Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

Теорія Розробки Програмного Забезпечення

РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ:
CLIENT-SERVER, PEER-TO-PEER,
SERVICE-ORIENTED ARCHITECTURE
Варіант 23

Виконав
студент групи ІА-14:
Фіалківський І.О.

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ:CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

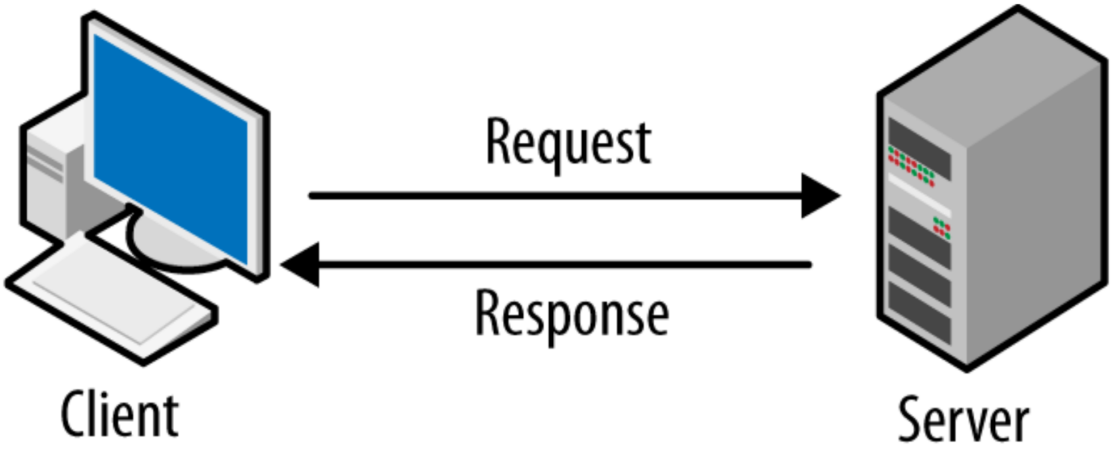
- Ознайомитися з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Застосування одного з розглянутих шаблонів при реалізації програми.

Хід роботи:

Тема: Згідно мого варіанту, було виконано CLIENT-SERVER.

Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server)

Архітектура клієнт-сервер - архітектура комп'ютерної мережі, в якій багато клієнтів (віддалених процесорів) запитують і отримують послуги від централізованого сервера (хост-комп'ютера). Клієнтські комп'ютери надають інтерфейс, що дозволяє користувачеві комп'ютера запитувати послуги сервера і відображати результати, які сервер повертає. Сервери чекають на запити від клієнтів, а потім відповідають на них.



Це ілюструє архітектуру клієнт-сервер, ключову ідею в обчислювальній техніці. У цьому підході клієнти, як ваш пристрій, звертаються до серверів, як система в ресторані, за послугами або ресурсами. Сервери обробляють ці запити і відповідають необхідними даними, що призводить до динамічної та ефективної взаємодії. Різноманітні онлайн-сервіси, включаючи веб-сайти та електронну пошту, підтримуються цією архітектурою, яка дозволяє безперешкодно обмінюватися ресурсами і спілкуватися між клієнтами і серверами через такі мережі, як Інтернет.

Приклад звернення до Server

```
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";

export const userApi = createApi({
  reducerPath: "userApi",
  baseQuery: fetchBaseQuery({
    baseUrl: `${import.meta.env.VITE_APP_API_URL}`,
  }),
  endpoints: (build) => ({
    createUser: build.mutation<CreateUserResponse, CreateUserInput>({
      query: (info) => ({
        url: "/auth/registration",
        method: "POST",
        body: info,
      }),
    }),
    loginUser: build.mutation<LoginUserResponse, LoginUserInput>({
      query: (data) => {
        const bodyFormData = new FormData();
        bodyFormData.append("username", data.username);
        bodyFormData.append("password", data.password);
        return {
          url: "/process_login",
          method: "POST",
          headers: {
            "Content-Type": "multipart/form-data",
          },
          body: bodyFormData,
          formData: true,
        };
      },
    }),
  }),
});
```

Цей код використовує бібліотеку @reduxjs/toolkit/query/react для створення API-інтерфейсу, який легко інтегрується з Redux Toolkit та React. Задача цього API-інтерфейсу полягає в забезпеченні можливості виконання асинхронних запитів до сервера, таких як реєстрація користувача (createUser) та вхід в систему (loginUser).

Спочатку визначається змінна userApi, яка містить створений за допомогою createApi об'єкт API. Цей об'єкт включає налаштування, такі як шлях до редуктора, базовий запит та кілька кінцевих точок (endpoints), які представляють конкретні операції.

На рисунку 2.3.1 зображено графічний інтерфейс системи з веб-браузера.

Він реалізований за принципом SPA. На першому компоненті ми момежемо помітити форму для реєстрації або автроризації юзера.

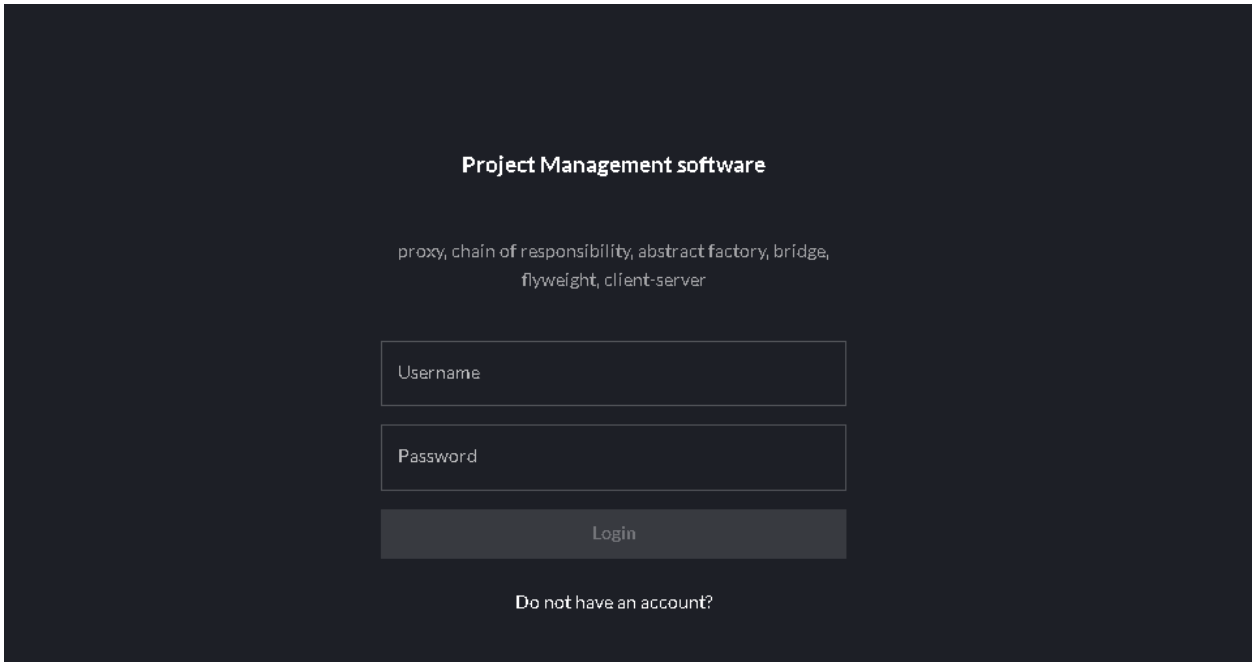


Рисунок 2.3.1 – Перший компонент клієнтської частини

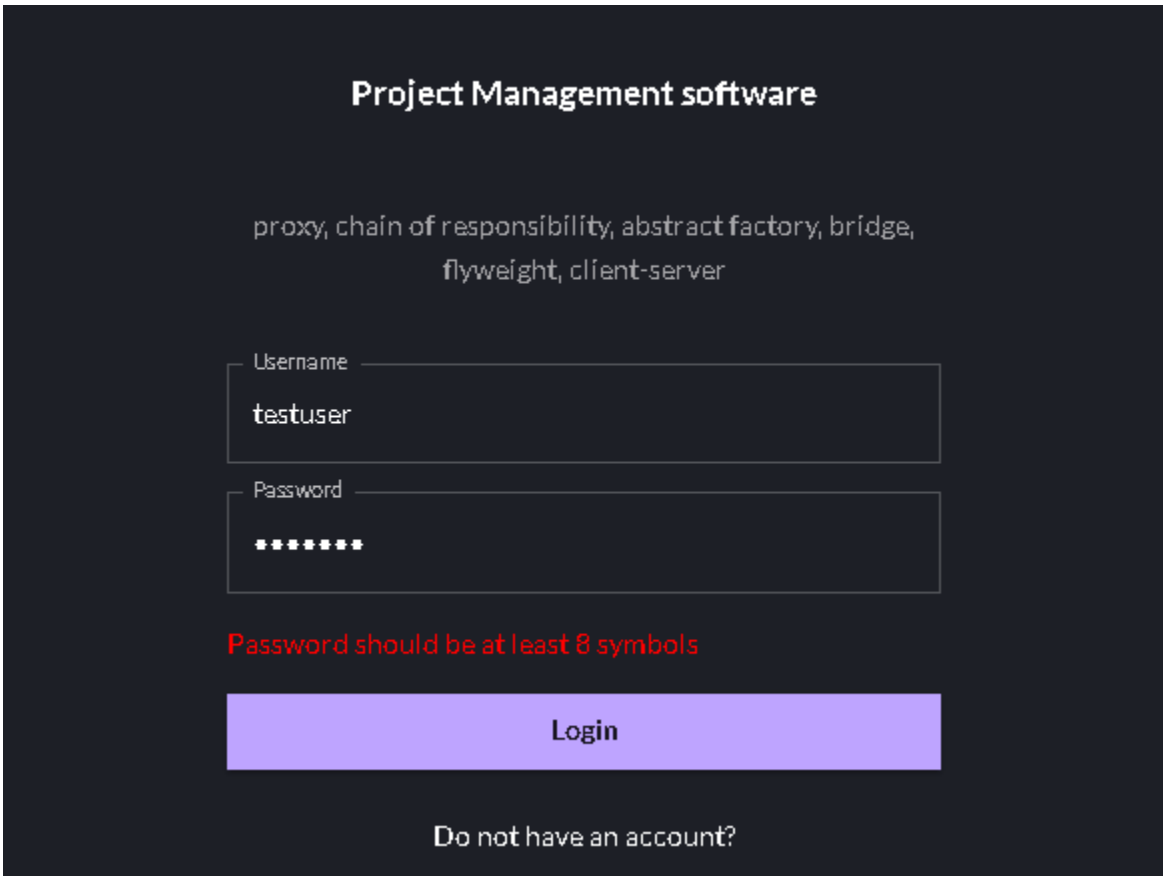


Рисунок 2.3.2 – Приклад валідації

При успішній авторизації юзер потрапляє на компонент зі своїми проектами.

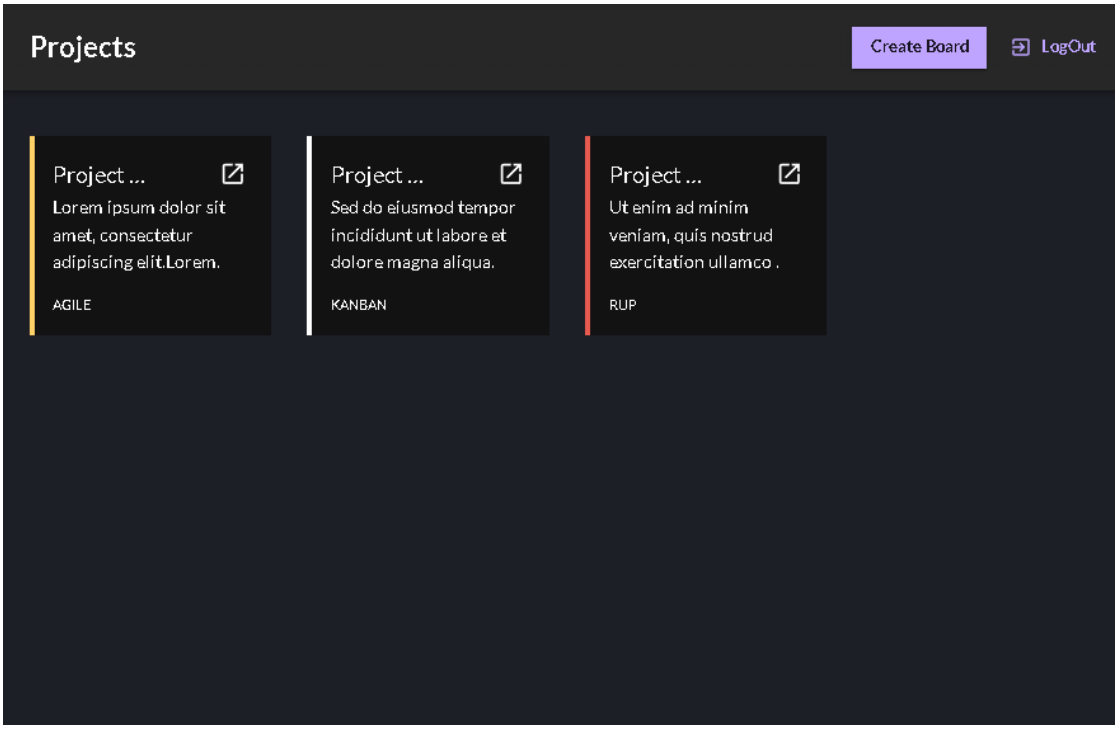


Рисунок 2.3.3– Приклад проектів юзера

Також реалізована функція додавання проектів до списка юзера.

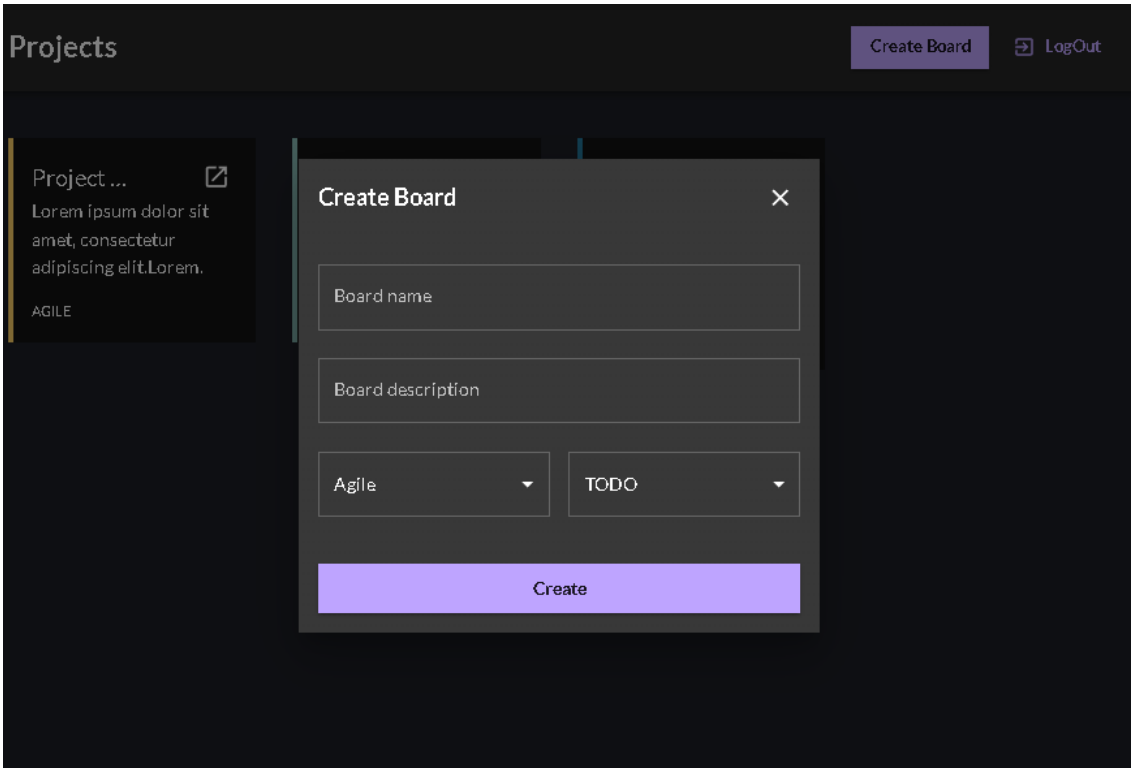


Рисунок 2.3.4– Приклад модального вікна для створення проекта.

.

Коли юзер перейде за одним з проектів, він отримає дошку завдань. Яка надає

можливість керувати статусами завдань(TODO, IN_PROGRESS, DONE).

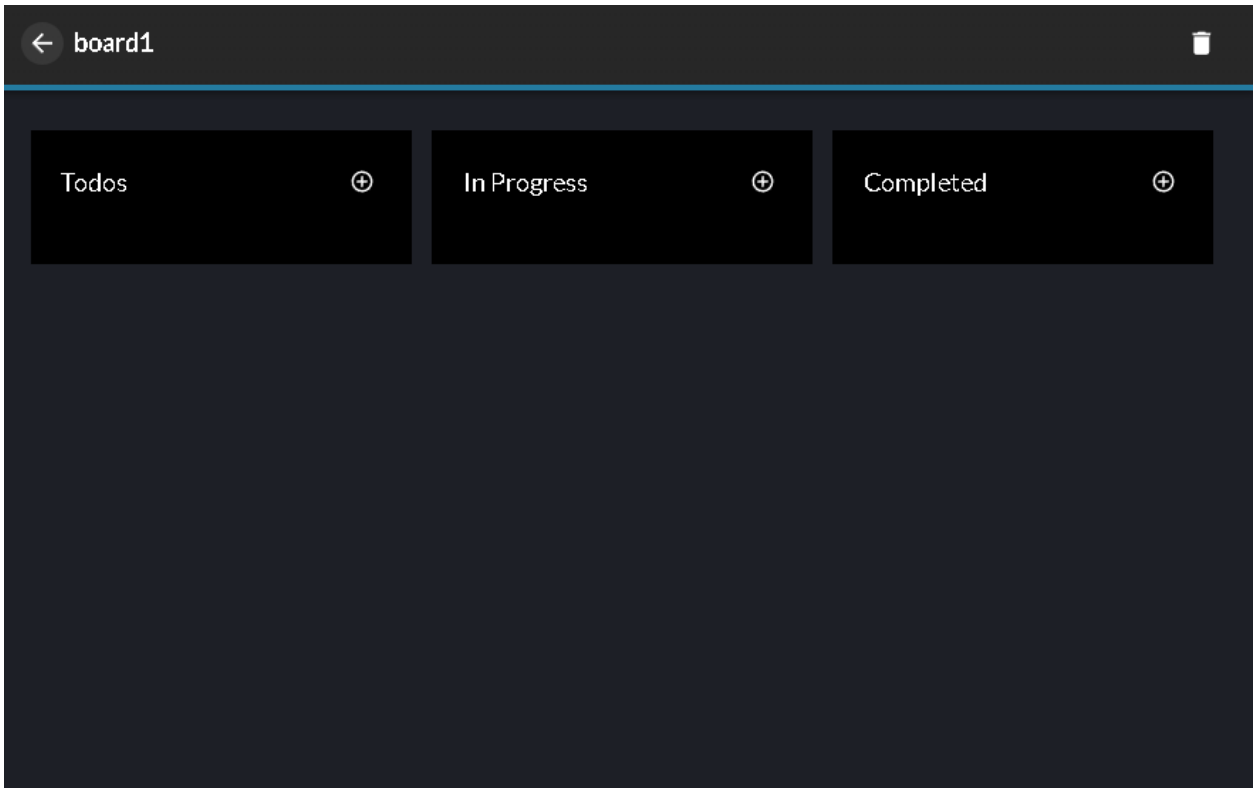
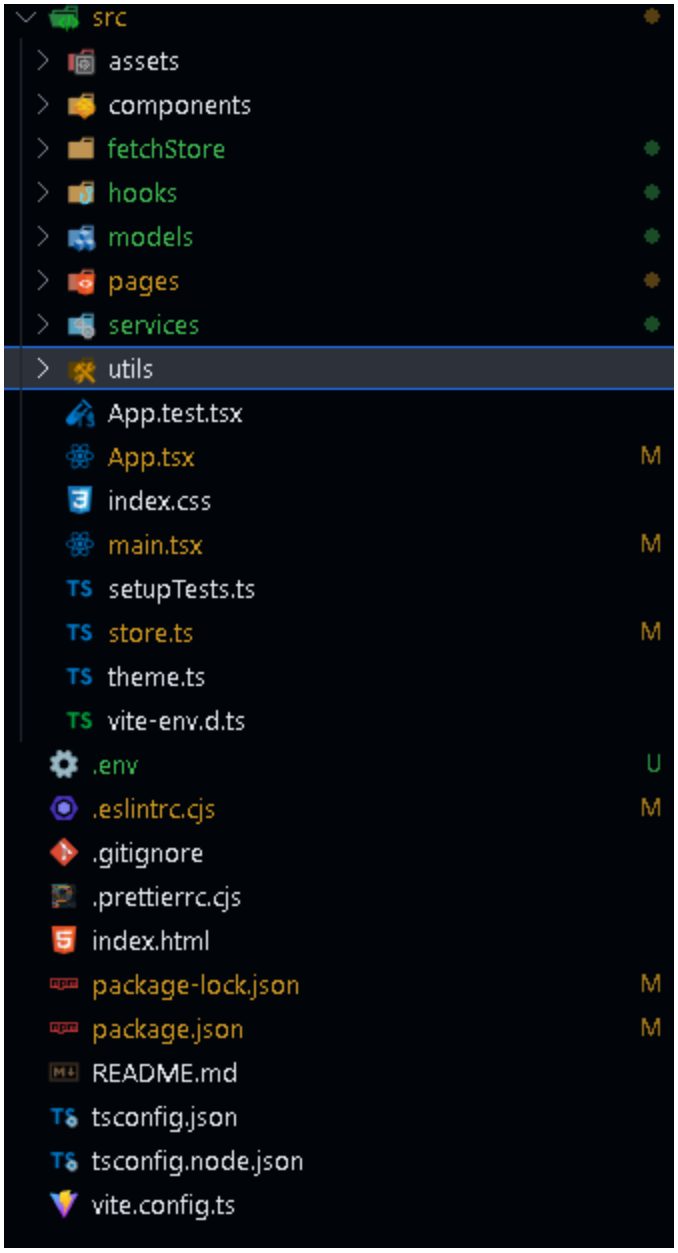


Рисунок 2.3.5– Компонент з завданнями.



Переглянути весь код можна за посиланням:

https://github.com/vergovters/myBoard/tree/master

Висновок: застосування шаблону "Client-Server" виявляється дуже ефективним при побудові архітектури систем, особливо в ситуаціях, коли необхідно розділити клієнтську (абстракцію користувача) та серверну (логічну обробку та управління ресурсами) частини, забезпечуючи можливість їх зміни та розширення незалежно одне від одного.

У цій архітектурі, клієнт і сервер взаємодіють через визначений інтерфейс, визначений на клієнтському боці (абстракція) і на серверному боці (реалізація). Кожен конкретний клас-спадкоємець від клієнтської частини може мати свої власні варіації взаємодії з користувачем, тоді як конкретні класи-спадкоємці від серверної частини можуть реалізовувати різні аспекти обробки та управління ресурсами.

Цей підхід полегшує розширення системи, оскільки можна вводити нові функціональні можливості на клієнтському або серверному боці без зміни існуючого коду. Шаблон "Client-Server" сприяє збереженню високого рівня гнучкості та розширюваності системи, дозволяючи динамічно змінювати функціональність та взаємодію між клієнтом і сервером.

Важливо відзначити, що такий підхід дозволяє відокремлювати клієнтську логіку від серверної, сприяючи ефективній розробці, тестуванню та підтримці системи. Також, ця архітектура покращує масштабованість, оскільки можна гнучко збільшувати кількість серверів для обробки зростаючого навантаження від клієнтів.