МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Web-орієнтована система проектування прототипів програмного забезпечення»

Здобувача групи ІТ.м-24 Артеменка Олександра Сергійовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Олександр АРТЕМЕНКО
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент, к.т.н., доцент каф. КН, Володимир НАГОРНИЙ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО

«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Артеменка Олександра Сергійовича
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи Web-орієнтована система проектування прототипів програмного забезпечення

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «11» грудня 2023 р.

3 Вхідні дані до кваліфікаційної роботи завдання на розробку Web-орієнтованої системи проектування прототипів програмного забезпечення

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, проектування системи, практична реалізація

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) актуальність роботи; мета та задачі; аналіз аналогів; результати моделювання; засоби реалізації; демонстрація розробки; висновки

6 Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Дослідження предметної області	04.09.23-19.09.23	
2	Формування мети і задач	15.09.23-19.09.23	
3	Аналіз аналогів та проблем використання	20.09.23-27.09.23	
4	Виявлення вимог до проекту	28.09.23-02.10.23	
5	Планування робіт та розробка макету	03.10.23-06.10.23	
6	Вибір засобів реалізації	07.10.23-09.10.23	
7	Проектування інформаційної системи	10.10.23-19.10.23	
8	Розробка інформаційної системи	20.10.23-09.11.23	
9	Тестування та завершення роботи	10.11.23-13.11.23	
10	Оформлення пояснювальної записки	14.11.23-01.12.23	

Магістрант _____ Олександр АРТЕМЕНКО

Керівник роботи _____ к.т.н., доц. Володимир НАГОРНИЙ

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Web-орієнтована система проектування прототипів програмного забезпечення».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 30 найменувань, додатків. Загальний обсяг роботи − 80 сторінки, у тому числі 56 сторінок основного тексту, 3 сторінки списку використаних джерел, 13 сторінок додатків.

Актуальність теми дипломної роботи обумовлена тим, що з одного боку, прототипування є важливим етапом у процесі розробки програмного забезпечення, а з іншого – web-орієнтовані системи набувають все більшої популярності завдяки своїм перевагам, таким як універсальність, доступність, сумісність з різними платформами та легкість оновлення.

Метою дипломної роботи є розробка та впровадження ефективної web-орієнтованої системи проектування прототипів програмного забезпечення, яка базується на сучасних методах, принципах та інструментах, та сприятиме оптимізації процесу проектування прототипів програмного забезпечення.

За результатами дипломної роботи web-орієнтованої системи проектування прототипів програмного забезпечення була виконана з використанням сучасних технологій та інструментів розробки. Розроблена система має широкий спектр функціональності, що дозволяє вирішувати різноманітні задачі проектування прототипів програмного забезпечення.

Ключові слова: проект, прототип, проектування, web-орієнтована система.

ЗМІСТ

ВСТУП

Актуальність теми дипломної роботи обумовлена рядом факторів, які відображають сучасні тенденції розвитку інформаційних технологій та потреби ринку програмного забезпечення. По-перше, прототипування є важливим етапом у процесі розробки програмного забезпечення, який дозволяє виявити та усунути можливі проблеми ще до початку реалізації проекту. По-друге, web-орієнтовані системи набувають все більшої популярності завдяки своїм перевагам, таким як універсальність, доступність, сумісність з різними платформами та легкість оновлення. По-третє, існуючі web-орієнтовані системи проектування прототипів мають обмежені можливості, високу вартість або складність у використанні. Розробка нової системи, яка б ураховувала потреби розробників та забезпечувала широкий спектр функціональності, може сприяти підвищенню ефективності процесу проектування програмного забезпечення.

Об'єктом дослідження дипломної роботи є процес проектування прототипів програмного забезпечення в контексті використання web-орієнтованих систем.

Предметом дослідження є методи, принципи та інструменти, що застосовуються для розробки та впровадження web-орієнтованої системи проектування прототипів програмного забезпечення.

Метою дипломної роботи є розробка та впровадження ефективної web-орієнтованої системи проектування прототипів програмного забезпечення, яка базується на сучасних методах, принципах та інструментах, та сприятиме оптимізації процесу проектування прототипів програмного забезпечення.

Для досягнення поставленої мети в роботі необхідно вирішити наступні задачі:

1. Провести аналіз предметної області, вивчити основні поняття та принципи проектування програмного забезпечення та прототипування, а також розглянути особливості web-орієнтованих систем.

2. Дослідити існуючі web-орієнтовані системи проектування прототипів, виявити їх переваги та недоліки, а також визначити потреби користувачів та вимоги до розроблюваної системи.

3. Виконати планування робіт, визначити основні етапи розробки web-орієнтованої системи проектування прототипів програмного забезпечення та розробити загальний план реалізації проекту.

4. Спроектувати архітектуру та дизайн web-орієнтованої системи проектування прототипів програмного забезпечення, враховуючи вимоги та результати аналізу предметної області.

5. Розробити програмну реалізацію web-орієнтованої системи проектування прототипів програмного забезпечення, використовуючи сучасні технології та інструменти розробки.

Задачі відповідають змістовній частині дипломної роботи, де кожна задача відображає основні аспекти роботи, які будуть розглянуті в окремих розділах та підрозділах кваліфікаційної роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз літературних джерел присвячених проблемі проектування прототипів програмного забезпечення

Для багатьох з нас прототипування є необхідним етапом створення успішного програмного забезпечення та позитивного користувацького досвіду. Прототипування забезпечує успіх, надаючи чітке представлення про вимоги до програмного забезпечення: замість опису вимог, воно візуалізує їх. У випадку правильного застосування, прототипування дозволяє нам експериментувати у безпечному форматі, який можна легко змінити без значних втрат часу чи зусиль, порівняно з перепрограмуванням програмного забезпечення. Правильно виконане прототипування дозволяє нам йти далі простого виконання вимог, дозволяючи експериментувати та шукати оптимальні рішення. Тим навпаки, при необережному прототипуванні легко може бути створено непрозору суміш ідей, загублених у різних версіях, необґрунтованих припущеннях та конкуруючих баченнях.

Визначення прототипу мало змінилось за більш ніж 90 років. Словник Вебстера 1913 року та сучасний словник класифікують прототип приблизно однаково – як модель кінцевого продукту [1]. Однак у новому визначенні є суттєва відмінність. На відміну від Вебстера словник сьогодення має невелику зміну: він уживає слово "етапи"- множина, що ілюструє ітеративний характер прототипування [2].

У даному випадку прототипування програмного забезпечення розглядається відповідно до Біла Верпланка, який стверджує: «Прототипування – це зовнішнє втілення та конкретизація дизайнерської ідеї з метою оцінювання» [3]. Такий підхід від Верпланка дозволяє досліджувати прототипи як чітке представлення програмного забезпечення, яке уможливлює команді розробників сприймати дизайн без необхідності програмувати саме програмне забезпечення.

Прототип – це будь-яка спроба втілення будь-якого аспекту вмісту програмного забезпечення. Наприклад, прототип може передавати взаємодію та навігацію з одного пункту продукту до іншого. Прототипом також може бути ієрархічна схема

інформаційного дизайну, відокремлена від зовнішнього вигляду та відчуття кінцевого програмного забезпечення [1]. Іншими аспектами прототипу є:

– поточний стан розвитку;

– вимоги;

– зміст.

Поточний стан розвитку – це контрольна точка того, що програма буде схожа на виріб, побудований лише на теперішніх знаннях команди розробки програмного забезпечення.

Вимоги можуть стосуватися бізнес вимог, технічних вимог, функціональних вимог, вимог кінцевих користувачів або будь-якої їх комбінації.

Зміст може охоплювати різні види вмісту, які творять прототип: інформаційний дизайн, інтерактивний дизайн, візуальний дизайн, редакційний вміст, брендінг продукту та системний потужність [4].

Задля стислості ми посилаємося на будь-яку взаємодію людина-комп'ютер як програмне забезпечення, незалежно від того, чи є вони продуктом або послугою, настільним програмним забезпеченням, мобільним програмним забезпеченням, web-сайтом, web-додатком або іншим інтерактивним цифровим продуктом [5].

Розробники програмного забезпечення не є першими, хто стикається з викликами винаходів та прототипування технологій. Історична перспектива допомагає нам зрозуміти природу, проблеми та переваги прототипування. Вважається доцільним коротко розглянути трьох творців прототипів минулого: Леонардо да Вінчі, Томас Едісон та Генрі Дрейфус [6]. Кожен з них зробив видатні вклади у дизайн та процес винаходу і кожен досліджував можливості своїх винаходів за допомогою прототипів.

Да Вінчі залишив за собою прототипи концепцій та ідей (у вигляді зображень), які мали б зайняти століття для втілення. Томас Едісон використовував виснажливе прототипування як двигун, що просував його винаходи. А Генрі Дрейфус використовував прототипи, щоб зробити промислові продукти більш орієнтованими на користувачів і ергономічно правильними [7]. Ці три особи ілюструють, як прототип служить одній основній меті: перенесенню ідеї з людської уяви у форму,

яку інші люди можуть легко бачити, розуміти, оцінювати, використовувати та подальше розвивати.

Малюнки Леонардо да Вінчі (1452-1519) є одними з найцікавіших прикладів використання прототипів для дослідження інновацій. Наприкінці 15-го століття да Вінчі створив детальні ескізи інженерних ідей за проханням свого покровителя, герцога Міланського. Ці паперові прототипи визволили да Вінчі від сучасних обмежень того, що було можливо побудувати. Незалежний від границь, да Вінчі став одним з найглибших та найбільш плідних винахідників в історії [8].

Томас Алва Едісон (1847–1931) – один з найбільш плідних та видатних американських винахідників. Він досліджував ідеї за допомогою екстенсивного прототипування як на папері, так і у фізичних моделях. З понад тисячі патентів, отриманих ним за життя, найвідомішими є фонограф та вдосконалення електричної лампи. Більшість робіт Едісона була спрямована на створення продуктів для масового ринку. Він працював у часи великого промислового переходу, зі захоплюючими змінами матеріалів та технологій виробництва [1]. Створення прототипу стало не тільки джерелом інновацій, як наостанок, для да Вінчі, але й засобом спілкування вимог до виробництва: які деталі були потрібні, які форми повинні були бути зроблені, якими будуть витрати на виробництво тощо [7]. Ці прототипи мали на меті поліпшити життя на масовому рівні. Інші американські винахідники часів Едісона, такі як Олександр Греєм Бел (1847-1922, винахідник телефону), Джордж Вашингтон Карвер (1864-1943, винахідник наук сільського господарства та продуктів харчування з арахісу) та Джон Уеслі Гаятт (1837-1920, винахідник целулоїду, раннього термопластика), прагнули поліпшити повсякденне життя, зменшуючи фізичну працю та впроваджуючи предмети розкоші та розваг для мас [1].

Промислові дизайнери минулого століття використовували прототипування та ітеративний дизайн для висловлення потреб продукту та кінцевого користувача. Насправді, боротьба промислових дизайнерів за визнання в світі інженерії та виробництва дуже схожа на боротьбу професіоналів галузі взаємодії людини та комп'ютера за визнання в світі розробки програмного забезпечення. Генрі Дрейфус (1904-1972) був архетипом промислових дизайнерів, зокрема, тому що залишив

суттєвий творчий доробок у цій галузі. Дрейфус використовував прототипування для поєднання перспектив принципів дизайну, орієнтованого на користувача, бізнес-інтересів та інжинірингу. Він вважав прототипування одним з найважливіших внесків промислового дизайнера у процес створення продукту, оскільки воно було необхідним для спілкування зі стейкхолдерами, зацікавленими у дизайні, та дозволяло йому оцінювати дизайн за придатністю для цільових користувачів [6].

Дрейфус підніс поняття дизайну та прототипування на новий рівень: прототип як ілюстрація продукту в процесі використання, розташованого в певному контексті. У багатьох аспектах його розуміння функції прототипу є початком нашого сучасного уявлення про прототип. Читаючи його роботу про процес прототипування, вперше опубліковану в 1955 році, можна відчути справжній сучасний процес клієнт орієнованого дизайну, оскільки цей процес пропагує такі самі принципи проектування, які ми застосовуємо до наших прототипів, щоб зробити програмне забезпечення кращим: участь представників різних дисциплін у дизайні та участь користувачів у оцінці.

Історичні потреби та мотиви прототипування – інновації (да Вінчі), удосконалення ідей та вимоги (Едісон) та спілкування зі зацікавленими сторонами та оцінка (Дрейфус) все ще актуальні для сучасного програмного забезпечення, але двигуни стають більш витонченими та складними [6]. Прототипи програмного забезпечення створюються під впливом різних вимог, які не завжди доповнюють одне одного, якщо не суперечать. Наведемо лише чотири приклади серед багатьох інших.

По-перше, мотивація прибутку відтворює успішні продукти на ринку вчасно.

По-друге, існує тиск, щоб залишатися перед конкуренцією.

По-третє, з сучасним дизайном та розробкою програмного забезпечення пов'язані ризики, такі як невідомість, чи буде продукт бажаним для користувачів, доки він не надійде на ринок.

Нарешті, кінцевий користувач може не хотіти того, що є прибутковим або життєздатним для виробництва. Таким чином, прототипи повинні бути більшими, ніж просто матеріальні відображення ідей; вони повинні відображати вимоги та компроміси, необхідні для досягнення найкращого балансу між ними.

Наші попередники у прототипуванні в основному зосереджувалися на випуску виготовлених продуктів для масового ринку. Хоча багато уроків, таких як відгуки користувачів та ітеративний дизайн, були вивчені з цих історичних зусиль, прототипізація програмного забезпечення є унікальною діяльністю. У процесі створення програмного забезпечення прототипи використовуються для різних цілей, від розробки внутрішньої загальної візії до розробки продуктів для споживання цільовими користувачами.

Прототип може бути використаний для тестування великих та маленьких ідей. Прототип може варіюватися від низької до високої деталізації, точно представляючи частково або повністю кінцевий інтерфейс користувача. Прототип може представляти прості початки надзвичайно складної системи або відтворювати складну та хитру частину взаємодії, для якої відгук користувача є життєво важливим. Розробка прототипу, який можна віддати на руки користувачам, допоможе вам перевірити, розвинути та оновлювати ваші ідеї, а також часто допоможе вам розкрити нові можливості [9].

Прототипування може допомогти відповісти на такі питання:

– чи буде дизайн працювати належним чином?

– чи може дизайн бути виготовлений економічно?

– як користувачі відреагують на дизайн?

– який підхід можна використати для переходу від концепції до продукту?

– як прототипування може підтримувати специфікацію дизайну продукту?

Прототипування є корисним інструментом для вирішення проблем, а також постановки та відповіді на питання. Як тільки концепція дизайну виявлена, ви можете створити прототип для ілюстрації її. Прототипування потім може допомогти розвинути та удосконалити дизайн. Це не замінює розмірковування над вашою концепцією дизайну. Але іноді метою є розробка нових концепцій та просто дослідження.

Генеративне дослідження спрямоване на розуміння того, що створити, а не тільки того, як вдосконалити те, що вже існує. Ви можете використовувати прототипи для дослідження ідей, не знаючи, куди вони можуть вести, перевіряючи можливості

на шляху. Прототипування ідей генерує нові напрямки для дослідження та нове розуміння того, як люди взаємодіють з програмним забезпеченням [10].

Участь у дизайні часто використовується у генеративних дослідженнях для залучення більшої кількості точок зору, особливо точки зору користувача, для розробки ідей [11]. Прикладом прототипування для генеративних досліджень є «blank model prototyping». Прототипи порожньої моделі створюються шляхом інтерактивного дослідження різних розмірів та форм ручного пристрою та розміщення кнопок на його зовнішній стороні. Дизайнер та користувачі працюють з блоками із пінопласту, кнопками з подвійним скотчем та іншими виробничими матеріалами, щоб досліджувати форму апаратного забезпечення та розташування кнопок. Під час дослідження учасники обговорюють, що вони роблять і чому. Кожен учасник працює зі своїми матеріалами, і можуть з'явитися декілька різних ідей. Блоки з пінопласту та кнопки були простими формами, реалістично пропорційними, але зовсім не опрацьованими або можливими для сплутування з реальним продуктом.

Метою зазначеного наочного дослідження є концентрація на фізичному дизайні та маніпуляції, а також знання, яке можна отримати з вправи для відповіді на дослідницькі питання та нічого більше [12].

Визначення масштабу та необхідних ресурсів є одними з найскладніших етапів у плануванні проектів програмного забезпечення. Без матеріального представлення продукту чи функціональних можливостей планування проводиться в основному у абстракції, на основі попереднього досвіду, навичок команди та зацікавленості в ринку, існуючих продуктах та цільових технологіях. Зазвичай плани продуктів та функціональних можливостей програмного забезпечення базуються на складному різноманітті протирічливої інформації, яка включає тенденції ринку, вимоги галузі, запити клієнтів щодо функцій, інновації та успіхи конкурентів, попередні плани проектів, попередній досвід членів команди, а також кошти в банку. Існує практика, де інженерні зусилля оцінювалися ще до того, як з’являвся інтерфейс. Ранній швидкий прототип може значно спростити сприйняття запланованих інновацій, функціональності та концепцій продукту. Це, в свою чергу, дозволяє точніше визначити обсяг робіт, що дозволяє забезпечити кращу оцінку, планування та

бюджетування. Невелика передбачувана робота може поліпшити планування проекту. Ранній прототип, який включає макети екранів, потоки взаємодії, переходи станів та вимоги до продуктивності, може надати необхідну інформацію для оцінки з вищим рівнем упевненості [13].

У даному розділі відзначені мотиви, що є критичними для успішного проектування в галузі інформаційних технологій. Вони формують основу для ефективного прототипування, визначаючи ключову роль структурованих дій у досягненні вдалих результатів у розробці програмного продукту. Процес прототипування, коріння якого входить у сферу винаходів, а також захисту інтелектуальної власності, має значущий історичний відтінок. Незважаючи на те, що визначення прототипування залишається практично незмінним протягом понад 90 років, сам процес став більш тонким і складним. Тепер прототипування виступає як важлива складова, спрямована не лише на концептуалізацію та дизайн продуктів, але й на визначення стратегії, уточнення специфікацій та розробку планування продукту в галузі інформаційних технологій.

1.2 Дослідження та аналіз існуючих методів створення прототипів програмного забезпечення

Продовжуючи розгляд теми прототипування, наступне важливе питання таке: як створюється прототипи? Дана робота є спробою зробити процес проектування прототипів не просто зрозумілим та керованим, а також виконуваним - будь-ким, хто бажає вдосконалити дизайн продукту та дізнатися більше про потреби користувачів.

За результатами дослідження характеристик майбутнього програмного забезпечення, визначається тип прототипу, який вам потрібен. Кожен метод створення прототипів, такий як створення прототипів на папері або за допомогою сюжетних дошок, має специфічні риси які найкраще підходять для певних типів прототипів [14].

У літературі виокремлюють наступні найпоширеніші методи створення прототипів [1]:

- сортування карток («Card sorting»);
- прототипування за допомогою каркасів («Wireframe prototyping»);
- прототипування з використанням сюжетних досок («Storyboard prototyping»);
- створення прототипів на папері («Paper prototyping»);
- цифрове прототипування («Digital prototyping»);
- прототипування з використанням порожніх моделей («Blank model prototyping»);
- відео-прототипування («Video prototyping»);
- прототипування за допомогою методу «Wizard-of-Oz prototyping»;
- кодоване прототипування (включаючи скрипти та HTML)

Прототипування з сортуванням карток – це абстрактний, інтерактивний, учасницький метод, проведений на початкових стадіях процесу створення програмного забезпечення [15]. Розробники програмного забезпечення використовують сортування карток для визначення найкращої інформаційної та навігаційної структури або для дослідження та перевірки термінології. Аудиторія кінцевого прототипу – це внутрішні члени дизайнерської команди, що дозволяє досягти спільного розуміння питань інформаційного проектування. Фактичні сесії сортування карток проводяться з передбаченими користувачами. Сесія сортування карток зазвичай починається зі стопки термінів або ідей на картках індексу. На картках можуть бути терміни, які призначені для випадаючих меню, навігаційних термінів веб-сайту або будь-якої групи інформації, яка повинна бути поміщена в ієрархію. Користувачі потім намагаються згрупувати терміни таким чином, щоб це було зрозуміло. Коли з даних користувачів виявляється спільна тенденція або закономірність, проаналізовані та синтезовані результати показують, як більшість учасників уявляє собі інформацію та термінологію, надану їм.

Прототипування за допомогою каркасів («Wireframe prototyping») – це наративний прототип, який зазвичай створюється на початку процесу дизайну.

Нарація зазвичай виводиться з випадку використання або сценарію, часто того самого сценарію, що використовується в розкадровці [16]. Цей прототип показує високорівневі наброски, візуалізуючи концептуальні припущення про структуру продукту і загальні принципи взаємодії. Основна мета цього методу – забезпечити згоду дизайнерської команди щодо базових концепцій та напрямків дизайну, які керують концептуальним проектуванням, а також детальнішими рішеннями щодо дизайну. Каркас починається з необробленого малюнка того, як може виглядати програмне забезпечення. Це може бути все – від примітивного намальованого інтерфейсу на серветці до більш ретельно пропрацьованих схем екранів програмного забезпечення за допомогою графічного інструменту. Каркаси зазвичай не мають зв'язаного з ними візуального дизайну, оскільки вони призначені для використання на ранній стадії процесу дизайну для визначення потоку взаємодії та навігаційної моделі. Згодом, коли сформується консенсус навколо концепції каркасу, створюються детальні дизайни на його основі. Каркаси як зазвичай перестають бути центральним акцентом, коли концептуальний дизайн завершено і може початися деталізоване прототипування (паперове або цифрове) [17].

Прототипування з використанням сюжетних дошок («Storyboard prototyping») – це наративний прототип, який зазвичай створюється на ранніх стадіях процесу створення програмного забезпечення для висловлення бізнес- та маркетингових вимог у вигляді сценарію використання чи історії. Ці історії розповідають про дії користувача, необхідні для створення прототипу розкадровки виконання задач відповідно до вимог ринку, клієнтів та користувачів. Ці вимоги збираються, аналізуються та синтезуються в сценарій перед початком процесу розкадровки. Оскільки вимоги спричиняють процес розкадровки, вони забезпечують раннє розуміння того, що користувачі, програмне забезпечення та система призначені для взаємодії один з одним. Основна мета сюжетної дошки полягає в узгодженні думки членів команди створення програмного забезпечення щодо цілей та поведінки продукту чи послуги без докладного оформлення екранного дизайну. Призначена аудиторія – це, перш за все, внутрішні члени дизайнерської команди, які формують спільне розуміння перед проектуванням дизайну. Додатковою аудиторією можуть

бути ключові зовнішні зацікавлені сторони, які підтверджують напрямок дій команди. Розкадровка починається з послідовної сценарної наративної історії, наприклад, день з життя користувача з очікуваним продуктом чи послугою. У міру того, як ідеї дозрівають, ілюстрації починають вставлятись у наратив розкадровки, роблячи його все більш і більш візуальним, відбувається поступова ітерація [1].

Прототип на папері («Paper prototyping») – це інтерактивний прототип, який складається з паперового макета користувацького інтерфейсу. Інтерфейс зазвичай повністю функціональний, навіть якщо все функціональність змодельована на папері. Прототипи на папері дозволяють перевірити дизайн з реальними користувачами. Перша призначена аудиторія – це потенційні користувачі, які тестують дизайн, та, в другу чергу, розробники програмного забезпечення, які використовують прототип

як засіб для повідомлення про заключні ітерації дизайну. Створення прототипу на папері можна зробити вручну або роздрукувати дизайни користувацького інтерфейсу.

Користувач може потім «керувати» інтерфейсом за допомогою ручного або голосового введення. Важливо використовувати паперові прототипи, допоки більшість проблем не будуть вирішені, щоб під час розробки програмного забезпечення забезпечити лише маленькі удосконалення [18].

Цифрове прототипування («Digital prototyping») майже є цифровою версією паперового прототипу. Однак, цифрові прототипи можуть варіюватися від серії низько-деталізовані, наративних екранів з можливістю послідовного переходу для швидкої візуалізації концепції дизайну до високодеталізованого інтерактивного зображення розвиненого дизайну, яке може бути використане як специфікація користувацького інтерфейсу [7]. Цифрове прототипування має ті самі цілі, що і прототипування на папері, тобто обидва можуть бути використані для:

- розуміння процесу виконання завдань та контексту використання;
- перевірки припущень у сценаріях, бізнес-вимогах та профілях користувачів;
- допомоги у формуванні чи підтвердженні послідовності виконання завдань та напрямку дизайну інтеракцій;

– допомоги у перетворенні прототипів з ранніх грубих набросків на наступний рівень деталізації;
– допомоги у розробці чи перевірці візуального напрямку дизайну.

Цифровий прототип подібний до прототипа на папері в тому, що обидва актуальні протягом одного й того ж етапу дизайну. Цифровий прототип зазвичай більше наративний, ніж інтерактивний, так як в цифровому прототипі є деякі прогалини, які не можуть бути вирішені на папері. Однак, оскільки він, перш за все, використовується для швидкого дослідження варіантів дизайну, то часто представляє собою послідовний чи слайд-шоу прототип, якщо не використовується для тестування ергономічності.

Зазвичай для створення цифрового прототипу використовуються програмні засоби, які можуть швидко створити цифровий прототип, такі як Photoshop або ті, які зазвичай використовуються для забезпечення процесу роботи в офісі, як Excel, Word та PowerPoint [19]. Ці інструменти можуть використовуватися для імітації мінімального взаємодії програмного забезпечення, але загалом не призначені для створення прототипів з повною взаємодією. Як і паперові прототипи, з їх допомогою можна працювати з моделями мислення на додаток до візуального дизайну та перевірки вимог з користувачами. Головна перевага цифрового прототипування над кодованим прототипуванням полягає в тому, що людина без технічних знань може легко оволодіти ним і швидко створювати образи дизайнерських ідей. Подібно до прототипування на папері, цифровий прототип створюється з набору дизайну екранів.

Цифровий прототип також може складатися з дизайнів екранів, створених на папері, а потім відсканованих цифровим способом. Екрани можуть бути імпортовані в програму для створення прототипів, таку як PowerPoint або Acrobat, та розташовані в заданому порядку сценарія чи послідовності завдань. Вміст цифрового прототипа може бути легко організовано, перегруповано та змінено для швидкого вивчення й перевірки ідей дизайну. Через цю гнучкість та здатність до змін, цифрові прототипи особливо корисні під час ранніх та середніх стадій дизайну, які загалом мають дослідний характер [3].

Прототипування з використанням порожніх моделей («Blank model prototyping») – це низько-деталізовані прототипи, які швидко створюються учасниками досліджень користувачів за допомогою доступних матеріалів мистецтва та ремесел, щоб представити свої уявлення про те, яким могло би бути призначене для апаратного / програмного дизайну [20]. Цей метод використовується в ранніх стадіях проектування продукту для виявлення користувацьких уявлень та моделей мислення стосовно форм-факторів апаратного забезпечення та елементів керування взаємодією у поєднанні з програмним користувацьким інтерфейсом. Дослідження порожніх моделей користувачів проводиться як індивідуальна сесія з модератором, в якій порожні моделі будуються на основі передбачуваного сценарію завдань або активності та напрямку технологічного продукту. Це ефективний спосіб отримати ранні уявлення користувачів про форм-фактор і функціонування потенційного дизайнерського артефакта, наприклад, пристрій дистанційного керування, який використовується з інтерактивним ТБ-сервісом. Перш за все, порожні моделі призначені для аналізу отриманих схожостей та відмінностей артефактів з подібного дослідження та включення результатів у наступні зусилля з дизайну. Цільовою аудиторією для дослідження порожньої моделі є перш за все внутрішні члени дизайнерської команди, щоб у всіх було спільне розуміння поведінки користувачів та моделей мислення, а також їхні уявлення про те, як керувати пристроями перед початком роботи з дизайном. Сесія порожньої моделі залежить від чітко сформульованого сценарію та концептуального напрямку дизайну, наприклад, кишеньковий пристрій, який може містити велику кількість інформації та бути доступним у ситуації віддаленого використання, наприклад, збут на полі чи автоматизація на полі. Сценарій та технологічна концепція мають використовуються для визначення різних профілів користувачів для залучення учасників дослідження користувачів. Отримані порожні моделі та висловлені учасниками думки ретельно аналізуються, порівнюються та синтезують у поточну діяльність з дизайну [3].

Відео-прототипування («Video prototyping») – це наративний прототип, який зазвичай використовується на ранніх стадіях інноваційного дизайну, як частину стадії генерації ідей процесу дизайну програмного забезпечення. Відео-прототипи

дозволяють візуалізувати сюжетну доску шляхом маніпулювання відео, щоб зобразити наймасштабнішу систему, якби вона була повноцінно функціонуючою. Метою створення відео-прототипів є розробка нових ідей взаємодії без розвитку системи. Першочергова аудиторія складаються з внутрішніх членів дизайнерської команди, які створюють спільне розуміння продукту або послуги перед початком роботи над дизайном. Відео-прототип може бути просто візуальним представленням програмного продукту й те, як він працює. Але для того, щоб встановити контекст та масштаб, зазвичай враховуються користувачі, зображені у своїх відповідних ролях та взаємодіях у правильному контексті. Коли ідея для інноваційного інтерфейсу знайдена, відео-прототипи зазвичай дають шлях до більш конкретних методів створення прототипів, таких як прототипування на папері [10].

Прототипування за допомогою методу «Wizard-of-Oz prototyping» – це метод, який дозволяє команді створення програмного забезпечення перевіряти експериментальні мовленнєві (природньо-мовні) або тактильні інтерфейси [21]. Це також техніка, яку можна використовувати майже в будь-якій інтерактивній сесії тестування ергономічності, як засіб для закриття функціональності, що залежить від нарощування або неіснуючих технологій. Прототип «Wizard-of-Oz prototyping» – це вид інтерактивного прототипу, під час якого сесія перевірки ергономічності, учасник вважає, що він взаємодіє з реально функціонуючою системою, використовуючи, як вже згадувалося, традиційно тактильну взаємодію або природно-мовні методи введення з комп'ютерною системою; однак функціональність я керую себе. Член команди дизайну або розробки або більш широкомасштабна комп'ютерна система «позаду завіси» інтерпретує вхідні директиви учасника потім відправляє проектовану системну відповідь учаснику. Цей метод адаптували для використання на ранніх стадіях створення нових особливостей або продуктів для моделювання різних системних відповідей, включаючи голосовий зворотний зв'язок від системи та відображення екрана відповіді на голосовий введення. Людський «чарівник» може надавати як голосові відгуки, так і відображення екрана для прототипів послуг, таких як інтерфейс користувача дерева телефонів, система відповіді агента чи система управління в автомобілі.

Кодоване прототипування є інтерактивним і створюється за допомогою мови програмування або сценарію, таких як Java або JavaScript. Зазвичай прототип із кодом створюється в цільовій мові розробки та має безпосередньо перетворитись на кінцевий продукт із кодом. Прототип із кодом, в основному, відрізняється високоякісним інтерфейсом користувача і краще створюється пізно у процесі дизайну після завершення всіх великомасштабних змін дизайну. Прототип із кодом особливо підходить для зберігання як якісних, так і кількісних даних через тестування на забезпечення ергономічності. Прототип із кодом, як правило, розробляється на мові програмування або скриптовому коді й повинен розглядатися як наступна ітерація інших форм низькодеталізованого й середньо-деталізованого прототипування. Прототип із кодом має перевагу, що дозволяє команді програмного забезпечення продуктивно використовувати код для фактичного завершеного продукту [15].

Резюмуючи, у цьому розділі були представлені різні методи прототипування, які можуть використовуватися для досягнення найкращих результатів у вашому прототипуванні. Маючи встановлені характеристики прототипів і відповідний метод, обраний для досягнення бажаної цілі, ви готові вибрати правильний інструмент. Інструмент, який вам потрібен, не обов'язково має відображатись на метод, який ви хочете використовувати, і на щастя, більш ніж один інструмент може використовуватися для конкретного методу прототипування. Також ви можете використовувати один інструмент з різними методами для досягнення різних результатів прототипу. В кінцевому підсумку найкраще створювати прототип за допомогою методів та інструментів, які вам найбільше відомі.

1.3 Огляд доступних на ринку рішень для проектування прототипів

У сучасному світі розробка програмного забезпечення вимагає ефективних інструментів для визначення, візуалізації та тестування концепцій. Один із таких

інструментів, який традиційно сприймався як презентаційний редактор, але стає все більш популярним для створення прототипів, – Microsoft PowerPoint (рис. 1.1).

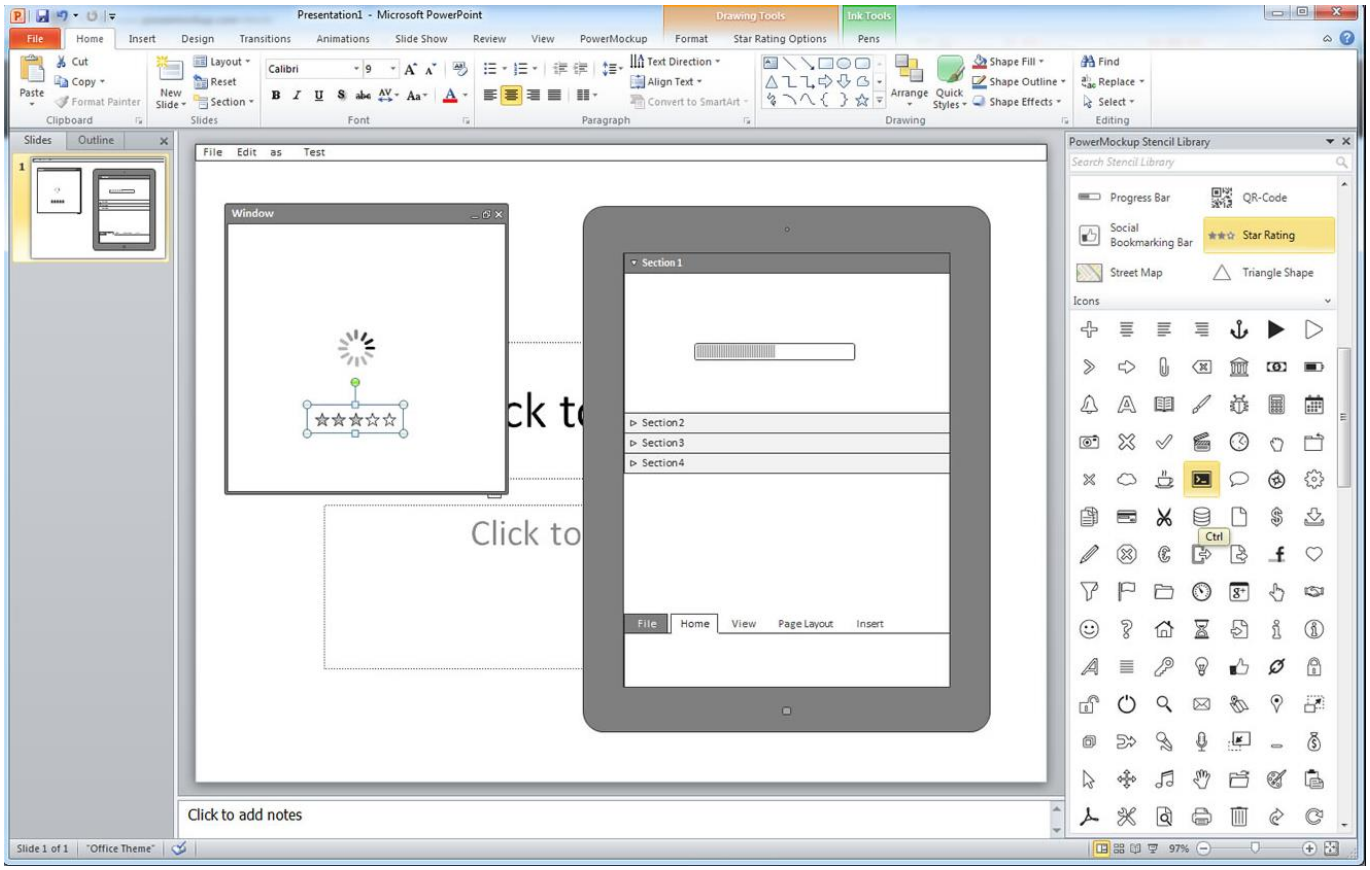

Рисунок 1.1 – Інтерфейс Microsoft PowerPoint

Джерело: побудовано автором (знімок з екрану)

PowerPoint, спочатку створений для створення слайд-презентацій, набув нових функцій, які роблять його важливим інструментом для створення прототипів програмного забезпечення. Програма пропонує широкий набір інструментів для розміщення тексту, графіки та елементів керування, дозволяючи детально моделювати. Зручний інтерфейс та простота використання роблять PowerPoint доступним для розробників будь-якого рівня.

Ще однією перевагою PowerPoint, яку доцільно розглянути, є розширена колекція шаблонів та готових елементів, які значно прискорюють процес створення прототипів. Вбудовані стилі та макети дозволяють швидко створювати привабливі та консистентні прототипи без додаткових зусиль. PowerPoint також дозволяє вставляти

інтерактивні елементи та анімації, що сприяють кращому розумінню функціоналу програми. Це особливо корисно при створенні прототипів для взаємодії з клієнтами або командою розробників [22].

Створення сценаріїв взаємодії та навігації є важливою частиною прототипування. PowerPoint дозволяє візуалізувати потоки користувача та структуру програми, полегшуючи розробникам розуміння їхньої роботи. Можливість експортувати прототипи в різні формати дозволяє легко ділитися ними серед команди. PowerPoint інтегрується з хмарними службами, забезпечуючи зручний обмін та збереження прототипів у безпеці.

PowerPoint може використовуватися для створення цілісних проектів прототипування, де кожен слайд відображає окремий варіант екрану чи функціональність. Це полегшує процес тестування та забезпечує повноту візуалізації.

Додавання анімацій до об'єктів на слайді дозволяє симулювати взаємодію користувача з програмою. Наприклад, можна використовувати анімації для показу переходів між екранами чи змін стану елементів [22].

Незважаючи на всі переваги, PowerPoint має свої обмеження в контексті прототипування. Потреба у розширенні можливостей та удосконаленні інтерфейсу для більш точного моделювання функціоналу залишається актуальною.

Наступним прикладом продукту для створення прототипів програмного забезпечення, розглянемо Microsoft Visio – це програмне забезпечення для діаграм, яке використовується для створення бізнес- та технічних діаграм, що документують та структурують складні ідеї, процеси та системи. Діаграми Visio дозволяють спілкуватися ідеями таким чином, що текст та цифри не можуть передати. Як інструмент для прототипування, Visio простий у використанні, достатньо перетягувати форми для представлення будь-чого - від настільного програмного забезпечення до веб-сайтів.

Прототипи сторібордів – Visio є ідеальним інструментом для перетворення вимог бізнесу та маркетингу на сценарії використання. Наприклад, Visio має вбудовані шаблони та шаблони для бізнес-процесів, візуалізації ідей, блок-схем та UML, які можна використовувати для створення сторібордів (рис. 1.2).

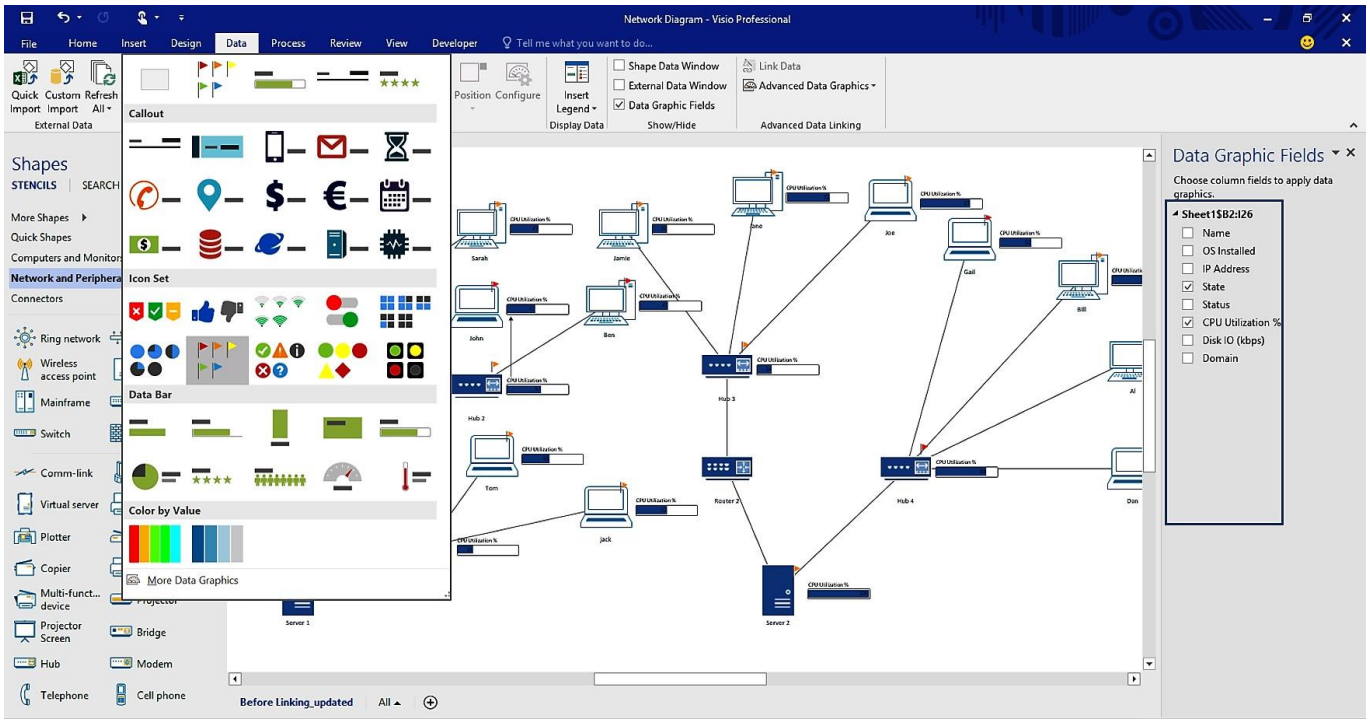

Рисунок 1.2 – Інтерфейс Microsoft Visio

Джерело: побудовано автором (знімок з екрану)

Прототипи каркасів – найпопулярніший спосіб використання Visio як інструмента прототипування. Вбудовані елементи за замовчуванням та низька крива навчання гарантують, що практично будь-хто може створювати прототипи.

Прототипування на папері – малюнки Visio можна роздрукувати для використання як паперових прототипів. Навіть якщо паперовий прототип пошкоджено або втрачено, його легко відновити. Деякі переваги Visio включають порівняно невисоку криву навчання, потужні можливості створення шаблонів, що дозволяють легко повторно використовувати бібліотеку об'єктів користувацького інтерфейсу, сітку для підтримки креслення та контролю компонування, а також включення графіки високої якості.

Visio є корисним для тестування таких аспектів дизайну. Дизайн інформації – прототипи низької якості, такі як каркаси для представлення дизайну інформації, є поширеним використанням Visio. Доступні безкоштовні сторонні шаблони, які допоможуть вам швидко побудувати каркаси.

Хоча Visio може використовуватись людьми з різними навичками, це особливо корисно для тих, хто має мало або жодних навичок малювання.

Visio надає корисні функції для швидкого та ефективного прототипування, включаючи колекцію корисних форм та шаблонів, потужні інструменти малювання, згрупування та приєднання форм, показ та приховування малюнків за допомогою шарів, додавання взаємодії за допомогою гіперпосилань, імпорт різних форматів зображень, потужні функції друку та експорту та налаштування фонового зображення.

При прототипуванні важлива різниця між Visio Standard та Visio Professional. Тільки Visio Professional має корисні шаблони, які допоможуть у прототипуванні, це варте додаткових витрат. Втім, Visio не є професійним інструментом для ілюстрації чи редагування зображень. Хоча Visio підтримує деякі базові функції малювання та редагування зображень, такі як шари та зміна розміру зображень, ви не зможете виконувати більш складні операції, такі як фільтри зображень, доступні у таких інструментах, як Photoshop. Коли потрібна більш передова настройка зображень, краще використовувати інструменти, такі як Photoshop чи Illustrator, а потім імпортувати ці зображення у Visio [23].

У Visio деякі форми не підтримують складну настройку [24]. Наприклад, існує обмеження на кількість рядків, які можна встановити у формі сітки, наданій шаблоном інтерфейсу користувача Windows. Також потрібно бути обережним при виборі кількох форм, оскільки вказівник миші може не вибрати всі форми, що призводить до несподіваних результатів. Також потрібно пам'ятати про порядок форм, розміщених на малюнку. Форми можуть бути затемнені іншими, розміщеними на малюнку пізніше або переміщеними поверх них. Також, якщо ви зміните розмір області малюнка, малюнок може неправильно друкуватися без додаткових налаштувань у налаштуваннях друку. У більшості випадків налаштування масштабування друку Visio на «підігнати до 1 аркуша вбік та 1 аркуш вниз» вирішує цю проблему [25].

Важко побачити та використати декілька документів Visio одночасно. Наприклад, копіювання та вставка кількох форм між двома документами Visio є

коштовним. Один із варіантів полягає у запуску окремих процесів Visio, кожен з яких має інший файл Visio. У Visio ви можете змінювати порядок сторінок у документі, але не можете контролювати порядок, в якому вставляється нова сторінка. На жаль, немає простого рішення. Коли це можливо, спробуйте використовувати шари або розділити малюнки на окремі документи Visio [26].

Крім функцій гіперпосилання та макросів, Visio не дуже добре підтримує взаємодію. Тому Visio найкраще використовувати як статичний прототип низької або середньої якості. Малюнки Visio можна імпортувати у Visual Studio або Dreamweaver для створення більш інтерактивних прототипів.

Наступним прикладом продукту для створення прототипів програмного забезпечення, розглянемо Adobe Acrobat (рис. 1.3). Це платформа для обміну документами, яка не тільки зберігає точне форматування документів, а також включає потужні функції перегляду та безпеки.

Завдяки точному відображенню екрана та здатностям до створення посилань, Acrobat є ідеальним інструментом для використання у цифрових інтерактивних прототипах. Acrobat є універсальним інструментом з можливістю захисту "тільки для читання", який дозволяє забезпечувати безпечність документів. Це масштабний інструмент; ви можете почати з простого використання і розвиватись разом з інструментом, коли зростають ваші потреби або рівень майстерності [27].

Серцевиною Acrobat є формат портативного документа (PDF). Adobe характеризує його як широко прийняту специфікацію, яку використовують організації у сфері стандартизації усього світу для більш безпечного, надійного електронного розповсюдження та обміну документами. PDF-файли можна створити з майже будь-якого додатка, від електронної пошти до AutoCAD, від паперу до веб-сайтів, зберігаючи вигляд, дизайн та цілісність документів. Шрифти, форматування, ілюстрації, медіа та форми відображаються точно так, як задумано. Також PDF-файли легко переробляються для доставки як друковані матеріали, веб-сторінки, електронні книги, CD-ROM та навіть прототипи [28].

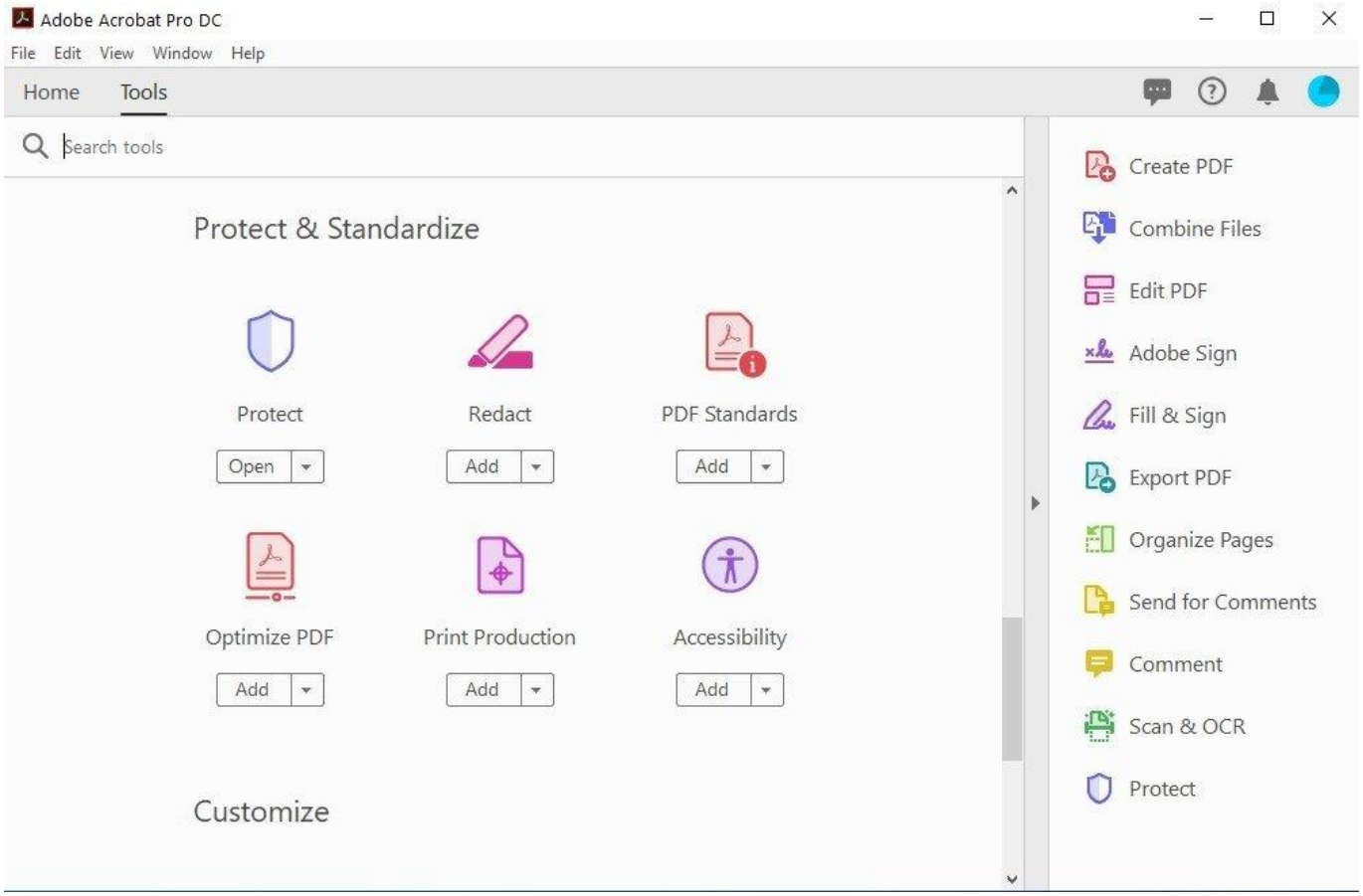

Рисунок 1.3 – Adobe Acrobat

Джерело: побудовано автором (знімок з екрану)

Хоча Acrobat не є додатком для дизайну, він включає набір утиліт, які дозволяють створювати цифрові прототипи. Використовуючи ці інструменти, ви легко можете додавати та редагувати гіперпосилання на вміст у межах та за межами документа PDF; додавати та редагувати форми (кнопки, текстові поля, прапорці, радіокнопки тощо), які імітують повну функціональність без кодування.

Тим не менше, Acrobat не є ідеальним середовищем прототипування для всіх та будь-якої ситуації. Прототипізація у Acrobat вимагає деяких хитрощів і виходу за межі призначеного використання продукту – ефективне керування посиланнями та формами вимагає плагіна. PDF-файли не реагують на інструменти навігації веб-браузера (Назад, Вперед та прокручування); і форми не є "справжніми" формами, а лише симуляціями.

Табличне порівняння розроблюваного web-додатку з аналогами конкурентів зображено в таблиці 1.1.

Таблиця 1.1 – Порівняння розроблюваного web-додатку з аналогами

Характеристика / Програмне забезпечення	**Microsoft PowerPoint**	**Visio**	**Adobe Acrobat**	**Web-орієнтована система власної розробки**
Тип програмного забезпечення	Презентаційний редактор	Діаграмний редактор	PDF редактор	Web-орієнтована система для прототипування
Підписка	Комерційна	Комерційна	Комерційна	Некомерційна
Зручність у використанні	Легкий у використанні, інтуїтивний інтерфейс	Середній рівень складності, орієнтований на діаграми	Легкий у використанні, але обмежений у редагуванні	Прискорює розробку прототипів, адаптований для легкої навігації та редагування
Функціональність	Орієнтований на створення слайдів для презентацій	Спеціалізований для створення різних типів діаграм та схем	Зорієнтований на роботу з PDF, обмежений у функціях прототипування	Розширені функції для створення і взаємодії з прототипами ПЗ
Варіативність в шаблонах та об'єктах	Обмежений в порівнянні з іншими інструментами прототипуванння	Багато шаблонів і об'єктів для створення діаграм, але обмежений у прототипуванні	Має обмежені можливості для створення інтерактивних прототипів	Представляє широкий спектр шаблонів та елементів для швидкого створення прототипів
Експорт та імпорт форматів	Підтримує різні формати експорту, але обмежений у вивантаженні прототипів	Підтримує різні формати експорту, включаючи зображення, PDF та інші	Підтримує експорт у різні формати, але обмежений у редагуванні під час імпорту	Дозволяє зручний експорт і імпорт прототипів для спільної роботи та аналізу

Джерело: побудовано автором

Аналіз порівняльної таблиці 1.1 наочно демонструє актуальність розроблюваної Web-орієнтованої системи проектування прототипів програмного забезпечення. Детальний огляд існуючих інструментів підкреслив ряд важливих функціональних недоліків – обмеженість доступу через комерційний характер підписки, недоступність та складність створення шаблонів, обмеження у розповсюдженні та спільної роботи. Все це важливо врахувати та усунути з метою покращення конкурентної позиції на ринку. На основі вищезазначеного аналізу у наступних розділах будуть сформульовані функціональні вимоги до нашої розробки, спрямовані на задоволення вимог сучасного ринку прототипування програмного забезпечення.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Програмне забезпечення в сучасному світі вимагає постійного вдосконалення та швидкого реагування на мінливість вимоги ринку. Одним із ключових етапів в розробці програмного забезпечення є проектування його прототипу – візуального представлення концепцій та функціональностей перед фактичною реалізацією. У цьому контексті, розробка web-додатку для проектування прототипів програмного забезпечення стає актуальним завданням.

Виходячи з цього, метою даної роботи є створення web-орієнтованої системи проектування прототипів програмного забезпечення, яка дозволить розробникам створювати прототипи додатків зручно і швидко, а також значно спростить процес координації роботи команди.

Обрання web-орієнтованої системи проектування обумовлено рядом переваг, які надаються цією технологією.

Універсальність. Web-додатки можуть працювати на різних операційних системах, відповідно, користувачі можуть використовувати систему проектування на Windows, Linux, MacOS або ж на мобільних платформах, без необхідності встановлення додаткових програм.

Зручність оновлення. Оскільки web-додатки завантажуються з сервера, користувачі завжди використовують останню версію системи, без потреби в ручному оновленні.

Легкість доступу. Web-додатки можуть бути доступними з будь-якого комп'ютера із підключенням до Інтернету, що дає змогу працювати з системою проектування дистанційно і співпрацювати з командою незалежно від місцезнаходження кожного учасника [4].

Отже, реалізація мети роботи полягає в створенні web-орієнтованої системи для проектування прототипів програмного забезпечення, забезпечуючи таким чином поліпшення процесу проектування та співпраці розробників програмного

забезпечення на всіх етапах її реалізації. Результати даної роботи мають практичну значимість для розробників програмного забезпечення і можуть бути використані для вирішення відповідних завдань у сфері інформаційних технологій.

Для досягнення мети роботи необхідно вирішити наступні задачі.

1. Проведення аналізу предметної області. Аналіз предметної області є важливим етапом у розробці будь-якої інноваційної системи. На даному етапі проводиться детальний огляд актуальних тенденцій, проблем і можливостей у сфері веб-орієнтованого проектування прототипів програмного забезпечення. Основна мета цього етапу – забезпечити повністю обґрунтоване розуміння особливостей предметної області для успішної розробки системи. Очікуваним результатом аналізу є визначення основних викликів, з якими стикаються фахівці у даній галузі, а також ідентифікація потенційних можливостей для вдосконалення та розвитку існуючих підходів до проектування прототипів.

2. Аналіз існуючих рішень. На даному етапі здійснюється докладний аналіз існуючих систем та програм, що використовуються для проектування прототипів. Зокрема, проводиться порівняльний аналіз функціональності, ефективності та інших характеристик, які визначають якість програмних засобів у даній галузі. Очікуваним результатом даного етапу є чітке розуміння переваг і недоліків існуючих рішень, що слугує основою для подальшого вдосконалення та розробки нового програмного продукту.

3. Визначення функціональних та нефункціональних вимог. Цей етап передбачає визначення конкретних функцій, які повинна виконувати веб-орієнтована система проектування прототипів, а також установлення нефункціональних вимог, таких як швидкодія, масштабованість та безпека. Це забезпечить точний фундамент для подальшої розробки системи. Очікуваним результатом є документ, який включає перелік всіх функціональних та нефункціональних вимог і їх пріоритети, що дозволяє забезпечити фокус на ключових аспектах системи.

4. Проведення моделювання бізнес-процесів. Моделювання бізнес-процесів є необхідним етапом для визначення і оптимізації потоків роботи у контексті веб-орієнтованого проектування прототипів. Це дозволяє точно визначити, як система

буде взаємодіяти з різними бізнес-процесами та внутрішніми чи зовнішніми стейкхолдерами. Очікуваним результатом є графічні та текстові моделі бізнес-процесів, які слугують основою для розробки системи та забезпечують її відповідність бізнес-вимогам.

5. Розробка програмного рішення. На даному етапі розробляється саме програмне рішення на основі визначених раніше вимог та бізнес-процесів. Це включає в себе вибір технологій, архітектурний дизайн та імплементацію системи. Очікуваним результатом є готовий до тестування прототип програмного продукту, який демонструє ключові функціональності системи.

6. Проведення тестування і впровадження. Останній етап включає в себе проведення різноманітних тестів для перевірки якості та надійності системи. Після успішного завершення тестів веб-орієнтована система проектування прототипів готова до впровадження. Очікуваним результатом є впровадження системи в реальне виробниче середовище, її готовність до використання та можливість взаємодії з користувачами та іншими системами.

У результаті виконання цих задач буде реалізовано web-додаток, що надасть ефективність процесу проектування прототипів програмного забезпечення та значно поліпшить досвід користувачів. Мета роботи буде досягнута, якщо в результаті впровадження web-додатку команди розробників програмного забезпечення зможуть оптимізувати процеси проектування та координації, що сприятиме досягненню кращих результатів в розробці та редакції програмного забезпечення.

Для досягнення мети роботи та якісного проектування прототипів програмного забезпечення необхідно виокремити основні функціональні вимоги до web-орієнтованого додатку. Розглянемо кожну з них більш детально.

1. Візуалізація зображення. Забезпечення ефективної роботи з прототипами програмного забезпечення має передбачати зручний інтерфейс для відображення елементів прототипу на веб-сторінці. Це полягає у створенні панелі інструментів для роботи з графічними об'єктами, яка дозволить користувачеві вибирати та кастомізувати елементи, а також взаємодіяти з ними на робочому полі. Візуалізація

зображення має бути реалізована таким чином, щоб користувачі могли легко створювати та редагувати елементи прототипу.

2. Створення зображень векторної графіки. Розробка веб-додатку має передбачати можливість створення графічних елементів за допомогою векторної графіки. Це забезпечить гнучкість при масштабуванні та редагуванні елементів без втрати якості. Векторні зображення мають відображатися коректно на різних роздільних здатностях та допоможуть користувачеві легко організувати та адаптувати прототип у відповідності до потреб програмного забезпечення.

3. Експорт у форматі SVG. Web-додаток повинен забезпечувати можливість кінцевим користувачам експортувати свої розроблені прототипи у форматі Scalable Vector Graphics (SVG), що є векторним форматом для зображень. Це дозволить зберегти детальність та якість елементів прототипу у кожному масштабі, доступному з браузера або з різних пристроїв. Експорт у форматі SVG дає можливість розробникам використовувати створені прототипи для подальшої розробки програмного забезпечення у векторних редакторах або інших графічних програмах.

4. Інструменти взаємодії. Для більш різноманітної роботи з графічними елементами, веб-додатку необхідно надати користувачам інструменти взаємодії, такі як виділення, переміщення та зміна розмірів вибраних елементів. Ці інструменти слід реалізувати таким чином, щоб користувач міг через web-інтерфейс швидко адаптуватися та вносити зміни без перешкод.

5. Інструменти малювання. Додатково до роботи з початковими елементами web-додаток має надавати функціональність ручного малювання графічних об'єктів, таких як прямі лінії, еліпси, прямокутники та інші фігури. Інструменти малювання повинні бути розвинені в плані дружньої та інтуїтивно зрозумілої взаємодії з користувачем.

6. Інструмент тексту та кольору. Для підвищення зручності та текучості роботи з прототипом, у web-додатку необхідно передбачити можливість додавати текстові елементи та змінювати колір фону або елементів для кращої візуалізації та вираження концепцій програмного забезпечення. Інструменти тексту та кольору можуть

допомогти при роботі з великою кількістю елементів або властивостей, що часто зустрічається при розробці програмного забезпечення.

7. Інструменти збереження, клонування та відміни змін. Черговим важливим аспектом функціональних вимог до web-додатку є можливість зберігати прототипи за потреби, клонувати окремі елементи для зручності кастомізації або відміняти останні зміни. Ці інструменти необхідні для забезпечення продуктивної роботи з прототипами та максимальної контролю користувача над процесом створення програмного забезпечення.

Реалізація приведених функціональних вимог допоможе створити якісний, гнучкий та зручний у використанні web-додаток для проектування прототипів програмного забезпечення, відповідний сучасному ринку та реальним потребам розробників.

Додатково у додатку А наводиться інформація щодо планування робіт задля успішної реалізації мети проекту, а саме:

– ідентифікація мети проекту методом SMART;

– опис фази розробки проекту;

– планування змісту структури робіт;

– діаграми WBS та OBS;

– аналіз та управління ризиками проекту;

– матриця відповідальності;

– календарний графік виконання проекту.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ WEB-ОРІЄНТОВАНОЇ СИСТЕМИ РОЗРОБКИ ПРОТОТИПІВ

3.1 Структурно-функціональне моделювання

Структурно-функціональне моделювання є ключовим етапом у процесі розробки програмного забезпечення, який полягає в створенні схематичного відображення функціоналу програмної системи з метою спрощення процесу реалізації [3, 6]. Моделювання за стандартом IDEF0 дозволяє створити чітку структуру задач та процесів, визначити потрібні ресурси, управління, механізми та пов'язані виводи [23].

На контекстному рівні системи представимо загальну функцію програмного забезпечення та забезпечуємо основний огляд процесу проектування прототипів програмного забезпечення (рис.3.1).

Рисунок 3.1 – Контекстна діаграма нотації IDEF0

Джерело: побудовано автором

У загальному вигляді (A-0:) проектування прототипів програмного забезпечення передбачає наступні складові:

1. Входи (I), які містять задачі та вимоги від користувачів.

2. Виходи (O), що передбачають функціональний web-додаток для проектування прототипів, зручний інтерфейс користувача, готові векторні прототипи програмного забезпечення для експорту.

3. Управління (C), яке включає відгук користувачів про хід розробки, основні стандарти веб-дизайну, результати аналізу потреб користувачів, результати тестування та відлагодження.

4. Механізми (M), які покривають розробники та проектувальники, обрані бібліотеки, фреймворки та інструменти, сервери та інфраструктура для розгортання та підтримки.

На наступному рівні моделі ми пропонуємо детальнішу декомпозицію функцій програмного забезпечення, що представляють основні процеси, пов'язані з розробкою та використанням системи проектування прототипів програмного забезпечення.

На рівні (A1:) створення керуючих елементів веб-інтерфейсу передбачає:

– I1: збір та аналіз вимог до інтерфейсу від користувачів і команди розробників. Наприклад, наявність різноманітних елементів керування, можливість швидкого доступу до основних функцій, зручна навігація тощо.

– O1: розробку інтерактивних елементів через створення кнопок, полів вводу, списків, вкладок та інших елементів, які взаємодіють з користувачем.

– C1: забезпечення відповідності розробленого інтерфейсу загальноприйнятим стандартам веб-дизайну, а також вивчення та аналіз вимог користувачів для забезпечення оптимального користувацького досвіду.

– M1: визначення та вибір необхідних бібліотек, фреймворків та інструментів для розробки керуючих елементів, а також забезпечення ефективної комунікації між розробниками для спільної роботи над створенням інтерфейсу.

На рівні (A2:) реалізація управління відображенням прототипів передбачає:

– I2: управління елементами веб-інтерфейсу;

– O2: представлення векторних прототипів програмного забезпечення;

– C2: відгук користувачів про функціональність;

– M2: розробники, бібліотеки та фреймворки для обробки векторної графіки.

A3: Розробка інструментів малювання та редагування:

– I3: запити користувачів на інструменти малювання та редагування;

– O3: набір інструментів для редакції прототипів;

– C3: вимоги до процесів малювання та редагування;

– M3: розробники, бібліотеки та фреймворки для реалізації графічних інструментів.

Використовуючи розроблені вище дані для декомпозиції процесу «Розробка інструментів малювання та редагування» представимо рисунок 3.2 як фрагмент декомпозиції.

Рисунок 3.2 – Декомпозиція процесу «Розробка інструментів малювання та редагування»

Джерело: побудовано автором

A4: Управління збереженням, клонуванням та відміною змін:

– I4: запити користувачів на збереження, клонування та відміну змін;

– O4: функціональність збереження, клонування та відміни змін;

– C4: відгук користувачів про функціональність збереження, клонування та відміни змін.

– M4: розробники, бібліотеки та фреймворки для реалізації даної функціональності:

A5: Імплементація експорту прототипів у форматі SVG:

– I5: Готові векторні прототипи програмного забезпечення;

– O5: Експортовані файли формату SVG;

– C5: Вимоги до формату експорту SVG;

– M5: Розробники, бібліотеки та фреймворки для реалізації експорту.

Підводячи підсумки, структурно-функціональна модель IDEF0 дає змогу аналізувати, проектувати та розробляти програмне забезпечення для створення прототипів програмного забезпечення з ефективно розібраною структурою та логікою функціонування. Завдяки даному стандарту можна забезпечити чітке визначення вимог до системи, спростити процес розробки та підготовки до реалізації, а також забезпечити можливість легкої адаптації системи до змінних умов і вимог.

3.2 Моделювання варіантів використання

Моделювання варіантів використання (Use Case Modeling) є ефективним інструментом для аналізу та документування вимог до системи, зокрема для веб-орієнтованих систем проектування прототипів програмного забезпечення. Варіанти використання допомагають ідентифікувати різні способи взаємодії користувачів або інших систем з системою. Розглянемо основні аспекти моделювання варіантів використання для веб-орієнтованих систем проектування прототипів програмного забезпечення через акторів взаємодії та варіанти використання.

Серед акторів взаємодії слід розрізняти:

– користувача системи проектування прототипів, де взаємодія з системою зводиться до перегляду, створення, редагування та експорту прототипів;

– розробника, яка відповідає за реалізацію функціональності системи. Взаємодія відбувається через реалізацію нових функцій, виправлення помилок, оптимізацію системи;

– адміністратора, який відповідає за адміністрування та управління системою і взаємодія якого зводиться до керування користувачами, надання дозволів, моніторинг та підтримка системи.

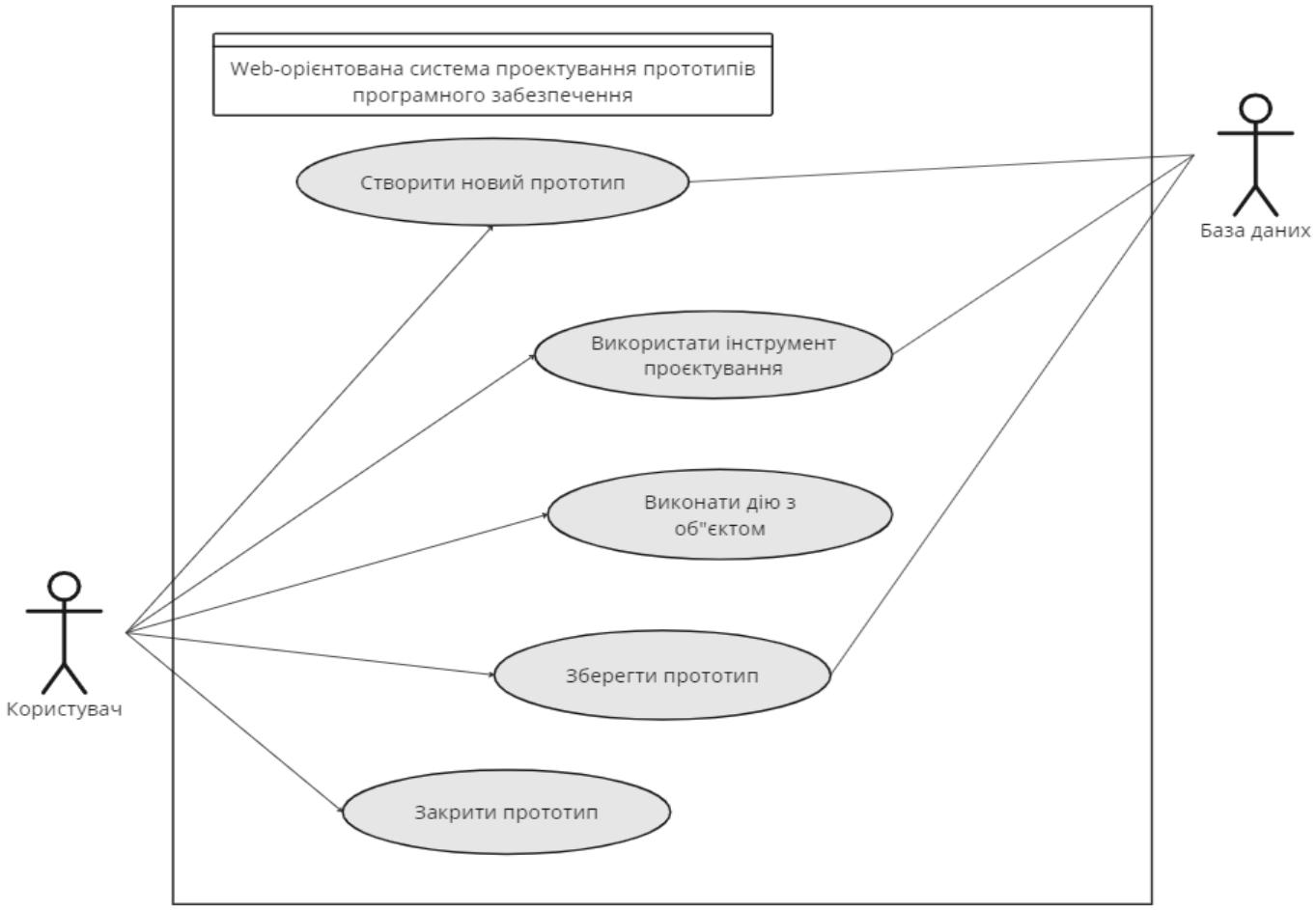

Рисунок 3.3 – Діаграма Use-case

Джерело: побудовано автором

Опис варіантів використання передбачає (рис. 3.3):

– створення нового зображення, де користувач обирає опцію "Створити новий прототип", вибирає інструменти для малювання (фігури, текст, кольори), розміщує елементи на полотні, зберігає свою роботу.

– використання інструментів (виділення, текст, фігура, колір), коли користувач взаємодіє з різними інструментами, такими як виділення областей, додавання тексту, малювання фігур та встановлення різних кольорів.

– збереження, де користувач обирає опцію "Зберегти", вибирає формат та розташування для збереження. Розробник реалізує функціонал збереження у системі.

– закриття зображення, де користувач закриває поточне зображення, система може попросити підтвердження, якщо внесені були не збережені зміни.

Для візуалізації функціональних можливостей в контексті системного аналізу, розроблена діаграма прецедентів, використовуючи мову моделювання UML (рис. 3.3). Діаграма прецедентів в мові моделювання UML надає зручний спосіб визначення функціональності системи та взаємодії її користувачів. На цій діаграмі представлені запропоновані функціональні можливості програмного рішення.

3.3 Алгоритм візуалізації векторної графіки

Візуалізація векторної графіки може бути реалізована за допомогою алгоритмів та технік, залежно від конкретного застосування і середовища використання. Ось загальний алгоритм для візуалізації векторної графіки:

1. Збір та представлення даних, таких як координати точок, типи об'єктів (лінії, криві, прямокутники тощо), кольори, розміри та інші атрибути. Це потребує створення моделі об'єктів для представлення різних типів графічних об'єктів (наприклад, класи для ліній, кривих, тексту тощо).

2. Перетворення координат через конвертацію абсолютних координат об'єктів в координати, що відображаються на екрані. Доречно використовувати матриці трансформацій для врахування масштабу, обертання та зміщення.

3. Візуалізація через відображення об'єктів. Залежно від типу об'єкта, використовуйте відповідні методи візуалізації.

Робочий процес програми починається зі створення файлу векторної графіки з масштабуванням або відкриття вже існуючого файлу, який використовується під час креслення і відтворення елементів файлу на екрані. На рисунку 3.4 наочно відображена взаємодія.

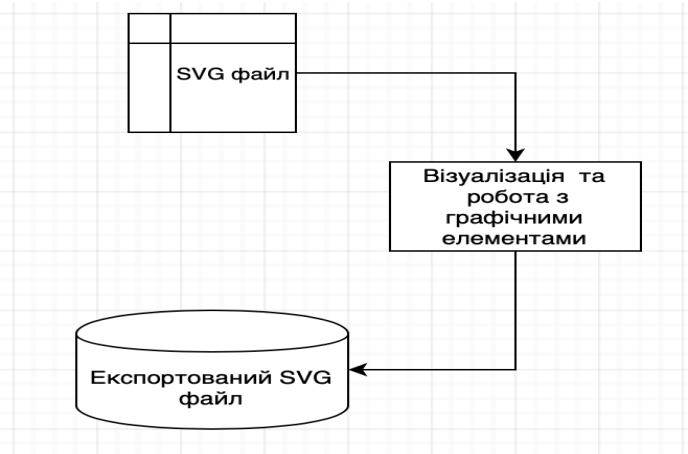

Рисунок 3.4 – Основні блоки системи

Джерело: побудовано автором

Тобто робота з графікою відбувається безпосередньо без необхідності конвертації, що є однією з можливих реалізацій процесу створення зображень. Водночас, такий підхід дозволяє зекономити системні ресурси під час роботи з зображеннями та експорту.

Наступним етапом розробки архітектури системи стало визначення класів та їх взаємодії в межах системи. Серед класів розрізняють: VectorGraphicSystem, який відповідає за управління всім системним функціоналом, GraphicObject як абстрактний клас, який представляє загальні характеристики графічного об'єкта, такі як координати, колір тощо, Line - підклас GraphicObject, представляє відрізок, Rectangle - підклас GraphicObject, представляє прямокутник, Text - підклас GraphicObject, представляє текстовий об'єкт, VectorEditor - відповідає за інтерфейс та взаємодію з користувачем для редагування та створення графічних об'єктів. Таким чином, були визначені необхідні класи та їх методи, які допоможуть в реалізації функціоналу окремих компонентів системи. Для більшої наочності, на UML діаграмі

відображені основні класи та їх наслідування, що допомагає глибше зрозуміти архітектуру програмного рішення (рис. 3.5).

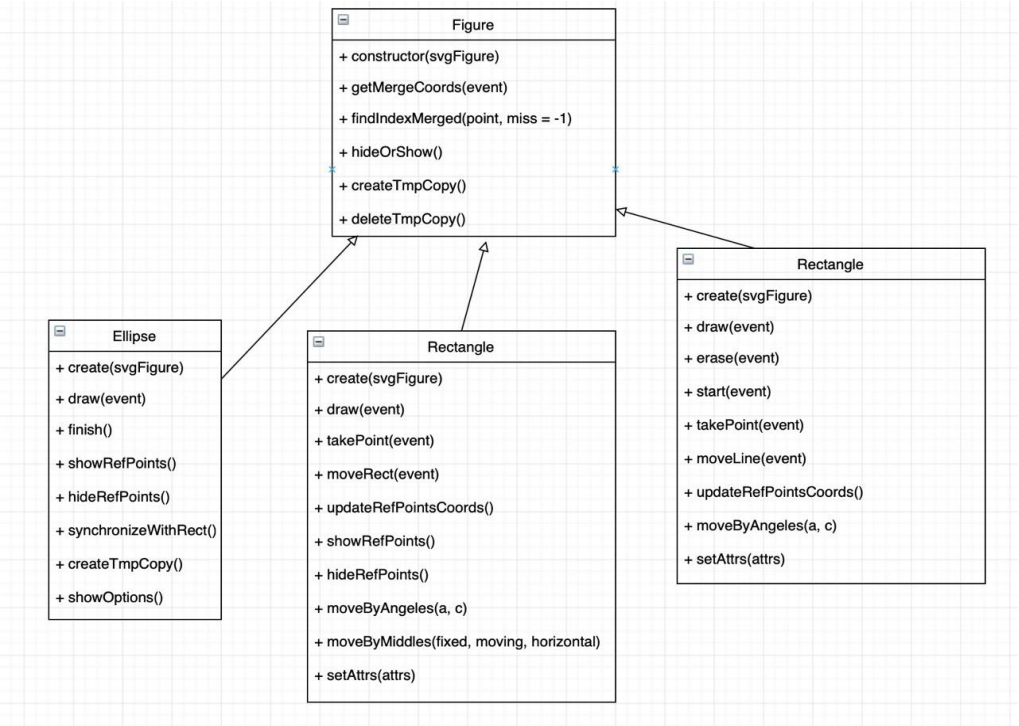

Рисунок 3.5 – UML-діаграма класів фігур

Джерело: побудовано автором

У ході виконання завдань, пов'язаних з інтеракцією з точками, були розроблені класи, спрямовані на полегшення та удосконалення процесів малювання та переміщення елементів по екрану (рис.3.6).

Функціональність програмного рішення напряму залежить від якості і, водночас, простоти реалізованих алгоритмів. Тому, в цьому етапі особлива увага приділялась деталізації та поетапній розробці алгоритмів. Мета полягала в тому, щоб передбачити можливі помилки та труднощі та, якщо можливо, уникнути їх. Це спростило подальший етап тестування

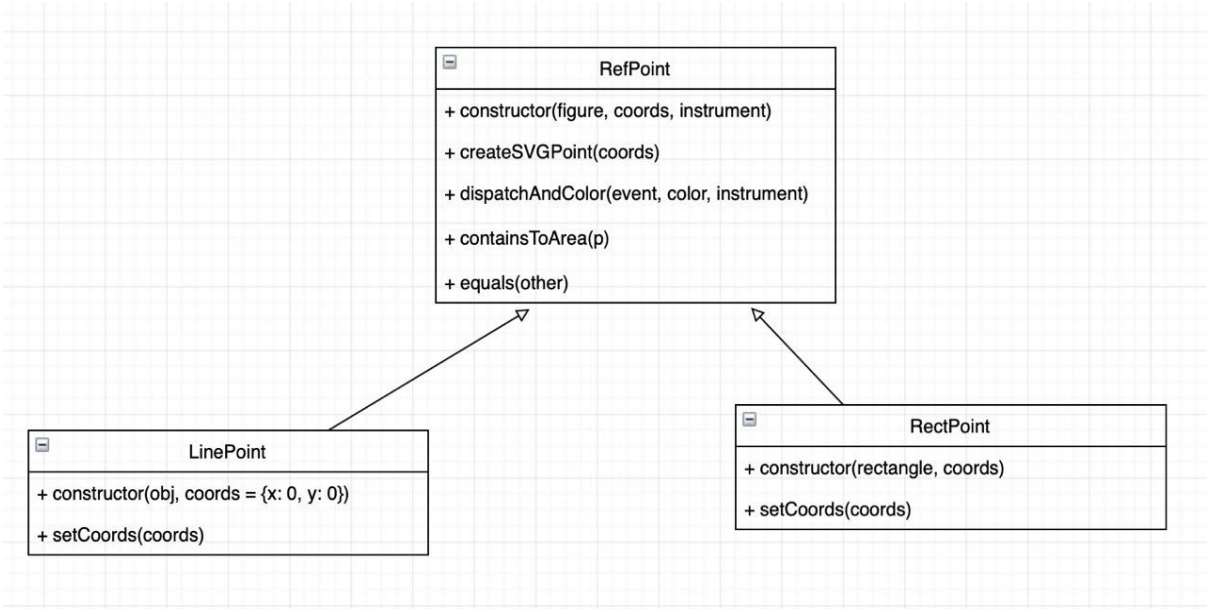

Рисунок 3.6 – Діаграма наслідування класів для роботи з точками

Джерело: побудовано автором

Початковий етап роботи включав розробку алгоритму для візуалізації векторної графіки, оскільки це є ключовим завданням та умовою успішної реалізації дипломної роботи. Підготовка до тестування включала в себе розробку алгоритму для відкриття та обробки SVG-зображень, зчитування їх інформації, визначення існуючих елементів та надання можливості взаємодії з ними (рис. 3.7).

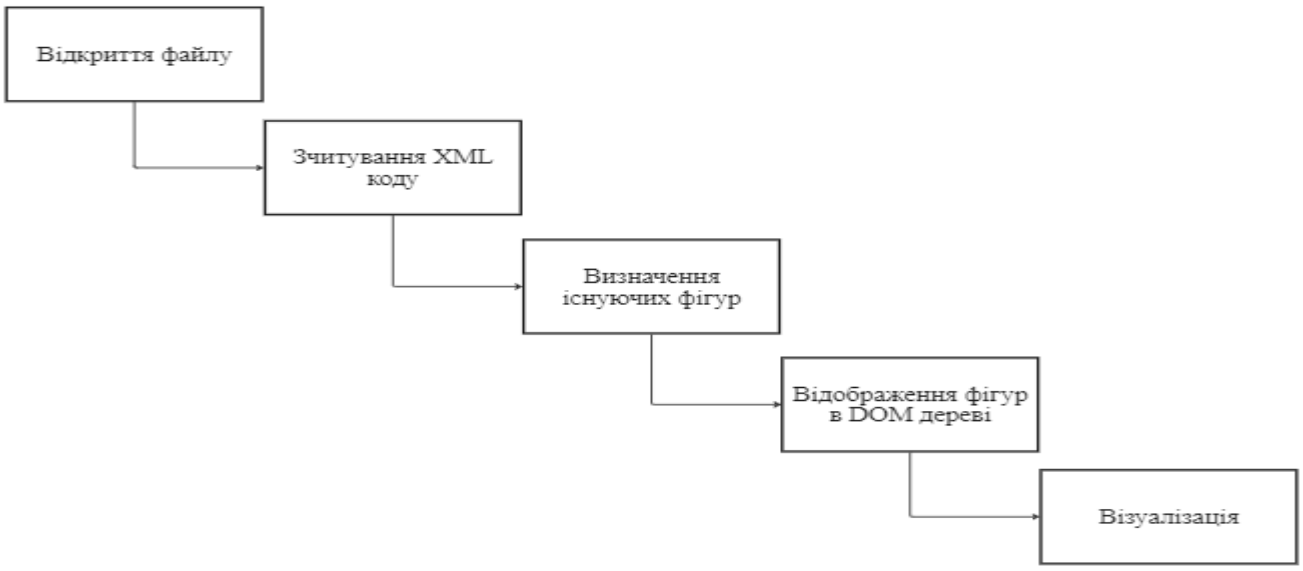

Рисунок 3.7 – Алгоритм візуалізації зображення у спрощеному вигляді

Джерело: побудовано автором

Описаний алгоритм включав наступні етапи:

– відкриття SVG-зображення: реалізація механізму для відкриття та зчитування інформації з файлів у форматі SVG;

– аналіз та визначення елементів: розробка процедури аналізу зчитаної інформації для визначення наявних елементів у зображенні;

– створення об'єктів для взаємодії: створення програмних об'єктів, які відображають елементи SVG та дозволяють взаємодіяти з ними.

– візуалізація зображення: реалізація алгоритму для відображення зчитаного та обробленого SVG-зображення на екрані.

Цей детально пропрацьований алгоритм не лише дозволяє ефективно візуалізувати векторну графіку, але й створює базу для подальших функціональних можливостей програмного рішення. Після успішного розроблення алгоритму візуалізації, наступне завдання –вдосконалення алгоритму створення нових елементів векторного зображення, а саме, малювання за допомогою графічних примітивів, характерних для формату SVG (рис. 3.8).

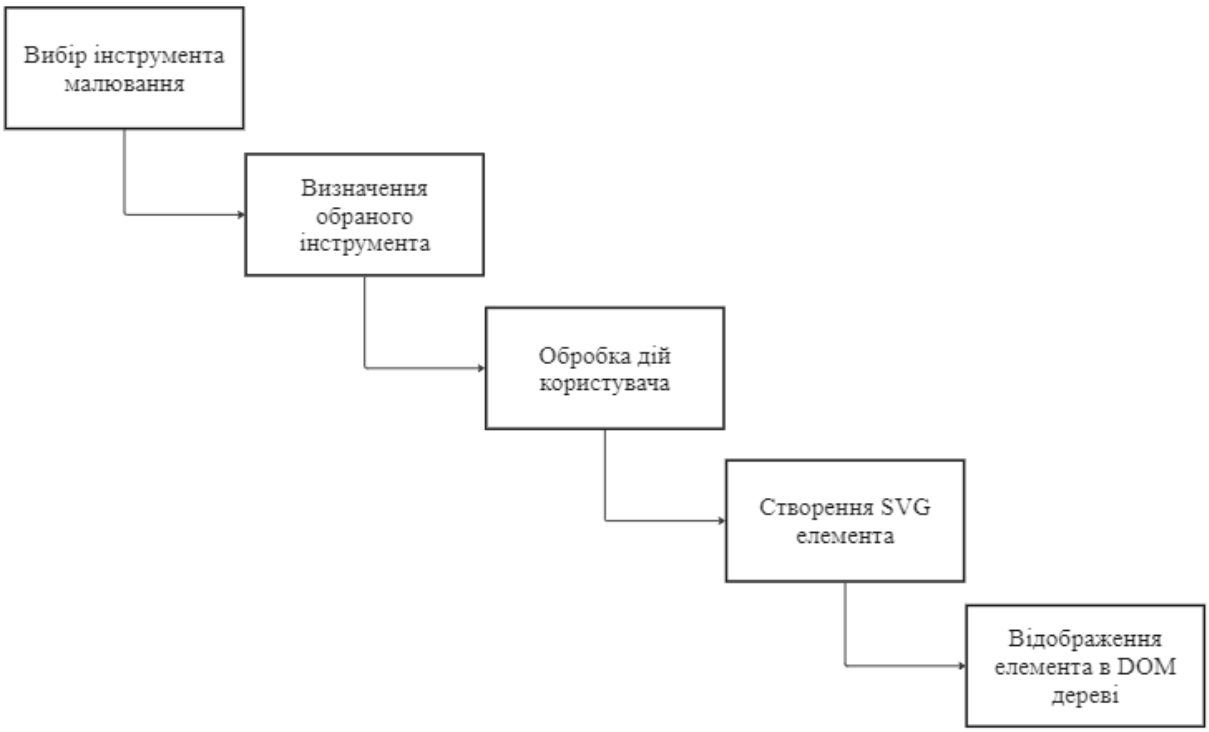

Рисунок 3.8 – Алгоритм малювання з використанням інструментів

Джерело: побудовано автором

3.4 Проектування бази даних

Проектування бази даних є важливим етапом розробки web-орієнтованої системи проектування прототипів програмного забезпечення. Відповідна структура бази даних дозволить забезпечити зберігання, пошук, відтворення та обмін даними користувачів та їх проектами з максимальною продуктивністю, а також з легкістю забезпечити розширення функціоналу системи в майбутньому.

Основними сутностями бази даних для системи проектування прототипів програмного забезпечення можуть бути сутності, пов'язані з користувачами, робочими проектами прототипів, графічними елементами та інструментами малювання та редагування. А саме:

– користувачі (users): інформація про користувачів системи та їх особисті налаштування;

– проекти (projects): інформація про створені користувачами проекти прототипів, їх структура та стадії розвитку;

– графічні елементи (graphic_elements): зберігання векторних зображень, що використовуються в проектах прототипів;

– інструменти малювання та редагування (tools): список наявних інструментів та їх налаштувань, які персоналізуються для кожного користувача.

На основі визначених сутностей можна створити таблиці бази даних, які описують структуру даних кожної сутності та взаємозв'язки між ними.

Зв'язки між таблицями (рис. 3.8):

– зв'язок "один до багатьох" між Users та Projects: кожен користувач може мати кілька проектів, а кожен проект належить одному користувачеві;

– зв'язок "один до багатьох" між Projects та Graphic_Elements: кожен проект може мати кілька графічних елементів, а кожний графічний елемент належить одному проекту;

– зв'язок "один до багатьох" між Users та Tools: кожен користувач може мати кілька інструментів малювання, а кожний інструмент належить одному користувачеві (рис. 3.9).

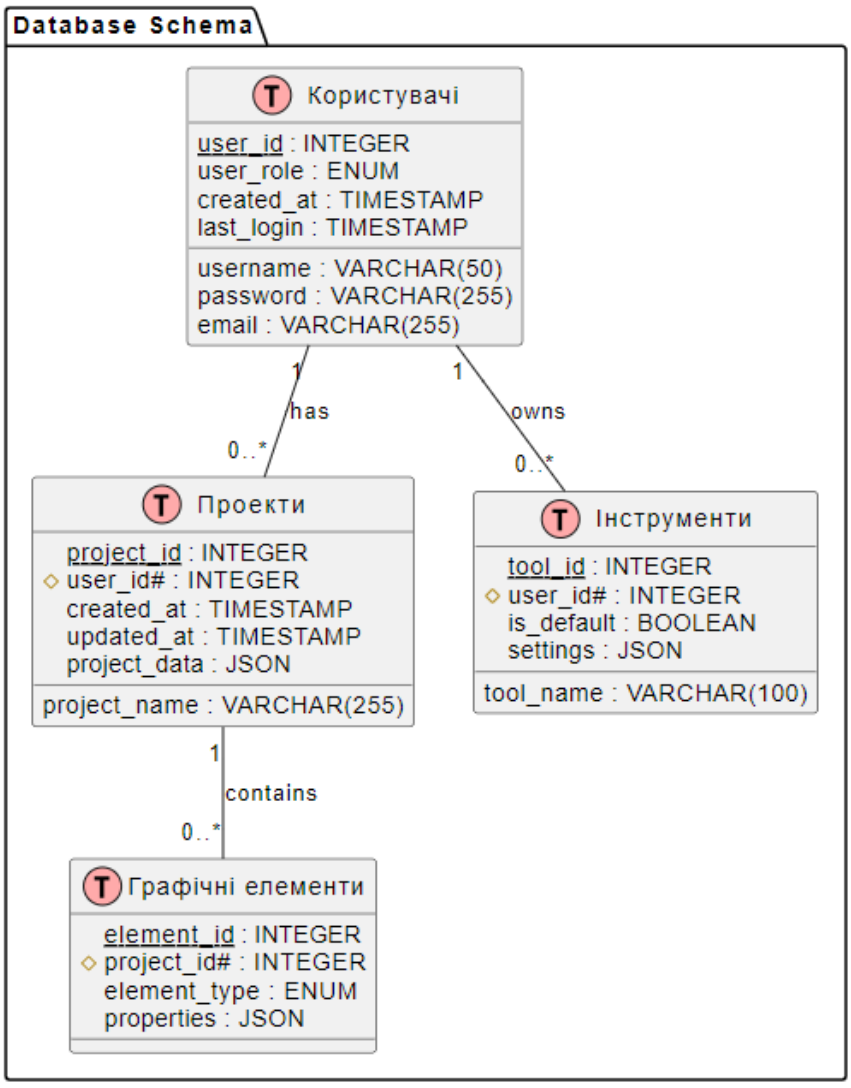

Рисунок 3.9 – Схема бази даних

Джерело: побудовано автором

Оптимізація та безпека бази даних. Для оптимізації продуктивності бази даних можна створити індекси для ключових полів, що часто використовуються при пошуку або фільтрації (наприклад, user_name, project_name тощо). Хешировані паролі можуть забезпечити безпеку даних користувачів від несанкціонованого доступу. Використання транзакцій, рядкового рівня

блокування та інших доступних опцій СКБД допоможе підтримувати консистентність даних.

Web-орієнтована система проектування прототипів програмного забезпечення потребує добре спроектованої бази даних, яка допомагає забезпечити гнучкість, продуктивність та безпеку. Проектування баз даних на основі принципів нормалізації, відокремлення сутностей та забезпечення оптимального зберігання даних є ключовим елементом успішної реалізації web-програмного забезпечення.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Розробка проекту

Проект «Web-орієнтованої системи проектування прототипів програмного забезпеченя» розроблено на основі Node.js, який є платформою для виконання JavaScript на сервері. Node.js було обрано через його широку підтримку, велику кількість доступних модулів та зручність використання для розробки веб-додатків.

Node.js – це середовище виконання JavaScript, яке працює на сервері [26]. Воно використовується для створення масштабованих мережевих додатків, особливо веб-серверів. Node.js використовує асинхронну модель подій, яка дозволяє обробляти багато запитів одночасно без необхідності блокувати виконання або використовувати багатопоточність, що робить його дуже ефективним для веб-додатків, які обробляють велику кількість короткочасних з’єднань.

Для керування пакетами в проекті використовується NPM (Node Package Manager). NPM – це менеджер пакетів для Node.js, який дозволяє розробникам встановлювати та керувати пакетами, які використовуються в їх проектах. NPM дозволяє розробникам легко встановлювати, оновлювати та видаляти пакети, а також керувати версіями пакетів та їх залежностями.

Файл «package-lock.json» в проекті відображає точну структуру встановлених модулів, що забезпечує консистентність встановлених модулів між різними інсталяціями проекту. Це важливо для забезпечення стабільності та передбачуваності роботи проекту, особливо в багатокористувацьких середовищах розробки (рис. 4.1).

У проекті використовується багато модулів, інформація про які зберігається в файлі «package-lock.json». Всі встановлені модулі зберігаються в папці «node_modules». Ці модулі включають в себе різноманітні бібліотеки та інструменти, які використовуються в проекті, включаючи бібліотеки для роботи з мережею, файловою системою, базами даних, криптографією, а також бібліотеки для тестування, збірки проекту, автоматизації задач та ін.

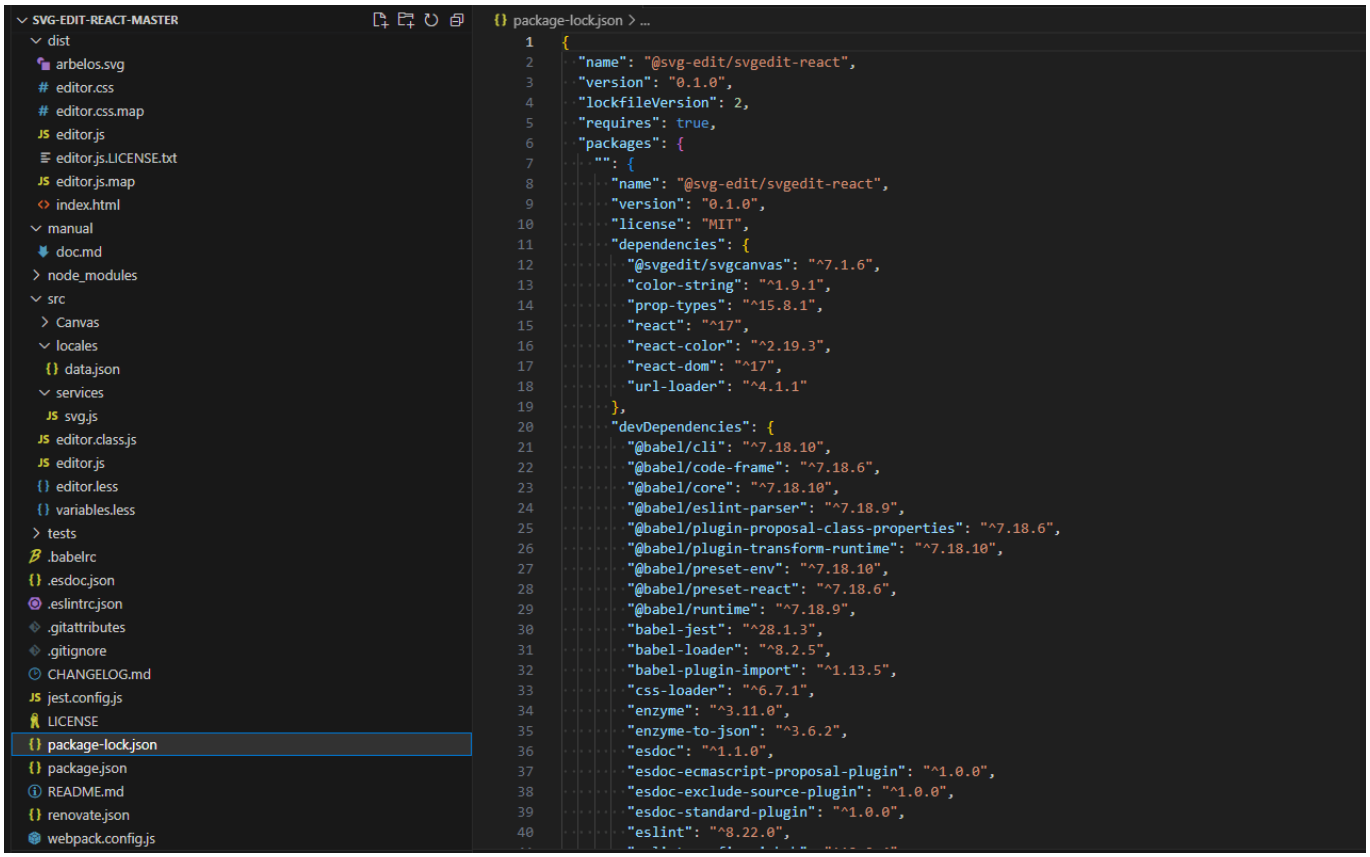

Рисунок 4.1 – Файл «package-lock.json»

Джерело: побудовано автором (знімок з екрану)

Для форматування коду використовується Prettier, налаштування якого зберігаються в файлі «settings.json» (рис. 4.2). Prettier – це інструмент для форматування коду, який підтримує багато мов програмування та допомагає забезпечити єдність стилю коду в проекті. Він автоматично форматує код відповідно до заданих налаштувань, що дозволяє розробникам зосередитися на написанні коду, а не на його форматуванні.

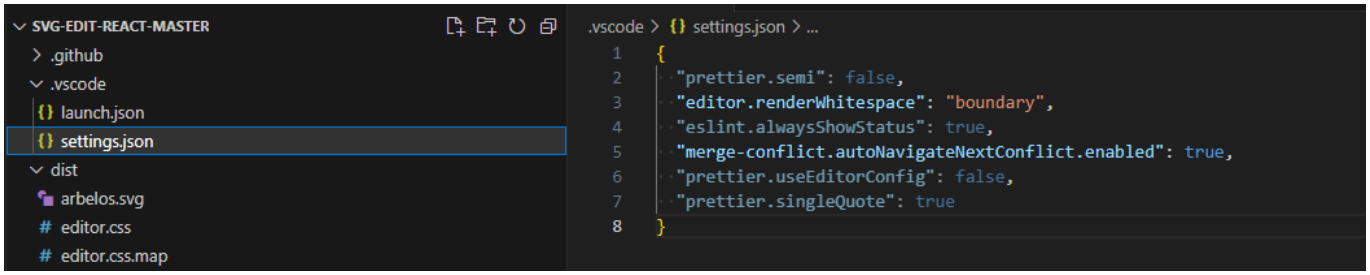

Рисунок 4.2 – Файл «settings.json»

Джерело: побудовано автором (знімок з екрану)

Веб-інтерфейс проекту розроблено за допомогою HTML, CSS та JavaScript [30]. HTML використовується для опису структури веб-сторінки, CSS – для опису її вигляду, а JavaScript – для опису її поведінки. Основна веб-сторінка проекту описана в файлі «index.html» (додаток Б), який включає в себе структуру сторінки, стилі та JavaScript-код для взаємодії з SVG-редактором.

SVG-редактор в проекті використовується для створення та редагування векторних графічних об'єктів. SVG, або Scalable Vector Graphics, – це формат векторної графіки, який дозволяє створювати графічні об'єкти, що масштабуються без втрати якості [30]. Він широко використовується в веб-дизайні та графічному дизайні.

SVG-редактор дозволяє користувачам створювати різноманітні графічні об'єкти, включаючи прямокутники, круги, еліпси, лінії, криві, текст та ін. Користувачі можуть вибирати колір, розмір, форму, стиль та інші властивості об'єктів, а також переміщати, масштабувати та повертати їх.

SVG-редактор також дозволяє користувачам редагувати SVG-код об'єктів безпосередньо, що дозволяє більш точно контролювати їх властивості та поведінку. Користувачі можуть вставляти SVG-код в редактор, редагувати його та бачити зміни в реальному часі.

SVG-редактор в проекті реалізований за допомогою JavaScript, HTML та CSS. JavaScript використовується для обробки дій користувача, створення та редагування SVG-об'єктів, взаємодії з SVG-кодом та ін. HTML використовується для створення інтерфейсу редактора, включаючи панелі інструментів, меню, діалогові вікна та ін. CSS використовується для стилізації інтерфейсу редактора.

SVG-редактор в проекті має ряд переваг. Він дозволяє користувачам створювати векторну графіку безпосередньо в браузері без необхідності встановлювати додаткове програмне забезпечення. Він простий у використанні, але при цьому має потужні можливості, що дозволяють створювати складні графічні об'єкти. Він також дозволяє користувачам редагувати SVG-код безпосередньо, що додає гнучкості та контролю над процесом створення графіки.

В цілому, SVG-редактор в проекті є важливим інструментом для створення та редагування векторної графіки. Він дозволяє користувачам легко створювати

графічні об'єкти, редагувати їх властивості та поведінку, а також редагувати SVG-код безпосередньо. Він реалізований за допомогою сучасних веб-технологій та вбудований безпосередньо в веб-сторінку проекту, що забезпечує зручність та доступність для користувачів.

В цілому, технологічний стек проекту було обрано з огляду на потреби проекту, зручність використання та широку підтримку від спільноти розробників. Він включає в себе сучасні та популярні технології, які дозволяють розробляти високоякісні веб-додатки.

Архітектура проекту базується на клієнт-серверній моделі, де серверна частина реалізована на Node.js, а клієнтська частина - на HTML, CSS та JavaScript.

Клієнт-серверна модель – це розподілена архітектура, в якій роль «клієнта» та «сервера» чітко визначена [26]. Клієнт – це програма, що відправляє запити до сервера для отримання даних або виконання дій. Сервер – це програма, що обробляє ці запити та відправляє відповіді назад до клієнта.

Серверна частина відповідає за обробку запитів від клієнта, управління даними та взаємодію з встановленими модулями. Вона організована за допомогою модульної структури Node.js, де кожен модуль виконує певну функцію. Інформація про встановлені модулі зберігається в файлі «package-lock.json», а самі модулі знаходяться в папці «node_modules» (рис. 4.3).

Модульна структура дозволяє розробникам організувати код таким чином, що кожна частина коду відповідає за певну функцію або набір пов'язаних функцій. Це допомагає зробити код більш читабельним, легким для розуміння та супроводу, а також полегшує повторне використання коду.

Клієнтська частина відповідає за відображення інтерфейсу користувача та взаємодію з користувачем. Вона реалізована за допомогою веб-технологій: HTML, CSS та JavaScript. Основна веб-сторінка описана в файлі «index.html», який включає в себе структуру сторінки, стилі та JavaScript-код для взаємодії з SVG-редактором.

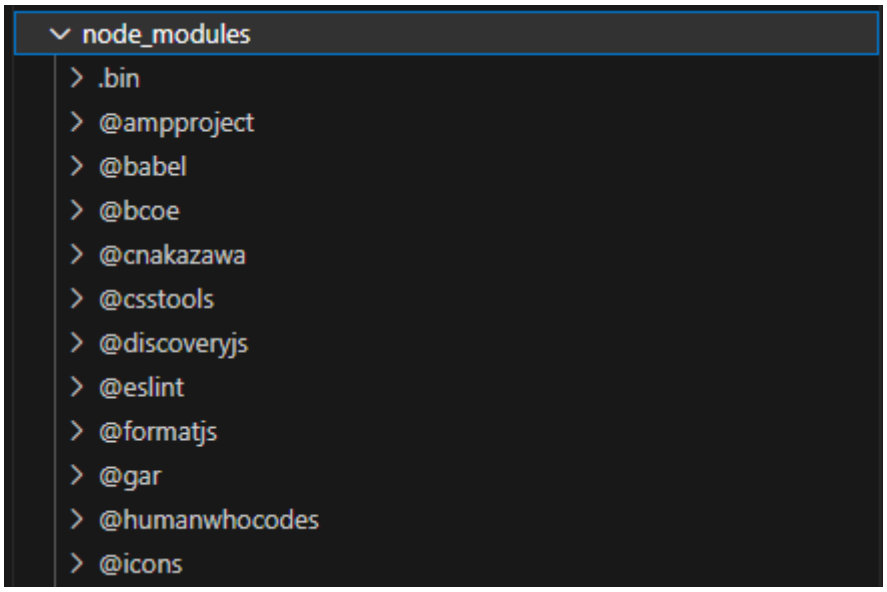

Рисунок 4.3 – Каталог «node_modules»

Джерело: побудовано автором (знімок з екрану)

HTML використовується для опису структури веб-сторінки, включаючи розміщення елементів на сторінці, їх властивості та взаємодію між ними. CSS використовується для опису вигляду елементів на сторінці, включаючи колір, розмір, шрифт, відступи, рамки та ін. JavaScript використовується для опису поведінки елементів на сторінці, включаючи реакцію на дії користувача, взаємодію з сервером, обробку даних та ін.

Взаємодія між серверною та клієнтською частинами відбувається за допомогою HTTP-запитів та відповідей [20]. Клієнт відправляє запит на сервер, сервер обробляє цей запит та відправляє відповідь назад клієнту. Це може включати отримання даних від сервера, відправку даних на сервер, виконання дій на сервері та отримання результатів цих дій.

Створення бази даних було здійснено за допомогою SQL (Structured Query Language) – мови запитів, що використовується для створення, управління та маніпулювання базами даних [20]. Були використані SQL-запити для створення таблиць, визначення полів та їх типів даних, установлення первинних та зовнішніх ключів, а також визначення зв'язків між таблицями.

Ця архітектура була обрана, оскільки вона дозволяє розділити логіку обробки даних та відображення інтерфейсу користувача, що спрощує розробку та супровід проекту. Вона також дозволяє розробникам використовувати найкращі практики та інструменти для кожної частини проекту, що підвищує якість та ефективність розробки.

Процес розробки проекту базується на принципах гнучкої методології, що дозволяє швидко адаптуватися до змін вимог та умов розробки.

Гнучка методологія – це набір принципів для розробки програмного забезпечення, який акцентує увагу на гнучкості, співпраці, відкритості до змін та постійному вдосконаленні [19]. Вона включає в себе ряд практик та методологій, таких як Scrum, Kanban, Lean Development та ін., які допомагають командам ефективно організувати роботу, реагувати на зміни та постійно вдосконалювати процес розробки.

В цілому, процес розробки організований таким чином, щоб забезпечити швидку розробку, високу якість коду та зручність його супроводу. Він включає в себе використання сучасних інструментів та практик, постійне вдосконалення процесу розробки, акцент на співпраці та відкритості до змін.

Структура файлів та папок проекту організована таким чином, щоб забезпечити легкість навігації та розуміння структури проекту. Вона відображає логіку організації коду, розподіл ролей між різними частинами проекту та процеси розробки, тестування та розгортання.

Основні файли проекту знаходяться в кореневій директорії. Це включає в себе файл «package.json» (рис. 4.4), який містить інформацію про проект, включаючи залежності, скрипти для запуску та тестування проекту, інформацію про автора та ліцензію. Файл «package-lock.json» відображає точну структуру встановлених модулів, що забезпечує консистентність встановлених модулів між різними інсталяціями проекту.

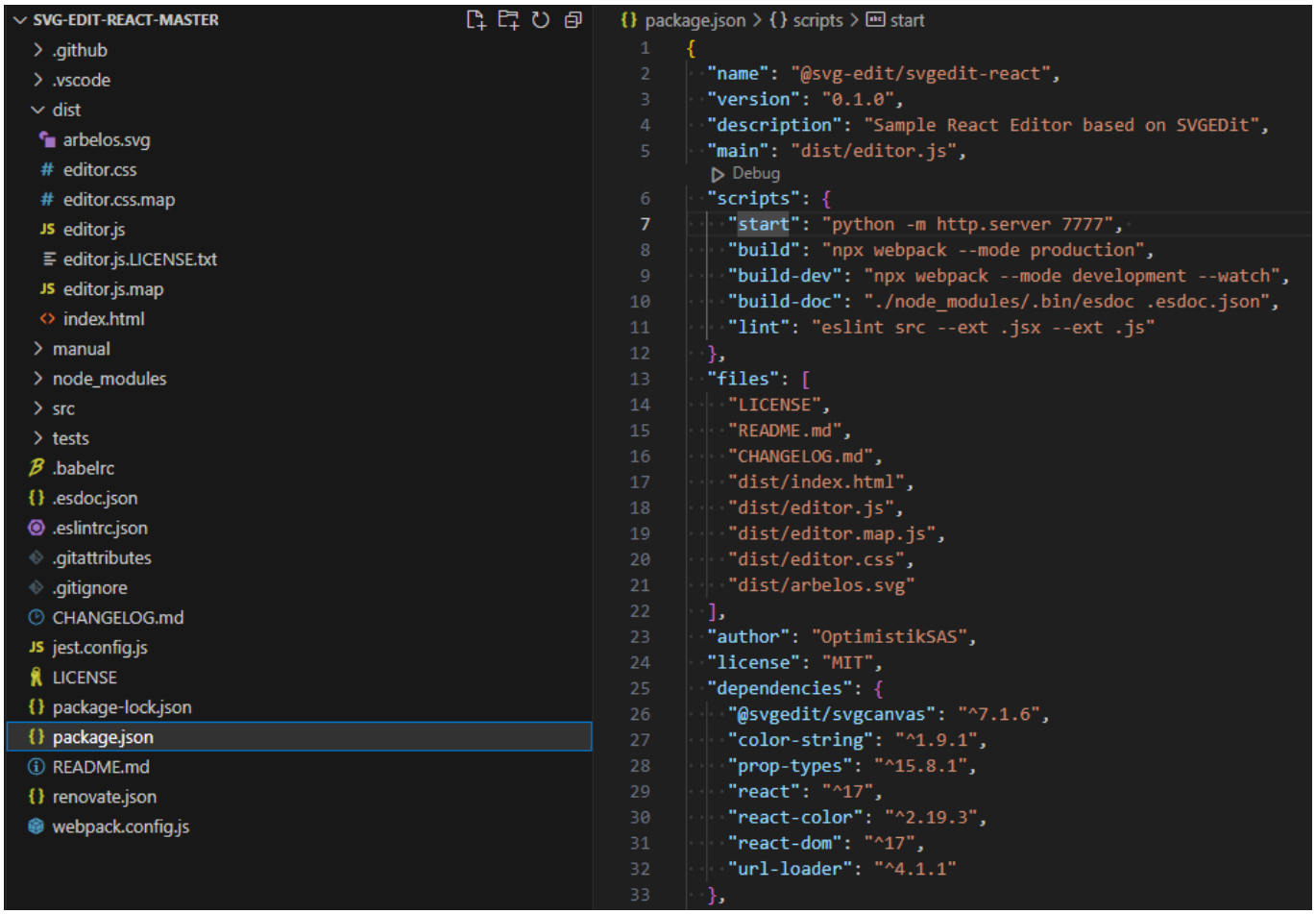

Рисунок 4.4 – Файл «package.json»

Джерело: побудовано автором (знімок з екрану)

Файл «settings.json» містить налаштування для редактора коду VS Code, включаючи налаштування для форматування коду, перевірки синтаксису, автоматичного доповнення коду та ін. Ці налаштування допомагають забезпечити єдність стилю коду в проекті та полегшують процес розробки (додаток Б).

Файл «index.html» – це основний HTML-файл проекту, який містить структуру веб-сторінки, стилі та JavaScript-код для взаємодії з SVG-редактором. Він відповідає за відображення інтерфейсу користувача та взаємодію з користувачем.

Файл «.babelrc» містить конфігурацію для Babel, який використовується для транспіляції коду JavaScript. Використовуються пресети @babel/preset-env та @babel/preset-react для транспіляції сучасного JavaScript та JSX відповідно. Також використовується плагіни @babel/plugin-transform-runtime та @babel/plugin-proposal-class-properties для оптимізації коду та підтримки властивостей класів (додаток Б).

Файл «webpack.config.js» містить конфігурацію для Webpack, який використовується для побудови проекту. У проекті використовується різні завантажувачі для обробки різних типів файлів, включаючи JavaScript, JSX, HTML, SVG, PNG, JPG, GIF, SCSS та LESS. Також використовуєте плагін mini-css-extract-plugin для витягування CSS з JavaScript. Згадана конфігурація також включає оптимізацію розбиття коду та обмеження розміру вихідних файлів (додаток Б).

Файл «jest.config.js» містить конфігурацію для Jest, який використовується для тестування коду. У проекті використовується babel-jest для транспіляції коду перед тестуванням. Також використовуєте enzyme-to-json/serializer для серіалізації знімків Enzyme (додаток Б).

Файл «Canvas.jsx» містить компонент Canvas, який є основним компонентом для відображення та редагування SVG-контенту в вашому додатку. Він використовує бібліотеку React для створення графічного інтерфейсу користувача та обробки подій, таких як оновлення SVG, закриття редактора, вибір елементів, зміна контексту та інше (додаток Б).

Всі встановлені модулі зберігаються в папці «node_modules». Ця папка автоматично створюється під час встановлення модулів за допомогою npm і не включається в систему контролю версій.

Код проекту організований за принципом модульності, де кожен модуль виконує певну функцію. Це дозволяє легко додавати, видаляти або модифікувати функціональність проекту. Модулі можуть бути організовані в папки відповідно до їх функцій, залежностей або інших критеріїв.

4.2 Функціональність та використання кінцевого продукту

Web-орієнтована система проектування прототипів програмного забезпечення, що розробляється в рамках цього дипломного проекту, представляє собою інструмент, що дозволяє користувачам створювати, редагувати та зберігати

прототипи програмного забезпечення в режимі реального часу. Цей інструмент надає користувачам можливість візуалізувати свої ідеї, створювати прототипи програмного забезпечення та тестувати їх в режимі реального часу.

Система має інтуїтивно зрозумілий інтерфейс, що нагадує функціональність Windows Paint, з робочою областю для проектування та панеллю інструментів (рис. 4.5).

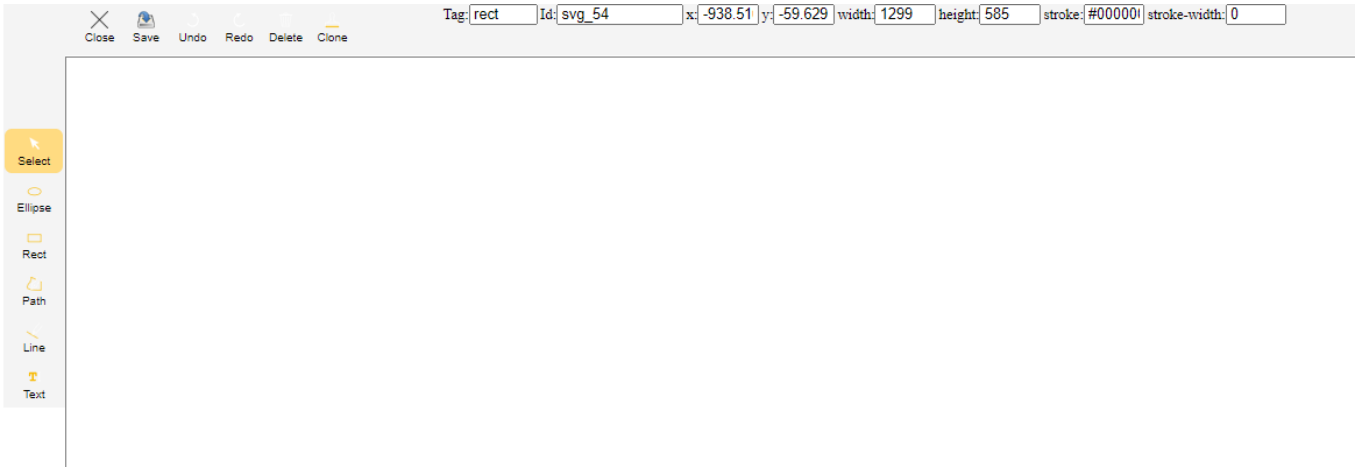

Рисунок 4.5 – Інтерфейс проекту

Джерело: побудовано автором (знімок з екрану)

Інструменти включають різні форми, такі як лінії, прямокутники та овали, а також інструменти для роботи з текстом. Користувачі можуть вибирати кольори для зафарбування та обведення об'єктів, змінювати розміри та пропорції фігур, клонувати та видаляти об'єкти.

Використання цієї системи починається з вибору потрібного інструменту з панелі інструментів. Після вибору інструменту, користувач може використовувати його на робочій області для створення об'єктів. Наприклад, для створення прямокутника, користувач повинен вибрати інструмент «Rect» з панелі інструментів, а потім натиснути та перетягнути курсор миші на робочій області для створення прямокутника (рис. 4.6).

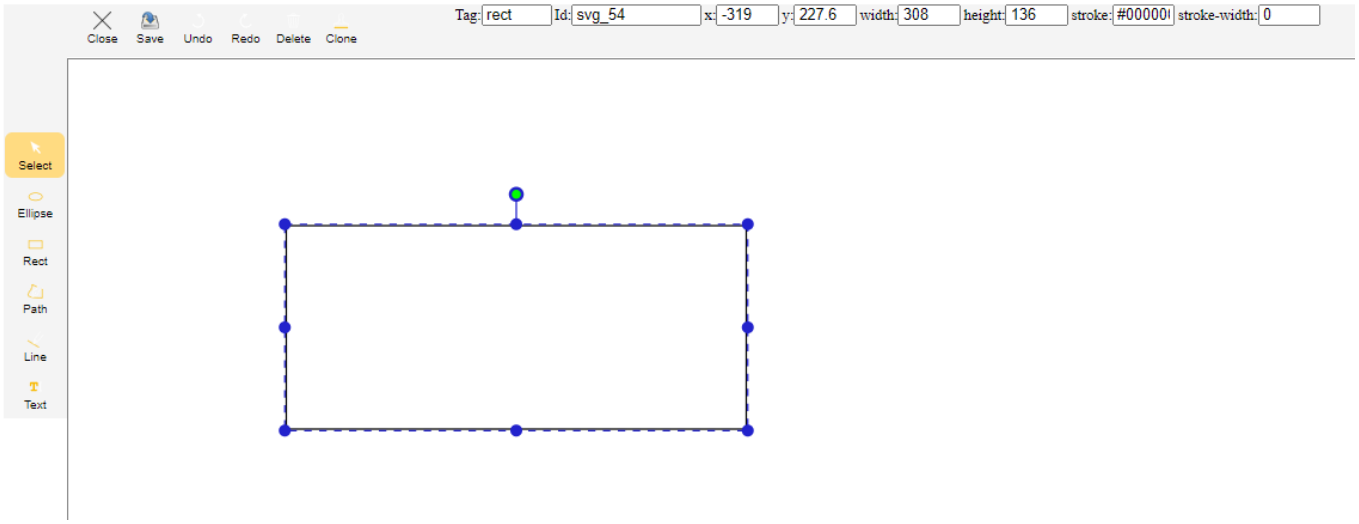

Рисунок 4.6 – Побудова фігури прямокутник

Джерело: побудовано автором (знімок з екрану)

Користувачі можуть вибирати різні кольори для зафарбування та обведення об'єктів. Для цього вони повинні вибрати потрібний колір з палітри кольорів, а потім застосувати його до об'єкта за допомогою інструменту "Зафарбувати" (рис. 4.7).

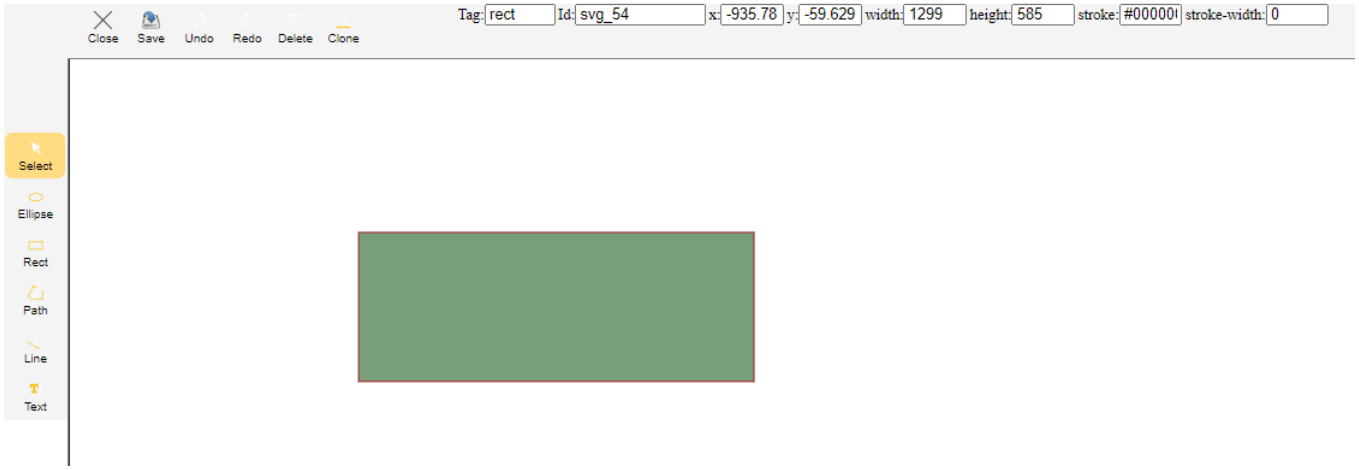

Рисунок 4.7 – Зафарбовування фігури

Джерело: побудовано автором (знімок з екрану)

Користувачі також можуть змінювати розміри та пропорції об'єктів за допомогою простого перетягування меж фігури (рис. 4.8).

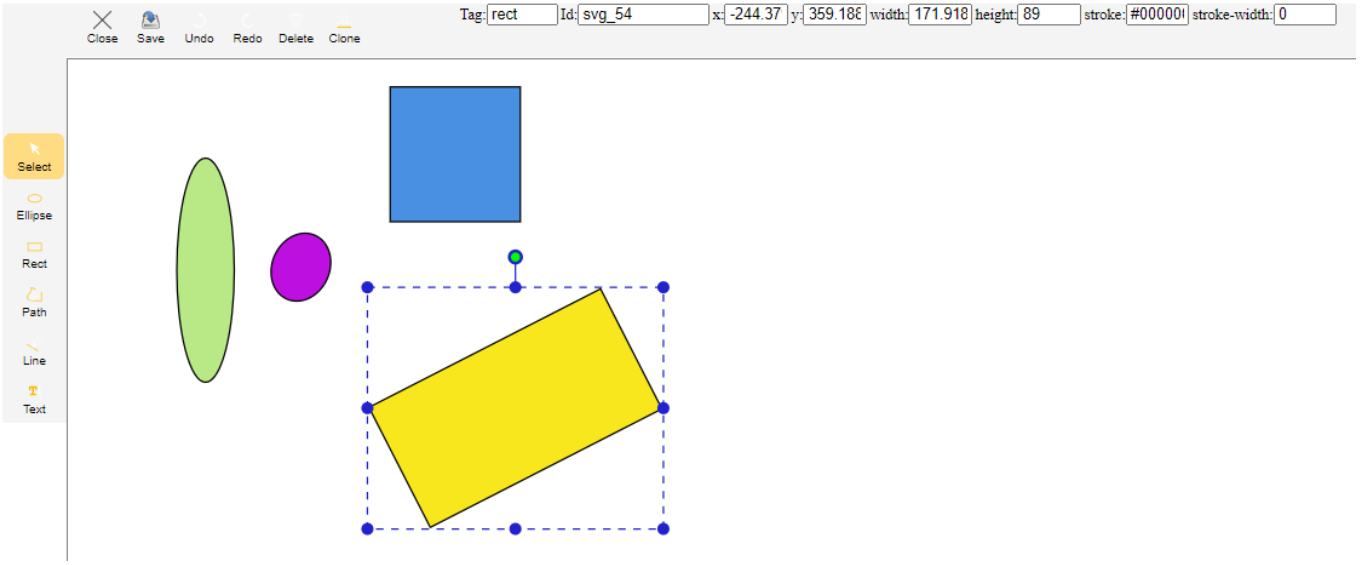

Рисунок 4.8 – Зміна розмірів та пропорцій фігури

Джерело: побудовано автором (знімок з екрану)

Крім того, система надає користувачам можливість клонувати та видаляти об'єкти. Для клонування об'єкта, користувач повинен вибрати об'єкт, а потім вибрати інструмент «Clone» (рис. 4.9).

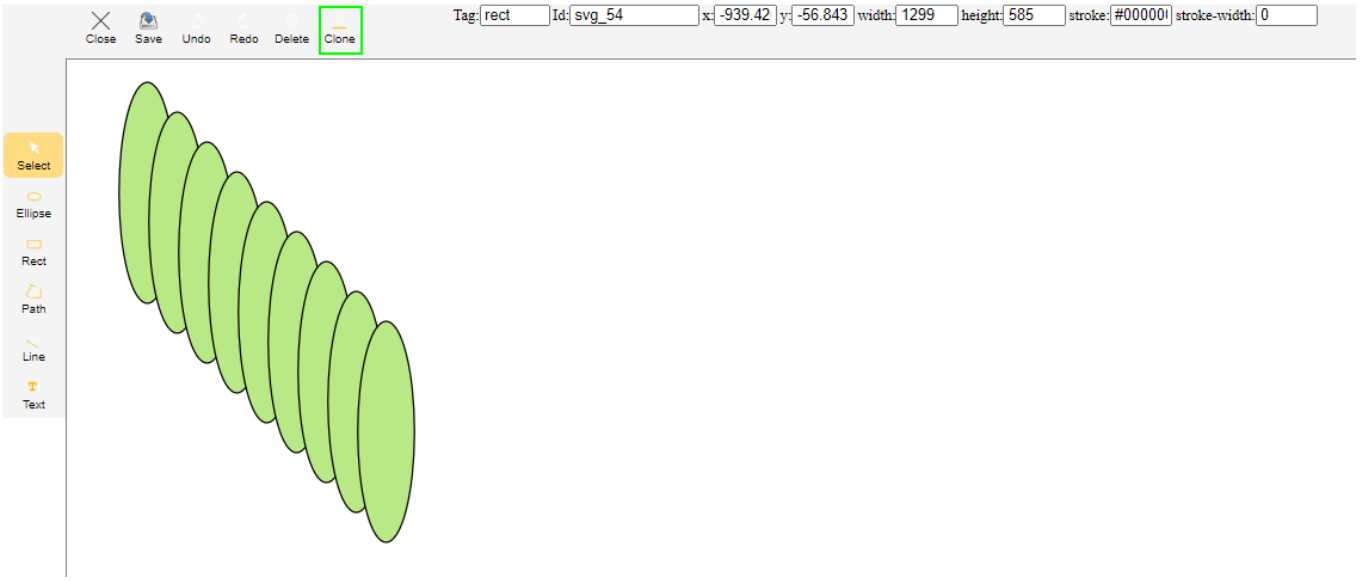

Рисунок 4.9 – Клонування фігури

Джерело: побудовано автором (знімок з екрану)

Для видалення об'єкта, користувач повинен вибрати об'єкт, а потім вибрати інструмент «Delete» (рис. 4.10).

Рисунок 4.10 – Видалення фігури

Джерело: побудовано автором (знімок з екрану)

Всі створені об'єкти можна зберегти у вигляді зображення для подальшого використання. Для цього користувач повинен вибрати опцію «Save» (рис. 4.11).

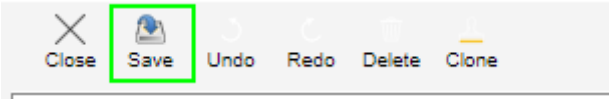

Рисунок 4.11 – Збереження прототипу

Джерело: побудовано автором (знімок з екрану)

В цілому, Web-орієнтована система проектування прототипів програмного забезпечення, розроблена в рамках цього дипломного проекту, має широкий спектр практичного застосування. Вона може бути використана розробниками програмного забезпечення, дизайнерами, менеджерами проектів, студентами та іншими професіоналами, які потребують ефективного інструменту для створення та тестування прототипів програмного забезпечення.

Розробники програмного забезпечення можуть використовувати цю систему для візуалізації та тестування своїх ідей перед розробкою повноцінного програмного забезпечення. Це може допомогти їм краще розуміти вимоги до проекту, виявити потенційні проблеми та виробити ефективні стратегії розробки.

Дизайнери можуть використовувати цю систему для створення прототипів дизайну інтерфейсу користувача (рис. 4.12). Це може допомогти їм візуалізувати свої ідеї, отримати відгук від користувачів та вдосконалити свої прототипи перед створенням кінцевого продукту.

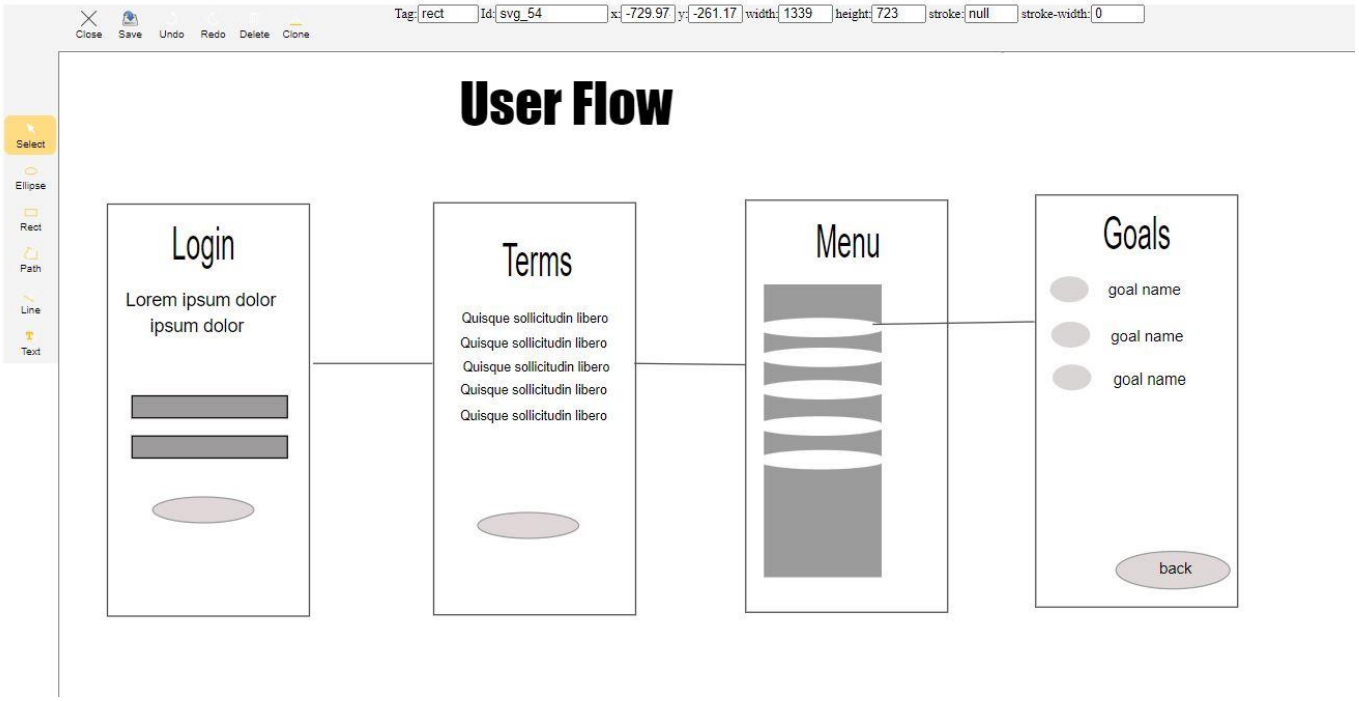

Рисунок 4.12 – Приклад прототипування користувацького досвіду

Джерело: побудовано автором (знімок з екрану)

Менеджери проектів можуть використовувати цю систему для планування та координації роботи над проектами. Це може допомогти їм візуалізувати структуру проекту, визначити завдання та ресурси, що необхідні для їх виконання, та відстежувати прогрес проекту.

Студенти можуть використовувати цю систему для навчання та практики в області розробки палітри кольорів програмного забезпечення (рис. 4.12). Це може допомогти їм краще розуміти процеси розробки програмного забезпечення, вдосконалити свої навички та підготуватися до реальних проектів.

У цілому, ця web-орієнтована система проектування прототипів програмного забезпечення є потужним інструментом, який може бути використаний в різних областях для ефективного створення, редагування та тестування прототипів програмного забезпечення.

Додатково необхідно зазначити, що Web-орієнтована система проектування прототипів програмного забезпечення має ряд переваг, які роблять її ефективним інструментом для створення прототипів програмного забезпечення.

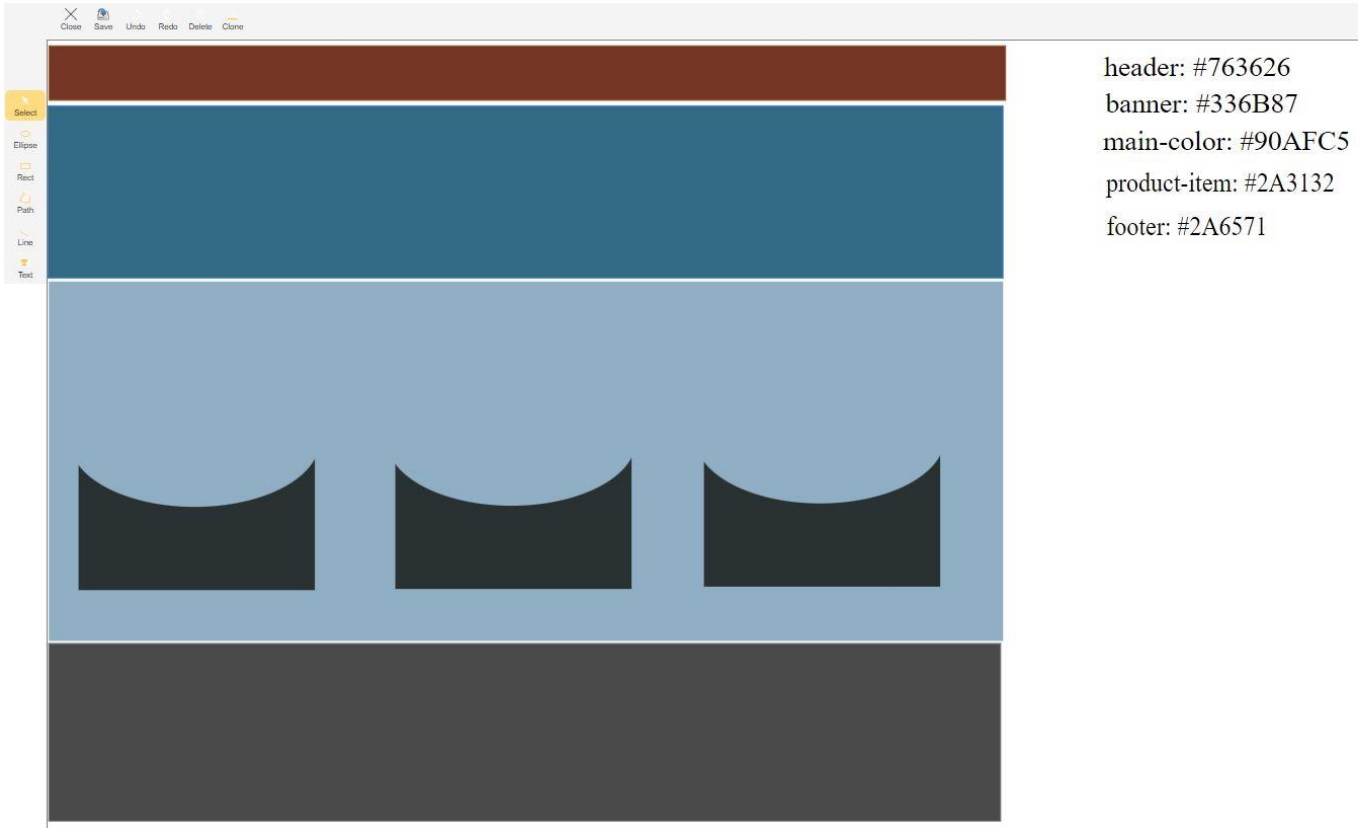

Рисунок 4.13 – Приклад прототипування палітри кольорів

Джерело: побудовано автором (знімок з екрану)

Перш за все, система є веб-орієнтованою, що означає, що користувачам не потрібно встановлювати додаткове програмне забезпечення на своїх комп'ютерах. Вони можуть просто відкрити веб-браузер, перейти на веб-сайт системи та почати створювати прототипи. Це робить систему легко доступною для користувачів з різних платформ та операційних систем.

Другою перевагою є інтуїтивно зрозумілий інтерфейс системи. Він нагадує функціональність Windows Paint, що робить його знайомим для багатьох користувачів. Це спрощує процес навчання та дозволяє користувачам швидко почати створювати прототипи.

Третьою перевагою є широкий спектр функціональних можливостей, які надає система. Користувачі можуть створювати об'єкти різних форм та розмірів, змінювати кольори, редагувати об'єкти, клонувати та видаляти їх, а також зберігати свої роботи у вигляді зображень. Це дозволяє користувачам візуалізувати та тестувати свої ідеї в режимі реального часу.

Четвертою перевагою є можливість співпраці та обміну прототипами. Користувачі можуть легко поділитися своїми прототипами з іншими користувачами, отримати відгук та вдосконалити свої ідеї. Це сприяє співпраці та інноваціям.

У цілому, web-орієнтована система проектування прототипів програмного забезпечення надає користувачам потужний інструмент, який допомагає їм ефективно візуалізувати та тестувати свої ідеї, співпрацювати з іншими користувачами та створювати інноваційні рішення.

Роздумуючи про майбутнє використання слід акцентувати, що Web-орієнтована система проектування прототипів програмного забезпечення, розроблена в рамках цього дипломного проекту, має великий потенціал для подальшого розвитку та використання. Ця система може бути вдосконалена та адаптована для використання в різних областях, включаючи освіту, наукові дослідження, промисловість та багато інших.

У сфері освіти, система може бути використана як навчальний інструмент для студентів, які вивчають розробку програмного забезпечення. Вона може допомогти студентам краще розуміти процеси розробки програмного забезпечення, вдосконалити свої навички та підготуватися до реальних проектів.

У сфері наукових досліджень, система може бути використана для створення прототипів нових технологій та ідей. Вона може допомогти дослідникам візуалізувати свої ідеї, провести експерименти та отримати результати в режимі реального часу.

У промисловості, система може бути використана для створення прототипів нових продуктів та технологій. Вона може допомогти інженерам та дизайнерам візуалізувати свої ідеї, тестувати їх та вдосконалити перед створенням кінцевого продукту.

Щодо подальшого розвитку системи, можливі напрямки включають додавання нових функцій та інструментів, покращення інтерфейсу користувача, інтеграцію з іншими системами та платформами, а також розширення можливостей співпраці та обміну прототипами.

ВИСНОВКИ

У процесі виконання дипломної роботи було досягнуто ряду важливих результатів, які відображають вирішення поставленої проблеми та її значення для практики.

У ході аналізу предметної області було вивчено основні поняття та принципи проектування програмного забезпечення та прототипування, а також розглянуто особливості web-орієнтованих систем. Це дало можливість глибше зрозуміти специфіку розробки web-орієнтованих систем проектування прототипів та визначити ключові вимоги до них.

Планування робіт включало визначення основних етапів розробки web-орієнтованої системи проектування прототипів програмного забезпечення та розробку загального плану реалізації проекту. Це допомогло ефективно організувати роботу над проектом та контролювати її хід.

Дослідження існуючих web-орієнтованих систем проектування прототипів дало змогу виявити їх переваги та недоліки, а також визначити потреби користувачів та вимоги до розроблюваної системи. В результаті було встановлено, що більшість існуючих систем мають обмежені можливості, високу вартість або складність у використанні, що підтверджує актуальність та практичну значимість розробки нової web-орієнтованої системи проектування прототипів програмного забезпечення.

У процесі проектування архітектури та дизайну системи було враховано вимоги та результати аналізу предметної області. Розроблена архітектура відповідає сучасним стандартам та принципам розробки програмного забезпечення, що забезпечує високу продуктивність, надійність та масштабованість системи. Дизайн системи є інтуїтивно зрозумілим та зручним для користувача, що сприяє ефективному використанню системи та підвищує її конкурентоспроможність.

Програмна реалізація web-орієнтованої системи проектування прототипів програмного забезпечення була виконана з використанням сучасних технологій та інструментів розробки. Розроблена система має широкий спектр функціональності,

що дозволяє вирішувати різноманітні задачі проектування прототипів програмного забезпечення. Крім того, система має високу продуктивність та надійність, що підтверджено результатами тестування.

Проведене тестування дозволило перевірити коректність роботи системи, її відповідність вимогам та виявити та усунути можливі помилки та недоліки. Результати тестування підтвердили високу якість розробленої системи та її готовність до впровадження.

Оцінка ефективності системи була проведена на основі аналізу її продуктивності, надійності, зручності використання та інших ключових показників. Результати оцінки показали, що розроблена система відповідає сучасним вимогам до web-орієнтованих систем проектування прототипів програмного забезпечення та може ефективно використовуватися для розв'язання відповідних задач.

Також було розглянуто можливі напрями продовження досліджень за темою роботи. Зокрема, було запропоновано розробити додаткові модулі для системи, що розширять її функціональні можливості, а також використати новітні технології для подальшого покращення продуктивності та надійності системи.

Таким чином вищенаведені висновки підтверджують, що поставлені задачі були вирішені, а результати роботи можуть бути використані для подальшого вдосконалення та впровадження web-орієнтованої системи проектування прототипів програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Munoz R. In search of the ideal prototype. New York: ACM Press, 1992. 604 p.
2. Baker A., Vander Hoek A. Ideas, subjects, and cycles as lenses for understanding the software design process // Design Studies. 2010. Vol. 31(6). P.590-613.
3. Behutiye W., Rodriquez P., Oivo M., Tosun A. Analysing the concept of technical debt in the context of agile software development // Information and Software Technology. 2017. Vol. 82. P. 139-158.
4. Bogner J., Verdecchia R., Gerostathopolos I. Characterising technical debt and antipatterns in AI-based systems // Int. Conf. on Technical Debt. 2021. Vol. 1. P. 64-73.
5. Loftus C. Can graduating students design: Revisited // In Proceedings of the 42nd SIGCSE technical symposium on Computer science education. 2011, March. P. 105–111.
6. Dreyfuss H. Designing for People. New York: Allworth Press, 1972. 135 p.
7. Constantine L. Beyond Chaos: The Expert Edge in Managing Software Development. San Francisco: Addison-Wesley, 2001. 182 p.
8. Izquierdo J., Molina J. Extracting models from source code in software modernisation // SoSyM. 2014. Vol. 13. P. 713-734.
9. Lano K., Kolahdouz-Rahimi S., Alwakeel L. Synthesis of mobile applications using Agile UML // ISEC. 2021. Vol. 9 P. 1-10.
10. Marinescu R. Assessing technical debt by identifying design aws in software systems //IBM Journal of Research and Development. 2012. Vol.56(5). P140-164.
11. Bowman M., Briand L., Labiche Y. Solving the class-responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms // IEEE TSE. 2010. Vol. 36(6). P. 817-837.
12. Buttner F., Gogolla M. On OCL-based imperative languages // Science of Computer Programming. 2017. Vol. 92. P. 162-178.

13. Nelson C., Siegel A. Parsimonious modelling of yield curves // Journal of Business. 1987. vol. 60(4). P. 473-489.
14. Fuentes-Fernandez R., Pavon J., Garijo F. A model-driven process for the modernisation of component-based systems // Science of Computer Programming. 2012. Vol. 77. P. 247-269.
15. Fuhr A., Horn T., Riediger V., Winter A. Model-driven software migration into service-oriented architectures // Comput. Sci. Res. Dev. 2013. Vol. 28. P. 35-84.
16. Gomes Rocha F., Misra S., Soares M., Guidelines for Future Agile Methodologies and Architecture Reconciliation for Software-Intensive Systems // Electronics. 2023. Vol.12. P. 1582-2023.
17. Sanchez D., Rojas A., Florez H. Towards a clean architecture for Android apps using model transformations // IJCS. 2022. Vol. 49. P.61-94.
18. Just R., Jalali D., Inozemtseva L., Ernst M., Holmes R., Fraser G. Are mutants a valid substitute for real faults in software testing // 22nd ACM SIGSOFT ISFSE. 2014. Vol 23. P. 654-665.
19. Krasteva I., Stavru S., Ilieva S. Agile software modernization to the service cloud // ICIW. 2013. Vol. 14. P. 1-9.
20. Kurtz T., History of Programming Languages // Science of Computer Programming. 2014. Vol. 13. P. 103-118.
21. Liu S., Li H., Jiang Z., Li X., Liu F., Zhong Y. Rigorous code review by reverse Engineering // Information and Software Technology. 2021 Vol. 133. P. 321-357.
22. Perez-Castillo R., Garcia-Rodriguez de Guzman I., Piattini M. Knowledge discovery metamodel ISO/IEC 19506: A standard to modernize legacy systems // Computer Standards and Interfaces. 2011. Vol. 33. P. 519-532.
23. Muggleton S., Raedt L. Inductive logic programming: theory and methods // Journal of Logic Programming. 1994. Vol.19. P. 629-679.
24. Nabavi E., Daniell K., Williams E., Bentley C. AI for Sustainability: A changing landcape, in artificial intelligence for better or worse // Future Leaders. 2019. Vol. 7. P. 17-58.

25. Salazar F., Brambilla M. Tailoring architecture concepts and process for mobile application development // 3rd International Workshop on Software Development Lifecycle for Mobile. 2015. Vol. 1. P. 21-24.
26. Marco A., Iancu V., Asinofsky I. COBOL to Java and newspapers still get delivered //International Conference on Software Maintenance and Evolution. 2018. Vol 3. P. 583-586.
27. Nguyen A., Nguyen T. Lexical statistical machine translation for language Migration // 9th Joint meeting on Foundations of Software Engineering. 2013. Vol. 8. P. 651-654.
28. Ogheneovo E. On the relationship between software complexity and maintenance Costs // Journal of Computer and Communications. 2014. Vol. 2. P. 1-16.
29. Pruitt J., Adlin T. The Persona Lifecycle. San Francisco: Morgan-Kaufman, 2006. 217 p.
30. Sneed H. Migrating from COBOL to Java: A report from the field // IEEE Proc. Of 26th ICSM. 2011. Vol. 2. P. 1-7.

ДОДАТОК А
ПЛАНУВАННЯ РОБІТ

А.1 Ідентифікація ідеї проекту

SMART-методологія є структурованим підходом до встановлення цілей проектів, які забезпечують їх досягнення через критерії, що ідентифікують цілі у відношенні до конкретності, вимірюваності, досяжності, релевантності та обмеженості умов цілі у часі. Розглянемо основну мету ІТ-проекту розробки веб-орієнтованої системи створення прототипів програмного забезпечення відповідно до цього підходу.

1. Конкретність (Specific). Розробка веб-орієнтованої системи для створення прототипів програмного забезпечення, яка надає користувачам можливість ефективно створювати, масштабувати, редагувати та експортувати векторні графічні зображення.

2. Вимірюваність (Measurable). Відходячи від задач проекту та зважаючи на обмеження, будуть вимірюванням мети проекту є число користувачів системи, а також їхнє задоволення від роботи з системою, швидкість системи та кількість обміну даними між користувачами.

3. Досяжність (Achievable): Мета проекту є здійсненною завдяки кваліфікованому персоналу, що має відповідний досвід роботи у розробці програмного забезпечення та відповідні знання з алгоритмів, графічних технологій та програмування. Технічне оснащення, також як і фінансові ресурси, буде доступним з тієї точки зору, що повинно прокласти відповідні шляхи досягнення мети.

4. Релевантність (Relevant): Мета є актуальна, оскільки вона відповідає потребам ринку, причому споживачі шукають нові досконалі технічні рішення для своїх поточних та майбутніх проектів. З погляду інтеграції в робочий процес, вона відповідає потребам ринку, дозволяючи розробникам, дизайнерам та іншим зацікавленим сторонам спільно працювати на створенні програмного забезпечення.

5. Обмеження у часі (Time-bound). Розробка веб-орієнтованої системи створення прототипів програмного забезпечення буде організована з метою досягнення успіху в установлені словах. У меті SMART визначений крайній термін комплектування та відлагодження системи. Усією командою буде розроблено чіткий план та графік роботи для досягнення мети відповідно до установленого дедлайну.

Таким чином, за допомогою методології SMART, мета IT проекту розробки веб-орієнтованої системи створення прототипів програмного забезпечення стає підсумком конкретних, вимірюваних, здійсненних, актуальних та обмежених у часі цілей. З використанням даної методології команда забезпечує досягнення мети проекту з максимальною продуктивністю та ефективністю.

А.2 Фази розробки проекту

Розробка IT-проектів, таких як веб-орієнтована система створення прототипів програмного забезпечення, полягає в різноманітних процесах залежно від забезпечення чіткої організації роботи команди і відповідної структури проекту. У цьому розділі ми взяли до уваги основні фази проекту, через які проходить команда розробників.

Фази розробки IT-проекту.

1. Ідея та аналіз потреб. На початку проекту команда працює над ідеєю та визначає основні потреби, які веб-орієнтована система проектування прототипів повинна задовольняти. У цій фазі досліджується ринок, аналізуються вимоги користувачів та виявляються основні технічні характеристики та функціональність продукту.

2. Розглядання технічної концепції та планування. На цьому етапі команда ретельно розглядає технічні аспекти розробки та виробляє план проекту. Це включає: вибір можливих архітектурних рішень, баз даних, мов програмування та наявність

інших ресурсів. Важливим етапом проекту є створення демонстраційних версій потрібних функцій з тим, щоб переконатися, що їх реалізовано успішно.

3. Проектування програмного проекту. Команда починає розробляти схему програмного забезпечення: структуру бази даних, архітектурні рішення, проектування API, зони обов'язків різних елементів програмного забезпечення та виділення ресурсів для розробки та тестування. У роботі над дизайном також беруться до уваги майбутні можливості підтримання та розвитку веб-орієнтованої системи.

4. Розвиток програмного забезпечення. У цій фазі команда фактично виконує програмування, створюючи ряд компонентів, що включають графічні інструменти, бази даних, API, взаємодію між компонентами та інші необхідні елементи для функціонування системи. Код розробляється у відповідності з обраним планом та використанням належних підходів.

5. Тестування. Після виконання розробки кожного етапу команда перевіряє продукт для достатньої якості, забезпечуючи відсутність помилок і застосування оновлення на частини системи. Це дозволяє забезпечити надійність, швидкість та стабільність програмного забезпечення.

6. Реалізація програмного забезпечення. На останньому етапі проекту команда проходить серію процесів впровадження, включаючи, установлення системи, навчання користувачів, технічну підтримку та налаштування. Ця фаза сприяє ефективній адаптації системи конечних користувачів.

У цілому успішна розробка IT-проекту веб-орієнтованої системи створення прототипів програмного забезпечення залежить від чіткої структури роботи команди та належного планування на кожній з фаз проекту. Забезпечуючи виконання плану проекту з чітким зосередженням на цілях, необхідному ресурсі та відповідних рішень по підтримці, команда розробників допомагає досягти означену мету, надає ефективні результати та допомагає у функціонуванні системи в довготривалому періоді.

Структура розкладу робіт (WBS) – це ієрархія та візуалізація всіх завдань, які потрібно виконати у рамках проекту. Нижче наведено WBS-структуру для проекту веб-орієнтованої системи створення прототипів програмного забезпечення (рис. А.1).

Рисунок А.1 – WBS-структура Web-орієнтованої системи проектування прототипів програмного забезпечення

Джерело: побудовано автором

WBS-структура допомагає чітко розуміти виділення ресурсів, спланувати етапи роботи та контролювати хід виконання проекту. Оскільки проект розбитий на окремі завдання, це сприяє відповідному контролю, оцінці та управлінню проектом під час його реалізації.

WBS-структура допомагає розробити OBS діаграму. OBS (Organization Breakdown Structure) – це ієрархія та представлення ролей, зони відповідальності та структури команди, включаючи розподіл роботи та взаємодію між учасниками проекту упродовж реалізації проекту веб-орієнтованої системи створення прототипів програмного забезпечення. Нижче перераховано основні елементи OBS (рис. А.2.).

OBS-структура допомагає керівництву проекту забезпечити ефективне організування команди та управління ресурсами, а також відповідні взаємодії та розподіл роботи між учасниками проекту веб-орієнтованої системи створення прототипів програмного забезпечення.

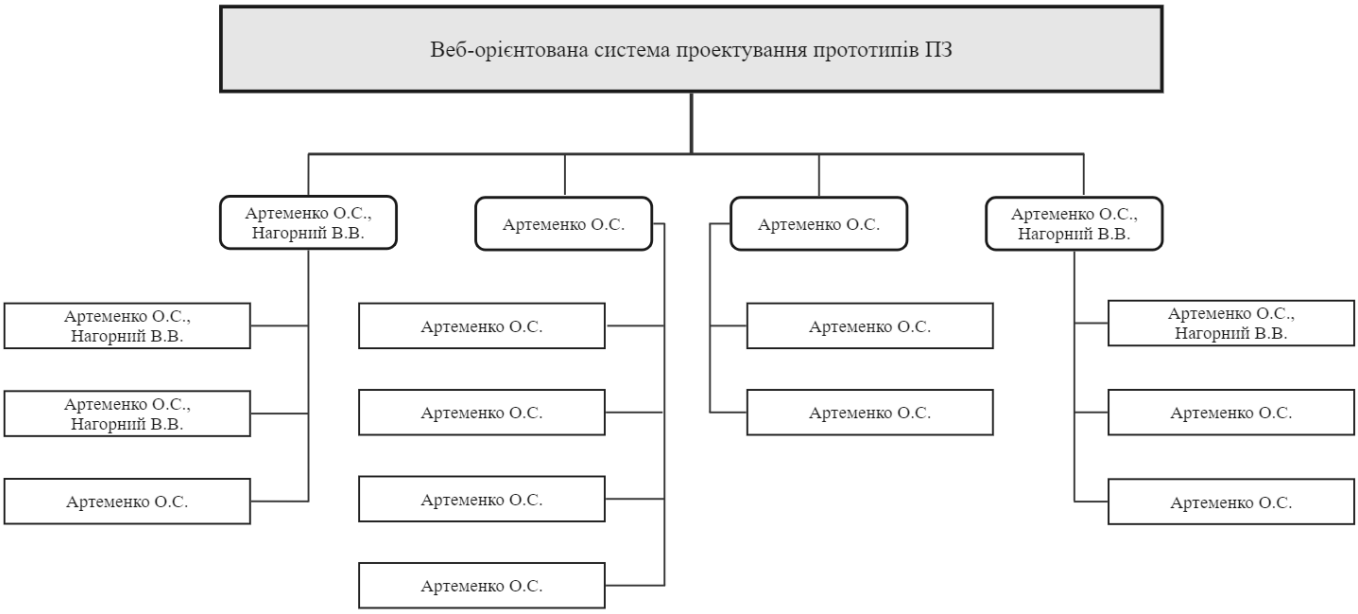

Рисунок А.2 – OBS-структура Web-орієнтованої системи проектування прототипів програмного забезпечення

Джерело: побудовано автором

У табл. А.1 побудована матриця відповідальності.

Таблиця А.1 – Матриця відповідальності проекту

Роль	ПІБ	Зона відповідальності
Розробник	Артеменко О.С.	Виконання front-end та back-end розробки
Проектувальник	Артеменко О.С.	Проектування бази даних та розробка структури web-додатку
Тестувальник	Артеменко О.С.	Тестування функціоналу та дизайну web-додатку
Керівник проекту	Нагорний В.В.	Формування завдань на розробку проекту
Менеджер проекту	Артеменко О.С.	Дотримання термінів, управління ризиками, звітування

Джерело: побудовано автором

При створенні графіка календаря, необхідно пам'ятайти, що точний графік залежить від доступності ресурсів, рівня взаємодії між учасниками проекту, узгодження та рішень в процесі реалізації проекту.

Враховуючи існуючі відомості та вихідні дані, прикладний календарний графік виконання проекту для Веб-орієнтованої системи створення прототипів виглядає наступним чином.

1. Огляд проекту (15-25 жовтня 2023 року).
2. Архітектура системи (26 жовтня - 12 листопада 2023 року):
   – вибір та створення архітектури бази даних;
   – специфікація та проектування API;
   – визначення технологічного стеку.
3. Розробка веб-інтерфейсу (13 листопада - 1 грудня 2023 року):
   – створення макету та структури інтерфейсу;
   – розробка графічних інструментів;
   – розробка взаємодії між компонентами;
   – інтеграція рішень для співпраці між користувачами.
4. Розробка серверної частини (1-30 листопада 2023 року):
   – реалізація API;
   – реалізація бази даних;
   – інтеграція з веб-інтерфейсом;
   – встановлення та налаштування серверів.
5. Тестування та відлагодження (1-8 грудня 2023 року):
   – розробка тестових сценаріїв;
   – функціональне тестування;
   – тестування продуктивності і сумісності;
   – виявлення і виправлення помилок.
6. Впровадження та підтримка (9-15 грудня 2023 року):
   – створення навчальних матеріалів та документації;
   – технічна підтримка користувачів;
   – оновлення та майбутні розширення.

Розроблений календарний графік, можна налаштувати відповідно до вашої команди разом з усіма необхідними адаптаціями згідно зі змінами в ході проекту.

Головне, стежте за належним моніторингом, контролем і комунікацією між учасниками проекту.

Розроблений календарний графік наочно відображений за допомогою діаграми Ганта (рис. А.3).

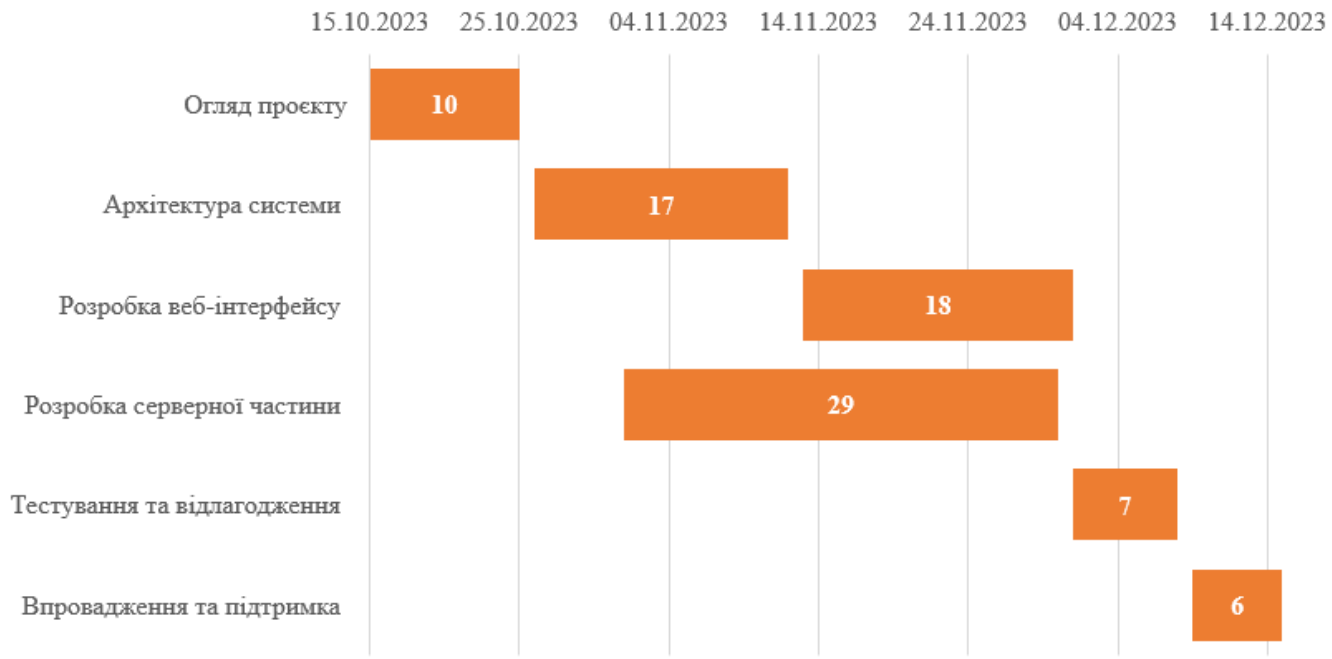

Рисунок А.3 – Календарний графік проекту

Джерело: побудовано автором

А.3 Аналіз ризиків проекту

Аналіз ризиків проекту – це важлива складова успішної реалізації обраного проекту веб-орієнтованої системи створення прототипів програмного забезпечення, оскільки він допомагає виявити, оцінити та в подальшому підготувати план дій для їх вирішення. В даному розділі ми зосередимося на аналізі потенційних ризиків проекту, випереджаючи можливі проблеми та несприятливі ситуації, які можуть мати негативний вплив на успішність проекту.

Перш ніж приступити до аналізу ризиків, слід визначити різні типи ризиків, які можуть виникнути у ході розробки проекту веб-орієнтованої системи створення прототипів програмного забезпечення. Ризики можна класифікувати за джерелом, їх впливом та ланцюжком причин і наслідків. Ось деякі основні категорії ризиків.

1. Технічні ризики: це ризики, пов'язані з технологічними аспектами розробки даної системи, такими як вибір правильного сервера, реалізація відповідних алгоритмів, архітектури й інтеграції з іншими системами.

2. Ризики управління проектом: Ці ризики стосуються управління ресурсами, керування часом, виділення бюджету, координації проектом та технічної документації.

3. Організаційні ризики: це ризики, які пов'язані з організаційною структурою команди розробників, культурою та спілкуванням між її членами, а також сумісність ролей та відповідальностей у проекті.

4. Ризики зовнішнього середовища: це ризики, які пов'язані з ринком, конкуренцією, регулюванням, законодавством або іншими факторами, що можуть вплинути на успішність проекту.

Для систематичного аналізу ризиків проекту, ми спробуємо визначити деякі потенційні ризики у кожній вищезазначеній категорії та оцінити ймовірність їх виникнення та можливі наслідки, що можуть виникнути від них.

1. Технічні ризики:

– вибір недостатньо продуктивного або несумісного сервера: це може обмежити віддалений обмін даними між користувачами або призвести до відмови системи;

– проблеми з безпекою даних: потенційні ризики, що пов'язані з витоком особистих даних користувачів або відмовою від доступу (DDoS-атаки);

– проблеми з інтеграцією: неправильна інтеграція з іншими системами може призвести до проблем у роботі системи та врешті-решт до невдачі проекту.

2. Ризики управління проектом:

– перевищення бюджету: неефективне керування ресурсами або неправильне розподілення бюджету може призвести до збільшення витрат проекту, що вимагатиме додаткових інвестицій;

– відхилення від графіка: відсутність чіткого планування роботи, загальна неконтрольованість процесів або складнощі у координації можуть призвести до затримок і вплинути на успішність проекту;

– нестабільний обсяг робіт: можуть виникнути ситуації, коли обсяг роботи зміниться, і це може стати причиною конфлікту ролей та дублювання робіт, що ускладнює процес реалізації проекту.

3. Організаційні ризики:

– недостатня комунікація: низька якість комунікацій або недостатня обмін інформацією між членами команди може перешкоджати ефективному вирішенню проблем;

– людські фактори: стрес, втому, перепрацювання та особисті конфлікти можуть вплинути на продуктивність працівників та викликати проблеми із виконанням завдань;

– відмова від проекту основних учасників: у ситуації, коли ключовий розробник або будь-який інший член команди покидає проект, це може відтермінувати процес реалізації.

4. Ризики зовнішнього середовища:

– зміна ринкових умов: зміни на ринку можуть вплинути на попит на наш продукт або відволікти нашу цільову аудиторію;

– юридичні ризики: зміна законодавчих актів або невідповідність до них може стати причиною санкцій або припинення роботи над проектом;

– зміна технологічного середовища: з'явлення нових технологій чи конкурентних продуктів може зробити проект менш рентабельним та акту.

Аналіз ризиків проекту допомагає зосередитись на мінімізації можливих проблем та несприятливих ситуацій у процесі впровадження веб-орієнтованої системи створення прототипів програмного забезпечення. Виявлення та оцінка різних видів ризиків дають змогу при потребі підготувати план дій для їх вирішення.

Як частина процесу аналізу ризиків проекту, можна зосередитись на певних видів потенційних ризиків та розробити стратегію та плани дій, які сприятимуть їх виявленню та запобіганню.

Таблиця А.2 – Оцінка рівня ризиків проекту

Ризик	Імовірність	Втрати	Рівень ризику
Відставання від графіку	4	4	Високий
Зміна ТЗ	2	3	Середній
Низька якість тестування	2	2	Низький
Зміна дат виконання роботи	3	4	Середній
Неточність ТЗ	1	4	Середній
Збої системи	3	3	Середній
Людський фактор	3	5	Високий

Джерело: побудовано автором

Зокрема, щодо технічних ризиків, необхідно забезпечити вибір правильної технологічної платформи та сервера для системи та розробити стратегії безпеки та інтеграції для захисту даних користувачів та сприяння сумісності з іншими системами.

Управління проектом включає ризики, пов'язані з розподілом бюджету, координацією ресурсів та чітким плануванням графіка виконання завдань. Розробка або вдосконалення процесів управління проектом може сприяти успіху проекту.

Нарешті, важливо враховувати ризики зовнішнього середовища, оскільки вони можуть вплинути на успіх проекту через зміни у відповідних ринків, конкуренції и правовому середовищі. Аналіз та стеження змін у ринковому середовищі та законодавстві допоможе команді реагувати на такі ризики та забезпечити стабільне успішне впровадження проекту.

Аналіз ризиків є ключовим компонентом успішного проекту, оскільки він допомагає виявити та оцінити певні ризики, що, в свою чергу, дозволяє прийняти кращі рішення та розробити відповідні плани дій для їх вирішення. Головне правило в очікуванні потенційних загроз та підготовці заходів з їх виявлення, попередження і запобігання – бути в курсі ситуації і безперервно оптимізувати та вдосконалювати процеси та пропозиції.

ДОДАТОК Б

Б.1 Лістинг програмного коду

Лістинг файлу «index.html»

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta
      http-equiv="Cache-Control"
      content=" no-store, no-cache, must-revalidate" />
    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Expires" content="0" />
    <title>Sample HTML SVG-Edit.react</title>
    <script src="./editor.js"></script>
    <link rel="stylesheet" href="editor.css" />
    <link rel="icon" href="data:;base64,iVBORw0KGgo=" />
  </head>

<body>
    <h3> Sample HTML SVG-Edit.react </h3>
    <div id="editcontainer"></div>
    <script>
      const element = document.getElementById("editcontainer")
      function logSvg(content) {
        console.log(content)
      }
      const editor = new Editor(element)
      editor.configure( 'saveHandler', logSvg )
       const svgContent = fetch("./arbelos.svg")
        .then(response => response.text())
        .then(svgContent => editor.load(svgContent))
    </script>
  </body>
</html>
```

Лістинг файлу «settings.json»

```
{
  "prettier.semi": false,
  "editor.showWhitespace": "boundary",
  "eslint.showStatusAlways": true,
  "merge-conflict.navigateNextConflict.auto": true,
  "prettier.configEditorUse": false,
  "prettier.quoteSingle": true
}
```

Лістинг файлу «.babelrc»

```
{
     "presets": ["@babel/env-preset", "@babel/react-preset"],
              "plugins": [
                "@babel/runtime-transform-plugin",
                "@babel/proposal-class-properties-plugin",
              ]
             }
```

Лістинг файлу «webpack.config.js»

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin')

const configuration = {
  entry: ['regenerator-runtime/runtime', './src/editor.js'],
  output: {
    path: `${__dirname}/dist/`,
    filename: 'editor.js',
    library: 'editor',
    libraryTarget: 'umd',
  },
  plugins: [
    new MiniCssExtractPlugin({
          filename: 'editor.css',
    }),
  ],
  module: {
    rules: [
      {
        test: /^(?!.*?\.module).*\.scss$/,
        use: ['css-loader', 'sass-loader'],
      },
```

```
    {
     test: /\.less$/i,
     use: [
       { loader: 'style-loader' },
       { loader: 'css-loader' },
       {
         loader: 'less-loader',
         options: { lessOptions: { strictMath: true } },
       },
     ],
    },
    { test: /\.(js|jsx)$/, exclude: /node_modules/, loader: 'babel-loader' },
    {
     test: /\.html$/,
     use: [{ loader: 'raw-loader' }],
    },
    {
     test: /\.svg$/,
     loader: 'svg-url-loader',
    },
    {
     test: /\.(png|jpg|gif)$/i,
     use: [
       {
         loader: 'url-loader',
         options: { limit: 40000 },
       },
     ],
    },
  ],
},
performance: {
  maxEntrypointSize: 1600000,
  maxAssetSize: 1600000,
},
optimization: {
  splitChunks: {
    cacheGroups: {
      styles: {
        name: 'styles',
        test: /\.css$/,
        chunks: 'all',
```

```
        enforce: true,
      },
    },
  },
 },
}

module.exports = (env, argv) => {
 if (argv.mode === 'development') {
  config.devtool = 'source-map'
 }
 return config
}
```

Лістинг файлу «jest.config.js»

```
// configuration for jest
module.exports = {
 verbose: true,
 transform: { '^.+\\.jsx?$': 'babel-jest' },
 moduleFileExtensions: ['js', 'jsx', 'json'],
 modulePaths: [
  '<rootDir>/node_modules/',
  '<rootDir>/tests/mock',
 ],
 moduleNameMapper: { '^(.*):(.*)$': '$1_$2' },
 unmockedModulePathPatterns: ['/^imports\\/.*\\.jsx?$/', '/^node_modules/'],
  setupFilesAfterEnv: ['raf/polyfill'],
 snapshotSerializers: ['enzyme-to-json/serializer'],
}
```