

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ
Дисципліна «Технології розроблення програмного забезпечення»
Курс 3 Група ІА-04 Семестр 5

ЗАВДАННЯ
на курсову роботу студента

Глушко Юлія Петрівна
(прізвище, ім'я, по батькові)

1. Тема роботи _____ HTTP-сервер
2. Строк здачі студентом закінченої роботи _____ 15.01.2023
3. Вихідні дані до роботи: перелік функцій, які має виконувати HTTP-сервер : сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), вміти працювати із динамічними jsp сторінками, вести статистику вхідних запитів, обробку запитів у багатопотоковому режимі.
4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)
постановка задачі, огляд існуючих рішень, визначення вимог до застосунку системи (функціональних та нефункціональних), опис сценаріїв використання, побудова концептуальної моделі системи, вибір та проектування бази даних, вибір та обґрунтуванні патернів проектування.
Додатки:
Додаток А- діаграма класів системи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Діаграми класів, діаграма розгортання, діаграма прецедентів, схема архітектури системи, знімки екрану фрагментів коду, знімки екрану із вікном графічного інтерфейсу

6. Дата видачі завдання _____ 06.10.2022

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Студент _____
(підпис)

Юлія ГЛУШКО
(Ім'я ПРІЗВИЩЕ)

Керівник _____
(підпис)

Валерій КОЛЕСНИК
(Ім'я ПРИЗВИЩЕ)

« 15 » січня 2023 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема _____ HTTP-сервер _____

Курсова робота

З дисципліни «Технології розроблення програмного забезпечення»

Керівник

ас. Колеснік В. М.

«Допущений до захисту»

(Особистий підпис керівника)

«15» _____ січня _____ 2023р.

Захищений з оцінкою

(оцінка)

Члени комісії:

(особистий підпис)

(особистий підпис)

Виконавець

ст. Глушко Ю.П.

залікова книжка № ІА -0406

гр. ІА-04 _____

(особистий підпис виконавця)

« 15» _____ січня _____ 2023р.

(розшифровка підпису)

(розшифровка підпису)

Київ – 2023

ЗМІСТ

ВСТУП	3
1 ПРОЄКТУВАННЯ СИСТЕМИ.....	4
1.1. Огляд існуючих рішень	4
1.2. Загальний опис проєкту.....	5
1.3. Вимоги до застосунків системи.....	6
1.3.1. Функціональні вимоги до системи.....	6
1.3.2. Нефункціональні вимоги до системи.....	8
1.4. Сценарії використання системи	8
1.5 Концептуальна модель системи	12
1.6 Вибір бази даних	20
1.7 Вибір мови програмування та середовища розробки.....	21
1.8 Проєктування розгортання системи.....	22
2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ	24
2.1. Структура бази даних	24
2.2. Архітектура системи.....	25
2.2.1. Специфікація системи	25
2.2.2. Вибір та обґрунтування патернів реалізації.....	26
2.3. Інструкція користувача.....	34
ВИСНОВКИ.....	36
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	37
ДОДАТКИ.....	38

ВСТУП

Інтернет з кожним роком все більше поглинає сучасний світ, проникаючи в усі сфери життя людини. Всесвітня мережа невинно розвивається, впроваджуючи нові, більш оптимізовані ніж попередні, технології роботи, доступу до даних, розподілення мереж. Кожен користувач мережі інтернет має доступ до інформаційних ресурсів, що містяться на різних серверах, що знаходяться в усіх куточках планети. Інтернет є засобом для віддаленого спілкування людей, середовищем для організації дистанційного навчання, медичних консультацій, бізнесу, реклами, торгівлі.

Веб-сервери відіграють величезну роль в Всесвітній мережі. Веб-серверами можуть бути комп'ютери або спеціальні програми, які виконують роль сервера. Коли користувач намагається отримати HTML-сторінку через введення адреси в командний рядок браузера, то браузер посилає запит через протокол передачі даних HTTP на сервер. Коли запит досягає потрібного веб-сервера, він отримує та обробляє запит та передає потрібну сторінку назад, також через протокол HTTP.

Веб-сервер може бути статичним або ж динамічним. Статичний сервер просто надсилає потрібні файли до браузера. Динамічний також вміє надсилати файли до браузера і на ньому встановлено додаткове програмне забезпечення, яке перед відправкою до браузера вміє обробляти динамічні сторінки - на льоту генерувати відповідь.

Метою роботи є створення власного web-серверу, що працює як P2P застосунок і передає дані відповідно до стандартів протоколу HTTP.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

Фундаментом Всесвітньої павутини є веб-сервери. Веб-сервером називають як програмне забезпечення, що виконує певні функції, так і обладнання, на якому це програмне забезпечення працює. Веб-сервери дають доступ до даних за певним протоколом. Зараз найвідомішими у світі веб-серверами є : [1,2]

- Cloudflare – це мережа серверів по всьому світу, до якої люди підключають свої сайти, щоб збільшити швидкість їх завантаження та захистити від DDoS-атак. Також за допомогою цього сервісу можна керувати записами DNS на домені і перевести сайт на HTTPS. Cloudflare має як безкоштовні тарифи, так і кілька платних.

- Nginx - безкоштовний веб-сервер і проксі-сервер. Є версії для сімейства Unix-подібних операційних систем та Microsoft Windows;

- Lighttpd — веб-сервер, розроблений з розрахунком на швидкість, захищеність і відповідність стандартам. Це вільне програмне забезпечення, що розповсюджується по ліцензії BSD;

- Jigsaw - походить від World Wide Web Consortium. Є відкритим, вільним і може працювати на різних платформах, таких як Linux, Unix, Windows, Mac OS X Free BSD. Написаний на Java і може працювати з CGI скриптами і PHP програмами;

- Google Web Server (GWS) — веб-сервер, який використовує Google для організації своєї веб-інфраструктури;

- Apache Web Server - веб-сервер http. Це безкоштовний і найпопулярніший веб-сервер у світі, розроблений Apache Software Foundation. Це веб-сервер з відкритим вихідним кодом, який може бути встановлений майже на всіх операційних системах.

- LiteSpeed - це веб-сервер з відкритим вихідним кодом, який може обробляти велику кількість запитів на секунду, що дозволяє йому працювати на тому ж рівні, що

і деякі апаратні балансувальники навантаження. У багатьох випадках можна виявити, що LiteSpeed швидше, ніж Apache і Nginx. Також цей сервер використовує менше ресурсів і в більшості випадків перевага в швидкості із Apache і Nginx сягає 15:1.

- Envoy — це високопродуктивний розподілений проксі-сервер, спроектований як для окремих сервісів і додатків, так і для роботи у вигляді шини комунікації в мікро сервісній архітектурі.

1.2. Загальний опис проєкту

Проєкт працює як додаток, розроблений за P2P архітектурою. Це означає, що кожен користувач є як клієнтом, так і сервером, а отже може сам обробляти запити, що надходять до нього, та надсилати запити до інших користувачів.

Серверна частина користувача підтримує два види HTTP-запитів : get та post, може обробляти як статичні html - , так і генерувати «на льоту» динамічні jsp – сторінки. Серверна частина прослуховує певний порт, до якого можуть надсилати запити інші користувачі в якості клієнта. На кожен вхідний запит генерується відповідь в залежності від даних та запиту. Всі запити та відповіді, за допомогою яких і спілкуються користувачі, відповідають стандартам HTTP/1.1. Це дозволяє підключитись до користувача та використовувати його в якості сервера для різних клієнтів-браузерів, що використовують для спілкування протокол HTTP/1.1 (наприклад Google Chrome).

HTTP є протоколом прикладного рівня, який найчастіше використовує можливості іншого протоколу – TCP (або TLS) – для пересилання своїх повідомлень, проте будь-який інший надійний транспортний протокол теоретично може бути використаний для доставки таких повідомлень.

Клієнти та сервери спілкуються обмінюючись одиночними повідомленнями. Користувач може зробити запит як клієнт до одного із активний користувачів, як до сервера. Для зручного вводу потрібних даних та відображення відповідей, було додано графічний інтерфейс, що базується на Swing[3].

1.3. Вимоги до застосунків системи

1.3.1. Функціональні вимоги до системи

Діаграма прецедентів системи представлена на рис. 1.1.

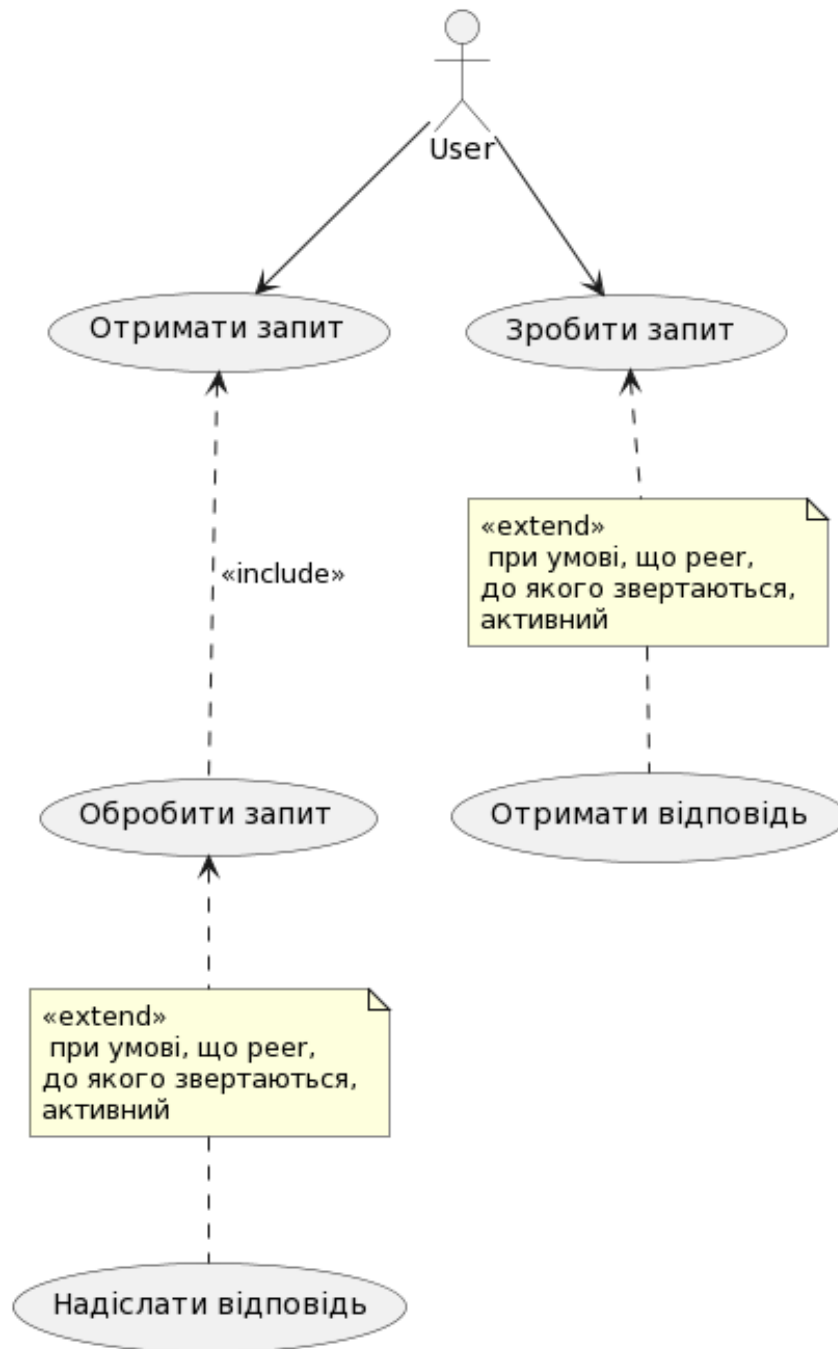


Рисунок 1.1 – Діаграма прецедентів

Система має відповідати наступним функціональним вимогам:

- надсилання запитів на серверну частину будь-якого активного користувача системи. Дані , що необхідні (адреса сторінки, метод, параметри, порт) визначає користувач, що хоче зробити запит. Запит відбувається по протоколу HTTP/1.1.
- отримання відповіді від серверної частини певного користувача відбувається лише за умови попереднього надсилання запиту на порт цього користувача та за умови, що користувач у поточний момент часу є активним. Інакше буде отримано повідомлення, що до вказаного порту сервера неможливо підключитись, а отже і передати дані. Відповідь приходить по протоколу HTTP/1.1 та відображається користувачу.
- отримання запиту на серверну частину користувача. Запит приходить по протоколу HTTP/1.1 та обробляється відповідно до вказаних даних. Сервер може обробляти як статичні html-сторінки, так і динамічні jsp, використовуючи надіслані у запиті параметри.
- обробка запиту відбувається після його отримання. Це генерація на льоту динамічних сторінок із попередньої їхньої обробкою, або просто видача статичних. На цьому етапі і відбуваються майже всі обчислення та дії, не пов'язані із передачею даних.
- надсилання відповіді від серверної частини відбувається після обробки отриманого запиту. Відповідь генерується по протоколу HTTP/1.1 та надсилається на клієнтську частину користувача, що надіслав запит. У випадку успішної відповіді надсилається оброблена html-сторінка; у випадках, якщо запитаної сторінки не було знайдено або на сервері трапились проблеми – відповідний код відповіді в протоколі HTTP/1.1, що сповістить клієнта про це. Цей пункт виконується, якщо клієнт, що повинен отримати відповідь, у поточний момент часу перебуває в активному стані. Інакше буде отримано повідомлення, що до порту клієнта неможливо підключитись, а отже і передати дані.

1.3.2. Нефункціональні вимоги до системи

Система має відповідати наступним нефункціональним вимогам:

- інтерфейс користувача має бути зручним та інтуїтивно-зрозумілим. Щоб цього досягнути було додано графічний інтерфейс для зв'язку системи і користувача. Для того, щоб система отримала необхідні дані від користувача, йому необхідно лише заповнити деякі поля.
- якщо клієнт зробив запит, він отримає відповідь від серверної частини певного користувача. Відповідь буде відображена у вікні графічного представлення. Графічний інтерфейс використовується для зручного спілкування застосунку і користувача.
- система повинна правильно відображати дані та надійно працювати.

1.4. Сценарії використання системи

Таблиця 1.1 – Сценарій використання «Зробити запит»

Назва	Зробити запит
Передумови	Відсутні
Післяумови	Інший користувач отримає запит
Сторони, що взаємодіють	Цей користувач як клієнт та інший користувач як сервер
Опис	Запит будується (із отриманих через графічний інтерфейс даних) по протоколу HTTP/1.1 та надсилається на вказану адресу (порт), який прослуховує інший користувач, що буде обробляти цей запит в якості сервера

Основний потік подій	1) Зчитуються отримані дані із графічного інтерфейсу 2) Будується запит 3) Відбувається спроба надсилання запиту або виняткова ситуація 1
Виняткові ситуації	1) Користувач, який мав обробляти запит як сервер не є активним у поточний момент часу.
Примітки	Запит можна зробити методами Get або Post, в залежності від цілей.

Таблиця 1.2 – Сценарій використання «Отримати відповідь»

Назва	Отримати відповідь
Передумови	Було зроблено запит
Післяумови	Користувач отримає відповідь на свій запит певної сторінки
Сторони, що взаємодіють	Цей користувач як клієнт та інший користувач як сервер
Опис	Користувач отримує дані та обробляє відповідним чином.
Основний потік подій	1) Користувач отримує дані на порт, що він прослуховує 2) Дані ідентифікуються як відповідь по протоколу HTTP/1.1 або виняткова ситуація 1 3) Дані виводяться в вікно графічного інтерфейсу.

Виняткові ситуації	1) Дані не відповідають протоколу НТТР/1.1
Примітки	Відсутні

Таблиця 1.3 – Сценарій використання «Отримати запит»

Назва	Отримати запит
Передумови	Було зроблено запит певним користувачем на порт , що прослуховує даний користувач
Післяумови	Запит перейде до процедури обробки
Сторони, що взаємодіють	Цей користувач як сервер та інший користувач як клієнт
Опис	Сервер отримує дані
Основний потік подій	<p>1) Користувач отримує дані на порт, що він прослуховує</p> <p>2) Дані ідентифікуються як запит по протоколу НТТР/1.1 або виняткова ситуація 1</p> <p>3) Дані передаються до програмного забезпечення, що має їх обробляти.</p>
Виняткові ситуації	1) Дані не відповідають протоколу НТТР/1.1
Примітки	Відсутні

Таблиця 1.4 – Сценарій використання «Обробити запит»

Назва	Обробити запит
Передумови	Було отримано дані та ідентифіковано, що по протоколу НТТР/1.1 це запит.

Післяумови	Після обробки даних буде готова відповідь, яка потім буде відправлена клієнту
Сторони, що взаємодіють	Цей користувач як сервер та інший користувач як клієнт
Опис	Сервер обробляє дані, що отримав
Основний потік подій	<ol style="list-style-type: none"> 1) Запит розкладається на поля 2) Відбувається обробка відповідно до отриманих даних 3) Будується відповідь із відповідними даними та заголовками протоколу HTTP/1.1
Виняткові ситуації	Збій роботи сервера через велике навантаження
Примітки	На цьому етапі отримуємо готову HTTP-відповідь

Таблиця 1.5 – Сценарій використання «Надіслати відповідь»

Назва	Надіслати відповідь
Передумови	Було згенеровано сервером відповідь по протоколу HTTP/1.1, на певний конкретний запит.
Післяумови	Певний клієнт отримає відповідь на свій запит.
Сторони, що взаємодіють	Цей користувач як сервер та інший користувач як клієнт
Опис	Сервер відправляє дані певному клієнту.

Основний потік подій	1) Відбувається спроба надсилання відповіді або виняткова ситуація 1
Виняткові ситуації	1) Користувач, який мав отримати відповідь як клієнт не є активним у поточний момент часу.
Примітки	Відсутні

1.5. Концептуальна модель системи

Проект складається із двох модулів : користувача(P2P клієнта) , та мережі, за допомогою якої обмінюються даними користувачі. Мережа є досить простим модулем та складається із з'єднань, що створили користувачі.

P2P учасник є багаторівневим модулем та складається із : рівня доступу до даних, рівня бізнес-логіки та рівня інтерфейсного зв'язку із користувачем(зовнішнім, людиною, яка буде використовувати застосунок).

Основні сутності та інтерфейси рівня доступу до даних наведені на рис. 1.2

В ході роботи проекту, під час запиту певної сторінки, потрібно отримати дані, що містяться в шаблоні сторінки . Для цього використовується паттерн проектування DAO(data access object)[4]. Інтерфейс DAO декларує метод отримання даних.

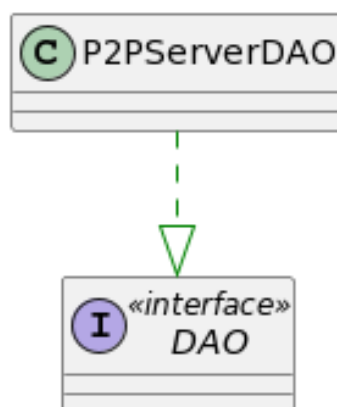


Рисунок 1.2 – Основні сутності та інтерфейси рівня доступу до даних

Клас P2PServerDAO розширює інтерфейс DAO та визначає, як саме будуть отримуватись дані. У даній курсовій роботі використовується база даних для зберігання всієї необхідної інформації. Клас P2PServerDAO містить метод `getData`, який виконує читання шаблонів сторінок із бази даних.

Також у базі даних зберігається статистика відвідувань сторінок. Клас P2PServerDAO містить методи для отримання кількості переглядів певної сторінки по адресі та додавання статистики.

Рівень бізнес-логіки містить в собі класи для взаємодії із запитами та відповідями, обробки сторінок та ведення статистики.

Основні сутності та інтерфейси, що стосуються створення та обробки запитів наведені на рис. 1.3

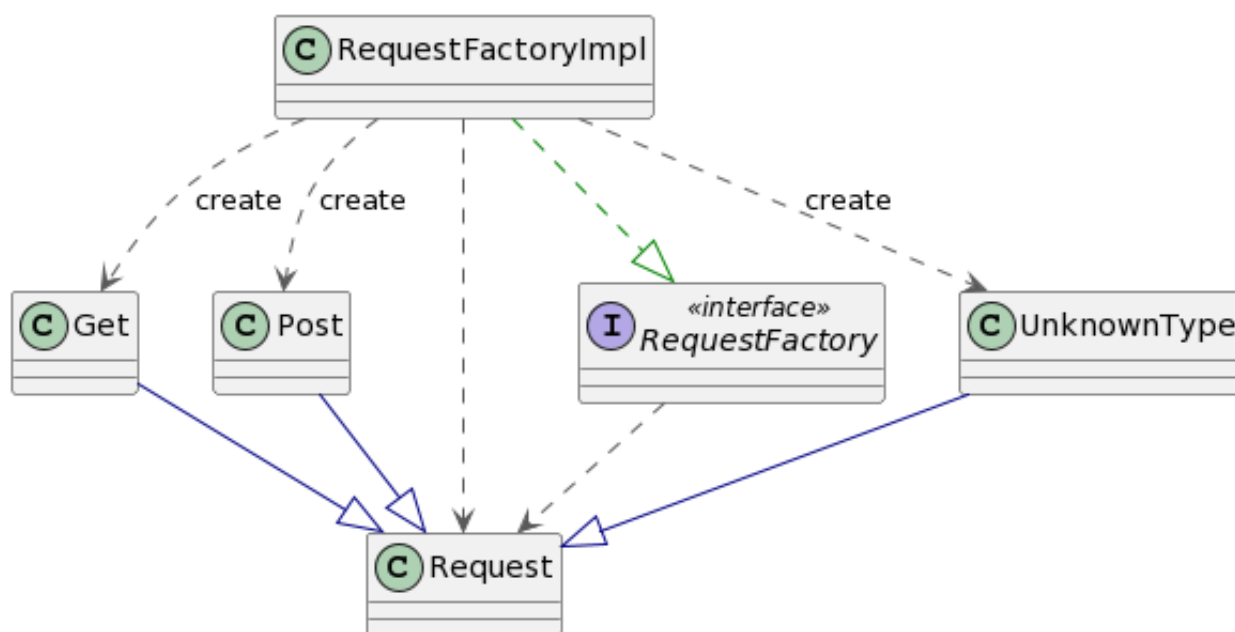


Рисунок 1.3 – Основні сутності та інтерфейси, що стосуються створення та обробки запитів

У курсовій роботі використовуються три конкретні типи запитів GET, POST та невідомий тип для випадків, коли тип запиту відрізнявся від двох інших. Вони розширюють базовий клас запиту Request, в якому відбувається розбиття запиту на поля. У конкретних типах запиту відбуваються конкретні дії, які потрібно зробити при певному запиті. Для визначення типу запиту із даних, що було отримано,

використовується шаблон проектування «Фабричний метод». Він реалізований у класі `RequestFactoryImpl` та імплементує загальний інтерфейс для фабричних методів запиту `RequestFactory`.

Основні сутності та інтерфейси, що стосуються взаємодії із відповідями наведені на рис. 1.4

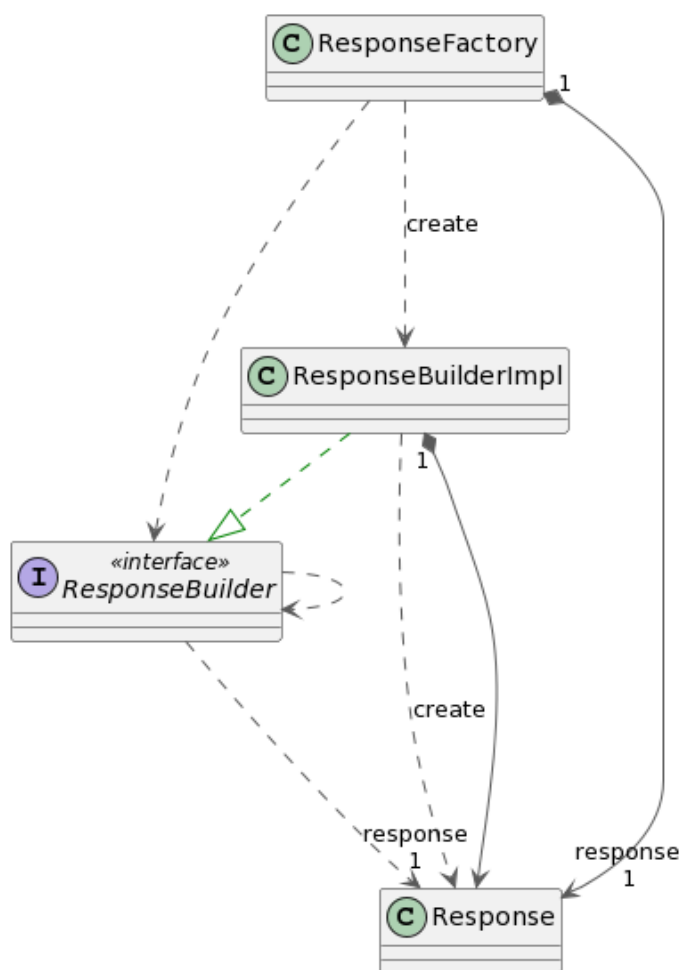


Рисунок 1.4 – Основні сутності та інтерфейси, що стосуються взаємодії із відповідями

За допомогою класів, зображених на рис 1.4 будується відповідь від сервера по протоколу HTTP1/1. У класі `ResponseFactory` відбувається створення декількох типових відповідей (все завершилось успішно, сторінку не було знайдено, проблеми із сервером), відповідно до обробки запиту. Відповіді створюються за допомогою

класу `ResponseBuilderImpl`, що реалізує інтерфейс `ResponseBuilder` та шаблон проектування «Будівельник». Клас `Response` – це основний об'єкт відповіді. Він містить перелік полів, необхідних для побудови відповіді та методи для взаємодії з ними.

Основні сутності та інтерфейси, що стосуються обробки сторінок та взаємодії з ними наведені на рис. 1.5 та 1.6

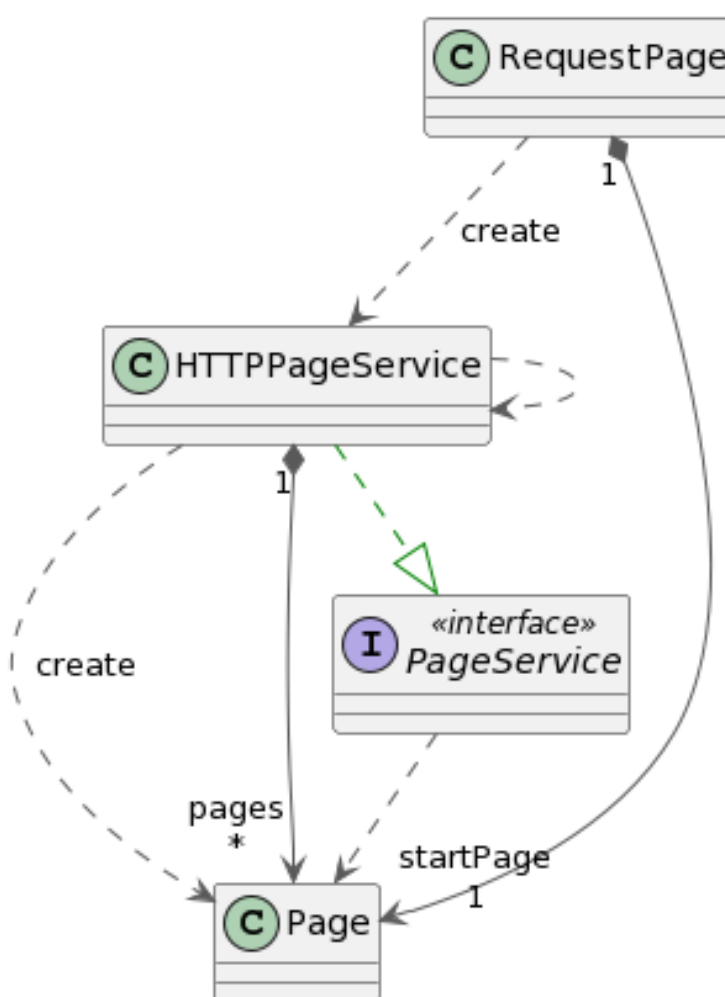


Рисунок 1.5 – Основні сутності та інтерфейси, що стосуються обробки сторінок

Обробка сторінки є досить ресурсно затратною операцією. Особливо, якщо сторінка динамічна. Адже кожна сторінка повинна бути інтерпретована, скомпільована, завантажена, під кожен сторінку мають бути створені окремі файли. Під час обробки сторінки, є операції, які можна виконати один раз, зберегти дані, а

потім із цієї точки продовжувати обробляти кожен запит певної сторінки індивідуально до параметрів та потреб. Саме ці дані містяться в класі Page. До кожної сторінки, що міститься в базі даних, прив'язаний один екземпляр класу Page. Тут зберігають дані, які скоріше за все не будуть змінюватись часто, а тому їх можна обробити один раз, а не при кожному запиті певної сторінки. Це зберігає час при обробці сторінки, адже тепер кожна сторінка буде компілюватись, завантажуватись, діставатись із бази даних лише один раз, всі дані будуть зберігатись в відповідному екземплярі класу Page. При кожному новому запиті на сторінку на певні місця будуть лише вставлятись передані параметри .

HTTPPageService це клас інтерфейсу PageService, який контролює, щоб до кожної сторінки із бази даних було створено не більше одного екземпляру класу Page. Також через цей клас здійснюється доступ до об'єктів класу Page. Клас Page може бути створений лише через HTTPPageService.

Екземпляри класу RequestPage прив'язані до запиту, та містять дані, що будуть часто змінюватись та для кожного запиту будуть унікальними. Це, наприклад, параметри, які потрібно додати при обробці сторінки. Екземпляр класу RequestPage бере оброблені до певного моменту дані із екземпляру класу Page через HTTPPageService та продовжує їх обробку унікально для певного запиту.

Обробка сторінки складається із декількох етапів (рис.1.6). Обробка залежить від внутрішнього стану сторінки, а саме – від типу : статична чи динамічна. Всі стани, в яких перебуває сторінка під час обробки належать до загального інтерфейсу DynamicPageState.

Першим станом, з якого починається обробка, є ServiceState. У цьому стані ми перевіряємо: задану сторінку ми обробляємо вперше, чи раніше уже її обробляли. Якщо сторінка відвідувана вперше – то наступним станом буде ReceivingDataState. Інакше, якщо сторінка уже була відвідана хоча б один раз , то : якщо сторінка динамічна - InsertParametersState(), якщо статична - FinishState().

У ReceivingDataState відбувається отримання даних шаблону сторінки із бази даних. Якщо сторінка динамічна, то наступним етапом буде

Через ці всі стани проходять лише екземпляри класу Page, що відповідають динамічним сторінкам.

Лише у наступному стані обробки - `InsertParametersState()` - бувають екземпляри класу `RequestPage`, де відбувається обробка сторінки із врахуванням отриманих параметрів. Попередні етапи обробки не виконують для `RequestPage`. Екземпляри класу `RequestPage` беруть дані про попередні етапи в відповідного екземпляру класу `Page`. Це відбувається для економії ресурсів та часу.

Останній етап `FinishState()`, сюди потрапляють лише повністю оброблені та готові до відправлення сторінки .

Також у проекті присутні класи для роботи з конфігурацією, що реалізовані за допомогою шаблону проектування «Одинак». Це клас `Configuration`, що являє собою безпосередньо дані із файлу конфігурації, та клас доступу `ConfigurationManager`.

Клас `Statistic` реалізує всю необхідну логіку для ведення статистики. Безпосередньо самі дані статистики по кожній існуючій сторінці зберігаються в базі даних.

Рівень зв'язку із користувачем(зовнішнім, людиною, яка буде використовувати застосунок) реалізовано за допомогою графічного інтерфейсу. Діаграма класів цього рівня зображена на рис. 1.7

Всі ці класи відповідають за «клієнта» , та потрібні для читання даних із форм графічного інтерфейсу та побудови і відправки запиту/відображення отриманої відповіді .

Клас `Client` є точкою запуску клієнтів. У ньому відбувається завантаження файлів конфігурації та запуск `ClientFrame`.

Клас `ClientListener` інтерфейсу `ConnectionListener` є посередником між користувачем і графічним інтерфейсом та відповідає за зв'язок між ними.

`ClientFrame` є основним класом, який відповідає за вікно графічного представлення та взаємодію із даними. Тут знаходяться складові графічного інтерфейсу, інформація про їхнє розташування, `ActionListener`, який запускає

виконання певних дій при натисканні кнопки на формі, та класи для взаємодії із отриманими даними від користувача.

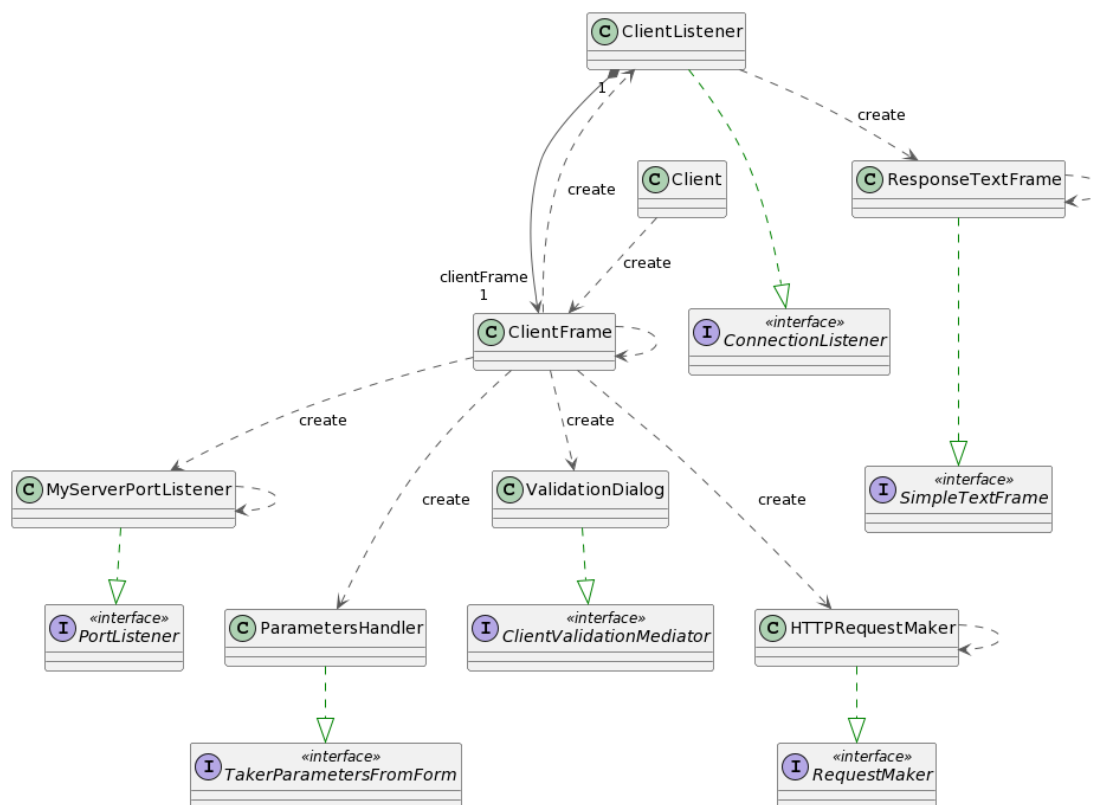


Рисунок 1.7 – Діаграма класів рівня взаємодії з користувачем

При натисканні кнопки на формі відбувається відправка певного повідомлення по протоколу HTTP/1.1 на потрібний порт. Щоб «скласти» це повідомлення, були використані наступні класи та інтерфейси .

Клас HTTPRequestMaker інтерфейсу RequestMaker повертає стрічку із уже готовим запитом, але для цього в конструктор класу потрібно передати метод, адресу сторінки та параметри.

Клас ParametersHandler інтерфейсу TakerParametersFromForm займається співставленням даних із форм, в результаті чого отримуємо HashMap із параметрами в форматі ключ-значення .

Клас ValidationDialog інтерфейсу ClientValidationMediator займається валідацією форма. Він є посередником між різними форматами та інформаційними

рядками, вказуючи, який напис потрібно відобразити в певному рядку в залежності від дій та ситуації.

Клас `ResponseTextFrame()` інтерфейсу `SimpleTextFrame` потрібен для відображення поля із відповіддю від сервера, яку отримує клієнт після запиту.

Клас `MyServerPortListener` інтерфейсу `PortListener` потрібен, щоб запустити прослуховування певного порту на даному клієнті та зробити його активним сервером, який може приймати запити.

Загальна діаграма класів наведена в Додатку А

1.6. Вибір бази даних

При виборі бази даних було вирішено використати PostgreSQL. PostgreSQL – це популярна об'єктно-реляційна система управління базами даних, що базується на мові SQL. Основна перевага PostgreSQL полягає в тому, що вона має відкритий код. PostgreSQL підтримує розширені типи даних та функції оптимізації продуктивності, які доступні лише у дорогих комерційних базах даних, таких як Oracle та SQL Server.

До основних особливостей PostgreSQL можна віднести :

- Сумісність із різними платформами
- Підтримка управління кількома версіями одночасності
- Відповідність стандарту ANSI SQL
- Повна підтримка архітектури мережі клієнт-сервер
- Реплікація SSL на основі журналу та тригера
- Висока доступність
- Об'єктна-орієнтованість
- Підтримка JSON дозволяє зв'язуватися з іншими сховищами даних, такими як NoSQL
- PostgreSQL може запускати динамічні веб-сайти та веб-програми як опцію стека LAMP

- Ведення журналу PostgreSQL з можливістю запису, що робить базу даних дуже стійкою до збоїв
- Вихідний код PostgreSQL знаходиться у вільному доступі
- PostgreSQL підтримує географічні об'єкти, це дає можливість використовувати його для служб, що базуються на розташуванні, та геоінформаційних систем
- PostgreSQL простий у використанні
- Низький рівень обслуговування та адміністрування як для вбудованого, так і для корпоративного використання PostgreSQL
- Що стосується показників продуктивності, PostgreSQL є повільнішою, ніж MySQL.

Враховуючи всі переваги і недоліки, можна сказати, що PostgreSQL – це потужна система управління базами даних із відкритим кодом, яка чудово підходить як для маленьких проєктів, так і для великих корпоративних застосунків.

1.7. Вибір мови програмування та середовища розробки

Під час вибору мови програмування, було вирішено реалізовувати проєкт на Java, адже вона має всі необхідні можливості, для реалізації проєкту, є об'єктно-орієнтованою та зручною.

В мові програмування Java досить зручно та просто працювати із Socket та ServerSocket, взаємодія яких є основою проєкту. Сокет — це просто кінцева точка для зв'язку між запущеними програмами (машинами). Клас Socket можна використовувати для створення сокета. Клас ServerSocket можна використовувати для створення серверного сокета. Цей об'єкт використовується для прослуховування певного порту та взаємодії із клієнтськими сокетами. [5] .

Java - багатопотокова мова, яка містить інструменти для зручного розпаралелювання роботи. Багатопотокові програми мають значні переваги у

порівнянні із тими програмами, що працюють в єдиному потоці, наприклад, швидкість, можливість паралельних обчислень.

Також причинами вибору саме цієї мови програмування було те, що Java — об'єктно-орієнтована мова, що підтримує принципи ООП, тобто концепцію наслідування, абстракції та поліморфізму. Синтаксис мови є досить простим та зрозумілим. Також Java має досить велику колекцію бібліотек та різних плагінів, які можуть значно спростити роботу.[6]

Є різні середовища розробки для платформи Java, найпопулярніші із них, це Eclipse, NetBeans, JetBrains IntelliJ IDEA та середовища розробки, що на них базуються. Під час розробки було вирішено використовувати JetBrains IntelliJ IDEA, через певні переваги середовища.

JetBrains IntelliJ IDEA має зручний інтерфейс, якісний відладник, зручну систему тестування, статичний аналізатор коду, що допомагає писати код та зменшує кількість помилок в ньому.

1.8. Проєктування розгортання системи

Система буде розгорнута в операційній системі Windows 10 за допомогою віртуальної машини Java (Java Virtual Machine). Проєкт містить файли конфігурації та зв'язок із базою даних через JDBC та за допомогою компонентів DAO, що реалізовані в проєкті.

Один із одним клієнти будуть спілкуватись по протоколу HTTP/1.1 за допомогою сокетів. Спілкування із користувачем відбувається за допомогою графічного інтерфейсу.

Діаграма розгортання системи зображена на рис. 1.8

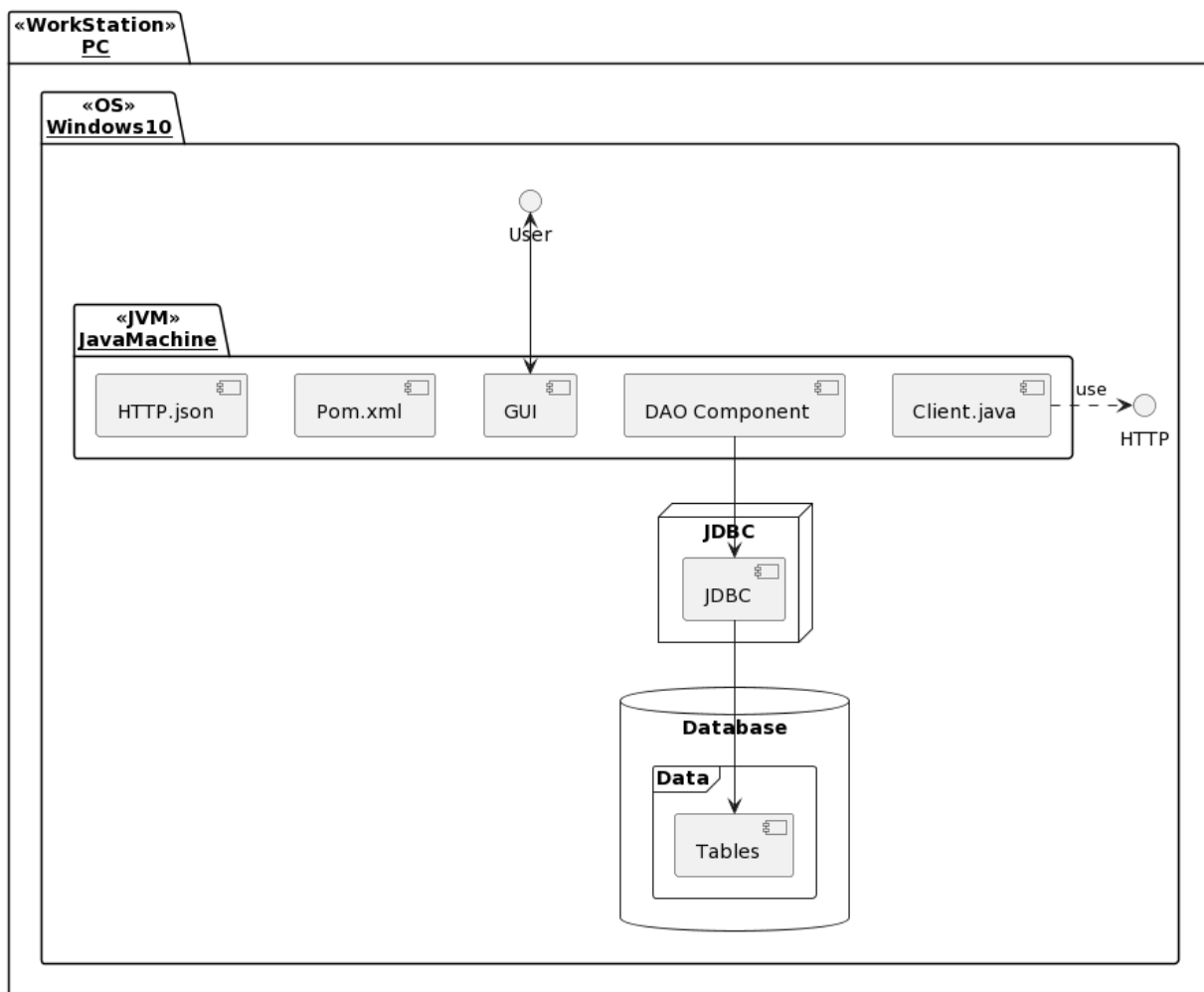


Рисунок 1.8 – Діаграма розгортання проєкту

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

Проект курсової роботи являє собою HTTP-сервер, який вміє «спілкуватись» по протоколу HTTP/1.1, обробляти динамічні та статичні сторінки, вести статистику. Фактично – це сервер, на який можна додати веб-сторінки, або цілі веб-сайти, що будуть мати взаємодію із базами-даних, в залежності від їхньої предметної області та призначення.

Сервер розроблений як система, яка отримує запит, опрацьовує його та повертає готову відповідь, тому проект курсової роботи не має якоїсь занадто складної структури бази даних.

Для реалізації курсової роботи була розроблена база даних, яка містить в собі шаблони сторінок, які може обробити сервер при необхідності та дані статистики.

За правилами проектування бази даних, всі ці поля можна помістити в одну таблицю `pages_data` (рис. 2.1)




Table:	pages_data		Existing
Comment:			
Columns (3)	Keys (1)	Indexes (1)	Foreign Keys
<div> <div>+</div> <div>-</div> <div>▲</div> <div>▼</div> </div>			
url	varchar	-- part of primary key mapped to url	
template_data	varchar	mapped to template_data	
number_of_views	integer	mapped to number_of_views	
<div> <div>Data Preview</div> <div>DDL Preview</div> </div>			
 url	 template_data	 number_of_views	
1 /1/index.html	<!doctype html><html lang="en">	0	
2 /2/a.html	<!DOCTYPE html>	0	
3 /2/4/b.html	<!DOCTYPE html>	0	
4 /3/q.html	<!DOCTYPE html>	0	
5 /1/test1.jsp	<!DOCTYPE html><html lang="en"><hea...	27	

Рисунок 2.1 – Таблиця `pages_data`

Оскільки при отриманні сторінки за її адресою вміст сторінки має однозначно визначатись, було вирішено встановити адресу сторінки (url), як первинний ключ, який не може бути нулем та є унікальним .

Стовпець `template_data` містить інформацію про шаблони сторінок для відповідної адреси, а стовпець `number_of_views` – кількість переглядів певної сторінки. Структура таблиці бази даних зображена на рисунку 2.2

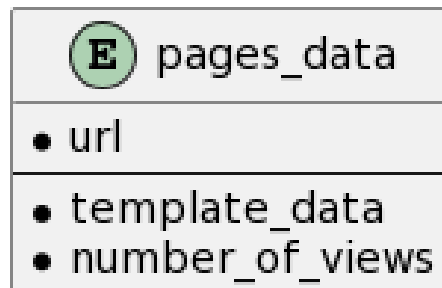


Рисунок 2.2 - Структура таблиці бази даних

2.2. Архітектура системи

2.2.1. Специфікація системи

Система розроблена за P2P архітектурою, тобто кожен учасник є рівноправним користувачем системи . Він одночасно може виступати як в ролі серверу, так і в ролі клієнта, тобто кожен учасник може як робити запити на певні сторінки, так і обробляти запити, що надійшли до нього. Графічно P2P архітектура зображена на рисунку 2.3

Мережі P2P мають багато переваг. Наприклад, відсутність центрального сервера , який потрібно обслуговувати. Також перевагою є те, що немає єдиної точки відмови, якщо один вузол залишає мережу, це мінімально впливає на всю систему. Якщо велика група однорангових пристроїв приєднується до мережі одночасно, мережа може легко впоратися зі збільшеним навантаженням. Також завдяки своїй

децентралізованій природі мережі P2P можуть досить добре витримувати хакерські атаки, оскільки відсутній централізований сервер.

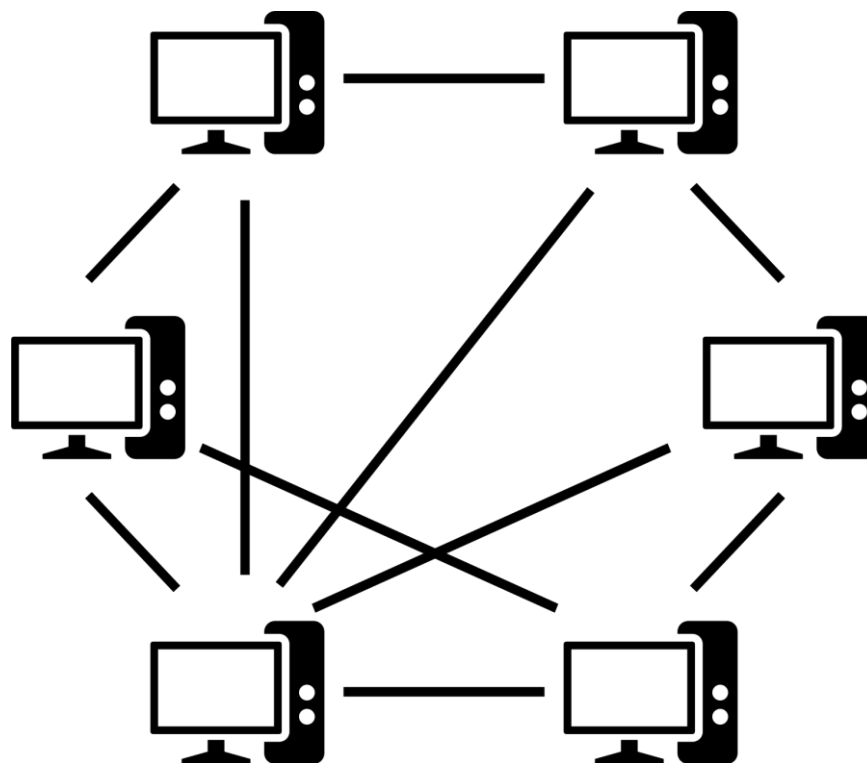


Рисунок 2.3 – Схема компонентів розроблюваної системи

Як і будь-яка мережева архітектура, P2P системи мають свої недоліки. В мережах P2P можуть з'являтися проблеми із безпекою. Якщо один учасник заражений вірусом та розповсюджує певні файли в мережі, які містять вірус, то вірус може швидко поширитися на інших учасників. Також в P2P мережах виникають проблеми із синхронізацією файлів та методами пошуку нових учасників.[7]

2.2.2. Вибір та обґрунтування патернів реалізації

Шаблони проєктування – це уже готові схеми для вирішення певних проблем, які часто зустрічаються в розробці ПЗ. Патерни проєктування не є готовими рішеннями, які можна трансформувати безпосередньо в код, а представляють

загальний опис вирішення проблеми, який можна використовувати в різних ситуаціях.[8].

При розробці проєкту були застосовані такі патерни, як «State», «Builder», «Factory Method», «Singleton».[9]

Патерн «State» дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану. Ідея полягає в тому, що програма може знаходитися в одному з декількох станів, які весь час змінюють один одного. Набір цих станів, а також переходів між ними, зумовлений і кінцевий. Перебуваючи в різних станах, програма може по-різному реагувати на одні і ті ж події, які відбуваються з нею. Патерн «State» пропонує створити окремі класи для кожного стану, в якому може перебувати об'єкт, а потім винести туди поведінку, яка відповідає цим станам. Замість того, щоб зберігати код всіх станів, початковий об'єкт, званий контекстом, буде містити посилання на один з об'єктів-станів і делегувати йому роботу, залежну від стану. Завдяки тому, що об'єкти станів матимуть загальний інтерфейс, контекст зможе делегувати роботу станам, не прив'язуючись до його класу. Поведінка контексту можна буде змінити в будь-який момент, підключивши до нього інший об'єкт-стан.[10]

При виконанні курсової роботи патерн «State» був застосований при реалізації динамічних web-сторінок. Життєвий цикл jsp сторінки включає досить багато станів, які переходять з одного в інший. Для спрощення коду було створено окремі класи для кожного стану, в яких може перебувати об'єкт, та винесено всю логіку, пов'язану із певним станом в самі класи станів. Завдяки цьому код стає значно краще виглядати, виконується принцип єдиного обов'язку, код не нагромаджується в одному місці. Діаграма класів, оптимізованих за допомогою патерну «State», зображена на рис.2.4

Першим станом, з якого починається обробка, є ServiceState. У ньому ми перевіряємо, чи задану сторінку ми обробляємо вперше, чи раніше уже її обробляли. Якщо сторінка відвідувана вперше – то наступним станом буде ReceivingDataState.

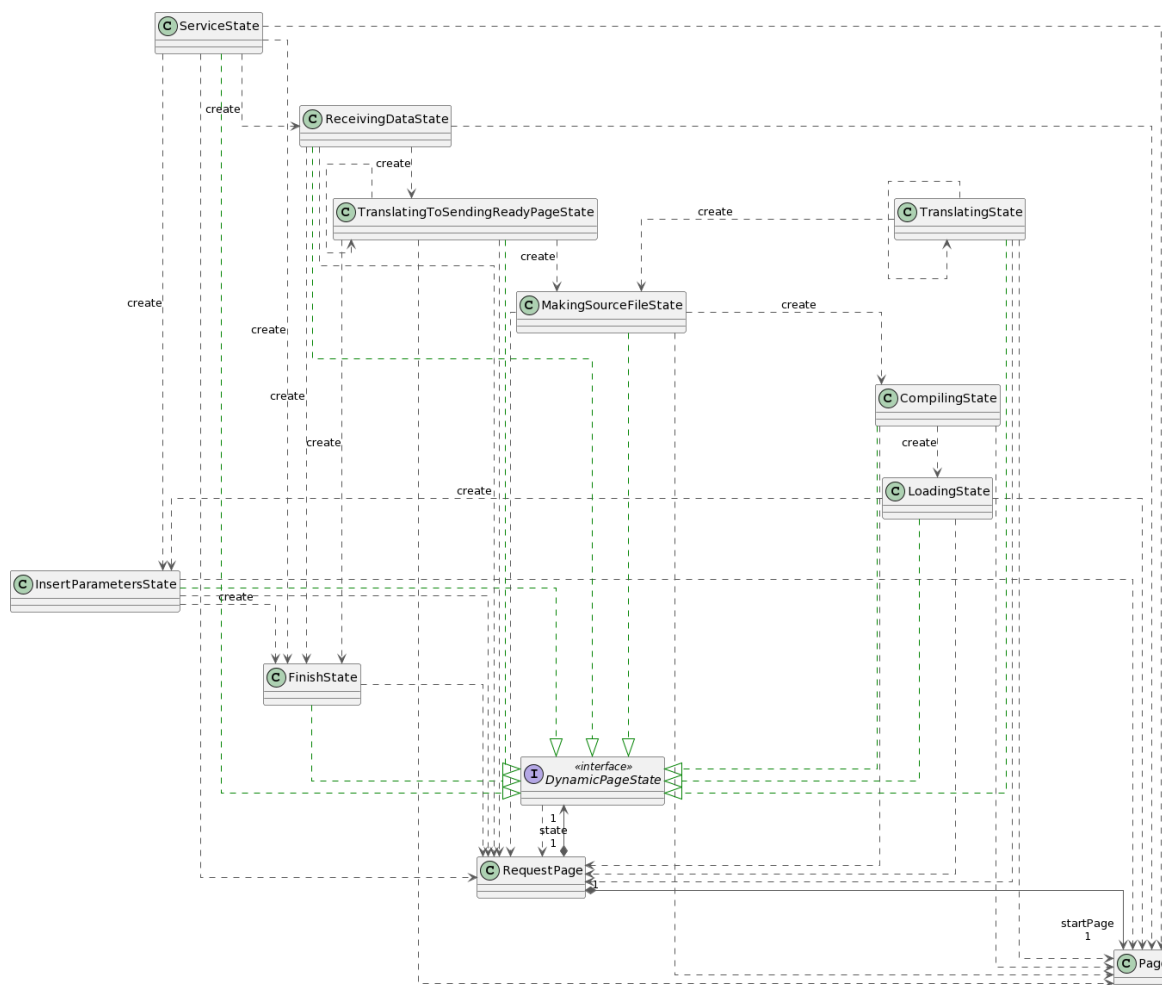


Рисунок 2.4 – Схема структурна компонентів патерну «стан»

Інакше, якщо сторінка вже була відвідана колись (а отже, уже оброблялась) - InsertParametersState(). Фрагмент коду зображено на рис.2.5

```

//якщо це перше відвідування сторінки
if (page.getStartPage().getTranslatedPage() == null) {
    page.changeState(new ReceivingDataState());
} else {
    //якщо це оновлення сторінки
    page.changeState(new InsertParametersState());
}
  
```

Рисунок 2.5 – Фрагмент коду класу ServiceState

У `ReceivingDataState` відбувається читання шаблону сторінки із бази даних. Якщо сторінка динамічна, то наступним етапом буде `TranslatingToSendingReadyPageState()`, в якому буде відбуватись інтерпретація даних.

Далі ідуть стани, які відповідають за обробку динамічних сторінок, `TranslatingToSendingReadyPageState` – за інтерпретацію, `MakingSourceFileState` – за створення файлу, який буде компілюватись та завантажуватись, `CompilingState` – за компіляцію, `LoadingState` – за завантаження.

У стані `InsertParametersState()` відбувається обробка сторінки із врахуванням отриманих параметрів. Останній етап `FinishState()`, сюди потрапляють лише повністю оброблені та готові до використання сторінки .

Мод для зміни стану містить сама сторінка, обробка якої й описана в станах. Сторінка містить метод для зміни свого поточного стану . Фрагмент коду зображено на рис.2.6

```
public void changeState(DynamicPageState state) {
    this.state = state;
    state.doAction(this);
}
```

Рисунок 2.6 – Фрагмент коду класу Page

Який стан буде наступним – визначається в поточному стані сторінки .

Всі стани імплементують загальний функціональний інтерфейс `DynamicPageState` та визначають, як буде працювати єдиний метод цього інтерфейсу.

Наступним патерном, що був використаний при розробці архітектури проекту став «Будівельник». Шаблон "Builder" (будівельник) використовується для відділення процесу створення об'єкту від його представлення. Це доречно у випадках, коли об'єкт має складний процес створення (наприклад, Web- сторінка як елемент повної відповіді web- сервера) або коли об'єкт повинен мати декілька різних форм створення[11].

При реалізації проекту патерн «Будівельник» був застосований при створенні відповіді від серверу. Оскільки відповідь від сервера за протоколом HTTP може бути дуже громісткою, мати різну конфігурацію полів, був застосований даний патерн. Він значно оптимізує код і покращує його структуру.

Діаграма класів, оптимізованих за допомогою патерну «Будівельник», зображена на рис.2.7

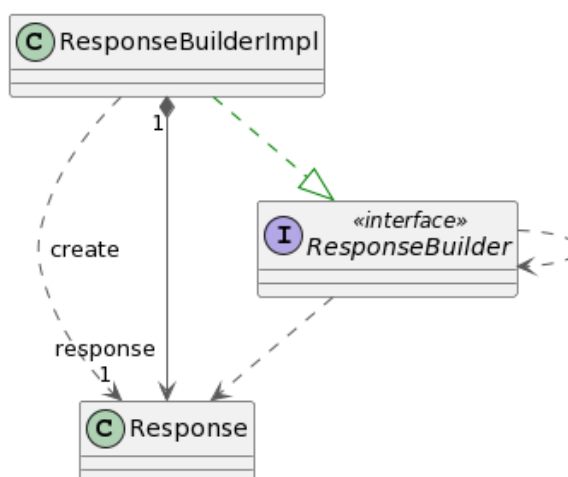


Рисунок 2.7 – Схема структурна компонентів патерну «будівельник»

```

@Override
public ResponseBuilder setStatusText(String statusText) {
    this.response.setStatusText(statusText);
    return this;
}

@Override
public ResponseBuilder setFileData(String fileData) {
    this.response.setFileData(fileData);
    return this;
}

@Override
public synchronized Response build() {
    return this.response;
}

```

Рисунок 2.8 – Фрагмент коду класу ResponseBuilderImpl

Інтерфейс `ResponseBuilder` описує основні методи, за допомогою яких можна встановлювати значення в певні поля до певної відповіді від серверу. `ResponseBuilderImpl`, що імплементує цей інтерфейс, визначає, як саме методи будуть працювати. Також він містить метод `build`, який повертає готовий об'єкт класу `Response`. Фрагмент коду зображено на рис.2.8

Наступним патерном, що було застосовано, став «Factory Method». Шаблон "фабричний метод" визначає інтерфейс для створення об'єктів певного базового типу. Це зручно, коли хочеться додати можливість створення об'єктів не базового типу, а деякого дочірнього. Фабричний метод у такому разі є зачіпкою для впровадження власного конструктора об'єктів. Основна ідея полягає саме в заміні об'єктів їх підтипами, що при цьому зберігає ту ж функціональність; інша частина поведінки об'єктів не є інтерфейсною (`AnOperation`) і дозволяє взаємодіяти із створеними об'єктами як з об'єктами базового типу. Тому шаблон "фабричний метод" носить ще назву "Віртуальний конструктор".

Патерн Фабричний метод пропонує відмовитись від безпосереднього створення об'єктів за допомогою оператора `new`, замінивши його викликом особливого фабричного методу. Об'єкти все одно будуть створюватися за допомогою `new`, але робити це буде фабричний метод.

Щоб ця система запрацювала, всі об'єкти, що повертаються, повинні мати спільний інтерфейс. Підкласи зможуть виготовляти об'єкти різних класів, що відповідають одному і тому самому інтерфейсу.[12]

При реалізації курсової роботи патерн «Factory Method» був застосований при обробці вхідного запиту до серверу. Найпростіший сервер повинен вміти обробляти GET та POST запити. Але в загальному їх існує досить багато.

Користь цього патерну в тому, що якщо раптом знадобиться розширити перелік запитів, які може обробляти сервер, за допомогою патерну «Factory Method» це легко і безпроблемно можна реалізувати. Всі класи, які працюють із запитом, не прив'язані тепер до якогось конкретного запиту, а працюють лише з батьківським класом всіх конкретних.

Діаграму класів, що реалізують даний патерн, можна побачити на рисунку 2.9

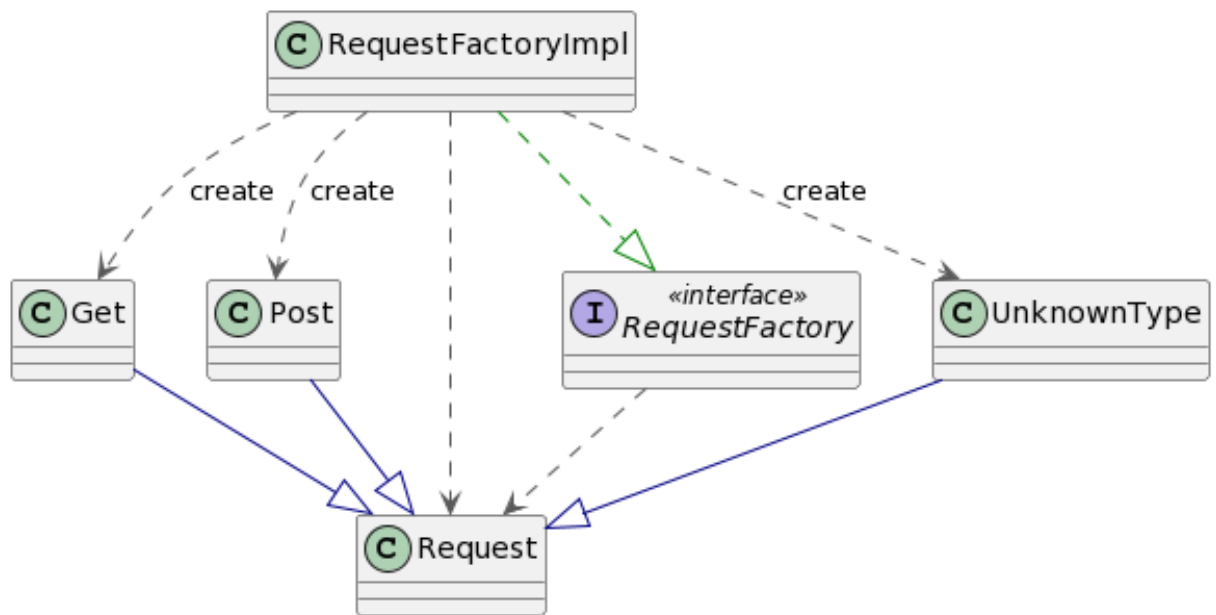


Рисунок 2.9 – Схема структурна компонентів патерну «фабричний метод»

```

public synchronized Request initialize() {

    //якщо запит був зроблений за допомогою методу GET
    if (method.equals("GET"))
    {
        return new Get(data);
    }

    //якщо запит був зроблений за допомогою методу POST
    if (method.equals("POST"))
    {
        return new Post(data);
    }

    //якщо запит був зроблений за допомогою методу,
    // оброблення якого не описане (методу, відмінного від GET та POST)
    else return new UnknownType(data);
}
  
```

Рисунок 2.10 – Фрагмент коду класу RequestFactoryImpl

Всі запити до серверу мають однакову структуру, структуру прописану в HTTP протоколі. На основі цієї структури був створений абстрактний запит. Оскільки GET та POST мають різне призначення, але однакову структуру, було використано патерн «Factory Method».

Клас `RequestFactoryImpl` імплементує інтерфейс `RequestFactory`. Сам фабричний метод виглядає наступним чином, як зображено на рис.2.10

Наступним паттерном, що було застосовано, став «Singleton». «Singleton» (Одинак) являє собою клас в термінах ООП, який може мати не більше одного об'єкта.[13]

При реалізації курсової роботи цей патерн був застосований при створенні класу конфігурації. Перед початком роботи програми завантажується файл конфігурації та створюється екземпляр класу. Потім цей екземпляр класу застосовується в тих місцях, де потрібні конфігураційні дані. Доступ до об'єкту конфігурації здійснюється за допомогою менеджера конфігурації, що надає доступ до неї. Фрагмент коду зображено на рис.2.11

```
public Configuration getCurrentConfiguration() {
    if (currentConfiguration == null) {
        throw new RuntimeException("Configuration not found");
    }
    return currentConfiguration;
}
```

Рисунок 2.11 – Фрагмент коду класу `ConfigurationManager`

Діаграму класів, що реалізують даний патерн, можна побачити на рисунку 2.12

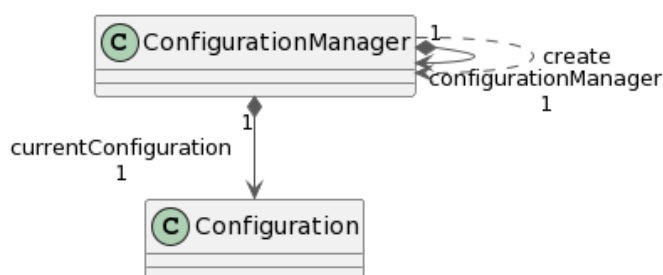
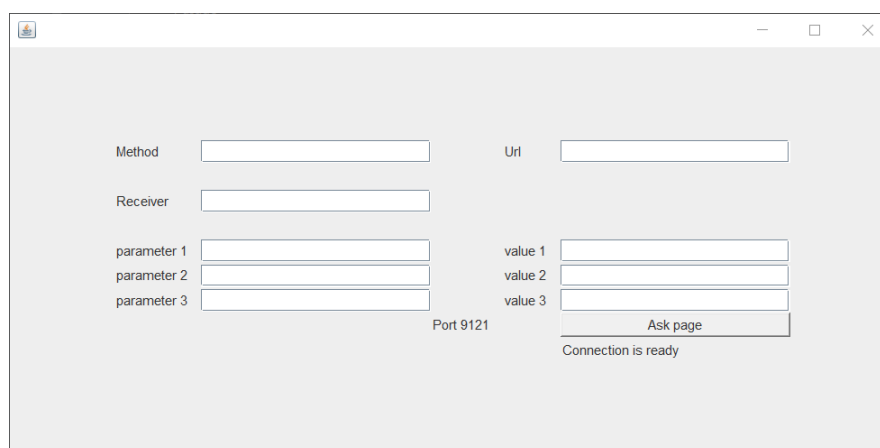


Рисунок 2.12 – Схема структурна компонентів патерну «одинак»

2.3. Інструкція користувача

Проект складається із 2 модулів: клієнтів та мережі. Клієнтів можна запустити декілька. Для запуску нового клієнта достатньо викликати метод `main` в класі `Client`. Новому клієнту автоматично буде вибрано випадковим чином порт, який він буде прослуховувати в якості серверу.

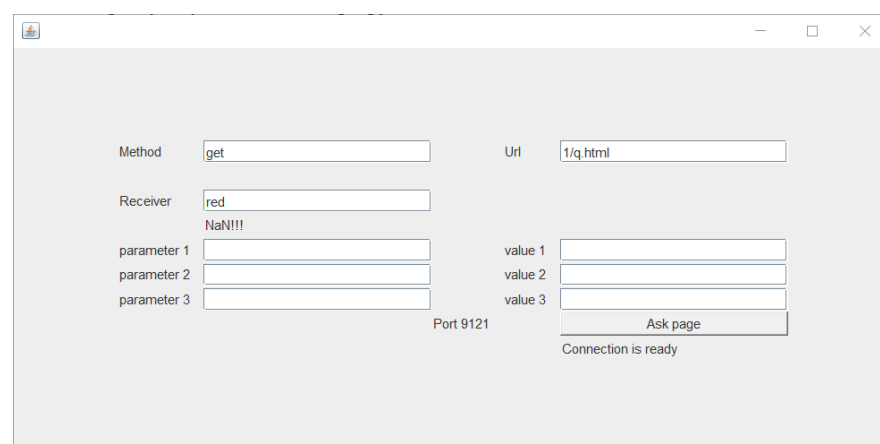
Система працює таким чином: через графічний інтерфейс передаються введені дані (рис.2.13), а саме, метод, який потрібно використати (`get` або `post`)(регістр не має різниці), `url` сторінки та порт, на який потрібно зробити запит. Дані валідуються (рис.2.14). Наступні поля – це параметри/значення, які потрібно передати. Також можна побачити номер порту даного реєр та кнопку для відправки даних.



The screenshot shows a GUI window with the following elements:

- Method:** An empty text input field.
- Receiver:** An empty text input field.
- parameter 1, parameter 2, parameter 3:** Three empty text input fields stacked vertically.
- Uri:** An empty text input field.
- value 1, value 2, value 3:** Three empty text input fields stacked vertically.
- Port 9121:** A label indicating the port number.
- Ask page:** A button with the text 'Ask page'.
- Connection is ready:** A status message at the bottom right.

Рисунок 2.13 – Графічний інтерфейс



The screenshot shows the same GUI window as Figure 2.13, but with the following changes:

- Method:** Contains the text 'get'.
- Receiver:** Contains the text 'red'.
- parameter 1:** Contains the text 'NaN!!!', which is highlighted in red, indicating a validation error.
- Uri:** Contains the text '1/q.html'.
- value 1, value 2, value 3:** Still empty text input fields.
- Port 9121:** The label remains.
- Ask page:** The button remains.
- Connection is ready:** The status message remains.

Рисунок 2.14 – Приклад валідації одного із полів

В подальшому створюється з'єднання клієнта з сервером. Результати роботи HTTP -відповідь від сервера . Випадок із успішною процедурою зображено на рис.2.15; випадок, коли запитану сторінку не було знайдено, зображено на рис.2.16.

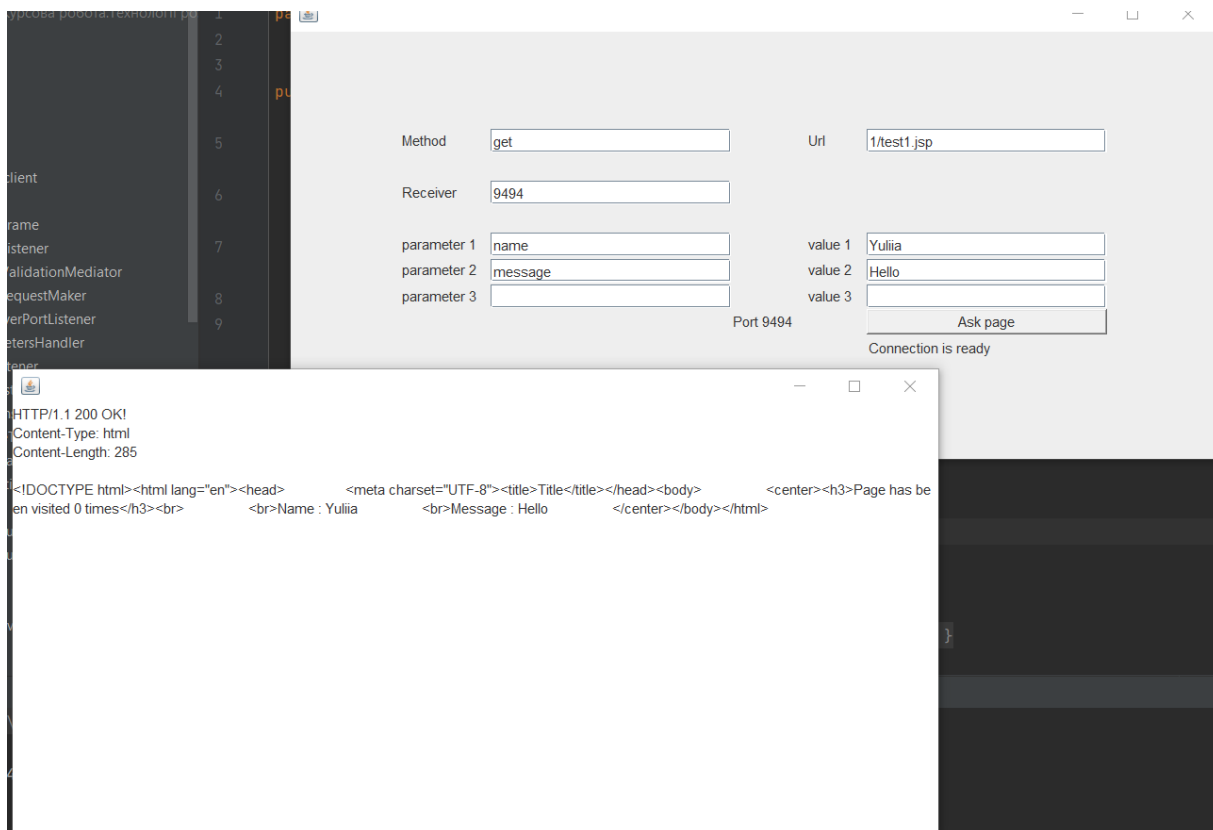


Рисунок 2.15 – Приклад успішного запиту

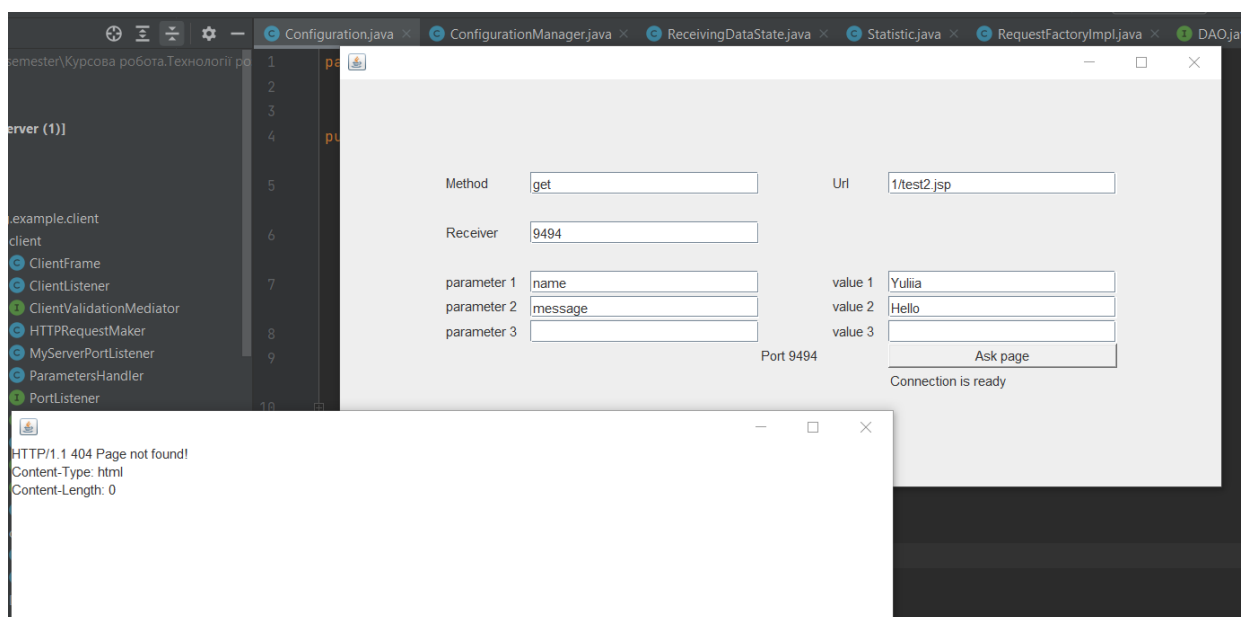


Рисунок 2.16 – Приклад запиту із кодом відповіді 404

ВИСНОВКИ

Під час написання роботи, були проаналізовані існуючі HTTP-сервери, оцінено їхні переваги та недоліки. Було вирішено розробити систему, яка буде спроектована за P2P архітектурою та дозволить користувачу просто та зручно використовувати можливості HTTP веб-серверу.

Першим кроком було сформульовано функціональні та нефункціональні вимоги до системи, що визначило очікувану поведінку системи. Наступним, обрано технології на яких буде написана система. Java була обрана в якості мови програмування, а в якості середовища розробки було обрано JetBrains IntelliJ IDEA , за його можливості та потужність, а також зручний інтерфейс для роботи.

Наступним кроком були описані сценарії використання. Система підтримує два види HTTP-запитів : get та post, може обробляти як статичні html - , так і генерувати «на льоту» динамічні jsp – сторінки. Серверна частина прослуховує певний порт, до якого можуть надіслати запити інші користувачі в якості клієнта. На кожен вхідний запит генерується відповідь в залежності від даних та запиту. Всі запити та відповіді, за допомогою яких і спілкуються користувачі, відповідають стандартам HTTP/1.1.

Далі було розроблено загальну архітектуру системи, враховано специфіку P2P архітектури.

Отже, підсумовуючи наведене вище, можна сказати, що основна специфіка створеної системи в простоті використання, та наявності всього необхідного функціоналу, а також те, що проект працює як P2P застосунок. Завдяки тому, що система має відкриту архітектуру, її розширення не є проблемою. Також графічний інтерфейс системи забезпечує зручний зв'язок застосунку із користувачем .

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. http://wiki.tneu.edu.ua/index.php?title=1_%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%B8_%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82
2. https://w3techs.com/technologies/overview/web_server
3. [https://uk.wikipedia.org/wiki/Swing_\(Java\)](https://uk.wikipedia.org/wiki/Swing_(Java))
4. https://en.wikipedia.org/wiki/Data_access_object
5. <https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html>
6. <https://mvnrepository.com/>
7. <https://www.enjoyalgorithms.com/blog/peer-to-peer-networks>
8. <https://refactoring.guru/design-patterns/what-is-pattern>
9. John Vlissides, Ralph Johnson, Richard Helm, Erich Gamma «Design Patterns: Elements of Reusable Object-Oriented Software»
10. <https://refactoring.guru/design-patterns/state>
11. <https://refactoring.guru/design-patterns/builder>
12. <https://refactoring.guru/design-patterns/factory-method>
13. <https://refactoring.guru/design-patterns/singleton>

