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 №4

Технології розроблення програмного забезпечення

*Тема роботи: «*ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY.*»*

Виконав студент групи ІА-23:

Каширов Д. О.

Перевірив:

Мягкий Михайло Юрійович

Київ 2024

Мета: реалізація частини функціоналу робочої програми у вигляді класів та їхньої взаємодії із застосуванням одного із розглянутих шаблонів проектування.

Хід роботи

Варіант:

..6 Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p) Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структуру html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) - переходи при перенаправленнях, відображення сторінок 404 і 502/503.

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Короткі теоретичні відомості:

Шаблон проектування (design pattern) — це повторюване рішення для типових проблем проєктування в об'єктно-орієнтованому програмуванні. Шаблони проектування надають готову структуру або набір кроків для вирішення задач, які постають перед програмістами під час розробки програмного забезпечення. Вони не є готовим кодом, а скоріше орієнтиром, який можна адаптувати до конкретних потреб проєкту.

Завдяки шаблонам проектування розробники можуть створювати більш гнучкі та підтримувані рішення, уникати поширених помилок і робити код зрозумілішим.

Шаблони проектування допомагають розробникам:

Покращити якість коду: Вони надають перевірені рішення для типових проблем, що дозволяє уникати помилок.

Зробити код легшим для підтримки та масштабування:

Правильно вибраний шаблон структурує код, роблячи його простішим для змін і доповнень.

Забезпечити гнучкість: Використання шаблонів дозволяє створювати розширюваний код, який легко адаптується до нових вимог. Стандартизувати підхід: Це полегшує командну роботу, оскільки розробники використовують загальноприйняті способи вирішення задач. Чим відрізняється шаблон «стратегія» від «стану»? 4 Обидва шаблони належать до поведінкових і використовують принципи поліморфізму, але мають різні цілі: Шаблон «стратегія» (Strategy) дозволяє вибирати алгоритм під час виконання програми. Замість того щоб використовувати один метод або набір логіки, шаблон Strategy дозволяє використовувати різні варіанти реалізації, що 4можуть динамічно змінюватися в залежності від умов. Прикладом може бути різне обчислення знижок для різних типів клієнтів. Шаблон «стан» (State) змінює поведінку об'єкта залежно від його поточного стану. Він корисний, коли об'єкт має обмежену кількість станів, і кожен з них змінює поведінку цього об'єкта. На відміну від «стратегії», «стан» передбачає, що об'єкт змінює свою внутрішню логіку залежно від стану.

В чому полягає ідея шаблона «одинак»?

Чому його вважають «анти- шаблоном»?

Шаблон «одинак» (Singleton) гарантує, що певний клас матиме лише один екземпляр, який буде доступний з будь-якої точки програми. Це корисно, коли необхідно мати єдиний доступний ресурс, наприклад, конфігурацію програми або з'єднання з базою даних. Проте, «одинак» часто вважається «анти-шаблоном» через такі проблеми: Погіршення тестованості: Singleton ускладнює написання юніт-тестів, оскільки його складно ізолювати та підмінити в тестах. Погіршення гнучкості: Singleton фіксує клас у стані одного екземпляра, що може обмежити можливість його розширення або зміни. Прихована залежність: Замість того щоб явно передавати залежність у методи чи класи, Singleton стає глобальною залежністю, що порушує принципи чистого коду. Шаблон «проксі» (Proxy) забезпечує інтерфейс для доступу до іншого об'єкта, часто додаючи додаткову поведінку або контролюючи доступ.

Проксі- клас виступає посередником, який обмежує або розширює функціональність оригінального об'єкта. «Проксі» використовують у наступних ситуаціях: Відкладене створення об'єкта: Proxy створює об'єкт лише тоді, коли він дійсно потрібен (наприклад, завантаження важких ресурсів). 5Контроль доступу: Proxy може перевіряти права доступу перед викликом методів об'єкта. Логування або кешування: Proxy дозволяє додавати логування, кешування або інші операції, не змінюючи логіку оригінального об'єкта. Мережеві операції: Proxy може представляти об'єкти, які фізично знаходяться в інших машинах або мережах, дозволяючи їм взаємодіяти через мережу.

Хід роботи:

Реалізація паттерну Proxy:

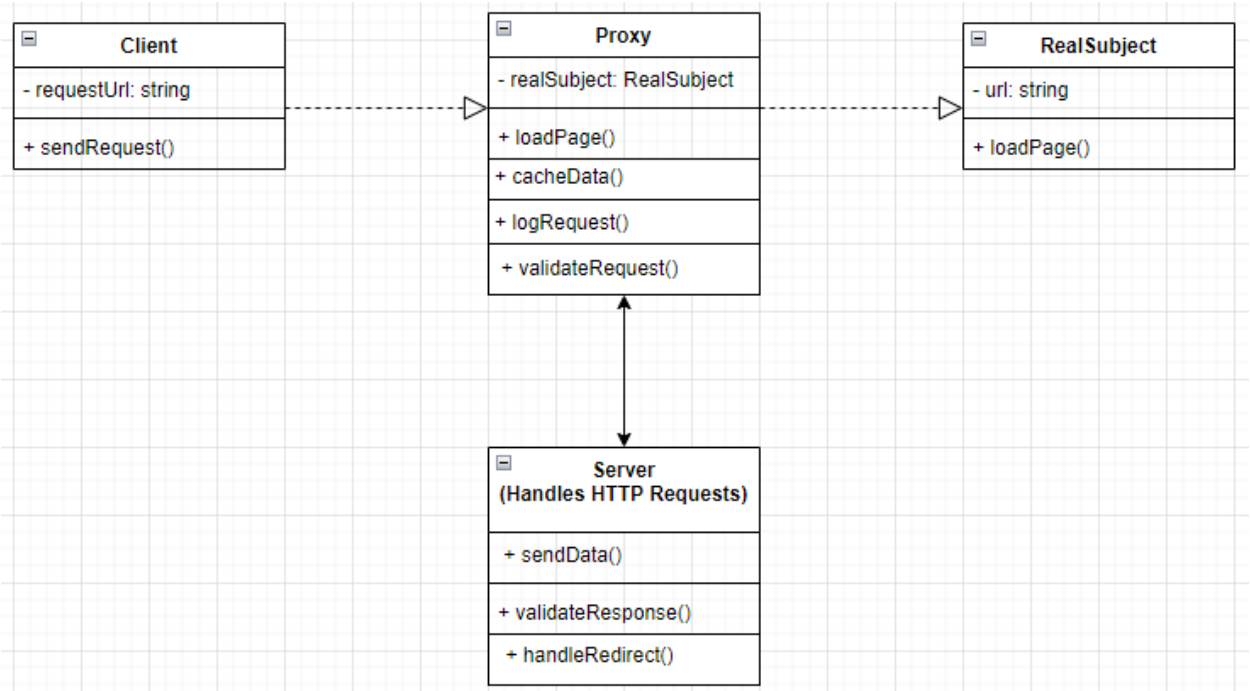


Рис 1 – реалізація паттерну Proxy

Опис компонентів діаграми:

1. **Client (Клієнт)**:
    - **requestUrl**: Це URL, який клієнт хоче завантажити.
    - **sendRequest()**: Метод, що ініціює запит до **Proxy** для завантаження сторінки.
2. **Proxy (Проміжний об'єкт)**:
    - **realSubject**: Посилання на реальний об'єкт (сервер), який фактично обробляє запит.
    - **loadPage()**: Метод, що викликає реальний об'єкт (сервер) для завантаження сторінки.
    - **cacheData()**: Логіка кешування даних, яка може бути додана в **Proxy** перед відправкою запиту.
    - **logRequest()**: Логування запиту, що обробляється через **Proxy**.

- **validateRequest()**: Перевірка запиту перед передачею на сервер.

3. **RealSubject (Реальний об'єкт)**:
    - **url**: URL сторінки, яку завантажує сервер.
    - **loadPage()**: Метод реального об'єкта, який фактично завантажує дані сторінки через сервер.
4. **Server (Сервер)**:
    - **sendData()**: Логіка, яка відправляє дані назад до **Proxy** або **Client**.
    - **validateResponse()**: Перевірка відповіді на запит (наприклад, код статусу HTTP).
    - **handleRedirect()**: Обробка перенаправлень, якщо сторінка має редиректи (наприклад, 301 або 302).

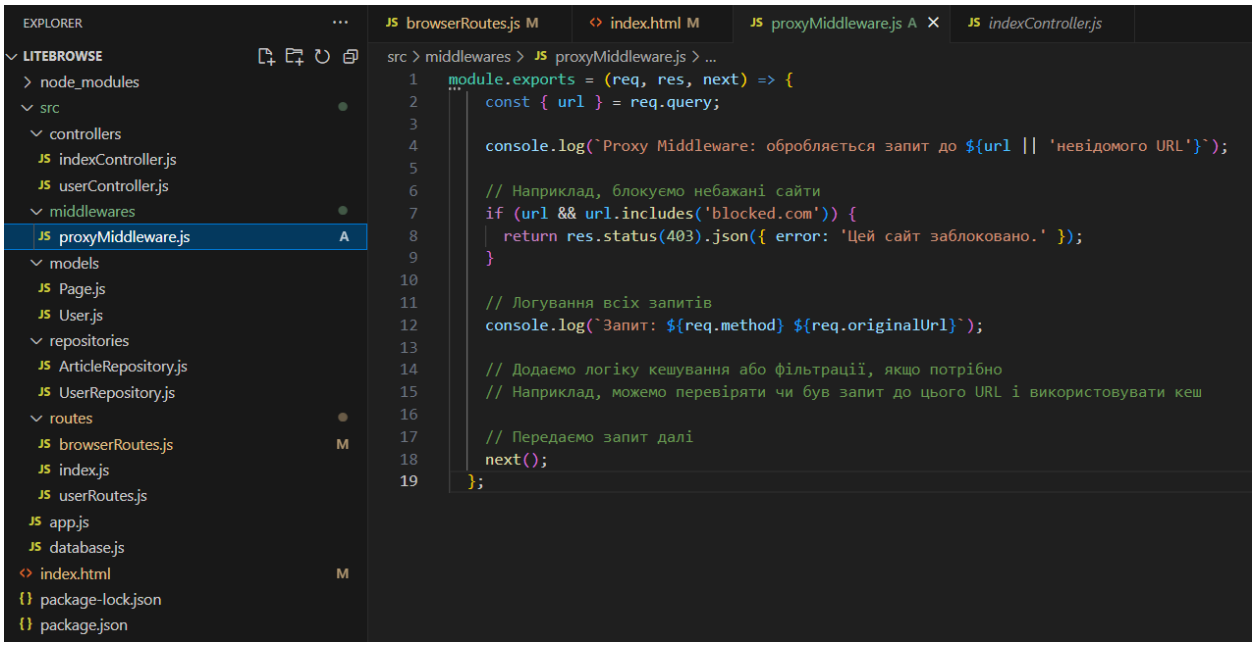
Опис зв'язків:

- **Client** звертається до **Proxy** через метод **sendRequest()**.
- **Proxy** може виконати попередню обробку запиту, як-от кешування або валідацію, а потім викликає метод **loadPage()** реального об'єкта **RealSubject**.
- **RealSubject** (сервер) фактично завантажує сторінку і відправляє її через метод **sendData()**.
- **Proxy** може обробляти додаткові логічні операції, як-от кешування, перед тим як передати дані назад до клієнта.
- **Server** обробляє статуси HTTP і може виконувати перенаправлення через метод **handleRedirect()**.

Як це застосовувати в проєкті:

У твоєму проекті ти використовуєш **Proxy** як проміжний рівень для обробки запитів і відповіді від серверу (через **axios**). Це дозволяє реалізувати додаткові функції (як-от кешування, фільтрація запитів, валідація або логування), не змінюючи основну логіку обробки запитів сервером.

Код з додавання патерну Proxy



Посилання на Git Hub на гілку main1 [https://github.com/DimytroKashiroff/TRPZ_Kashirov_All-laboratory-work.git]

Висновок:

Тепер твоя UML-діаграма і патерн **Proxy** надають чітке уявлення про те, як ти організував обробку запитів через проміжний рівень. Це дозволяє додавати додаткові функції, такі як кешування, фільтрація чи логування, не змінюючи основну бізнес-логіку.