МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ ТА ДИСТАНЦІЙНОЇ ФОРМИ НАВЧАННЯ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Web-додаток для редагування діаграм UML»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТдн-84жт Рихлюк Богдан Олександрович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____ «___» _____ 2022 р.

Науковий керівник _____ _____ к.т.н., доц., Неня В. Г.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент_____
(підпис)

Суми-2022

Сумський державний університет
Центр заочної та дистанційної форми навчання
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. кафедри ІТ

_____ В. В. Шендрик
«__» _____ 2022 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Рихлюк Богдан Олександрович

1 Тема роботи *Web-додаток редагування діграм UML*

керівник роботи *Неня Віктор Григорович, к.т.н., доцент*

затверджені наказом по університету від « 11 » травня 2022 р. №0333-VI

2 Строк подання студентом роботи «14» червня 2022 р.

3 Вхідні дані до роботи Результати аналізу предметної області, вимоги державних стандартів

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Актуальність досліджуваної задачі, Перелік вимог до інформаційної системи, Аналіз існуючих методів та технологій, Огляд існуючих продуктів для вирішення поставлених задач, Призначення й мета інформаційної системи, Вимоги до інформаційної системи, Вимоги до видів забезпечення, Засоби розробки інформаційної системи

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Ілюстрації з кодом, інтерфейсом та функціоналом програми

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання___5 жовтня 2022_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	10.01.22 - 03.02.22	
1.1	Актуальність досліджуваної задачі	10.01.22 - 17.01.22	
1.2	Перелік вимог до інформаційної системи	18.01.22 - 21.01.22	
1.3	Аналіз існуючих методів, технологій	24.01.22 - 26.01.22	
1.4	Огляд існуючих програмних продуктів	27.01.22 - 07.02.22	
2	Постановка задачі	08.02.22 - 09.02.22	
2.1	Призначення й мета інформаційної системи	09.02.22 - 09.02.22	
2.2	Вимоги до інформаційної системи	09.02.22 - 09.02.22	
2.3	Вимоги до видів забезпечення	09.02.22 - 09.02.22	
3	Програмна реалізація	09.02.22 - 04.04.22	
3.1	Засоби розробки інформаційної системи	09.02.22 - 09.02.22	
3.2	Загальне представлення об’єктів	10.02.22 - 14.02.22	
3.3	Реалізація роботи Робочого поля	15.02.22 - 03.03.22	
3.4	Реалізація Панелі Інструментів	04.03.22 - 14.04.22	
3.5	Реалізація геометричних фігур	15.04.22 - 17.053.22	
3.6	Реалізація додаткового інтерфейсу	18.05.22 - 23.05.22	
3.7	Реалізація ліній	24.05.22 - 04.06.22	

Студент _____ Рихлюк Б. О.
(підпис)

Керівник роботи _____ к.т.н., доц. Неня В. Г.
(підпис)

РЕФЕРАТ

Тема роботи «Web-додаток для редагування діаграм UML»

Пояснювальна записка – 73 сторінки, кількість розділів – 3, кількість рисунків – 20, кількість використаних джерел – 20.

Дипломний проект присвячений розробці Web-додатка для редагування діаграм UML. В роботі проведено аналіз предметної області з аналізом та порівнянням уже існуючих рішень, визначення мети проекту, засобів реалізації, планування та проектування роботи. У роботі розглянуто тему UML-діаграм, їх види та спосіб застосування. Представлений опис та розробка всіх елементів додатку. Результатом проведеної роботи є Web-додаток для редагування діаграм UML.

Ключові слова: UML, Web, HTML, CSS, Javascript, React.

ЗМІСТ

ВСТУП

Абревіатура UML розшифровується як Unified Modeling Language. Вперше він з'явився наприкінці 1990-х років і продовжує грати життєво важливу роль у розробці програмного забезпечення.

Розробники систем та програмного забезпечення можуть використовувати UML для візуалізації, створення та документування артефактів для програмних систем, а також для бізнес-моделювання та інших непрограмних систем.

UML надає групам розробників потужний набір інструментів для створення різноманітних діаграм. Ці діаграми поділяються на два типи: структурні та поведінкові діаграми. Структурні діаграми відбивають статичні структурні компоненти системи. Діаграми поведінки зображають динамічну поведінку системи або те, як вона реагує на стимули. Усередині цих двох типів діаграм є багато інших видів діаграм.

Для побудови UML-діаграм потрібен інструмент що надає користувачу необхідний функціонал, тобто дозволяє діаграми будь-якого виду, серед них:

- Діаграма активностей (Activity diagram)
- Діаграма класів (Class diagram)
- Діаграма співпраці (Collaboration diagram)
- Діаграма розгортання (Deployment diagram)
- Діаграма послідовності (Sequence diagram)
- Діаграма станів (Statechart diagram)
- Діаграма варіантів використання (UseCase diagram)

Окрім можливості побудови різного типу діаграм, інструмент повинен надавати необхідний функціонал для роботи з ними, тобто можливість редагування уже готових діаграм.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність досліджуваної задачі

Моделі та діаграми є уявленнями реального додатка. Моделі забезпечують абстрактне уявлення про систему, а різні діаграми надають конкретні уявлення про систему[1].

Уніфікована мова моделювання (UML) Моделі представляють системи з різними рівнями деталізації. Деякі моделі описують систему з більш високого, більш абстрактного рівня, тоді як інші моделі надають більш детальну інформацію[2]. Моделі UML містять елементи моделі, такі як актори, варіанти використання, класи та пакети, а також одну або кілька діаграм, які показують конкретну перспективу системи. Модель також може містити інші, більш детальні моделі.

Вміст проекту моделювання організовано в три типи логічних папок: діаграми, моделі та профілі. Ця структура відображає логічне утримання елементів моделі UML, незалежно від того, де вони фізично зберігаються. Моделі в проекті моделювання відображаються в папці Models або вузлі. Ці вузли не є файлами фізичної моделі, які мають .emx як розширення імені файлу, а є кореневими елементами моделі моделей. Аналогічно, відповідні діаграми та профілі відображаються в папках Diagrams та Profiles відповідно[1].

Можна використовувати діаграми моделювання, щоб охопити випадки використання системи в моделі варіантів використання під час фази збору вимог. Визначається область застосування в моделі аналізу на етапі аналізу системи, а також уточнюється модель програми в моделі проектування під час детального аналізу[3].

Діаграма уніфікованої мови моделювання (UML) забезпечує візуальне представлення аспекту системи[3].

Діаграми UML ілюструють кількісно вимірювані аспекти системи, які можна описати візуально, такі як відносини, поведінка, структура та

функціональність. Наприклад, діаграма класів описує структуру системи або деталі реалізації, а діаграма послідовності показує взаємодію між об'єктами в часі[13].

На діаграмі UML елементи діаграми візуально представляють класифікатори в системі або додатку. Ці класифікатори є схематичним представленням вихідного елемента. Діаграми UML забезпечують уявлення вихідних елементів; однак елементи діаграми не мають семантичного значення.

Діаграми UML можуть допомогти системним архітекторам і розробникам співпрацювати та розробляти додаток. Архітектори та менеджери високого рівня можуть використовувати діаграми UML для візуалізації всієї системи чи проекту та окремих додатків на менші компоненти для розробки[4].

Системні розробники можуть використовувати діаграми UML для визначення, візуалізації та документування додатків, що може підвищити ефективність та покращити їх дизайн. Діаграми UML також можуть допомогти визначити моделі поведінки, які можуть надати можливості для повторного використання та оптимізації додатків.

Візуальне уявлення системи, яке надають діаграми UML, може запропонувати як низькорівневі, так і високорівневі уявлення про концепцію та дизайн програми[1].

Використовуються різноманітні типи діаграм для моделювання системи або програми на основі системи, яку потрібно створити. Залежно від вибору діаграми, можна вибрати деталізацію та рівень абстракції, які відображаються на діаграмах.

Типова модель UML може складатися з багатьох різних типів діаграм, кожна з яких представляє різний вигляд системи, що моделюється. Деякі приклади діаграм UML 2.1 і новіших включають діаграми варіантів використання, діаграми станів, діаграми послідовності та зв'язку, а також діаграми теми та перегляду. Деякі діаграми UML 2.1 і новіших версій також дозволяють використовувати довільні форми та форми, що не є UML[1].

Можна використовувати діаграми довільної форми для представлення загальних високорівневих уявлень моделі або альтернативних видів моделі, які

не можуть бути представлені стандартною нотацією UML. Існує два типи діаграм довільної форми:

1. Чиста діаграма вільної форми, яка не має семантичного контексту UML
2. Діаграми UML, які дозволяють використовувати елементи діаграми довільної форми, які є діаграмами варіантів використання, класів, компонентів і розгортань

Різниця між моделями UML та діаграмами: моделі - це абстрактні уявлення про систему. Модель використовує UML або іншу нотацію для опису системи на різних рівнях абстракції. Моделі часто містять одну або кілька діаграм, які графічно відображають аспект моделі або підмножину елементів моделі. Таким чином, діаграма представляє певний аспект або частину моделі. Діаграми можуть існувати поза моделлю або всередині моделі. Коли діаграму видаляють з моделі, елементи залишаються частиною моделі[4].

Діаграми ілюструють кількісно вимірювані аспекти системи, які можна описати візуально, такі як відносини, поведінка, структура та функціональність. Наприклад, діаграма класів описує структуру системи або деталі реалізації, тоді як діаграма послідовності показує взаємодію між об’єктами в часі.

1.2 Перелік вимог до Web-додатку для редагування діаграм UML

Моделювання проекту можна уявити як розслідування, і якщо бути конкретнішим, це дослідження об'єктів. Проектування означає співпрацю ідентифікованих об'єктів.

Отже, важливо розуміти концепції аналізу та проектування. Найбільш важливою метою аналізу є виявлення об'єктів проектованої системи. Цей аналіз також робиться для існуючої системи. Тепер ефективний аналіз можливий лише тоді, коли ми можемо почати думати, щоб можна було ідентифікувати об'єкти. Після ідентифікації об'єктів їх відносини ідентифікуються і, нарешті, створюється дизайн.

Мета аналізу та проектування може бути описана як :

1.Ідентифікація об'єктів системи.

2.Виявлення їхніх стосунків.

3.Ідентифікація об'єктів системи.

4.Виявлення їхніх стосунків.

5.Створення моделі всього проекту або системи за допомогою UML.

UML – це мова моделювання, яка використовується для моделювання програмних та непрограмних систем. Хоча UML використовується для непрограмних систем, акцент робиться на моделювання ОО (об’єктно-орієнтованих) додатків. Більшість UML-діаграм, що обговорювалися досі, використовуються для моделювання різних аспектів, таких як статичний, динамічний і т. д. Тепер, незалежно від аспекту, артефакти - це не що інше, як об'єкти[2].

Якщо ми подивимося на діаграму класів, діаграму об'єктів, діаграму співпраці, діаграми взаємодії, всі вони будуть в основному розроблені на основі об'єктів.

Отже, зв'язок між ОО-дизайном та UML є дуже важливим для розуміння. Конструкція ОО перетворюється на UML-діаграми відповідно до вимог. Перш ніж розпочати детальне розуміння UML, необхідно правильно вивчити

концепцію ОО. Як тільки аналіз та проектування виконані, наступний крок дуже простий. Вихідні дані для аналізу та проектування ОО є вхідними даними для діаграм UML[2].

Оскільки UML описує систему реального часу, дуже важливо створити концептуальну модель, а потім діяти поступово. Концептуальну модель UML можна освоїти, вивчивши три основні елементи:

- UML будівельні блоки
- Правила підключення будівельних блоків
- Загальні механізми UML

1.3 Аналіз існуючих методів та технологій

Стандарти UML визначають 13 типів діаграм

1.3.1 Схема класів

Оскільки багато програмного забезпечення засноване на об’єктно-орієнтованому програмуванні, де розробники визначають типи функцій, які можна використовувати, діаграми класів є найбільш часто використовуваним типом діаграми UML. Діаграми класів показують статичну структуру системи, включаючи класи, їх атрибути та поведінку, а також відносини між кожним класом.

Клас представлений прямокутником, який містить три відсіки, розміщені вертикально: верхній відсік містить ім’я класу і є обов’язковим, але два нижні відсіки дають детальну інформацію про атрибути класу та операції або поведінку класу[9].

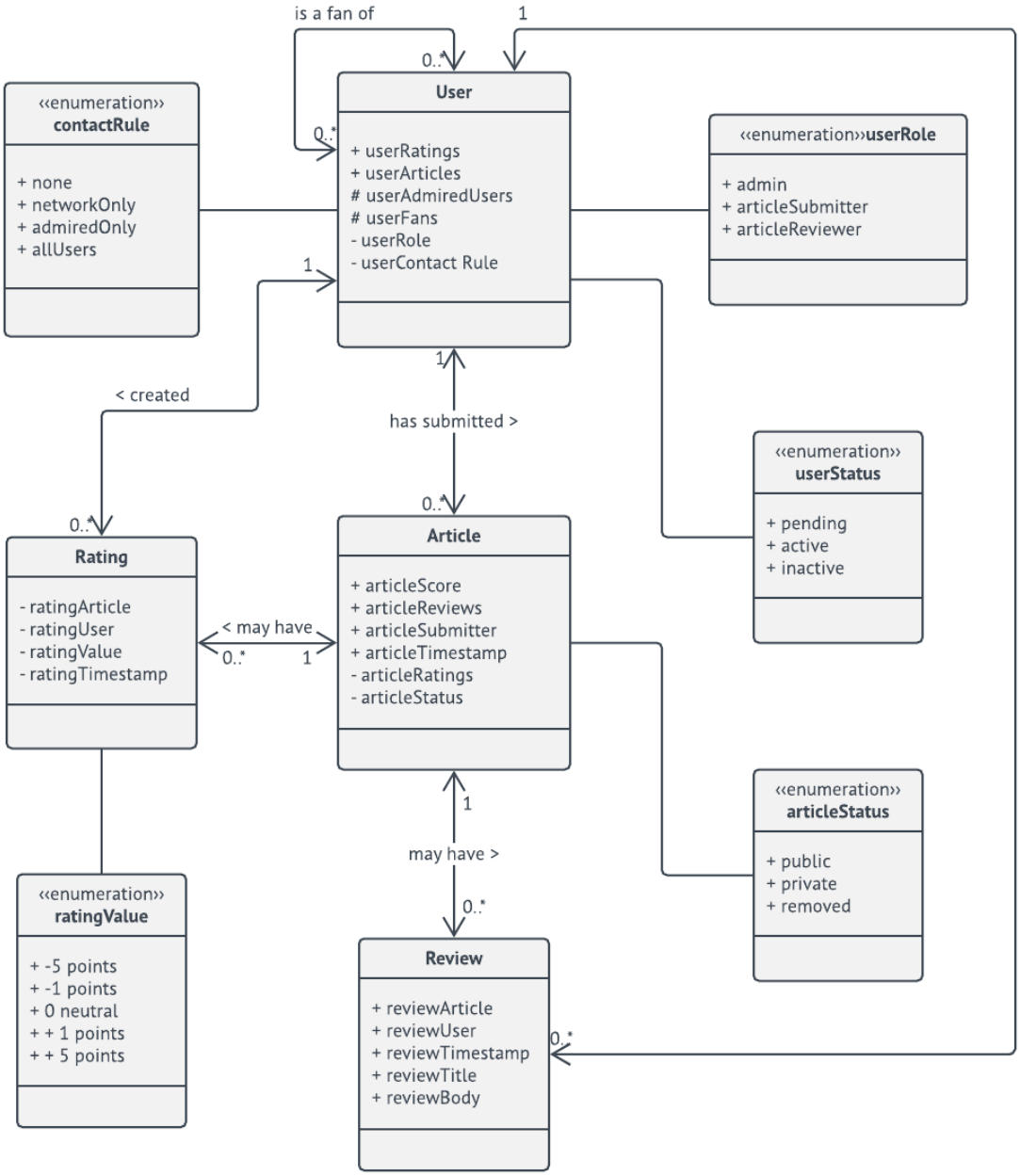


Рисунок 1.3.1 - Модель домену шаблон діаграми класів UML

1.3.2 Компонентна схема

Діаграма компонентів, по суті, є більш спеціалізованою версією діаграми класів — для обох застосовуються ті самі правила позначення. Діаграма компонентів розбиває складну систему на більш дрібні компоненти та візуалізує взаємозв’язок між цими компонентами[9].

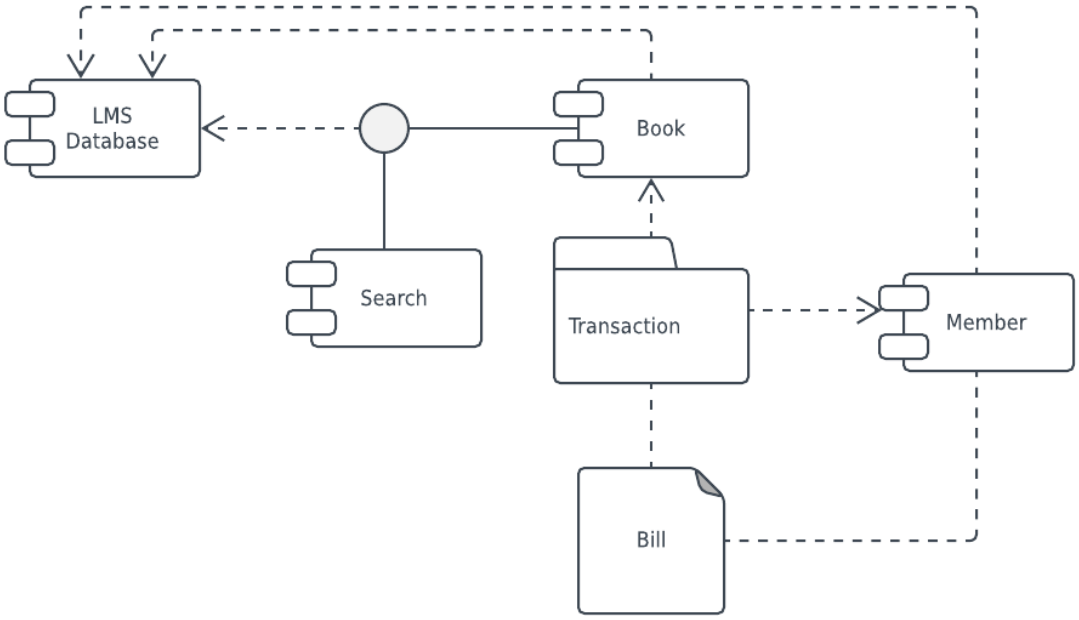


Рисунок 1.3.2 - Діаграма компонентів UML для системи управління бібліотекою

1.3.3 Схема розгортання

Діаграми розгортання показують, як програмне забезпечення розгортається на апаратних компонентах системи. Ці діаграми найбільш корисні для системних інженерів, і вони зазвичай показують продуктивність, масштабованість, ремонтопридатність і переносимість. Коли апаратні компоненти відображаються по відношенню один до одного, легше відстежувати всю вашу апаратну сітку та переконатися, що всі елементи враховуються під час розгортання[9].

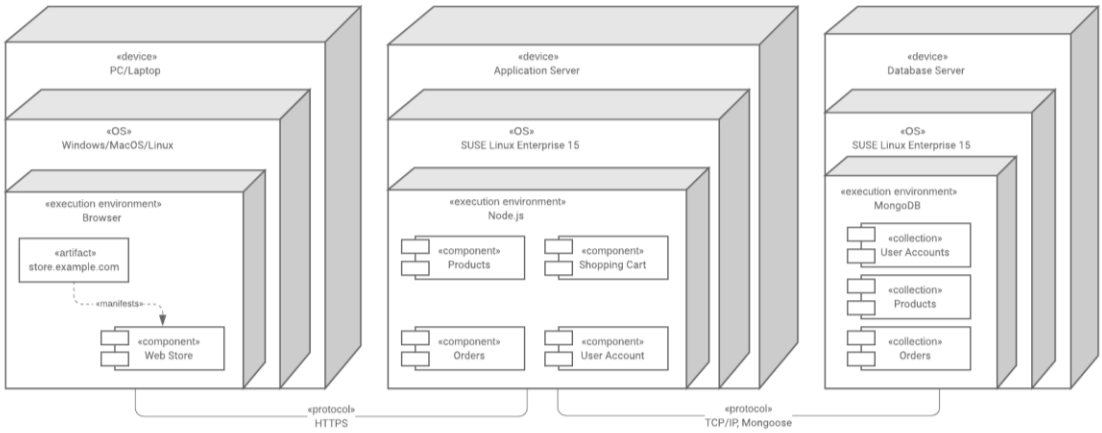


Рисунок 1.3.4 - Діаграма розгортання UML

1.3.5 Складена структура структури

Ці типи діаграм, по суті, є кресленнями внутрішньої структури класифікатора. Їх також можна використовувати, щоб показати поведінку спільної роботи або взаємодії класифікатора з їх середовищем через порти. Вони можуть легко відобразити внутрішні компоненти будь-якого обладнання, щоб глибше зрозуміти внутрішню роботу[9].

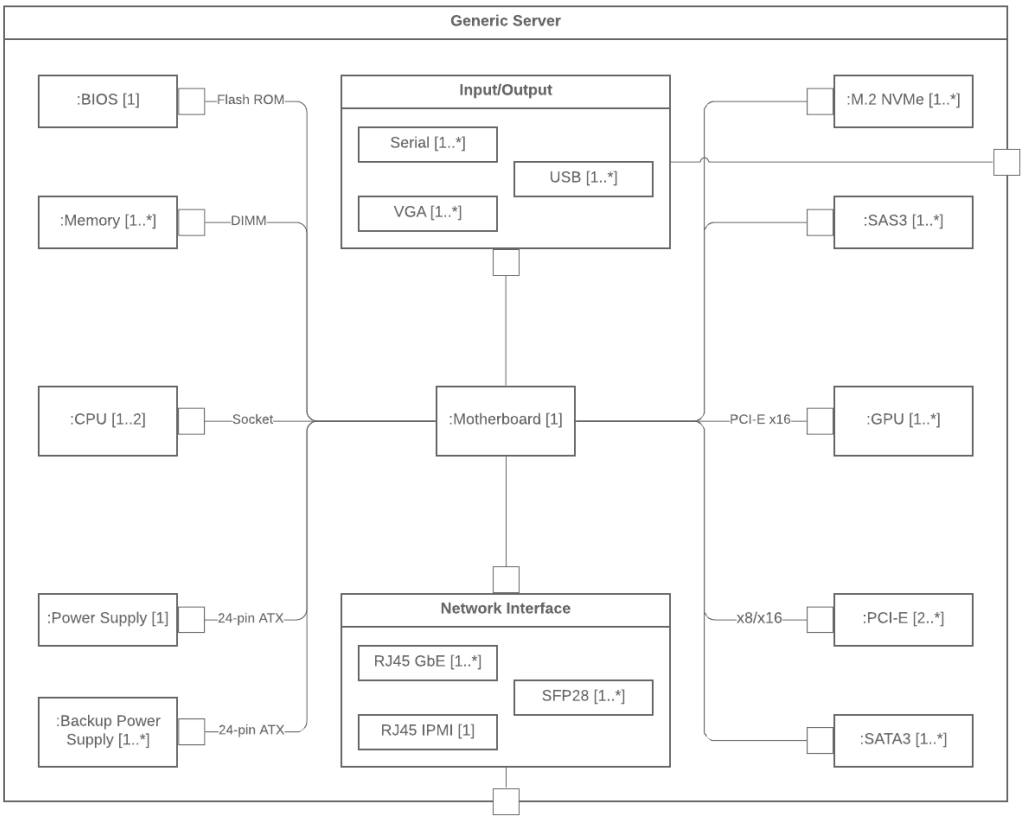


Рисунок 1.3.5 - Складена структура UML

1.3.6 Діаграма об'єкта

Діаграми об’єктів показують приклади структур даних у певний момент часу. Ви можете використовувати діаграму класів, щоб показати структуру, а потім використовувати діаграми об’єктів як тестові випадки, щоб перевірити повноту діаграми класів. Або ви можете створити діаграму об’єктів, щоб знайти інформацію про елементи моделі та їх зв’язки[9].

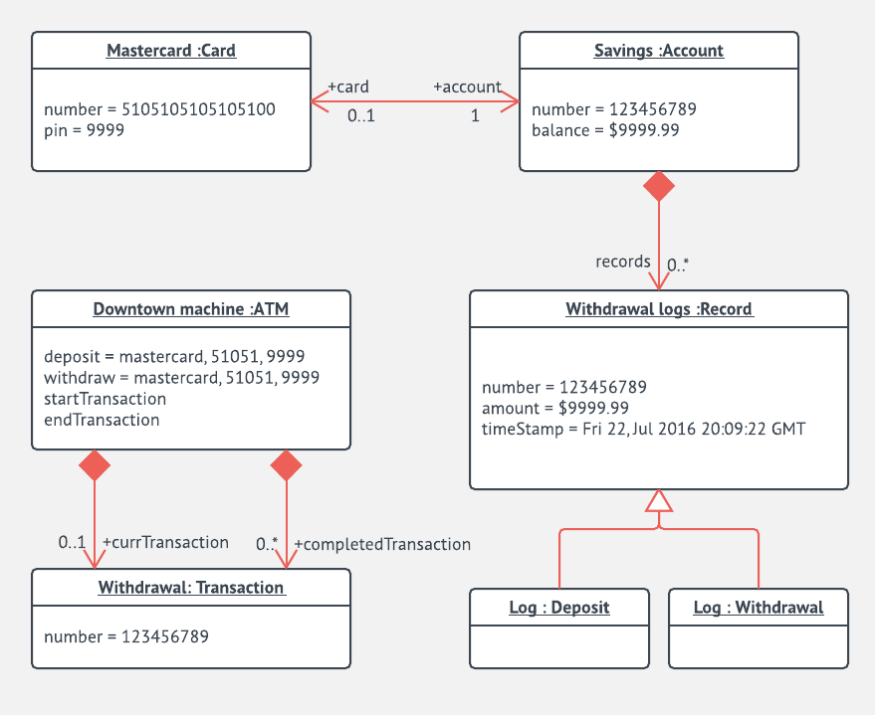


Рисунок 1.3.6 - Шаблон діаграми об'єкта банкомата

1.3.7 Схема пакета

Діаграми пакетів використовуються, щоб показати залежності між різними пакетами в системі. Пакет, зображений у вигляді папки файлів, організовує елементи моделі, такі як варіанти використання або класи, у групи[9].

Рисунок 1.3.7 - Схема пакету UML

1.3.8 Поведінкові діаграми UML

Ці UML-діаграми візуалізують, як система поводиться і взаємодіє сама з собою та з користувачами, іншими системами та іншими об’єктами[13].

1.3.9 Часова діаграма

Часто описується як діаграма оберненої послідовності, часова діаграма показує, як об’єкти взаємодіють один з одним протягом певного періоду часу. Використовуйте ці діаграми, щоб побачити, скільки часу займає кожен крок процесу, і знайти області для покращення[13].

1.3.10 Оглядова діаграма взаємодії

На цій діаграмі наведено огляд потоку керування між взаємодіючими вузлами. Вони включають початкові вузли, кінцеві вузли потоку, кінцеві вузли діяльності, вузли прийняття рішень, вузли злиття, вузли розгалуження та вузли приєднання[13].

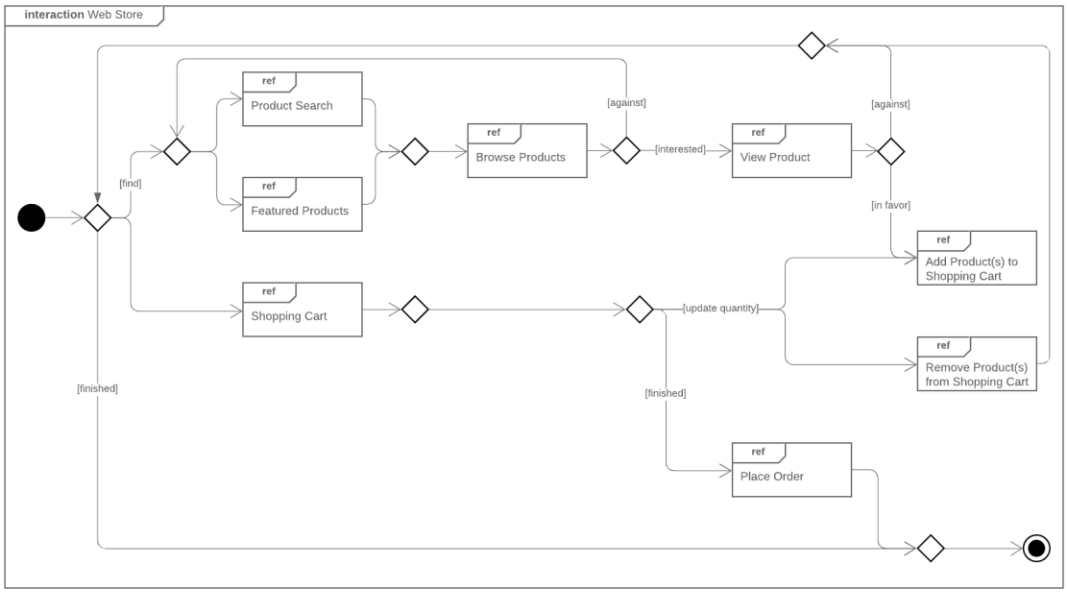


Рисунок 1.3.10 - Оглядова діаграма взаємодії UML

1.3.11 Схема зв'язку

Діаграми зв'язку, які раніше називали діаграмами співпраці, показують, як об'єкти пов'язані один з одним. Вони моделюють, як об'єкти асоціюються та з'єднуються через повідомлення в архітектурному проекті системи. Вони також можуть показувати альтернативні сценарії у випадках використання або операцій, які вимагають співпраці різних об’єктів та взаємодій[13].

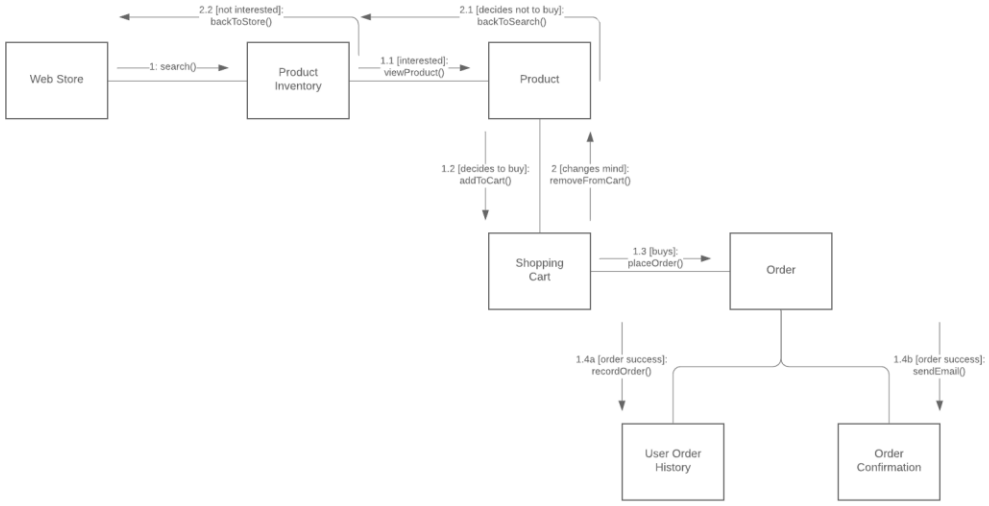


Рисунок 1.3.11 - Схема зв'язку UML

1.3.12 Діаграма стану

Діаграми станів, простіше кажучи, зображують стани та переходи. Стан відноситься до різних комбінацій інформації, які може містити об’єкт, і ця UML-діаграма може візуалізувати всі можливі стани та спосіб переходу об’єкта з одного стану в інший [13].

1.4 Огляд існуючих програмних продуктів для вирішення поставлених задач

Сьогодні існує великf кількість ресурсів, які допомагають у створенні таких UML діаграм онлайн. Деякі з них пропонують собою сайти, а деякі додатки та програми, які встановлюються на телефоні чи ПК.

1.4.1 Draw.io

Один з найбільш відомих ресурсів, використовуваних для створення блок-схем онлайн. Популярність сервісу обумовлена рядом факторів, але найбільш явні причини — широкий функціональний і простий використання. Інструменти зручно зібрані в боковому меню. На сайті відсутстня обов'язкова реєстрація, що ще більше спрощує роботу [11].

Особливості:

- Можна створювати різні типи схем.
- Є 5 шаблонів для блок-схем.
- Є кілька форматів імпорту, які підтримують сервіси: JPEG, PNG, SVG, сервіси Gliffy, Lucidchart і VSDX. Експорт можливий у форматах HTML, JPEG, PDF, PNG, SVG та XML.
- Можливість командної роботи завдяки підтримці віртуальних сховищ: Google Drive, OneDrive і DropBox.
- Широкий інструментарій, за допомогою якого можна створювати організаційні діаграми, блок-схеми (флоучарти), сетевые діаграми, UML, принципові електросхеми.

1.4.2 Wireflow

Сервісом Wireflow, як правило, користуються розробники додатків і веб-дизайнери. Він безкоштовний, як і Draw.io, і теж має великий набір різних функцій. За допомогою Wireflow можна розробляти каркаси сайтів і користувацькі потоки [11].

З ресурсом легко працювати, він інтуїтивно зрозумілий.

Особливості:

- Сервіс Wireflow безкоштовний.
- Наличие шаблонів і графічних об'єктів (более десяти).
- Сохранение проекта только в форматі JPG.
- Сервіс дозволяє працювати над проектом в команді.
- Є підтримка чата.
- Нет русскоязычной версии сервиса.

1.4.3 Microsoft Visio

Відома розробка компанії Microsoft, використовувана для створення UML-діаграм і різних блок-схем. Це не єдина функція програми: за допомогою Visio також створюються інженерні проекти (плани будівлі та поміщення).

Програма Microsoft Visio в її класичному вигляді передбачає установку на комп'ютер. Але вона також підтримує одночасну роботу в Інтернеті та в Office 365. У цьому випадку блок-схеми оновлюються, якщо змінюються вихідні дані, з якими вони пов’язані [11].

Особливості:

- За допомогою Visio створюються не лише блок-схеми, діаграми, організаційні діаграми, а також інженерні проекти та багато іншого.
- Блок-схеми та діаграми можна підключати до даних в реальному часі.
- Програма дозволяє працювати декільком людям над діаграмами онлайн практично з будь-якого місця.

1.4.4 Lucidchart

Lucidchart — один із найвідоміших сьогодні ресурсів для побудови блок-схем онлайн. Він простий та багатофункціональний [11].

Особливості:

- Командна робота підтримується, але є ряд обмежень.
- Імпорт файлів у форматі Visio (VSDX), Amazon Web Service (AWS) і Omnigraffle, JPEG, PDF, PNG, SVG.
- Проекти можна зберегти у форматах JPG, PNG, SVG, PDF та Visio.

- У сервісі є власне хмарне сховище, куда можна зберегти чорні версії проектів.
- Lucidchart надає готові шаблони.
- Ресурс підтримує кілька мов.

У сервісі є безкоштовна і платна версія. Платна вартість включає в себе три тарифи, які відрізняються ціною (від $5,59 до $20 на місяць) і наповненням.

1.4.5 Cacoo

Cacoo схожий з Lucidchart. Його відмінність в тому, що він більш спрямований на створення блок-схем. Сервіс містить безліч інструментів, шаблонів і форм [11].

Спочатку ресурс розроблявся для командної роботи, тому цей аспект реалізований дуже добре.

Особливості:

- Підтримується коментування, звичайний чат, а також відеочат.
- Сервис платний. Вартість залежить від обраного тарифу і становить від $4,95 до $18. Перші 14 днів надається безкоштовна пробна версія.
- Можна вибрати різні типи проектів.
- Наявність готових шаблонов.
- Є власне сховище, яке вміщає до 1000 файлів.
- Імпорт і експорт документів виробляються у форматі SVG.
- Перегляд історії змін.

1.4.6 Google Малюнки

Компанія Google також надає сервіс, здатний допомогти у створенні блок-схем онлайн. Функція називається Google Drawings або «Google Малюнки». Ресурс представлений стандартним набором функцій. Він нескладний та інтуїтивний, як і всі сервіси компанії.

Google Drawing, як та інші сервіси Google, пов’язані з Google Диском, тому є можливість командної роботи з розробкою настроєї автоматично. Для

надання доступу всім учасникам команд копіюється і відправляється посилання на проект [11].

Особливості:

- Проект експортується у форматі PNG або SVG. Також є можливість публікації в інтернеті.
- На диску доступно тільки 15 Гб.
- Є доступ до великої бази зображень.
- Сервіс підтримує гіперпосилання.
- Можна додавати видео.
- Поддерживается русский язык.

1.4.7 Gliffy

Gliffy — це додаток, яке працює в режимі офлайн. Акцент ставиться на простоту та швидку роботу, тому розробники не робили широкий функціонал, але базові інструменти в додатках є, а також є багато шаблонів [11].

Особливості:

- Є підтримка командної роботи.
- Сервис платный. Версія для самостійного використання коштує $7,99, для командного використання — $4,99 для кожного користувача.
- Сервіс має нестандартний інтерфейс, до якого потрібно привикнути.
- Предоставляется можливість викладати проект на зовнішні ресурси. Також можна приглашати інших користувачів переглянути, коментувати та редагувати файл.
- Доступ до перегляду історії редагування.

1.4.8 Textografo

Сервіс сильно відрізняється від інших. Усі роботи в тому, що блок-схеми та діаграми створюються за допомогою тексту. Використовується спеціальний

синтаксис, користувач вводить необхідні слова, і програма сама створює схему або діаграму [11].

Особливості:

- Підтримується командная работа.
- Є власне хранилище, куди зберігаються файли.
- Сервіс платний.

1.4.9 OmniGraffle

Сервіс розроблений для MacOS та iOS. Крім блок-схем надається можливість роботи з векторною графікою. Є можливість редагування данных з клавіатури. JavaScript може допомогти оптимізувати та автоматизувати роботу ресурсу, якщо ви знаєте мову [11].

Особливості:

- Сервіс платний, але є безкоштовна пробна версія, яку можна використовувати протягом 14 днів. Далі можна вибрати найбільш відповідний тариф від $50 до $250.
- Програма розроблена для MacOS та iOS, так що підійде не всім.
- Можливість «підгоняти» систему під себе.
- Хороший инструментарий.

1.4.10 SmartDraw

Цей сервіс вважається аналогом Visio, так як пропонує дві версії — програму на комп’ютер і сервіс для роботи в онлайн-режимі. Крім того, схожість спостерігається в можливостях проектування не тільки блок-схем і UML-діаграм, а й різних інженерних проектів, наприклад, планів будівлі по поверхам [11].

Особливості:

Два варіанти роботи: онлайн та за допомогою попередньо встановленої програми.

- Підходить як для Windows, так і для MacOS.
- Подходит для інженерних проектів.

- Програма пропонує кілька десятків шаблонів.
- Сервіс платний. Вартість використання — $15 в місяць.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ
2.1 Призначення й мета інформаційної системи

2.1.1 Призначення інформаційної системи

Інформаційна система повинна Web-додаток для редагування діаграм UML

2.1.2 Мета створення інформаційної системи –

Отримати додати для роботи UML діаграмами.

2.1.3 Цільова аудиторія

У цільовій аудиторії інформаційної системи можна виділити наступні групи:

1. Програмісти.
2. Менеджери.
3. Архітектори.
4. Тестувальники.
5. Особи що вивчають програмування.

2.2 Вимоги до WEB-додатку

2.2.1 Вимоги до WEB-додатку в цілому

Додаток повинен буди представлений у вигляді сайду, що складається з двох частин, списку з інструментами та робочим полем, на якому можна створювати UML-діаграми та редагувати їх. Користувач повинен мати змогу створювати повноцінні UML-діаграми в робочому полі за допомогою інструментів, що розташовані в окремій вкладці. Інтерфейс повинен давати можливість повноцінно їх редагувати, в тому числі вільно переміщувати схеми по робочому полю, як всі разом так різні елементи окремо, додавати та видаляти елементи, наносити текст на схеми та з'єднувати їх між собою, мати просту навігацію та бути інтуїтивно зрозумілим.

2.2.2 Структура сайту

Сайт являє собою односторінковий додаток з полем, на якому можна малювати та редагувати діаграми, та панелі інструментів на якому відображаються доступні діаграми, а також інші інструменти такі як «Текст» або «Вільне малювання»

2.2.3 Панель інструментів

Панель інтрументів являє собою список геометричних фігур, кожну з яких можна перенести на робоче поле, навівши миш на відповідну фігуру і з зажатою лівою клавішею, перетягнути її у бажане місце. Також в панелі присутні такі інструменти як «Текст», який дозволяє додавати текстові поля на робоче поле, інструмент «Вільне малювання», що дає змогу малювати на робочому полі довільні зображення, а також інструмент «Лінія», за допомогою якої можна з'єднувати розташовані геометричні фігури між собою.

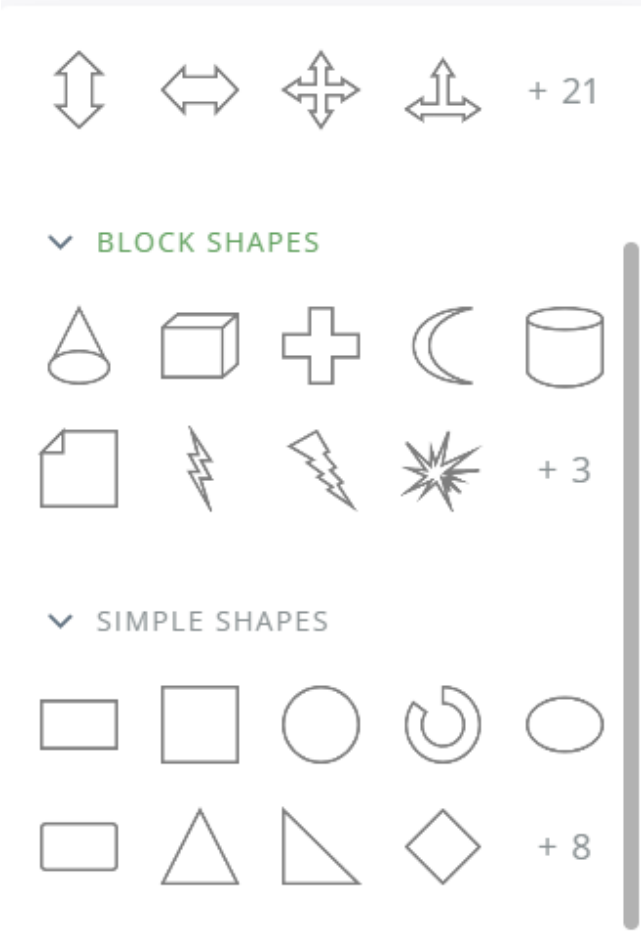


Рисунок 2.2.3 - Панель інструментів

2.2.4 Робоче поле

Робоче поле являє собою частину інтерфейсу додатку на якій можна безпосередньо створювати діаграми, переміщуючи їх з Панелі інструментів. На ньому можна розміщати геометричні фігури, текст, тощо, вільно переміщувати їх по полю, змінювати їх розмір та кут нахилу. На Робочому полі розміщувати текст, як всередині, так і поза межами фігур, а також додавати вільні малюнки. Користувач повинен мати змогу вільні переміщуватись по Робочому полю, змінюючи його масштаб та положення. Також повинна бути реалізована функція виділення однієї або декількох об’єктів на полі, а у кожного об’єкта, при потраплянні на поле, мають бути додаткові інтерфейси для більш зручної взаємодії з ними.

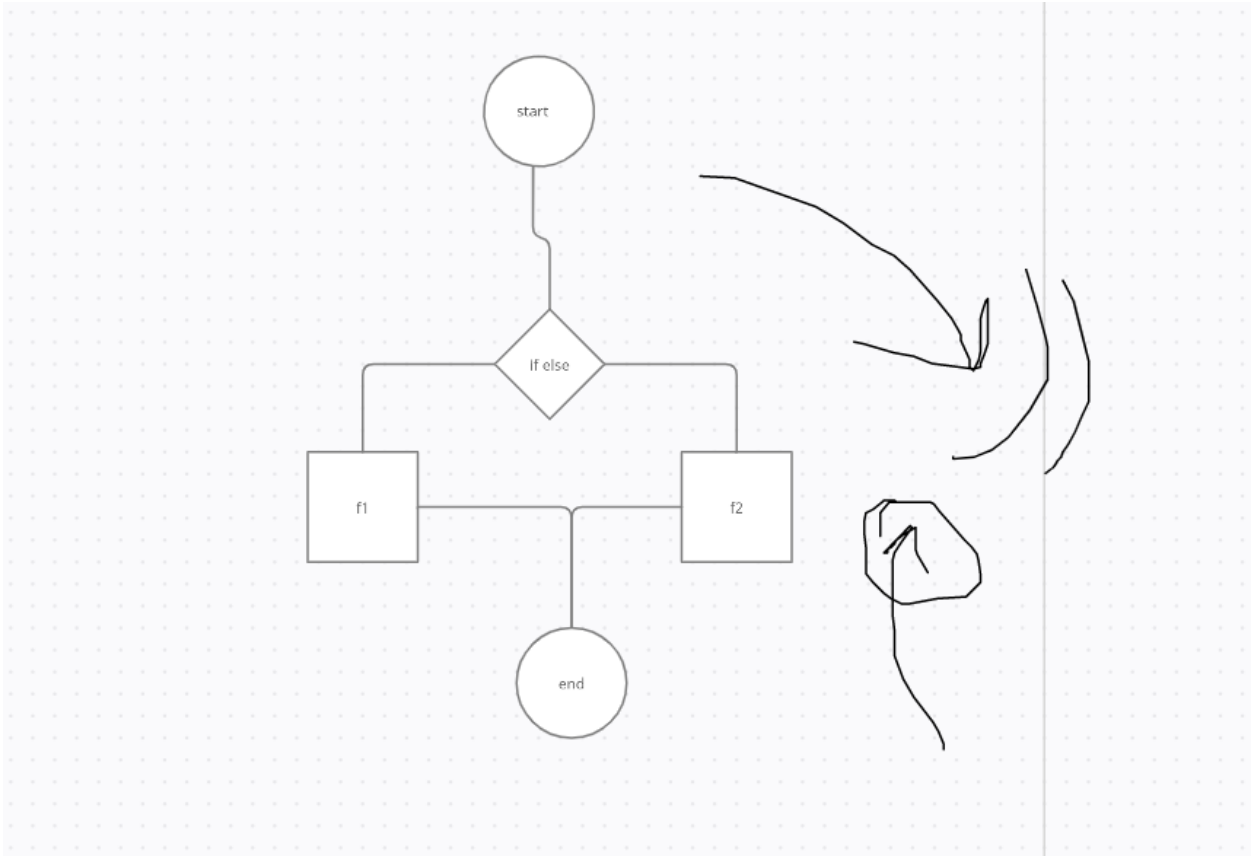


Рисунок 2.2.4 - Робоче поле

2.2.5 Геометричні фігури

При потраплянні на Робоче поле у геометричних фігур з’являється додатковий інтерфейс, що дозволяє змінити розмір та кут нахилу фігури, а також одразу створити наступну фігуру, що уже буде об’єднана з даною фігурою за допомогою лінії

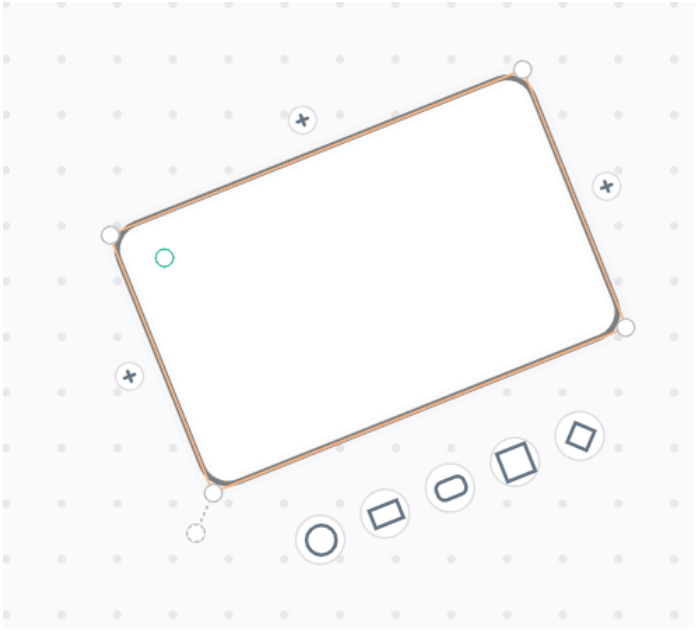


Рисунок 2.2.5 - Додатковий інтерфей геометричної фігури

2.2.6 Вільне малювання

Малюнки створені за допомогою даного інструменту, мають такий самий додатковий інтерфейс як і фігури, і можуть не лише змінювати маштаб і кут нахилу, а навіть бути з’єднані лініями з іншими малюнками або фігурами

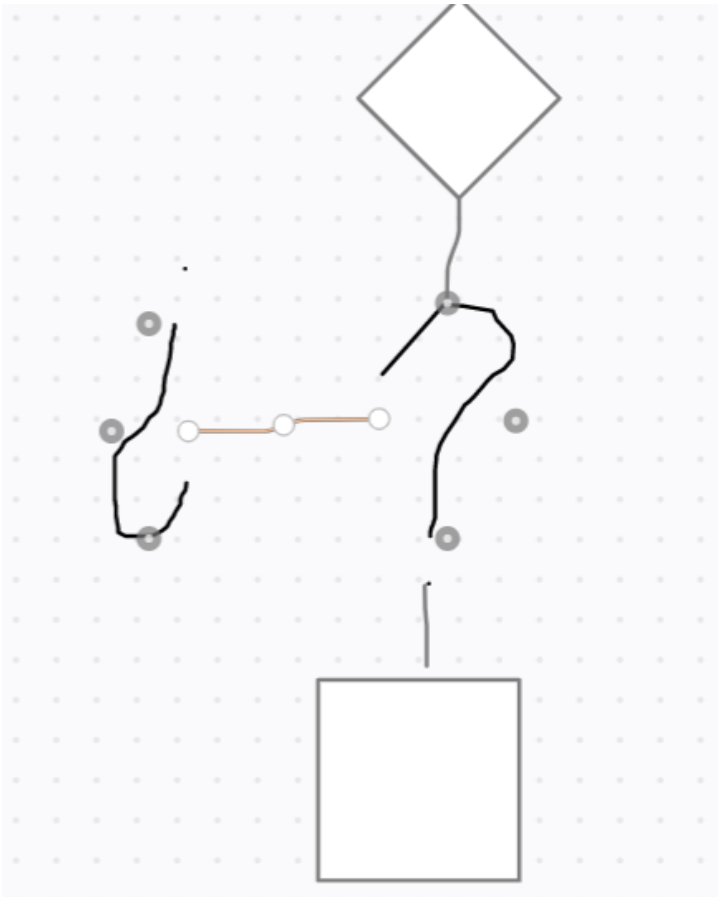


Рисунок 2.2.6 - Додатковий інтерфейс у малюнків

2.2.7 Лінії

При Лінії дозволяють з’єднувати між собою фігури і малюнки розташовані на Робочому полі. При на веденні інструменту «Лінія» на об’єкт, що знаходиться на полі, у даного з’являються точки до яких можна приєднати лінію. При переміщенні фігури, лінія не розриваються.

2.3 Вимоги до видів забезпечення

2.3.1 Вимоги до інформаційного забезпечення

Реалізація сайту відбувається з використанням:

- HTML
- CSS
- JavaScript
- React.js

2.3.2 Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- Веб-браузер: Internet Explorer 7.0 і вище, або Firefox 3.5 і вище, або Opera 9.5 і вище, або Safari 3.2.1 і вище, або Chrome 2 і вище;
- Включена підтримка javascript, Flash і cookies.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Засоби розробки інформаційної системи

3.1.1 VS Code

Visual Studio Code є одним із популярних текстових редакторів, які використовуються професіоналами та рекомендовані новачкам. Будучи однією із відомих програм Microsoft, VS Code є безкоштовним у використанні, з відкритим вихідним кодом і сумісним із Windows, Linux та macOS [12].

Однією з ключових причин, чому код VS Code виділяється, є його універсальність і сучасні функції.

Оскільки редактор коду не може надати рішення для всіх можливих проблем, VS Code підтримує розширення.

Розширення — це функції сторонніх розробників, які програмісти можуть додати, щоб покращити якість, функціональність та зовнішній вигляд коду. Програмісти також можуть додавати мови, налагоджувачі та інші інструменти за допомогою розширень.

Наприклад, VS Code не підтримує C++, хоча це все ще найкраще середовище програмування для C++. Програмування на C++ можливе завдяки розширенням C/C++ редактора. Після додавання таких корисних розширень розробникам стає легше кодувати за допомогою VS Code.

Враховуючи, що VS Code є програмою відкритим кодом, сторонні розширення з інноваційними функціями постійно надходять у редактор і регулярно оновлюються.

3.1.2 HTML

HTML, скорочення від HyperText Markup Language («мова розмітки гіпертексту»), є мовою розмітки для створення веб-сторінок . Це стандарт, який

слугує довідником для програмного забезпечення, яке пов’язує з розробкою веб-сторінок у різних його версіях, визначає основну структуру та код (званий HTML-кодом) для визначення вмісту веб-сторінки, наприклад тексту, зображення, відео тощо. HTML вважається найважливішою веб-мовою, і його винахід мав вирішальне значення у появі, розвитку та розширенні всесвітньої мережі [8].

Мова HTML базує свою філософію розвитку на диференціації. Щоб додати зовнішній елемент на сторінку (зображення, відео, сценарій тощо), він не вбудовується безпосередньо в код сторінки, а посилання на розташування зазначеного елемента здійснюється за допомогою тексту. Таким чином, веб-сторінка містить лише текст, а на веб-браузер (інтерпретатор коду) покладається завдання об’єднання всіх елементів і відображення кінцевої сторінки. Будучи стандартом, HTML прагне бути мовою, яка дозволяє будь-якій веб-сторінці, написаній у певній версії, інтерпретуватися однаково (стандартно) будь-яким оновленим веб-браузером.

HTML – це мова розмітки, яка дозволяє нам вказувати структуру нашого документа за допомогою тегів. Ця мова пропонує нам чудову адаптивність та логічну структуру [8].

У різних її його версіях були включені та вилучені різні функції, щоб зробити її більш ефективною і полегшити розробку веб-сторінок, сумісних з різними браузерами та платформами (ПК, ноутбуки, смартфони , планшети, тощо.) Однак, щоб правильно інтерпретувати нову версію HTML, розробники веб-браузерів повинні включити ці зміни, а користувач повинен мати можливість використовувати нову версію браузера з внесеними змінами. Іноді зустрічаються ситуації, коли застарілий браузер не може правильно інтерпретувати веб-сторінку, написану у новішій версії HTML

3.1.3 CSS

CSS розшифровується як каскадні таблиці стилів. Ця мова надає веб-сайту потрыбний вигляд. Поряд із HTML, CSS є основою для розробки веб-сторінок. Без нього веб-сайти були б звичайним текстом на білому тлі.

До розробки CSS у 1996 році Консорціумом World Wide Web Consortium, веб-сторінки були надзвичайно обмеженими як за формою, так і за функціями. Ранні браузери представляли сторінку як гіпертекст - звичайний текст, зображення та посилання на інші гіпертекстові сторінки [7].

CSS здіснив кілька нововведень у макети веб-сторінок, наприклад, можливість:

- Вказувати шрифти, відмінні від стандартних для браузера
- Вказувати колір і розмір тексту та посилань
- Застосовувати кольори до фону
- Поміщати елементи веб-сторінки в блоки і переміщувати ці блоки на певні позиції на сторінці

Першим комерційним браузером, який читав і використовував CSS, був Internet Explorer 3 від Microsoft у 1998 році. До цього дня підтримка деяких функцій CSS залежить від браузера. W3C, який досі контролює і створює веб-стандарти, нещодавно випустив новий стандарт для CSS - CSS3. З CSS3 розробники сподіваються, що всі основні браузери читатимуть і відображатимуть кожну функцію CSS однаково.

3.1.4 JavaScript

JavaScript – це мова програмування, яка виконується в браузері. Він перетворює статичні веб-сторінки HTML на інтерактивні веб-сторінки шляхом динамічного оновлення вмісту, перевірки даних форм, керування мультимедіа, анімованих зображень і майже всього іншого на веб-сторінках [5].

JavaScript є третьою за важливістю веб-технологією після HTML і CSS. JavaScript можна використовувати для створення веб та мобільних додатків, створення веб-серверів, створення ігор тощо.

JavaScript можна використовувати в різних видах діяльності, як-от перевірка даних, відображення спливаючих повідомлень, обробка подій елементів HTML, зміна CSS тощо.

У світі немає жодного веб-сайту, який би не використовував JavaScript або фреймворки на основі JavaScript.

На початку 1995 року Брендан Айх з Netscape розробив і впровадив нову мову для програмістів, щоб надати нещодавно додану підтримку Java в браузер Netscape. Спочатку він називався Mocha, потім LiveScript і, нарешті, JavaScript [5].

Зараз JavaScript може виконуватися не тільки в браузерах, а й на сервері або будь-якому пристрої з JavaScript Engine. Наприклад, Node.js — це фреймворк на основі JavaScript, який виконується на сервері.

3.1.5 React

React зарекомендував себе як найбільш використовуваний інтерфейсний фреймворк/бібліотека JS у світі, оскільки він популярний як у розробників програмного забезпечення, так і у спонсорів проектів [15].

Мінімальним кваліфікаційним критерієм є або має бути те, що даний фреймворк або бібліотека відповідають технічним характеристикам програми. В ідеалі вибір буде гнучким відповідно до потреб широкого спектру потенційних специфікацій. Це дозволить одній і тій же команді працювати над різними заявками роботодавця [15].

Існують інші причини, чому React можна вважати технологією, яка є особливо перспективною. Бібліотеку підтримує одна з найбільших компаній у світі Facebook. Він також має, частково через велику кількість розробників React, особливо активну спільноту з відкритим кодом. Поєднання підтримки Facebook і великої та залученої спільноти означає, що React постійно

оновлюється та вдосконалюється, і в осяжному майбутньому не зможе застаріти.

3.1.6 Redux

Redux представля форму побудови архітектури програми на React. Redux є контейнером для управління станом програми і багато в чому нагадує Flux. Redux не прив'язаний безпосередньо до React.js і може використовуватися з іншими js-бібліотеками та фреймворками [16].

Ключові моменти Redux:

- Сховище (store): зберігає стан програми
- Дії (actions): деякий набір інформації, що походить від додатка до сховища і який вказує, що саме потрібно зробити. Для передачі інформації у сховища викликається метод dispatch().
- Творці дій (action creators): функції, що створюють дії
- Reducer : функція (або кілька функцій), яка отримує дію і відповідно до цієї дії змінює стан сховища

3.1.7 Canvas

Тег Canvas в HTML використовується для плавного малювання та створення графіки. Він функціонує як контейнер у документі HTML, який містить намальовану графіку.

Для малювання такої графіки можна використовувати таку мову сценаріїв, як JavaScript.

Тег canvas сам по собі є прозорим, тобто він не створює графіку самостійно. Елемент canvas використовується для малювання «растрової» графіки у веб-додатках. Canvas API надає два контексти малювання: 2D і 3D.

Теги Canvas не мають вбудованих налаштувань або сцен. Це процедурний низькорівневий тег для растрового зображення (пікселізації зображень у сітки). Теги Canvas є додатковою функцією в HTML5. Тому старіші версії HTML їх не підтримують. Тег Canvas — це тег-контейнер. Це означає, що для нього потрібні як відкриваючі, так і закриваючі теги [10].

3.2 Моделювання Web-додатку для редагування діаграм UML

Проектування будь-яких обє'ктів розпочинається із формування концептуальних уявлень про роботу об'єкту. Для представлення цих уявлень і запропонована саме мова UML. Результатом початкового етапу проектування \ файл, у якому зберігається опис цієї моделі. Проектанти надають проектні пропозиції, із яких створюється опис моделі. Прицьому необхідно дотримуватися методик проектування та витримувати вимоги стандартів, які діють у предметній області. Графічне оформлення результатів та контроль коректності застосування такого підходу є стправою кропіткою та трудоємною і саме тому необхідні інструментальні засоби, які підримують концептуальне моделювання. Вказаний контекст аналізовано етапу проектування відображає рисунок 3.2.1.

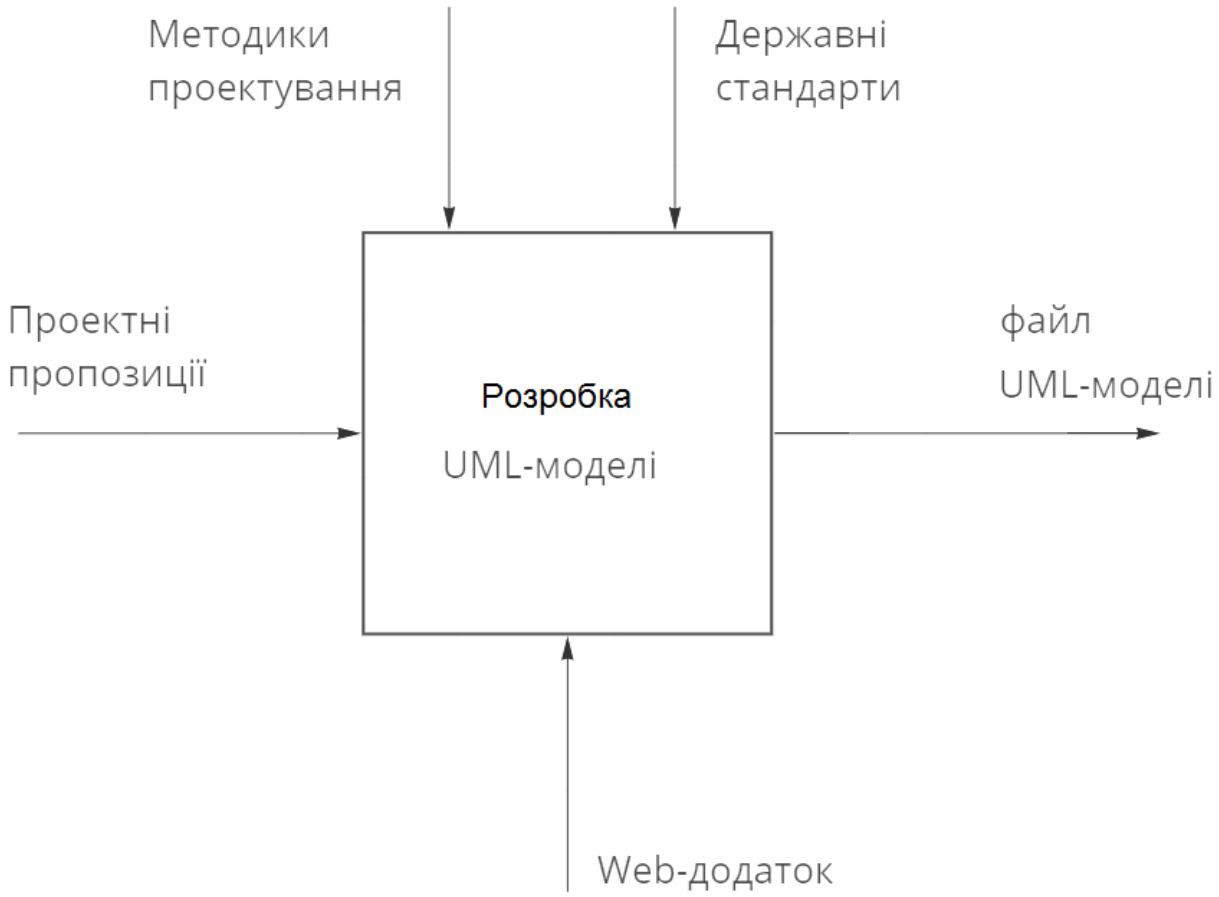


Рисунок 3.2.1 - Контекстна діаграма застосування web-додатку для редагування діаграм UML

Розроблюваний web- додаток редагування діаграм UML повинен підтримувати цей контекст: створювати файл UML- діаграм на основі проектних пропозицій користувачів, підтримувати вимоги методик проектування та стандартів. Найбільш сприятливим, доступним та функціональним середовищем його функціонування є web-браузер. Зрозуміло, що браузер «не знає» усіх думок проектантів та шляхів їх розвитку, тому web-додаток використовує потенціальні можливості проектанта, як це показано на рисунку 3.2.2.

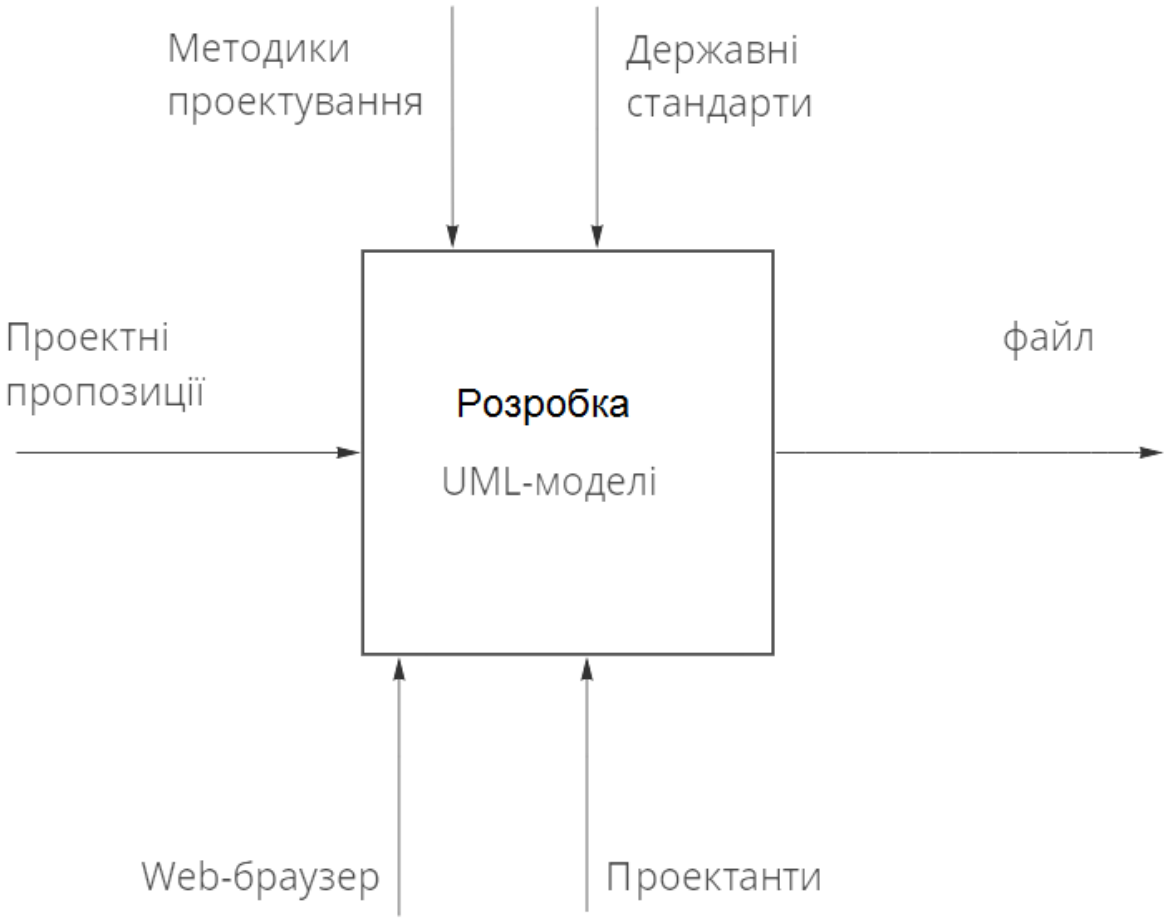


Рисунок 3.2.2 - Функціональна діаграма web-додатку для редагування діаграм UML

Діаграма на цьому рисунку дає загальне уявлення про роботу web-додатку. Виконувані ним функції отримуємо шляхом декомпозиції функції «Розробка UML-моделі» на функції:

- вибирання UML- елементу, який відповідає проектному задуму,
- встановлення звязків між уже введеними елементами та поточним,
- призначення параметрів елементам,
- формування файлу моделі.

Вказані функції показано на рисунку 3.2.3. При цьому функції/ процедури читання поточного стану моделі та запису результатів не показані, оскільки являються тривіальними і використовуваним за замовчуванням.

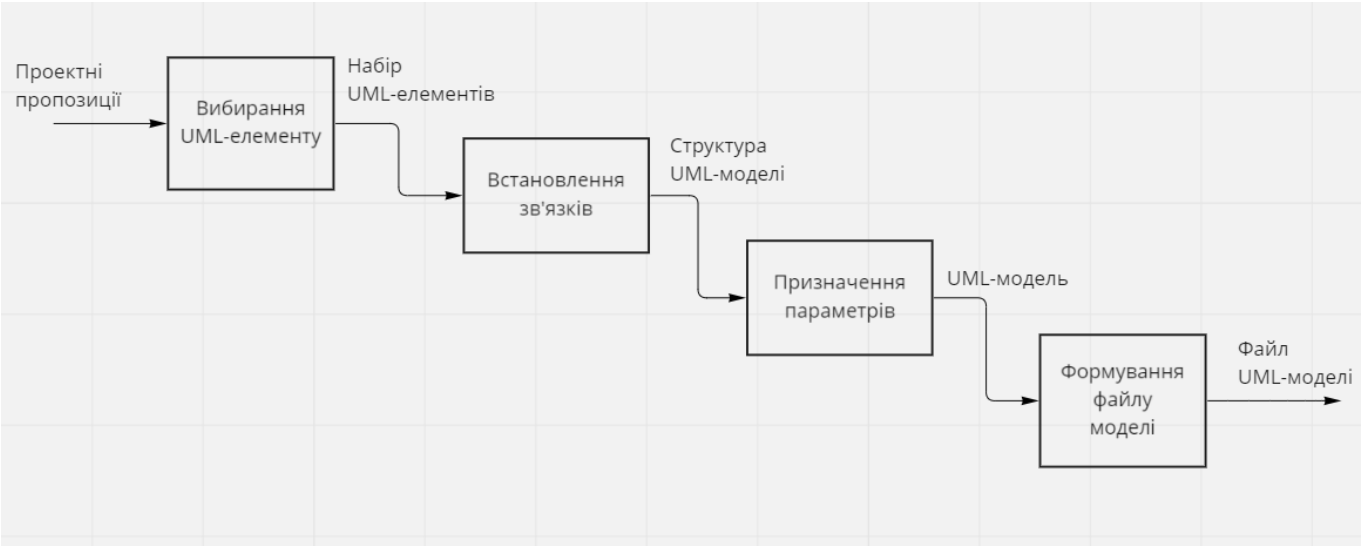


Рисунок 3.2.3 - Діаграма декомпозиції функції web- додатку для редагування діаграм UML

Web- додаток передбачається використовувати у складі системи проектування більше високого рівня, у якій кожен проектувальник є клієнтом цієї системи і має відповідні права доступу. Їх перевірка передбачена, але поки що відключена.

Налаштування прав доступу виконує адміністратор системи, результати налаштування зберігаються у базі даних, яка є для нашого додатку зовнішньою.

Формування ка коригування UML- моделі здійснюється клієнтом.

Керівник має можливості коригування моделі, а також виконувати її кортроль та затвердження.

Форомування файлу моделі здійснюється автоматично буз участі людей. При цьому формуються також мета дані, які зберігаються у базі даних. Файл моделі зберігається у сховищі даних (репозиторії).

Відмічені елементи роботи функції web- додатку для редагування діаграм UML відображені на рисунку 3.2.4.

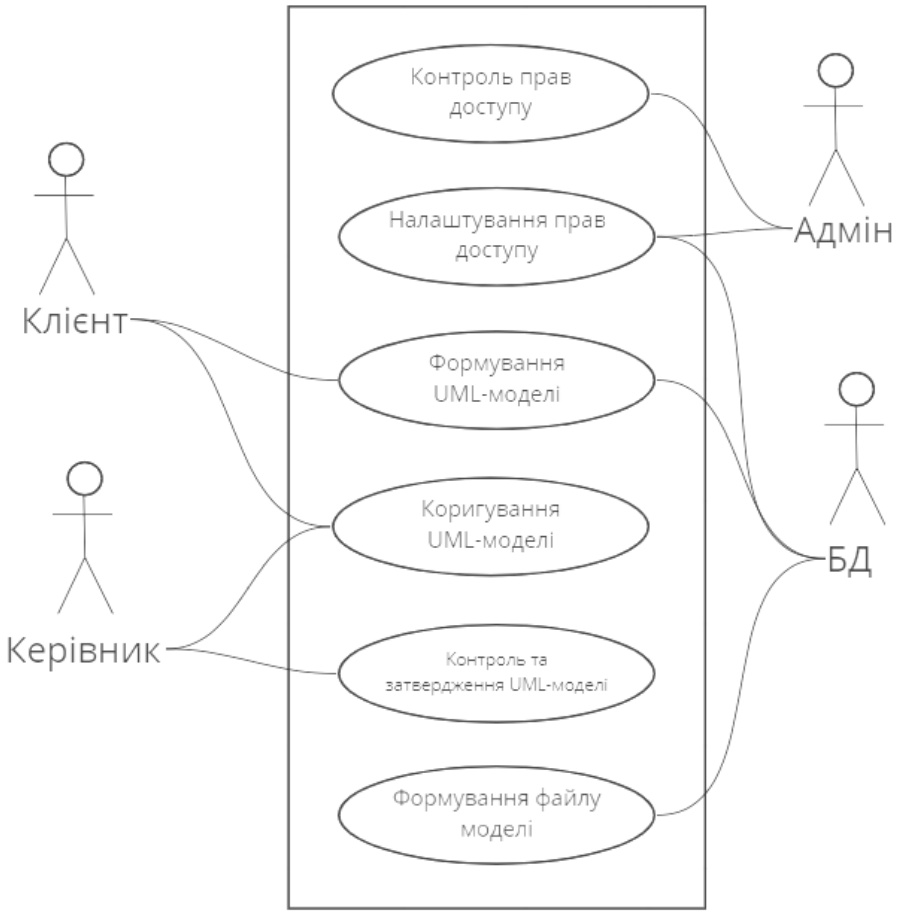


Рисунок 3.2.4 - Діаграма варіантів використання web-додатку для редагування діаграм UML

Тепер є можливість зосередитися на питаннях практичної реалізації розробки web-додатку.

3.2 Загальне представлення обьектів.

Додаток скаладається з двох основних основних компонентів – Робочого поля та випадаючої Панелі Інструментів.

Всі об’єкти на що розташовані на Робочому полі є окреміми React-компонентами. Компоненти є основним будівельним блоком React-додатку. Їх використання значно завдання створення елементів інтерфейсу. Вони

працюють незалежно один від одного і об’єднуються в батьківський компонент, який буде кінцевим інтерфейсом програми. Саме поле є canvas-елементом, що дозволяє малювати на ньому довільні малюнки.

Всі елементи на полі позиціонуються на полі за допомогою css-властивості position: absolute. Абсолютне позиціонування робить дві речі: Елемент зникає з того місця, де він має бути і знову позиціонується. Інші елементи розташовуються так, ніби цього елемента ніколи не було. Координати та інші дані кожного обьекта записуються в стейт, реалізований за допомогою бібліотеки Redux, а також зберігаються в local storage – сховище, яке дозволяє сайтам і додаткам JavaScript зберігати пари ключ-значення у веб-переглядачі без терміну дії. Це означає, що дані, збережені в браузері, зберігатимуться навіть після закриття вікна браузера.

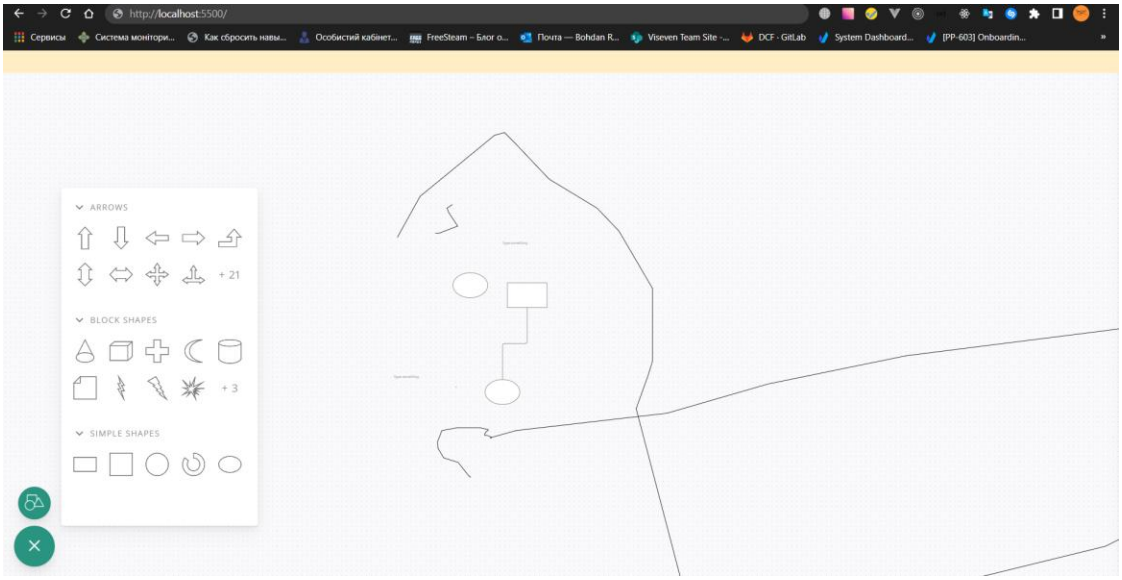


Рисунок 3.2.1 - Вигляд додатку

3.3. Принцип роботи Робочого поля.

Робоче поле представлене у вигляді блока з css-властивістю position: relative, для того щоб елементи якій розташовувались на ньому позиціонувалсь відносно поля, а не відносно всієї сторінки.

У Робочого поля реалізовиний механіз зміни маштабу за допомогою css-властивості transform-style. Властивість transform-style дозволяє перетвореним 3D-елементам та їх 3D-нащадкам використовувати загальний тривимірний простір, вибудовуючи ієрархії тривимірних об'єктів. Відображення 3D-нащадків визначається моделлю - так званим контекстом 3D-рендерінгу. Відображення залежить від z-позиції елементів у тривимірному просторі, і якщо 3D-перетворення цих елементів викликають перетин, вони відображаються з перетином. Властивість встановлюється батьківського елемента. За допомогю цього механізму реалізована можливість переміщення по ньому. За допомогою JavaScript зчитуться координати миші і весь блок зміщуеться відповідно них.

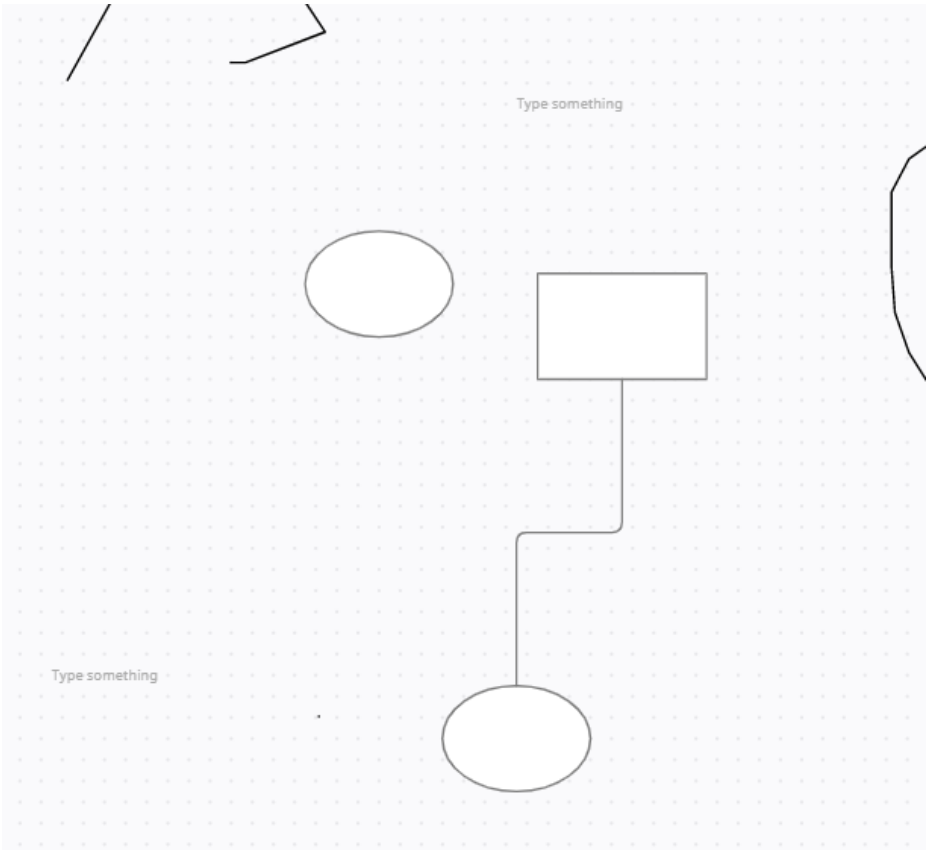
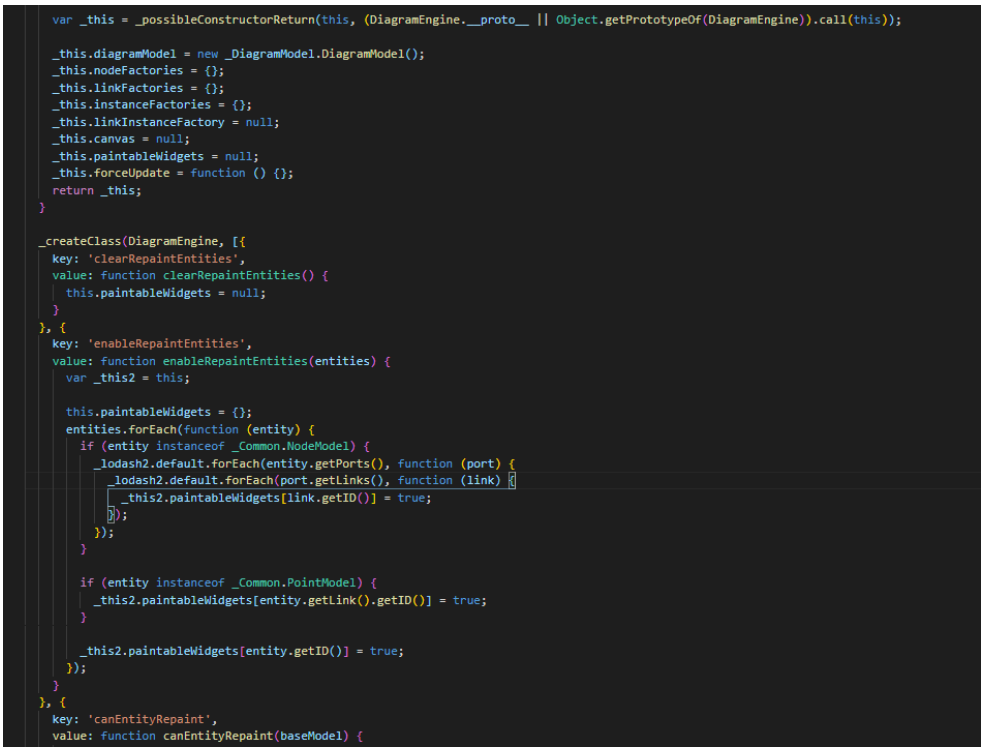


Рисунок 3.3.1 - Робоче поле



```
  var _this = _possibleConstructorReturn(this, (DiagramEngine.__proto__ || Object.getPrototypeOf(DiagramEngine)).call(this));

  _this.diagramModel = new _DiagramModel.DiagramModel();
  _this.nodeFactories = {};
  _this.linkFactories = {};
  _this.instanceFactories = {};
  _this.linkInstanceFactory = null;
  _this.canvas = null;
  _this.paintableWidgets = null;
  _this.forceUpdate = function () {};
  return _this;
}

_createClass(DiagramEngine, [{
  key: 'clearRepaintEntities',
  value: function clearRepaintEntities() {
    this.paintableWidgets = null;
  }
}, {
  key: 'enableRepaintEntities',
  value: function enableRepaintEntities(entities) {
    var _this2 = this;

    this.paintableWidgets = {};
    entities.forEach(function (entity) {
      if (entity instanceof _Common.NodeModel) {
        _lodash2.default.forEach(entity.getPorts(), function (port) {
          _lodash2.default.forEach(port.getLinks(), function (link) {
            _this2.paintableWidgets[link.getID()] = true;
          });
        });
      }

      if (entity instanceof _Common.PointModel) {
        _this2.paintableWidgets[entity.getLink().getID()] = true;
      }

      _this2.paintableWidgets[entity.getID()] = true;
    });
  }
}, {
  key: 'canEntityRepaint',
  value: function canEntityRepaint(baseModel) {
```

Рисунок 3.3.2 - Код реалізації Робочого поля

3.4 Принцип роботи Панелі інструментів.

Панель інструментів виконана у вигляді випадаючого списку, що рендериться при натисканні на відповдну кнопу. В списку представлений набір елементів кожен з яких можна перенести на робоче поле, навівши миш на відповідну фігуру і з зажатою лівою клавішею, перетягнути її у бажане місце. Всі елементи у списку мають унікальний id, при віборі якоїсь фігури, додаток шукає відповідний для нього об’єкт, який використовується для пренесення на Робоче поле.

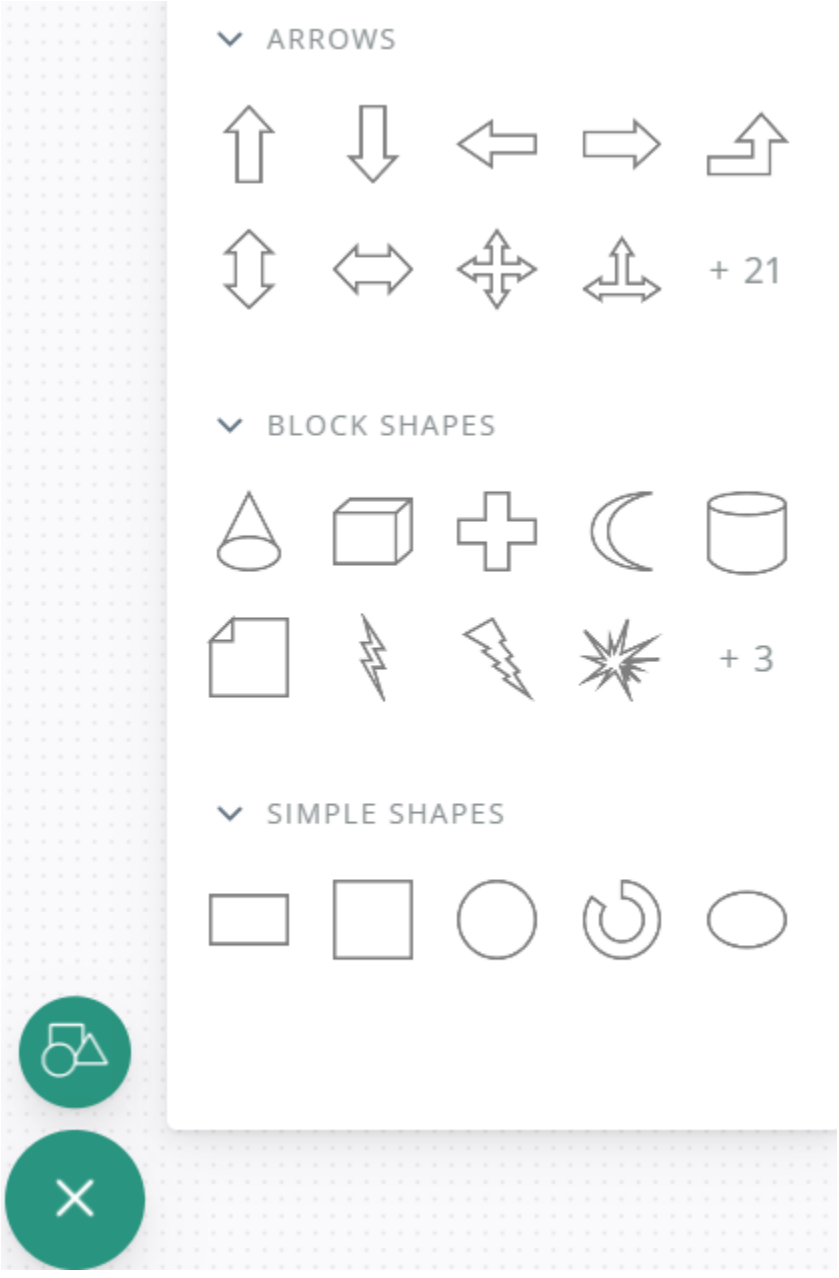


Рисунок 3.4.1 - Набір інструментів у розгорнутому стані

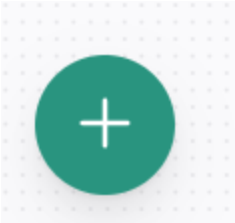


Рисунок 3.4.2 - Кнопка для відкриття Набору інструментів

```
var BaseModel = function (_BaseEntity) {
  _inherits(BaseModel, _BaseEntity);

  function BaseModel() {
    _classCallCheck(this, BaseModel);

    var _this = _possibleConstructorReturn(this, (BaseModel.__proto__ || Object.getPrototypeOf(BaseModel)).call(this));

    _this.selected = false;
    return _this;
  }

  _createClass(BaseModel, [{
    key: 'deSerialize',
    value: function deSerialize(ob) {
      _get(BaseModel.prototype.__proto__ || Object.getPrototypeOf(BaseModel.prototype), 'deSerialize', this).call(this, ob);
      this.selected = ob.selected;
    }
  }, {
    key: 'serialize',
    value: function serialize() {
      return _extends({}, _get(BaseModel.prototype.__proto__ || Object.getPrototypeOf(BaseModel.prototype), 'serialize', this).call(this), {
        _class: this.constructor.name,
        selected: this.selected
      });
    }
  }, {
    key: 'getID',
    value: function getID() {
      return this.id;
    }
  }, {
    key: 'isSelected',
    value: function isSelected() {
      return this.selected;
    }
  }, {
    key: 'setSelected',
    value: function setSelected(selected) {
      var _this2 = this;

      this.selected = selected;
      this.itterateListeners(function (listener) {
        if (listener.selectionChanged) {
          listener.selectionChanged(_this2, selected);
        }
      });
    }
  }, {
    key: 'remove',
```

Рисунок 3.4.3 - Код реалізації Панелі інструментів

3.5 Реалізація геометричних фігур

Всі геоиетричні фігури є окремими компонентами зі своїм набором. При при виборі якоїсь фігури зі списку фігур у наборі інструментів, додаток шукає відповідний для нього об’єкт, що являє собою інший компонет із тим самим зображенням, але уже призначеним для відображення на Робочому полі. Зображення фігур виконано у форматі svg. Координати фігури зберігаються у стейті додатку. При зміні координат, викликається метод що змінює

координати на інші. Зміна координат виконана зі зберігання принципів Redux, тобто коли користувач перетягує якийсь елемент, зчитується положення миші, потім ці дані переносяться в стейт, і уже потім, зі стейту, вносяться в властивості положення самого елементу.

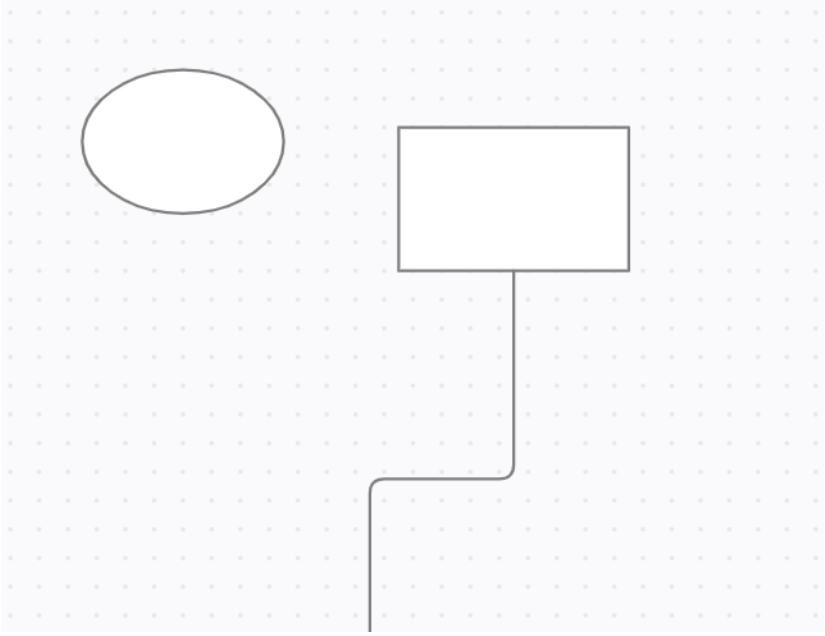


Рисунок 3.5.1 - Геометричні фігури на Робочому полі

```
var DiagramModel = exports.DiagramModel = function (_BaseEntity) {
  _inherits(DiagramModel, _BaseEntity);

  function DiagramModel() {
    _classCallCheck(this, DiagramModel);

    var _this = _possibleConstructorReturn(this, (DiagramModel.__proto__ || Object.getPrototypeOf(DiagramModel)).call(this));

    _this.links = {};
    _this.nodes = {};
    _this.offsetX = 0;
    _this.offsetY = 0;
    _this.zoom = 100;
    _this.rendered = false;
    return _this;
  }

  _createClass(DiagramModel, [{
    key: 'deSerializeDiagram',
    value: function deSerializeDiagram(object, diagramEngine) {
      var _this2 = this;

      this.deSerialize(object);
      this.offsetX = object.offsetX;
      this.offsetY = object.offsetY;
      this.zoom = object.zoom;


      _lodash2.default.forEach(object.nodes, function (node) {
        var nodeOb = diagramEngine.getInstanceFactory(node._class).getInstance();
        nodeOb.deSerialize(node);


        _lodash2.default.forEach(node.ports, function (port) {
          var portOb = diagramEngine.getInstanceFactory(port._class).getInstance();
          portOb.deSerialize(port);
          nodeOb.addPort(portOb);
        });

        _this2.addNode(nodeOb);
      });


      _lodash2.default.forEach(object.links, function (link) {
        var linkOb = diagramEngine.getInstanceFactory(link._class).getInstance();
        linkOb.deSerialize(link);

        if (link.target) {
          linkOb.setTargetPort(_this2.getNode(link.target).getPortFromID(link.targetPort));
        }
```

Рисунок 3.5.2 - Код реалізації геометричних фігур

3.6 Реалізація додаткового інтерфейсу

У кожного елементу що знаходиться на робочому полі є набір додаткових функцій, що дозволяють створити новий елемент, приєднаний лінію до того, від якого він створюється, змінити маштаб, кут нахилу або форму елемена. При натисканні на будь-який елемент, у нього відобважається набів кнопок, кожна з яких відповідає за одну з вище перечислених дій.

При додаванні елементу, у нього викликається той самий метод що рендерить елемент, при переносі його з Панелі інструментів, а також метод що відповідає за об'єднання елементів лініями.

Зміна форми елеменів, а точніше зміна пропорцій висоти і ширини реаізована за допомогою зміни css-властивості transform: scaleX() і transform: scaleY(). Зміна координат виконана зі зберігання принципів Redux, тобто коли користувач змінює пропорції елементу, дані переносяться в стейт, і уже потім, зі стейту, вносяться в властивості положення самого елементу. Зміна кута нахилу реалізована по такому самоу принципу, але уже з використанням властвості transform: rotate замість transform: scale.

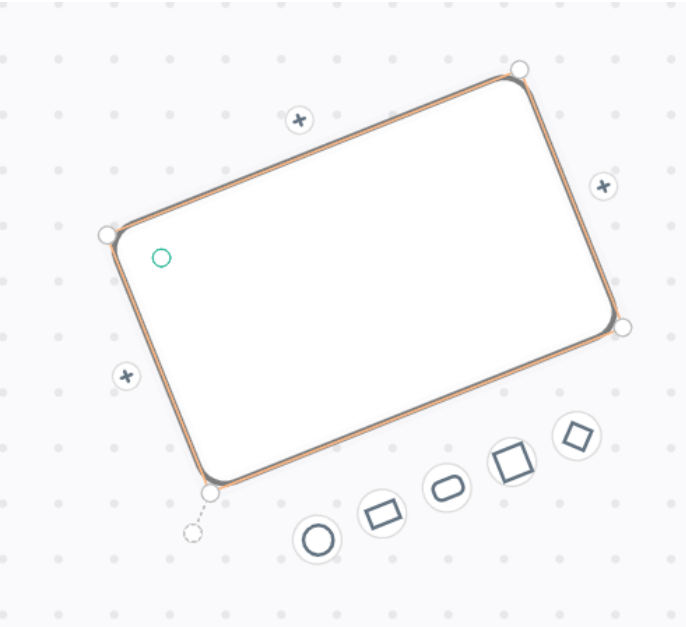


Рисунок 3.6.1 - Додатковий інтерфейс

```
      this.itterateListeners(function (listener) {
        if (listener.entityRemoved) {
          listener.entityRemoved(_this3);
        }
      });
    }
  }]);

  return BaseModel;
}(_BaseEntity2.BaseEntity);

var PointModel = exports.PointModel = function (_BaseModel) {
  _inherits(PointModel, _BaseModel);

  function PointModel(link, points) {
    _classCallCheck(this, PointModel);

    var _this4 = _possibleConstructorReturn(this, (PointModel.__proto__ || Object.getPrototypeOf(PointModel)).call(this));

    _this4.x = points.x;
    _this4.y = points.y;
    _this4.link = link;
    return _this4;
  }

  _createClass(PointModel, [{
    key: 'deSerialize',
    value: function deSerialize(ob) {
      _get(PointModel.prototype.__proto__ || Object.getPrototypeOf(PointModel.prototype), 'deSerialize', this).call(this, ob);
      this.x = ob.x;
      this.y = ob.y;
    }
  }, {
    key: 'serialize',
    value: function serialize() {
      return _extends({}, _get(PointModel.prototype.__proto__ || Object.getPrototypeOf(PointModel.prototype), 'serialize', this).call(this), {
        x: this.x,
        y: this.y
      });
    }
  }, {
    key: 'remove',
    value: function remove() {
      _get(PointModel.prototype.__proto__ || Object.getPrototypeOf(PointModel.prototype), 'remove', this).call(this);
```

Рисунок 3.6.2 - Код реалізації додаткового інтерфейсу

3.7 Реалізація ліній

Лінії з’їднууть 2 об’єкти на Робочому полі. Вони виконані у вигляді окремого інтрументу, а також амтоматично спрацьовують, при створенні нової фігури за домпомогою додаткового інтерфейсу. При використанні інструменту «Лінія», користувач вказує координати початку малювання, клікнувши на один із об’єктів на полі, в стейт заноситься ідентифікатор обьекту, до якого приєднана лінія, а також координати конкретного місця на самому об’єкті. Коли перша точка задана, додаток починає малювати лінію до другої точки, якою поки що виступає миш. Коли користувач вказує другу фігуру, в стейт вносяться відповідні дані і лінія прикріплюється до об’єкта. При створенні нового об’єкту за допомогою додаткового інтерфейсу, інструмент лінія спрацьовує автоматично. Маючи координати обох точок, додаток малює лінію, а при переміщенні обдного з об’єктів – додаток її перемальовує.

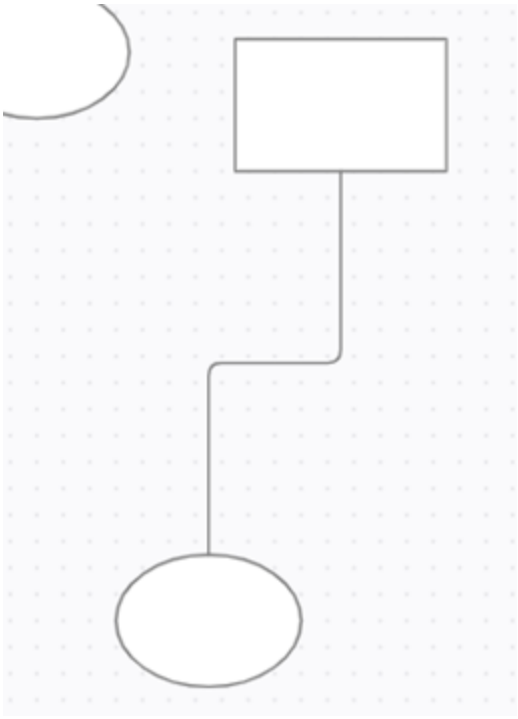


Рисунок 3.7.1 - Лінія

```
}

function drainQueue() {
    if (draining) {
        return;
    }
    var timeout = runTimeout(cleanUpNextTick);
    draining = true;

    var len = queue.length;
    while(len) {
        currentQueue = queue;
        queue = [];
        while (++queueIndex < len) {
            if (currentQueue) {
                currentQueue[queueIndex].run();
            }
        }
        queueIndex = -1;
        len = queue.length;
    }
    currentQueue = null;
    draining = false;
    runClearTimeout(timeout);
}

process.nextTick = function (fun) {
    var args = new Array(arguments.length - 1);
    if (arguments.length > 1) {
        for (var i = 1; i < arguments.length; i++) {
            args[i - 1] = arguments[i];
        }
    }
    queue.push(new Item(fun, args));
    if (queue.length === 1 && !draining) {
        runTimeout(drainQueue);
    }
};

function Item(fun, array) {
    this.fun = fun;
    this.array = array;
}
Item.prototype.run = function () {
    this.fun.apply(null, this.array);
};
process.title = 'browser';
process.browser = true;
process.env = {};
process.argv = [];
process.version = '';
process.versions = {};
```

Рисунок 3.7.2 - Код реалізації лінії

ВИСНОВКИ

При виконанні даної роботи, була проаналізована предметна область, в томі числі питання UML-діаграм, їх види, а також основні способи їх використання. Були розглянуті основні рішення на ринку, а також їх переваги та недоліки. Поставлені конкретні вимоги до написання інформаційної системи.

Було сформульовано технічне завдання, поставленні цілі виконання даної роботи та сформульована цільова аудиторія, яку може зацікавити даний додаток, поставлені вимоги до програмного забезпечення. Були вибрані інструменти розробки, а також було виконано обґрунтування використання. Також був описаний процес розробки і основні принципи роботи всіх аспектів додатку. Всі цілі були досягнуті і виконані.

Подальшим напрямком удосконалення розробленого web-додатку може бути його розвиток у напрямку інтеграції із введенням параметрів використовуваних об’єктів та розрахунку їх характеристик та характеристик модельованої системи загалом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / Мартин Фаулер. 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с.
2. Шмуллер Д. Освой самостоятельно UML за 24 часа./ Джозеф Шмуллер 3-е изд. — М.: Вильямс, 2005. — 401 с.
3. Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd Edition. – Pearson, 2004. – 736 Pages.
4. Джеймс Рамбо, Грейді Буч, Айвар Джекобсон 2020 «UML. Посібник користувача» [Електронний ресурс] – Режим доступу до ресурсу: https://kaf401.rloc.ru/TRPO/UMLBooch/UMLBoochContent.htm Дата доступу 03.04.2022.
5. Опис мови програмування JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript. Дата доступу 03.04.2022.
6. UML – Короткий посібник [Електронний ресурс] – Режим доступу до ресурсу: https://coderlessons.com/tutorials/akademicheskii/uchit-uml/uml-kratkoe-rukovodstvo Дата доступу 03.04.2022.
7. Опис мови програмування CSS[Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/CSS Дата доступу 03.04.2022.
8. Опис мови програмування HML [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/HTML Дата доступу 03.04.2022.
9. Types of UML Diagrams [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/blog/types-of-UML-diagrams
10. Опис елемента Canvas [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Canvas Дата доступу 03.04.2022.

11. Топ-19 сервісів для створення блок-схем та UML [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/top-servisov-uml/ Дата доступу 03.04.2022.
12. Опис редактора VS Code [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Visual_Studio_Code Дата доступу 03.04.2022.
13. UML models and diagrams [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-uml-models Дата доступу 03.04.2022.
14. Опис менеджера Redux [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Redux Дата доступу 03.04.2022.
15. Опис фреймворку React.js [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/React Дата доступу 03.04.2022.
16. Документація менеджера Redux [Електронний ресурс] – Режим доступу до ресурсу: https://react-redux.js.org/ Дата доступу 03.04.2022.
17. Навіщо нам UML? Або як зберегти собі нерви та час [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/458680/
18. Позиціонування в CSS [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/position Дата доступу 03.04.2022.
19. Властивість transform-style [Електронний ресурс] – Режим доступу до ресурсу: https://html5book.ru/3d-transform/ Дата доступу 03.04.2022.
20. Документація фреймворку React.js [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/docs/components-and-props.html Дата доступу 03.04.2022.

ДОДАТКИ

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Технічне завдання
на створення програмного продукту

«Розробка web-додаток для редагування діаграм UML»

2022

1. Призначення й мета створення web-додатку

1.1 Призначення web-додатку

Додаток повинен надавати функціонал створення та редагування діаграм UML та блок-схем.

1.2 Мета створення web-додатку

Метою даного дослідження є розробка web-додатку для роботи з UML-діаграмами

1.3 Цільова аудиторія

У цільовій аудиторії інформаційної системи можна виділити наступні групи:

6. Програмісти.
7. Менеджери.
8. Архітектори.
9. Дизайнери.
10. Особи що вивчають програмування.

2. Вимоги до web-додатку

2.1 Вимоги до структури й функціонування web-додатку

Додаток повинен буди представлений у вигляді сайду, що складається з двох частин, списку з інструментами та робочим полем, на якому можна створювати UML-діаграми та редагувати їх. Користувач повинен мати змогу створювати повноцінні UML-діаграми в робочому полі за допомогою інструментів, що розташовані в окремій вкладці. Інтерфейс повинен давати можливість повноцінно їх редагувати, в тому числі вільно переміщувати схеми по робочому полю, як всі разом так різні елементи окремо, додавати та видаляти елементи, наносити текст на схеми та з’єднувати їх між собою, мати просту навігацію та бути інтуїтивно зрозумілим

2.2 Структура web-додатку

2.2.1 Панель інструментів

Панель інтрументів являє собою список геометричних фігур, кожну з яких можна перенести на робоче поле, навівши миш на відповідну фігуру і з зажатою лівою клавішею, перетягнути її у бажане місце. Також в панелі присутні такі інструменти як «Текст», який дозволяє додавати текстові поля на робоче поле, інструмент «Вільне малювання», що дає змогу малювати на робочому полі довільні зображення, а також інструмент «Лінія», за допомогою якої можна з’єднувати розташовані геометричні фігури між собою.

2.2.2 Робоче поле

Робоче поле являє собою частину інтерфейсу додатку на якій можна безпосередньо створювати діаграми, переміщуючи їх з Панелі інструментів. На ньому можна розміщати геометричні фігури, текст, тощо, вільно переміщувати їх по полю, змінювати їх розмір та кут нахилу. На Робочому полі розміщувати текст, як всередині, так і поза межами фігур, а також додавати вільні малюнки. Користувач повинен мати змогу вільні переміщуватись по Робочому полю, змінюючи його масштаб та положення. Також повинна бути реалізована функція виділення однієї або декількох об’єктів на полі, а у кожного об’єкта, при потраплянні на поле, мають бути додаткові інтерфейси для більш зручної взаємодії з ними.

2.3 Вимоги до видів забезпечення

2.3.1 Вимоги до інформаційного забезпечення

Реалізація сайту відбувається з використанням:

- HTML
- CSS
- JavaScript
- React.js

2.3.2 Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

– Веб-браузер: Internet Explorer 7.0 і вище, або Firefox 3.5 і вище, або Opera 9.5 і вище, або Safari 3.2.1 і вище, або Chrome 2 і вище;

ДОДАТОК Б

Планування робіт

UML був створений в результаті хаосу навколо розробки програмного забезпечення та документації. У 1990-х роках існувало кілька різних способів представлення та документування програмних систем. Виникла потреба в більш уніфікованому способі візуального представлення цих систем, і в результаті в 1994-1996 роках UML був розроблений трьома інженерами-програмістами, які працювали в Rational Software.

В основному UML використовувався як мова моделювання загального призначення в області розробки програмного забезпечення. Однак тепер вона знайшла своє застосування в документації кількох робочих або бізнес-процесів. Наприклад, діаграми дій, тип діаграм UML, можна використовувати в якості заміни блок-схем. Вони забезпечують як більш стандартизований спосіб моделювання робочих процесів, так і більш широкий набір функцій для підвищення удобочитаємості та ефективності.

Деталізація мети проекту методом SMART. Для збільшення конкурентноздатності та успішності проекту необхідно коректно визначити його мету, що можна зробити за допомогою SMART-методу. Деталізований результат методу зображено в таблиці Б.1.

Таблиця Б.1 – Деталізація мети проекту методом SMART

Specific (конкретна)	Розробити всі компоненты web-додатку та протестувати їх роботу
Measurable (вимірювана)	Розробити визначені компоненти до кінця 4 курсу, використовуючи мінімальний об’єм ресурсів необхідний для виконання завдання.
Achievable (досяжна, узгоджена)	Для розробки проекту потрібні знання HTML, CSS, мови програмування JavaScript і її бібліотек: React та Redux

Продовження таблиці Б.1

Relevant (реалістична)	Створений web-додаток буде надавати функціонал для створення та редагування діаграм UML
Time-framed (обмежена в часі)	Термін досягнення мети проекту визначено за навчальним планом до кінця червня 2022 року.

Планування змісту робіт. Ключовим результатом проекту, що розбиває командну роботу на відслідковувані та керовані частини – це WBS (Work Breakdown Structure – Ієрархічна структура робіт). Іншими словами, це візуальний план компонентів проекту, які згруповано ієрархією. Також WBS надає необхідний вигляд для оцінки термінів та контролю за графіком виконання.

Структурно, на найвищому рівні розміщено кінцевий результат проекту. На другому рівні знаходяться різноманітні дії та заходи для досягнення мети проекту. Таким чином декомпозиція відбувається до моменту, коли дії стають атомарними і мають чіткий та однозначний результат і мають одного виконавця для якого можна розрахувати витрати на працю і часові затрати. На рисунку Б.1 зображено WBS з розробки додатку.

Планування структури виконавців. Наступним кроком після декомпозиції процесів є розробка організаційної структури виконавців або ще OBS, яка зображується як графічна структура, що відображає учасників проекту відповідальних за певний етап реалізації.

До відповідальних осіб відносяться співробітники, що виконують елементарну роботу зазначену в OBS.

На рисунку Б.2 зображено організаційну структуру планування проекту

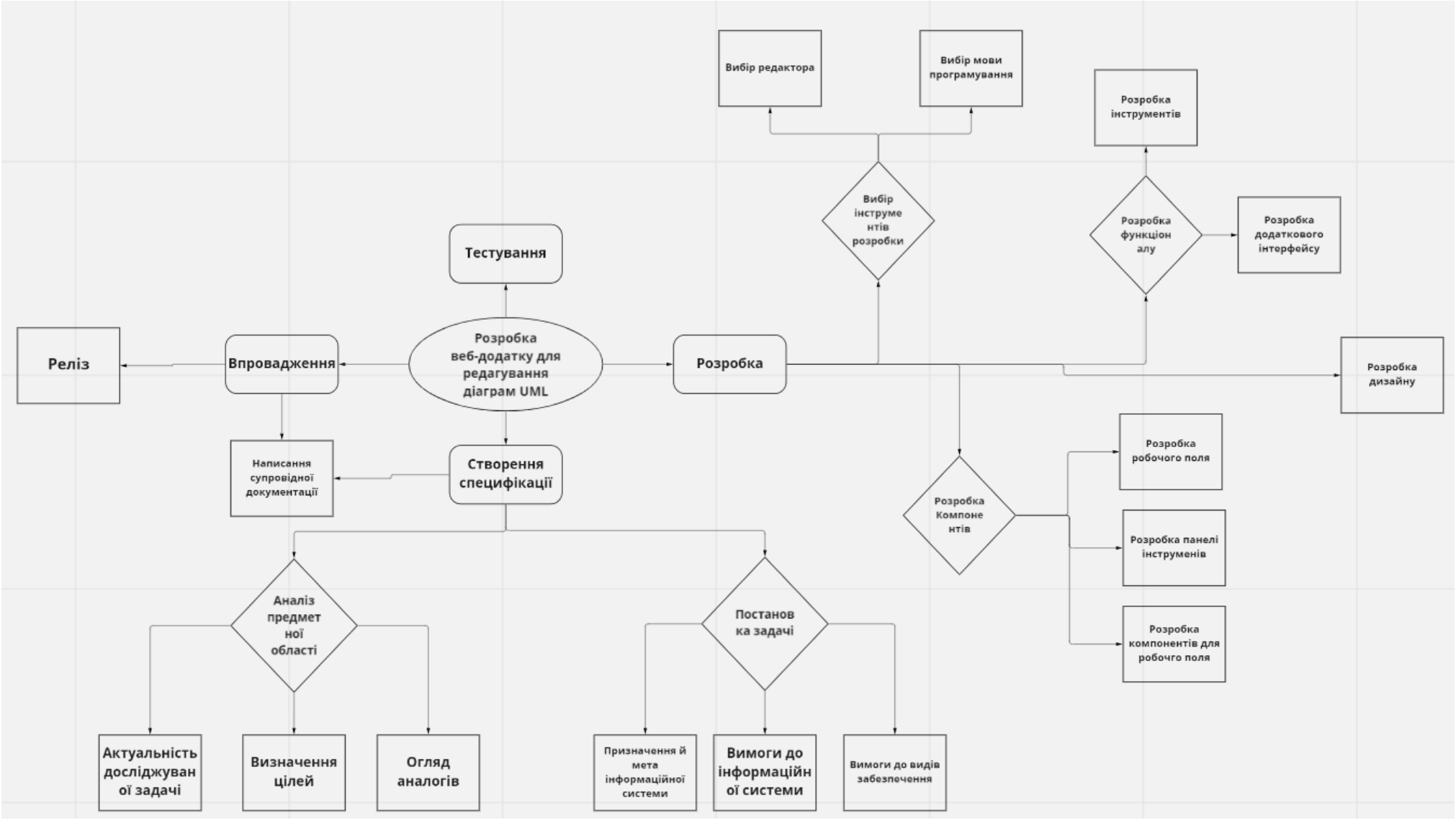


Рисунок Б.1 – WBS-структура робіт проекту

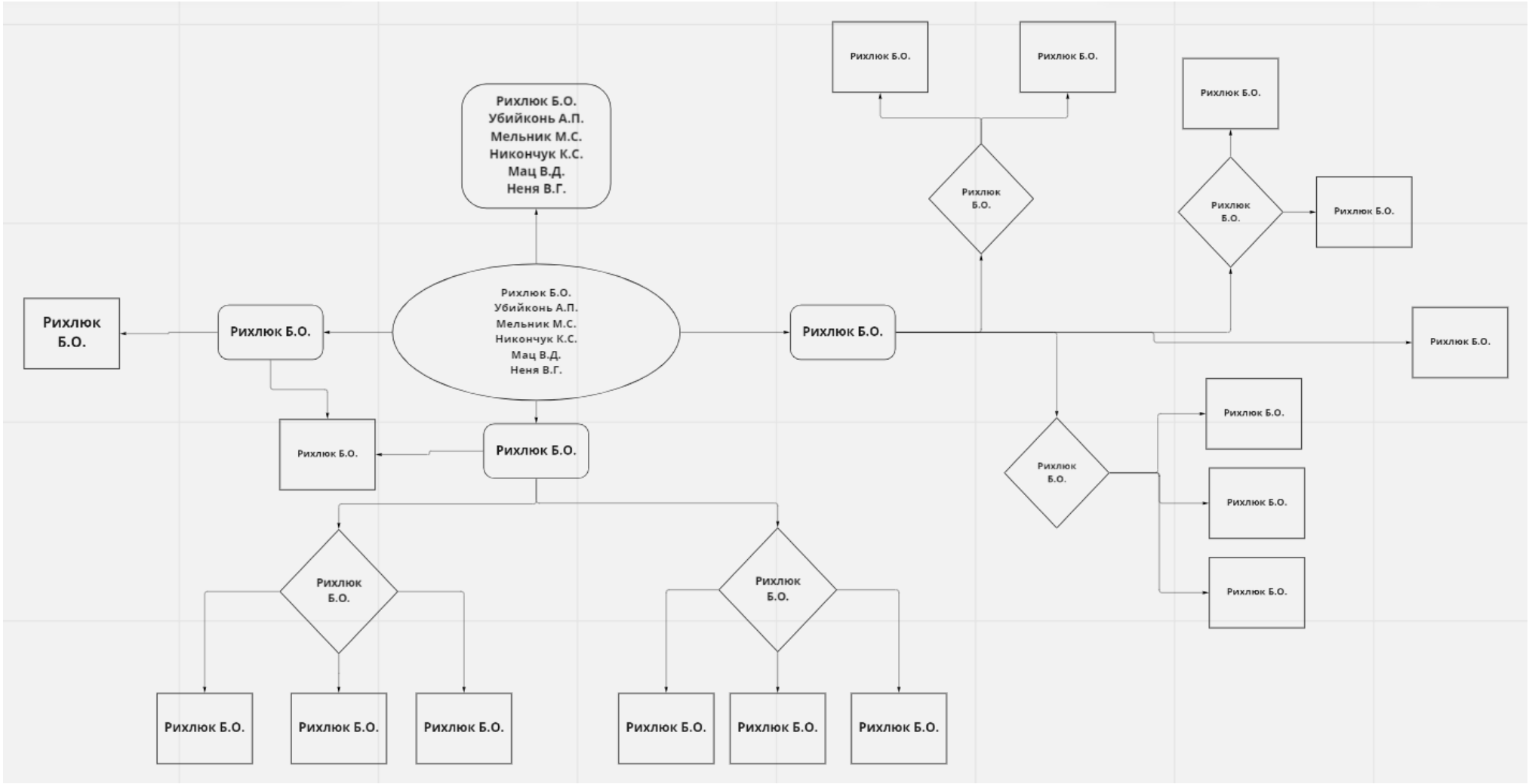


Рисунок Б.2 – OBS-структура робіт проекту

Список виконавців даного проекту описано в таблиці Б.2.

Таблиця Б.2 – Виконавці проекту.

Роль	**Ім'я**	**Проектна роль**
Розробник	Рихлюк Б.О.	Виконує розробку модулів.
Проектувальник	Рихлюк Б.О.	Відповідає за проектування та структуру програмного забезпечення.
Тестувальник	Рихлюк Б.О. Убийконь А.П. Мельник М.С. Никончук К.С. Мац В.Д. Неня В.Г.	Здійснює тестування дизайну та функціональності створених компонентів.
Керівник проекту	Неня В.Г	Видає завдання на розробку проекту.
Менеджер проекту	Рихлюк Б.О.	Відповідальний за розподіл ресурсів та виконання завдань

Діаграма Ганта. Побудова календарного графіку (діаграми Ганта) є одним з найважливіших етапів планування проекту, що виглядає як розбиття виконання робіт з реальним розподілом дат. Завдяки їй можливо отримати достовірне уявлення про тривалість процесів з обмеженнями у ресурсах, урахуванням вихідних днів та свят. Календарний графік проекту представлено на рисунку Б.3.

	Название задачи	Длительность	Начало	Окончание	Предше
1	**Web-додаток для редагування діаграм UML**	**71 дней**	**Пн 10.01.22**	**Пн 18.04.22**	
2	**1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**	**19 дней**	**Пн 10.01.22**	**Чт 03.02.22**	
3	1.1 Актуальність досліджуваної задачі	6 дней	Пн 10.01.22	Пн 17.01.22	
4	1.2 Перелік вимог до інформаційної системи	4 дней	Вт 18.01.22	Пт 21.01.22	3
5	1.3 Аналіз існуючих методів та технологій	3 дней	Пн 24.01.22	Ср 26.01.22	4
6	1.4 Огляд існуючих програмних продуктів для е	8 дней	Чт 27.01.22	Пн 07.02.22	5
7	**2. ПОСТАНОВКА ЗАДАЧІ**	**2 дней**	**Вт 08.02.22**	**Ср 09.02.22**	**6**
8	2.1 Призначення й мета інформаційної системи	2 дней	Вт 08.02.22	Ср 09.02.22	
9	2.2 Вимоги до інформаційної системи	1 день	Ср 09.02.22	Ср 09.02.22	8
10	2.3 Вимоги до видів забезпечення	1 день	Ср 09.02.22	Ср 09.02.22	9
11	**3. ПРОГРАМНА РЕАЛІЗАЦІЯ**	**39 дней**	**Ср 09.02.22**	**Пн 04.04.22**	
12	3.1 Вибір засобіврозробки інформаційної систем	1 день	Ср 09.02.22	Ср 09.02.22	
13	3.2 Загальне представлення обьектів	3 дней	Чт 10.02.22	Пн 14.02.22	12
14	3.3.Реалізація Робочого поля	13 дней	Вт 15.02.22	Чт 03.03.22	13
15	3.4 Реалізація Панелі інструментів	7 дней	Пт 04.03.22	Пн 14.03.22	14
16	3.5 Реалізація геометричних фігур	3 дней	Вт 15.03.22	Чт 17.03.22	15
17	3.6 Реалізація додаткового інтерфейсу	4 дней	Пт 18.03.22	Ср 23.03.22	16
18	3.7 Реалізація ліній	8 дней	Чт 24.03.22	Пн 04.04.22	17
19	Оформлення звіту	10 дней	Вт 05.04.22	Пн 18.04.22	18

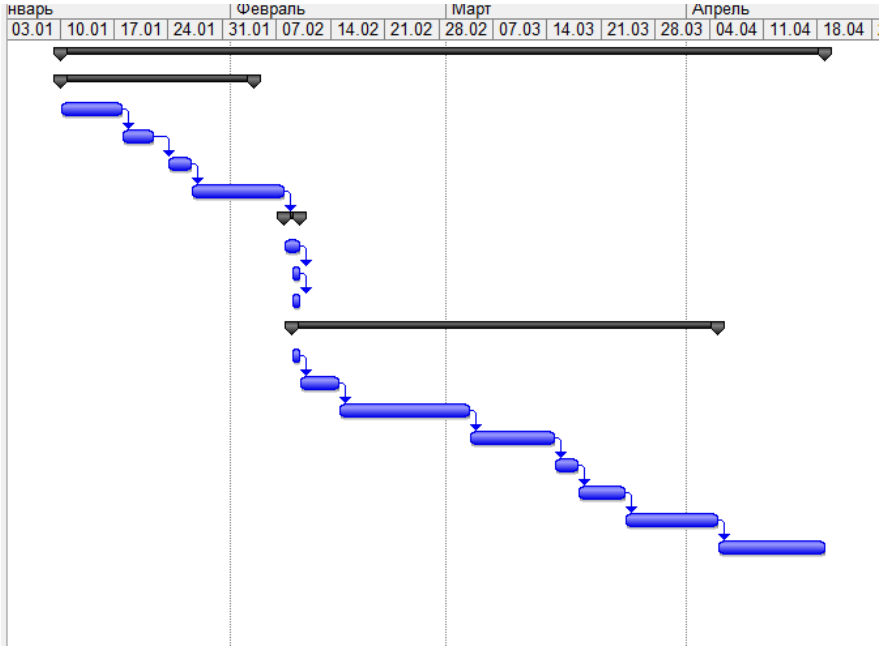


Рисунок Б.3 – Календарний графік розробки web-додатку для редагування діаграм UML

ДОДАТОК В

Програмний код

```
import React from "react";

const CREATE = 'CREATE';
const CORRECT = 'CORRECT';
const SET_ITEMS = 'SET ITEMS';
const SET_LIST_ITEMS ='SET LIST ITEMS';
const TOTAL_COUNT = 'TOTAL COUNT';

let initialState  = {
    ITEMSData: [],
    pageSize: 100,
    totalItemsCount: 0,
    listState:  false
}
const itemsReducer = (state = initialState,  action) => {
    switch(action.type){
        case CREATE:
            return{
                ...state,
                itemsData:  state.itemsData.map(u  =>{
                    if(u.id  === action.figId){
                        return {...u, CREATE: true}
                    }
                    return  u;
                })
```

```
        }
      case CORRECT:
        return{
          ...state,
          itemsData: state.itemsData.map(u =>{
            if(u.id === action.figId){
              return {...u, CREATE: false}
            }
            return u;
          })
        }
      case SET_ITEMS:{
        return {...state, itemsData: action.ITEMS}
      }
      case SET_LIST_ITEMS:{
        return {...state, currentState: action.newcurrentState}
      }
      case TOTAL_COUNT:{
        return {...state, totalITEMSCount: action.totalCount}
      }
      default:
        return state;
    }
    return state;
}

export const CREATEAC = (figId) => ({type: CREATE, figId});
export const CORRECTAC = (figId) => ({type: CORRECT, figId});
export const setITEMSAC = (ITEMS) => ({type: SET_ITEMS, ITEMS});
```

```
export const changePagiAC = (newcurrentState) =>({type: SET_LIST_ITEMS, newcurrentState})
export const totalCountAC = (totalCount) =>({type: TOTAL_COUNT, totalCount})

export default itemsReducer;
const initialState = {
    w: 0,
    h: 0,
    posElX: 0,
    posElY: 0,
    side1: false,
    side2: false,
    side3: false,
    side4: false,
    opas: 1,
    color: '#fff',
    scale: false,
    rot: false,
    char: false
}
const fieldReducer = (state = initialState, action) => {
  if(action.type === 'ADD_ELEMENT'){
      const currentCartItems = !state.items[action.payload.id]
          ? [action.payload]
          : [...state.items[action.payload.id].items, action.payload]


      const newItems = {
          ...state.items,
```

```
        [action.payload.id]:{
            items: currentCartItems,
            totalPrice: getTotalElems(currentCartItems)
        }
    }

    const items  = Object.values(newItems).map(obj => obj.items)
    const allFig  = [].concat.apply([], items)
    const total = getTotal(allFig)

    return {
        ...state,
        items: newItems,
        totalCount: allFig.length,
        totalPrice
    }
} else if(action.type === 'CHANGE_BRACH'){
    ctx.lineWidth  = config.lineSize;
    ctx.lineJoin  = 'round';
    ctx.lineCap  = 'round';
    ctx.strokeStyle = config.color;
    ctx.fillStyle  = config.color;
   return {
      items: {},
      ctx.lineWidth  = config.lineSize;
       ctx.lineJoin  = 'round';
       ctx.lineCap  = 'round';
       ctx.strokeStyle = config.color;
       ctx.fillStyle  = config.color;
    }
```

```
} else if(action.type === 'DELETE_ITEM'){
    const newItems = {
        ...state.items
    }
    var sketch = document.getElementById("sketch");
     var x = posX;
     var y = posY;

     var drawing = setInterval(() => {
        var currentX = x.shift();
        var currentY = y.shift();
        if (x.length <= 0 && y.length <= 0) {
            clearInterval(drawing);
            switchIndicator(true);
            isRec = false;
            newDraw = true;
        }else {
            if(currentX == undefined && currentY == undefined) {
                ctx.beginPath();
            }else {
                drawLine(currentX,  currentY);
            }
        }
     }, 1);

     if(sketch != null) {
        sketch.style.visibility = 'hidden';
     }
    return {
        ...state,
```

```
            items: newItems,
            totalFig: state.totalFig - currentFigIndex,
            totalCount: state.totalCount - currentTotalCount
        }
    }
    return state
}
export default fieldReducer
let initialState = {
    fieldState: 0
}
const figReducer = (state = initialState, action) =>{

    switch (action.type) {
        case DRAW_BEGIN: {
            if(newDraw) {
                ctx.clearRect(0, 0, canvas.width, canvas.height);
                newDraw = false;
                if(sketch != null) {
                    sketch.style.visibility = 'visible';
                }
            }
            ctx.beginPath();

            let stateCopy = {...state}
            stateCopy.canData=[...state.canData]
            stateCopy.canData.push(newCan);
            return stateCopy;
        }
```

```
        case STOP: {
            canvas.onmousemove = null;
            posX.push(undefined);
            posY.push(undefined);
            let stateCopy = {...state}
            stateCopy.fieldState = action.newValue;
            return stateCopy;
        }

        default:
            return state;
    }

}

export const addCanActionCreator = () => ({type:ADD_CAN})

export const changeValueActionCreator  = (text) => ({type:NEW_CAN_TEXT,
newValue:text})

const ADD_CAN = 'ADD-CAN';
const NEW_CAN_TEXT = 'NEW-CAN';



export default  figReducer;

let store = createStore(reducers);

window.store = store;
```

```
export default store;

const ADD_NEW_TEXT = 'ADD NEW TEXT'
const WRITE_NEW_TEXT = 'WRITE NEW TEXT'

let initialState  = {
    textData: [],
    currentText: ''
}

const textReducer = (state = initialState,  action) => {
    switch (action.type) {
        case ADD_NEW_TEXT: {
            let newtext = {id: 3, text: state.currenttext}
            return {...state, textData: [...state.textData, newtext]}
        }
        case WRITE_NEW_TEXT: {
            return {...state, currenttext: action.text}
        }
        default:{
            return state;
        }
    }
}

export const writeNewtextAC  = (text) => ({type:WRITE_NEW_TEXT,
text:text});
export const addNewtextAC = () => ({type:ADD_NEW_TEXT})

export default textReducer;
```

```
const ADD_NEW_FIG = 'ADD NEW FIG'
const WRITE_LINE = 'WRITE LINE';

let initialState  = {
    canData:[],
    currentCan: ''
}

const menuReducer = (state = initialState,  action) =>{
    switch(action.type){
        case ADD_NEW_FIG:{
            var sketch = action.payload;
            var x = posX;
            var y = posY;

            var drawing = setInterval(() => {
                var currentX = x.shift();
                var currentY = y.shift();
                if (x.length <= 0 && y.length <= 0) {
                    clearInterval(drawing);
                    switchIndicator(true);
                    isRec = false;
                    newDraw  = true;
                }else {
                    if(currentX  == undefined && currentY == undefined) {
                        ctx.beginPath();
                    }else {
                        drawLine(currentX,  currentY);
                    }
```

```
            }
        }, 1);

        if(sketch != null) {
            sketch.style.visibility = 'hidden';
        }
        return {...state, sketch: action.payload}
    }

    case WRITE_LINE:{
        ctx.lineTo(x, y);
        ctx.stroke();
        if(enable) {
            indicator.classList.add('isWrite');
        }else {
            indicator.classList.remove('isWrite');
        }
        return {...state, currentCan: action.payload}
    }
    default:{
        return state;
    }
  }
}

export const writingNewCanAC = (text) => ({type:WRITE_NEW_CAN,
text:text});
export const addNewCanAC = () => ({type:ADD_NEW_CAN});

export default menuReducer
```

```
const SET_ITEMS = 'set ITEMS'
const CREATE = 'CREATE'
const CORRECT = 'CORRECT'

let initialState  = {
    itemsData: []
}


let ITEMSReducer =  (state = initialState,  action) => {
    switch (action.type) {
        case CREATE:
            return{
                ...state,
                ITEMSData: state.ITEMSData.map(u =>{
                    if(u.id === action.figId){
                        return {...u, CREATE: true}
                    }
                    return u;
                })
            }
        case CORRECT:
            return{
                ...state,
                ITEMSData: state.ITEMSData.map(u =>{
                    if(u.id === action.figId){
                        return {...u, CREATE: false}
                    }
                    return u;
```

```
            })
        }
    case SET_ITEMS:
        return{...state, ITEMSData: action.payload}
    default:
        return state;
}}

export const CREATEAC = (figId) => ({type:CREATE, figId});
export const CORRECTAC = (figId) => ({type:CORRECT, figId});
export const setITEMSAC = (ITEMS) => ({type: SET_ITEMS, ITEMS})

export default ITEMSReducer;

import React from "react";

const ADD_NEW_CAN = 'ADD NEW CAN';
const WRITE_NEW_CAN = 'WRITE NEW CAN';



let initialState  = {
    current: ''
}

const menuReducer = (state = initialState,  action) =>{

    switch (action.type) {
        case ADD_NEW_CAN:{
            let THEnewCan = {id:1, text:state.currentCan};
            let stateCopy = {...state};
```

```
            stateCopy.canData = [...state.canData];

            stateCopy.canData.push(THEnewCan);
            stateCopy.currentCan ='';

            return stateCopy;
        }
        case WRITE_NEW_CAN:{
            let stateCopy = {...state};
            stateCopy.currentCan = action.text;

            return stateCopy;
        }
        default:
            return state;
    }
}

export const addCanActionCreator =()=>({type:ADD_NEW_CAN})
export const writeCanActionCreator  =(txt)=>({type:WRITE_NEW_CAN,
text:txt})

export default menuReducer;
```