Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування закладу вищої освіти)

Навчально-науковий інститут інформаційних технологій і робототехніки
(повне найменування інституту, назва факультету (відділення))

Кафедра автоматики, електроніки та телекомунікацій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи

магістр
(ступінь вищої освіти)

на тему **Розробка інформаційної системи підприємства на основі web-технологій**

Виконав: студент 6 курсу, групи 601ТТ
спеціальності 172 «Електроенергетика, електротехніка та електромеханіка
(шифр і назва напряму підготовки, спеціальності)

Кальченко С.Ю.
(прізвище та ініціали)

Керівник Штомпель М.А.
(прізвище та ініціали)

Рецензент Жученко О.С.
(прізвище та ініціали)

Полтава - 2024 рік

Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Інститут Навчально-науковий інститут інформаційних технологій і робототехніки
Кафедра Автоматики, електроніки та телекомунікацій
Ступінь вищої освіти Магістр
Спеціальність 172 «Телекомунікації та радіотехніка»

ЗАТВЕРДЖУЮ
Завідувач кафедри автоматики,
електроніки та телекомунікацій
_____О.В. Шефер
Шефер
“ 04 ” 09 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ
Кальченко Сергій Юрійович

1. Тема проекту (роботи) **«Розробка інформаційної системи підприємства на основі web-технологій»**
керівник проекту (роботи) Штомпель Микола Анатолійович, д.т.н., професор
затверджена наказом вищого навчального закладу від “___”___2023 року № ____

2. Строк подання студентом проекту (роботи) 13.12.2023 р.

3. Описання поточного стану інформаційної системи підприємства та ідентифікація недоліків у її функціонуванні. Визначення завдань для розробки інформаційної системи підприємства на основі web-технологій з метою оптимізації бізнес-процесів. Розробка концепції інформаційної системи, враховуючи вимоги підприємства та особливості його діяльності. Визначення потреб у технічних ресурсах, обладнанні та програмному забезпеченні для реалізації web-технологій. Розробка системи керування доступом та безпеки інформації на основі web-технологій. Вибір інструментів та технологій для створення ефективного веб-інтерфейсу, який відповідає потребам користувачів. Реалізація функціоналу інформаційної системи для управління підприємством, включаючи облік ресурсів, фінансовий облік, керування персоналом та інші аспекти. Тестування та оптимізація розробленої інформаційної системи на основі web-технологій. Визначення якості та надійності роботи системи під різними умовамита обставинами. Висновки по роботі інформаційної системи, оцінка досягнутих результатів та рекомендації щодо подальших можливостей вдосконалення.

4. Дата видачі завдання 02.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів магістерської роботи	Термін виконання етапів роботи			Примітка (плакати)
1	Аналіз вимог розробка інформаційної системи	11.10.23		15%	Пл. 1
2	Проектування системи	18.10.23	I	30%	Пл. 2
3	Розробка серверної частини (backend)	25.10.23		40%	Пл. 4
4	Тестування і валідація	14.11.23		50 %	Пл. 5
5	Розробка серверної частини інформаційної системи	21.11.23	II	60%	Пл. 6
6	Сервер інформаційної системи	28.11.23		70%	Пл. 7,8
7	Розробка клієнтської частини інформаційної системи	13.12.23	III	100%	Пл. 9

Магістрант _____ Кальченко С.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Штомпель М.А.
(підпис) (прізвище та ініціали)

Вступ

У сучасному діловому світі, що швидко змінюється, ключовим фактором успіху є вміння ефективно використовувати інноваційні технології для вдосконалення внутрішніх процесів і надання якісних послуг клієнтам. Інформаційні системи, розроблені на основі передових веб-технологій, визначають новий стандарт у сфері автоматизації та оптимізації бізнес-процесів.

Метою даної магістерської роботи є створення та впровадження інформаційної системи, яка відповідає потребам та завданням нашого підприємства. Розроблено комплексний підхід до вдосконалення роботи організації, який охоплює всі аспекти від внутрішніх комунікацій до взаємодії з клієнтами.

Зокрема, це дослідження зосереджено на таких аспектах:

Розробка технічної архітектури:

Він орієнтований на побудову високоефективної та масштабованої системи, що забезпечує швидкий та безперебійний обмін даними між усіма складовими підсистемами.

Забезпечення безпеки даних:

Для захисту конфіденційності та цілісності даних, а також для забезпечення надійного доступу враховуються сучасні стандарти безпеки.

Інтеграція інтерфейсу користувача:

Розроблено зручний та інтуїтивно зрозумілий інтерфейс для кінцевих користувачів, спрямований на максимальне спрощення робочого процесу.

Оптимізація бізнес-процесів:

Вивчені та впроваджені передові методи та прийоми підвищення ефективності та продуктивності роботи різних підрозділів підприємства.

Ця робота є результатом глибоких досліджень, етапів аналізу та розробки, проведених мною особисто, з метою створення інтегрованої та інноваційної інформаційної системи. Продовжуючи читати цей звіт, ви

1.Розробка інформаційної системи підприємства на основі web-технологій

1.1 Аналіз вимог розробка інформаційної системи

Аналіз вимог є критичним етапом розробки інформаційної системи підприємства на основі web-технологій. Цей етап допомагає зрозуміти, які функції та характеристики системи необхідно реалізувати, а також визначити, як система має задовольняти потреби користувачів та бізнес-вимоги. Ось деякі ключові елементи аналізу вимог:

1. Збір вимог:

- Провести співбесіди з представниками різних відділів та користувачами, щоб з'ясувати їхні потреби та вимоги до системи.

- Виокремити основних зацікавлених сторін та їхні ролі в системі.

2. Документування вимог:

- Створення документа, який містить всі вимоги до системи. Це може бути специфікація вимог або інший документ, який служить як основний джерело інформації для розробки.

3. Формалізація вимог:

- Визначення пріоритетів вимог і їхній важливості для бізнесу.

- Визначення та документування умов відміни або зміни вимог під час розробки.

4. Аналіз функціональних вимог:

- Визначення, як система повинна виконувати конкретні завдання та операції.

- Розробка діаграм, які ілюструють функціональність системи, такі як діаграми потоку даних чи діаграми варіантів використання.

5. Аналіз нефункціональних вимог:

- Визначення нефункціональних характеристик системи, таких як продуктивність, безпека, надійність та інші.

- Визначення обмежень, які повинні бути враховані під час розробки.

6.Валідація вимог:

- Перевірка вимог на відповідність потребам користувачів і бізнес-цілям.

- Визначення, як будуть перевірятися вимоги під час розробки та тестування.

7. Залучення зацікавлених сторін:

- Забезпечення зв'язку з представниками відділів та користувачами на протязі всього процесу розробки для уточнення та вдосконалення вимог.

8. Документування змін вимог:

- Створення процедур для додавання, зміни та видалення вимог в процесі розробки, і як це впливає на розклад і бюджет проекту.

Аналіз вимог - це постійний процес, який може включати в себе ітерації та зміни вимог під час розробки системи. Детальний та відповідальний аналіз вимог є ключовим кроком для створення успішної інформаційної системи, яка відповідає потребам підприємства і користувачів.

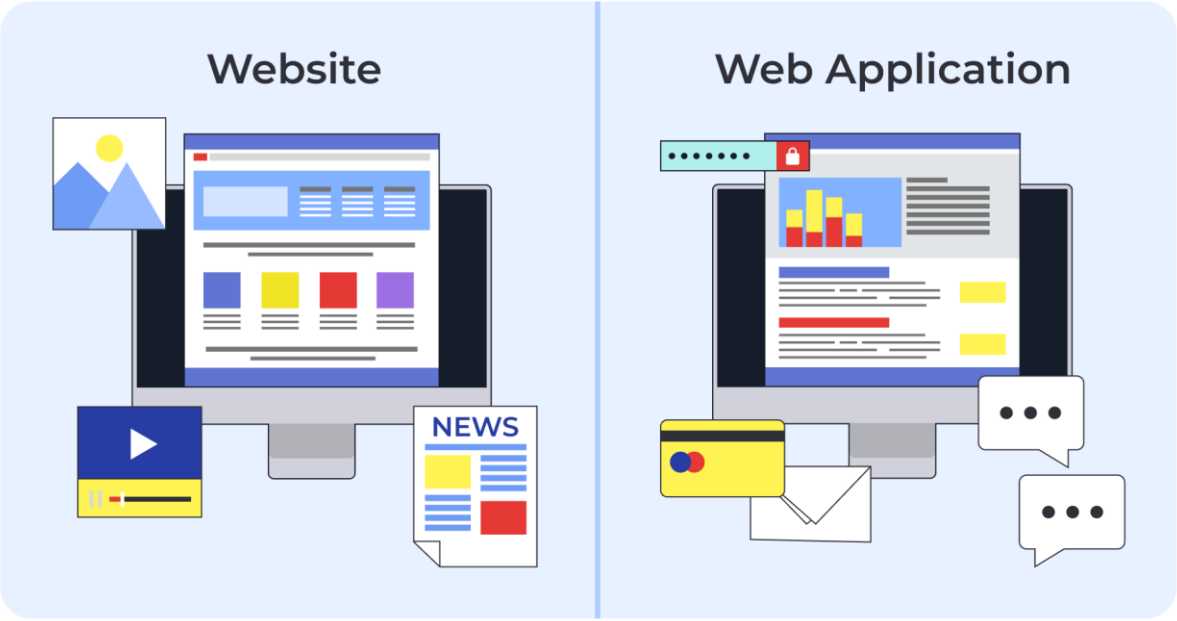


Рис. 1.1 Відмінності між website та web application

1.2 Проектування системи

Проектування системи є критичним етапом у розробці інформаційної системи підприємства на основі web-технологій. Цей процес передбачає створення детального плану системи та визначення архітектури, технологій і інфраструктури, необхідних для реалізації задуманих функцій та вимог. На цій сторінці ми розглянемо ключові кроки проектування інформаційної системи.

1. Архітектура системи:

- Визначення загальної структури системи.
- Розробка моделі даних та взаємодії між компонентами.

2. Вибір технологій:

- Вибір мов програмування, фреймворків та інших інструментів розробки.
- Оцінка сумісності обраних технологій з бізнес-вимогами.

3. Інфраструктура:

- Розгляд можливостей для хостингу системи (на власних серверах, у хмарних сервісах тощо).
- Визначення вимог до обладнання та мережевої інфраструктури.

4. Безпека системи:

- Розробка стратегії захисту даних та доступу до системи.
- Впровадження механізмів автентифікації та авторизації.

5. Шкалювання та продуктивність:

- Планування для масштабованості системи зростанням обсягу даних та користувачів.
- Оптимізація швидкодії системи та реакції на запити.

6. Резервне копіювання та відновлення даних:

- Розробка стратегії регулярного резервного копіювання даних.
- План відновлення системи в разі непередбачуваних випадків.

7. Масштабованість:

- Розгляд можливостей розширення функціоналу системи в майбутньому.

- Забезпечення гнучкості для додавання нових функцій.

8. Документація проекту:

- Створення технічної документації, яка описує архітектуру, конфігурацію та інші ключові аспекти системи.

9. Завдання для розробників:

- Розподіл функціональних завдань серед розробників.

- Планування робочих ітерацій для розробки.

10. Терміни та бюджет:

- Визначення часових рамок для проекту та розподіл бюджету між різними етапами.

Проектування системи визначає основний каркас та стратегію розробки. Вірно спроектована система має бути готовою до ефективної реалізації наступних етапів, таких як розробка та тестування. Деталізоване проектування допомагає забезпечити успішну реалізацію вашої інформаційної системи.

1.3 Проектування інтерфейсу користувача

Рис. 1.2 Процес проектування інтерфейсу

Приклад створення макету показаний на рисунку 4.1

Інтерфейс користувача (UI) є важливою складовою будь-якої інформаційної системи на основі web-технологій, оскільки він визначає спосіб взаємодії користувача з системою. На цій сторінці ми розглянемо ключові аспекти проектування інтерфейсу користувача.

1. Визначення цільової аудиторії:

- Аналіз та визначення характеристик користувачів, їх потреб і очікувань від системи.

- Створення персонажів користувачів для кращого розуміння аудиторії.

2. Розробка інформаційної архітектури:

- Структурування та організація інформації на веб-сайті або додатку.

- Створення дерева навігації та карт сайту.

3. Прототипування інтерфейсу:

- Створення прототипів веб-сторінок або екранів додатку, які відображають основний функціонал.

- Тестування прототипів на користувачах для збору фідбеку.

4. Дизайн інтерфейсу:

- Розробка естетичного та функціонального дизайну веб-сторінок або екранів додатку.

- Вибір кольорової палітри, шрифтів, іконок і графіки.

5. Адаптивний дизайн:

- Забезпечення того, що інтерфейс коректно відображається на різних пристроях та розмірах екранів (респонсивний дизайн).

- Тестування на мобільних пристроях, планшетах та комп'ютерах.

6. Взаємодія з користувачем:

- Розробка логіки взаємодії та анімацій, які полегшують користувачеві використання системи.

- Додавання інтерактивних елементів, таких як кнопки, форми та вікна сповіщень.

7. Тестування і валідація інтерфейсу:

- Проведення тестів інтерфейсу для перевірки на відповідність дизайну та функціональності.

- Збір фідбеку від користувачів та внесення змін на основі результатів тестів.

8. Доступність:

- Забезпечення того, щоб інтерфейс був доступний для всіх користувачів, включаючи людей з обмеженими можливостями.

9. Документація інтерфейсу:

- Створення документації, яка описує структуру і функціональність інтерфейсу для розробників і тестувальників.

10. Ітеративний процес:

- Проектування інтерфейсу - це ітеративний процес, який може включати в себе кілька раундів прототипування та вдосконалення.

Проектування інтерфейсу користувача є ключовим етапом, оскільки від нього залежить сприйняття та зручність використання системи

користувачами. Грамотно розроблений інтерфейс може покращити користувальницький досвід і сприяти успішності проекту.

1.4 Розробка серверної частини (backend)

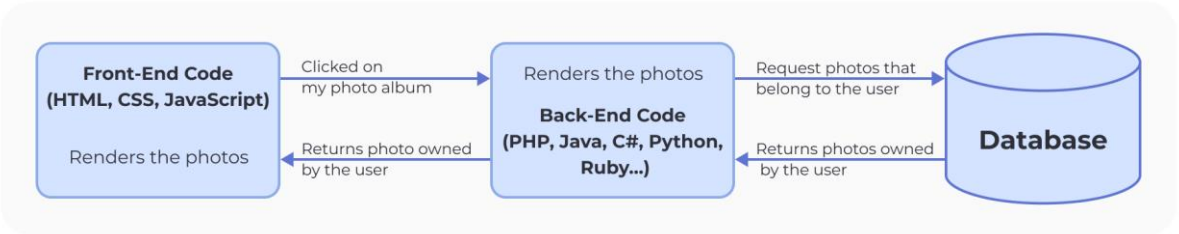


Рис. 1.3 Приклад взаємодії Front-End та Back-End

Розробка серверної частини, або backend, є однією з найважливіших складових процесу створення інформаційної системи підприємства на основі web-технологій. Backend відповідає за обробку бізнес-логіки, зберігання даних та взаємодію з базою даних.

1. Вибір технологій та стеку:

- Визначення мови програмування та фреймворків для розробки backend.

- Вибір системи управління базами даних (наприклад, MySQL, PostgreSQL, MongoDB).

2. Архітектура серверної частини:

- Розробка архітектури системи, включаючи визначення серверних компонентів та їх взаємодію.

- Розподіл завдань та ролей між серверними компонентами.

3. Розробка бізнес-логіки:

- Написання коду, який виконує бізнес-логіку системи, включаючи обробку запитів від користувачів та взаємодію з базою даних.

- Реалізація алгоритмів обчислень та операцій над даними.

4. Безпека серверної частини:

- Захист від потенційних загроз, таких як атаки на внедрення SQL, перехоплення даних тощо.

- Використання методів шифрування та аутентифікації для забезпечення безпеки даних.

5. API розробка:

- Створення API (Application Programming Interface) для взаємодії з клієнтською частиною (frontend) та іншими системами.

- Документування API для зручності розробників.

6. Управління сеансами та сесіями:

- Реалізація механізмів управління сесіями користувачів та зберігання даних сесій.

7. Тестування та налагодження:

- Виконання модульних, інтеграційних та функціональних тестів для перевірки функціональності backend.

- Виявлення та виправлення помилок.

8. Масштабування:

- Розгляд можливостей масштабування серверної частини для підтримки зростання обсягів даних та користувачів.

9. Документація та підтримка:

- Створення технічної документації для backend частини системи.

- Забезпечення підтримки та вдосконалення системи після впровадження.

10. Захист даних:

- Захист конфіденційних даних у базі даних та під час передачі через мережу.

Розробка серверної частини вимагає уважності до деталей та здійснюється з урахуванням багатьох аспектів, включаючи безпеку, продуктивність та масштабованість. Грамотно спроектований та реалізований backend є важливою передумовою успіху вашої інформаційної системи.

1.5 Розробка клієнтської частини (frontend)

Рис. 1.4 Мови програмування для Backend розробки та Front-End розробки

Front-End розробка містить все, з чим користувачі стикаються безпосередньо: кольори та стилі тексту, зображення, графіки та таблиці, кнопки, кольори та меню навігації.

На даний момент у клієнтській частині застосунку можна реалізувати і всю логіку та не використовувати Back-End. У такому підході є як свої плюси, так і мінуси. Але ви можете зустріти застосунки, де не буде бекенду, а дані будуть оброблятись і зберігатись у Front-End частині застосунку.

Структура, дизайн, поведінка та вміст всього, що відображається на екрані браузера при відкритті веб-сайтів, веб-застосунків або мобільних програм, реалізується **Front-End розробниками.**

Розробка клієнтської частини, або frontend, є однією з ключових складових створення інформаційної системи підприємства на основі web-технологій. Frontend відповідає за інтерфейс, який бачать та використовують користувачі системи. На цій сторінці ми розглянемо основні етапи розробки клієнтської частини.

1. Вибір технологій та стеку:

- Визначення основних технологій для розробки frontend (HTML, CSS, JavaScript) та фреймворків (наприклад, React, Angular, Vue.js).

- Вибір інструментів для роботи з дизайном та розробки (наприклад, Adobe XD, Figma)

2. Прототипування та дизайн:

- Створення макетів та дизайну інтерфейсу.

- Розробка прототипів для валідації концепції та збору фідбеку.

3. Розробка користувальницького інтерфейсу (UI):

- Створення HTML-структури та CSS-стилів для відображення інтерфейсу.

- Розробка компонентів і шаблонів для більшої гнучкості та повторного використання.

4. Програмування (JavaScript):

- Написання JavaScript-коду для реалізації функціональності, взаємодії з користувачем та обробки даних.

- Використання бібліотек та фреймворків для прискорення розробки.

5. Оптимізація продуктивності:

- Оптимізація завантаження сторінок та взаємодії з сервером для швидкого реагування на запити користувачів.

- Кешування та зменшення кількості запитів до сервера.

6. Адаптивний дизайн:

- Забезпечення коректного відображення на різних типах пристроїв та розмірах екранів (респонсивний дизайн).

- Тестування на мобільних пристроях, планшетах та комп'ютерах.

7. Валідація та тестування:

- Перевірка правильності роботи функціональності та відповідність дизайну вимогам та специфікаціям.

- Виконання модульних та інтеграційних тестів.

8. Документація та навчання користувачів:

- Створення документації для користувачів щодо використання інтерфейсу.

- Підготовка навчальних матеріалів та інструкцій.

9.Управління станом та даними:

- Зберігання та керування станом додатку та взаємодія з сервером через API. Приклад наведено в додатку на рисунку 4.4 та 4.5.

10. Забезпечення безпеки:

- Захист даних на клієнтській стороні, включаючи обробку валідації та санітарізацію введених даних.

11. Інтеграція з сервером:

- Взаємодія з сервером для отримання та відправлення даних через API.

12. Міжнародна локалізація та локалізований контент:

- Підтримка мовних версій та локалізованих варіантів контенту для різних регіонів.

Розробка клієнтської частини - це процес, який вимагає уваги до деталей та спрямований на створення зручного та ефективного інтерфейсу для користувачів. Грамотно реалізована frontend частина дозволить користувачам легко взаємодіяти з системою та користуватися її функціональністю.

1.6 Розробка бази даних

База даних є однією з найважливіших складових будь-якої інформаційної системи підприємства на основі web-технологій. Вона відповідає за зберігання, організацію та доступ до даних, які використовуються в системі. На цій сторінці ми розглянемо ключові аспекти розробки бази даних.

1. Вибір системи управління базами даних (СУБД):

- Визначення СУБД, яка найкраще підходить для потреб вашого проекту (наприклад, MySQL, PostgreSQL, Microsoft SQL Server, MongoDB).

- Вибір між реляційними та нереляційними базами даних в залежності від типу даних та вимог.

2. Проектування схеми бази даних:

- Створення схеми, яка визначає структуру та взаємозв'язки між таблицями бази даних.

- Визначення ключів, індексів та обмежень для забезпечення цілісності даних.

3. Розробка SQL-запитів:

- Написання SQL-запитів для створення таблиць, вставки, оновлення та видалення даних.

- Визначення запитів для отримання і аналізу даних з бази даних.

4. Оптимізація бази даних:

- Використання індексів для покращення швидкодії запитів.

- Моніторинг та налагодження бази даних для забезпечення продуктивності та масштабованості.

5. Безпека даних:

- Захист бази даних від несанкціонованого доступу та атак на безпеку (наприклад, впровадження SQL-ін'єкцій).

- Використання ролей та прав доступу для контролю доступу до даних.

6. Резервне копіювання та відновлення:

- Розробка стратегії регулярного резервного копіювання бази даних.

- Визначення процедур відновлення даних в разі аварій.

7. Міграції бази даних:

- Використання міграцій для управління змінами структури бази даних в ході розвитку проекту.

8. Інтеграція з backend і frontend:

- Взаємодія з серверною та клієнтською частинами системи через API для забезпечення обміну даними.

9. Складні запити та операції:

- Розробка складних SQL-запитів для виконання операцій, таких як звітність, агрегація даних тощо.

10. Моніторинг та підтримка:

- Встановлення систем моніторингу для виявлення і усунення проблем з базою даних.

- Підтримка та оптимізація бази даних після впровадження системи.

Розробка бази даних вимагає глибокого розуміння потреб системи та впевненості в її надійності та безпеці. Грамотно спроектована та оптимізована база даних грає ключову роль у забезпеченні ефективності та надійності інформаційної системи.

1.7 Тестування і валідація

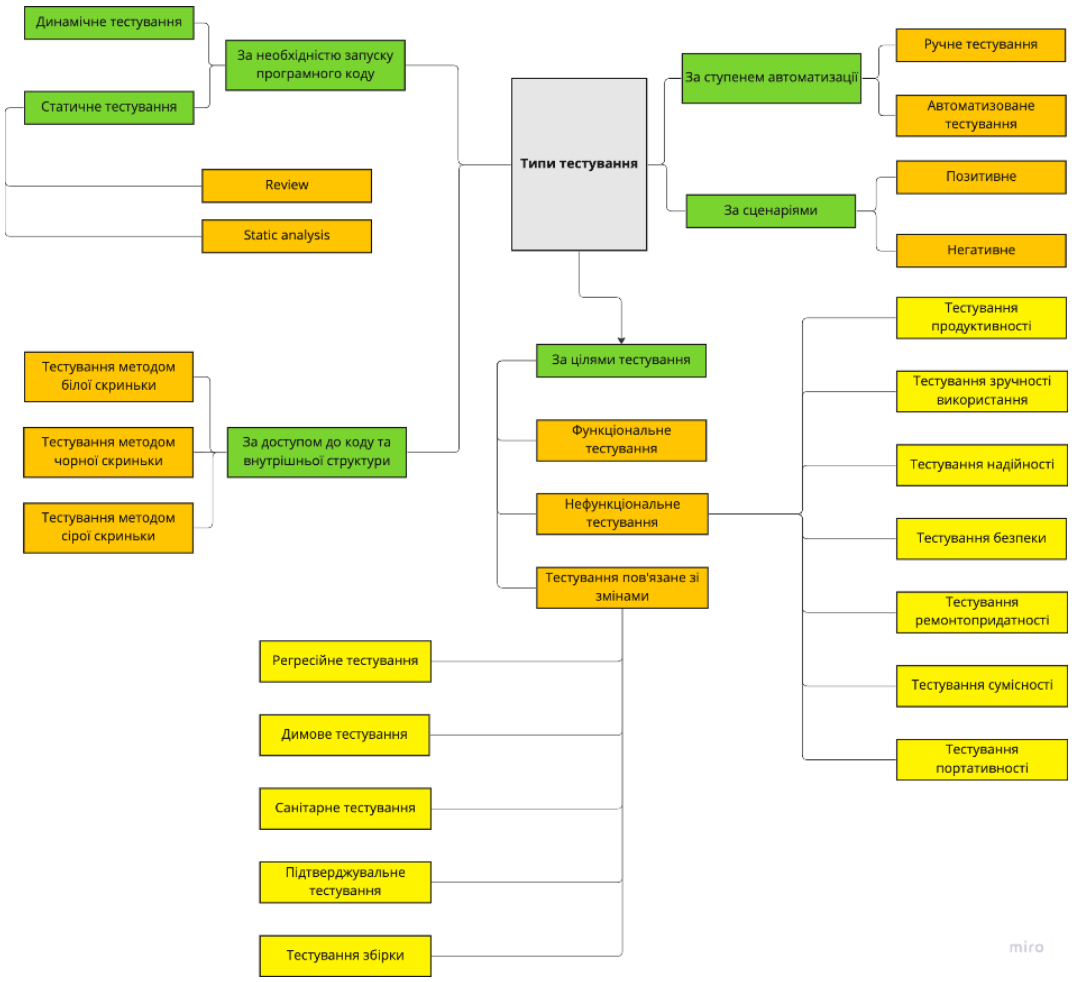


Рис. 1.5 Типи тестуання

Тестування і валідація є важливими етапами у розробці інформаційної системи на основі web-технологій. Ці процеси допомагають впевнитися в якості та надійності системи перед її впровадженням. На цій сторінці ми розглянемо ключові аспекти тестування і валідації.

1. Планування тестування:

- Створення плану тестування, включаючи обсяг тестування, ресурси та графік.

- Визначення цілей тестування та критеріїв прийняття.

2. Типи тестування:

- Модульне тестування: перевірка окремих компонентів та функцій системи.

- Інтеграційне тестування: перевірка взаємодії між компонентами.

- Функціональне тестування: перевірка відповідності функціональності вимогам.

- Тестування продуктивності: вимірювання швидкодії та масштабованості системи.

- Тестування безпеки: виявлення та виправлення потенційних загроз безпеці даних.

- Тестування на мобільних пристроях: перевірка на різних платформах та розмірах екранів.

3. Автоматизація тестування:

- Використання автоматичних тестових скриптів для прискорення процесу тестування.

- Автоматичне виконання регресійних тестів для виявлення помилок після змін в коді.

4. Генерація тестових даних:

- Створення тестових наборів даних для різних сценаріїв тестування.

- Використання реальних та синтетичних даних для тестів.

5. Тестові сценарії та сценарії випробувань:

- Розробка деталей тестових сценаріїв, включаючи послідовність дій користувача та очікувані результати.

- Тестування різних варіантів взаємодії з системою.

6. Валідація відповідно до вимог:

- Перевірка того, що система відповідає всім вимогам, визначеним на початковому етапі проекту.

- Виявлення та виправлення відхилень від вимог.

7. Тестування безпеки:

- Проведення тестів на виявлення вразливостей та атак на безпеку.

- Виправлення виявлених проблем безпеки.

8. Звітність та фіксація помилок:

- Документування результатів тестування та створення звітів про помилки.

- Прийняття заходів для виправлення помилок та перевірка їх виправлення.

9. Відстеження та контроль версій:

- Використання систем контролю версій для відстеження змін в коді та тестах.

- Забезпечення синхронізації версій коду між розробниками.

10. Аналіз покриття тестами:

- Визначення рівня покриття тестами коду та в

1.8 Впровадження

Впровадження інформаційної системи є фінальним етапом у процесі її розробки та готовності до використання підприємством. Правильне впровадження гарантує успішний запуск системи та її прийняття користувачами. На цій сторінці ми розглянемо ключові аспекти впровадження.

1. Планування впровадження:

- Створення детального плану впровадження, включаючи дати, кроки та відповідальних осіб.

- Визначення необхідних ресурсів та бюджету для впровадження.

2. Тестування перед впровадженням:

- Проведення остаточного тестування системи перед впровадженням для переконанняся в її готовності.

- Виправлення всіх виявлених помилок та проблем.

3. Підготовка користувачів:

- Надання навчання та підтримки користувачам системи.

- Створення навчальних матеріалів та інструкцій.

4. Впровадження поетапно:

- Виконання впровадження поетапно для зменшення ризику та навантаження на підприємство.

- Запуск системи на обмеженій аудиторії або в обмеженому режимі.

5. Моніторинг та підтримка:

- Надання технічної підтримки та моніторингу системи після впровадження.

- Виявлення та виправлення можливих проблем у реальному часі.

6. Заохочення користувачів:

- Залучення користувачів до активного використання системи та збору їхнього фідбеку.

- Впровадження заохочення та мотиваційних заходів для користувачів.

7. Аналіз та оцінка впровадження:

- Проведення оцінки ефективності та відповідності системи бізнес-вимогам після впровадження.

- Аналіз результатів та прийняття рішень щодо подальших кроків.

8. Регулярні оновлення та підтримка:

- Встановлення процедур регулярних оновлень та вдосконалень системи.

- Підтримка та впровадження змін з урахуванням потреб бізнесу.

9. Завершення проекту:

- Формальне завершення проекту впровадження та передача системи у повний використання підприємством.

- Оцінка та документування досягнень та вивчених уроків після завершення проекту.

Впровадження інформаційної системи є важливим моментом у життєвому циклі проекту. Правильно підготовлене та проведене впровадження сприяє успішному функціонуванню системи та задоволенню потреб користувачів та підприємства.

1.9 Моделі життєвого циклу розробки ПЗ

Життєвий цикл розробки ПЗ англійською називають Software Development Lifecycle Models або використовують абревіатуру SDLC.

Модель життєвого циклу розробки програмного забезпечення (SDLC) — це сукупність різних видів активностей, які виконують на кожному етапі процесу розробки програмного забезпечення, ці активності співвідносяться між собою логічно і хронологічно.

Кожна модель SDLC потребує різних підходів до тестування.

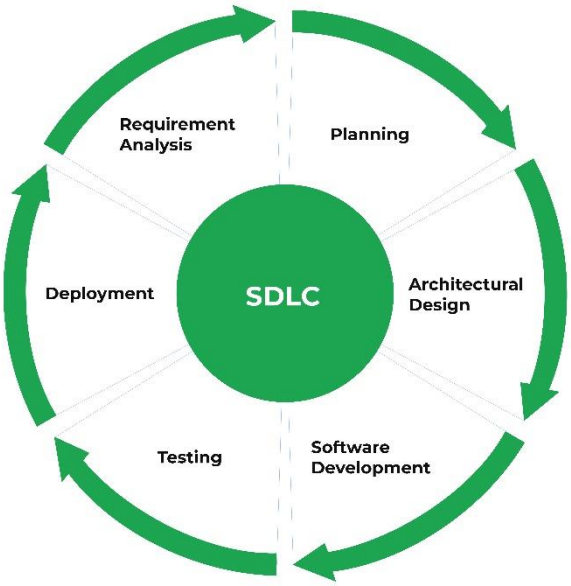


Рис. 1.6 Модель життєвого циклу

Розробка та тестування програмного забезпечення

Модель життєвого циклу, прийнята для проєкту, матиме великий вплив на тестування.

Проте є декілька характеристик хорошого тестування в будь-якій моделі SDLC:

- Для кожної активності з розробки існує відповідна активність тестування.
- Кожен рівень тестування має цілі, специфічні для цього рівня.
- Аналіз та проєктування тестів для заданого рівня тестування розпочинаються під час відповідної активності з розробки.
- Тестувальники беруть участь в обговореннях, щоб визначити та уточнити вимоги та дизайн, а також беруть участь у перевірці робочих продуктів (наприклад, вимог, дизайну, user-story тощо).

Найпоширенішими моделями SDLC є **послідовна модель розробки** та **ітеративна та інкрементальна модель розробки.**

Рис. 1.7 Послідовна модель розробки

Послідовна модель розробки — це модель, в якій процес розробки ПЗ є лінійним, послідовним потоком активностей.

Це означає, що будь-яка фаза у процесі розробки має починатися після завершення попередньої, одна за одною чи one by one.

Найбільш відомі моделі послідовної розробки:

- Waterfall;
- V-модель;

Основні особливості Waterfall-моделі:

- Активності з розробки (наприклад, аналіз вимог, дизайн, кодування) виконуються одна за одною (кожен наступний етап починається лише після завершення попереднього).

Тестування проводиться після завершення етапу кодування (розробки).Waterfall з англійської перекладається як “водоспад”. Тож і модель життєвого циклу розробки ПЗ схожа на водоспад процесів та активностей. Схема нижче демонструє послідовну Waterfall-модель SDLS:

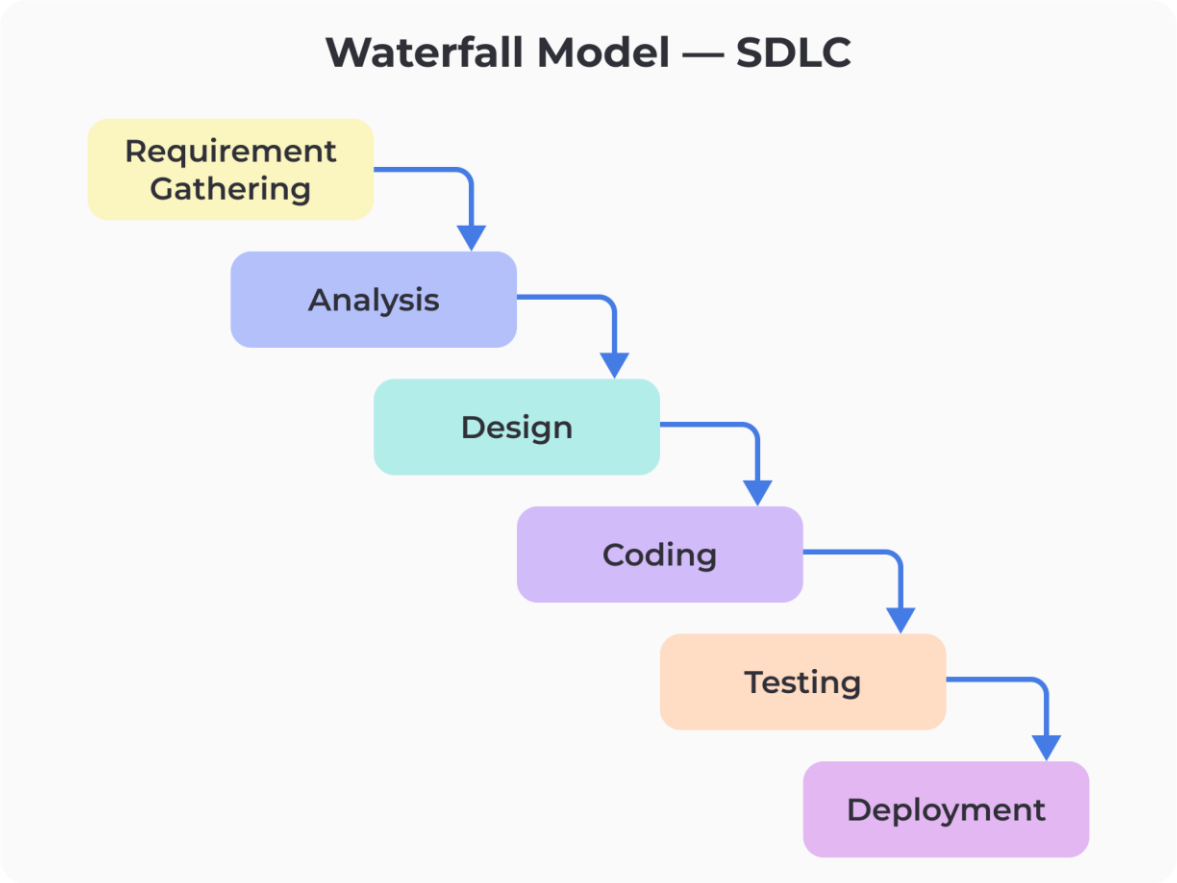


Рис. 1.8 Waterfall модель

А ось тобі приклади того, коли Waterfall модель можна і коли не можна застосовувати:

- *На заводі, який виготовляє заклепки для фюзеляжу літака, оператори проводять перевірки для оцінки заклепок на конвеєрній стрічці. Ця оцінка може виявити відсоток дефектних заклепок. Зазвичай цей відсоток невеликий і не призводить до відмови від усієї партії заклепок. Отже, більшість продукту може бути випущена. Для цього процесу можна застосувати Waterfall модель, оскільки ця модель передбачає тестування в кінці процесу, як фінальний етап.*
- *Тепер розглянемо той же літак, але продуктом є програмне забезпечення, що управляє дисплеєм, яким користується екіпаж. Що буде далі, якщо в момент тестування буде виявлено багато дефектів? Чи*

можемо ми випустити лише частини системи? Очевидно, що не можемо, адже ПЗ літака має функціонувати згідно зі встановлених критеріїв. Тому в такому разі обирають іншу модель життєвого циклу.

Основні особливості V-моделі

Наступна модель SDLC — це **V-модель.** За цією моделлю тестування необхідно починати якомога раніше у життєвому циклі. Основна ідея V-моделі у тому, що завдання розробки та тестування є відповідними активностями однакової важливості. Дві гілки літери V символізують це.

V-model для розробки ПЗ:

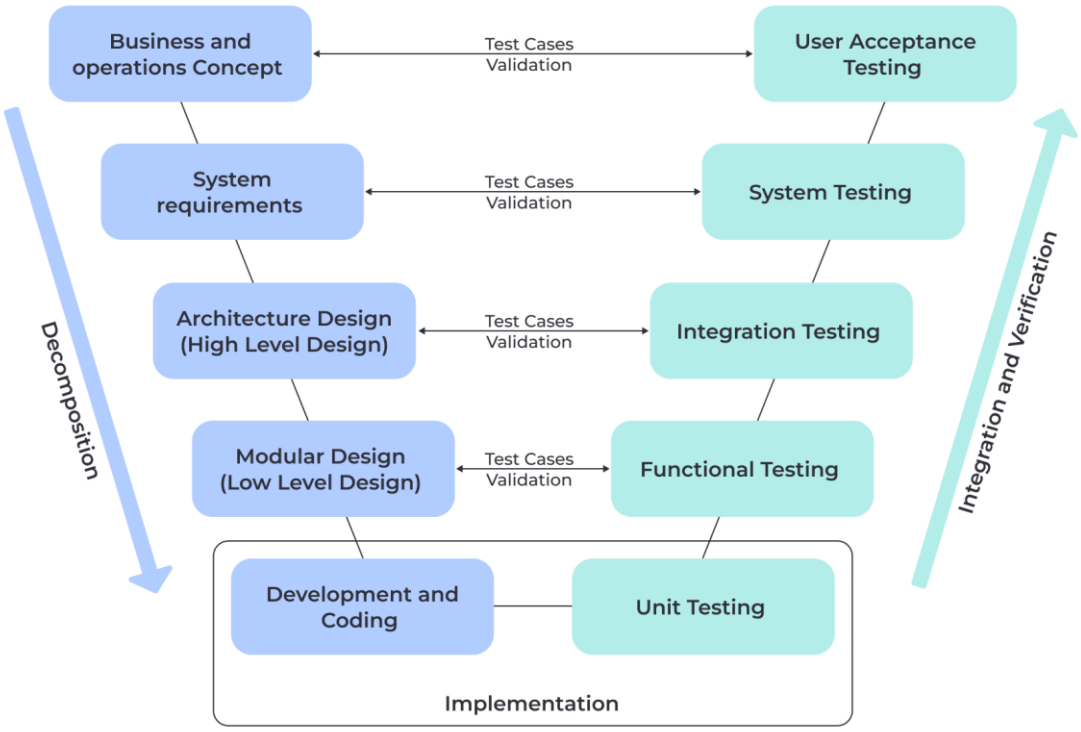


Рис. 1.9 V-модель

Активності в лівій частині моделі — це активності, відомі з моделі Waterfall, і вони орієнтовані на вимоги:

- **Специфікація вимог** (Requirement specification) — облік потреб користувачів.
- **Функціональна специфікація** (Functional specification) — визначення функцій, необхідних для задоволення потреб користувача.
- **Технічна специфікація** (Technical specification) — технічний дизайн функцій, зазначених у функціональній специфікації.
- **Специфікація програми** (Program specification) — докладний дизайн кожного модуля або блоку, який буде побудований з урахуванням необхідної функціональності.

Отже, ми з тобою розібрали послідовні моделі розробки ПЗ, а саме Waterfall-модель та V-модель. Час перейти до моделей розробки зовсім іншого типу: ітеративної та інкрементальної.

***І**теративна та інкрементальна модель розробки.*

Бачимо, що в послідовній моделі розробки міститься повний набір функцій і зазвичай потрібні місяці, а то й роки для поставки зацікавленим сторонам і користувачам готового продукту.

Інкремент — операція, що збільшує існуючу перемінну (в нашому випадку — це ПЗ чи окремий модуль ПЗ) на один крок.

Інкрементальні моделі розробки містять постановку вимог, дизайн, кодування (розробку) і тестування системи по частинах. Це означає, що функціональність ПЗ збільшується інкрементально. Тобто до того, що є постійно додається якась маленька частина (інкремент). Інкрементом може бути як і невелике поліпшення існуючої фічі, так і нова функціональність.

Incremental model:

- Є чіткі вимоги;
- Ділимо вимоги на збірки;
- Створюємо першу велику збірку і потім нарощуємо фічі, поки не буде готове все ПЗ.

Інкремент — це поступове збільшення на певну величину.

Якби Леонардо да Вінчі писав свою “Джоконду” за інкреметальною моделлю, то його процес виглядав би так:

Рис. 1.10 Інкрементна модель

Отже, під час **інкрементальної моделі** розробки, постійно додається якась невелика частина ПЗ. А що ж відбувається, коли застосовують **ітеративну модель** розробки? Давай розбиратися.

Ітеративна модель розробки здійснюється, коли групи функцій визначаються, проєктуються, створюються та тестуються разом у серії циклів, за фіксований відрізок часу. Кожна ітерація надає працююче програмне забезпечення.

Рис. 1.11 Ітеративна модель

Iterative model:

- Немає повних вимог;
- Крок за кроком нарощуємо функціонал;
- Головне, щоб кожна збірка була працездатною.

Якщо повернутися до аналогії із створенням картини то, за ітеративної моделлю, писати картину логічніше так: спочатку ескіз, потім перші грубі мазки і — фінальні делікатні штрихи. Таким чином, на кожному етапі є певна версія завершеної картини:

Рис. 1.12 Інкрементальна модель

Інкрементальна та ітеративна моделі розробки не мають чіткого work flow (робочого потоку). Як, наприклад, у моделі Waterfall, де етапи чітко

визначені: збір вимог —> аналіз вимог—> дизайн —> кодинг —> тестування і так далі. Тому для цих моделей потрібні спеціальні інструменти, які б дозволяли повноцінно користуватися перевагами інкрементальої чи ітеративної моделі розробки, та при цьому дозволяли б вкладатися в терміни та бюджет, виділений на розробку. І такі інструменти є — це гнучкі методи розробки ПЗ.

Гнучкі методи розробки ПЗ:

Інкрементальна та ітеративна моделі працюють за **гнучким підходом** до розробки ПЗ або **Agile software development** або просто **Agile.**

Agile — гнучка методологія розробки ПЗ, що ґрунтується на ітеративній розробці, головна перевага якої — забезпечення високої продуктивності розробки та швидке реагування на зміни.

Scrum та Канбан є двома видами Agile, які найчастіше використовують.

Scrum — кожна ітерація зазвичай буває відносно короткою (наприклад, дні або кілька тижнів), і додавання фіч, відповідно, невеликі. Наприклад, кілька поліпшень та/або дві або три нові фічі.

Канбан — реалізується з ітераціями фіксованої довжини або без них, по завершенню яких випускається або єдине доопрацювання, або функціональність або група функціональностей, об'єднаних разом.

Отже, гнучкі підходи до розробки ПЗ дозволяють ефективно керувати процесом розробки, збільшувати продуктивність команди та швидко пристосовуватися до змін.

Зазвичай тестувальник за свою кар’єру працює з різними проєктами та методологіями розробки. При переході, наприклад, з розробки банківського ПЗ до стартапу із розробки штучного інтелекту, QA-інженер неодмінно

зіткнеться зі зміною методології на гнучку. Розберемося, які будуть переваги та складності.

Переваги для тестувальників під час переходу до гнучкої методології розробки:

- Орієнтація на працююче програмне забезпечення та якісний код;
- Включення тестування як частини та відправної точки розробки програмного забезпечення (test-driven development);
- Доступність зацікавлених сторін бізнесу, щоб допомогти тестувальникам вирішувати питання про очікувану поведінку системи;
- Самоорганізовані команди, в яких вся група відповідає за якість і дає тестувальникам більше автономії у їх роботі;
- Простота дизайну, яку має бути легше перевірити.

Значні проблеми для тестувальників під час переходу до підходу гнучкої розробки:

- Тестувальники, які звикли працювати з добре задокументованими вимогами, розроблятимуть тести на основі іншого типу тестування — менш формального та схильного до змін. У маніфесті не йдеться, що документація більше не потрібна або вона не має цінності, але це часто інтерпретується саме так.
- Оскільки розробники більше проводять модульне тестування (unit testing), може скластися враження, що тестувальники не потрібні. Але модульне тестування та приймальне тестування, яке проводиться тільки представниками бізнесу, може упустити серйозні проблеми. Системне тестування з його ширшою перспективою та акцентом на нефункціональне тестування, а також безперервне (end-to-

end) функціональне тестування необхідне, навіть якщо воно не підходить для спринту.

- Роль тестувальника інша. Оскільки в гнучкій команді менше документації і більше особистої взаємодії, тестувальникам необхідно адаптуватися до цього стилю роботи, що для декого може бути складно. Тестувальники можуть діяти переважно як тренери з тестування як для зацікавлених сторін, так і для розробників, які можуть не мати достатніх знань про тестування.
- Хоча за одну ітерацію потрібно тестувати менше, ніж для всієї системи, існує постійний дефіцит часу і менше часу на те, щоби думати про тестування нових функцій.
- Оскільки кожне збільшення додається до існуючої робочої системи, регресійне тестування стає надзвичайно важливим, а автоматизація стає більш вигідною.

Приклад тестування API зображений на рисунку 4.2 та 4.3.

1.10 Висновок по розділу «Розробка інформаційної системи підприємства на основі web-технологій»

У результаті розгляду процесу розробки інформаційної системи для підприємства на основі web-технологій можна визначити важливі переваги та перспективи використання цього підходу. Розробка web-базованої системи виявляється важливим інструментом для оптимізації бізнес-процесів, підвищення продуктивності та покращення взаємодії всередині підприємства.

Однією з ключових переваг є доступність системи з будь-якого місця та пристрою з підключенням до мережі Інтернет, що полегшує роботу персоналу та забезпечує оперативний обмін інформацією. Крім того, використання web-технологій дозволяє підприємству зменшити витрати на

обладнання, оскільки користувачам не потрібні спеціальні програми, а система функціонує у веб-браузері.

Також слід відзначити можливість легкої масштабованості системи, додавання нових функцій та швидке внесення змін. Це важливо у змінному бізнес-середовищі, де підприємствам часто доводиться швидко адаптуватися до нових умов та вимог ринку.

Таким чином, розробка інформаційної системи на основі web-технологій є стратегічно важливим напрямком для сучасних підприємств, спрямованим на підвищення ефективності та конкурентоспроможності в умовах динамічного бізнес-середовища.

2. Теоретичні відомості клієнт-серверна архітектура

2.1 Клієнт-серверна архітектура

Архітектура клієнт-сервер - це спосіб організації програмного забезпечення, при якому два компоненти, клієнт і сервер, співпрацюють для виконання завдання. Клієнт - це програма, яка використовується користувачем для взаємодії з сервером. Сервер - це програма, яка обробляє запити клієнта і надає йому відповіді.

Архітектура клієнт-сервер має ряд переваг, зокрема:

- Розподілений контроль: Клієнт і сервер можуть бути розташовані на різних комп'ютерах, що дозволяє розосередити навантаження і підвищити доступність системи.
- Зручність використання: Клієнт-серверні програми часто простіші в використанні, ніж програми, які все обробляють на одному комп'ютері.
- Ефективність: Архітектура клієнт-сервер дозволяє оптимізувати використання ресурсів комп'ютера.

Приклади архітектури клієнт-сервер:

- Веб-браузер - клієнт, який використовується для взаємодії з веб-сервером.
- Електронна пошта - клієнт-серверна система, при якій клієнт (наприклад, Outlook) надсилає запити на сервер (наприклад, Gmail), який обробляє їх і надає відповіді.
- Файловий сервер - сервер, який надає доступ до файлів клієнтам.

Протоколи клієнт-сервер:

Протокол - це набір правил, які регламентують взаємодію між двома програмами. Протоколи клієнт-сервер визначають, як клієнти і сервери відправляють і отримують дані один від одного.

Деякі поширені протоколи клієнт-сервер:

- FTP - протокол передачі файлів, який використовується для перенесення файлів між комп'ютерами.
- SMTP - простий протокол передачі пошти, який використовується для відправлення електронних листів.
- HTTP - протокол передачі гіпертексту, який використовується для відображення веб-сторінок.

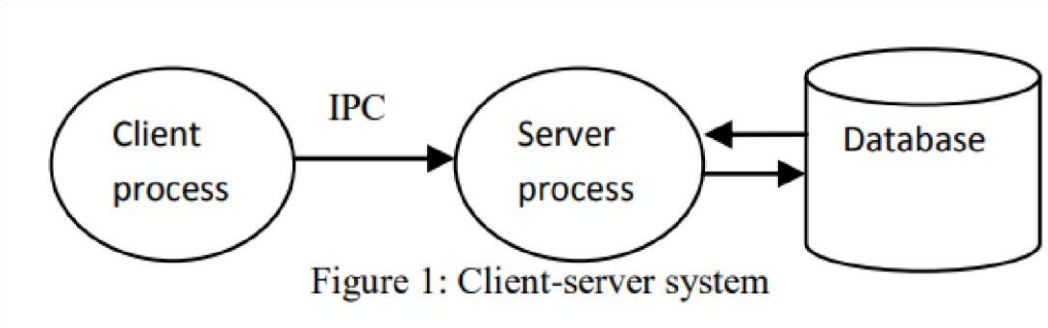


Figure 1: Client-server system

Рис. 2.1 Клієнт-серверна система

Клієнт-серверна архітектура ефективно розподіляє обов'язки з обробки даних між клієнтами, які зазвичай є персональними комп'ютерами, та сервером, який може бути робочою станцією високого класу або "мейнфреймом". Персональні комп'ютери мають значну обчислювальну

потужність, що дозволяє їм обробляти необроблені дані, отримані від сервера, і форматувати результат для подальшого виведення. Прикладні програми та процесори запитів можна зберігати та виконувати на персональних комп'ютерах.

Мережевий трафік у цій архітектурі зводиться до запитів на обробку даних, які надсилаються з персональних комп'ютерів на сервер баз даних, і передачі оброблених даних у відповідь на ці запити. Це призводить до значно меншого обсягу мережевого трафіку і теоретично поліпшує продуктивність.

Сучасні архітектури клієнт/сервер обмінюються повідомленнями через локальні мережі, і хоча деякі старі мережі Token Ring все ще використовуються, більшість сучасних локальних мереж базуються на стандартах Ethernet.

Клієнт-серверна архітектура подібна до традиційної централізованої архітектури в тому сенсі, що система управління базами даних (СУБД) розташована на одному комп'ютері. Багато сучасних "мейнфреймів" фактично діють як великі швидкі сервери через необхідність обробки об'ємних даних, але розподіл деяких обчислень змінився.

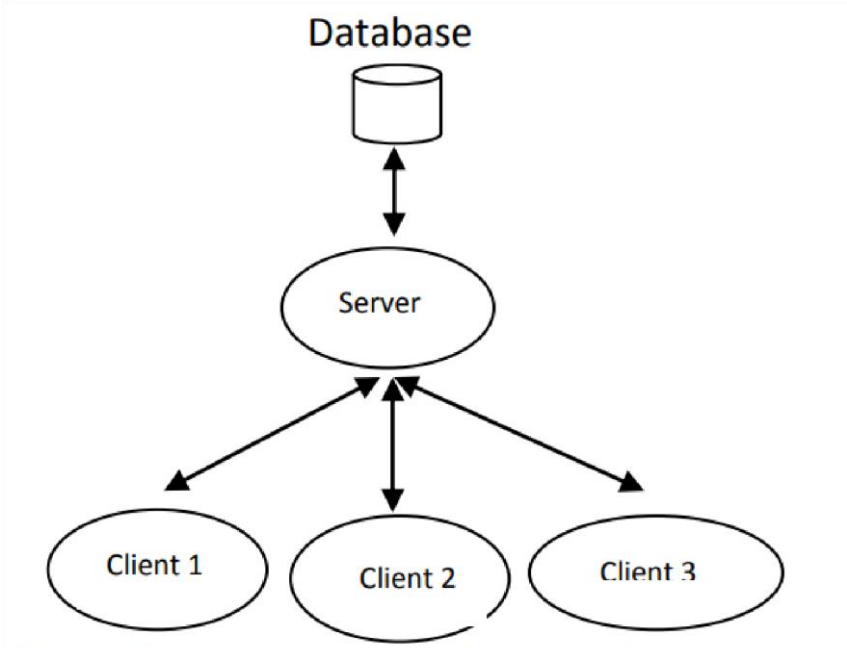


Figure 2: Interprocess communication among client and server

Рис. 2.2 Міжпроцесовий зв’язок між клієнтом і сервером

2.1.1 Переваги та недоліки клієнт-серверної архітектури

Покращений обмін даними: Дані зберігаються та обробляються на сервері, доступ до яких надається авторизованим користувачам, що спрощує обмін ресурсами між клієнтом і сервером.

Інтеграція служб: Кожен клієнт отримує зручний доступ до корпоративної інформації через інтерфейс робочого столу, усуваючи необхідність входу в термінальний режим або процесор.

Спільні ресурси між різними платформами: Додатки для клієнт-серверної моделі створюються незалежно від апаратної платформи, що дозволяє відкрите обчислювальне середовище.

Можливість обробки даних, незважаючи на місцезнаходження: Користувачі можуть безпосередньо взаємодіяти з системою, незалежно від їхнього місця або типу пристрою.

Простота обслуговування: Розподілену модель легко обслуговувати, замінювати, оновлювати та переносити сервери, залишаючи клієнтів без змін. Це забезпечується інкапсуляцією.

Безпека: Сервери забезпечують кращий контроль доступу та ресурсів, гарантуючи, що тільки авторизовані користувачі мають доступ та контроль над даними, а оновлення серверів ефективно адмініструються.

Недоліки:

Перевантажені сервери: Часті одночасні клієнтські запити можуть перевантажити сервер, призводячи до перевантаження трафіку.

Вплив централізованої архітектури: Критичні відмови сервера можуть призвести до невиконання клієнтських запитів, що піддає сумнівам надійність мережі.

З урахуванням цих фактів можна стверджувати, що переваги переважають недоліки, вказуючи на важливість та необхідність клієнт-серверної технології.

2.1.2 Класифікація серверів

Однорівнева архітектура: У цій архітектурі всі параметри конфігурації клієнт/сервер існують в одній системі, що може бути надійною, але складною у використанні через різні рівні та дисперсію даних для реплікації всієї роботи.

Приклад однорівневої архітектури:

У даному випадку, рівень презентації, бізнесу та доступу до даних організовано за допомогою єдиного програмного пакета, і всі дані зберігаються локально на комп'ютері. Додатки, такі як MP3-плеєр і MS Office, управляють усіма трьома рівнями, проте вони використовують однорівневу архітектуру, що означає, що вони виконують всі функції локально, без розділення на клієнтський та серверний компоненти.

Дворівнева архітектура:

Дворівнева архітектура клієнт-сервер - це спосіб організації програмного забезпечення, при якому два компоненти, клієнт і сервер, співпрацюють для виконання завдання.

Клієнт - це програма, яка використовується користувачем для взаємодії з сервером. Сервер - це програма, яка обробляє запити клієнта і надає йому відповіді.

У дворівневій архітектурі клієнт-сервер інтерфейс користувача зберігається на клієнтській системі, а база даних розміщується на серверній машині. Бізнес-логіка та логіка бази даних розташовані на клієнтському сервері, але їх необхідно підтримувати.

Існує два основних типи дворівнєвої архітектури клієнт-сервер:

- Архітектура тонкого серверного клієнта передбачає, що логіка даних та бізнес-логіка обробляються на клієнтському боці. Цей тип архітектури простий у реалізації та підтримці, але він може призвести до зниження

продуктивності системи, оскільки клієнтська система повинна виконувати більшу частину обробки.

- Архітектура товстого клієнтського сервера передбачає, що логіка даних та бізнес-логіка обробляються на серверній машині. Цей тип архітектури забезпечує більшу продуктивність, але він може бути складнішим у реалізації та підтримці.

Трирівнева архітектура:

У трьохрівневій архітектурі потрібне проміжне програмне забезпечення. Коли клієнтська машина відправляє запит на сервер, спершу цей запит приймається на проміжному рівні, а тільки потім направляється на сервер. Вся логіка даних і бізнес-логіка розміщуються в проміжному програмному забезпеченні, що забезпечує більшу гнучкість і відмінну продуктивність.

Багаторівнева архітектура:

Багаторівнева архітектура є розширеною формою трирівневої архітектури. У цій структурі рівні презентації, обробка додатків та управління даними знаходяться в ізольованому стані один від одного.

Переваги:

Забезпечує гнучкість та можливість використання додатків у багаторазових сценаріях.

Обмеження:

Реалізація багаторівневої архітектури ускладнена через складну структуру та необхідність компонування різних рівнів.

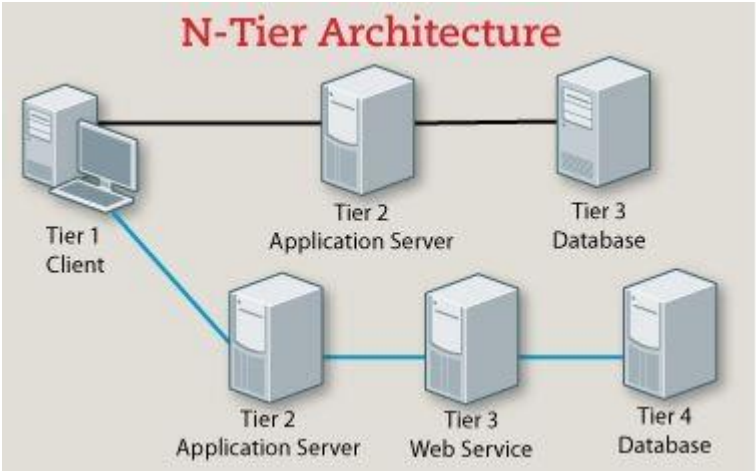


Рис.2.3 N-Tier архітектура

2.1.3 Приклади клієнт-серверної архітектури

• Веб-сервери:

Веб-сервери, такі як Apache або Microsoft IIS, є високопродуктивними комп'ютерними системами, що розміщують кілька веб-сайтів. Вони забезпечують доступ до цих сайтів через Інтернет, забезпечуючи швидкість передачі даних.

• Поштові сервери:

Сервери електронної пошти відправляють та отримують електронні листи. Адміністратор може управляти обліковими записами електронної пошти для будь-якого домену, розміщеного на сервері. Використовуються протоколи, такі як SMTP, IMAP та POP3.

• Файлові сервери:

Файлові сервери централізують зберігання файлів і надають доступ кільком термінальним системам для отримання файлів.

• DNS:

Сервер доменних імен (DNS) має велику базу даних IP-адрес, пов'язаних з хост іменами.

Ці сервери допомагають забезпечити доступ клієнтів (наприклад, ПК, смартфонів, ноутбуків) до різних ресурсів, таких як файли, електронні листи та веб-сайти.

2.1.4 Інші принципи взаємодії:

Peer-to-peer networking (P2P):

Поза класичною клієнт-серверною архітектурою існують інші підходи до розподілу інформації через мережу. Один з таких підходів - це Peer-to-peer networking (P2P).

P2P представляє собою розподілену архітектуру додатків, де завдання або робочі навантаження розділяються між вузлами. В одноранговій мережі кожен вузол виконує ролі як клієнта, так і сервера. Замість централізованого сервера, який розповсюджує інформацію, всі вузли функціонують як сервери, до яких можуть підключатися інші вузли.

Кожен вузол, використовуючи відповідну контрольну інформацію, визначає, до якого іншого вузла підключитися для отримання конкретної інформації. Такий підхід дозволяє вузлам взаємодіяти без необхідності централізованого посередника.

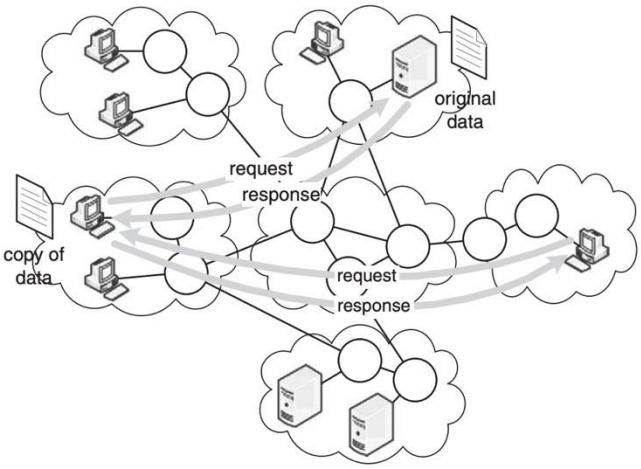


Рис.2.4 Peer-to-peer networking

Мережа доставки контенту (CDN) представляє собою географічно розподілену систему проксі-серверів та їх центрів обробки даних.

Ідея ініціативного розподілу вмісту полягає в тому, що клієнти можуть отримувати доступ до копій вмісту, які знаходяться географічно найближче

до них. Це призводить до покращення продуктивності доступу, порівняно з ситуацією, коли клієнт звертається безпосередньо до оригінального сервера.

Для використання системи розподілення контенту необхідно мати мережу, яка підтримує механізми для перенаправлення запитів на локальні копії контенту.

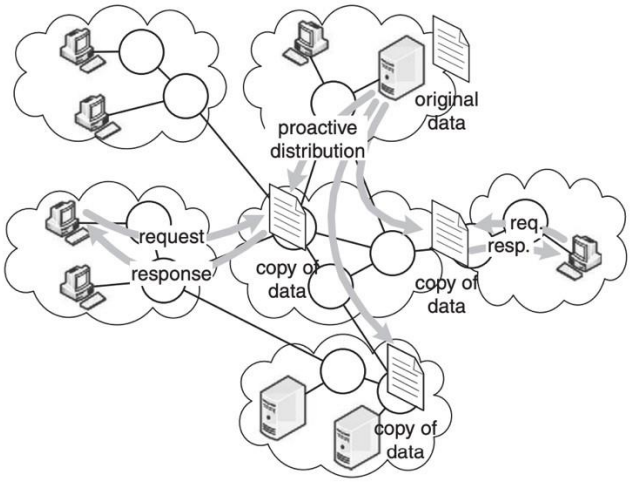


Рис.2.5 Мережі доставки контенту

Мережі, стійкі до затримки (DTN), представляють собою мережеві системи, що не можуть гарантувати постійне з'єднання. Цей підхід спрямований на вирішення технічних проблем у неоднорідних мережах, де може бути відсутній постійний мережевий зв'язок. Додатки, що використовують такі мережі, можуть охоплювати транспортні системи, де транспортні засоби періодично втрачають з'єднання. Це вимагає принципових змін у функціональності мережевих систем, щоб вузли могли зберігати дані протягом тривалого періоду.

Прикладами таких мереж є системи, що працюють у мобільних або екстремальних земних умовах, або мережі у космосі.

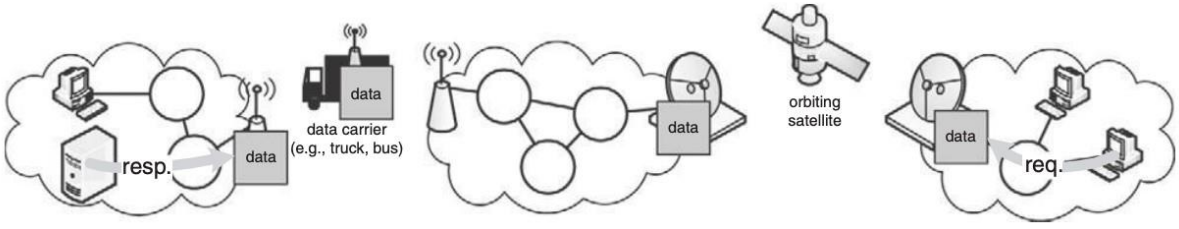


Рис.2.6 Мережа, стійка до затримок

2.2 Вибір та аналіз мови розробки застосунків

Вибір мови програмування є ключовим аспектом у процесі розробки додатків. Існує розмаїття мов програмування, кожна з яких володіє унікальними характеристиками та має свої переваги та недоліки. При виборі мови важливо враховувати сферу використання, навички розробників, фінансові можливості, цільовий функціонал додатку та інші фактори. Докладне вивчення сильних і слабких сторін кожної мови є критичним для забезпечення ефективності та масштабованості програми. Не існує універсальної відповіді на те, яка мова є найкращою, оскільки вибір залежить від конкретних вимог та умов проекту. Після оцінки переваг і недоліків можна зробити інформований вибір мови та технології, що найбільше відповідає поставленим завданням.

Мови програмування, такі як C++ і C#, вже тривалий час утримують свою популярність. Проте, за останній рік, актуальність Java зменшилася на 4.54% у порівнянні з попереднім роком. Незважаючи на це зменшення, Java продовжує лідирувати над C++ і C#. Детальний рейтинг мов програмування на травень 2021 року представлений у таблиці, яка відображає ймовірність зниження актуальності у відсотках та порівняння з травнем 2020 року.

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	^	Python	11.87%	+2.75%
3	2	v	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%
6	6		Visual Basic	4.02%	-0.16%
7	7		JavaScript	2.45%	-0.23%
8	14	^^	Assembly language	2.43%	+1.31%
9	8	v	PHP	1.86%	-0.63%
10	9	v	SQL	1.71%	-0.38%

Рис. 2.7 Статистика актуальних мов

Java систематично оновлюється двічі на рік починаючи з 2018 року. Кожне оновлення вносить значний перелік нововведень. Наприклад, у Java 14 додано такі ключові функції як відповідність шаблону для `instanceof`,

текстові блоки, корисні виняткові ситуації `NullPointer`, розподіл пам'яті з урахуванням NUMA для G1 та вилучено збирач сміття з паралельним знаком (CMS).

C# і C++ - це потужні мови програмування, які використовуються для створення широкого спектру програм. Обидві мови пройшли через кілька версій, кожна з яких додавала нові функції та можливості.

C# був випущений у 2002 році і з тих пір пройшов 12 оновлень. Деякі з найважливіших нововведень включають:

- Методи інтерфейсу за замовчуванням, які дозволяють програмістам визначати поведінку методів інтерфейсу, навіть якщо вони не реалізовані в базовому класі.
- Поліпшена відповідність шаблонам, яка спрощує використання шаблонів у програмах.
- Використовування декларацій, яке дозволяє програмістам визначати типи змінних в момент їхнього використання.
- Статичні локальні функції, які дозволяють програмістам створювати функції, доступні лише в межах блоку коду.
- Асинхронні потоки, які дозволяють програмістам виконувати завдання паралельно.
- Індекси та діапазони, які дозволяють програмістам отримувати доступ до елементів колекції за допомогою індекса або діапазону.
- Поліпшення інтерпольованих дослівних рядків, які дозволяють програмістам вставляти змінні в текстові рядки.

C++, розроблений у 1983 році, є однією з найпопулярніших мов програмування в світі. З тих пір він пройшов через кілька версій, остання з

яких, C++23, була випущена у 2023 році. Деякі з найважливіших нововведень C++23 включають:

- Тестові макроси, які дозволяють програмістам створювати макроси, які можуть приймати параметри та повертати значення.
- Призначені ініціалізатори, які дозволяють програмістам ініціалізувати об'єкти за допомогою більш короткого синтаксису.
- Вилучення вимоги використовувати typename для усунення різниці між типами у багатьох контекстах.
- Сукупна ініціалізація за допомогою дужок, яка дозволяє програмістам ініціалізувати кілька змінних за допомогою одного оператора.
- Модулі, які дозволяють програмістам розділяти код на модулі, які можна використовувати в інших програмах.
- Скорочені шаблони функцій, які дозволяють програмістам створювати шаблони функцій, які використовують менше коду.

Затребуваність програмістів C++, Java та C# є високою. Дослідження показало, що C++ є однією з найзатребуваніших мов програмування, особливо для створення систем, які вимагають високої продуктивності. Це пояснює його включення до списку найбільш високооплачуваних мов програмування.

Java і C# також є високозатребуваними мовами програмування. Вони мають практично однаковий рівень оплати праці, оскільки обидві мови є об'єктно-орієнтованими, мають схожий синтаксис і рівень складності для вивчення. Проте, Java має вищий рівень оплати, що пояснюється її великою популярністю серед розробників. Статистика свідчить, що вона використовується приблизно на 10% частіше, ніж C#.

Ось деякі особливості, які відрізняють C# і C++:

- C# є мовою, скомпільованою в машинний код, тоді як C++ є мовою, яка може бути скомпільована в машинний код або інтерпретована.
- C# має вбудовану підтримку для обробки об'єктів, тоді як C++ вимагає від програмістів самостійно реалізувати підтримку об'єктів.
- C# має більш простий синтаксис, ніж C++, що робить його більш доступним для початківців.

Ось деякі програми, для яких часто використовуються C# і C++:

- C# часто використовується для розробки веб-додатків, мобільних додатків та настільних додатків.
- C++ часто використовується для розробки систем реального часу, ігор та 3D-графіки.

2.3 Висновок по розділу «Теоретичні відомості клієнт-серверна архітектура»

У даному розділі ми детально розглянули теоретичні аспекти клієнт-серверної архітектури, яка є однією з ключових концепцій у сучасних системах розподілених обчислень. За допомогою цієї архітектури досягається розділення функціональних обов'язків між клієнтом і сервером, що сприяє покращенню ефективності, масштабованості та управління системою.

Переваги клієнт-серверної архітектури включають можливість розподіленого обчислення, простоту управління та підтримку багатокористувацькості. Крім того, ця архітектура сприяє створенню гнучких та масштабованих систем.

Проте, важливо враховувати й можливі недоліки клієнт-серверної архітектури, такі як точка відмови, проблеми безпеки та залежність від

мережі. Ці аспекти вимагають уважного планування та розробки для забезпечення найвищої надійності та безпеки системи.

Узагальнюючи, теоретичні відомості, представлені у цьому розділі, стверджують актуальність та важливість клієнт-серверної архітектури в сучасному програмному забезпеченні та дозволяють зрозуміти основні принципи її функціонування.

3. Деталі впровадження програмного забезпечення

3.1 Обґрунтування вибору засобів розробки

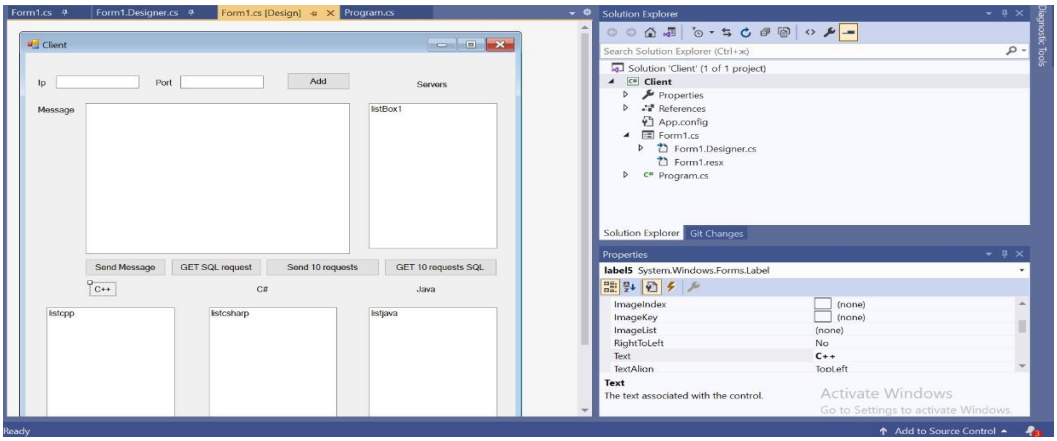
Рішення використовувати Java, C++ і C# як потенційні мови розробки випливає з появи Java як життєздатної альтернативи C++, тоді як C#, представлений Microsoft Corporation, включає принципи C++ з додатковими перевагами, подібними до Java. Порівняльний аналіз цих трьох мов вважається важливим для оцінки їх ефективності в розробці серверного компонента програми. Процес розробки спрощений спільним синтаксисом цих мов.

MS SQL Server було обрано як систему керування базами даних (СУБД) завдяки бездоганній інтеграції з операційною системою Windows і середовищем розробки Visual Studio.

3.2 Розробка клієнтської компоненти

3.2.1 Огляд інтерфейсу користувача

Інтерфейс користувача було створено за допомогою Windows Forms у .NET Framework. Цей вибір був зроблений на основі його простоти та зрозумілості. Реалізація передбачає розміщення елементів керування у формі та створення коду для обробки дій користувача, таких як клацання мишею або введення з клавіатури. Згодом автоматично генерується код інтерфейсу.



3.1 Опис користувацького інтерфейсу

Інтерфейс містить:

- Три текстові поля (System.Windows.Forms.TextBox) для введення IP-адреси, порту та тексту для обробки серверами.
- Чотири елементи для відображення списків (System.Windows.Forms.TextBox) для представлення результатів, пов’язаних з обробкою повідомлень і отриманням даних з бази даних.

П'ять кнопок:

«Відправити повідомлення» - для передачі введеного повідомлення на сервер.

«Надіслати 10 запитів» - для відправки 10 повідомлень.

«Додати» — для додавання IP-адреси та порту сервера до TextBox.

«GET SQL запит» - для відправки SQL запиту до бази даних.

```
//відправлення одного повідомлення
1 reference
private void button1_Click(object sender, EventArgs e){
    try
    {
        if (listBox1.SelectedIndex != -1)
        {
            TcpClient client = connects[listBox1.SelectedIndex]; //вибираємо підключення із існуючих, за допомогою listbox
            {
                var startTime = System.Diagnostics.Stopwatch.StartNew();  //починаємо відлік часу час

                NetworkStream stream = streams[listBox1.SelectedIndex]; // вибираємо потік із існуючих, за допомогою listbox
                byte[] data = Encoding.UTF8.GetBytes(textBox3.Text + "\r\n"); //конвертуємо строку повідомлення в байти

                stream.Write(data, 0, data.Length); //відправляємо на сервер
```

"GET 10 запитів SQL" - для ініціювання 10 ідентичних запитів до бази даних.

3.2.2 Клієнтська реалізація

Мережева взаємодія клієнта з використанням протоколів TCP і UDP здійснюється через сокети. У структурі .NET сокети втілюються в клас System.Net.Sockets, що пропонує низькорівневий інтерфейс для передачі та прийому повідомлень через мережу. Для надсилання даних використовується метод Writer(), а для процесу читання використовується метод Read(). Це дозволяє передавати дані на сервер.

```
string[] str = listBox1.Items[listBox1.SelectedIndex].ToString().Split(':'); // Розділяємо рядок на IP та порт
switch (str[1]) //порівнюємо порти і додаємо в свій список
{
    case "6000":
        {
            byte[] data2 = new byte[256]; // масив байтів для отримання повідомлення з серверу

            int bytes = stream.Read(data2, 0, data2.Length); //отримання повідомлення з серверу

            string message = Encoding.UTF8.GetString(data2, 0, bytes); //перекодовуємо в строку

            var resultTime = startTime.Elapsed;  //закінчуємо відлік часу
            string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",
                resultTime.Hours, resultTime.Minutes, resultTime.Seconds, resultTime.Milliseconds);
            listcpp.Items.Add("MSG: " + elapsedTime);

            MessageBox.Show(message);
            break;
        }
    case "7000":
        {
            byte[] data2 = new byte[256];
```

Якщо клієнт намагається надіслати запит на недоступне підключення до сервера, відображається сповіщення про те, що встановлення з’єднання неможливе. Крім того, IP-адреса та порт видаляються зі списку «Сервери».

```
    }
    catch (Exception ex)
    {
        //якщо підключення недоступне, то виводимо повідомлення та видаляємо ip та порт
        MessageBox.Show(ex.Message);
        connects.RemoveAt(listBox1.SelectedIndex);
        streams.RemoveAt(listBox1.SelectedIndex);
        listBox1.Items.RemoveAt(listBox1.SelectedIndex);

    }

}
```

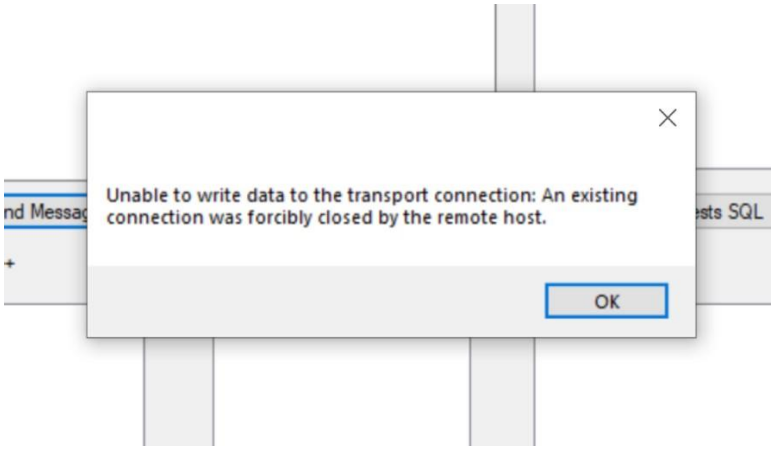


Рис. 3.4 Обробка винятків.

3.3 Реалізація сервера C++

Для створення сервера використовувався інтерфейс Winsock для оптимізації розробки мережевих додатків на платформі Windows. Winsock служить інтерфейсом між програмою та основним транспортним протоколом для передачі даних.

Щоб використовувати Winsock, важливо зв’язати та включити всі необхідні бібліотеки та файли .h.

Рис. 3.5 Бібліотеки для роботи Winsock

Наступний етап передбачає ініціалізацію. Щоб ініціювати Winsock, необхідно викликати функцію WSAStartup().

```
int main()
{
    setlocale(0, ""); // для виводу кирилиці в консоль
    char buff[1024]; // виділення пам*яті для буфера

    printf("TCP SERVER START\n");

    // перевірка чи змогли ми ініціалізувати бібілотеку WinSock2 з нашим сокетом та буфером
    // якщо ні, то повертаємо помилку

    if (WSAStartup(0x0202, (WSADATA*)&buff[0]))
    {
        // Помилка!
        printf("Error WSAStartup %d\n", WSAGetLastError());
        return -1;
    }
```

Встановлення основного каналу зв’язку в Winsock передбачає створення сокета.

```
// Ініціалізація сокету
SOCKET mysocket;

// AF_INET - сокет Інтернета
// SOCK_STREAM - потоковий соке
// 0 - за замовчування для TCP протоколу

if ((mysocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    // Помилка!
    printf("Error socket %d\n", WSAGetLastError());
    WSACleanup(); // звільнення/очищення від ресурсів зайнятих Winsock
    return -1;
}


sockaddr_in local_addr;                 // описує сокет для роботи з протоколом
local_addr.sin_family = AF_INET;       // вказуємо тип сокету
local_addr.sin_port = htons(MY_PORT); // порт
local_addr.sin_addr.s_addr = 0;      // адресу
```

```
// Отримуємо повідомлення з черги

SOCKET client_socket;     // сокет для клієнта
sockaddr_in client_addr; //  адреса клієнта - заповнюється системою

//розмір адреси клієнта
int client_addr_size = sizeof(client_addr);


// https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-accept
// Функція accept витягує перше з'єднання в черзі очікуваних з'єднань на сокетах

while ((client_socket = accept(mysocket, (sockaddr*)&client_addr, \
    & client_addr_size)))
{
    nclients++; // збільшуємо кількість клієнтів

    // витягуємо ім*я хоста

    HOSTENT* hst;
    hst = gethostbyaddr((char*)&client_addr.sin_addr.s_addr, 4, AF_INET);

    // в термінал виводимо ім*я та інформацію по клієнту
    printf("+%s [%s] new connect!\n",
        (hst) ? hst->h_name : "", inet_ntoa(client_addr.sin_addr));
    PRINTNUSERS

     // через CreateThread створюємо новий потік для роботи в ньому з клієнтом
        DWORD thID;
    CreateThread(NULL, NULL, SexToClient, &client_socket, NULL, &thID);
}

return 0;
```

Функція bind() пов’язує локальну адресу з сокетом. Це важливо для непідключених сокетів і викликається під час підготовки сокета до прослуховування за допомогою функції listen().

Ми отримуємо початкове підключення з черги за допомогою функції accept(), а потім створюється новий потік для взаємодії з клієнтом.

```
// викликаємо bind для зв*язування
if (bind(mysocket, (sockaddr*)&local_addr, sizeof(local_addr)))
{
    // Помилка
    printf("Error bind %d\n", WSAGetLastError());
    closesocket(mysocket); // закриваємо сокет!
    WSACleanup();
    return -1;
}

// підключення до сокету
// розмір черги - 0x100
if (listen(mysocket, 0x100))
{
    // Помилка
    printf("Error listen %d\n", WSAGetLastError());
    closesocket(mysocket);
    WSACleanup();
    return -1;
}

printf("ochikuvani pidkluchenya...\n");
```

```
// Отримуємо повідомлення з черги

SOCKET client_socket;     // сокет для клієнта
sockaddr_in client_addr; //  адреса клієнта - заповнюється системою
```

Після цього розробляється функція для обробки сокетів і передачі відповіді клієнту. Для передачі даних використовується функція надсилання. У сценарії одиночного SQL-запиту результат запиту відображається на консолі сервера (початковий випадок), що аналогічно роботі з повідомленням (третій випадок). Крім того, клієнту надсилається повідомлення «Сервер ОК».

```
// ця функція буде викликатись лише в новому потоці для
DWORD WINAPI SexToClient(LPVOID client_socket){

    SOCKET my_sock;
    my_sock = ((SOCKET*)client_socket)[0];
    char buff[20 * 1024];


        // приймання рядка від клієнта та повернення її клієнтові
    int bytes_recv;
    // Функція recv отримує дані з підключеного сокета
    while ((bytes_recv = recv(my_sock, &buff[0], sizeof(buff), 0)) &&
        bytes_recv != SOCKET_ERROR){

        buff[bytes_recv] = '\0';

        std::string test(buff, bytes_recv);
         if (test._Equal("sql\r\n"))
        {
            SAConnection con;
            con.Connect("User", "", "", SA_SQLServer_Client);
            SACommand cmd(
                &con,
                _TSA("SELECT * FROM person"));
            cmd.Execute();
            while (cmd.FetchNext())
            {
                SAString sName = cmd.Field(_TSA("name"));
                long nId = cmd.Field(_TSA("id"));
                printf("Name: %s, Id: %d \n", sName.GetMultiByteChars(), nId);

            }
            std::string send_arr = "Server OK";
            send(my_sock, send_arr.c_str(), sizeof(send_arr), 0);
        }
        else if (test._Equal("sql10\r\n"))
        {
            SAConnection con;
            con.Connect("User", "", "", SA_SQLServer_Client);
            SACommand cmd(
                &con,
                _TSA("SELECT * FROM person"));
            cmd.Execute();
            while (cmd.FetchNext())
            {
                SAString sName = cmd.Field(_TSA("name"));
                long nId = cmd.Field(_TSA("id"));

            }
            std::cout << "SQL 10 REQUESTS" << std::endl;
            std::string send_arr = "Server OK";
            send(my_sock, send_arr.c_str(), sizeof(send_arr), 0);
        }
        else
        {
            std::cout << test << std::endl;
            std::string send_arr = "Server OK";
            send(my_sock, send_arr.c_str(), sizeof(send_arr), 0);
        }

    }
```

Останнім кроком є закриття з’єднання. Активне закриття з’єднання виконується за допомогою функції closesocket().

```
}
nclients--; // зменшуємо кількість активних клієнтів
printf("-disconnect\n"); PRINTNUSERS

    // закриваємо сокет
    closesocket(my_sock);
return 0;
```

Встановлення з’єднання з базою даних було здійснено шляхом взаємодії з класом SQLAPI++ для СУБД.

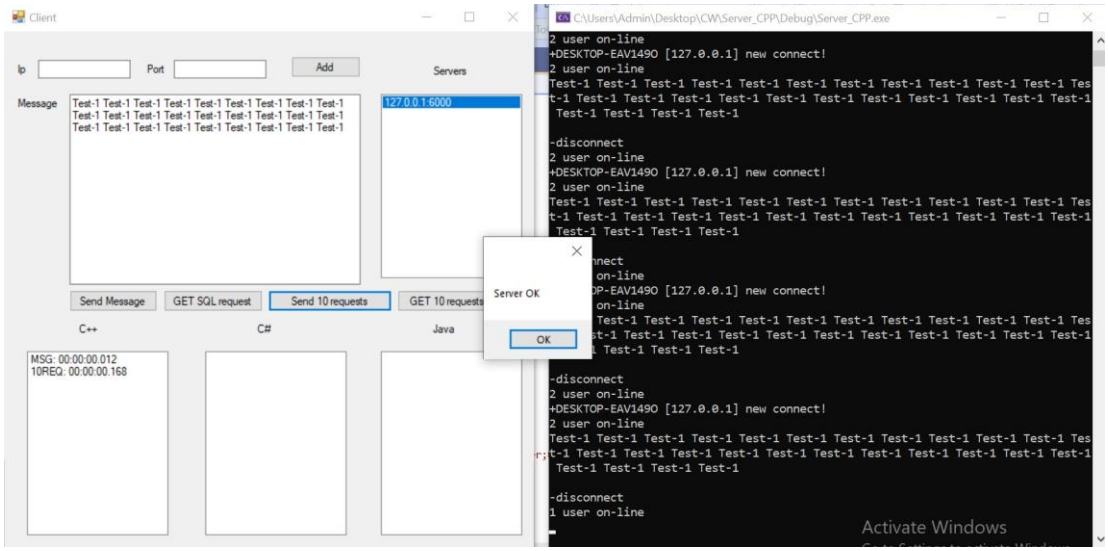


Рис. 3.6 Демонстрація роботи сервера

3.4 Сервер C#

Спочатку програма ініціює створення TCP-клієнта разом з ініціалізацією клієнтського об’єкта.

```
3 references
public class ClientObject
{
    public TcpClient client;
    1 reference
    public ClientObject(TcpClient tcpClient)
    {
        client = tcpClient;
    }
```

Далі ми встановлюємо необхідний потік та ініціалізуємо його за допомогою отриманого потоку від нашого клієнта. Згодом програма захоплює повідомлення з клієнтського потоку, створюючи об’єкт StringBuilder і створюючи рядок.

Після цього отриманий рядок проходить перевірку. Якщо він відповідає "sql\r\n" або "sql10\r\n", формулюється рядок підключення.

Цей рядок містить усі необхідні дані для підключення до бази даних, включаючи ім’я сервера, ім’я бази даних і рядок Trusted_Connection, що представляє автентифікацію Windows.

На наступному кроці, поряд із створенням рядка запиту на мові SQL для бази даних, сам процес підключення відбувається з використанням попередньо створеного ConnectionString.

```
1 reference
public void Process()
{
    NetworkStream stream = null;
    try
    {
        stream = client.GetStream();
        byte[] data = new byte[64];
        while (true)
        {

            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            do
            {
                bytes = stream.Read(data, 0, data.Length);
                builder.Append(Encoding.UTF8.GetString(data, 0, bytes));
            }
            while (stream.DataAvailable);

            string message = builder.ToString();

            if(message.Equals("sql\r\n")|| message.Equals("sql10\r\n"))
            {
                string connectionString = @"Server=localhost;Database=User;Trusted_Connection=True;";
                string sql = "select * from [dbo].[person]";

                SqlConnection connection = new SqlConnection(connectionString);
```

Далі встановлюється блок використання, який після завершення коду в ньому звільняє простір пам’яті, зайнятий даними, створеними в ньому.

У цьому блоці ініціюється з’єднання з базою даних, з неї витягуються дані та інформація виводиться на консоль користувача.

Згодом наша змінна даних записується в попередньо згенерований потік.

На третьому кроці програма займається моніторингом клієнта через нові потоки. Це передбачає встановлення об’єкта TcpListener на порту «127.0.0.1» і виклик його методу Start(), щоб розпочати прослуховування пакетів на цьому порту.

Після цього створюється новий потік для обслуговування нового клієнта. Зрештою, у блоці finally, який виконується навіть у разі виняткової ситуації, яка виникає через роботу програми, процес прослуховування зупиняється.

```
0 references
class Program
{

    const int port = 7000;
    static TcpListener listener;
    0 references
    static void Main(string[] args)
    {
        try
        {
            listener = new TcpListener(IPAddress.Parse("127.0.0.1"), port);
            listener.Start();
            Console.WriteLine("Ochikuvania pidkluchenya...");

            while (true)
            {
                TcpClient client = listener.AcceptTcpClient();
                ClientObject clientObject = new ClientObject(client);

                // створюємо новий потік для роботи з клієнтом
                Thread clientThread = new Thread(new ThreadStart(clientObject.Process));
                clientThread.Start();
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        finally
        {
            if (listener != null)
                listener.Stop();
        }


    }
}
```

3.5 Реалізація сервера Java

У Java обробка сокетів була реалізована за допомогою таких бібліотек:

java.net.ServerSocket: ця бібліотека полегшує реалізацію серверного сокета, який очікує запитів, надісланих до нього. Він також здатний надсилати різні дані у відповідь.

java.net.Socket: використовується як сокет для зв’язку між сервером і клієнтами.

Крім того, для операцій з базою даних використовувався microsoft.sqlserver.jdbc.SQLServerDataSource. Ця бібліотека дозволяє конфігурувати зв'язок із сховищем даних SQL Server.

```
class Main {

    static ExecutorService executeIt = Executors.newFixedThreadPool( nThreads: 10);

    public static void main(String[] args) throws IOException {

        // стартуємо сервер на порту 8000 та ініціалізуємо змінну для обробки консольних команд з самого сервера
        try (ServerSocket server = new ServerSocket( port: 8000);
             BufferedReader br = new BufferedReader(new InputStreamReader(System.in))) {
            System.out.println("Server socket created, command console reader for listen to server commands");

            // заходимо в цикл, за умови що серверний сокет не закритий
            while (!server.isClosed()) {

                Socket client = server.accept();


                executeIt.execute(new MonoThreadClientHandler(client));
                System.out.print("Connection accepted.");
            }

            executeIt.shutdown();
        } catch (IOException e) {
            e.printStackTrace();
```

Рис. 3.7 Бібліотека для реалізації сокетів java.net.ServerSocket

```
class MonoThreadClientHandler implements Runnable {

    public Socket clientDialog;
    private static String URL = "jdbc:sqlserver://localhost\\SQLEXPRESS:1434;DatabaseName=User;authenticationScheme=JavaKerberos;use
    public MonoThreadClientHandler(Socket client) {
        clientDialog = client;
    }

    @Override
    public void run() {

        try {
            // ініціалізуємо канали для спілкування в сокеті для сервера

            BufferedWriter out = new BufferedWriter(new OutputStreamWriter(clientDialog.getOutputStream()));
            // читаємо з сокета
            DataInputStream in = new DataInputStream(clientDialog.getInputStream());
            BufferedReader d = new BufferedReader(new InputStreamReader(in));
            System.out.println("DataInputStream created");
            System.out.println("DataOutputStream  created");
```

Рис. 3.8 Бібліотека для реалізації сокетів java.net.Socket

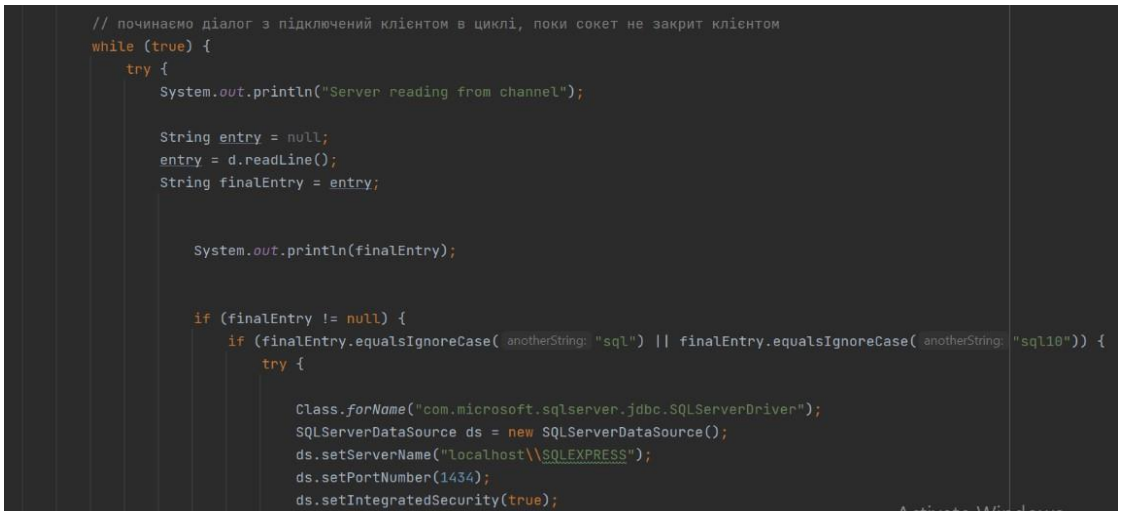


Рис. 3.9 Бібліотека для реалізації сокетів microsoft.sqlserver.jdbc.SQLServerDataSource

3.6 Виклики та проблеми, що виникають у процесі розробки

Найбільше часу зайняла розробка на C++, насамперед через численні проблеми із запитами до бази даних. Це потребувало окремого завантаження та налаштування бібліотеки SQLAPI++, що додало значну кількість часу до процесу. Реалізація сервера на C++ виявилася найскладнішою, вимагаючи приблизно на 30% більше коду, ніж C#, і на 15% більше, ніж Java, щоб досягти еквівалентної функціональності.

Так само під час створення сервера на Java виникла проблема щодо підключення до бази даних. Ця складність виникла через використання бібліотеки, де автентифікація відбувається через обліковий запис SQL Server.

3.7 Результати тестування та виконання програми

Для проведення тестування необхідно включити сервери в список «Сервери»:

127.0.0.1:6000 для C++

127.0.0.1:7000 для C#

127.0.0.1:8000 для Java

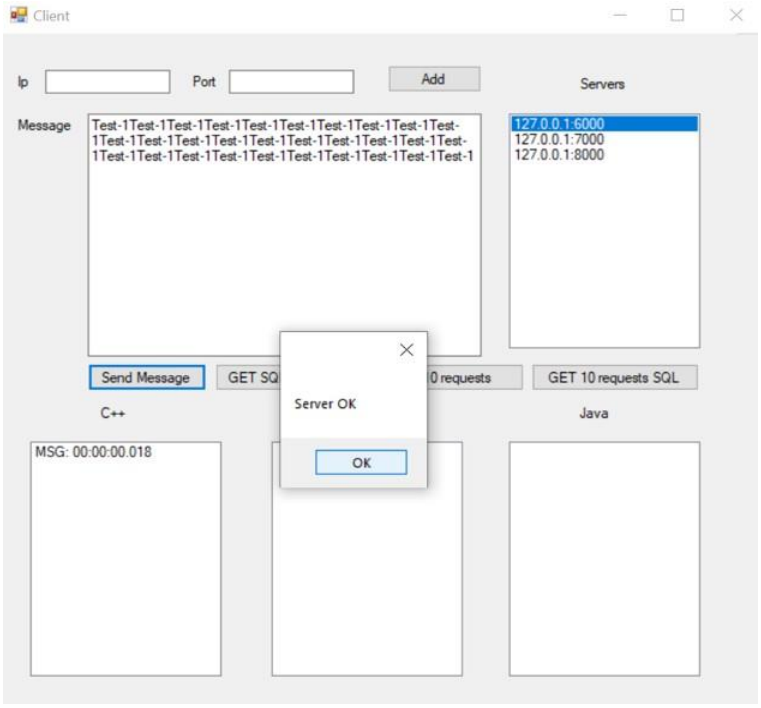


Рис. 3.10 Передача першого повідомлення в 840 символів

Після успішної обробки запиту сервер відповідає повідомленням «Сервер ОК». На стороні клієнта таймер припиняє роботу, а час, що минув, реєструється у відповідній таблиц

Початковий тест передбачає відправку текстового повідомлення, що містить 840 символів, повторене 9 разів. Результат такий:

Якщо результат 0, виконання відбулося менш ніж за 1 мілісекунду.

Для всіх трьох мов результати демонструють послідовність без значних коливань у часі. Java демонструє найбільш оптимальну та стабільну продуктивність, тоді як C++ займає найнижчу позицію.

У другому тесті сценарій передбачає надсилання текстового повідомлення з 350 символів, яке повторюється 10 разів із загалом 9 повторень. Результати такі:

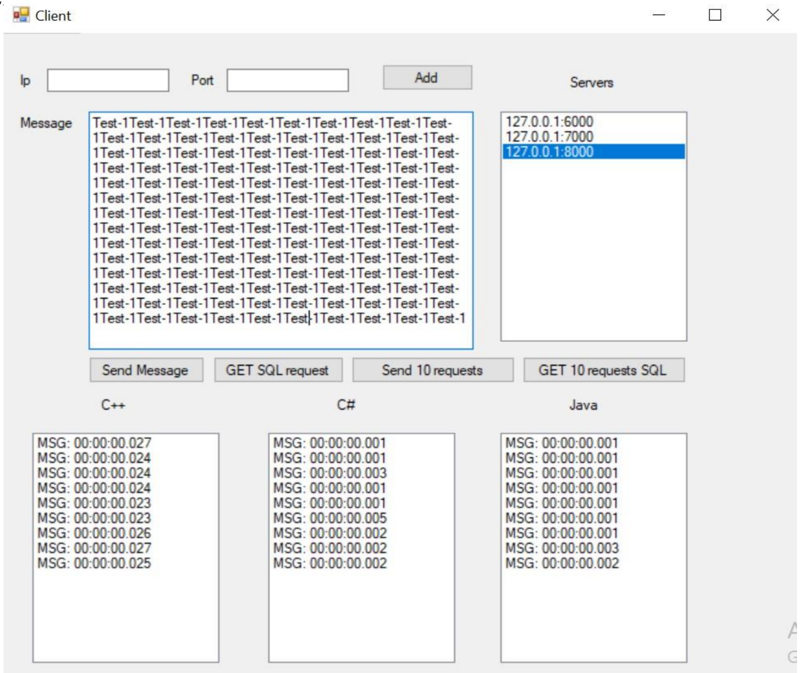


Рис. 3.10 Передача другого повідомлення в 350 символів

У цьому експерименті Java виявилася найстабільнішою та найефективнішою, демонструючи найшвидше виконання. C# продемонстрував порівняльні результати, але його продуктивність була менш стабільною та стикався з кількома збоями. C++ відставав у швидкості, закріпивши останню позицію, але його результати залишалися стабільними без помітного зниження швидкості.

У третьому тесті, який включає SQL-запит на MS SQL Server, із кількістю повторень 5 і 9 завершених рядків, результати відрізняються від попередніх двох тестів. Цікаво, що всі три мови продемонстрували майже однакові результати швидкості. Серед них C# продемонстрував найменші флуктуації, що свідчить про найвищий рівень стабільності.

3.8 Висновок по розділу впровадження програмного забезпечення

Під час розробки курсової роботи я провів аналіз актуальності C++, Java та C#, дослідивши їхню популярність у сучасному середовищі, ефективність розробки клієнт-серверних додатків та виклики, з якими стикаються програмісти. Крім того, я порівняв швидкість і стабільність їх роботи.

Дослідження підкреслило, що C++ має унікальний набір характеристик, які відрізняють його, і ні Java, ні C# не можуть служити прямою заміною. Процес ретельного освоєння C++ і написання коду цією мовою вимагає більше часу, а погодинна ставка для програміста на C++, як правило, вища в порівнянні з колегами, які використовують Java і C#.

У практичному аспекті курсової роботи було розроблено клієнт-серверний додаток, що стало основою для висновків у дослідженні. Сервери додатків були реалізовані на C++, C# та Java, тоді як клієнт був розроблений з використанням .NET.

Під час виконання курсового проекту я зіткнувся з проблемами, зокрема численними помилками та проблемами зі зв’язуванням бібліотек під час роботи над кодом C++. Процес кодування в C++ займав на 25-30% більше часу, ніж в інших мовах. Іншим помітним недоліком C++ у порівнянні з Java і C# є необхідність створювати новий виконуваний файл для кожної архітектури процесора (x86, x64, ARM) і операційної системи (Windows, Mac, Linux).

З точки зору продуктивності, C++ продемонстрував нижчі результати, тоді як Java і C# продемонстрували схожі рівні продуктивності.

Висновок

У ході даної дипломної роботи були проведені масштабні дослідження та розробка інформаційної системи для підприємства на основі веб-технологій. Реалізація проекту включала наступні основні етапи та досягнення:

Аналіз вимог та розробка інформаційної системи:

Проведено поглиблений аналіз потреб бізнесу та визначено основні вимоги до інформаційної системи.

Дизайн інтерфейсу користувача:

Для полегшення взаємодії з системою створено інтуїтивно зрозумілий і зручний інтерфейс.

Розробка серверної та клієнтської частин:

Обидві частини системи успішно розроблені, що забезпечує їх високу ефективність і взаємодію на оптимальному рівні.

Розробка бази даних:

Створено оптимізовану та надійну базу даних для забезпечення ефективного зберігання та управління інформацією.

Тестування та перевірка:

Було проведено широке тестування системи, виявлені та усунені можливі проблеми, що гарантує надійність та стабільність роботи.

Підводячи підсумок, хочу підкреслити, що кожен етап розробки і кожне прийняте рішення було ретельно обґрунтовано і виконано мною особисто. Впроваджена інформаційна система стала не тільки результатом масштабних досліджень, а й стала вагомим кроком до підвищення ефективності та конкурентоспроможності нашого підприємства в сучасному бізнес-середовищі.

Список літератури

1. Василенко К. В., Клепіков С. В., Омельченко В. В. Розробка web-систем. Навчальний посібник. - К.: Видавничий дім "Освіта", 2022. - 336 с.
2. Гущин В. В., Скрипко О. В., Тарасов А. В. Розробка web-систем: навчальний посібник. - К.: Київський національний університет імені Тараса Шевченка, 2021. - 224 с.
3. Кириченко О. В., Шевчук В. В. Розробка web-систем: навчальний посібник. - К.: Національний технічний університет України "Київський політехнічний інститут", 2020. - 304 с.
4. Зайцев А. А., Зайцев В. А., Кухаренко Л. В. Розробка інформаційних систем і технологій: навчальний посібник. - К.: Видавничий дім "АСТ", 2022. - 368 с.
5. Кравчук В. В., Кравчук О. А. Розробка інформаційних систем: навчальний посібник. - К.: Видавничий дім "Інститут вищої освіти", 2022. - 424 с.
6. Морозов В. П., Морозова О. В. Розробка інформаційних систем і технологій: навчальний посібник. - К.: Видавничий дім "Навчальна книга", 2021. - 352 с.
7. Романюк В. В., Романюк І. В. Розробка інформаційних систем: навчальний посібник. - К.: Видавничий дім "Світ", 2021. - 320 с.
8. Смирнов О. А., Смирнов В. А. Розробка інформаційних систем: навчальний посібник. - К.: Видавничий дім "Академія", 2021. - 304 с.
9. Василенко К. В., Клепіков С. В., Омельченко В. В. Аналіз сучасних підходів до розробки web-систем. // Вісник Національного університету водного господарства та природокористування. Серія: Інформатика, 2022. Вип. 3(75). С. 11-18.
10. Гущин В. В., Скрипко О. В., Тарасов А. В. Архітектура web-систем: сучасні тенденції. // Інформаційні технології і засоби навчання, 2021. Вип. 73. С. 15-23.

11. Кириченко О. В., Шевчук В. В. Використання сучасних технологій в розробці web-систем. // Вісник НТУУ "КПІ". Серія: Інформаційні технології та управління. 2020. Вип. 3(123). С. 5-11.
12. Андрієв О. В., Андрієв В. В. Сучасні підходи до розробки web-систем для підприємств. // Вісник Національного університету "Львівська політехніка". Серія: Прикладна інформатика, 2022. Вип. 971. С. 12-19.
13. Бойко В. В., Бойко О. В. Архітектура web-систем для підприємств: сучасні тенденції. // Вісник Національного університету "Львівська політехніка". Серія: Прикладна інформатика, 2021. Вип. 961. С. 17-24.
14. **Гаврилюк О. А., Гаврилюк І. В. Використання сучасних технологій в розробці web-систем
15. "Веб-програмування на мові Python: Django" - Сергій Мавродій
16. "Розробка веб-додатків з використанням Java та Spring" - Ігор Погожев
17. "Сучасний Front-end на прикладі Angular та TypeScript" - Сергій Карелин
18. "JavaScript. Підручник з сучасних технік програмування" - Етан Браун
19. "Node.js в дії" - Майк Кантелон
20. "React.js. Розробка великих веб-застосунків" - Алекс Бенкс, Ева Порселло
21. "Vue.js в дії" - Ерік Чун

Додаток

1. Development of the enterprise information system based on web-technologies

1.1 Requirements Analysis, Information System Development

Requirements analysis is a critical stage in the development of an enterprise information system based on web-technologies. This stage helps to understand what functions and characteristics of the system need to be implemented, as well as to determine how the system should meet user needs and business requirements. Here are some key elements of requirements analysis:

1. Gathering Requirements:

- Conduct interviews with representatives of different departments and users to find out their needs and requirements for the system.

- Identify the main stakeholders and their roles in the system.

2. Documenting Requirements:

- Creating a document that contains all the requirements for the system. This can be a requirements specification or other document that serves as the primary source of information for development.

3. Formalization of requirements:

- Prioritizing requirements and their importance to the business.

- Defining and documenting the conditions for canceling or changing requirements during development.

4. Functional Requirements Analysis:

- Determining how the system should perform specific tasks and operations.

- Developing diagrams that illustrate the functionality of the system, such as data flow diagrams or use case diagrams.

5. Non-Functional Requirements Analysis:

- Determination of non-functional characteristics of the system, such as performance, security, reliability, and others.

- Identification of constraints that must be taken into account during development.

6.Requirements Validation:

- Validate requirements to meet user needs and business goals.

- Determining how requirements will be validated during development and testing.

7. Stakeholder Involvement:

- Liaising with department representatives and users throughout the development process to clarify and improve requirements.

8. Documenting Requirements Changes:

- Creating procedures for adding, modifying, and removing requirements during the development process, and how this affects the project schedule and budget.

Requirements analysis is an ongoing process that can involve iterations and changes to requirements during system development. A detailed and responsible analysis of requirements is a key step in creating a successful information system that meets the needs of the enterprise and users.

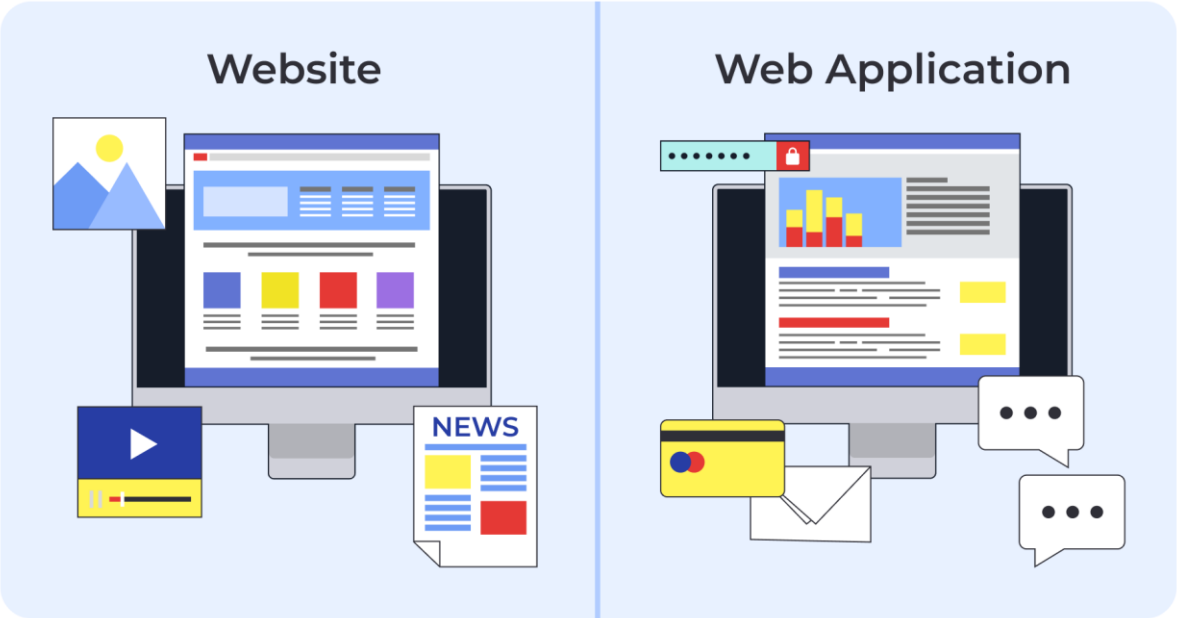


Fig. 1.1 Differences between website and web application

1.2 System Design

System design is a critical stage in the development of an enterprise information system based on web-technologies. This process involves creating a detailed system plan and identifying the architecture, technology, and infrastructure required to implement the intended features and requirements. On this page, we will look at the key steps in designing an information system.

1. System Architecture:

- Determination of the general structure of the system.
- Development of a data model and interaction between components.

2. Choice of Technology:

- Choice of programming languages, frameworks, and other development tools.
- Assessment of the compatibility of selected technologies with business requirements.

3. Infrastructure:

- Consideration of opportunities for hosting the system (on its own servers, in cloud services, etc.).
- Determination of requirements for equipment and network infrastructure.

4. System Security:

- Development of a strategy for data protection and access to the system.
- Implementation of authentication and authorization mechanisms.

5. Scaling and Performance:

- Planning for scalability of the system by increasing the amount of data and users.
- Optimization of system performance and response to requests.

6. Data Backup & Restore:

- Developing a strategy for regular data backup.
- System recovery plan in case of unforeseen events.

7. Scalability:

- Consideration of opportunities to expand the functionality of the system in the future.

- Providing flexibility to add new features.

8. Project Documentation:

- Creation of technical documentation that describes the architecture, configuration and other key aspects of the system.

9. Tasks for developers:

- Distribution of functional tasks among developers.

- Planning working iterations for development.

10. Timeline and Budget:

- Defining a time frame for the project and allocating the budget between different phases.

The design of the system determines the basic framework and development strategy. A well-designed system must be ready for the effective implementation of subsequent stages, such as development and testing. Detailed design helps ensure the successful implementation of your information system.

1.3 User Interface Design

Rice. 1.2 Interface Design Process

The user interface (UI) is an important component of any web-based information system, as it determines the way the user interacts with the system. On this page, we'll explore the key aspects of UI design.

1. Defining the Target Audience:

- Analysis and determination of user characteristics, their needs and expectations from the system.

- Creating user personas to better understand the audience.

2. Information Architecture Development:

- Structuring and organizing information on a website or app.

- Creation of a navigation tree and sitemaps.

3. Interface Prototyping:

- Creating prototypes of web pages or application screens that reflect the main functionality.

- Testing prototypes on users to collect feedback.

4. Interface Design:

- Development of aesthetic and functional design of web pages or application screens.

- Choice of color palette, fonts, icons, and graphics.

5. Responsive Design:

- Ensuring that the interface is displayed correctly on different devices and screen sizes (responsive design).

- Testing on mobile devices, tablets and computers.

6. User Experience:

- Development of interaction logic and animations that make it easier for the user to use the system.

- Adding interactive elements such as buttons, forms, and notification boxes.

7. Interface Testing and Validation:

- Conducting interface tests to check for compliance with design and functionality.

- Collecting feedback from users and making changes based on test results.

8. Accessibility:

- Ensuring that the interface is accessible to all users, including people with disabilities.

9. Interface Documentation:

- Creation of documentation that describes the structure and functionality of the interface for developers and testers.

10. Iterative Process:

- Interface design is an iterative process that can involve several rounds of prototyping and refinement.

The design of the user interface is a key stage, as the perception and usability of the system by users depends on it. A well-designed interface can improve the user experience and contribute to the success of a project.

1.4 Backend development

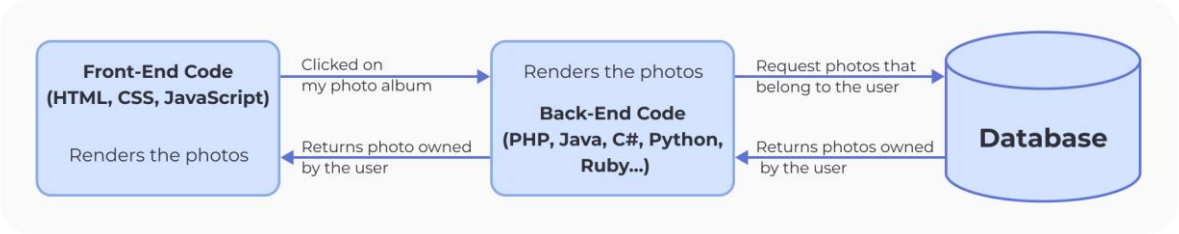


Fig. 1.3 Example of Front-End and Back-End interaction

The development of the server part, or backend, is one of the most important components of the process of creating an enterprise information system based on web-technologies. The backend is responsible for processing business logic, storing data, and interacting with the database. On this page, we'll look at the key stages of backend development.

1. Choice of Technology and Stack:

- Definition of programming language and frameworks for backend development.

- Selection of a database management system (e.g., MySQL, PostgreSQL, MongoDB).

2. Backend Architecture:

- Development of the system architecture, including the definition of server components and their interaction.

- Distribution of tasks and roles between server components.

3. Developing Business Logic:

- Writing code that executes the system's business logic, including processing requests from users and interacting with the database.

- Implementation of algorithms for calculations and operations on data.

4. Backend Security:

- Protection against potential threats such as SQL injection attacks, data interception, etc.

- Using encryption and authentication methods to ensure data security.

5. API Development:

- Creation of API (Application Programming Interface) for interaction with the client part (frontend) and other systems.

- Documenting the API for the convenience of developers.

6. Session and Session Management:

- Implementation of mechanisms for managing user sessions and storing session data.

7. Testing and Debugging:

- Performing unit, integration, and functional tests to verify backend functionality.

- Error detection and correction.

8. Scaling:

- Consider scaling the backend to support data and user growth.

9. Documentation and Support:

- Creation of technical documentation for the backend part of the system.

- Providing support and improvement of the system after implementation.

10. Data Protection:

- Protection of sensitive data in the database and during transmission over the network.

Backend development requires attention to detail and is done with many aspects in mind, including security, performance, and scalability. A well-designed and implemented backend is an important prerequisite for the success of your information system.

1.5 Development of the client part (frontend)

Fig. 1.4 Programming languages for Backend development and Front-End development

Front-end development includes everything that users encounter directly: text colors and styles, images, graphs and tables, buttons, colors, and navigation menus.

At the moment, you can implement all the logic in the client part of the application and not use Back-End. This approach has its pros and cons. But you may encounter applications where there will be no backend, and the data will be processed and stored in the Front-End part of the application.

The structure, design, behavior, and content of everything displayed on the browser screen when opening websites, web applications, or mobile applications is implemented **by Front-End developers.**

The development of the client part, or frontend, is one of the key components of creating an enterprise information system based on web-technologies. Frontend is responsible for the interface that users of the system see and use. On this page, we will look at the main stages of developing the client part.

1. Choice of Technology and Stack:

- Identification of the main technologies for frontend development (HTML, CSS, JavaScript) and frameworks (e.g., React, Angular, Vue.js).

- Choice of design and development tools (e.g., Adobe XD, Figma)

2. Prototyping and Design:

- Creation of layouts and interface design.

- Development of prototypes for concept validation and feedback collection.

3. User Interface (UI) Development:

- Creating an HTML structure and CSS styles to display the interface.

- Designing components and templates for greater flexibility and reuse.

4. Programming (JavaScript):

- Writing JavaScript code to implement functionality, user experience, and data processing.

- Use of libraries and frameworks to speed up development.

5. Performance Optimization:

- Optimization of page loading and interaction with the server to quickly respond to user requests.

- Caching and reducing the number of requests to the server.

6. Responsive Design:

- Ensuring correct display on different types of devices and screen sizes (responsive design).

- Testing on mobile devices, tablets and computers.

7. Validation and Testing:

- Checking the correctness of the functionality and compliance of the design with the requirements and specifications.

- Performing unit and integration tests.

8. User Documentation and Training:

- Creation of documentation for users on how to use the interface.

- Preparation of training materials and instructions.

9.Status & Data Management:

- Storing and managing the state of the application and interacting with the server via API.

10. Security:

- Client-side data protection, including processing, validation, and sanitization of the data entered.

11. Server Integration:

- Interaction with the server to receive and send data via API.

12. International Localization and Localized Content:

- Support for language versions and localized content options for different regions.

Client-side development is a process that requires attention to detail and aims to create a user-friendly and efficient interface. A well-implemented frontend part will allow users to easily interact with the system and use its functionality.

1.6 Database development

A database is one of the most important components of any enterprise information system based on web-technologies. It is responsible for storing, organizing, and accessing the data used in the system. On this page, we will look at the key aspects of database development.

1. Selection of a database management system (DBMS):

- Determining the DBMS that best suits your project needs (e.g., MySQL, PostgreSQL, Microsoft SQL Server, MongoDB).

- Choice between relational and non-relational databases depending on the type of data and requirements.

2. Database Schema Design:

- Create a schema that defines the structure and relationships between database tables.

- Define keys, indexes, and constraints to ensure data integrity.

3. SQL Query Development:

- Writing SQL queries to create tables, insert, update, and delete data.

- Define queries to retrieve and analyze data from the database.

4. Database Optimization:

- Use indexes to improve query performance.

- Monitor and debug the database to ensure performance and scalability.

5. Data Security:

- Database protection from unauthorized access and security attacks (e.g., SQL injection).

- Use of roles and permissions to control access to data.

6. Backup & Restore:

- Developing a strategy for regular database backups.

- Defining procedures for data recovery in case of accidents.

7. Database Migrations:

- Use of migrations to manage changes in the database structure during the development of the project.

8. Integration with Backend and Frontend:

- Interaction with the server and client parts of the system via API to ensure data exchange.

9. Complex Queries and Operations:

- Development of complex SQL queries to perform operations such as reporting, data aggregation, etc.

10. Monitoring and Support:

- Installation of monitoring systems to identify and fix database problems.

- Support and optimization of the database after the implementation of the system.

Database development requires a deep understanding of the system's needs and confidence in its reliability and security. A well-designed and optimized database plays a key role in ensuring the efficiency and reliability of the information system.

1.7 Testing and validation

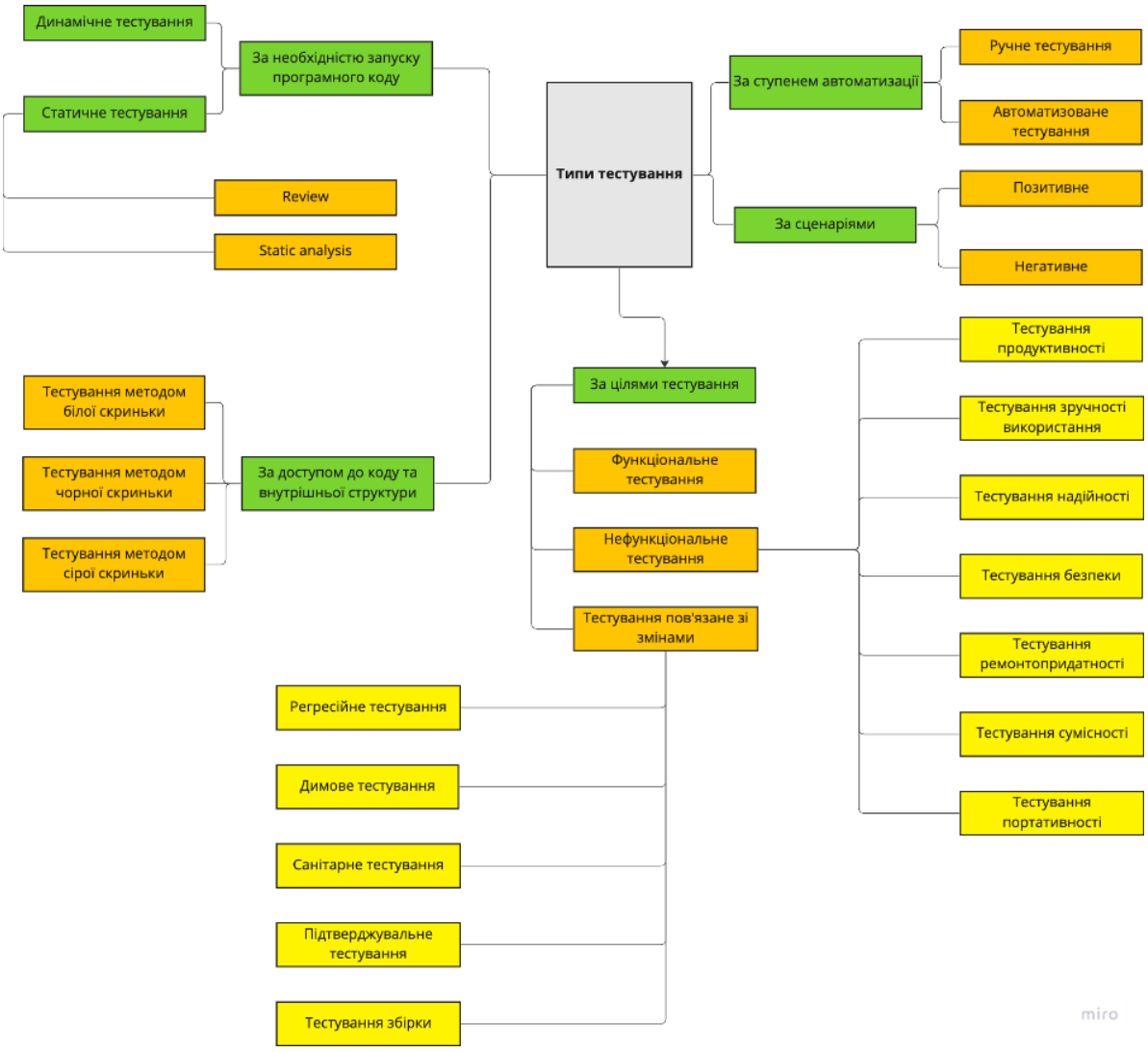


Fig. 1.5 Types of testing

Testing and validation are important stages in the development of an information system based on web technologies. These processes help to ensure the quality and reliability of the system before it is implemented. On this page, we'll look at the key aspects of testing and validation.

1. Test Planning:

- Creating a test plan, including testing scope, resources, and timeline.
- Definition of testing goals and acceptance criteria.

2. Types of Testing:

- Unit testing: verification of individual components and functions of the system.

- Integration testing: checking the interaction between components.

- Functional testing: checking the compliance of functionality with the requirements.

- Performance testing: measuring the speed and scalability of the system.

- Security testing: Identify and remediate potential data security threats.

- Mobile Testing: Testing on different platforms and screen sizes.

3. Test Automation:

- Using automatic test scripts to speed up the testing process.

- Automatic execution of regression tests to detect errors after changes in the code.

4. Test Data Generation:

- Creating test datasets for different test scenarios.

- Use of real and synthetic data for tests.

5. Test Scenarios and Test Scenarios:

- Development of details of test scenarios, including the sequence of user actions and expected results.

- Testing different options for interacting with the system.

6. Validation as required:

- Verification that the system meets all the requirements defined at the initial stage of the project.

- Identification and correction of deviations from requirements.

7. Security Testing:

- Conducting tests to identify vulnerabilities and security attacks.

- Fixing identified security issues.

8. **Reporting and Recording Errors**:

- Documenting test results and generating bug reports.

- Taking action to correct errors and verifying their correction.

9. Version Tracking and Control:

- Using version control systems to track changes in code and tests.
- Ensuring that code versions are synchronized between developers.

10. Test Coverage Analysis:

- Determining the level of coverage by code tests and

1.8 Implementation

The implementation of the information system is the final stage in the process of its development and readiness for use by the enterprise. Proper implementation guarantees the successful launch of the system and its acceptance by users. On this page, we'll look at the key aspects of implementation.

1. Implementation Planning:

- Creating a detailed implementation plan, including dates, steps, and responsible persons.
- Determination of the necessary resources and budget for implementation.

2. Pre-Implementation Testing:

- Conducting final testing of the system before implementation to ensure its readiness.
- Correction of all detected errors and problems.

3. User Preparation:

- Providing training and support to users of the system.
- Creation of training materials and instructions.

4. Implementation in stages:

- Implementation in stages to reduce risk and burden on the enterprise.
- Running the system in a limited audience or in a limited mode.

5. Monitoring and Support:

- Providing technical support and monitoring of the system after implementation.
- Real-time detection and correction of possible problems.

6. User Incentives:

- Involving users in the active use of the system and collecting their feedback.

- Implementation of incentives and motivational measures for users.

7. Analysis and evaluation of implementation:

- Assessing the effectiveness and compliance of the system with business requirements after implementation.

- Analysis of results and decision-making on next steps.

8. Regular Updates and Support:

- Establishing procedures for regular updates and improvements to the system.

- Support and implementation of changes tailored to business needs.

9. Project Completion:

- Formal completion of the project, implementation and transfer of the system to full use by the enterprise.

- Assessing and documenting achievements and lessons learned after the completion of the project.

The implementation of the information system is an important point in the life cycle of the project. Properly prepared and carried out implementation contributes to the successful functioning of the system and meeting the needs of users and the enterprise.

1.9 Software Development Life Cycle Models

The software development life cycle is called Software Development Lifecycle Models or the abbreviation SDLC is used.

The Software Development Life Cycle Model (SDLC)** is a set of different types of activities that are performed at each stage of the software development process, these activities are related to each other logically and chronologically.

Each SDLC model requires different testing approaches.

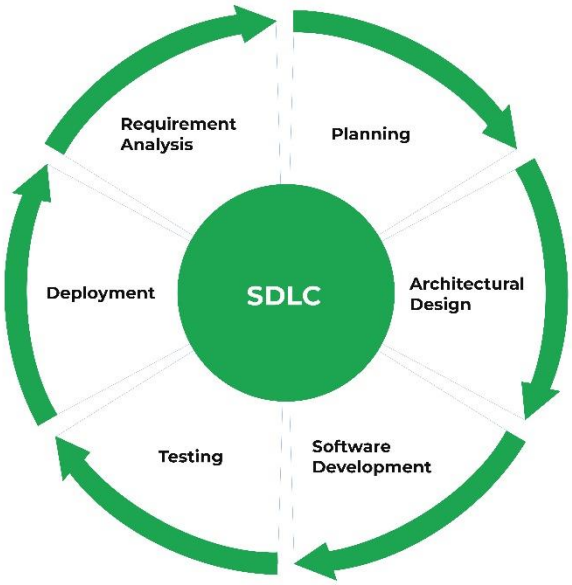


Fig. 1.6 Life Cycle Model

Software Development & Testing

The lifecycle model adopted for the project will have a big impact on testing.

However, there are a few characteristics of good testing in any SDLC model:

- For each development activity, there is a corresponding testing activity.
- Each level of testing has goals specific to that level.
- The analysis and design of tests for a given level of testing begins during the corresponding development activity.
- Testers participate in discussions to define and refine requirements and design, and participate in the validation of work products (e.g., requirements, design, user-story, etc.).

The most common SDLC models are **the sequential** development model and the **iterative and incremental development model.**

Fig. 1.7 Sequential development model

A sequential development model is a model in which the software development process is a linear, sequential flow of activities.

This means that any phase in the development process must begin after the completion of the previous one, one after the other or one by one.

The most well-known models of sequential development are:

- Waterfall;
- V-model;

The main features of the Waterfall model:

- Development activities (e.g., requirements analysis, design, coding) are performed one after the other (each subsequent stage begins only after the previous one is completed).

Testing is carried out after the coding (development) stage is completed. Waterfall is translated from English as "waterfall". Therefore, the model of the software development life cycle is like a waterfall of processes and activities. The diagram below shows a sequential SDLS Waterfall model:

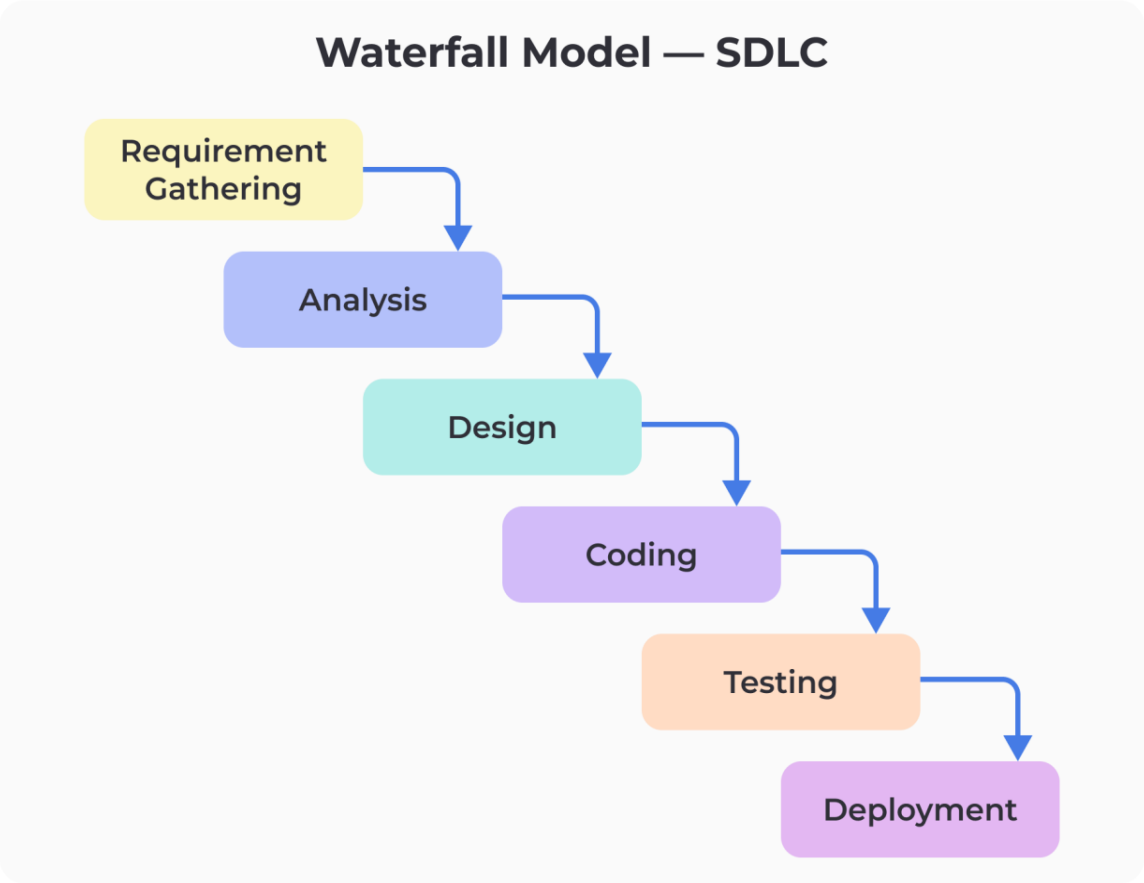


Fig. 1.8 Waterfall model

And here are examples of when the Waterfall model can and cannot be applied:

- In a factory that manufactures rivets for the fuselage of an aircraft, operators perform inspections to evaluate rivets on a conveyor belt. This score can reveal the percentage of defective rivets. Usually, this percentage is small and does not lead to the abandonment of the entire batch of rivets. So, most of the product can be released. The Waterfall model can be applied to this process, as this model involves testing at the end of the process, as the final stage.
- Now consider the same aircraft, but the product is the software that controls the display used by the crew. What happens next if many defects are found at the time of testing? Can we release only parts of the system? Obviously,

we cannot, because the aircraft's software must function according to the established criteria. Therefore, in this case, a different life cycle model is chosen.

Main featuresV-Models

The next SDLC model is the **V model.**Under this model, testing should begin as early in the life cycle as possible. The basic idea of the V-model is that development and testing tasks are relevant activities of equal importance. The two branches of the letter V symbolize this.

V-model for software development:

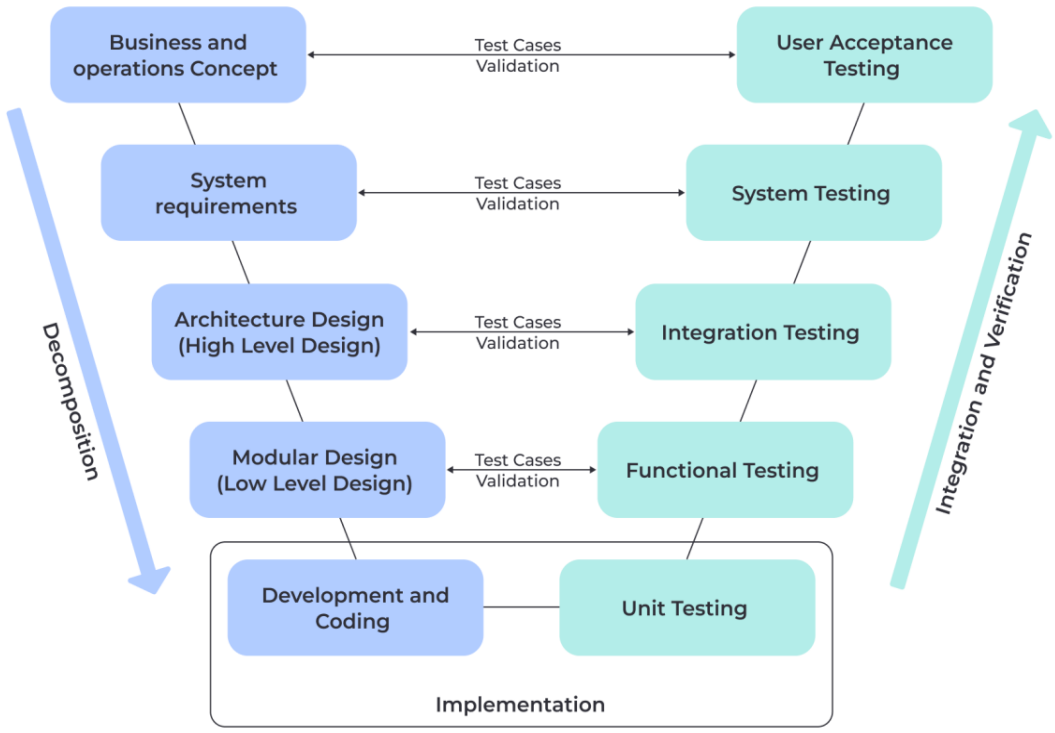


Fig. 1.9 V-model

The activities on the left side of the model are the activities known from the Waterfall model, and they are demand-oriented:

- **Requirement** specification — taking into account the needs of users.

- **Functional** specification is the definition of the functions required to meet the needs of the user.
- **Technical** specification is the technical design of the functions specified in a functional specification.
- **Program** specification is a detailed design of each module or block that will be built taking into account the required functionality.

So, we have analyzed the sequential models of software development, namely the Waterfall model and the V-model. It's time to move on to a completely different type of development model: iterative and incremental.

Iterative *and incremental development model.*

We can see that the sequential development model contains a complete set of functions and usually takes months or even years to deliver the finished product to stakeholders and users.

An increment is an operation that increments an existing variable (in our case, it is software or a separate software module) by one step.

Incremental development models include requirements statement, design, coding (development), and testing of the system in parts. This means that the functionality of the software increases incrementally. That is, some small part (increment) is constantly added to what is. An increment can be either a small improvement to an existing feature or a new functionality.

Incremental model:

- There are clear requirements;
- We divide the requirements into assemblies;
- We create the first major build and then build up the features until all the software is ready.

An increment is a gradual increase by a certain amount.

If Leonardo da Vinci had written his "La Gioconda" according to an incremental model, his process would have looked like this:

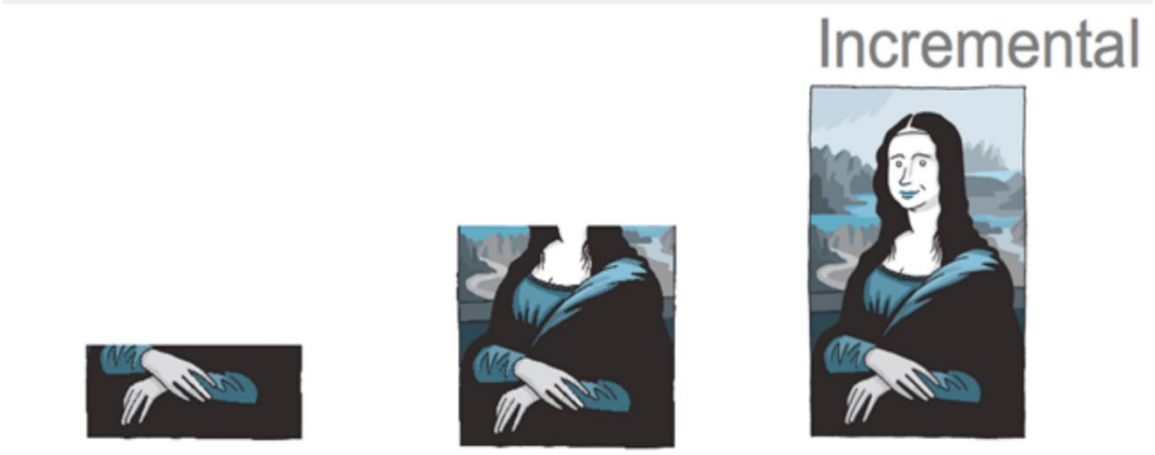


Рис. 1.10 Incremental model

So, during **the incremental** development model , some small piece of software is constantly being added. And what happens when an **iterative development model** is used.

An iterative development model is accomplished when groups of features are defined, designed, built, and tested together in a series of cycles, over a fixed period of time. Each iteration provides working software.

Рис. 1.11 Iterative development model

iterative development model:

- There are no complete requirements;
- Step by step, we increase the functionality;
- The main thing is that each assembly is workable.

If we go back to the analogy with the creation of a painting, then, according to the iterative model, it is more logical to paint a picture as follows: first a sketch, then the first rough strokes, and — the final delicate strokes. Thus, at each stage, there is a certain version of the completed picture:

Lysis. 1.12 Iterative model

Incremental and iterative development models do not have a clear work flow. Like, for example, in the Waterfall model, where the stages are clearly defined: requirements gathering > requirements analysis > design > coding > testing, and so on. Therefore, these models require special tools that would allow you to take full advantage of the incremental or iterative development model, and at the same time allow you to meet the deadlines and budget allocated for development. And there are such tools — these are agile methods of software development.

Agile software development methods:

Incremental and iterative models work on an **agile approach to software development or Agile** software development or **simply** Agile.

Agile is an agile software development methodology based on iterative development, the main advantage of which is to ensure high development productivity and respond quickly to changes.

Scrum and Kanban are the two most commonly used types of Agile.

Scrum — each iteration is usually relatively short (for example, days or a few weeks), and the addition of features is correspondingly small. For example, several improvements and/or two or three new features.

Kanban is implemented with or without iterations of a fixed length, upon completion of which either a single revision or a functionality or a group of functionality combined together is released.

Thus, agile approaches to software development allow you to effectively manage the development process, increase team productivity, and quickly adapt to changes.

Usually, a tester works with various projects and development methodologies during their career. When transitioning, for example, from banking software development to an artificial intelligence startup, a QA engineer will certainly face a change in methodology to a flexible one. Let's figure out what the advantages and difficulties will be.

Benefits for testers when transitioning to an agile development methodology:

- Focus on working software and high-quality code;
- Inclusion of testing as part and starting point of software development (test-driven development);
- Availability of business stakeholders to help testers decide about the expected behavior of the system;

- Self-organized teams in which the whole group is responsible for quality and gives testers more autonomy in their work;
- Simplicity of design that should be easier to test.

Significant challenges for testers when transitioning to an agile development approach:

- Testers who are used to working with well-documented requirements will develop tests based on a different type of testing — less formal and prone to change. The manifesto doesn't say that documentation is no longer needed or has no value, but it's often interpreted that way.
- As developers do more unit testing, it may seem that testers are not needed. But unit testing and acceptance testing, which is conducted only by business representatives, can miss serious problems. Systems testing, with its broader perspective and emphasis on non-functional testing, as well as end-to-end functional testing, is necessary even if it is not suitable for a sprint.
- The role of the tester is different. Because an agile team has less documentation and more face-to-face interaction, testers need to adapt to this style of work, which can be challenging for some. Testers can act primarily as test coaches for both stakeholders and developers who may not have sufficient knowledge about testing.
- While there is less testing to be done in one iteration than for the entire system, there is a constant shortage of time and less time to think about testing new features.
- As each increase is added to the existing working system, regression testing becomes extremely important and automation becomes more profitable.

Матеріали з розробки

Розробка дизайну та референсу

Пошук референсів є важливою частиною розробки дизайну сайту, оскільки він надає іншим ідеї та натхнення. Нижче наведено деякі особливості та поради, які можуть допомогти вам ефективно знаходити референси під час розробки дизайну сайту:

Визначте цілі та аудиторію:

- Розумійте, яку цільову аудиторію ви спрямовуєте на свій сайт.
- Визначте основні цілі вашого веб-сайту (інформаційний, електронна торгівля, розважальний і т. д.).

Досліджуйте конкуренцію:

- Переглядайте веб-сайти конкурентів, аналізуйте їх дизайн і функціональність.
- Виділіть ті аспекти, які вам подобаються або які можна поліпшити.
- Вивчайте тренди дизайну:
- Слідкуйте за актуальними трендами у веб-дизайні.
- Розглядайте роботи визначних дизайнерів та агентств, щоб залишатися в курсі нововведень.

Використовуйте онлайн-ресурси:

- Використовуйте платформи для пошуку референсів, такі як Dribbble, Behance, Pinterest.
- Створюйте збірки або дошки зі зразками дизайну, які вам подобаються.

Аналізуйте успішні веб-сайти:

- Розглядайте веб-сайти, які вже отримали позитивний відгук або мають високий трафік.
- Звертайте увагу на їхню структуру, навігацію та елементи дизайну.

Вивчайте мобільний дизайн:

- Враховуйте мобільні аспекти при пошуку референсів, оскільки багато користувачів використовують мобільні пристрої.
- Досліджуйте адаптивний та мобільний дизайн інших веб-сайтів.

Контекстуалізуйте референси:

- Розглядайте референси в контексті власного проекту.
- Розумійте, які аспекти референсу можна впровадити у вашому конкретному випадку.

Запитуйте відгуки та поради:

- Залучайте своїх колег або замовників до обговорення референсів.
- Поділіться своїми думками та запитайте зауваження для отримання об'єктивної думки.

Експериментуйте та адаптуйте:

- Не бійтеся експериментувати з ідеями з різних референсів.
- Адаптуйте ідеї, щоб вони відповідали вашому проекту та його унікальним вимогам.

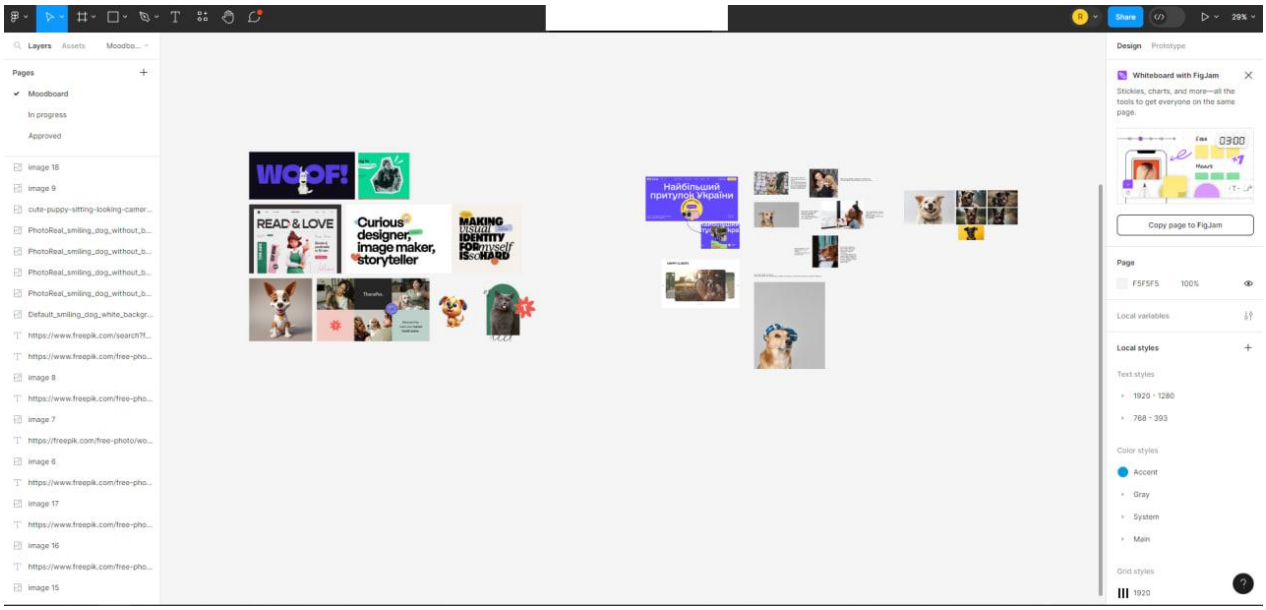


Рис. 4.1 Приклад референсу у Figma

Робота с кодом у репозиторію GitHub

Django - це відкритий веб-фреймворк для розробки веб-додатків на мові програмування Python. Він надає розробникам набір зручних інструментів та високорівневих абстракцій для швидкої розробки веб-сайтів і веб-додатків.

Основні особливості Django включають:

Модель-Вигляд-Контролер (MVC) архітектура:

- Django використовує архітектурний підхід Model-View-Controller (MVC), але відомий як Model-View-Template (MVT). Модель відповідає за обробку даних, Вигляд - за логіку відображення та контролер - за обробку вхідних даних.

Адміністративний інтерфейс:

- Django надає готовий адміністративний інтерфейс, який дозволяє легко створювати, змінювати та видаляти дані в базі даних без написання власного адмін-інтерфейсу.

ORM (Object-Relational Mapping):

- Django використовує ORM, що дозволяє вам взаємодіяти з базою даних, використовуючи об'єкти Python замість SQL-запитів.

Шаблонізація:

- Django використовує власну систему шаблонів для розділення логіки відображення та дизайну веб-сторінок.

Автентифікація та авторизація:

- Django надає вбудовану систему автентифікації та авторизації, що полегшує управління користувачами та їх правами доступу.

Автоматична адміністрація форм:

- Django вміє автоматично генерувати форми для введення даних на основі моделей даних.

Система URL-маршрутизації:

- Django використовує зручну систему маршрутизації, яка дозволяє визначити, як обробляти різні URL-шляхи.

Спрощена робота з базами даних:

- Django підтримує різні Системи Управління Базами Даних (СУБД), такі як SQLite, PostgreSQL, MySQL і інші.

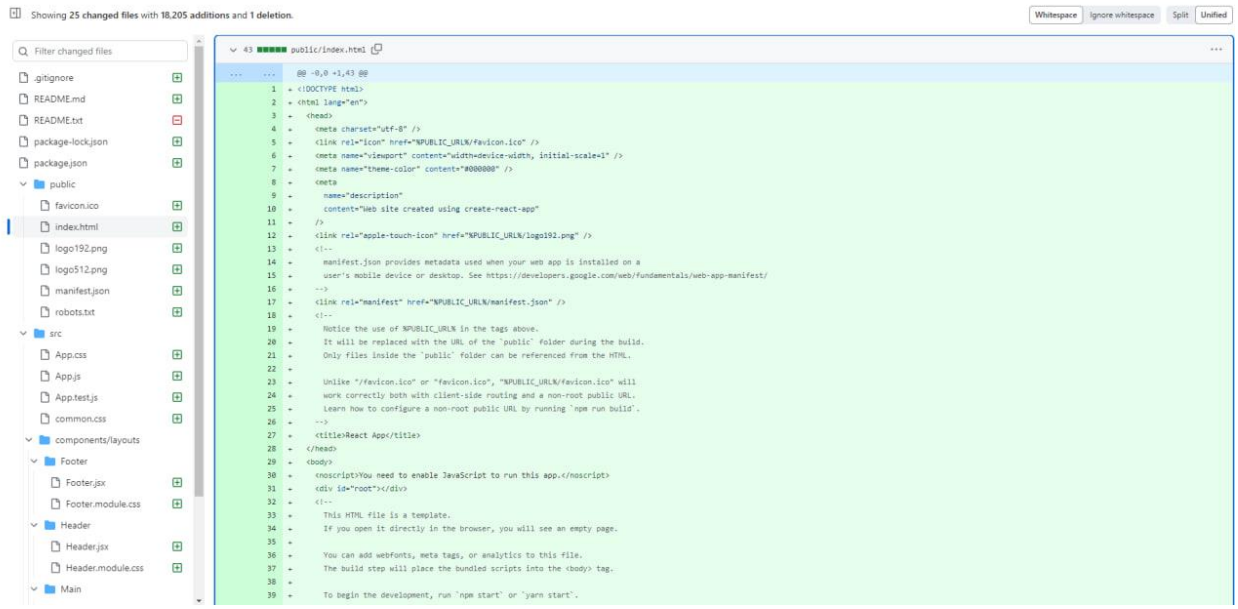


Рис. 4.2 Приклад роботи в репозиторію GitHub

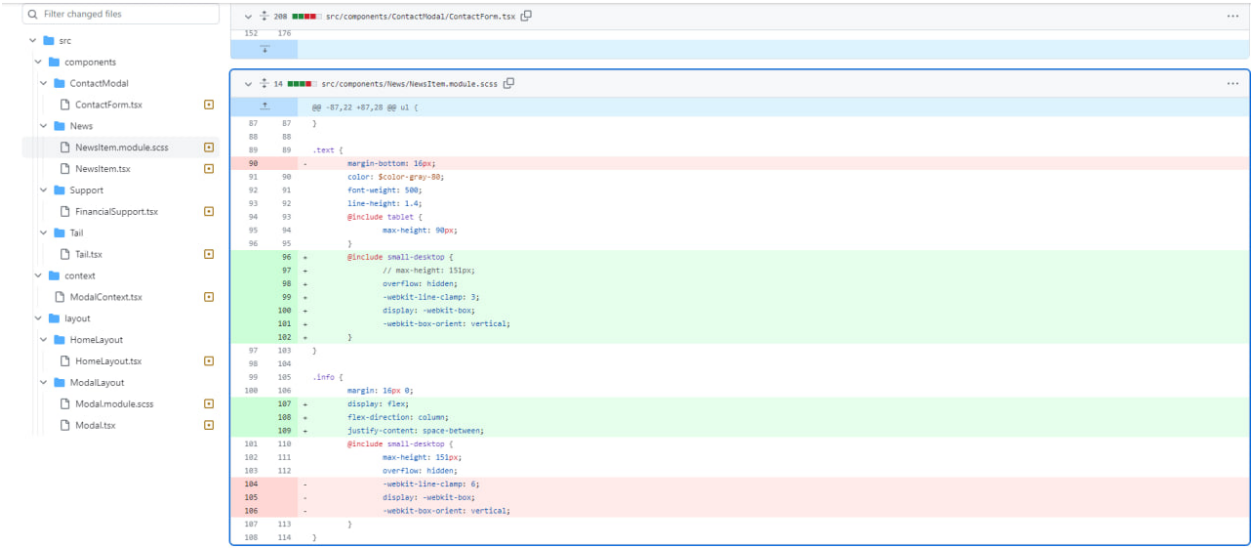


Рис. 4.3 Приклад роботи в репозиторію GitHub

Тестування API та DevTools

Тестування API (інтерфейсу програмування застосунків) є важливою частиною розробки програмного забезпечення для забезпечення правильності та стабільності вашого API. Тестування дозволяє виявляти помилки, переконатися, що API працює за очікуванням, і забезпечити зручність розробки. Основні типи тестування API включають:

1. Юніт-тести API:

- Тести, які перевіряють окремі компоненти або функції API, такі як функції обробки даних, перевірка автентифікації та авторизації, обробка помилок тощо. Юніт-тести дозволяють перевірити, чи працює кожна частина API окремо.
- 2. Інтеграційні тести API:

  - Тести, які перевіряють взаємодію між різними частинами API. Вони дозволяють визначити, чи взаємодіють різні ендпоінти чи сервіси так, як очікується.

3. Функціональні тести API:

  - Тести, які перевіряють, чи відповідає API очікуванням функціональності. Вони перевіряють, чи правильно обробляються запити та чи повертаються очікувані результати.

4. Тести витривалості та навантаження API:

  - Тести, які перевіряють, як API працює під навантаженням та витримує тривалу роботу. Це дозволяє виявити проблеми з продуктивністю та оптимізацією.

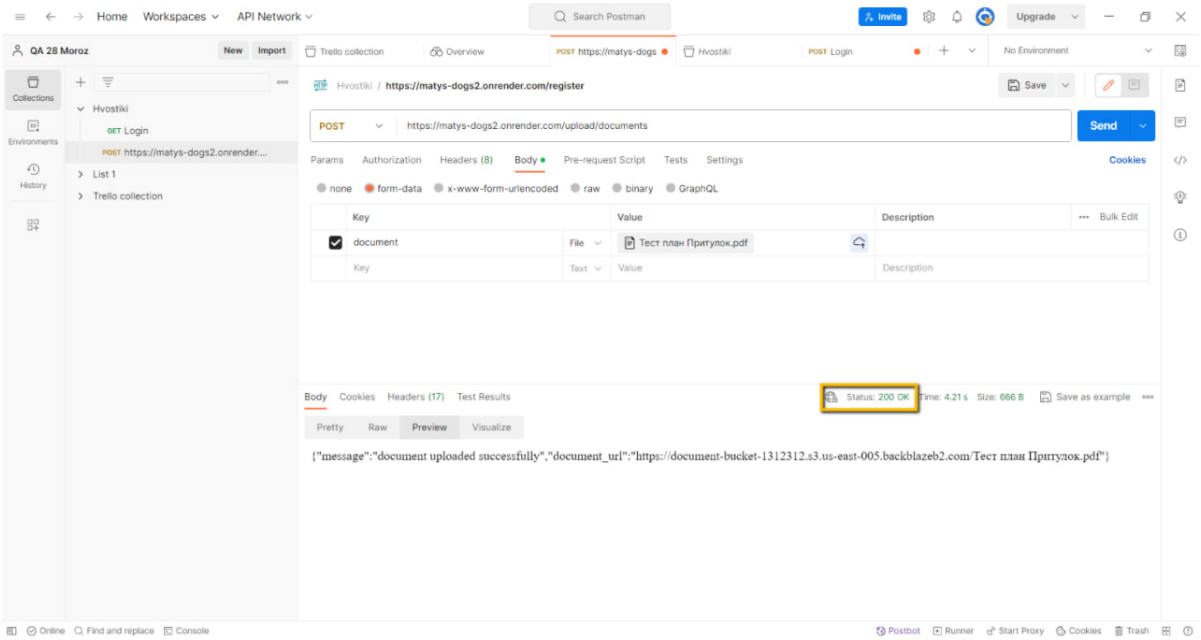
5. Тести безпеки API:

  - Тести, які перевіряють безпеку API, включаючи аутентифікацію, авторизацію, обробку недостовірних даних, запобігання атакам тощо.

Щоб виконати тести API, ви можете використовувати різні інструменти для автоматизації тестування, такі як:

- Postman: Інструмент для створення та виконання тестів API. Він дозволяє легко створювати запити, визначати тести та перевіряти результати.
- pytest: Бібліотека для тестування в Python, яка може бути використана для написання та виконання тестів API.
- JUnit та TestNG: Для тестування в Java, ці бібліотеки дозволяють створювати і виконувати тести API.
- Swagger/OpenAPI: Якщо ваш API документовано за допомогою Swagger чи OpenAPI, ви можете використовувати цю документацію для генерації автоматичних тестів.

Налаштуйте тести так, щоб вони відображали бізнес-логіку вашого API та визначали його поведінку відповідно до



очікувань.

Рис. 4.4 Тестуування додавання файлу на сервер через Postman

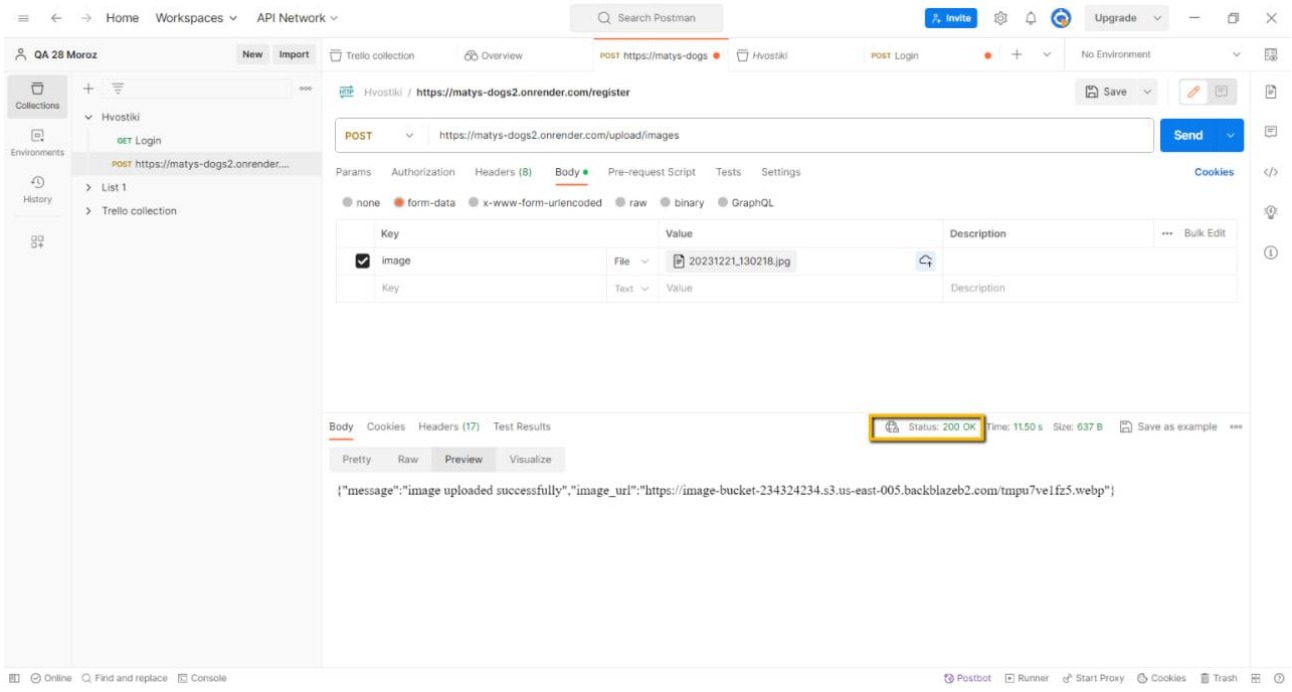


Рис. 4.5 Тестуування додавання файлу на сервер через Postman

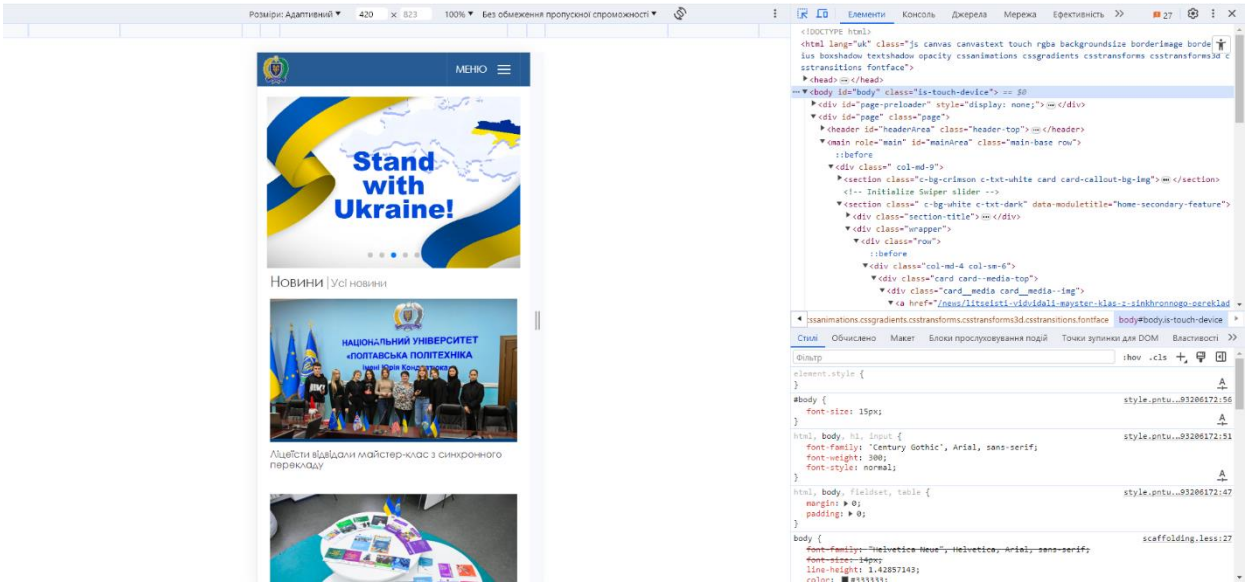


Рис. 4.6 Робота в DevTools

"DevTools" або інструменти розробника, як правило, вказують на набір засобів для налагодження та оптимізації веб-сторінок, які вбудовані в веб-браузери. Ці інструменти надають розробникам набір функцій для інспектування, налагодження та оптимізації веб-сторінок та веб-додатків.

Нижче подано огляд основних можливостей, які зазвичай знаходяться в інструментах розробника браузера:

1. Інспектування елементів:

- Розробники можуть інспектувати та маніпулювати HTML та CSS веб-сторінки в реальному часі. Це включає перегляд об'єктної моделі документа (DOM) та стилів, застосованих до елементів.

2. Консоль:

- Консоль JavaScript дозволяє розробникам реєструвати інформацію, виконувати команди JavaScript та виявляти помилки. Це потужний інструмент для налагодження та експериментування з кодом.

3. Моніторинг мережі:

- DevTools надають висновки про активність мережі, дозволяючи розробникам аналізувати продуктивність веб-сторінки за допомогою інформації про мережеві запити, час відгуку та розміри ресурсів.

4. Джерела (відлагоджувач):

- Розробники можуть встановлювати точки зупинки, крокувати код та інспектувати змінні в реальному часі. Це необхідно для виявлення та виправлення проблем у коді.

5. Профілювання продуктивності:

- DevTools пропонують інструменти для аналізу та оцінки продуктивності веб-сторінок. Це включає графіки для завантаження, виконання скриптів, відображення та використання пам'яті. Профілювання продуктивності допомагає виявляти та оптимізувати швидкість веб-сторінок.

6. Панель програм:

- Надає висновки про ресурси, пов'язані з програмою, такі як робочі робітники, кеш та сховища. Розробники можуть відстежувати та усувати проблеми, пов'язані з можливостями офлайн та зберіганням даних.

Це лише кілька основних можливостей, які надають інструменти розробника веб-браузера. Ці інструменти грають важливу роль у розробці, дозволяючи розробникам ефективно налагоджувати, оптимізовувати та перевіряти якість веб-додатків.

УДК 681.518
***М.А. Штомпель**, к.т.н., професор,*
***С.Ю. Кальченко**, магістрант*
Національний університет «Полтавська політехніка імені Юрія Кондратюка»

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДПРИЄМСТВА НА ОСНОВІ WEB-ТЕХНОЛОГІЙ

Розробка інформаційної системи підприємства на основі web-технологій - це процес створення програмного забезпечення, яке дозволяє підприємству ефективно управляти своєю діяльністю за допомогою інтернет-технологій. Така система може включати в себе різноманітні функції, такі як управління обліком, замовленнями, складом, фінансами, комунікацією з клієнтами і співробітниками, аналітику та звітність.

Основні кроки, які слід виконати для розробки інформаційної системи підприємства на основі web-технологій:

− Визначення вимог: Першим етапом є збір і аналіз вимог до системи. Важливо з'ясувати, які функції повинна виконувати система, які процеси повинна оптимізувати, і які потреби мають клієнти і співробітники підприємства.

− Проектування: На цьому етапі розробляється архітектура системи, визначаються технології, які будуть використовуватися, та структура бази даних. − Розробка: Розробка включає в себе створення програмного коду, інтерфейсу користувача, бази даних та інших компонентів системи. Для webтехнологій, таких як HTML, CSS, JavaScript, можуть використовуватися різні фреймворки і бібліотеки.

− Тестування: Після розробки систему слід протестувати на відповідність вимогам, а також на виявлення помилок і уразливостей.

− Розгортання: Після успішного тестування систему можна розгорнути на сервері, щоб користувачі могли отримувати доступ до неї через веб-браузери.

− Підтримка і поновлення: Після впровадження системи в експлуатацію важливо забезпечити її підтримку і регулярно вносити оновлення для покращення функціональності та безпеки.

Web-технології дозволяють забезпечити доступ до інформаційної системи з будь-якого пристрою, підключеного до Інтернету, що робить їх дуже зручними для підприємств з розподіленою структурою або багатьма філіями. Також вони спрощують можливість інтеграції з іншими системами та забезпечують можливість забезпечити безпеку даних та контроль доступу.

Розробка інформаційної системи підприємства на основі web-технологій - це складний і багатоетапний процес, і для його успішної реалізації зазвичай залучають професійних розробників, проектних менеджерів і тестувальників.

Загальна успішність розробки інформаційної системи на основі webтехнологій залежить від того, наскільки добре вона відповідає потребам підприємства і користувачів. Добре спланований та реалізований проект може значно полегшити роботу підприємства і покращити його конкурентоспроможність

ЛІТЕРАТУРА:

1. MDN Web Docs: Документація та приклади з HTML, CSS та JavaScript.
[Електронний ресурс]- Режим доступу:https://developer.mozilla.org/ru/

2. Stack Overflow: Форум для розробників з відповідями на технічні питання.
[Електронний ресурс] - Режим доступу:https://stackoverflow.com/

3. GitHub: Платформа для спільної розробки програмного забезпечення
[Електронний ресурс]- Режим доступу: https://github.com/

DEVELOPMENT OF AN ENTERPRISE INFORMATION SYSTEM BASED ON WEB TECHNOLOGIES

M. Shtompel, *Doctor of Science, Professor,*
S. Kalchenko*, master's student*
Yuriy Kondratyuk Poltava Polytechnic National University

Міністерство освіти та науки України
Національний університет «Полтавська політехніка імені Юрія Кондратюка»

Кафедра автоматики, електроніки та телекомунікацій

Розробка інформаційної системи підприємства на основі web-технологій

Кваліфікаційна робота магістра

Виконав:
Студент 601ТТ групи Кальченко С.Ю.

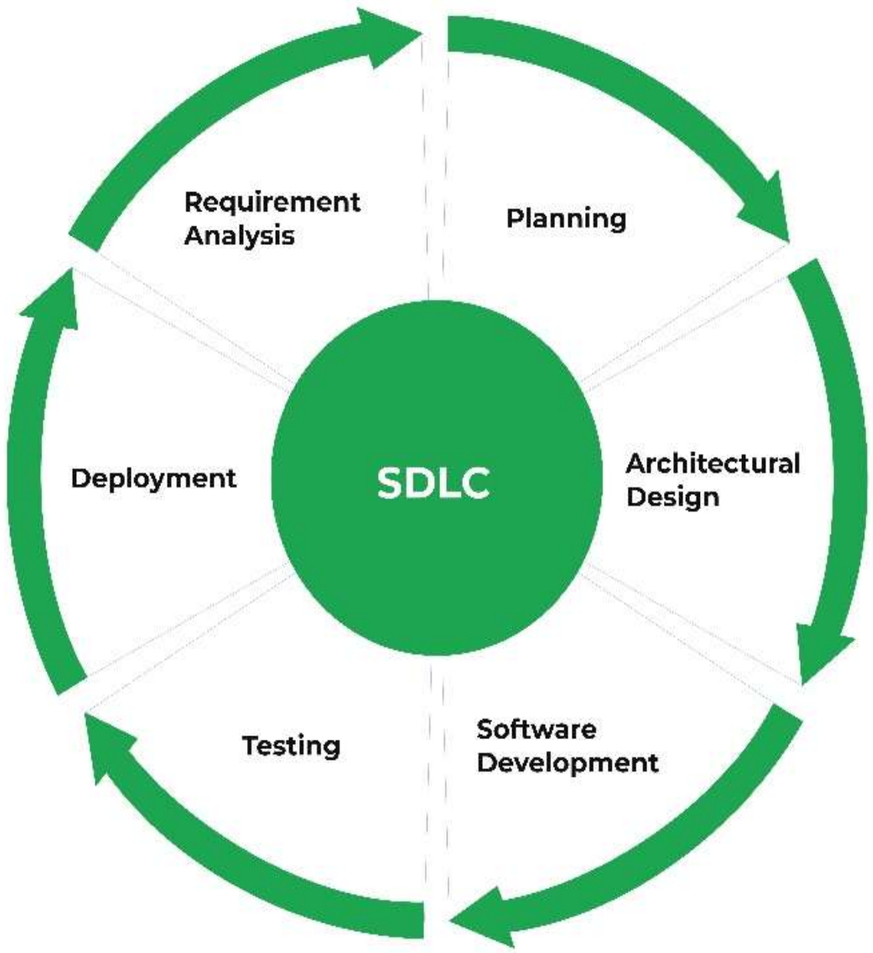
Керівник:
канд. техн. наук Штомпель М.А.

Полтава 2023

Розробка інформаційної системи підприємства на основі web-технологій

У сучасному діловому світі, що швидко змінюється, ключовим фактором успіху є вміння ефективно використовувати інноваційні технології для вдосконалення внутрішніх процесів і надання якісних послуг клієнтам. Інформаційні системи, розроблені на основі передових веб-технологій, визначають новий стандарт у сфері автоматизації та оптимізації бізнес-процесів.

Метою даної магістерської роботи є створення та впровадження інформаційної системи, яка відповідає потребам та завданням нашого підприємства. Розроблено комплексний підхід до вдосконалення роботи організації, який охоплює всі аспекти від внутрішніх комунікацій до взаємодії з клієнтами.

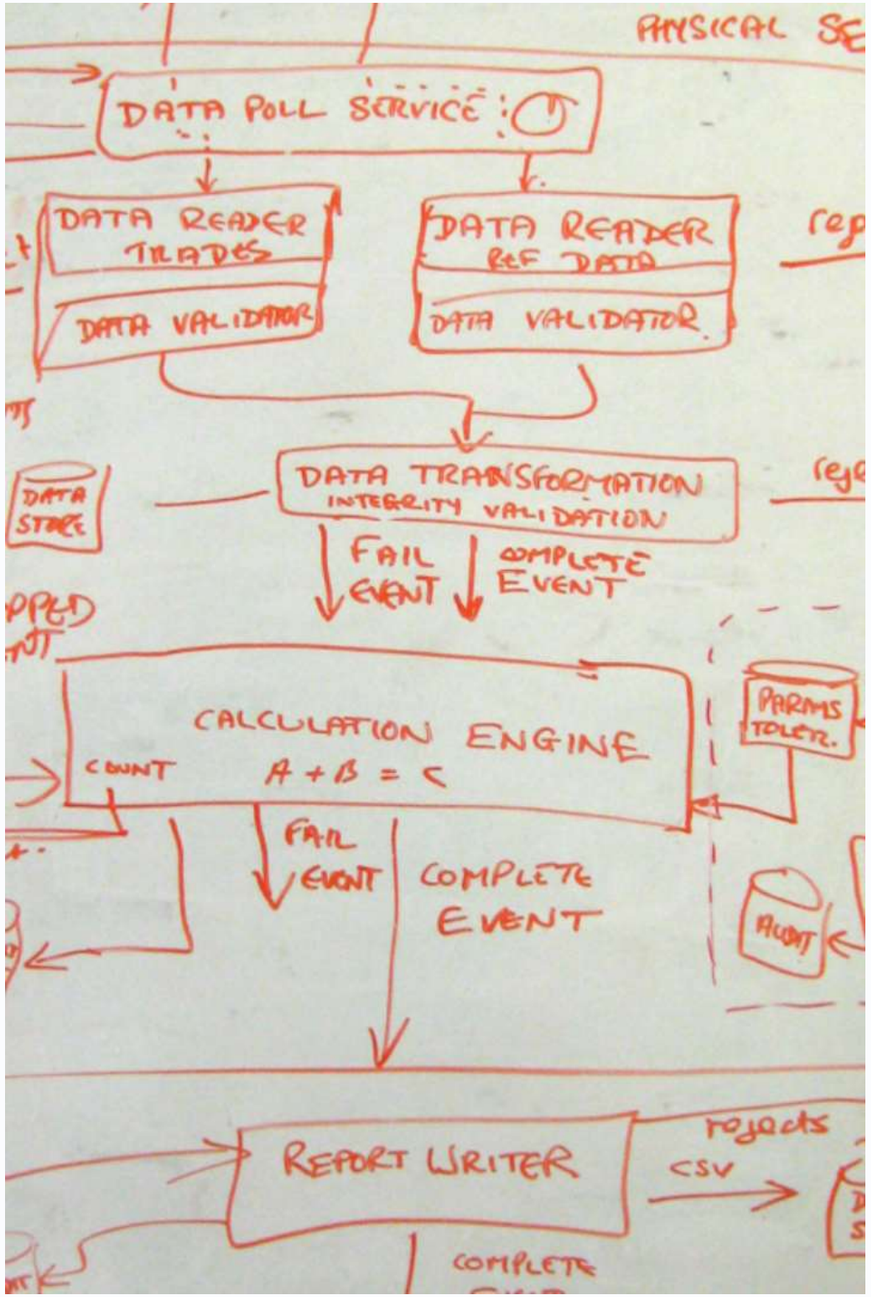


Збір вимог та аналіз (Requirement Analysis)

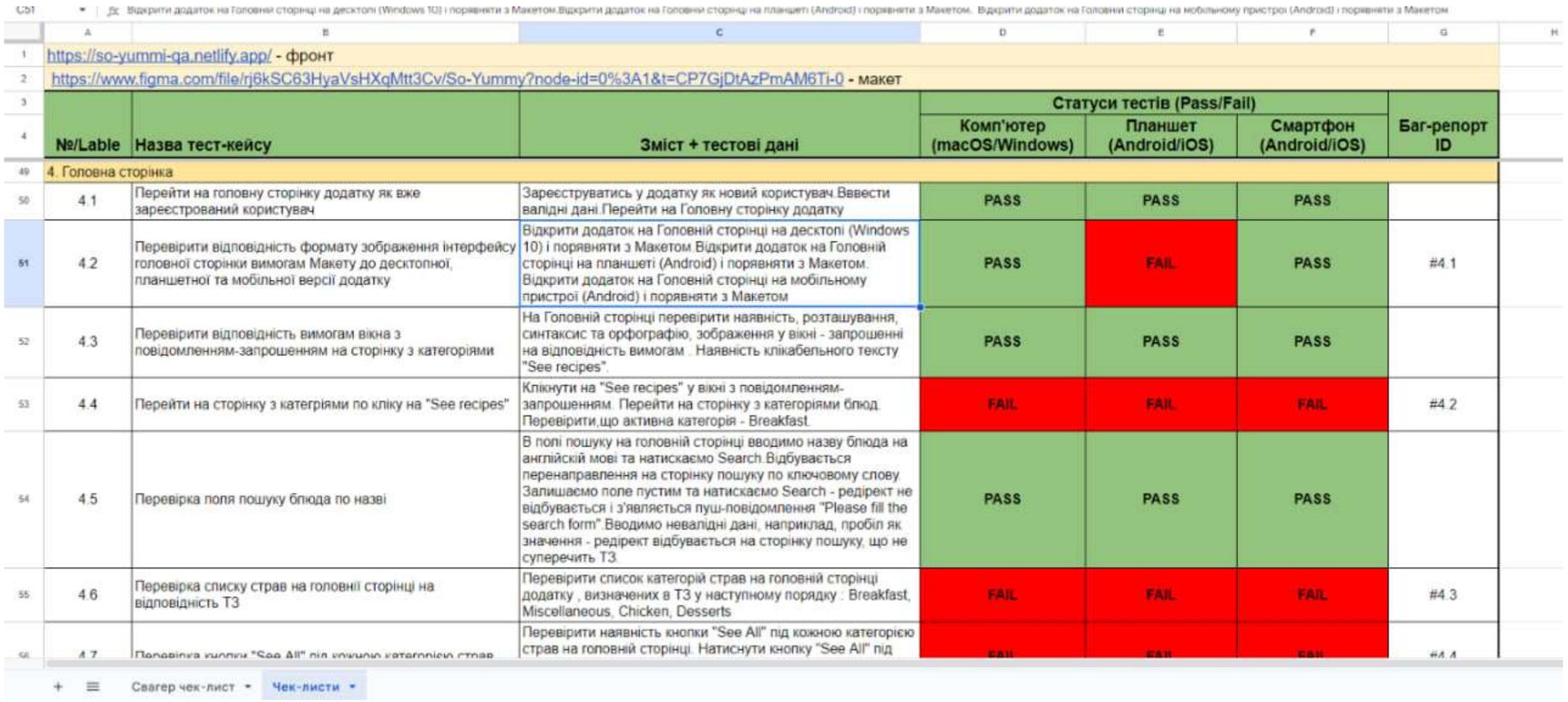
На цьому етапі команда отримує список вимог (зазвичай від замовника чи менеджера проєкту). Тестувальник працює над статичним тестуванням вимог: досліджує їх повноту, логічність, однозначність і визначеність.

Планування (Planning)

На цьому етапі відбувається пошук слабких місць у тестовому покритті (частини функціоналу, на яких можуть бути певні труднощі з покриттям їх тестами) і виявляє можливі ризики. Ризики можуть бути викликані різними причинами, завдання тестувальника на цьому етапі — визначити ці ризики та довести до відома відповідальних осіб.

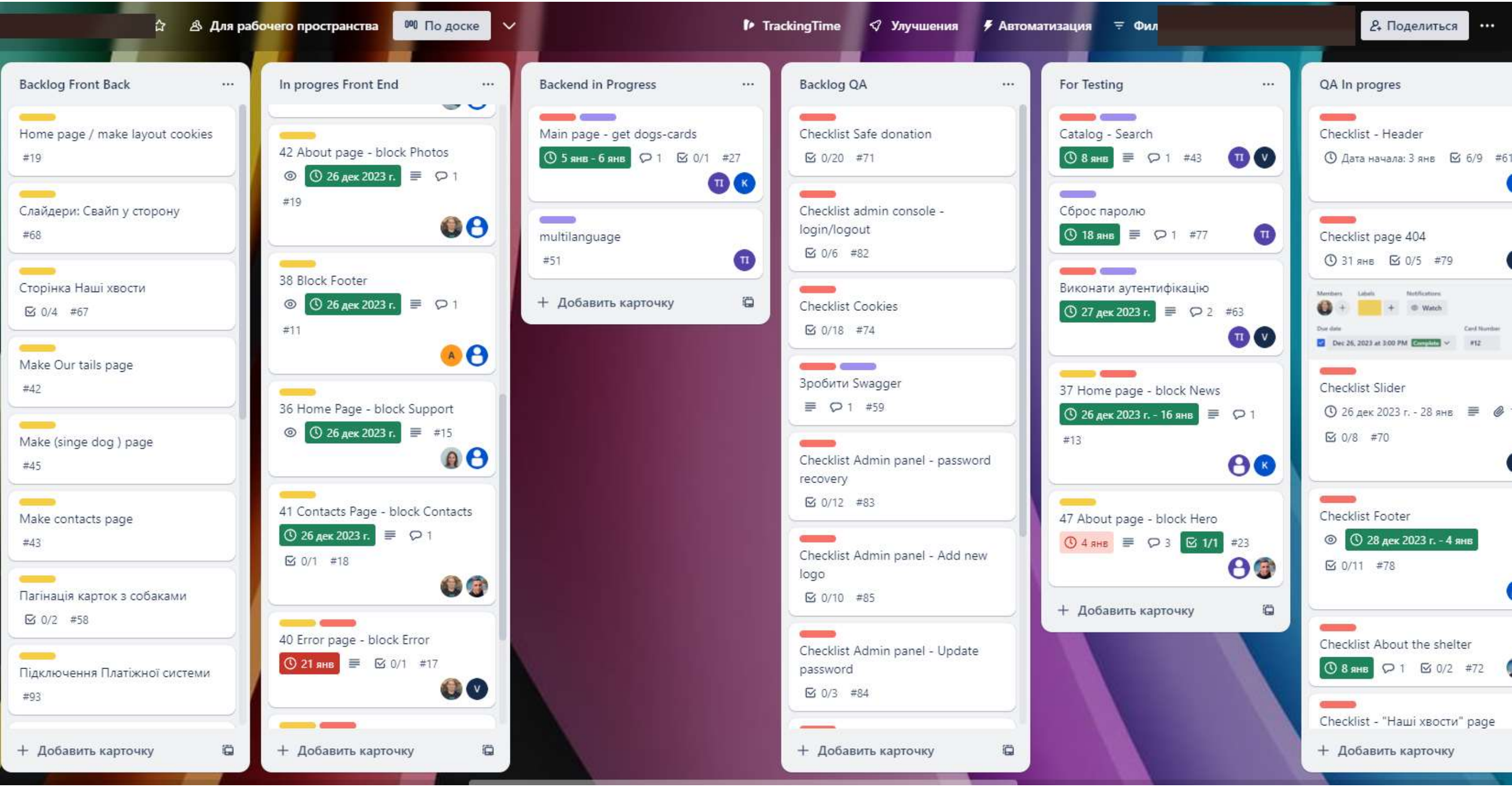


Приклад написання документації



	A	B	C	D	E	F	G	H
1	https://so-yummi-qa.netlify.app/ - фронт							
2	https://www.figma.com/file/rj6kSC63HyaVsHXqMtt3Cv/So-Yummy?node-id=0%3A1&t=CP7GjDtAzPmAM6Ti-0 - макет							
3				Статуси тестів (Pass/Fail)				
4	№/Lable	Назва тест-кейсу	Зміст + тестові дані	Комп'ютер (macOS/Windows)	Планшет (Android/iOS)	Смартфон (Android/iOS)	Баг-репорт ID	
49	4. Головна сторінка							
50	4.1	Перейти на головну сторінку додатку як вже зареєстрований користувач	Зареєструватись у додатку як новий користувач.Ввести валідні дані.Перейти на Головну сторінку додатку	PASS	PASS	PASS		
51	4.2	Перевірити відповідність формату зображення інтерфейсу головної сторінки вимогам Макету до десктопної, планшетної та мобільної версії додатку	Відкрити додаток на Головній сторінці на десктопі (Windows 10) і порівняти з Макетом.Відкрити додаток на Головній сторінці на планшеті (Android) і порівняти з Макетом. Відкрити додаток на Головній сторінці на мобільному пристрої (Android) і порівняти з Макетом	PASS	FAIL	PASS	#4.1	
52	4.3	Перевірити відповідність вимогам вікна з повідомленням-запрошенням на сторінку з категоріями	На Головній сторінці перевірити наявність, розташування, синтаксис та орфографію, зображення у вікні - запрошенні на відповідність вимогам . Наявність клікабельного тексту "See recipes".	PASS	PASS	PASS		
53	4.4	Перейти на сторінку з категріями по кліку на "See recipes"	Клікнути на "See recipes" у вікні з повідомленням-запрошенням. Перейти на сторінку з категоріями блюд. Перевірити,що активна категорія - Breakfast.	FAIL	FAIL	FAIL	#4.2	
54	4.5	Перевірка поля пошуку блюда по назві	В полі пошуку на головній сторінці вводимо назву блюда на англійській мові та натискаємо Search.Відбувається перенаправлення на сторінку пошуку по ключовому слову Залишаємо поле пустим та натискаємо Search - редірект не відбувається і з'являється пуш-повідомлення "Please fill the search form".Вводимо невалідні дані, наприклад, пробіл як значення - редірект відбувається на сторінку пошуку, що не суперечить ТЗ.	PASS	PASS	PASS		
55	4.6	Перевірка списку страв на головній сторінці на відповідність ТЗ	Перевірити список категорій страв на головній сторінці додатку , визначених в ТЗ у наступному порядку : Breakfast, Miscellaneous, Chicken, Desserts	FAIL	FAIL	FAIL	#4.3	
56	4.7	Перевірка кнопки "See All" під кожною категорією страв	Перевірити наявність кнопки "See All" під кожною категорією страв на головній сторінці. Натиснути кнопку "See All" під	FAIL	FAIL	FAIL	#4.4	

Свагер чек-лист | Чек-листи


Для рабочего пространства
По доске
TrackingTime
Улучшения
Автоматизация
Поделиться
Backlog Front Back
Home page / make layout cookies
#19
Слайдери: Свайп у сторону
#68
Сторінка Наші хвости
0/4 #67
Make Our tails page
#42
Make (singe dog) page
#45
Make contacts page
#43
Пагінація карток з собаками
0/2 #58
Підключення Платіжної системи
#93
Добавить карточку
In progres Front End
42 About page - block Photos
26 дек 2023 г.
1
#19
38 Block Footer
26 дек 2023 г.
1
#11
36 Home Page - block Support
26 дек 2023 г.
#15
41 Contacts Page - block Contacts
26 дек 2023 г.
1
0/1 #18
40 Error page - block Error
21 янв
0/1 #17
Добавить карточку
Backend in Progress
Main page - get dogs-cards
5 янв - 6 янв
1
0/1 #27
multilanguage
#51
Добавить карточку
Backlog QA
Checklist Safe donation
0/20 #71
Checklist admin console - login/logout
0/6 #82
Checklist Cookies
0/18 #74
Зробити Swagger
1 #59
Checklist Admin panel - password recovery
0/12 #83
Checklist Admin panel - Add new logo
0/10 #85
Checklist Admin panel - Update password
0/3 #84
Добавить карточку
For Testing
Catalog - Search
8 янв
1 #43
Сброс паролю
18 янв
1 #77
Виконати аутентифікацію
27 дек 2023 г.
2 #63
37 Home page - block News
26 дек 2023 г. - 16 янв
1
#13
47 About page - block Hero
4 янв
3
1/1 #23
Добавить карточку
QA In progres
Checklist - Header
Дата начала: 3 янв
6/9 #61
Checklist page 404
31 янв
0/5 #79
Checklist Slider
26 дек 2023 г. - 28 янв
0/8 #70
Checklist Footer
28 дек 2023 г. - 4 янв
0/11 #78
Checklist About the shelter
8 янв
1
0/2 #72
Checklist - "Наші хвости" page
Добавить карточку

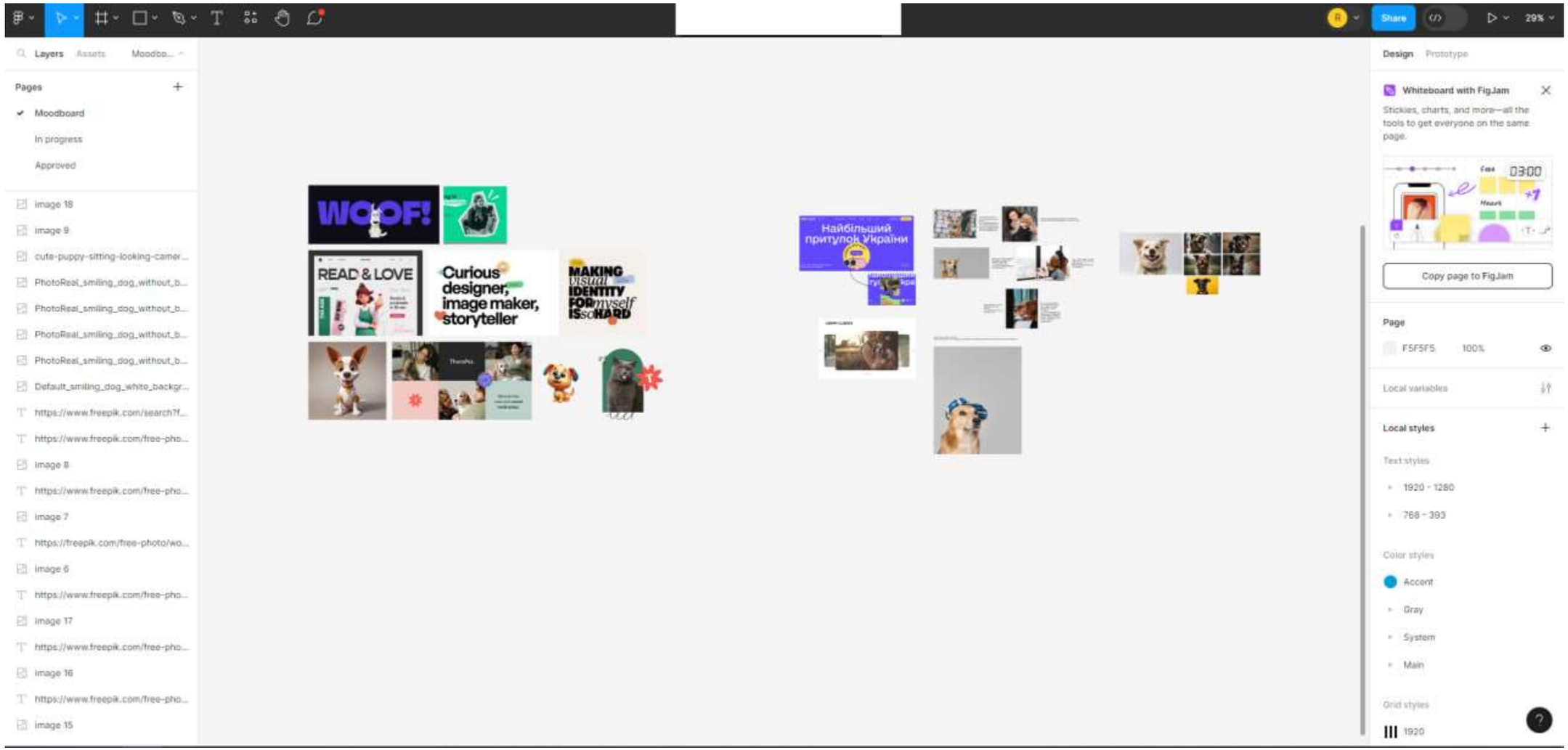
Процес проектування інтерфейсу

1. Визначення ЦА, складання детальної характеристики користувача.
2. Структурування інформації, побудова інформаційної архітектури.
3. Проектування каркасу, прототипування
4. Створення UI-дизайну
5. Тестування користувацького інтерфейсу.

Проєктування (дизайн) (Architectural Design)

Один із найважливіших складових етапів, тому що користувачеві доведеться постійно взаємодіяти з системою. На даному етапі тестувальник перевіряє існуючі прототипи ПЗ на відповідність вимогам замовника, коректність відображення візуальних елементів і зручність використання.

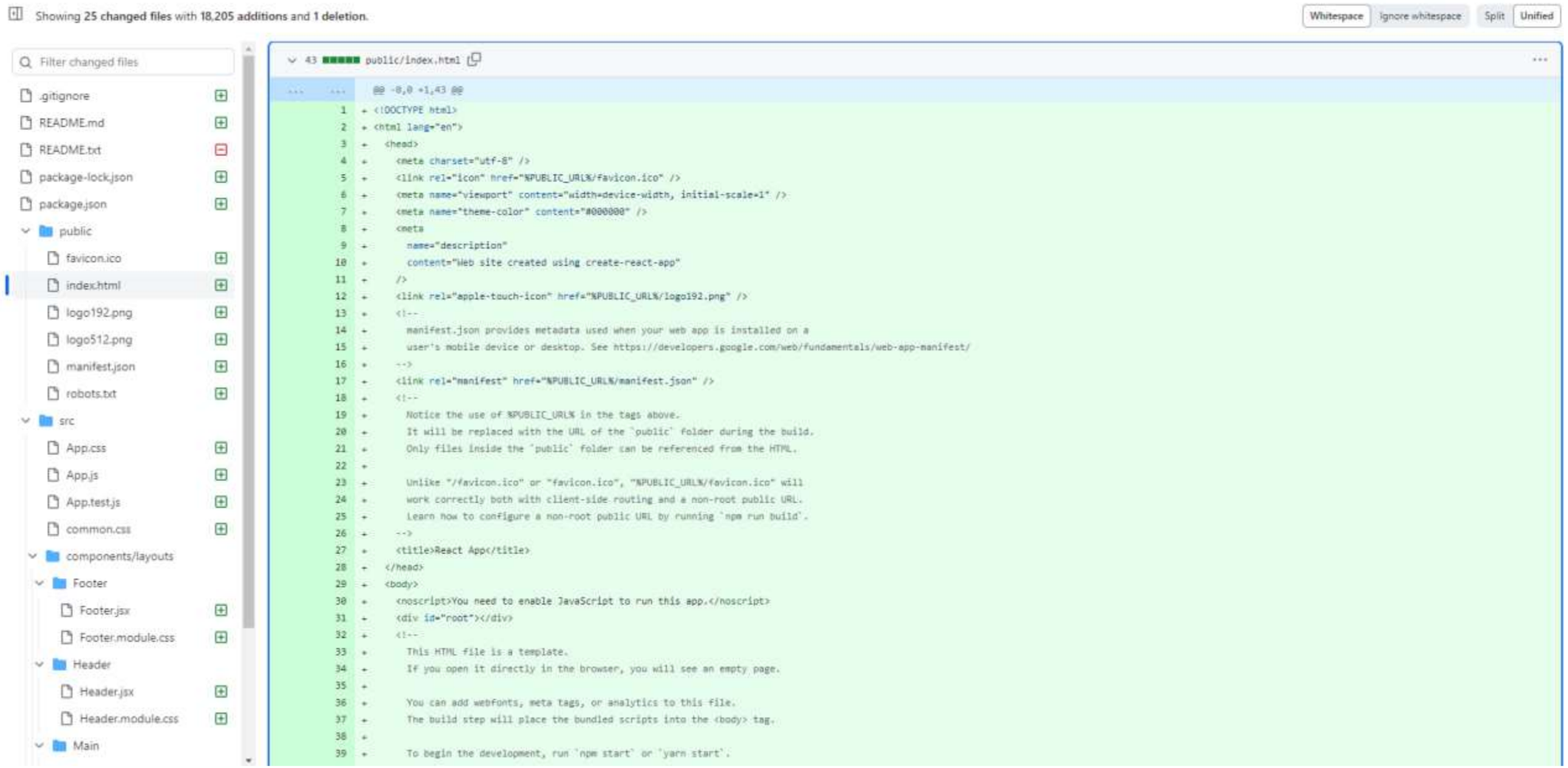
Приклад створення макету

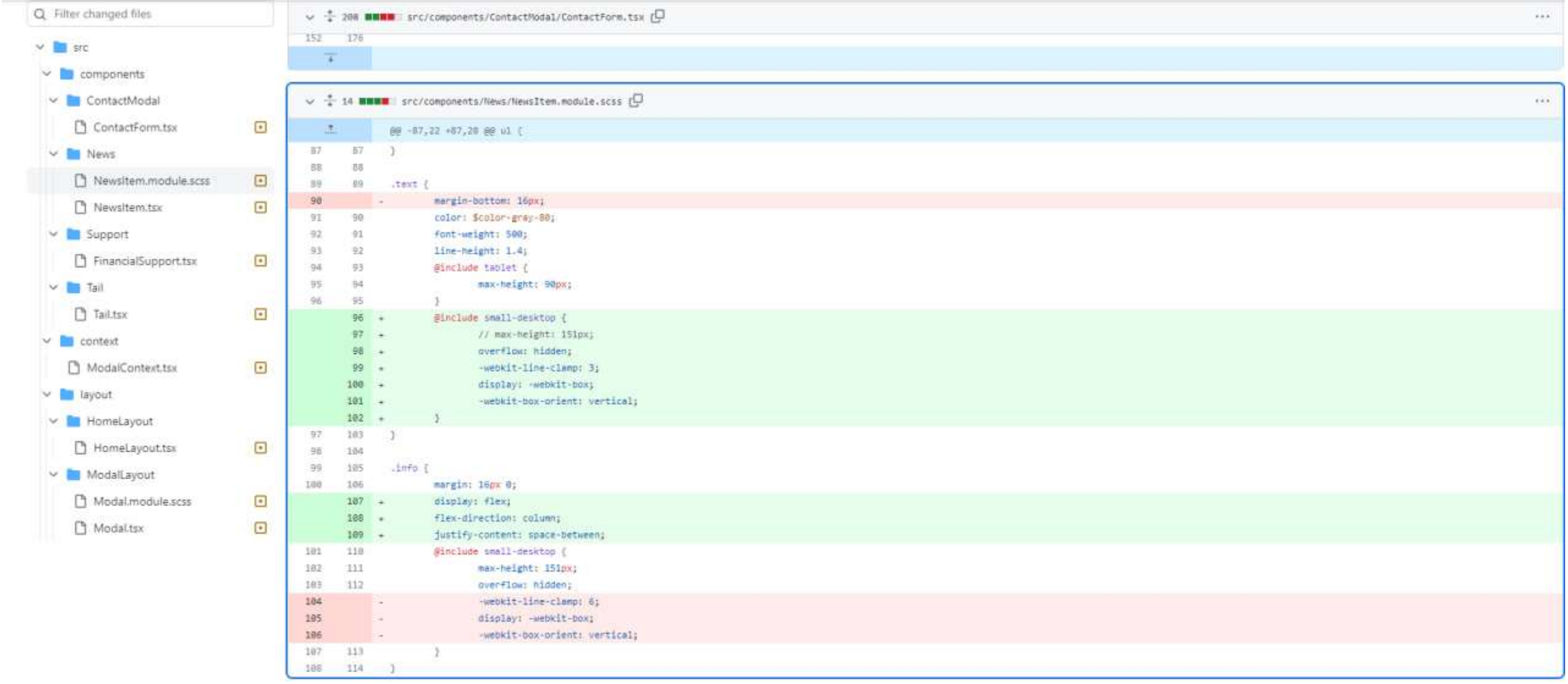


Розробка (Software Development)

Під час процесу розробки системи необхідно провести модульне (перевірка окремих компонентів системи), інтеграційне (взаємодія певних компонентів між собою) тестування.

Приклад роботи в репозиторію GitHub



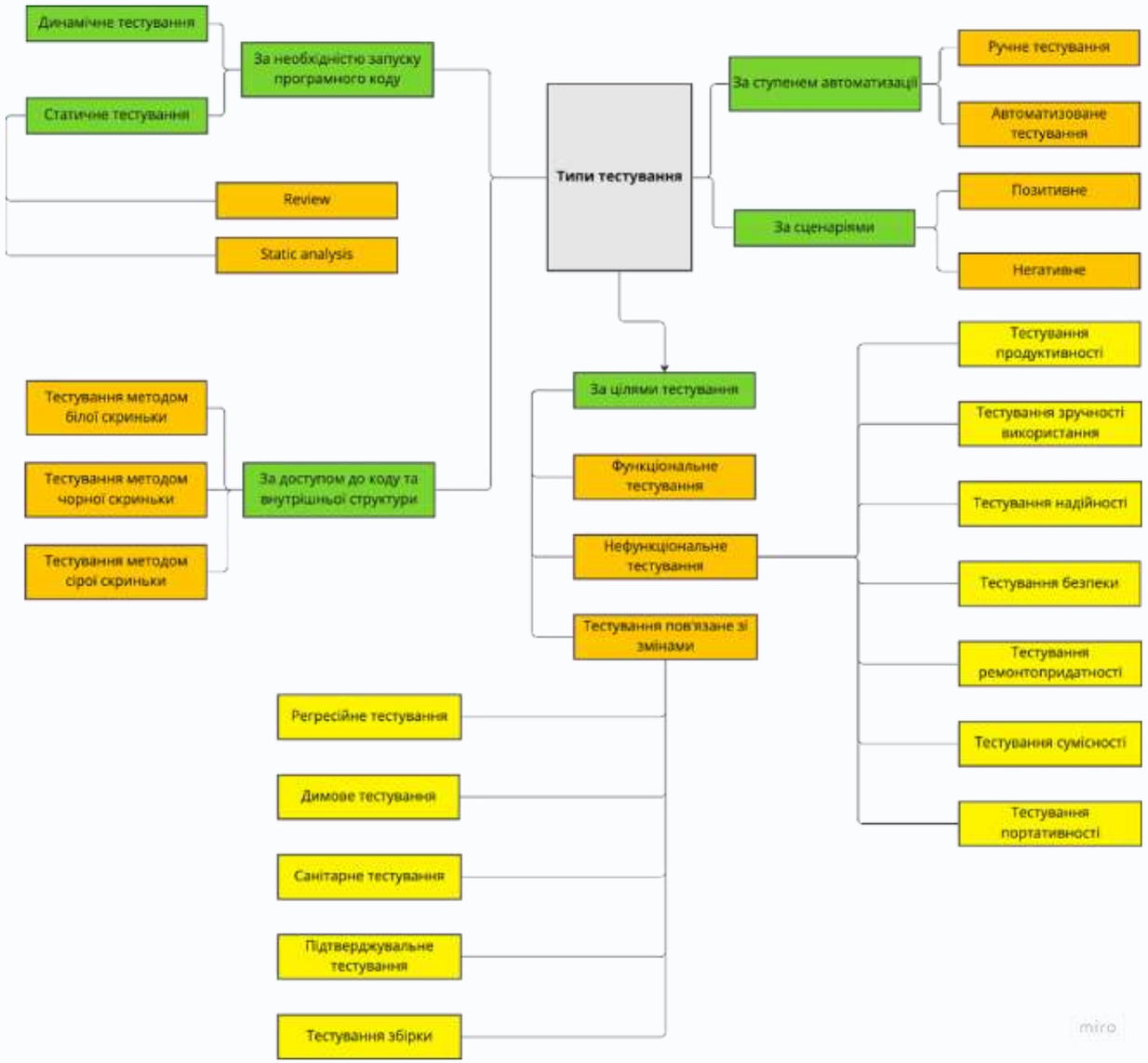

Filter changed files
src
components
ContactModal
ContactForm.tsx
News
NewsItem.module.scss
NewsItem.tsx
Support
FinancialSupport.tsx
Tail
Tail.tsx
context
ModalContext.tsx
layout
HomeLayout
HomeLayout.tsx
ModalLayout
Modal.module.scss
Modal.tsx
208 src/components/ContactModal/ContactForm.tsx
152 176
14 src/components/News/NewsItem.module.scss
@@ -87,22 +87,28 @@ ul {
87 87 }
88 88
89 89 .text {
90 - margin-bottom: 16px;
91 90 color: $color-gray-80;
92 91 font-weight: 500;
93 92 line-height: 1.4;
94 93 @include tablet {
95 94 max-height: 90px;
96 95 }
96 + @include small-desktop {
97 + // max-height: 151px;
98 + overflow: hidden;
99 + -webkit-line-clamp: 3;
100 + display: -webkit-box;
101 + -webkit-box-orient: vertical;
102 + }
97 103 }
98 104
99 105 .info {
100 106 margin: 16px 0;
107 + display: flex;
108 + flex-direction: column;
109 + justify-content: space-between;
101 110 @include small-desktop {
102 111 max-height: 151px;
103 112 overflow: hidden;
104 - -webkit-line-clamp: 6;
105 - display: -webkit-box;
106 - -webkit-box-orient: vertical;
107 113 }
108 114 }

Розробка серверної частини (backend)

Розробка серверної частини, або backend, є однією з найважливіших складових процесу створення інформаційної системи підприємства на основі web-технологій. Backend відповідає за обробку бізнес-логіки, зберігання даних та взаємодію з базою даних. На цій сторінці ми розглянемо ключові етапи розробки серверної частини.

Тестування (Testing)

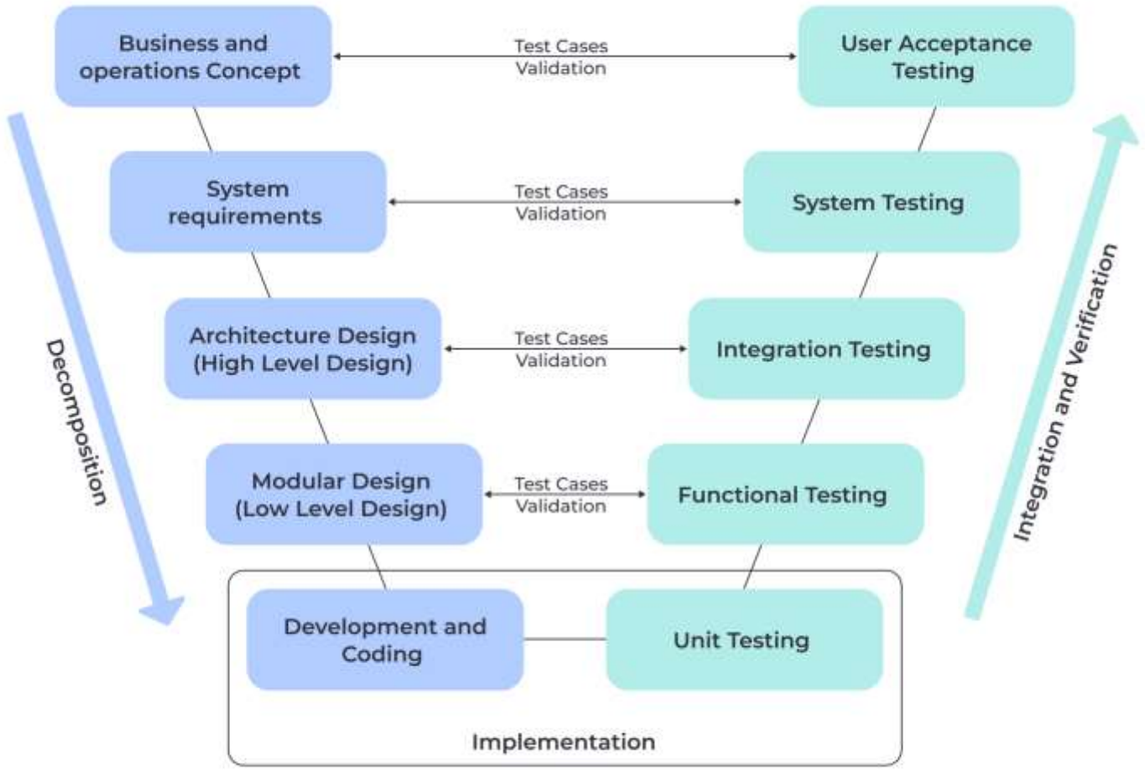
На даному етапі тестувальники перевіряють систему на наявність дефектів незалежно від того, чи відбувалось це раніше. Проводиться повне тестування інтерфейсу та функціоналу продукту. Всі виявлені помилки повинні бути задокументовані в системі відслідковування багів — баг-трекер.



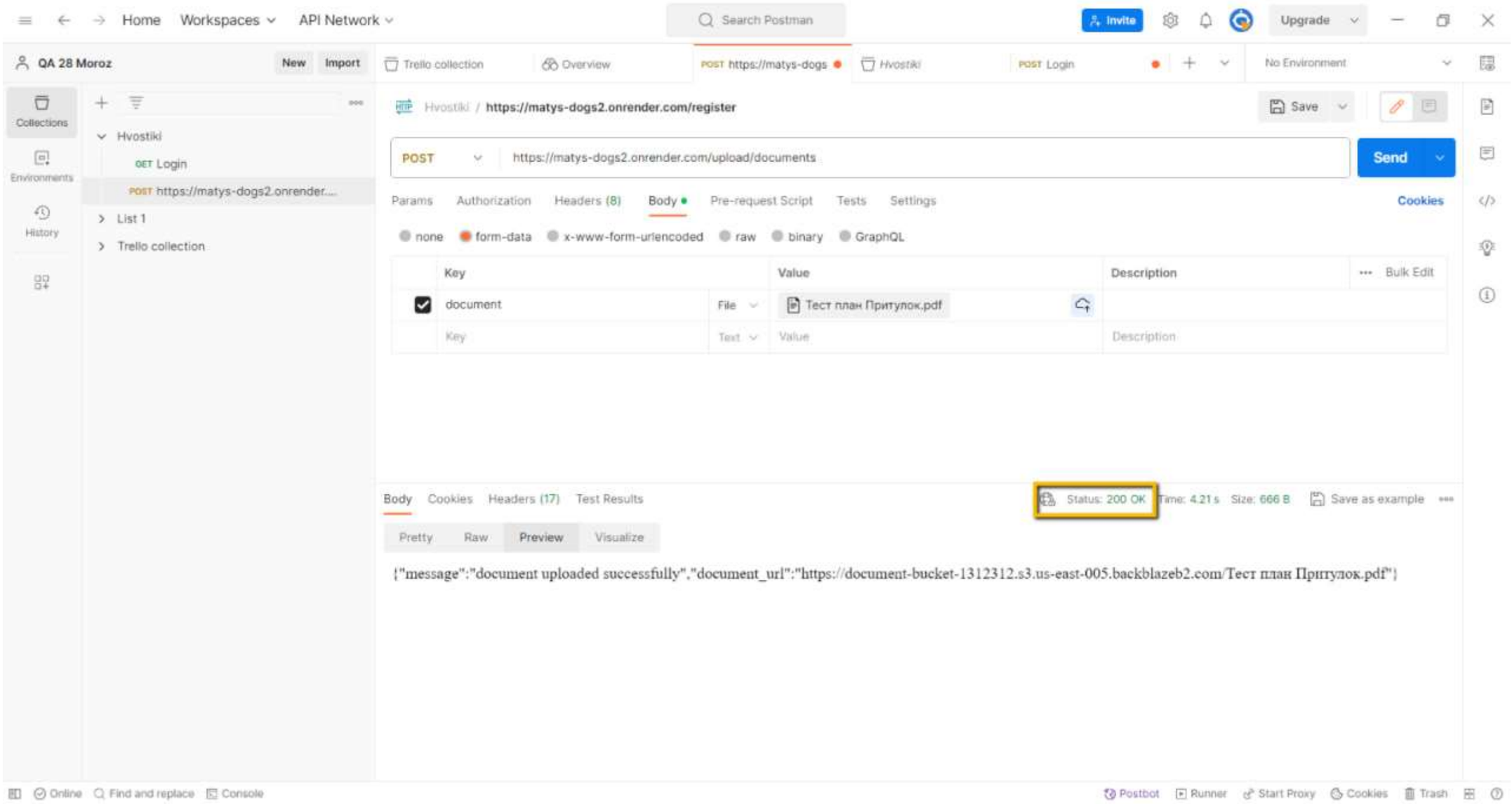
Основні особливості V-моделі

Наступна модель SDLC — це **V-модель.** За цією моделлю тестування необхідно починати якомога раніше у життєвому цикл
Основна ідея V-моделі у тому, що завдання розробки та тестування є відповідними активностями однакової важливості.
Дві гілки літери V символізують це.
V-model для розробки ПЗ:

Активності в лівій частині моделі — це активності, відомі з модел
Waterfall, і вони орієнтовані на вимоги:
•**Специфікація вимог** (Requirement specification) — облік потреб користувачів.
•**Функціональна специфікація** (Functional specification) — визначення функцій, необхідних для задоволення потреб користувача.
•**Технічна специфікація** (Technical specification) — технічний дизайн функцій, зазначених у функціональній специфікації.
•**Специфікація програми** (Program specification) — докладний дизайн кожного модуля або блоку, який буде побудований з урахуванням необхідної функціональності.

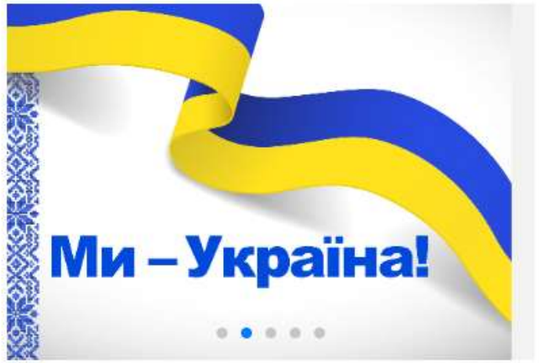


Тестування API



Розміри: Адаптивний 420 × 823 100% Без обмеження пропускної спроможності

МЕНЮ



Новини | Усі новини

Ліцеїсти відвідали майстер-клас з синхронного перекладу

Елементи Консоль Джерела Мережа Ефективність 14

```
<!DOCTYPE html>
<html lang="uk" class="js canvas canvastext no-touch rgba backgroundsize borderimage bo rac
ius boxshadow textshadow opacity cssanimations cssgradients csstransforms csstransforms3d css
ransitions fontface">
  ▶ <head> … </head>
  ▼ <body id="body"> == $0
    ▶ <div id="page-preloader" style="display: none;"> … </div>
    ▶ <div id="page" class="page"> … </div>
      <script src="/assets/65fafed5/jquery.min.js?v=1678289540"></script>
      <script src="/js/pntu/min_slider/modernizr-custom-v2.7.1.min.js?v=1680712727"></script>
      <script src="/js/pntu/min_slider/jquery-finger-v0.1.0.min.js?v=1680712727"></script>
      <script src="/js/pntu/min_slider/swiper.jquery.min.js?v=1680712727"></script>
      <script src="/assets/6e0413fe/yii.js?v=1698766748"></script>
      <script src="/assets/5d7427f8/js/bootstrap.js?v=1550073338"></script>
      <script src="/js/pntu/script.js?v=1680712727"></script>
      <script src="/js/pntu/lazyload.js?v=1680712727"></script>
    ▶ <script> … </script>
    ▶ <script> … </script>
  </body>
</html>
```

:hadow.opacity.cssanimations.cssgradients.csstransforms.csstransforms3d.csstransitions.fontface body#body

Стилі Обчислено Макет Блоки прослуховування подій Точки зупинки для DOM Властивості

Фільтр :hov .cls

```
element.style {
}

#body {                                                    style.pntu.…93206172:56
  font-size: 15px;
}

html, body, h1, input {                                    style.pntu.…93206172:51
  font-family: 'Century Gothic', Arial, sans-serif;
  font-weight: 300;
  font-style: normal;
}

html, body, fieldset, table {                              style.pntu.…93206172:47
  margin: ▶ 0;
  padding: ▶ 0;
}

body {                                                     scaffolding.less:27
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  font-size: 14px;
  line-height: 1.42857143;
  color: #333333;
```

Консоль Що нового

Технічна підтримка та розгортання (Deployment)

Після того, як продукт надходить у реліз (випускається для користування), залишається необхідність у тестуванні, так як буде відбуватися оновлення ПЗ, будуть з'являтися нові баги, які були випущені або баги, пов'язані з експлуатацією кінцевим споживачем. В такому випадку потрібне втручання відділу QA.

Висновок

У ході даної дипломної роботи були проведені масштабні дослідження та розробка інформаційної системи для підприємства на основі веб-технологій. Реалізація проекту включала наступні основні етапи та досягнення:

Аналіз вимог та розробка інформаційної системи:

Проведено поглиблений аналіз потреб бізнесу та визначено основні вимоги до інформаційної системи.

Дизайн інтерфейсу користувача:

Для полегшення взаємодії з системою створено інтуїтивно зрозумілий і зручний інтерфейс.

Розробка серверної та клієнтської частин:

Обидві частини системи успішно розроблені, що забезпечує їх високу ефективність і взаємодію на оптимальному рівні.