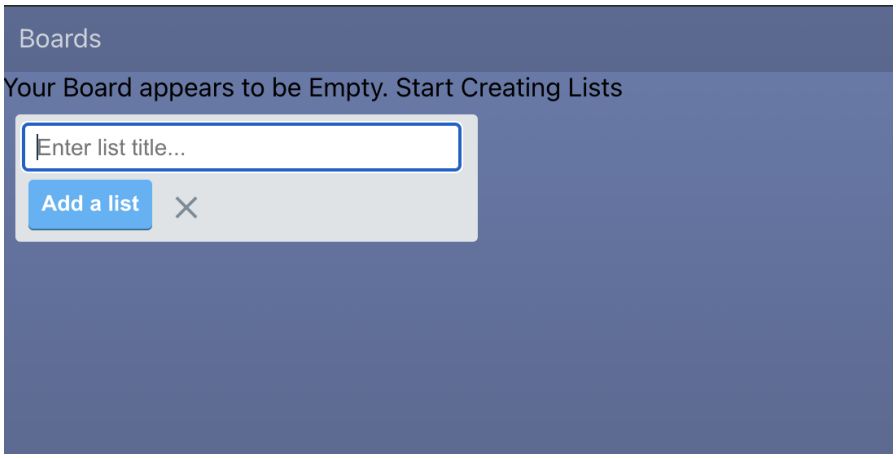


Рис. 3.5 Додати дошку

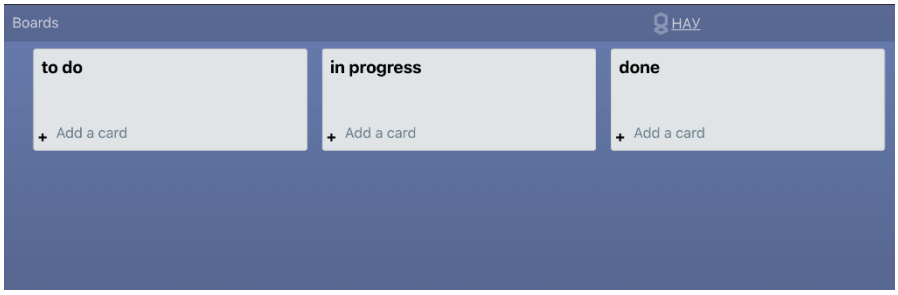


Рис. 3.6 Приклад Канбан-дошки

Функція додавання нової задачі до проекту

Ця функція буде додавати нові задачі до проекту у нас не має обмежень на кількість задач

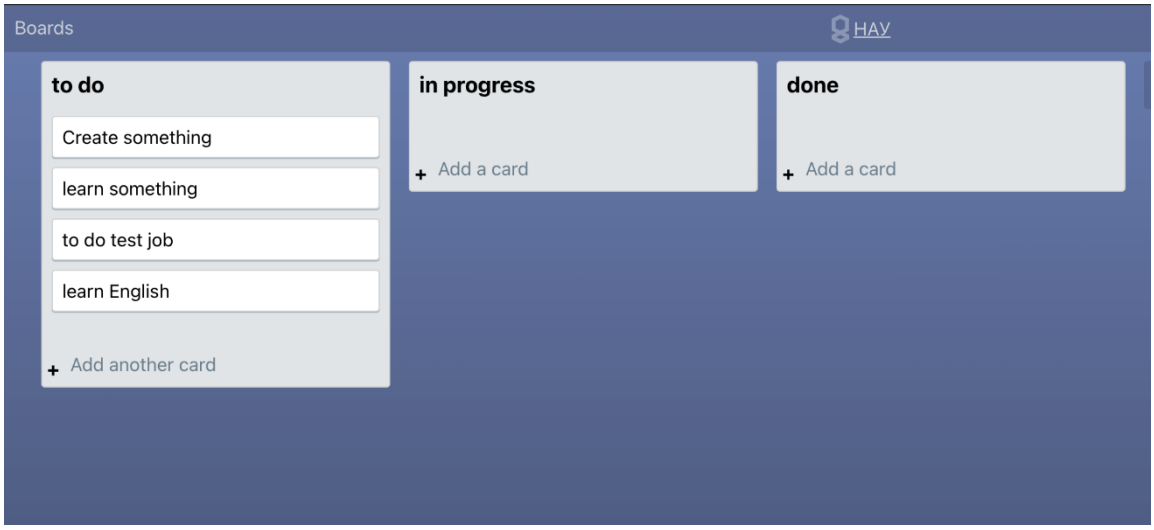


Рис. 3.7 Функція додавання нової задачі

Функція Drag and Drop

За допомогою цієї функції ми можемо перетягувати завдання з однієї дошки в іншу тим самим змінюючи статус завдань

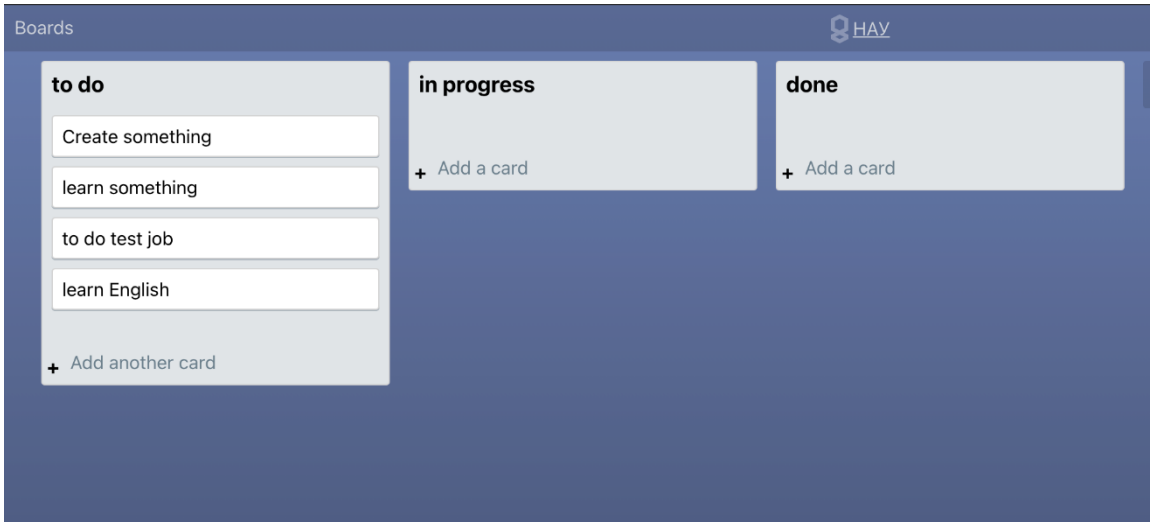


Рис. 3.8 Drag and Drop

3.2 Принципи проектування React

Що таке React – бібліотека JavaScript для роботи з інтерфейсами користувача (UI), створена розробниками Facebook. Використання бібліотеки на сайті цієї соціальної мережі почалося в 2011 році. А в 2013 році Facebook випустив React як рішення з відкритим кодом. Використовуючи React, розробники створюють веб-додатки, які змінюють відображення без перезавантаження сторінки. Завдяки цій програмі вони швидко реагують на дії користувача, такі як заповнення форм, застосування фільтрів, додавання товарів у кошик тощо.

React використовується для відтворення компонентів інтерфейсу користувача. Крім того, бібліотека може повністю керувати інтерфейсом. У цьому випадку React використовується з бібліотеками управління станом і маршрутизацією, такими як Redux і React Router.

Розробка React стосується опису того, що потрібно відобразити на сторінці (а не того, як це зробити в браузері). Крім того, це означає значне зменшення кількості коду шаблону.

Angular, з іншого боку, має інструменти командного рядка, які генерують шаблонний код для компонентів. Чи здається це трохи незвичним для сучасних інструментів розробки інтерфейсу користувача? Насправді справа в тому, що в Angular стільки шаблонного коду, що навіть був створений спеціальний інструмент для його генерації.

Коли ви починаєте розробку в React, ви просто починаєте писати код. Немає стандартного коду компонента, який потрібно якось згенерувати. Звичайно, перед розробкою потрібна певна підготовка, але якщо говорити про компоненти, то їх можна назвати чистими функціями.

У React усі компоненти інтерфейсу можна виразити як набір чистих функцій. Використання чистих функцій для розробки інтерфейсу користувача можна порівняти з ковтком свіжого повітря.

Тепер, коли ми розглянули причини популярності React, які цілком можуть схилити вас саме до цієї бібліотеки при виборі інструментів для розробки інтерфейсу користувача, давайте перейдемо до практики.

Ключові особливості React: декларативність, універсальність, компонентний підхід, віртуальний DOM, JSX Однією з ключових особливостей React є універсальність. Цю бібліотеку можна використовувати з React Native на серверах і мобільних платформах. Це принцип «Навчись раз, пиши будь-де». Ще одна важлива особливість бібліотеки – декларативність. Використовуючи React, розробник описує, як виглядають компоненти інтерфейсу в різних станах. Декларативний підхід скорочує код і робить його зрозумілим.

React базується на компонентах, що є ще однією важливою особливістю бібліотеки. Кожен компонент повертає частину інтерфейсу користувача зі своїм станом. Поєднуючи компоненти, програміст створює повний інтерфейс веб-додатку. Важливою особливістю React є використання JSX. Це розширення синтаксису JavaScript, яке зручно використовувати для опису інтерфейсу. JSX схожий на HTML, але все ще є JavaScript. Приклад JSX можна побачити нижче:

```
const header = text ? <h1>{text}</h1> : null;

const vdom = (
  <div>
    {header}
    <Hello />
  </div>
);
```

Рис. 3.9 Приклад JSX

Для розробки кваліфікаційної роботи я також використовував об’єктну модель документа (DOM), яка була однією з причин вибору фреймворку React DOM – це спосіб представлення об’єктів у документах HTML, XHTML і XML для взаємодії з ними.

Відповідно до цієї моделі кожен із цих документів є ієрархічним деревом елементів, яке називається деревом DOM. За допомогою спеціальних методів

ми можемо отримати доступ до певних елементів нашого документа та змінити їх на свій смак. Коли ми створюємо динамічну інтерактивну веб-сторінку, ми хочемо, щоб DOM оновлювався якомога швидше після зміни стану певного елемента. Для цього завдання деякі фреймворки використовують техніку під назвою «брудна перевірка», яка полягає в періодичному запиті статусу документа та перевірці змін у структурі даних. Як ви можете собі уявити, таке завдання може стати справжньою проблемою у сильно завантажених програмах. Віртуальний DOM, у свою чергу, зберігається в пам'яті.

Ось чому React може змінити віртуальний DOM одразу, коли змінюється «реальний» DOM. React «збирає» такі зміни, порівнює їх зі станом DOM, а потім перемальовує змінені компоненти. За такого підходу ви не виконуєте регулярні оновлення DOM. Це може досягти вищої продуктивності програм React. Другий наслідок – ізоморфна природа React: ви можете рендерити на стороні сервера так само, як і на стороні клієнта.

Одним із найпопулярніших принципів є розміщення файлів CSS і JS у папках, згрупованих за функціями або маршрутами.

У нашому прикладі ми бачимо, що ми створюємо окрему папку для кожного елемента.

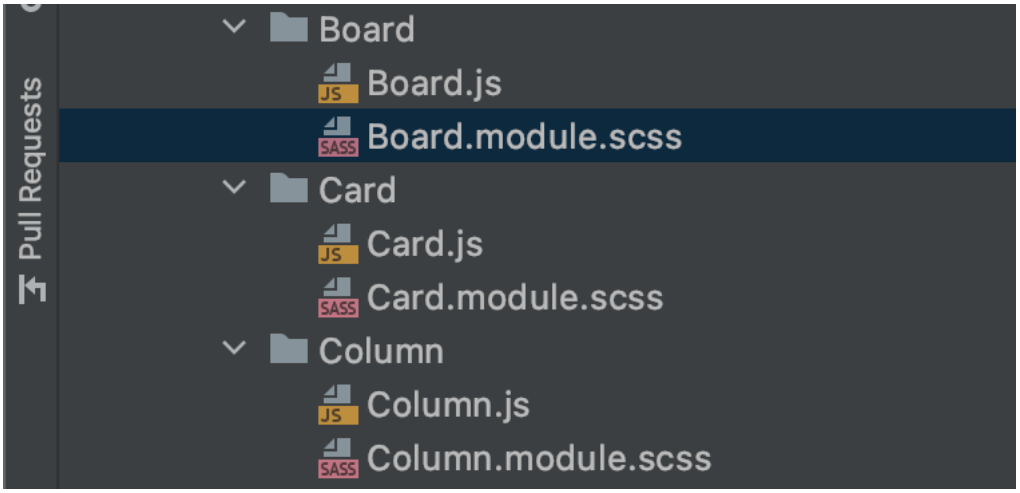


Рис. 3.10 Принципи проектування React Drag and Drop

Це ще одна причина вибрати цю бібліотеку js

Спочатку мені потрібно було вибрати бібліотеку перетягування. В інтернеті їх багато: DraggableJS, Dragula, Interactjs.io тощо.

І є лише дві бібліотеки, оптимізовані для використання з React: React-DnD і React-Beautiful-Dnd. Бібліотека React Beautiful DND чудово виглядає в демонстраціях, але, на жаль, вона була випущена лише після реалізації проекту. Тому я використав React-DnD.

Ця бібліотека надає нам набір компонентів вищого порядку (HOC). Простіше кажучи:

DragSource – дозволяє перетягувати компонент;

DropTarget – додає можливість взаємодії з компонентами перетягування;

DragLayer – дозволяє реалізувати власний попередній перегляд для перетягуваного елемента;

DragDropContext – має ініціалізувати бібліотеку. Ще один важливий компонент, без якого React DnD не працює, це бекенд перетягування. Бібліотека для забезпечення кросбраузерної сумісності, абстракція над стандартним API браузера.

Приклад функцій, написаних на js для D&D у моєму проекті

Лістинг 3.1 є прикладом функції списку перетягування

```
        return;
    }
    // Moving from one list to another list
    const startTaskIds = Array.from(start.taskIds);
    startTaskIds.splice(source.index, deleteCount: 1);
    const newStart = {
        ...start,
        taskIds: startTaskIds
    };
    const finishTaskIds = Array.from(finish.taskIds);
    finishTaskIds.splice(destination.index, deleteCount: 0, draggableId);
    const newFinish = {
        ...finish,
        taskIds: finishTaskIds
    };

    const newState = {
        ...this.state,
        columns: {
            ...this.state.columns,
            [newStart.id]: newStart,
            [newFinish.id]: newFinish
        }
    };

    this.setState(newState);
```

Композиція

Ключовою особливістю React є склад компонентів. Написані компоненти повинні добре поєднуватися. Важливо мати можливість додавати функціональність до компонента, не викликаючи хвилі змін у всьому коді. Наприклад, повинна бути можливість призначити внутрішній стан компоненту без зміни компонентів, які від нього залежать. Також повинна бути можливість додати код ініціалізації та знищення до кожного компонента, якщо це необхідно. Немає нічого «поганого» у використанні методів стану або життєвого циклу в компонентах. Як і будь-яку потужну функцію, їх слід використовувати економно. Навпаки, я вважаю, що вони є невід'ємною частиною того, що робить React корисним. У майбутньому ми можемо додати більше функціональних шаблонів програмування, але як внутрішній стан, так і

методи життєвого циклу будуть частиною цього шаблону. Компоненти часто називають «просто функціями», але ми вважаємо, що вони мають бути більшими, ніж це, щоб бути корисними.

Переваги структурованого проекту на React

Членам команди не потрібно турбуватися про структуру проекту. Натомість вони можуть зосередитися на розробці продукту. Оскільки React має величезну екосистему, вам завжди потрібно думати: Redux чи Mobx? HOC чи рендеринг? Реакційний рух чи реактивна пружина? Усунення однієї з проблем – узгоджена структура. Загальна структура проектів допомагає новим фахівцям швидко освоїти роботу. Короткий опис і ви можете здогадатися, що знаходиться в кожній папці і для чого потрібні файли в ній. Навіть люди з меншим досвідом можуть створювати масштабовані проекти. Спільне використання та повторне використання коду. 5 цілей входу в структуру:

• Підвищення продуктивності. Нечіткий пошук файлів у редакторі має бути легшим.

• Природний процес передачі файлів.

• Ненав'язлива та розумна гнучкість – розробники повинні мати певну свободу.

Структура повинна забезпечувати можливість масштабування та повторного використання. Структурований код повинен бути відносно простим і мати мінімальний поріг введення.

Файл компонента має бути в CamelCase. Назви компонентів мають називатися TabSwitcher, а не tabSwitcher, tab-switcher тощо. У цій нотації немає потреби називати інші типи файлів, оскільки CamelCase сигналізує нам, що файл є компонентом React.

За умовчанням уникати експорту

Стандартний експорт є окремим випадком іменованого експорту, тому його потрібно обробляти особливим чином. Це поширена причина плутанини. Я імпортував компонент таким чином:

Лістинг 3.2 Імпорт компонентів

3.3 Послідовність, життєвий цикл React компонента

ComponentWillMount не дуже відрізняється від конструктора – він викликається лише один раз у початковому життєвому циклі

Конструктори є ядром ООП - це спеціальна функція, яка викликається кожного разу, коли створюється новий об'єкт. Дуже важливо викликати суперфункцію, коли наш клас розширює поведінку іншого класу, який має конструктор.

Виконання цієї спеціальної функції викликає конструктор нашого батьківського класу та дозволяє йому ініціалізувати себе. З цієї причини ми маємо доступ до this.props лише після виклику super. (Це означає виклик super (props) у класі успадкування React.Component.) Таким чином, конструктори є чудовим місцем для ініціалізації компонента – створення довільних полів (змінні починаються з них) або ініціалізації стану компонента на основі отриманих props. Це також єдине місце, де ви можете змінити/встановити стан напряму, замінивши поле this.state. У всіх інших випадках ви повинні використовувати this.setState.

Існує кілька причин, чому компонент може бути перемальований, і залежно від причини викликаються різні функції, які дозволяють розробнику оновлювати певні частини компонента. Створення компонента Перший цикл — це створення компонента, яке зазвичай відбувається, коли компонент вперше виявляється в аналізованому дереві JSX:

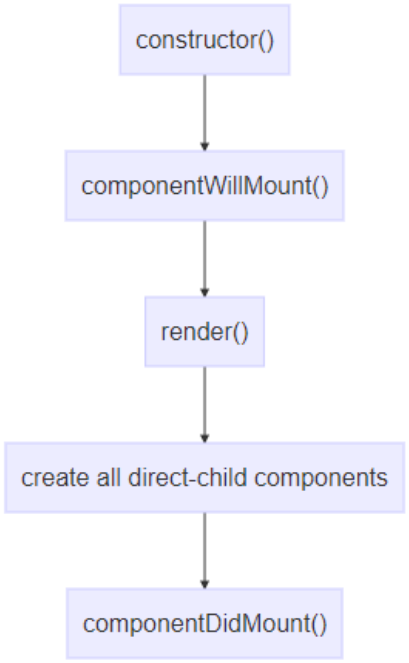


Рис. 3.11 ComponentWillMount

Компонент перемальовується в зв'язку з перемальовуванням батьківського компонента

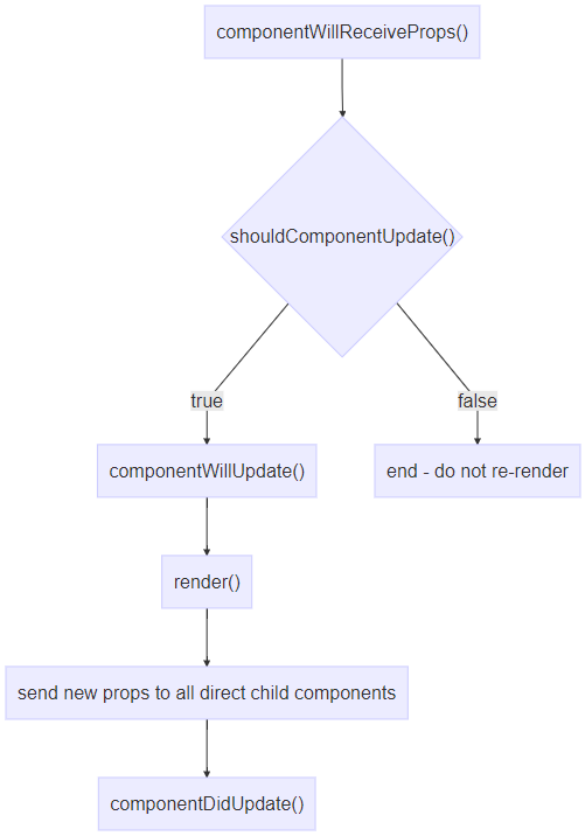


Рис. 3.12 shouldComponentUpdate

Компонент перерисовывается в связи с внутренними изменениями (например вызов this.setState())

3.4 Розгортання React app

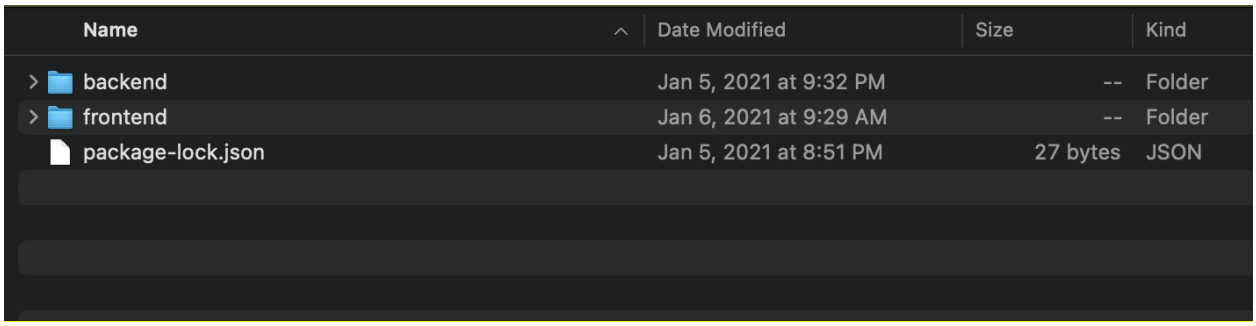


Рис. 3.13 Папка с проектом

Папка з проектом містить у собі дві папки Frontend і Backend для запуску програми необхідно послідовно запустити обидва пакети. У кожній з папок лежить файл згенерований React'ом Package.json

Увага! Повинен бути встановлений node js

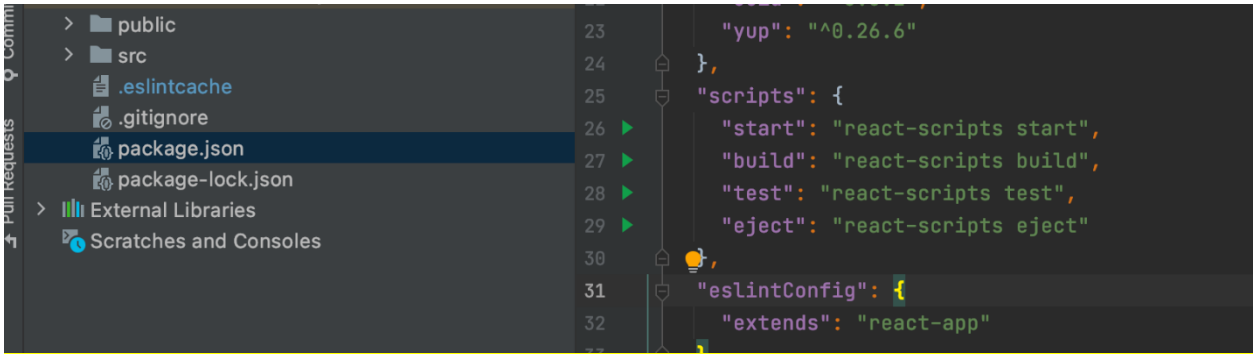


Рис. 3.14 Команди Package.json

Package.json містить необхідні команди для запуску програми в режимах Dev і Prod

Приклад: у командному рядку спочатку введіть npm run start for backEnd, а потім frontend.

Асинхронне середовище виконання JavaScript, кероване подіями, Node.js розроблено для створення масштабованих мережевих програм. У наведеному нижче прикладі Hello World багато підключень можуть оброблятися одночасно.

Зворотний виклик запускається після кожного з'єднання, але коли не потрібно виконувати жодної роботи, Node.js переходить у сплячий режим.

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Це на відміну від сьогоднішньої більш поширеної моделі паралелізму, в якій використовуються потоки ОС. Мережа на основі потоків відносно неефективна і дуже складна у використанні. Крім того, користувачі Node.js не хвилюються про блокування процесу, оскільки блокувань немає. Майже жодна функція в Node.js безпосередньо не виконує введення-виведення, тому процес ніколи не блокується, за винятком випадків, коли введення-виведення виконується за допомогою синхронних методів стандартної бібліотеки Node.js. Оскільки ніщо не блокує, масштабовані системи дуже розумно розробляти на Node.js.

Якщо частина цієї мови незнайома, є повна стаття про блокування та неблокування.

Node.js подібний за дизайном до таких систем, як Ruby's Event Machine і Python's Twisted. Node.js розвиває модель подій трохи далі. Він представляє цикл подій як конструкцію часу виконання, а не як бібліотеку. В інших системах завжди існує блокуючий виклик для запуску циклу подій. Як правило, поведінка визначається через зворотні виклики на початку сценарію, а в кінці сервер запускається через блокуючий виклик, наприклад EventMachine::run(). У Node.js немає такого виклику запуску циклу подій. Node.js просто входить у

цикл подій після виконання вхідного сценарію. Node.js виходить із циклу подій, коли більше немає зворотних викликів для виконання. Така поведінка схожа на JavaScript у браузері — цикл подій прихований від користувача.

HTTP є першокласним громадянином у Node.js, розробленому з урахуванням потокового передавання та низької затримки. Це робить Node.js добре придатним для створення веб-бібліотеки або фреймворку.

Оскільки Node.js розроблено без потоків, це не означає, що ви не можете скористатися перевагами кількох ядер у своєму середовищі. Дочірні процеси можуть бути породжені за допомогою нашого API child_process.fork(), і вони створені для зручності спілкування. На цьому ж інтерфейсі побудовано кластерний модуль, який дозволяє вам спільно використовувати сокети між процесами, щоб забезпечити балансування навантаження на ваші ядра.

Як і багато фреймворків, Node.js користується популярністю в різних розробках. Ось кілька основних областей, де Node.js демонструє себе найбільш кращим чином:

Розробка серверних додатків. Node.js ідеально підходить для створення високопродуктивних серверів. Завдяки своїй асинхронній та подійно-орієнтованій моделі програмування Node.js здатний обробляти безліч одночасно запитів і забезпечувати високу відгукність серверних додатків. Він може бути використаний для створення веб-серверів, API, мікросервісів та інших серверних програм.

Веб-розробка. Node.js став популярним інструментом для розробки веб-приложень. Якщо встановити Node.js, він надасть зручні інструменти та бібліотеки для створення серверної частини веб-додаток, а також забезпечить просту взаємодію з клієнтською стороною на основі JavaScript. За допомогою фреймворків, таких як Express.js або Koa.js, розробники можуть створювати потужні та масштабовані веб-додатки.

Розробка інструментів командної строки. Node.js надає можливість створювати інструменти командного рядка (CLI), які спрощують автоматизацію розробки завдань. Завдяки JavaScript розробники можуть створювати вбудовані

скрипти для виконання різних завдань, таких як збірка проектів, керування залежностями та багато іншого.

Розробка додатків реального часу. Завдяки асинхронному програмуванню Node.js є відмінним вибором для розробки програм реального часу, таких як чати, онлайн-ігри або системи моніторингу. Його ідеальна здатність обробляти безліч одночасових підключень і забезпечувати низьку затримку робить його оптимальним для таких програм.

Інтернет речей (IoT). Node.js може бути використаний у сфері інтернет-речей – для розробки програм, керуючих пристроями IoT. Завдяки легковажній архітектурі та ефективності на node js можна написати серверні додатки, які можуть обробляти дані різними пристроями IoT і керувати ними.

Це лише деякі приклади використання Node.js. Завдяки своїй гнучкості, продуктивності та обширній екосистемі Node.js стає все більш популярним інструментом у розробці програм різного роду.

Характеристики

Node.js має низку унікальних характеристик, які роблять його привабливим для розробників. Ось деякі з ключових особливостей Node.js:

Висока швидкість виконання. Node.js заснований на движку JavaScript V8 від Google, який забезпечує швидку та ефективну обробку коду. Це дозволяє Node.js виконувати операції швидко, що особливо важливо для створення високопродуктивних серверних програм.

Універсальність та гнучкість. Node.js дозволяє використовувати JavaScript як мову програмування для розробки серверних і клієнтських додатків. Це дозволяє розробникам використовувати одну мову та ділитися кодом між серверною та клієнтською стороною. Завдяки цій універсальності, Node.js полегшує розробку повноцінних веб-додатків та спрощує командну взаємодію.

Безліч модулів та бібліотек. Node.js має велику екосистему модулів та бібліотек, доступних через вбудований менеджер пакетів npm. Розробники можуть використовувати ці модулі та бібліотеки для розширення

функціональності своїх додатків та прискорення розробки. Завдяки величезній спільноті розробників, яка підтримує npm, можна знайти рішення практично для будь-якого завдання.

Сумісність із кількома платформами. Node.js може бути запущений на різних операційних системах, таких як Windows, MacOS та Linux. Це робить його універсальним та забезпечує розробникам свободу вибору платформи розгортання своїх додатків. Крім того, Node.js також сумісний з контейнеризацією та хмарними платформами, що спрощує розгортання та масштабування програм.

Ці характеристики роблять Node.js потужним інструментом розробки різноманітних додатків. Він забезпечує високу швидкість виконання, універсальність і гнучкість, а також має велику екосистему модулів і бібліотек. Завдяки своїй сумісності з різними платформами, Node.js надає розробникам гнучкість у виборі інфраструктури для своїх проектів.

Сфери застосування Node.js

Node.js надає потужні можливості та знаходить широке застосування у різних сферах розробки. Ось деякі з головних областей, де Node.js є особливо корисним:

Чати у реальному часі. Node.js блискуче справляється із розробкою чатів у реальному часі. Завдяки своїй асинхронній та подієво-орієнтованій моделі програмування, Node.js дозволяє обробляти одночасні підключення та передавати повідомлення миттєво. Це робить його ідеальним інструментом для створення інтерактивних чат-додатків, командних чатів та систем обміну миттєвими повідомленнями.

Інтернет речей (IoT). Node.js є ідеальним вибором для розробки програм, пов'язаних з Інтернетом речей. Завдяки своїй ефективності та можливості обробки великої кількості одночасних з'єднань, Node.js дозволяє керувати пристроями IoT та обробляти потоки даних у режимі реального часу. Від систем розумного будинку до індустріальних рішень IoT, Node.js допомагає

створювати потужні та масштабовані програми, які пов'язують пристрої та хмари.

Потокове передавання даних. Node.js має вбудовану підтримку потоків, що робить його ідеальним для розробки додатків, яким необхідна ефективна потокова передача даних. Завдяки цьому можна обробляти та передавати потоки відео, аудіо, файлів та інших форматів даних у режимі реального часу. Node.js дозволяє створювати системи потокової передачі даних, стрімінгові платформи та медіа-сервери з легкістю та високою продуктивністю.

Складні односторінкові програми (SPA). Node.js у поєднанні з фреймворками, такими як Express.js та Next.js, надає чудову платформу для розробки складних односторінкових програм (SPA). Node.js забезпечує серверну частину програми, а JavaScript на клієнтській стороні дозволяє створювати інтерактивні та чуйні інтерфейси користувача. Це дозволяє розробникам створювати потужні Node.js програми, які працюють плавно та ефективно.

Програми на основі REST API. Node.js є популярним вибором для розробки програм на основі REST API. З його допомогою можна легко створювати та розгортати гнучкі та масштабовані API для обміну даними між різними системами та додатками. Node.js надає зручні інструменти для обробки запитів та створення серверів API, дозволяючи розробникам швидко створювати надійні та гнучкі веб-сервіси.

На закінчення, "нід" є потужним інструментом, що володіє широким спектром можливостей. Node.js застосовується у розробці чатів у реальному часі, систем IoT, потокової передачі даних, складних SPA та додатків на основі REST API. З його допомогою розробники можуть створювати ефективні, масштабовані та високопродуктивні програми у різних сферах.

Node.js є однією з найпопулярніших технологій веб-розробки, і багато відомих компаній вибрали його для створення своїх сайтів та додатків. Ось деякі з популярних сайтів та приклади додатків, які використовують Node.js:

Netflix. Цей стрімінговий сервіс став однією з найвідоміших платформ на цій платформі. Node.js використовується для створення серверної інфраструктури, що забезпечує швидку та надійну доставку відео контенту своїм мільйонам користувачів.

Uber. Uber, популярна служба таксі та каршерингу, також заснована на Node.js. Його використовують для обробки мільйонів запитів на зіставлення пасажирів та водіїв у режимі реального часу.

PayPal. Одна з провідних онлайн-платіжних систем використовує Node.js у своїй серверній частині. Node.js дозволяє їм обробляти величезні обсяги транзакцій та забезпечувати високу продуктивність та надійність платіжних операцій.

eBay. Найбільший світовий майданчик для онлайн-торгівлі також використовує Node.js для обробки своїх запитів і забезпечення швидкої і плавної роботи процесів eCommerce.

GoDaddy. Один з найбільших хостинг-провайдерів та реєстраторів доменних імен, використовує Node.js для своїх внутрішніх систем управління та обробки запитів клієнтів.

LinkedIn. Найбільша соціальна мережа для професійних контактів також використовує Node.js у своїй інфраструктурі. Node.js допомагає їм обробляти величезні обсяги даних та надавати миттєві оновлення та сповіщення користувачам.

Walmart. Одна з найбільших роздрібних мереж у світі, використовує Node.js для розробки своїх внутрішніх інструментів та систем управління, що дозволяють їм ефективно управляти своїм бізнесом та забезпечувати високий рівень обслуговування клієнтів.

Це лише деякі приклади продуктів, що базуються на Node.js. Він довів свою потужність та ефективність у різних галузях, забезпечуючи високу продуктивність, масштабованість та гнучкість розробки.

Перспективи розвитку та використання Node.js залишаються дуже обнадійливими. Зростаюча популярність та прийняття Node.js в індустрії

розробки програмного забезпечення вказують на те, що він продовжуватиме розвиватися та використовуватися ще багато років.

По-перше, продуктивність і масштабованість Node.js згодом лише покращуватиметься. За останні роки платформа отримала значні покращення, такі як покращена робота з пам'яттю та оптимізація обробки одночасних запитів. Ці апдейти дозволяють Node.js ефективно працювати з великими обсягами даних та забезпечувати високу продуктивність навіть за високих навантажень.

По-друге, спільнота розробників Node.js продовжуватиме розробляти нові модулі, бібліотеки та інструменти, розширюючи можливості фреймворку. Різноманітність екосистеми Node.js дозволяє розробникам швидко та ефективно створювати програми різного призначення, і ця тенденція тільки посилиться в майбутньому.

Одним з найважливіших факторів успіху Node.js стала його універсальність та сумісність із різними платформами. Це відкриває двері для розробки програм на різних операційних системах та використання різних хмарних сервісів. Зі зростанням інтересу до розробки додатків для мобільних пристроїв та Інтернету речей, Node.js буде відігравати важливу роль у цих галузях завдяки своїй ефективності та гнучкості.

Нарешті, з урахуванням розвитку технологій, таких як штучний інтелект, машинне навчання та аналіз великих даних, Node.js може відігравати важливу роль у обробці та аналізі даних у реальному часі. Його асинхронна природа та можливість обробки великої кількості одночасних запитів роблять його привабливим для таких додатків, де потрібний швидкий та ефективний обмін даними.

В цілому, Node.js має яскраві перспективи розвитку та використання в майбутньому. Його швидкість, гнучкість і масштабованість роблять його привабливим вибором для розробників та компаній, які прагнуть створювати інноваційні програми та послуги. Вже сьогодні на Node.js працюють сотні

успішних проектів із тривалим життєвим циклом, а отже в найближчому майбутньому він точно нікуди не зникне.

3.3. База даних MongoDB

Для свого проекту я вибрав базу даних MongoDB вона реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які притаманні об'єктно-реляційних баз даних. З часів динозаврів було звичайною справою зберігати всі дані в реляційних базах даних (MS SQL, MySQL, Oracle, PostgresSQL).

При цьому було не так важливо, а чи підходять реляційні бази даних для зберігання даного типу даних чи ні. На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати. Але, навіть враховуючи всі недоліки традиційних баз даних і гідності MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні.

В якійсь ситуації MongoDB дійсно поліпшить продуктивність вашої програми, наприклад, якщо треба зберігати складні за структурою дані. В іншій же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних - в традиційних БД. Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері.

Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність

Формат даних в MongoDB

Одним з популярних стандартів обміну даними та їх зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних в MongoDB в цьому плані схожий на JSON, хоча

формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (Бісон) або скорочення від binary JSON. BSON дозволяє працювати з даними швидше: швидше виконується пошук і обробка. Хоча треба зазначити, що BSON на відміну від зберігання даних в форматі JSON має невеликий недолік: в цілому дані в JSON-форматі займають менше місця, ніж в форматі BSON, з іншого боку, даний недолік з лишком окупається швидкістю.

Якщо в традиційному світі SQL є таблиці, то в світі MongoDB є колекції. І якщо в реляційних БД таблиці зберігають однотипні жорстко структуровані об'єкти, то в колекції можуть містити найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Реплікація

Система зберігання даних в MongoDB представляє набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Всі вторинні вузли зберігають цілісність і автоматично оновлюються разом з оновленням головного вузла. І якщо основний вузол з якихось причин виходить з ладу, то один з вторинних вузлів стає головним.

Простота у використанні

Відсутність жорсткої схеми бази даних і в зв'язку з цим потреби при щонайменшій зміні концепції зберігання даних пересоздавать цю схему значно полегшують роботу з базами даних MongoDB і подальшим їх масштабуванням. Крім того, економиться час розробників. Їм більше не треба думати про пересоздании бази даних і витрачати час на побудову складних запитів.

3.6. Визначення виду і класу проекту

Вигляд проекту - первинний критерій, що відображає основну сферу діяльності, в рамках якої здійснюється проект. Вид проекту визначається на підставі його мети / цілей і предметної області, в якій здійснюється проект. При наявності декількох цілей проекту і предметних областей вид проекту визначається на підставі його головної мети / предметної області. Вид проекту

визначається на фазі «Концепція» Офісом управління проектами (далі - ОУП) і затверджується Проектним комітетом.

При необхідності вид проекту може бути переглянутий за рішенням Проектної комітету. Рішення про зміну виду проекту, прийняті суб'єктами корпоративної системи управління проектами рівня Групи, є обов'язковими для виконання для суб'єктів дивізіонального рівня.

Коммерческий ИТ-проект Специальный

Проекти, що реалізуються в рамках комерційної діяльності, метою яких є розвиток пропозиції клієнтам продуктів і / або послуг. Проекти, спрямовані на розробку, впровадження та / або поліпшення програмних продуктів, корпоративної IT-інфраструктури.

Таблиця 1 - Види проектів

інвестиційний	Проекти, спрямовані на вкладення капіталу (інвестування) в розширення або вдосконалення основних фондів Групи і / або окремого дивізіону.
інноваційний	Проекти, що включають об'єкти творчої діяльності, винаходи, відкриття, які впливають на продуктивність і конкурентоспроможність Групи, дивізіону, продукту.
організаційний	Проекти, спрямовані на оптимізацію діяльності Групи і дивізіонів та їх функціонування в умовах ринку (процесів і процедур), в тому числі проекти щодо зміни структури управління Групи, масової розробці (коригуванні) комплектів організаційних документів, що регламентують діяльність окремих посадових осіб і підрозділів, впровадження систем управління, оптимізації бізнес-процесів.

соціальний	Проекти, спрямовані на створення матеріально-духовних цінностей, метою яких є підтримка і підвищення якості життя працівників підприємств Групи, їх сімей та суспільства в цілому.

Вторинний (після виду проекту) критерій, що відображає рівень складності і важливості проекту. Клас проекту визначається на підставі дерева рішень для кожного з видів проектів. Параметри, які використовуються для визначення класу проекту, включають параметри, загальні для всіх видів проектів, і додаткові параметри для окремих видів проектів. Є спільними для всіх видів проектів, включають:

- профільність проекту;
- крос-дивизиональную проекту;
- обсяг інвестицій в рамках проекту / бюджет проекту.

Розширені можливості пошуку для окремих видів проектів включають:

- масштаб впливу (для організаційних проектів), відображає вплив результатів проекту на бізнес-процеси і оргструктуру;
- рівень залучення (для соціальних і комерційних проектів), відображає залучення зовнішніх сторін - органів влади і суб'єктів комерційної діяльності;
- планований обсяг реалізації (виручки) (для комерційних)

За рішенням ОУП Групи для визначення класу проекту може бути використаний інший підхід. Клас проекту визначається на фазі «Концепція» ОУП і затверджується Проектним комітетом. При необхідності клас проекту може бути переглянутий за рішенням Проектної комітету.

Рішення про зміну класу проекту, прийняті суб'єктами корпоративна система управління проектами
рівня Групи, є обов'язковими для виконання для суб'єктів дивізіонального рівня. Клас проекту використовується для вибору підходу до управління проектом - визначення моделі управління проектом. Виділяються п'ять класів проектів з урахуванням прийнятих видів проектів:

• Клас А. Непрофільні проекти Групи.

• Клас Б. Профільні складні проекти і / або крос-дивізіональні проекти.

• Клас В. Профільні середні проекти і / або проекти, що реалізуються в рамках одного дивізіону.

• Клас Г. Профільні прості проекти і / або проекти, що реалізуються в рамках одного структурного підрозділу.

• Клас Д. Профільні проектні активності, що реалізуються поза методологічного контуру управління проектами.

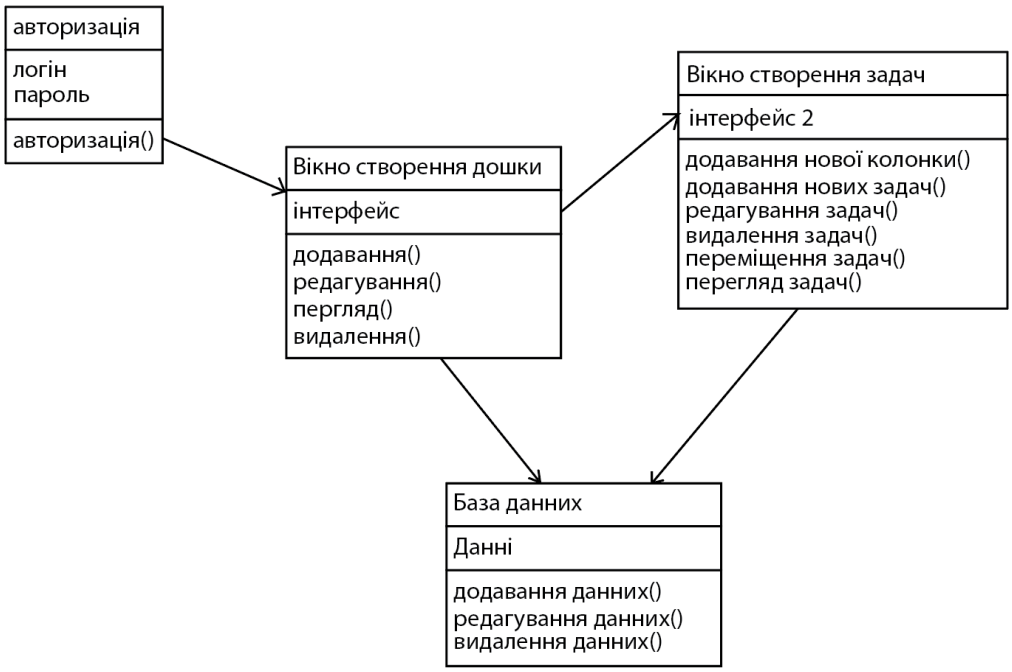


Рис. 3.12. Діаграма класів

3.7. Архітектура

Для програми важливо визначити архітектуру. Найбільш придатною була б структура клієнт-серверної моделі, яка розподіляє завдання робочого навантаження між виробниками, які називаються серверами, та споживачами, які називаються клієнтами (рис. 3.13.).

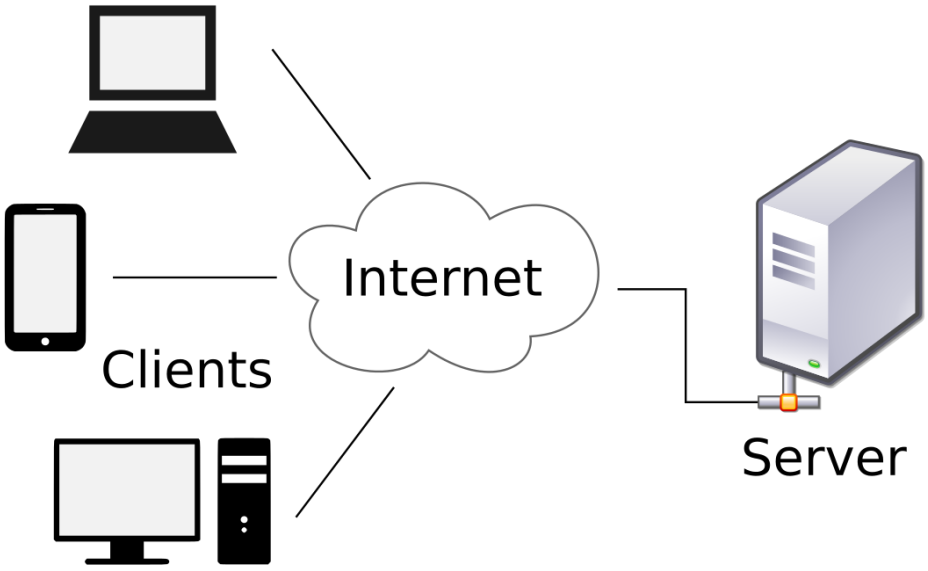


Рис. 3.13. Клієнт-серверна архітектура

Найкращим вибором для програми буде така архітектура, яка дозволяє динамічно переміщувати, додавати та видаляти завдання, не турбуючи користувача постійним перезавантаженням сторінки. Тому було обрано дещо модифіковану односторінкову архітектуру програми, яка представлятиме головну сторінку як статичну, а іншу сторінку як SPA (рис. 3.14).

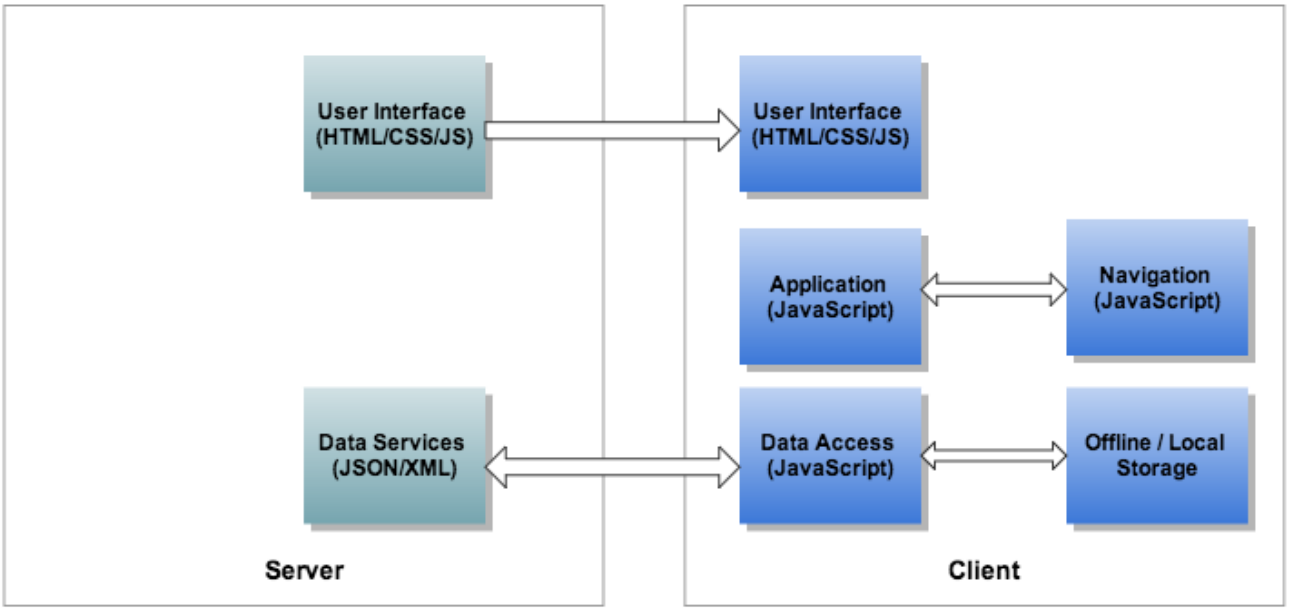


Рис. 3.14. Односторінкова архітектура програми

3.8. Діаграма потоку даних

Діаграма потоку даних показує, як дані обробляються в системі. Він дає нам інформацію про інформацію, необхідну суб'єктам, а також її входи та виходи. Для застосування буде використана діаграма потоку даних, показана на рисунку 3.15.

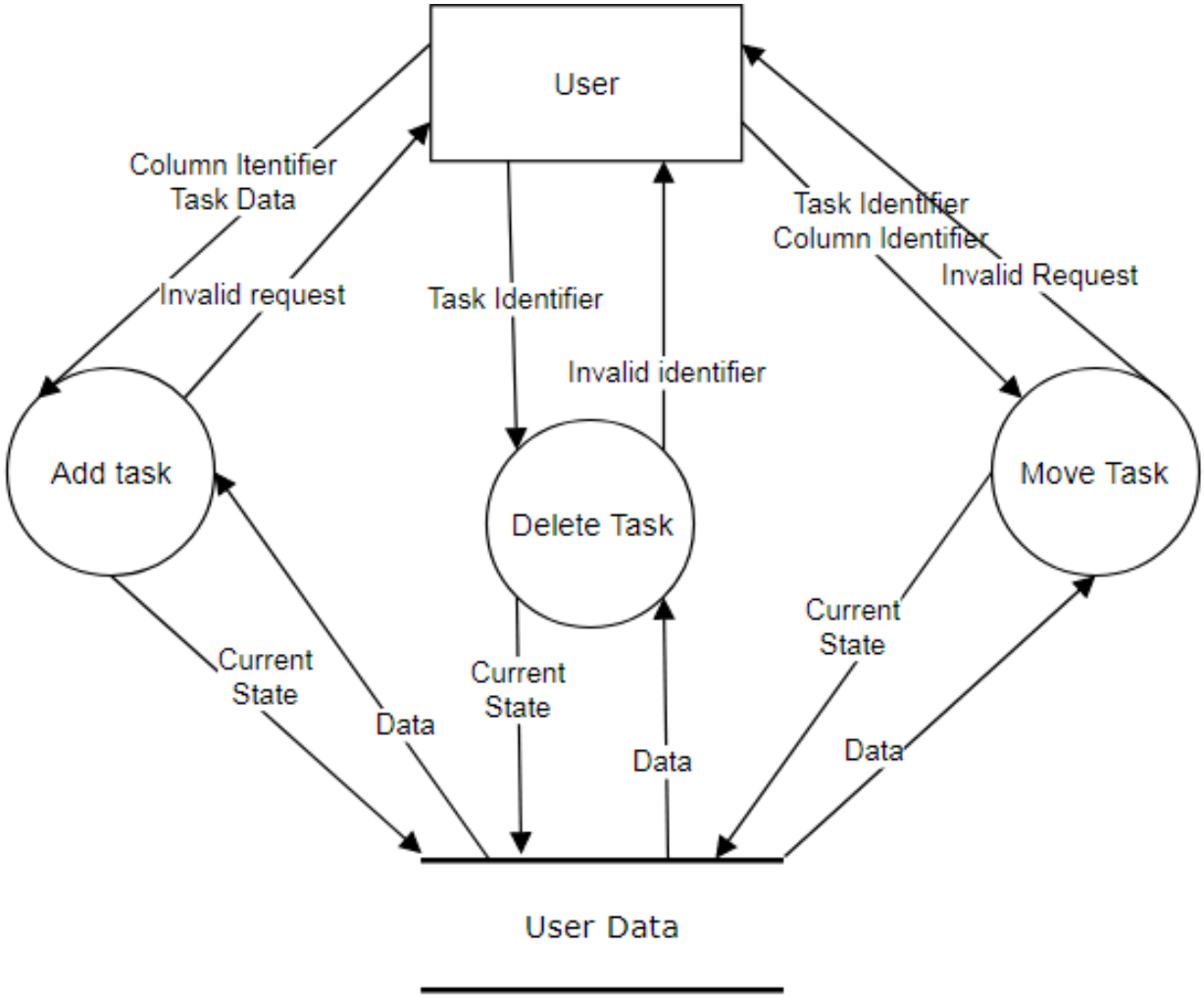


Рис. 3.15. Діаграма потоку даних

Висновки по розділу

Була створена зручна архітектура інтерфейсу для користувачів. Одним з важливих пунктом можливість реєстрації і входом під своїм особистим обліковим записом це стало можливим використовуючи підключену базу даних однак спочатку планувалося зробити через локальне сховище браузера. Так само головний інтерфейс містить в собі всі необходмимі функції для зручної

роботи. Я використовував найсучасніші технології для реалізації моєї дипломної роботи

ВИСНОВКИ

Дана кваліфікаційна робота містить методику та програмний застосунок для управління pre-sales етапом розробки ПЗ, починаючи з етапу аналізу предметної області і актуальності такого дослідження, включаючи етапи створення специфікації вимог і проектування архітектури програмного засобу, а також закінчуючи реалізацією такого програмного засобу.

Даний програмний продукт може використовуватись аналітиками, менеджерами, розробниками, так як має відносно слабкі межі всередині свого призначення, а саме - управління pre-sales етапом розробки ПЗ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. BABOK. Business Analysis Body of Knowledge. – V3. – 2015. – 502 p.
2. Karl Wiegers and Joy Beatty. Software Requirements, Third Edition. – 2013. – 637 p.
3. DAMA-DMBOK: Data Management Body of Knowledge. – 2017. – 590 p.
4. A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Seventh Edition and The Standar. – 2021. – 274 p.
5. Лавріщева К.М. Програмна інженерія. Підручник. – К.: Академперіодика, 2008. – 320 с.
6. Test-Driven Development with React and TypeScript: Building Maintainable React Applications 2nd ed. Edition, 2023.
7. React 18 Design Patterns and Best Practices: Design, build, and deploy production-ready web applications with React by leveraging industry-best practices 4th ed. Edition, 2023.
8. React Hooks in Action: With Suspense and Concurrent Mode, 2021.
9. Software Architecture: The Hard Parts. Modern Trade-Off Analyses for Distributed Architectures. 1st Ed., 2021.
10. Software Architecture in Practice (SEI Series in Software Engineering) 4th Edition, 2022.
11. Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series) 1st Edition, 2018.
12. Agile Software Architecture Aligning Agile Processes and Software Architectures, 2017.