Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему «Розроблення програмного забезпечення для проектування, відладки та тестування API»

Виконав: студент групи ІПЗ19-2
Спеціальність 121 Інженерія програмного забезпечення
Свергун А.О.
(прізвище та ініціали)
Керівник к.е.н., доц. Яковенко Т. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Університет митної справи та фінансів
(місце роботи)
В.о. завідувача кафедри кібербезпеки та інформаційних технологій
(посада)
к.т.н., доц. Прокопович-Ткаченко Д.І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2023

АНОТАЦІЯ

Свергун А.О. Розроблення програмного забезпечення для проектування, відладки та тестування API.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справи та фінансів, Дніпро, 2023.

Ця кваліфікаційна робота спрямована на розробку програмного забезпечення для проектування, відладки та тестування API. Для досягнення цієї мети був сформульований перелік задач, які охоплюють аналіз існуючих методологій та інструментів, дослідження вимог розробки API, проектування інтерфейсу користувача, створення інструментів відладки та тестування.

Починаючи з аналізу існуючих методологій та інструментів, кваліфікаційна робота передбачає огляд та аналіз різних підходів, які використовуються у розробці, налагодженні та тестуванні API. Це дозволяє визначити переваги та недоліки кожного підходу і обрати найбільш підходящий для проекту. Дослідження вимог щодо розробки API включає вивчення потреб та обмежень розробників API, а також визначення функціональних та нефункціональних вимог до програмного забезпечення.

Ця кваліфікаційна робота є комплексним підходом до розробки програмного забезпечення для проектування, відладки та тестування API. Вона охоплює всі необхідні аспекти розробки, починаючи з аналізу існуючих методологій і інструментів до тестування та оцінки результатів. Результати цієї роботи можуть бути використані для поліпшення процесу розробки API, забезпечення якості та зручності використання програмного забезпечення розробниками API.

Ключові слова: методології, інструменти, розробка, відладка, тестування, API, інструменти відладки, мова програмування Java, бібліотека Swing, JavaFX.

ABSTRACT

Svergun A. Software development for API design, debugging and testing.

Qualification work for a bachelor's degree in specialty 121 «Software Engineering». – University of Customs and Finance, Dnipro, 2023.

This qualification work is aimed at developing software for designing, debugging and testing APIs. To achieve this goal, a list of tasks was formulated, which include analyzing existing methodologies and tools, researching API development requirements, designing the user interface, creating debugging and testing tools.

Starting with the analysis of existing methodologies and tools, the qualification work involves reviewing and analyzing various approaches used in API development, debugging, and testing. This allows you to determine the advantages and disadvantages of each approach and choose the most suitable one for the project. API development requirements research includes studying the needs and constraints of API developers, as well as identifying functional and non-functional software requirements.

This qualification is a comprehensive approach to software development for API design, debugging, and testing. It covers all the necessary aspects of development, starting from the analysis of existing methodologies and tools to testing and evaluation of results. The results of this work can be used to improve the API development process, ensure the quality and usability of software by API developers.

Keywords: methodologies, tools, development, debugging, testing, API, debugging tools, Java programming language, Swing library, JavaFX.

ЗМІСТ

ВСТУП

У сучасному світі програмні інтерфейси набули величезного значення для взаємодії між різними додатками та системами. Вони відіграють ключову роль обміну даними та функціональності, забезпечуючи ефективну інтеграцію між різними компонентами програмного забезпечення. Однак розробка, відладка та тестування таких API може бути складним та трудомістким процесом, що потребує спеціалізованих інструментів та методологій.

Актуальність роботи полягає у кількох аспектах:

1) швидке зростання розробки API. З появою сучасних технологій, таких як хмарні обчислення, мобільні програми та Інтернет речей (IoT), розробка API стала ключовим компонентом для забезпечення взаємодії між різними програмами та системами. Зі зростанням числа API та їх значущості виникає потреба в ефективних інструментах та методах розробки, налагодження та тестування;

2) збільшення складності API. API стають все більш складними та різноманітними, включаючи RESTful API, GraphQL, мікросервіси та інші. Це призводить до необхідності поглибленого аналізу та проектування, а також ретельного тестування, щоб забезпечити їх надійність, безпеку та продуктивність;

3) значення якісного API. Якість API має прямий вплив на взаємодію користувача з додатками, а також на ефективність і стабільність систем в цілому. Добре спроектований, надійний і легко використовується API може значно покращити досвід користувачів та підвищити конкурентоспроможність додатків та сервісів;

4) поліпшення процесу розробки. Розробка програмного забезпечення для проектування, налагодження та тестування API дозволяє скоротити час, що витрачається на розробку та інтеграцію API, покращити ефективність роботи розробників та підвищити стабільність та надійність систем, що розробляються.

Всі ці фактори наголошують на важливості розробки програмного забезпечення, спеціалізованого для роботи з API. Робота в цій галузі має актуальність та потенціал для значного внеску у сучасну розробку програмного забезпечення.

Метою роботи є розроблення програмного забезпечення для проектування, відладки та тестування API.

Для досягнення поставленої мети роботи можуть бути сформульовані наступний перелік задач:

1) аналіз існуючих методологій та інструментів. Провести огляд та аналіз існуючих методологій, інструментів та підходів, що використовуються у розробці, налагодженні та тестуванні API. Вивчити їх переваги та недоліки, а також їх застосування для різних типів API;

2) дослідження вимог щодо розробки API. Вивчити вимоги та очікування розробників API, а також потреби та обмеження, з якими вони стикаються в процесі роботи. Визначити основні функціональні та нефункціональні вимоги до програмного забезпечення, яке буде розроблено;

3) проектування інтерфейсу користувача. Розробити інтуїтивно зрозумілий і зручний інтерфейс користувача, що дозволяє розробникам проектувати API і визначати його структуру, функціональність і параметри. Врахувати важливість чіткості та легкості розуміння інтерфейсу для підвищення продуктивності розробників;

4) створення інструментів налагодження та тестування. Розробити інструменти, які допоможуть розробникам виявляти та виправляти помилки в API, а також перевіряти його працездатність та продуктивність. Включити функції автоматичного тестування та перевірки відповідності API стандартам та специфікаціям;

5) інтеграція засобів розробки. Забезпечити інтеграцію розробленого програмного забезпечення з існуючими засобами розробки та популярними інструментами, такими як інтегровані середовища розробки (IDE), системи контролю версій та інструменти безперервної інтеграції та доставки (CI/CD).

Це дозволить розробникам легко інтегрувати створювані API у існуючі проекти та системи;

6) розробка документації та версіонування API. Розробити механізми для документування та версіонування API, щоб забезпечити ясність, стабільність та зворотну сумісність інтерфейсу для розробників-споживачів. Включити можливість автоматичного створення документації на основі визначень API;

7) тестування та оцінка результатів. Провести тестування розробленого програмного забезпечення, оцінити його ефективність та надійність, а також провести порівняльний аналіз з існуючими інструментами та методами розробки API. Виправити виявлені проблеми та покращити функціональність за результатами тестування.

Ці завдання допоможуть у досягненні мети роботи, а саме розробці програмного забезпечення, здатного полегшити процес проектування, налагодження та тестування API, підвищити якість інтерфейсів, що розробляються, і покращити досвід розробників.

Об'єктом дослідження є процеси роботи програмного забезпечення для проектування, відладки та тестування API.

Предметом дослідження є апаратно-програмне забезпечення для розроблення програмного забезпечення для проектування, відладки та тестування API.

Практичне значення одержаних результатів полягає у підвищенні якості проектування, відладки та тестування API.

Результатами роботи є програмне забезпечення для автоматизації проектування, відладки та тестування API.

Кваліфікаційна робота складається зі вступу, 3-х розділів, висновків, використаних джерел з 23 найменувань, 1-го додатка.

Обсяг роботи складається з 57 сторінок основного тексту з 75 сторінок кваліфікаційної роботи та 20 рисунків.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Визначення API та його ролі в сучасному програмному забезпеченні

Визначення API (Application Programming Interface) та його роль у сучасному програмному забезпеченні є ключовими поняттями, пов'язаними із взаємодією та інтеграцією різних компонентів та систем.

API є набір певних методів, функцій, протоколів і угод, які дозволяють додаткам взаємодіяти один з одним. API визначає, як різні компоненти програмного забезпечення можуть спілкуватися, обмінюватися даними та використовувати функціональність один одного.

Роль API у сучасному програмному забезпеченні може бути описана таким чином:

1) інтеграція додатків. API дозволяє різним додаткам та сервісам обмінюватися даними та функціональністю. Вони забезпечують механізми взаємодії між різними компонентами системи, дозволяючи їм працювати разом та використовувати можливості один одного. Завдяки API розробники можуть створювати додатки, які взаємодіють з іншими системами та сервісами, розширюючи їх функціональність та покращуючи користувальницький досвід;

2) створення розширень та плагінів. API надає можливість створювати розширення та плагіни, які можуть інтегруватися з основним програмним забезпеченням. Розробники можуть використовувати API для створення додаткових модулів, функцій чи можливостей, які можуть бути легко інтегровані з основною програмою або сервісом;

3) створення та використання сторонніх сервісів. API дозволяє розробникам створювати та надавати сторонні сервіси, які можуть бути використані іншими програмами та розробниками. Це відкриває можливості

для створення екосистеми сервісів та програм, які можуть взаємодіяти один з одним, обмінюватися даними та надавати додаткові функції та сервіси;

4) мобільна та веб-розробка. API відіграють важливу роль у розробці мобільних додатків та веб-сайтів. Вони дозволяють взаємодіяти з серверами, отримувати дані, надсилати запити та виконувати різні операції. API забезпечують інтеграцію із зовнішніми сервісами, платіжними системами, соціальними мережами та іншими важливими компонентами сучасних програм;

5) стандартизація та уніфікація. API сприяють стандартизації та уніфікації взаємодії між різними системами. Вони визначають загальні угоди та протоколи, які роблять взаємодію більш передбачуваною та зручною. Завдяки цьому розробники можуть створювати програми, які можуть працювати з різними сервісами та компонентами, використовуючи загальні стандарти та інтерфейси.

У програмному забезпеченні API стали невід'ємною частиною розробки. Вони дозволяють створювати складні та гнучкі системи, інтегрувати різні сервіси та розширювати функціональність додатків. API стимулюють інновації та дозволяють розробникам створювати більш ефективні та потужні програмні рішення.

1.2 Огляд різноманітних типів API

Огляд різних типів API, таких як RESTful, GraphQL та інші, є важливим аспектом дослідження в галузі розробки програмного забезпечення.

RESTful API (рис. 1.1) заснований на архітектурному стилі REST, який є набором принципів і обмежень для побудови розподілених систем. Він використовує протокол HTTP для взаємодії між клієнтом та сервером. У RESTful API ресурси представлені з допомогою унікальних URI (Uniform Resource Identifier), а операції над ресурсами виконуються з допомогою HTTP-методів, як-от GET, POST, PUT і DELETE.

Переваги RESTful API включають:

– простота та зрозумілість використання;

– масштабованість та гнучкість взаємодії;

– кешування даних для підвищення продуктивності;

– незалежність від мови програмування та платформи.

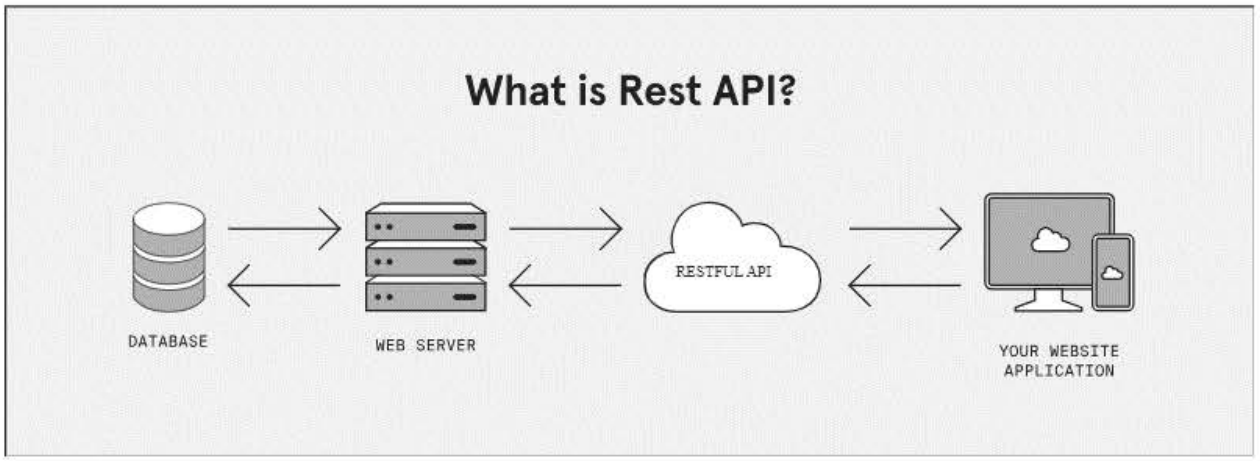

Рисунок 1.1 – Схема роботи Rest API

GraphQL (рис. 1.2) є мовою запитів і середовищем виконання для API, розробленим компанією Facebook. На відміну від RESTful API, де клієнт отримує визначені дані від сервера, у GraphQL клієнт може запросити лише ті дані, які йому потрібні, і отримати їх в одному запиті. GraphQL надає гнучку модель запитів, де клієнт визначає структуру та обсяг даних, які йому необхідні.

Переваги GraphQL включають:

– гнучкість запитів та можливість вибирати лише потрібні дані;

– зменшення кількості запитів до сервера, що покращує продуктивність;

– можливість розробки клієнтського додатку незалежно від серверної частини;

– документація та інструменти автодокументування запитів.

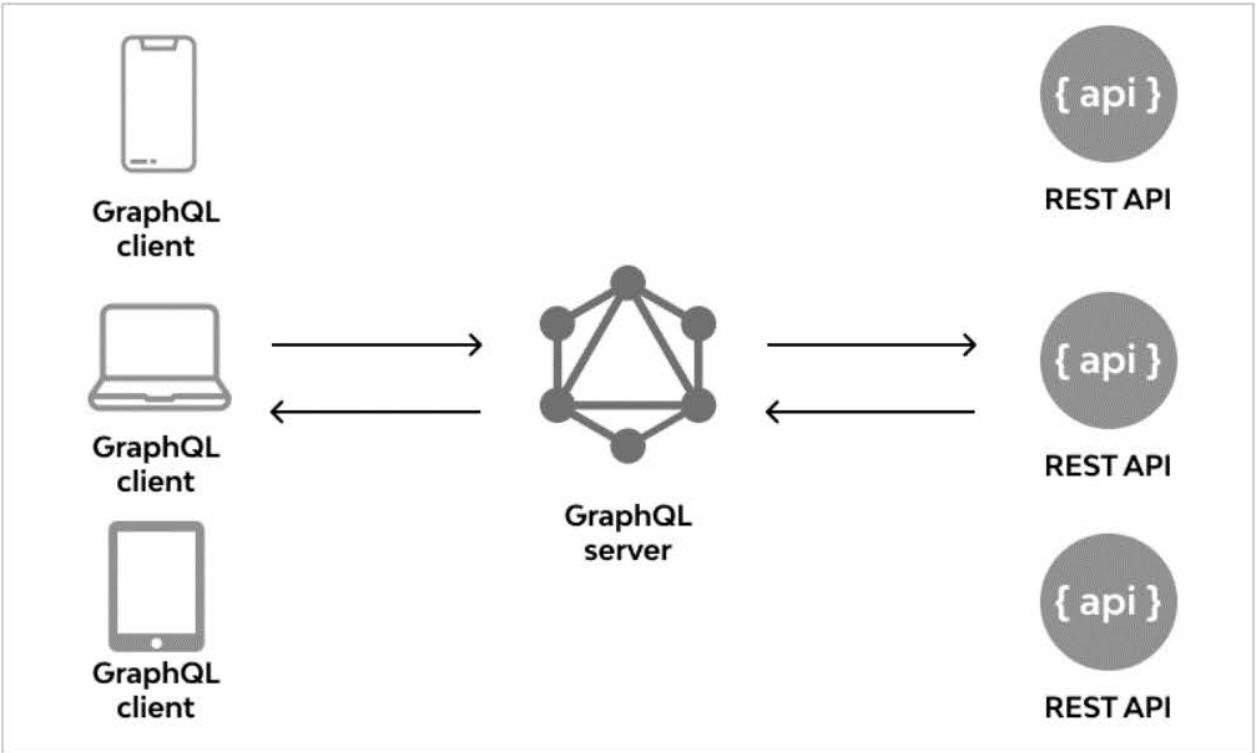

Рисунок 1.2 – Схема роботи GraphQL

SOAP (рис. 1.3) є протоколом обміну структурованими повідомленнями між компонентами програмного забезпечення. Він визначає формат повідомлень у форматі XML і використовує протоколи, такі як HTTP для передачі повідомлень. SOAP орієнтований обмін повідомленнями між додатками, використовуючи формальні описи інтерфейсів.

Переваги SOAP включають:

– розширюваність та підтримка формалізованих описів інтерфейсів;

– стійкість до різних протоколів передачі даних (HTTP, SMTP та ін.);

– безпека та надійність передачі даних.

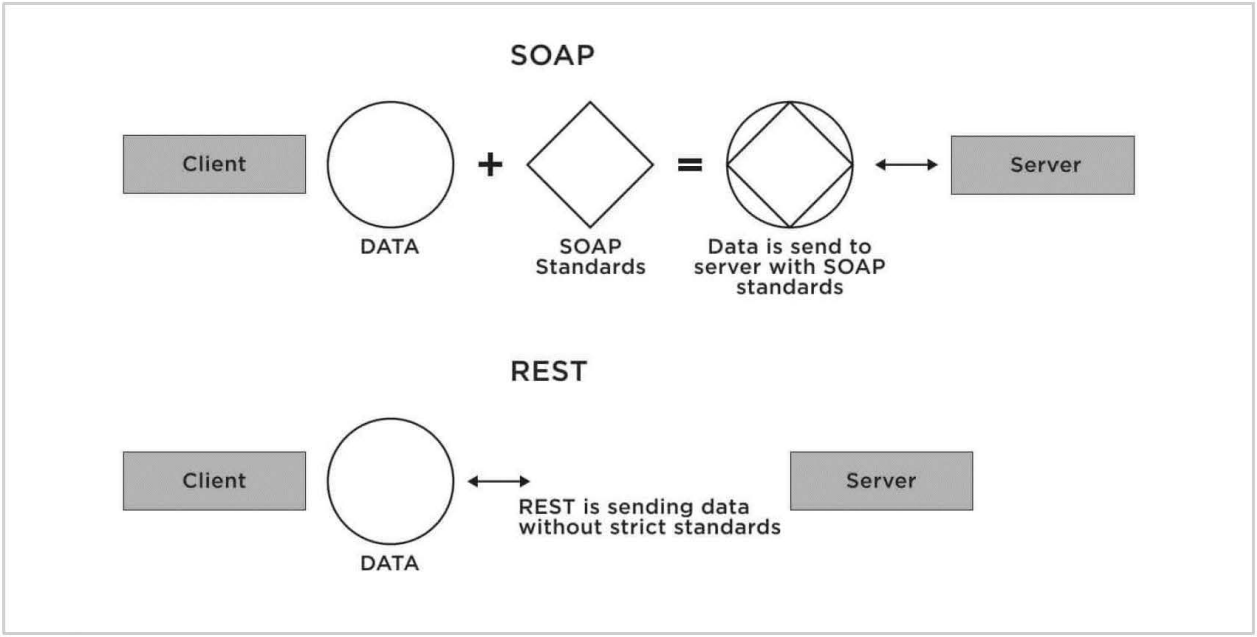

Рисунок 1.3 – Схема роботи SOAP

Крім вищеперелічених типів API, також існують інші, такі як gRPC, OData, JSON-RPC тощо. Кожен з них має свої особливості, переваги та обмеження, і вибір типу API залежить від вимог конкретного проекту.

Результати огляду різних типів API включають:

– порівняльний аналіз основних характеристик кожного типу API, таких як структура запитів, протоколи передачі, підтримка формалізованих описів та документації тощо;

– оцінка переваг та недоліків кожного типу API з урахуванням вимог проекту та контексту використання;

– рекомендації щодо вибору відповідного типу API для конкретного проекту або сценарію використання.

Це докладне дослідження різних типів API дозволяє розробникам та командам розробки приймати обґрунтовані рішення при виборі відповідного типу API для своїх проектів з огляду на їх особливості та вимоги.

1.3 Аналіз існуючих методологій розробки API

Аналіз існуючих методологій розробки API є важливою частиною дослідження розробки програмного забезпечення

Розбір методології розробки RESTful API:

– вивчення принципів архітектури REST та їх застосування у розробці API;

– аналіз найкращих практик та угод при розробці RESTful API, таких як використання правильних HTTP-методів, унікальних URI, статусних кодів тощо;

– оцінка інструментів та фреймворків, призначених для розробки RESTful API, таких як Express.js, Spring Boot, Django Rest Framework та інших;

– дослідження методів аутентифікації та авторизації у RESTful API, включаючи токени доступу, JWT (JSON Web Tokens) та OAuth.

Аналіз методології розробки GraphQL API:

– вивчення основних концепцій та принципів GraphQL, таких як схеми, типи даних, запити та мутації;

– розгляд найкращих практик при проектуванні GraphQL API, включаючи правильне моделювання даних та керування версіями схеми;

– оцінка інструментів та бібліотек для розробки GraphQL API, таких як Apollo Server, GraphQL Yoga, Graphene та інших;

– дослідження методів авторизації та аутентифікації в GraphQL API, включаючи використання middleware та ролей доступу.

Огляд інших методологій розробки API:

– розгляд методології розробки SOAP API, включаючи використання WSDL (Web Services Description Language) та XML-повідомлень;

– аналіз методології розробки gRPC API, заснованої на протоколі Protocol Buffers та двонаправленому потоці даних;

– вивчення методології розробки OData API, що забезпечує стандартизований доступ до даних та запитів на основі протоколу HTTP.

Порівняльний аналіз методологій:

– порівняння основних характеристик кожної методології, таких як структура запитів, протоколи передачі, гнучкість запитів тощо;

– оцінка переваг та недоліків кожної методології з урахуванням вимог проекту та контексту використання;

– рекомендації щодо вибору найбільш підходящої методології розробки API залежно від конкретного сценарію використання.

Результати аналізу існуючих методологій розробки API надають розробникам і командам розробки інформацію про різні підходи до створення API, їх особливості та потенційні плюси та мінуси. Це дозволяє вибирати найбільш підходящу методологію відповідно до вимог проекту та забезпечує успішну розробку високоякісних API.

1.4 Огляд інструментів та фреймворків для проектування, відладки та тестування API

Огляд інструментів та фреймворків для проектування, налагодження та тестування API надає інформацію про різні інструменти та технології, які допомагають розробникам у створенні, налагодженні та тестуванні API.

Інструменти для проектування API:

– Swagger (рис. 1.4). Swagger (тепер відомий як OpenAPI) надає набір інструментів для створення та документування RESTful API. Він дозволяє описувати структуру та функціональність API з використанням мови специфікації OpenAPI, а потім генерувати клієнтські бібліотеки та інтерактивну документацію;

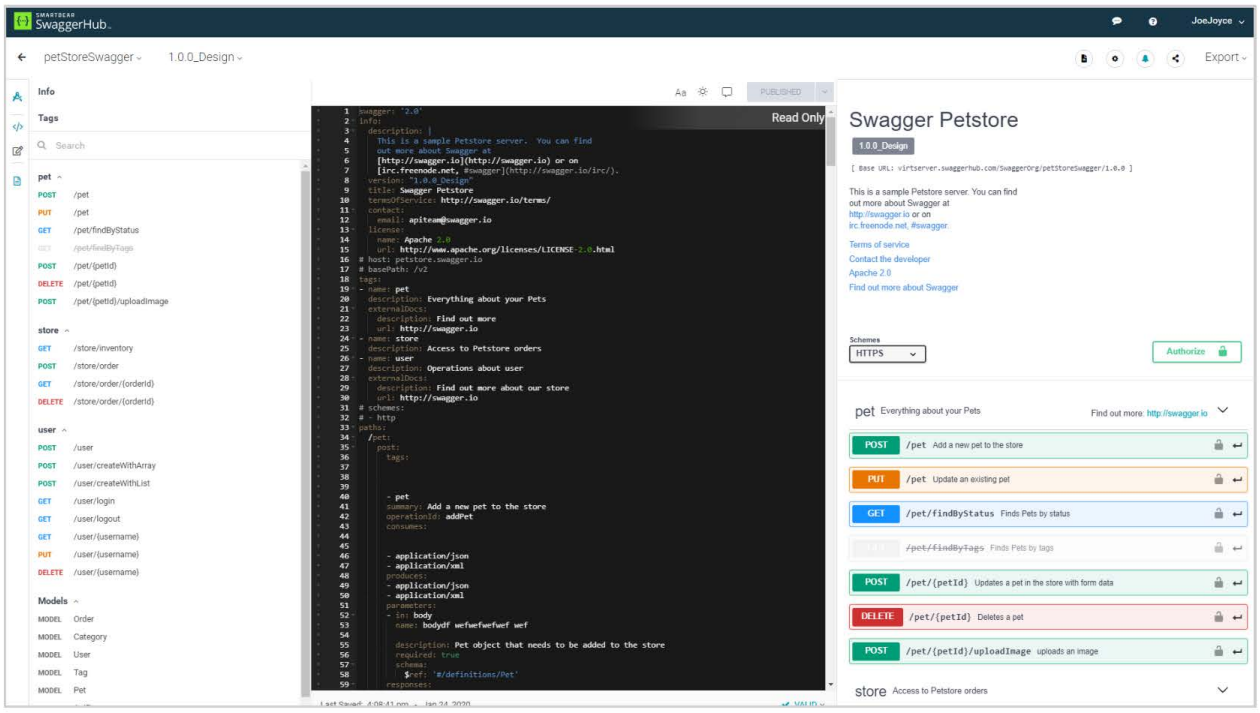

Рисунок 1.4 – Інтерфейс Swagger

– RAML (рис. 1.5). RAML (RESTful API Modeling Language) також є специфікацією для опису RESTful API. Він пропонує простий та читаний синтаксис для визначення ресурсів, методів, параметрів та інших аспектів API;

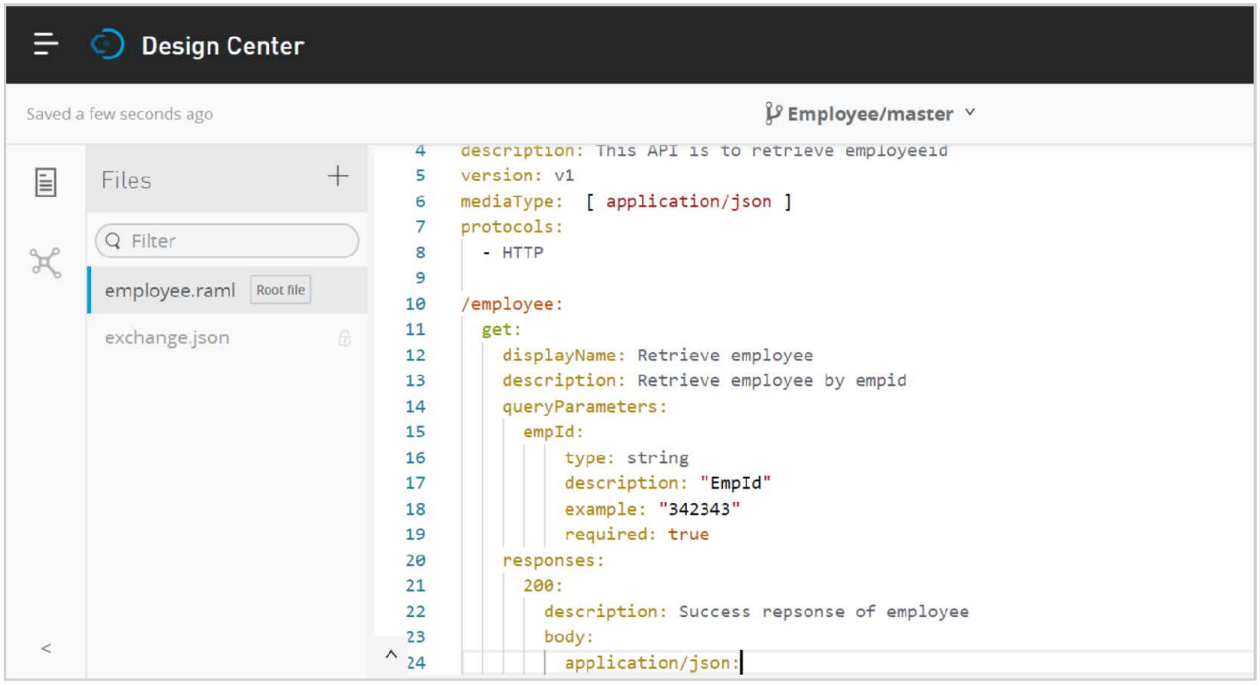

Рисунок 1.5 – Використання RAML

– API Blueprint. API Blueprint – це ще одна специфікація для опису RESTful API. Він використовує синтаксис, заснований на Markdown, що робить його легким для читання та написання.

Інструменти для відладки API:

– Postman (рис. 1.6). Postman – це популярний інструмент для тестування та налагодження API. Він надає інтуїтивно зрозумілий інтерфейс для створення запитів до API, перевірки відповідей, автоматизації тестів та багато іншого;

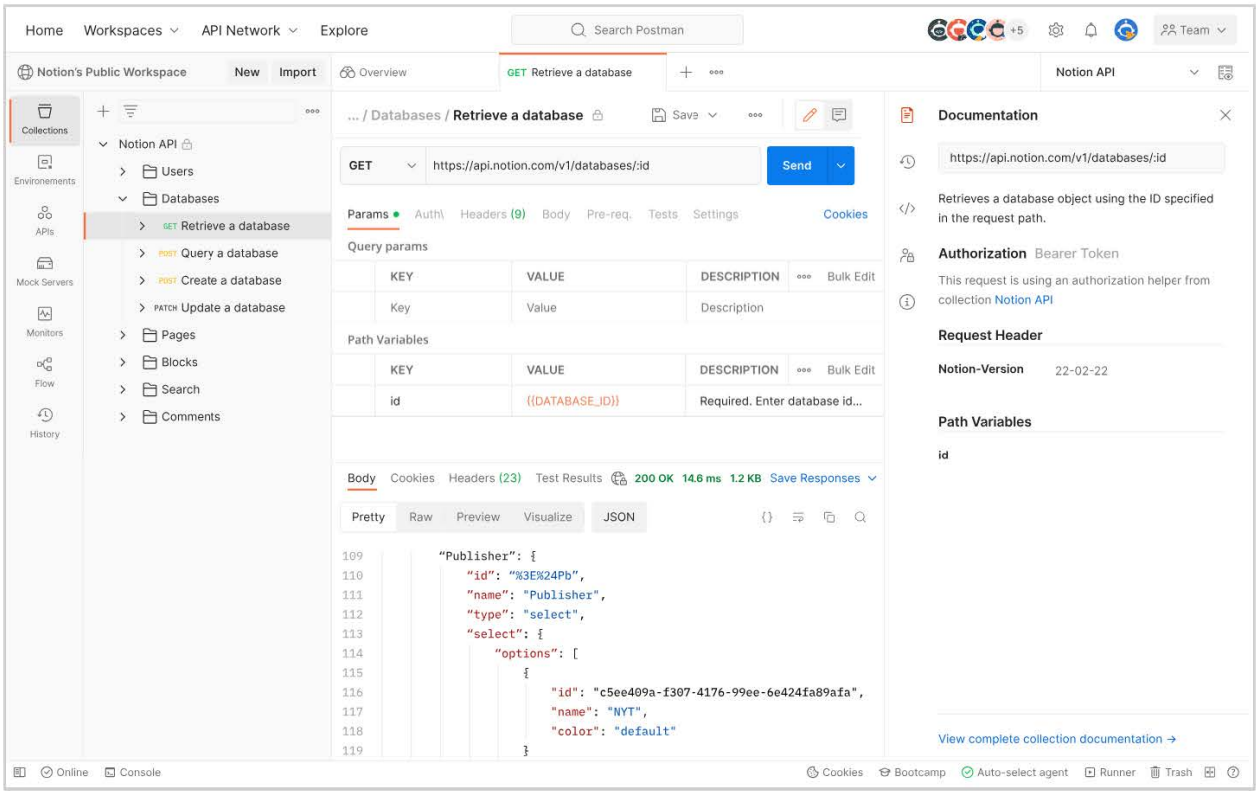

Рисунок 1.6 – Інтерфейс Postman

– Insomnia (рис. 1.7). Insomnia – це ще один інструмент для тестування та налагодження API, який пропонує схожий набір функцій, як і Postman. Він підтримує різні методи авторизації та дозволяє організовувати запити в колекції для повторного використання;

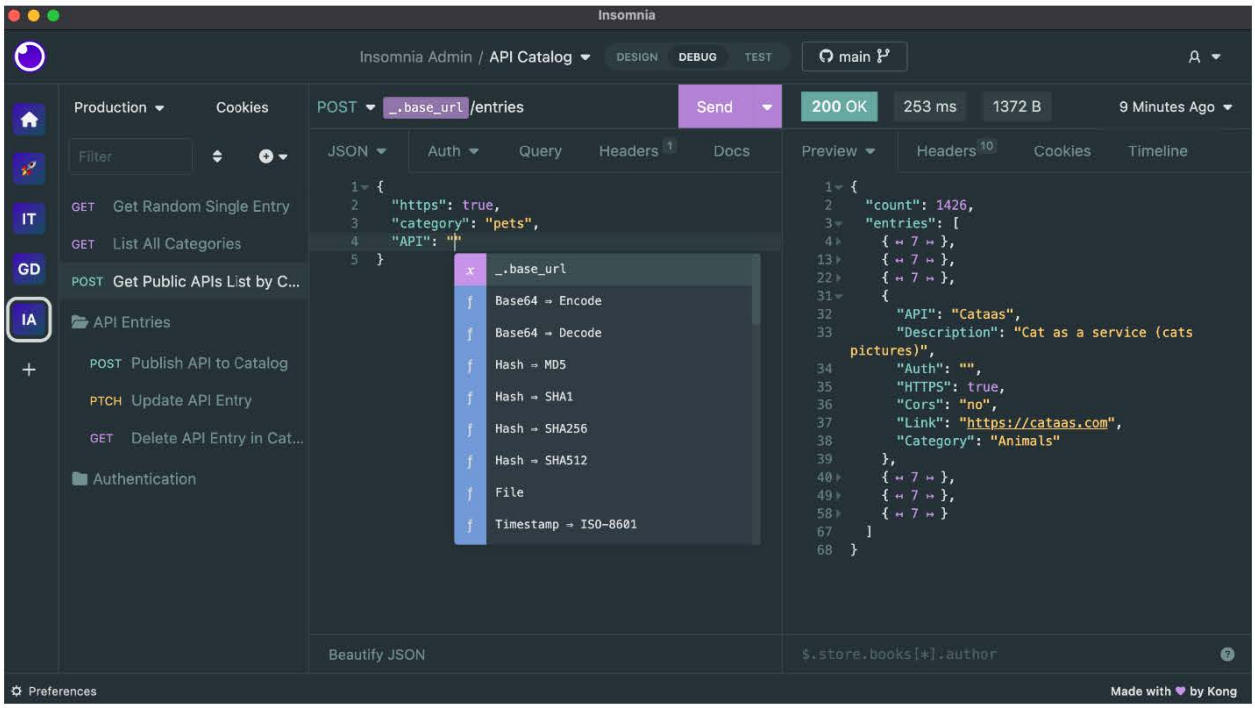

Рисунок 1.7 – Інтерфейс Insomnia

– Curl (рис. 1.8). curl це командний рядок, що дозволяє надсилати HTTP-запити до API та отримувати відповіді. Він широко використовується в розробці API та пропонує безліч опцій для налаштування запитів.

Фреймворки для розробки API:

– Express.js. Express.js – це популярний фреймворк для розробки веб-додатків на Node.js. Він полегшує створення RESTful API з використанням простого та гнучкого маршрутизатора та обробників запитів;

– Django Rest Framework. Django Rest Framework – це фреймворк для розробки RESTful API мовою Python з використанням фреймворку Django. Він надає потужні інструменти для серіалізації даних, авторизації, валідації запитів та інших функцій;

– Spring Boot. Spring Boot це фреймворк для розробки веб-застосунків на мові Java. Він надає зручні засоби для створення RESTful API з використанням анотацій та конфігурації через код.

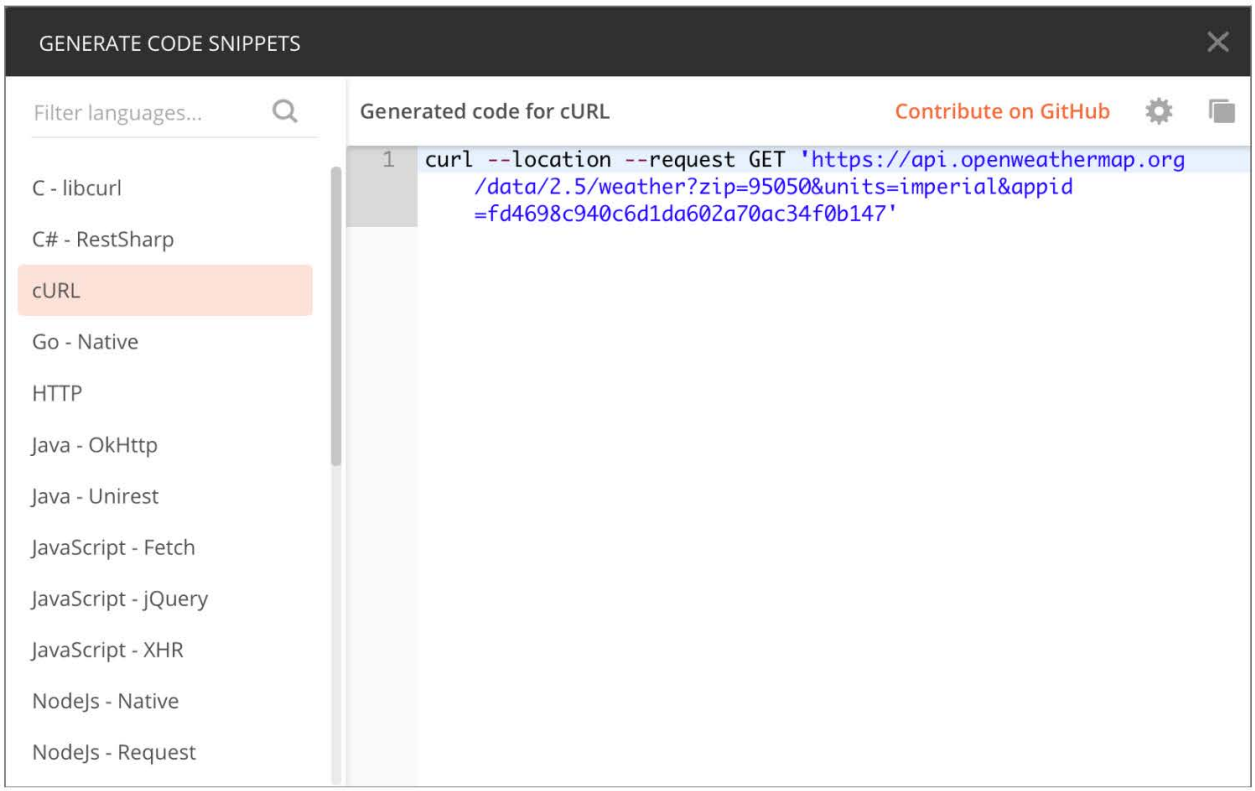

Рисунок 1.8 – Використання Curl

Інструменти для тестування API:

– Postman (вже згадувався вище). Postman дозволяє створювати та запускати автоматизовані тести для API, перевіряти коди стану, перевіряти значення полів та багато іншого;

– Newman. Newman – це командний рядок для запуску колекцій тестів, створених у Postman. Він дозволяє інтегрувати тестування API у процес розробки чи безперервної інтеграції;

– Jest. Jest це фреймворк для тестування JavaScript, який також може бути використаний для тестування API. Він надає набір функцій для створення та запуску тестів, перевірки відповідей API та затвердження результатів.

Це лише деякі з безлічі інструментів та фреймворків, доступних для проектування, налагодження та тестування API. Вибір конкретних інструментів залежить від вимог проекту, переваг розробника та технологій, що використовуються.

1.5 Висновки до першого розділу. Постановка задач дослідження

В результаті проведеного дослідження предметної галузі можна зробити такі загальні висновки:

1) API (Application Programming Interface) відіграє важливу роль у сучасному програмному забезпеченні, дозволяючи різним додаткам та сервісам взаємодіяти та обмінюватися даними. API надає набір функцій, протоколів та стандартів для розробки, інтеграції та використання програмного забезпечення;

2) в огляді різних типів API було розглянуто основні типи, такі як RESTful, GraphQL, SOAP, gRPC та OData. Кожен тип має свої особливості, переваги та обмеження, і вибір типу API залежить від вимог та контексту проекту;

3) аналіз існуючих методологій розробки API дозволяє розробникам обрати найбільш підходящий під час створення API. Методології, такі як RESTful, GraphQL та інші, надають набір принципів, угод та інструментів для розробки, документування та підтримки API;

4) огляд інструментів та фреймворків для проектування, налагодження та тестування API надає широкий вибір інструментів, які допомагають розробникам спростити та прискорити процес створення та тестування API. Інструменти, такі як Swagger, Postman, Express.js та інші, пропонують потужні можливості для розробки, налагодження та тестування API.

Загалом проведене дослідження дозволяє розробникам та командам розробки приймати обґрунтовані рішення при розробці та використанні API. Воно надає огляд основних понять, типів API, методологій розробки та інструментів, які допомагають досягти успішних результатів при розробці програмного забезпечення на основі API.

Метою роботи є розроблення програмного забезпечення для проектування, відладки та тестування API.

Для досягнення поставленої мети роботи можуть бути сформульовані наступний перелік задач:

1) аналіз існуючих методологій та інструментів. Провести огляд та аналіз існуючих методологій, інструментів та підходів, що використовуються у розробці, налагодженні та тестуванні API. Вивчити їх переваги та недоліки, а також їх застосування для різних типів API;

2) дослідження вимог щодо розробки API. Вивчити вимоги та очікування розробників API, а також потреби та обмеження, з якими вони стикаються в процесі роботи. Визначити основні функціональні та нефункціональні вимоги до програмного забезпечення, яке буде розроблено;

3) проектування інтерфейсу користувача. Розробити інтуїтивно зрозумілий і зручний інтерфейс користувача, що дозволяє розробникам проектувати API і визначати його структуру, функціональність і параметри. Врахувати важливість чіткості та легкості розуміння інтерфейсу для підвищення продуктивності розробників;

4) створення інструментів налагодження та тестування. Розробити інструменти, які допоможуть розробникам виявляти та виправляти помилки в API, а також перевіряти його працездатність та продуктивність. Включити функції автоматичного тестування та перевірки відповідності API стандартам та специфікаціям;

5) інтеграція засобів розробки. Забезпечити інтеграцію розробленого програмного забезпечення з існуючими засобами розробки та популярними інструментами, такими як інтегровані середовища розробки (IDE), системи контролю версій та інструменти безперервної інтеграції та доставки (CI/CD). Це дозволить розробникам легко інтегрувати створювані API у існуючі проекти та системи;

6) розробка документації та версіонування API. Розробити механізми для документування та версіонування API, щоб забезпечити ясність, стабільність та зворотну сумісність інтерфейсу для розробників-споживачів.

Включити можливість автоматичного створення документації на основі визначень API;

7) тестування та оцінка результатів. Провести тестування розробленого програмного забезпечення, оцінити його ефективність та надійність, а також провести порівняльний аналіз з існуючими інструментами та методами розробки API. Виправити виявлені проблеми та покращити функціональність за результатами тестування.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРОЕКТУВАННЯ, ВІДЛАДКИ ТА ТЕСТУВАННЯ API

2.1 Вибір програмних засобів для розроблення програмного забезпечення для проектування, відладки та тестування API

Цей проект може бути створений завдяки використанню наступних технологій та інструментів:

1) мова програмування. Java об'єктно-орієнтована мова програмування, що використовується для розробки програм;

2) бібліотека Swing. Swing є частиною Java Foundation Classes (JFC) та надає набір класів та компонентів для створення графічного інтерфейсу користувача (GUI) у Java;

3) AWT (Abstract Window Toolkit). AWT також є частиною Java Foundation Classes та надає основні класи для створення графічного інтерфейсу користувача. У цьому проекті AWT використовується для розміщення компонентів GUI та керування компонуванням елементів інтерфейсу;

4) Java IO. Java IO (Input/Output) надає класи та методи для роботи з потоками вводу-виводу, такими як читання та запис файлів;

5) HttpURLConnection. Це клас Java, який надає функціональність для надсилання HTTP-запитів та отримання відповідей. Він використовується для взаємодії з API та виконання HTTP-запитів;

6) JFileChooser. JFileChooser – це компонент Swing, який надає діалогове вікно для вибору файлу чи директорії на комп'ютері;

7) Java Collections Framework. Java Collections Framework надає класи та інтерфейси для роботи з колекціями об'єктів, такими як списки, множини та відображення. У цьому проекті HashMap використовується для збереження параметрів запиту.

Загалом проект використовував Java та відповідні бібліотеки та інструменти для створення GUI додатка та виконання HTTP-запитів до API.

2.2 Мова програмування Java

Java є об'єктно-орієнтованою мовою програмування, розробленою компанією Sun Microsystems (пізніше придбаною компанією Oracle). Вона була випущена у 1995 році і з того часу стала однією з найпопулярніших мов програмування у світі.

Основні особливості та можливості мови Java:

1) платформонезалежність. Однією з головних переваг Java є його здатність працювати на різних платформах без необхідності перекомпіляції вихідного коду. Це досягається завдяки використанню віртуальної машини Java (JVM), яка виконує байт-код Java. Таким чином, програми, написані на Java, можуть виконуватись на будь-якій платформі, яка підтримує JVM;

2) об'єктно-орієнтованість. Java повністю об'єктно-орієнтована мова, що означає, що всі елементи програми (класи, об'єкти, методи тощо) є об'єктами. Він підтримує основні принципи ООП, такі як інкапсуляція, успадкування, поліморфізм та абстракція;

3) простота та зрозумілість. Java розроблений з урахуванням простоти та зрозумілості синтаксису. Він дозволяє розробникам легко розуміти і створювати програми, що робить його придатним для розробників-початківців;

4) багата стандартна бібліотека. Java поставляється з багатьма класами та бібліотеками, які забезпечують готові рішення для різних завдань. Це включає роботу з мережею, базами даних, інтерфейсом користувача, багатопоточністю, введенням-виводом і багатьма іншими аспектами розробки програмного забезпечення;

5) багатопоточність. Java пропонує вбудовану підтримку багатопоточності, що дозволяє розробляти багатопоточні програми.

Багатопоточність Java забезпечується за допомогою класів і інтерфейсів, таких як Thread і Runnable, а також механізмів синхронізації та управління потоками;

6) безпека. Java було розроблено з упором на безпеку. Віртуальна машина Java (JVM) контролює доступ до пам'яті та виконання програми, запобігаючи небажаним операціям, таким як переповнення буфера та несанкціонований доступ до пам'яті;

7) розширюваність. Java підтримує розширення шляхом створення власних класів і бібліотек. Розробники можуть створювати власні класи, успадковувати класи зі стандартної бібліотеки та створювати модулі та плагіни для розширення функціональності мови.

Java широко використовується для розробки різних типів програм, включаючи веб-додатки, мобільні програми, настільні програми, ігри, серверні програми та багато іншого. Він також є основною мовою для розробки Android-додатків.

В цілому, Java пропонує потужні засоби розробки та хорошу підтримку спільноти розробників, що робить його популярним вибором для створення надійного та масштабованого програмного забезпечення.

2.3 Бібліотека Swing

Swing – це бібліотека інтерфейсу користувача (UI) для мови програмування Java. Вона надає набір компонентів та інструментів для створення графічних інтерфейсів, таких як вікна, кнопки, текстові поля, таблиці, меню та багато іншого. Swing є частиною стандартної бібліотеки Java і вона використовується для розробки настільних додатків з графічним інтерфейсом (рис. 2.1).

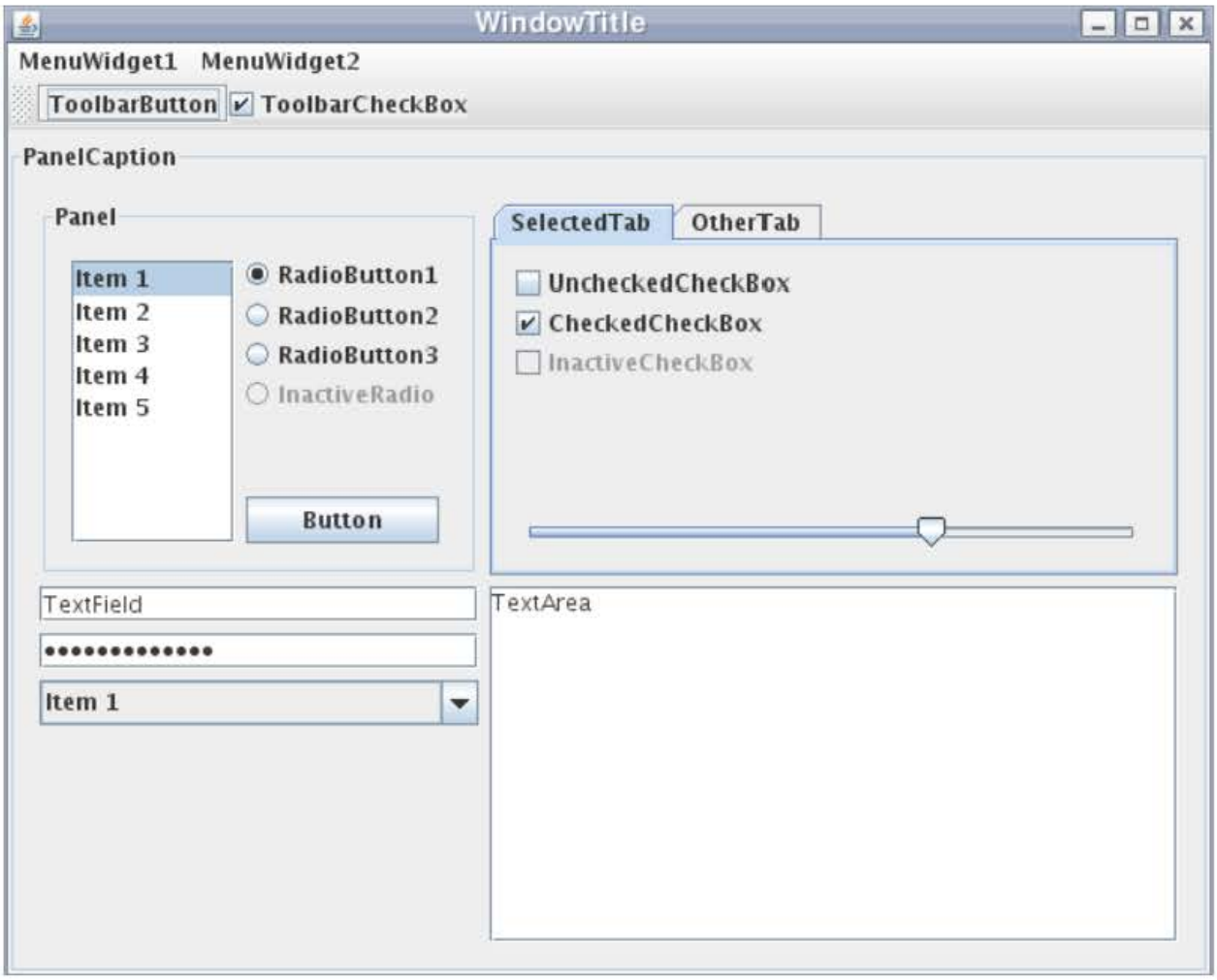

Рисунок 2.1 – Інтерфейс, розроблений за допомогою бібліотеки Swing

Деякі особливості та можливості бібліотеки Swing:

1) компоненти та контейнери. Swing надає широкий набір компонентів, таких як JButton, JLabel, JTextField, JTable та інші. Компоненти можуть бути розміщені в контейнерах, таких як JFrame, JPanel, JDialog та інших, щоб створити ієрархічну структуру інтерфейсу користувача;

2) налаштування користувача. Swing дозволяє розробникам налаштовувати зовнішній вигляд і поведінку компонентів за допомогою різних властивостей і методів. Це включає зміну кольору, шрифту, розміру, розташування та інших компонентів атрибутів;

3) макети. Для керування розташуванням компонентів у контейнерах Swing використовує макети (layout managers). Макети визначають, як компоненти будуть розподілені та вирівняні усередині контейнера. Деякі з

найпоширеніших макетів у Swing включають BorderLayout, GridLayout, FlowLayout та GroupLayout;

4) подіюча модель. Swing використовує модель подій для обробки дій користувача. Розробники можуть прив'язувати обробники подій до компонентів, щоб реагувати на дії користувачів, такі як натискання кнопки або вибір елемента зі списку;

5) рендеринг та графіка. Swing надає можливості для рендерингу компонентів та відображення графічних елементів, таких як іконки, зображення та фігури. Він підтримує різні формати зображень, включаючи JPEG, PNG та GIF;

6) міжплатформність. Завдяки використанню віртуальної машини Java (JVM), програми, створені за допомогою Swing, можуть бути запущені на різних платформах, включаючи Windows, macOS та Linux, без необхідності переписувати код;

7) підтримка багатомовності. Swing надає підтримку локалізації та багатомовності, що дозволяє розробникам створювати програми різними мовами і використовувати різні набори символів;

8) підтримка платформи Java 2D. Swing інтегрований з платформою Java 2D, що дозволяє створювати графічні компоненти та малювати на них із використанням різних інструментів та ефектів. Java 2D надає можливості для створення елементів управління, анімацій і малювання графічних об'єктів;

9) підтримка перетягування та перетягування. Swing надає механізми для реалізації перетягування та перетягування компонентів. Це дозволяє користувачам перетягувати елементи інтерфейсу з одного місця в інше або між різними компонентами;

10) вбудовані діалогові вікна. Swing пропонує різні стандартні діалогові вікна, такі як вікна повідомлень, вікна введення, вікна вибору файлів та директорій. Ці діалогові вікна спрощують взаємодію користувача та дозволяють отримати інформацію або запросити введення даних;

11) підтримка доступності. Swing забезпечує підтримку для людей з обмеженими можливостями. Це включає можливість роботи з клавіатурою для навігації по інтерфейсу, використання екранних читачів для читання вмісту та підтримку конфігурації шрифтів та кольорів для покращення видимості;

12) підтримка панелей інструментів та меню. Swing пропонує гнучку систему для створення панелей інструментів та меню. Панелі інструментів дозволяють розміщувати кнопки та інші елементи керування у верхній частині програми для швидкого доступу до функціональності, а меню надають ієрархічну структуру команд та опцій;

13) можливість розширення. Swing надає можливість створювати власні компоненти і розширювати функціональність за допомогою успадкування та створення класів користувача. Розробники можуть створювати власні компоненти з унікальною поведінкою та зовнішнім виглядом, щоб відповідати специфічним вимогам програми.

Swing був розроблений як заміна старої бібліотеки AWT (Abstract Window Toolkit) і пропонує більш гнучкий і потужний набір інструментів для розробки графічних інтерфейсів в Java. Він активно використовується розробниками для створення різноманітних настільних додатків, включаючи редактори тексту, графічні редактори, інструменти керування даними та багато іншого.

Хоча в останні роки з'явилися й інші альтернативи, такі як JavaFX, Swing залишається популярним і широко використовуваним інструментом для створення графічних інтерфейсів в Java.

2.4 Бібліотека JavaFX

JavaFX – це сучасна бібліотека для створення графічних інтерфейсів (GUI) в мові програмування Java. Вона надає потужні інструменти та компоненти для розробки інтерактивних та стильних програм, включаючи

настільні програми, мобільні програми, програми для інтернету речей (IoT) та інші.

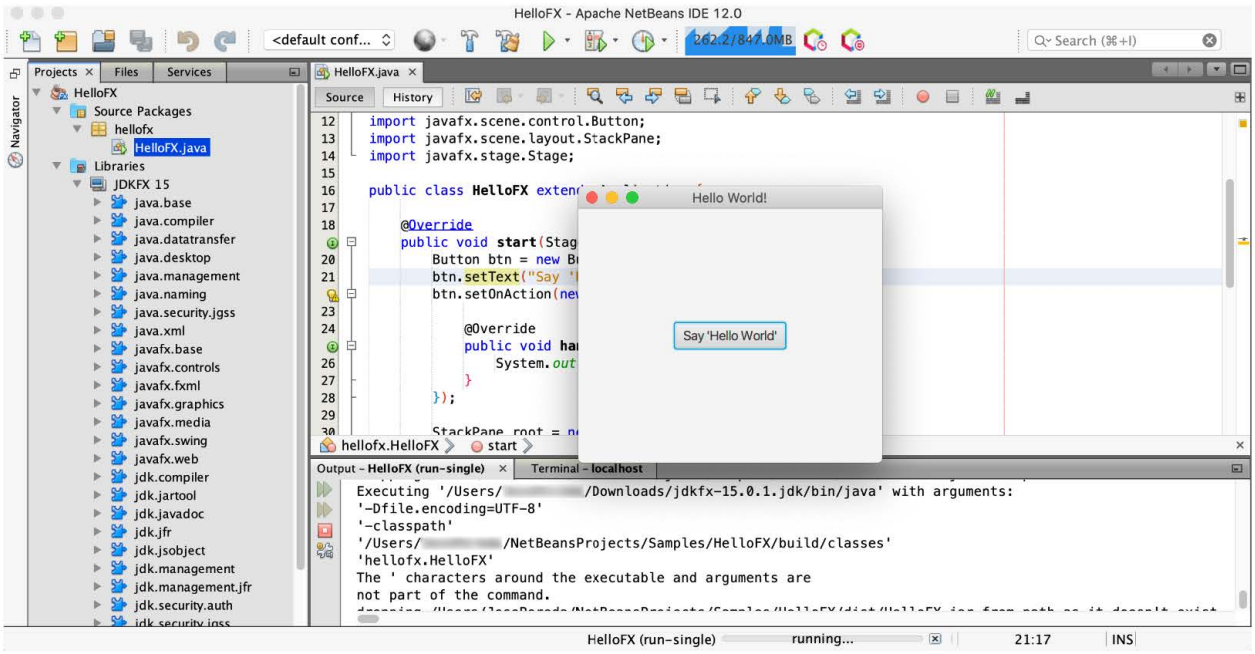

Рисунок 2.2 – Розробка застосунку за допомогою JavaFX

Нижче наведено основні особливості та можливості JavaFX:

1) декларативна мова розмітки. JavaFX використовує мову розмітки FXML, яка дозволяє розробникам описувати інтерфейс користувача у вигляді декларативного коду, відокремлюючи його від логіки програми. FXML заснований на мові розмітки XML і дозволяє створювати ієрархію компонентів, визначати їх властивості та пов’язувати події;

2) модульність та розширюваність. JavaFX побудований на основі модульної системи, яка дозволяє розробникам вибирати та використовувати лише необхідні компоненти та функціональності. Це робить JavaFX гнучким та легким для підтримки різних вимог додатків. Також розробники можуть створювати власні компоненти користувача і розширювати функціональність JavaFX;

3) графічна бібліотека. JavaFX надає потужні інструменти для малювання та маніпуляції графічними об’єктами. Вона підтримує різні типи

форм, ліній, текстових об'єктів, зображень та ефектів. Розробники можуть створювати кастомні форми, застосовувати анімацію та трансформації до графічних об'єктів;

4) мультимедіа та анімація. JavaFX має вбудовану підтримку для роботи з мультимедійним контентом, таким як аудіо, відео та зображення. Він пропонує можливості для відтворення, запису та маніпулювання мультимедійними даними. JavaFX також надає потужні інструменти для створення анімацій і переходів, що дозволяє створювати інтерактивні та привабливі інтерфейси користувача;

5) CSS-стилізація. JavaFX підтримує стилізацію інтерфейсу користувача за допомогою мови CSS. Це дозволяє розробникам легко змінювати зовнішній вигляд компонентів, вказуючи кольори, шрифти, фони та інші стильові властивості. Стилі можна застосовувати глобально для всієї програми або локально для окремих компонентів;

6) подієва модель. JavaFX має потужну подієву модель, яка дозволяє відстежувати та обробляти різні події, такі як кліки, наведення курсору, зміна значень та інші дії користувача. Це дозволяє створювати чуйні та інтерактивні програми, що реагують на взаємодію користувача;

7) інтеграція з Java. JavaFX добре інтегрований з мовою програмування Java, що дозволяє використовувати безліч можливостей Java для розробки програм. Розробники можуть використовувати стандартні класи та бібліотеки Java, працювати з колекціями, мережевими операціями, базами даних та іншими компонентами Java;

8) підтримка багатьох платформ. JavaFX підтримує різні платформи, включаючи настільні операційні системи (Windows, macOS, Linux), мобільні пристрої (Android, iOS) та вбудовані системи (embedded systems). Це дозволяє розробникам створювати програми для різних цільових платформ із загальним кодом та переносити їх між платформами;

9) сцена та сценічний граф. У JavaFX додаток складається зі сцени, яка є контейнером для компонентів інтерфейсу. Сцена містить у собі сценічний

граф, який є ієрархією компонентів, розташованих на сцені. Це дозволяє розробникам створювати складні макети інтерфейсу користувача, включаючи вкладені компоненти і компонування у вигляді дерева;

10) прив'язка даних. JavaFX пропонує механізм прив'язки даних, який дозволяє автоматично синхронізувати значення властивостей компонентів із моделлю даних. Це спрощує керування даними у додатку та оновлення інтерфейсу при зміні даних. Прив'язка даних також підтримує можливість створення двостороннього зв'язку, що дозволяє автоматично оновлювати дані з інтерфейсу, так і з моделі даних;

11) візуалізація даних. JavaFX надає потужні інструменти для візуалізації даних. Це включає графіки, діаграми, таблиці, дерева та інші компоненти, які можуть відображати дані в зручній та інформативній формі. Розробники можуть налаштовувати зовнішній вигляд та поведінку цих компонентів для відображення різних типів даних та уявлень;

12) анімація та переходи. JavaFX пропонує багаті можливості для створення анімацій і переходів, які надають інтерактивності та привабливості користувальницькому інтерфейсу. Розробники можуть створювати анімовані ефекти, змінювати властивості компонентів у часі, створювати переходи між різними станами та багато іншого. Це дозволяє створювати плавні і привабливі інтерфейси користувача;

13) міжплатформна підтримка. JavaFX має можливість працювати на різних платформах, включаючи настільні операційні системи (Windows, MacOS, Linux), мобільні пристрої (Android, iOS) та вбудовані системи (embedded systems). Це робить JavaFX універсальним інструментом для розробки програм, які можуть бути запущені на різних пристроях та платформах без значних змін;

14) інтеграція з браузером. JavaFX надає можливість інтеграції веб-вмісту додатка за допомогою компонента WebView. WebView дозволяє відображати веб-сторінки, включати JavaScript, взаємодіяти з вмістом

сторінки і навіть створювати власні елементи керування для роботи з веб-контентом.

JavaFX являє собою потужний інструмент для розробки графічних інтерфейсів в Java. Він має великі можливості для створення стильних та інтерактивних додатків, а також інтеграції з іншими технологіями. Завдяки своїй гнучкості, JavaFX дозволяє розробникам створювати різноманітні програми, що відповідають вимогам сучасних користувачів.

2.5 Бібліотека AWT

AWT (Abstract Window Toolkit) – це перша бібліотека для створення графічних інтерфейсів (GUI) в мові програмування Java. Вона була введена разом з мовою Java і надає базовий набір компонентів і класів для створення віконних додатків (рис. 2.3).

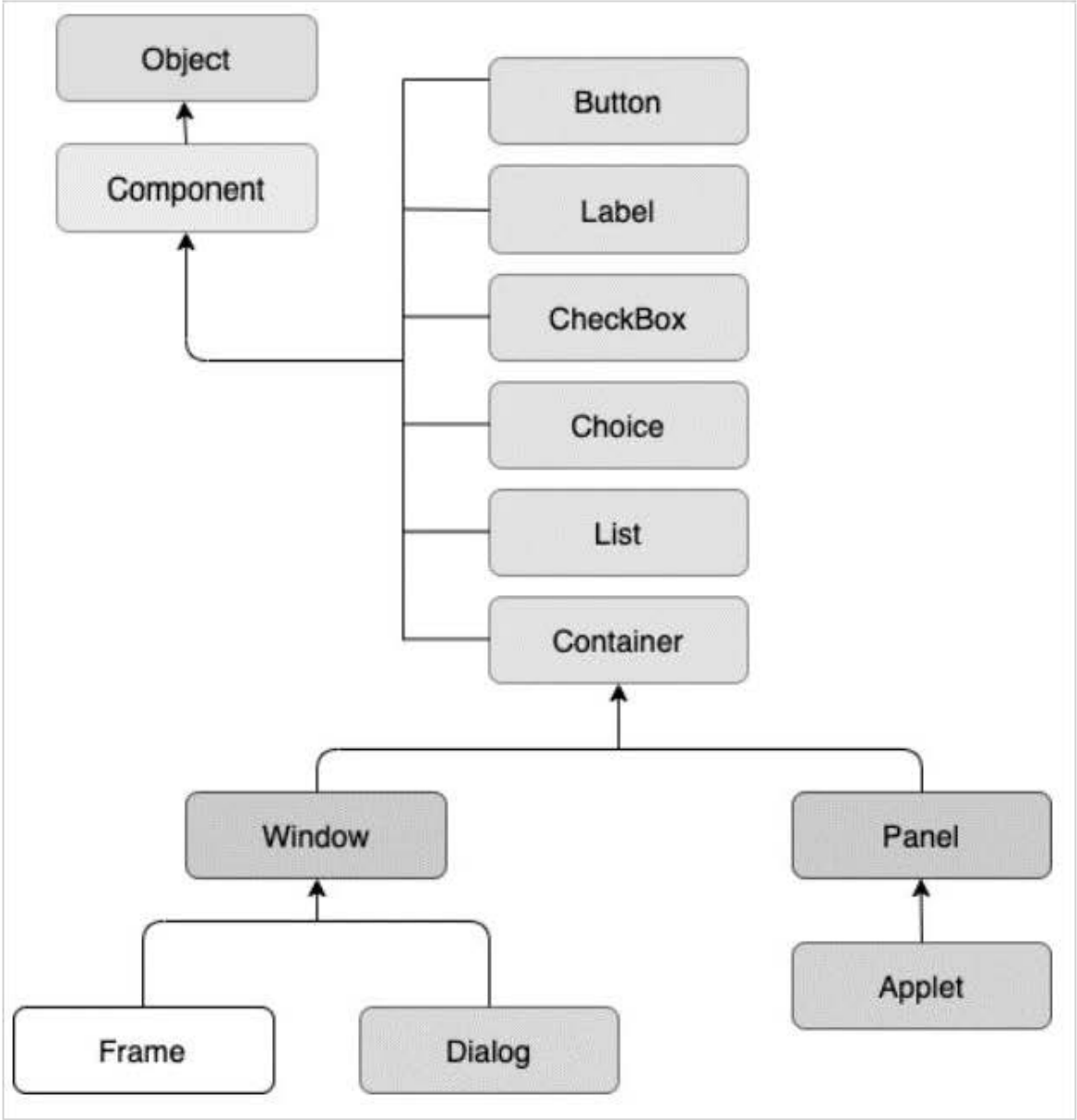

Рисунок 2.3 – Схема використання AWT

Ось основні аспекти та можливості AWT:

1) компоненти інтерфейсу користувача. AWT надає набір компонентів, таких як кнопки, текстові поля, мітки, панелі, списки та інші. Кожен компонент є об'єктом класу, що успадковується від класу Component. Компоненти можна комбінувати та розташовувати на екрані за допомогою менеджерів компонування;

2) менеджери компонування. AWT надає різні менеджери компонування для впорядкування та позиціонування компонентів на екрані. Деякі з них

включають BorderLayout, GridLayout, FlowLayout та CardLayout. Менеджери компонування дозволяють гнучко керувати розміщенням компонентів та їх розмірами у вікні програми;

3) графічний контекст. AWT надає класи та методи для малювання графічних об'єктів на екрані. Класи Graphics та Graphics2D надають функціональність для малювання ліній, фігур, тексту та зображень. Розробники можуть використовувати ці класи для створення інтерфейсу користувача і відображення графічного контенту;

4) обробка подій. AWT надає механізм обробки подій, таких як кліки миші, натискання клавіш, переміщення курсору та інші. Розробники можуть реагувати на події, визначати обробники подій та виконувати відповідні дії у відповідь на взаємодію користувача з інтерфейсом. Класи Event та EventHandler використовуються для роботи з подіями;

5) багатомовна підтримка. AWT підтримує багатомовні програми, що дозволяє розробникам створювати інтерфейси різними мовами. Вона підтримує кодування Юнікод, що забезпечує відображення символів різних мов, включаючи символи різних письмових систем;

6) підтримка різних платформ. AWT надає абстрактний рівень доступу до функціональності операційної системи, що дозволяє програмам працювати на різних платформах. Однак, AWT компоненти зазвичай використовують оформлення та стиль операційної системи, що робить програму більш нативною на кожній платформі;

7) робота з вікнами та діалоговими вікнами. AWT надає класи для створення та управління вікнами програми. Класи Frame та Window використовуються для створення основних вікон, а класи Dialog та JOptionPane надають можливості для створення діалогових вікон. Розробники можуть встановлювати заголовки вікон, задавати розміри, розташовувати їх на екрані та керувати їх видимістю;

8) підтримка графічних примітивів. AWT дозволяє створювати та маніпулювати різними графічними примітивами, такими як лінії,

прямокутники, еліпси та багато іншого. Розробники можуть використовувати класи, такі як Graphics та Graphics2D, для малювання та модифікації графічних елементів на полотні;

9) робота із зображеннями. AWT надає можливості для завантаження, відображення та маніпулювання зображеннями. Класи Image та ImageIcon дозволяють завантажувати зображення з файлів або ресурсів програми. Розробники можуть використовувати методи для зміни розмірів зображень, накладання фільтрів та перетворення зображень;

10) робота зі шрифтами та текстом. AWT надає функціональність для роботи з текстом та шрифтами. Розробники можуть встановлювати шрифти для відображення тексту на компонентах, змінювати їх розміри, стиль та колір. Класи Font та FontMetrics використовуються для роботи зі шрифтами та вимірюваннями тексту;

11) міжплатформна сумісність. Однією із сильних сторін AWT є її міжплатформна сумісність. Програми, розроблені з використанням AWT, можуть працювати різних операційних системах без необхідності внесення значних змін. Це робить AWT зручним вибором для створення додатків, що переносяться;

12) розширюваність. AWT надає можливість розширювати свою функціональність шляхом створення власних компонентів та класів. Розробники можуть створювати власні компоненти користувача, реалізовувати нові менеджери компонування і додавати специфічні для застосування функції.

AWT була першою бібліотекою для створення графічних інтерфейсів в Java, і вона поклала основу для розвитку більш сучасних і потужних бібліотек, таких як Swing і JavaFX. Проте, AWT досі надає базовий та надійний набір інструментів для створення простих графічних інтерфейсів та підтримує зворотну сумісність із старішими версіями Java.

2.6 Java IO (Input/Output)

Java IO (Input/Output) є частиною стандартної бібліотеки Java і надає потужний набір класів та методів для роботи з введенням та виведенням даних. Він надає можливості для читання та запису даних з різних джерел, таких як файли, мережеві з'єднання, пам'ять та інші (рис. 2.4).

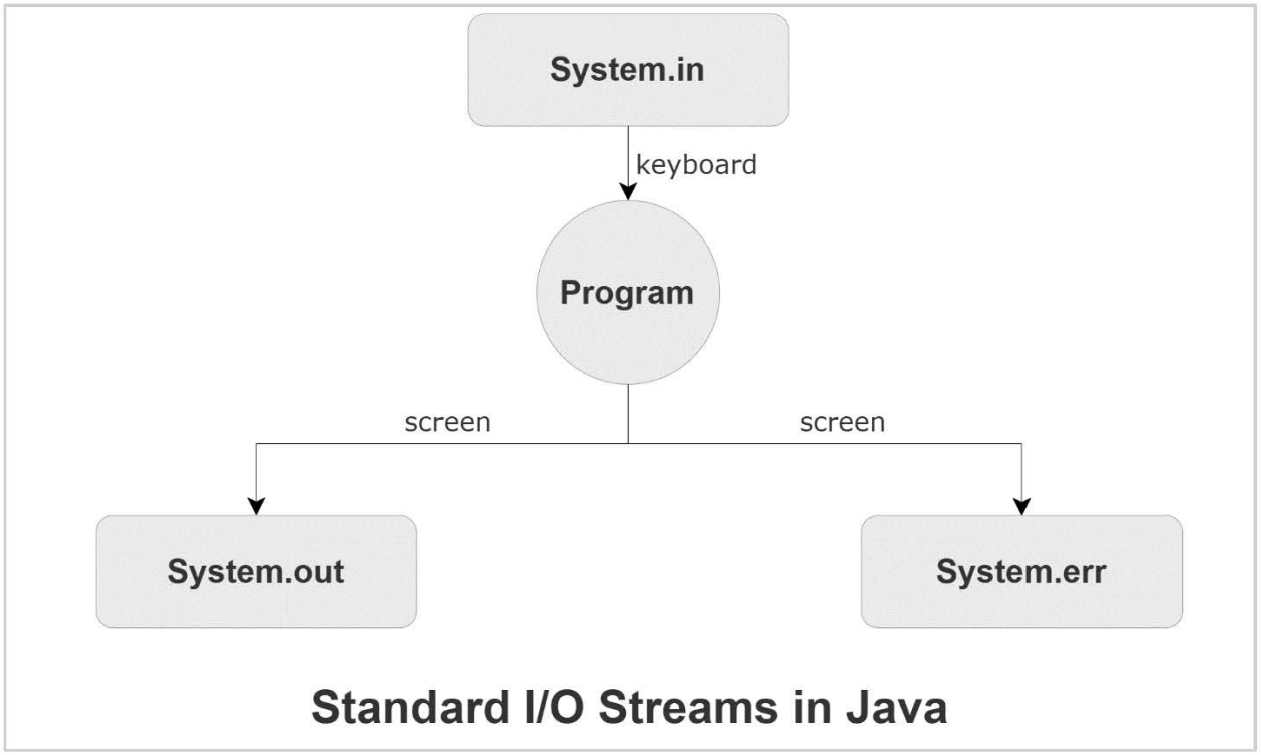

Рисунок 2.4 – Схема роботи Java IO (Input/Output)

Java IO має ієрархічну структуру класів, основними компонентами якої є потоки даних (streams) та класи для роботи з файлами. Ось деякі ключові аспекти та класи Java IO:

1) потоки даних. Потоки даних є послідовністю байтів або символів, які можуть бути прочитані або записані. У Java IO є два основних типи потоків байтові потоки (Byte Streams) та символьні потоки (Character Streams). Байтові потоки працюють із байтами даних, а символьні потоки працюють із символами (кодування Unicode);

2) байтові потоки. Класи, такі як FileInputStream та FileOutputStream, використовуються для читання та запису байтових даних. Ці потоки зручні під час роботи з двійковими файлами чи низькорівневими операціями вводу-вывода;

3) символьні потоки. Класи, такі як FileReader та FileWriter, використовуються для читання та запису символьних даних. Ці потоки надають зручні методи для роботи з текстовими файлами та підтримують різні кодування, дозволяючи правильно обробляти різні мови та символи;

4) робота з файлами. Java IO надає класи та методи для роботи з файловою системою. Класи, такі як File, Path і Files, дозволяють створювати, переміщувати, копіювати, видаляти та перейменовувати файли та директорії. Вони також надають методи для перевірки існування файлів, отримання інформації про файли (розмір, атрибути тощо) та багато іншого;

5) буферизація. Java IO надає можливості для буферизації читання та запису даних, що підвищує продуктивність. Класи, такі як BufferedInputStream, BufferedOutputStream, BufferedReader та BufferedWriter, обертають базові потоки даних і додають буферизацію операцій читання та запису;

6) серіалізація. Java IO надає механізм серіалізації, який дозволяє об'єктам бути перетвореними в послідовність байтів і збережені або передані. Класи ObjectOutputStream та ObjectInputStream використовуються для серіалізації та десеріалізації об'єктів;

7) робота з мережею. Java IO також надає класи для роботи з мережевими з'єднаннями. Класи, такі як Socket та ServerSocket, дозволяють створювати клієнт-серверні програми та обмінюватися даними по мережі. За допомогою цих класів можна встановлювати з'єднання, надсилати та приймати дані, а також керувати мережевими ресурсами;

8) робота з ZIP-архівами. Java IO надає класи для роботи із ZIP-архівами. Класи, такі як ZipInputStream та ZipOutputStream, дозволяють читати та

записувати дані в ZIP-архіви. Це зручно для архівування та розпакування файлів та директорій;

9) читання та запис даних. Java IO надає різні методи для читання та запису даних. Наприклад, методи read() та write() використовуються для читання та запису окремих байтів. Методи readLine() та write(String) використовуються для читання та запису текстових рядків. Крім того, Java IO підтримує форматований введення та виведення з використанням класів, таких як Scanner та PrintWriter;

10) обробка помилок. Java IO надає механізм обробки помилок та винятків, які можуть виникнути під час читання та запису даних. Розробники можуть використовувати блоки try-catch для обробки винятків IOException та інших винятків, пов'язаних із введенням-виводом. Це дозволяє надійніше керувати помилками та запобігати збоям у роботі програми;

11) фільтри потоків (Stream Filters). Java IO підтримує використання фільтрів потоків, які дозволяють перетворювати дані під час їх читання чи запису. Фільтри потоків реалізують інтерфейс java.io.FilterInputStream або java.io.FilterOutputStream і можуть додавати додаткову функціональність, таку як шифрування, стискування або кодування даних;

12) робота з символами та кодуваннями. Java IO символьні потоки (Character Streams) надають зручний спосіб роботи з текстовими даними. Вони дозволяють використовувати символи Unicode та підтримують різні кодування, такі як UTF-8, UTF-16 та інші. Класи InputStreamReader та OutputStreamWriter використовуються для перетворення байтових потоків у символьні та навпаки, забезпечуючи правильне читання та запис текстових даних;

13) неблокуючий ввод-виведення (NIO). У нових версіях Java з'явилася бібліотека NIO (Non-blocking IO), яка пропонує альтернативний підхід до роботи з введенням-виведенням даних. NIO надає класи, такі як Channel та Selector, які дозволяють ефективно обробляти безліч з'єднань одночасно без

блокування потоку виконання. Це особливо корисно для розробки серверних програм з великою кількістю клієнтів;

14) потоки об'єктів. Java IO надає можливість серіалізації та десеріалізації об'єктів за допомогою потоків ObjectInputStream та ObjectOutputStream. Серіалізація дозволяє зберегти об'єкти у файли або передавати їх по мережі, а десеріалізація дозволяє відновити об'єкти зі збережених даних. Це корисно при роботі зі складовими об'єктами, які потрібно зберегти та відновити у вихідному стані.

Java IO надає потужний та гнучкий набір інструментів для роботи з введенням та виведенням даних. Він має широкі можливості та підтримує різні типи джерел даних. Однак, у сучасних версіях Java рекомендується використовувати новішу бібліотеку NIO (Non-blocking IO) або NIO.2, яка пропонує ще більш ефективні та гнучкі можливості для роботи з введенням-виведенням даних.

2.7 Клас HttpURLConnection

HttpURLConnection – це клас у мові програмування Java, що надає можливості для створення та управління HTTP-з'єднаннями. Він є частиною пакету java.net та надає зручні методи для надсилання HTTP-запитів на сервер та отримання відповідей (рис. 2.5).

```
static private class Connection extends AsyncTask<String, String, String> {

    String serviceType;


    @Override
    protected String doInBackground(String... params) {
        // TODO Auto-generated method stub
        URL url;
        serviceType = params[params.length - 1];


        HttpURLConnection connection;
        try {
            url = new URL(params[0]);
            Log.d("hjh" , String.valueOf(url));

            connection = (HttpURLConnection) url.openConnection();
            connection.setDoOutput(true);
            connection.setDoInput(true);
            connection.setInstanceFollowRedirects(false);
            connection.setRequestMethod("GET");
            connection.setConnectTimeout(60000); // 60 Seconds
            connection.setReadTimeout(60000); // 60 Seconds
            connection.setRequestProperty("X-Requested-With", "XMLHttpRequest");
            connection.setRequestProperty("Content-Type",
                    "application/x-www-form-urlencoded;charset=UTF-8");
//              OutputStreamWriter writer = new OutputStreamWriter(
//                      connection.getOutputStream());
//              writer.write("");
//              writer.flush();
            int responseCode = connection.getResponseCode();
            Log.d("responsecode" , Integer.toString(responseCode));
            String line, retJson = "";
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(connection.getInputStream()));
```

Рисунок 2.5 – Приклад використання HttpURLConnection

Ось докладний опис функціональності HttpURLConnection:

– створення з'єднання. Для встановлення з'єднання із сервером використовується метод openConnection() класу URL. Він повертає об'єкт HttpURLConnection, що представляє відкрите з'єднання.

– налаштування запиту. Для налаштування параметрів запиту HTTP можна використовувати методи об'єкта HttpURLConnection. Наприклад, методи setRequestMethod() і setDoOutput() дозволяють встановити метод запиту (GET, POST, PUT, DELETE тощо) та вказати, чи будуть передаватися дані в тілі запиту.

– встановлення заголовків. Заголовки HTTP-запиту можна встановити за допомогою методу setRequestProperty(). Цей метод приймає ім'я заголовка та його значення;

– надсилання запиту. Для надсилання запиту на сервер використовується метод connect(). Він встановлює фактичне з'єднання та надсилає запит;

– отримання відповіді. Після надсилання запиту можна отримати відповідь від сервера. І тому доступні різні методи. Наприклад, метод getResponseCode() повертає HTTP код відповіді, метод getInputStream() повертає потік введення для читання вмісту відповіді;

– читання відповіді. Отриману відповідь можна прочитати з потоку введення. Наприклад, можна використовувати клас BufferedReader для читання текстового вмісту;

– закриття з'єднання. Після читання відповіді з'єднання має бути закрите викликом методу disconnect().HttpURLConnection надає безліч інших методів та можливостей для управління HTTP-запитами та відповідями. Наприклад, можна встановити тайм-аут з'єднання, відправляти параметри в запиті, працювати з cookie та багато іншого.

Цей клас є потужним інструментом для роботи з HTTP-протоколом в Java і широко використовується для створення клієнтських програм, веб-скрапінгу, взаємодії з API та інших сценаріїв, пов'язаних із HTTP-з'єднаннями.

2.8 Клас JFileChooser

JFileChooser – це клас Java, який надає можливість створення діалогового вікна вибору файлів. Він є частиною пакету javax.swing та забезпечує зручний спосіб для користувачів вибирати файли чи директорії на комп'ютері.

Ось докладний опис функціональності JFileChooser:

1) створення об'єкта JFileChooser. Для створення об'єкта JFileChooser необхідно використовувати конструктор за замовчуванням або конструктор із зазначенням початкової директорії. Початкова директорія визначає, з якої папки буде відкрито діалогове вікно вибору файлів;

2) відображення діалогового вікна вибору файлів. Для відображення діалогового вікна вибору файлів використовується метод showDialog(). Він приймає батьківський компонент (зазвичай це JFrame) та рядок із зазначенням тексту на кнопці;

3) обробка вибору файлу. Після відображення діалогового вікна та вибору файлу або директорії користувачем необхідно обробити результат вибору. Метод showDialog() повертає одну з таких констант. JFileChooser.APPROVE_OPTION, JFileChooser.CANCEL_OPTION або JFileChooser.ERROR_OPTION. Якщо користувач обрав файл, можна отримати його за допомогою методу getSelectedFile();

4) фільтрування файлів. JFileChooser дозволяє фільтрувати файли, що відображаються у діалоговому вікні, за допомогою методу addChoosableFileFilter(). Можна використовувати певні фільтри, такі як FileNameExtensionFilter або створити власний фільтр, реалізуючи інтерфейс FileFilter;

5) множинний вибір файлів. JFileChooser дозволяє вибирати кілька файлів відразу. Для цього слід викликати метод setMultiSelectionEnabled() та встановити значення true;

6) вибір лише директорій. Якщо потрібно вибирати тільки директорії, а не файли, можна викликати метод setFileSelectionMode() та встановити значення JFileChooser.DIRECTORIES_ONLY;

7) встановлення початкової директорії. Можна указати початкову директорію, яка відображатиметься при відкритті діалогового вікна, за допомогою методу setCurrentDirectory();

8) визначення доступних типів файлів. Для визначення типів файлів, які будуть відображатися та доступні для вибору у діалоговому вікні, можна використовувати метод setFileFilter(). Цей метод дозволяє вказати розширення файлів або імена файлів, які відображатимуться;

9) встановлення заголовка діалогового вікна. За допомогою методу setDialogTitle() можна встановити заголовок діалогового вікна;

10) налаштування кнопок діалогового вікна. JFileChooser дозволяє налаштувати тексти кнопок, які відображаються у діалоговому вікні. Методи setApproveButtonText(), setApproveButtonToolTipText() та setCancelButtonToolTipText() дозволяють встановити тексти та підказки для кнопок вибору файлу, скасування та ін;

11) використання додаткових функцій. JFileChooser надає інші функції, такі як можливість створення нових папок (setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES)), установка розширень файлів за промовчанням (setExtensions()), визначення допустимих розмірів файлів і директорій (setFileHiding ()) і багато іншого.

JFileChooser має велику гнучкість і дозволяє розробникам створювати дружні та зручні інтерфейси вибору файлів та директорій у Java-додатках. Він є корисним інструментом для роботи з файловою системою та взаємодії з користувачем.

2.9 Фреймворк Java Collections Framework

Java Collections Framework (Фреймворк колекцій Java) являє собою набір інтерфейсів, класів і алгоритмів, призначених для зручної та ефективної роботи з колекціями об'єктів Java. Він надає високорівневі абстракції та інструменти для зберігання, обробки та маніпулювання групами об'єктів.

Основні компоненти Java Collections Framework включають наступні:

1) інтерфейс Collection. Є базовим інтерфейсом для всіх колекцій Java. Він визначає загальні операції для роботи з колекціями, такі як додавання елемента, видалення, перевірка наявності елемента та інші;

2) інтерфейс List. Розширює інтерфейс Collection та визначає послідовність елементів з доступом за індексом. Включає методи для вставлення, видалення та оновлення елементів у списку. Реалізації інтерфейсу List включають ArrayList, LinkedList та інші;

3) інтерфейс Set. Розширює інтерфейс Collection та представляє унікальні елементи без дублювання. Не гарантує порядок елементів. Реалізації інтерфейсу Set включають HashSet, TreeSet та інші;

4) інтерфейс Map. Відображає ключ-значення, де кожен ключ зіставляється з певним значенням. Дозволяє швидко знаходити значення ключа. Реалізації інтерфейсу Map включають HashMap, TreeMap, LinkedHashMap та інші;

5) інтерфейс Queue. Представляє колекцію елементів, де елементи додаються до кінця та видаляються з початку. Включає методи додавання, видалення та обробки елементів. Реалізації інтерфейсу Queue включають LinkedList, PriorityQueue та інші;

6) класи-утиліти. Фреймворк також надає набір класів-утиліт для зручної роботи з колекціями, таких як Collections, Arrays та інші. Вони містять методи сортування, пошуку, копіювання та інших операцій над колекціями.

Переваги використання Java Collections Framework:

– зручність. Фреймворк надає зручні та прості у використанні інтерфейси та класи для роботи з колекціями, що спрощує розробку додатків;

– розширюваність. Фреймворк дозволяє створювати користувацькі реалізації колекцій та алгоритмів, щоб відповідати специфічним вимогам додатка;

– ефективність. Класи фреймворку оптимізовані для забезпечення високої продуктивності та ефективного використання пам'яті;

– потокобезпека. Фреймворк надає потокобезпечні реалізації колекцій, що дозволяє безпечно використовувати їх у багатопотокових середовищах;

– розширені можливості. Фреймворк пропонує широкий набір операцій та алгоритмів для роботи з колекціями, таких як сортування, фільтрація, пошук та інші.

Java Collections Framework є невід'ємною частиною розробки програм Java. Він надає потужні та гнучкі інструменти для роботи з колекціями даних, спрощуючи процес розробки та покращуючи продуктивність додатків.

2.10 Висновок до другого розділу

У другому розділі роботи було проведено аналіз різних засобів розробки програмного забезпечення, які використовуються для проектування, налагодження та тестування API. Було зроблено вибір найбільш підходящих інструментів та технологій для виконання поставлених завдань. Нижче наведено основні висновки щодо кожного з розглянутих коштів:

1) мова програмування Java. Java є широко використовуваною мовою програмування, що має безліч можливостей для розробки програмного забезпечення. Він надає високу продуктивність, крос-платформність, багату бібліотеку класів та потужні інструменти для роботи з API;

2) бібліотека Swing. Swing є стандартною бібліотекою інтерфейсу користувача для Java. Вона надає широкий набір компонентів та можливостей для створення графічного інтерфейсу користувача. Swing дозволяє розробляти крос-платформні програми з привабливим та інтуїтивно зрозумілим інтерфейсом;

3) бібліотека JavaFX. JavaFX є сучасною платформою для розробки інтерактивних графічних програм на Java. Вона пропонує багатий набір компонентів, можливості для створення анімацій та ефектів, підтримку мультимедіа та взаємодії з веб-вмістом. JavaFX має високу продуктивність і є рекомендованим вибором для розробки нових додатків;

4) бібліотека AWT. AWT (Abstract Window Toolkit) є старою бібліотекою для створення графічного інтерфейсу Java. Вона надає базові компоненти для відображення графічних елементів та взаємодії з користувачем. Однак AWT має обмежені можливості порівняно з Swing та JavaFX, і рекомендується використовувати більш сучасні бібліотеки;

5) Java IO (Input/Output). Java IO надає механізми для роботи з введенням та виведенням даних. Він включає класи для читання та запису файлів, обміну даними через потоки, серіалізації об'єктів та інших операцій введення-

виведення. Java IO надає гнучкий і потужний інструментарій для роботи з різноманітними джерелами та форматами даних;

6) клас HttpURLConnection. HttpURLConnection є класом у Java, який надає зручний спосіб для надсилання HTTP-запитів та отримання HTTP-відповідей. Він дозволяє взаємодіяти з віддаленими серверами, обмінюватися даними та виконувати різні операції з використанням протоколу HTTP;

7) клас JFileChooser. JFileChooser є компонентом інтерфейсу користувача, який дозволяє вибирати файли та директорії на комп'ютері. Він надає зручний діалоговий інтерфейс для вибору файлів, вказівки шляху та керування файловою системою;

8) фреймворк Java Collections Framework. Java Collections Framework є набір інтерфейсів, класів і алгоритмів до роботи з колекціями об'єктів Java. Він надає зручні абстракції та інструменти для зберігання, обробки та маніпулювання групами об'єктів. Java Collections Framework має високу ефективність, розширюваність і широкі можливості для роботи з колекціями.

В результаті аналізу було обрано відповідні інструменти та технології для розробки програмного забезпечення для проектування, налагодження та тестування API. Ці інструменти та технології надають розробникам широкі можливості для створення високоякісних та ефективних програм Java.

РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРОЕКТУВАННЯ, ВІДЛАДКИ ТА ТЕСТУВАННЯ API

3.1 Структура проекту

Структура проекту описана нижче:

1) клас APITester. Головний клас програми, який успадковується від JFrame і представляє головне вікно програми;

2) поля класу APITester:

– urlTextField. JTextField для введення URL-адреси API;

– methodComboBox. JComboBox для вибору способу запиту (GET, POST, PUT, DELETE);

– headersTextArea. JTextArea для введення HTTP-заголовків;

– parametersTextArea. JTextArea для введення параметрів запиту;

– bodyTextArea. JTextArea для введення тіла запиту;

– responseTextArea. JTextArea для відображення відповіді від API;

– sendButton. JButton для надсилання запиту;

– loadFileButton. JButton для завантаження файлу;

– saveFileButton. JButton для збереження файлу.

3) конструктор класу APITester. Встановлює заголовок вікна, розміри та компонування елементів інтерфейсу. Ініціалізує усі компоненти, додає слухачів подій;

4) метод sendRequest(). Приймає URL, метод запиту, заголовки, параметри та тіло запиту. Відправляє HTTP-запит на вказану URL із заданими параметрами та повертає відповідь у вигляді рядка;

5) метод main(). Точка входу у додаток. Створює екземпляр класу APITester і відображає його на екрані.

Структура проекту слідує патерну «Model-View-Controller» (MVC), де APITester виступає в ролі контролера, що управляє взаємодією між моделлю (даними) та поданням (графічним інтерфейсом).

При запуску програми користувач вводить URL-адресу, вибирає метод запиту, задає заголовки, параметри і тіло запиту. Після натискання кнопки «Send Request» викликається метод sendRequest(), який надсилає запит на вказану URL із заданими параметрами. Відповідь від API відображається у текстовій області responseTextArea.

Кнопка «Load File» дозволяє користувачеві обрати файл із тілом запиту та завантажити його в текстову область bodyTextArea. Кнопка «Save File» дозволяє зберегти вміст bodyTextArea у обраний користувачем файл.

Таким чином, структура проекту складається з класу APITester, який є точкою входу і контролером програми, і графічного інтерфейсу, створеного з використанням бібліотеки Swing.

3.2 Архітектура проекту

Архітектуру проекту APITester можна описати так:

1) Model (Модель):

– у даному проекті моделлю є клас APITester, який містить дані та логіку пов'язану із взаємодією з API та обробкою відповідей;

– поля класу APITester представляють дані, такі як URL API, метод запиту, заголовки, параметри і тіло запиту, а також відповідь від API;

– метод sendRequest() виконує відправку HTTP-запиту на вказану URL із заданими параметрами та повертає відповідь у вигляді рядка.

2) View (Подання):

– у виставі використовується бібліотека Swing для створення графічного інтерфейсу користувача (GUI);

– головне вікно програми створюється за допомогою класу JFrame і містить різні компоненти, такі як текстові поля, список, кнопки і текстові області, що випадає, для введення і відображення даних;

– компоненти інтерфейсу розміщені з використанням менеджерів компонування, таких як BorderLayout та GridLayout, щоб забезпечити правильне розташування та вирівнювання елементів.

3) Controller (Контролер):

– контролером у цьому проекті є клас APITester, який управляє взаємодією між моделлю та поданням;

– контролер реагує на дії користувача, такі як натискання кнопки «Send Request», «Load File» та «Save File»;

– при натисканні кнопки "Send Request" викликається метод sendRequest() моделі для виконання запиту та отримання відповіді, який потім відображається у текстовій області;

– кнопка «Load File» та «Save File» також пов'язані з відповідними методами контролера, які виконують завантаження та збереження файлу.

Таким чином, архітектура проекту слідує патерну «Model-View-Controller» (MVC), де модель (APITester) містить дані та бізнес-логіку, представлення (графічний інтерфейс) (JFrame, компоненти Swing) відображає дані користувачеві, а контролер (APITester) управляє взаємодією між моделлю та поданням, обробляє дії користувача та оновлює інтерфейс відповідно до змін у моделі.

3.3 Опис принципу роботи проекту

Принцип роботи проекту APITester можна описати наступним чином по тексту нижче.

Запуск програми:

– додаток запускається викликом методу main() у класі APITester;

– створюється екземпляр класу APITester і відображається головне вікно програми.

Введення даних користувачем:

– користувач вводить URL API у текстове поле urlTextField;

– користувач вибирає метод запиту (GET, POST, PUT, DELETE) з списку methodComboBox;

– користувач вводить заголовки в текстову область headersTextArea;

– користувач вводить параметри запиту до текстової області parametersTextArea;

– користувач вводить тіло запиту в текстову область BodyTextArea.

Надсилання запиту:

– при натисканні кнопки «Send Request» (sendButton) викликається слухач подій ActionListener;

– у слухачі подій метод sendRequest() викликається з передачею в нього введених користувачем даних, таких як URL, метод запиту, заголовки та тіло запиту;

– метод sendRequest() створює об'єкт URL на основі введеного URL API та обєкт HttpURLConnection для відправки HTTP-запиту;

– метод sendRequest() встановлює метод запиту для HttpURLConnection, розбиває введені заголовки на окремі рядки та встановлює їх у HttpURLConnection за допомогою методу setRequestProperty();

– якщо тіло запиту не порожнє, метод sendRequest() встановлює прапор setDoOutput(true) для дозволу виведення даних запиту та записує тіло запиту у HttpURLConnection;

– метод sendRequest() надсилає запит на вказаний URL за допомогою методу getResponseCode() для отримання коду відповіді та методу getInputStream() для читання відповіді від сервера;

– відповідь від сервера читається рядок за рядком і додається до об'єкта StringBuilder;

– код відповіді та текст відповіді зберігаються в рядок response та повертаються з методу sendRequest().

Відображення відповіді:

– отримана відповідь від сервера відображається у текстовій області responseTextArea.

Завантаження та збереження файлу:

– при натисканні кнопки «Load File» (loadFileButton) викликається слухач подій ActionListener;

– у слухачі подій викликається метод loadFile() для вибору файлу за допомогою діалогового вікна JFileChooser;

– обраний файл завантажується і його вміст відображається в текстовій області BodyTextArea;

– при натисканні кнопки «Save File» (saveFileButton) викликається слухач подій ActionListener.

– у слухачі подій викликається метод saveFile() для вибору файлу та збереження вмісту текстової області bodyTextArea в обраний файл.

Таким чином, принцип роботи проекту APITester полягає в тому, що користувач вводить дані для відправки HTTP-запиту, натискає кнопку «Send Request» для відправки запиту, а потім отримує і відображає відповідь від API. Також користувач може завантажувати та зберігати файли з тілом запиту для зручності роботи з даними.

3.4 Опис процесу проектування користувацького інтерфейсу

Процес проектування інтерфейсу користувача (ІІ) в додатку APITester може бути розділений на декілька етапів.

Визначення вимог:

– на початку процесу визначаються основні вимоги до інтерфейсу користувача на основі функціональності програми;

– наприклад, потрібно надати користувачеві можливість введення URL-адреси API, вибору методу запиту, введення заголовків і тіла запиту, а також відображення відповіді від сервера.

Створення схематичного макету:

– створюється схематичний макет інтерфейсу користувача, який візуально представляє розташування компонентів на екрані і основні елементи інтерфейсу;

– можна використовувати папір та олівець або спеціальні програми для створення прототипів інтерфейсу, таких як Adobe XD, Sketch або Figma;

– схематичний макет включає елементи, такі як текстові поля, список, кнопки і текстові області, розташовані відповідно до вимог і логічної організацією інтерфейсу.

Вибір технології розробки інтерфейсу:

– на основі вимог і можливостей вибирається підходяща технологія для розробки інтерфейсу користувача;

– у цьому випадку була обрана бібліотека Swing для створення графічного інтерфейсу Java.

Розміщення компонентів та створення макета:

– на основі схематичного макета та обраної технології створюється макет інтерфейсу користувача;

– використовуються різні менеджери компонування (наприклад, BorderLayout, GridLayout) для встановлення розташування та вирівнювання компонентів на екрані;

– компоненти, такі як текстові поля, список, кнопки та текстові області, що випадає, створюються і додаються на відповідні панелі та вікна відповідно до макету.

Завдання обробників подій:

– для інтерактивності інтерфейсу та реагування на дії користувача, такі як натискання кнопок, задаються обробники подій (наприклад, ActionListener);

– обробники подій зв'язуються з відповідними компонентами (наприклад, кнопками) та містять логіку виконання операцій, пов'язаних з обробкою подій;

– в даному випадку обробник події для кнопки «Send Request» викликає метод sendRequest(), а обробник події для кнопок «Load File» та «Save File» викликають методи loadFile() та saveFile() відповідно.

Оформлення та стилізація інтерфейсу:

– у цьому додатку використані стандартні стилі та елементи інтерфейсу, що надаються бібліотекою Swing;

– можна додати додаткові елементи стилізації, такі як кольори, шрифти та іконки, щоб покращити візуальне враження інтерфейсу та зробити його привабливішим для користувача.

Інтерфейс головного вікна програмного забезпечення зображений на рисунку 3.1.

Таким чином, процес проектування інтерфейсу користувача в додатку APITester включає визначення вимог, створення схематичного макета, вибір технології розробки, розміщення компонентів, завдання обробників подій та оформлення інтерфейсу. Цей процес спрямований на створення зручного та інтуїтивно зрозумілого інтерфейсу для користувача, який забезпечує ефективну взаємодію з програмою.

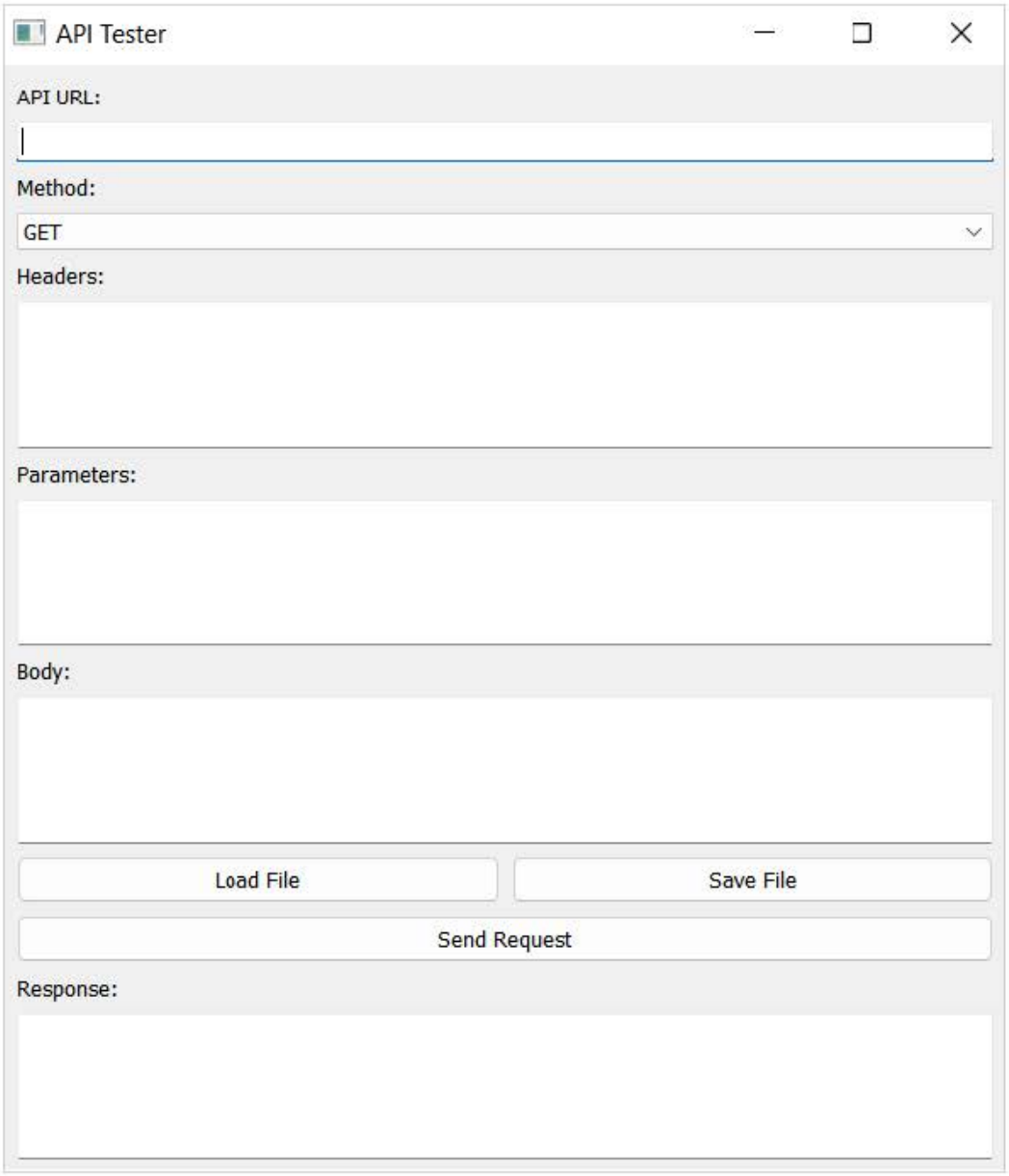

Рисунок 3.1 – Інтерфейс головного вікна програмного забезпечення

3.5 Тестування програмного забезпечення для проектування, відладки та тестування API

На рисунку 3.2 зображене головне вікно програмного забезпечення для проектування, відладки та тестування API.

На рисунку 3.3 наведено заповнення текстового поля для посилання запиту.

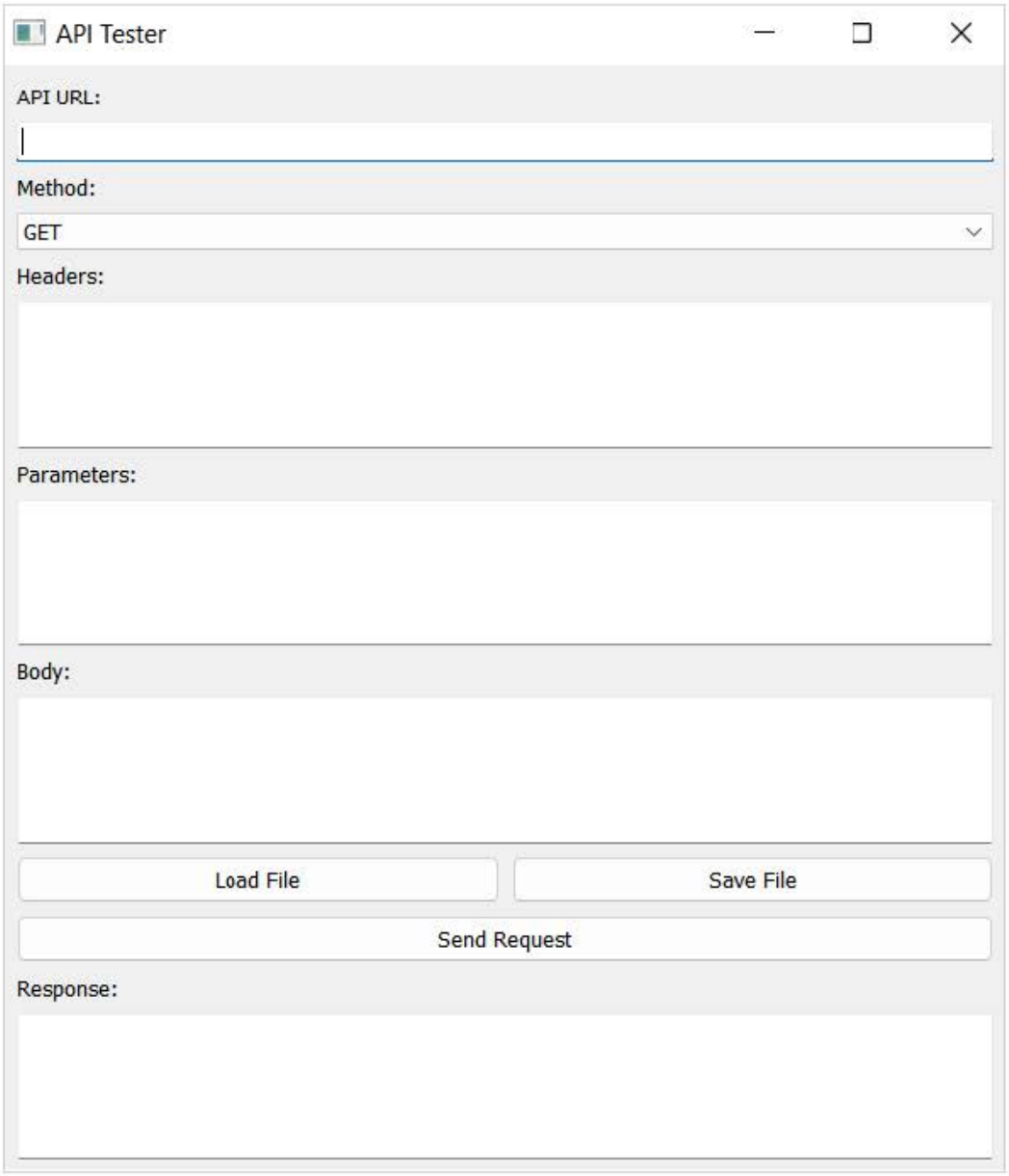

Рисунок 3.2 – Головне вікно програмного застосунку після його запуску

На рисунку 3.4 зображено вибір метода запиту.

На рисунку 3.5 зображена відповідь від сервера та результати запиту.

На рисунку 3.6 зображений процес завантаження файлу.

На рисунку 3.7 зображений процес збереження файлу.

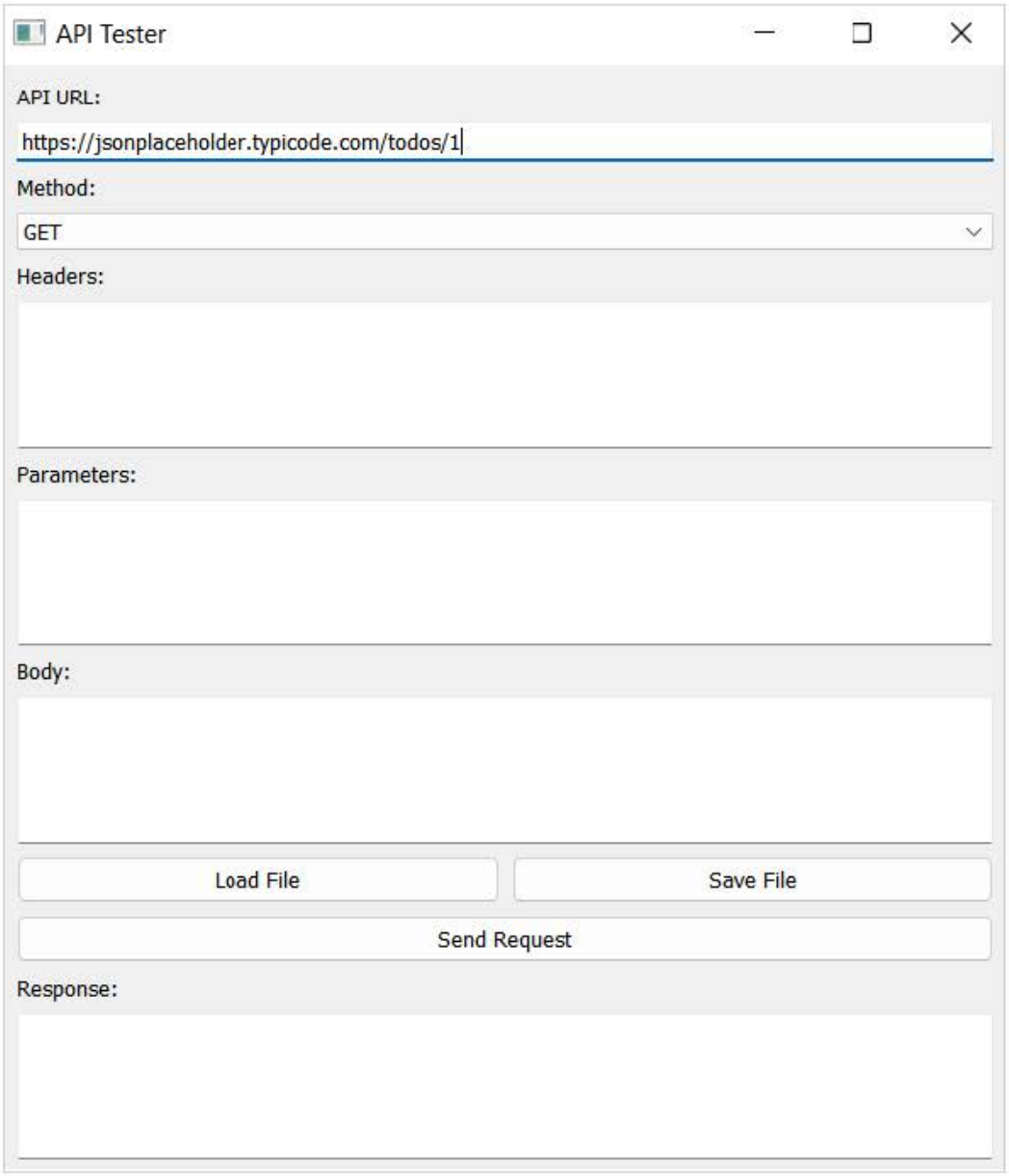

Рисунок 3.3 – Заповнення поля для посилання

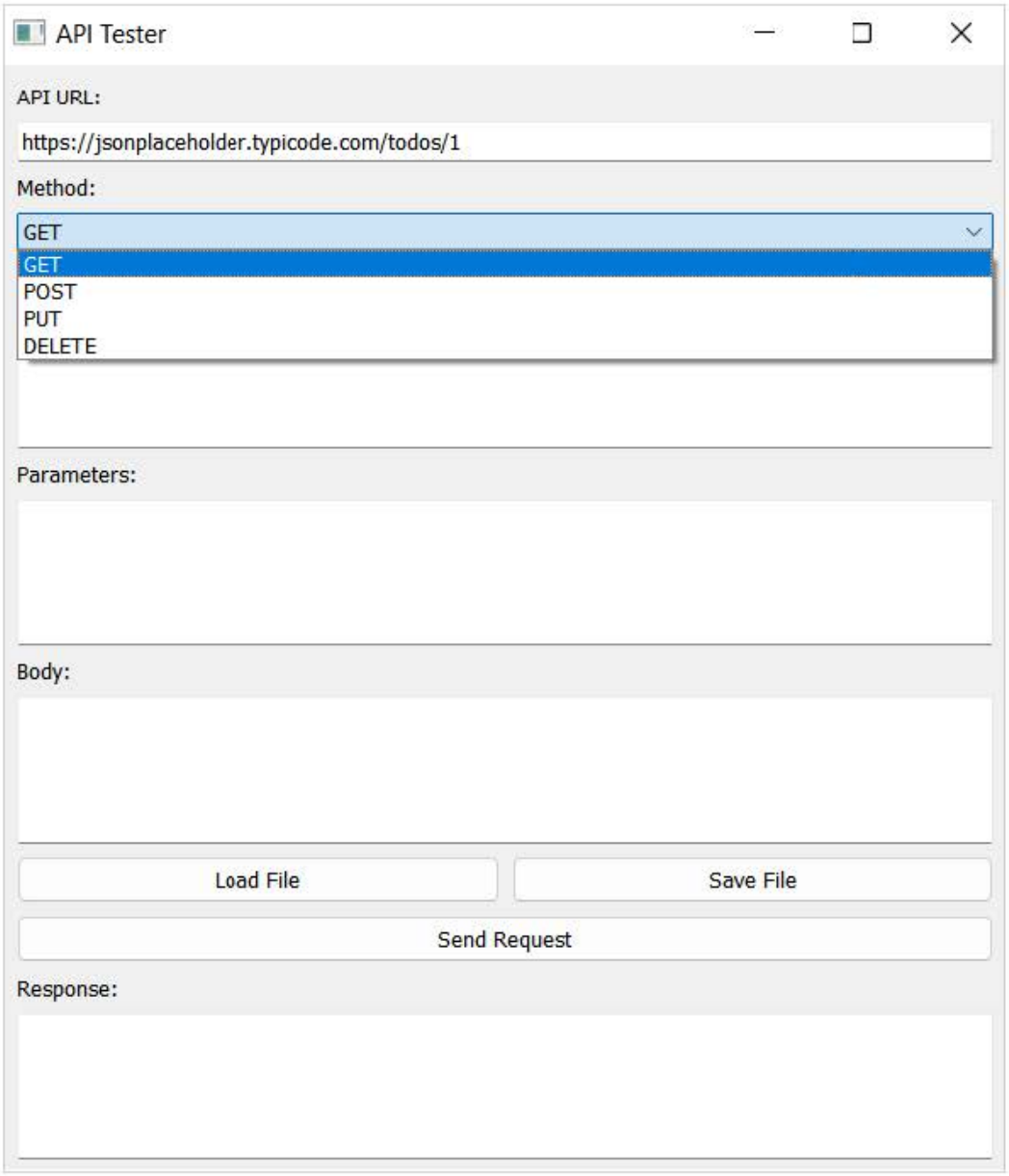

Рисунок 3.4 – Вибір метода запиту

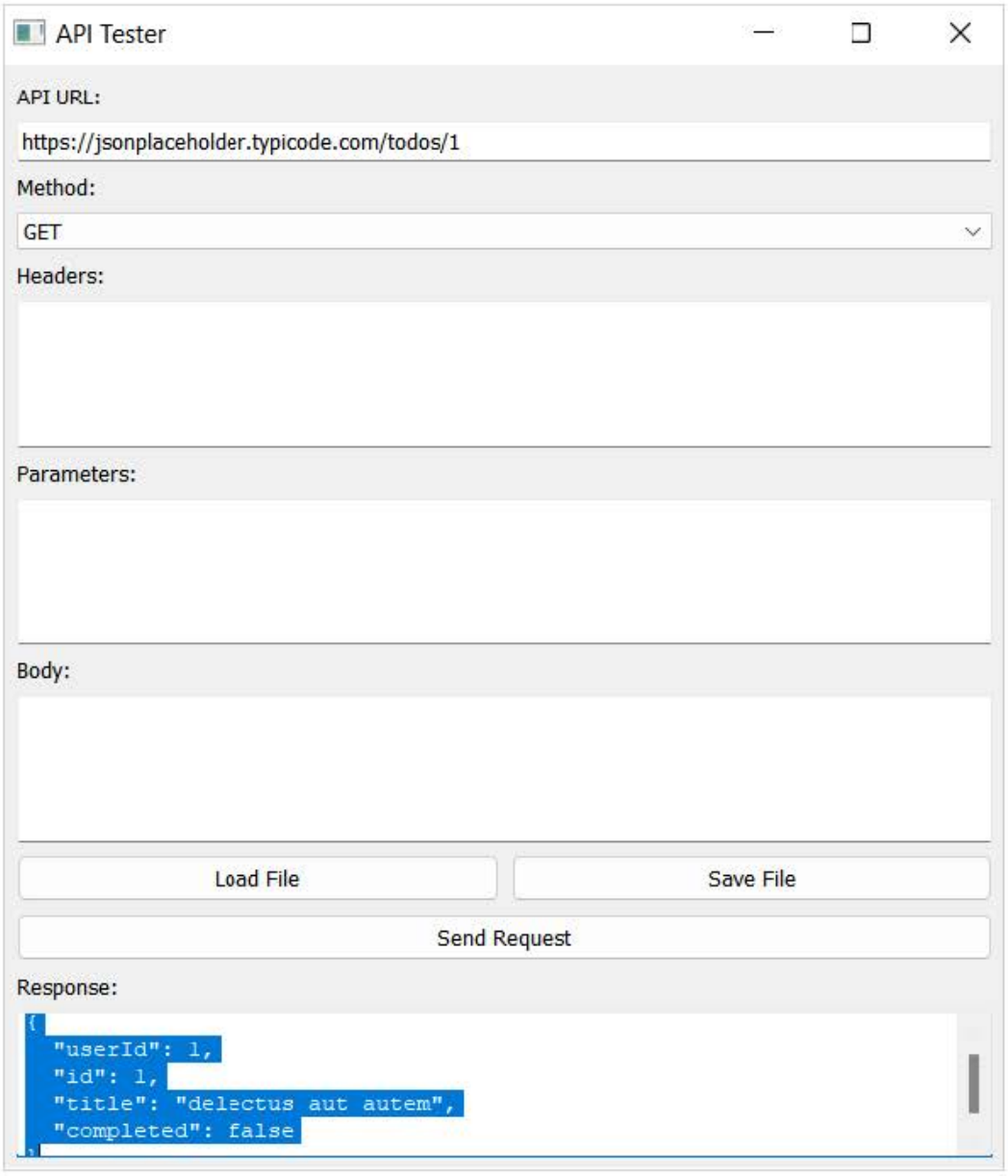

Рисунок 3.5 – Відповідь від сервера. Результати запиту

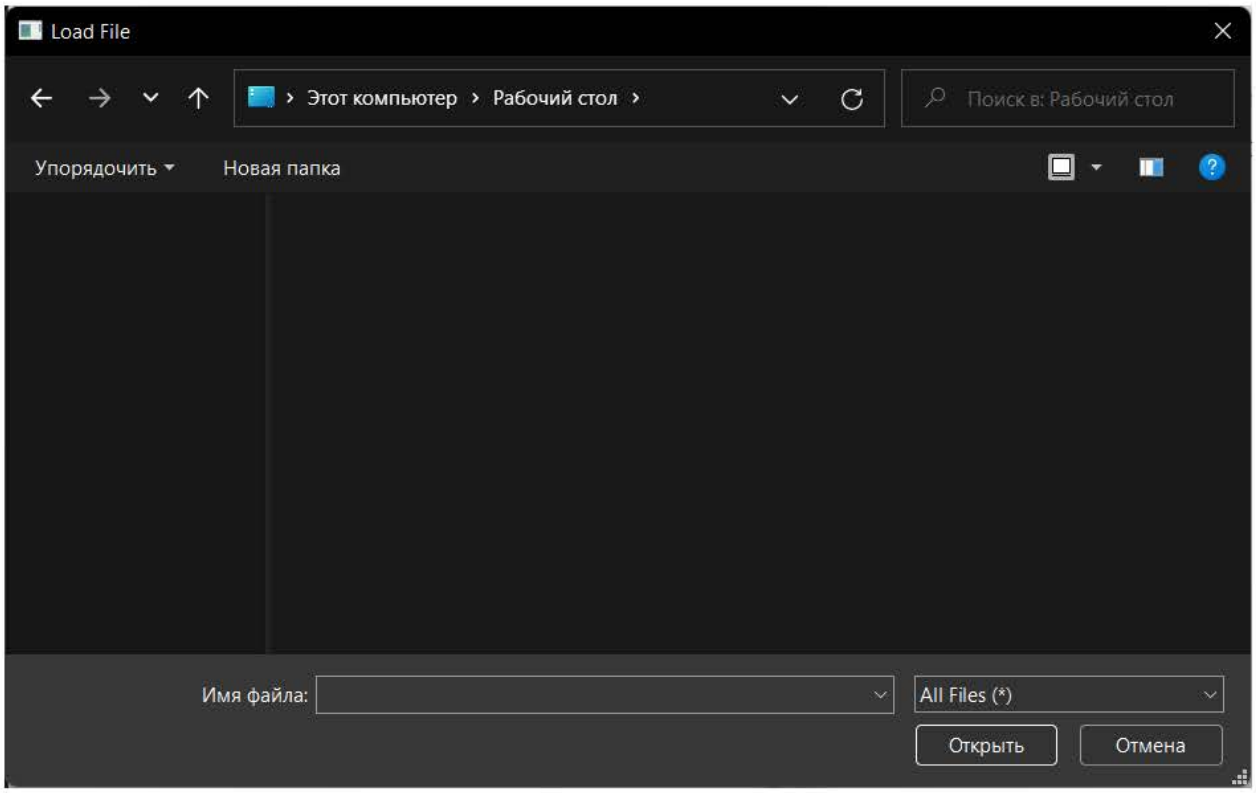

Рисунок 3.6 – Завантаження файлу

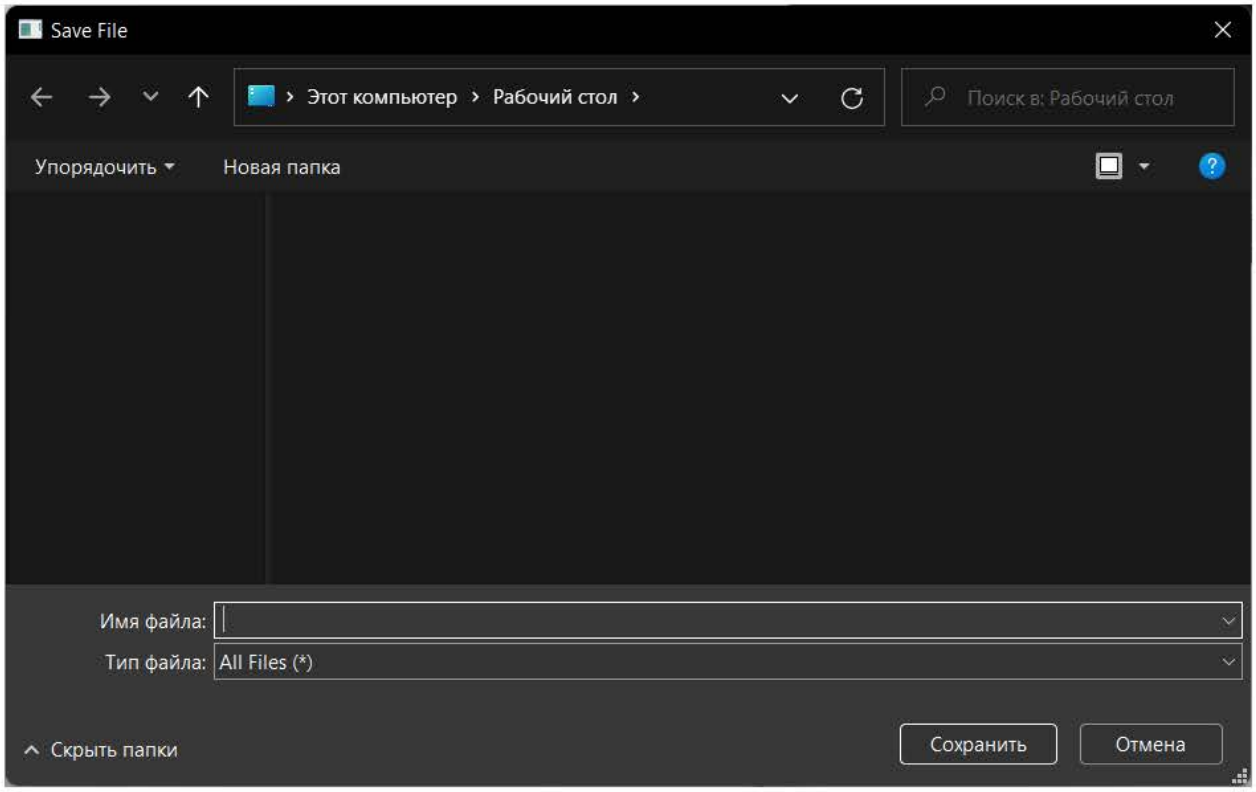

Рисунок 3.7 – Збереження файлу

3.6 Висновки до третього розділу

Розділ 3 «Розроблення програмного забезпечення для проектування, відладки та тестування API» містить опис розробки програмного забезпечення, яке дозволяє проектувати, робити відладку та тестувати API. В ньому розглядаються такі аспекти, як структура проекту, архітектура проекту, принцип роботи програмного забезпечення, проектування користувацького інтерфейсу та тестування програмного забезпечення.

Структура проекту визначає організацію файлів та каталогів у проекті. Цей розділ пояснює, як правильно структурувати проект, щоб забезпечити його зрозумілість та легкість розробки.

Архітектура проекту описує загальну організацію компонентів програмного забезпечення. У даному випадку, описується архітектура програмного забезпечення для проектування, відладки та тестування API, що допомагає розробникам ефективно працювати з API.

Опис принципу роботи проекту включає в себе пояснення того, як програмне забезпечення для проектування, відладки та тестування API працює з API. Це включає у себе обробку запитів, взаємодію з сервером, отримання відповідей та інші деталі роботи з API.

Опис процесу проектування користувацького інтерфейсу пояснює, як було розроблено графічний інтерфейс програмного забезпечення. Він включає опис вікон, елементів керування та їх розміщення, що допомагає користувачам зрозуміти та ефективно використовувати програму.

Тестування програмного забезпечення для проектування, відладки та тестування API визначає процес перевірки програмного забезпечення на належну роботу та виявлення помилок. В цьому розділі пояснюється, як були проведені тести, які види тестів були використані та які результати були отримані.

Загальні висновки до розділу містять коротке підсумкове уявлення про весь розділ і підкреслюють його важливість у контексті розробки програмного

забезпечення для проектування, відладки та тестування API. Вони також можуть включати рекомендації та висновки, що стосуються подальшого вдосконалення та розширення розробленого програмного забезпечення.

ВИСНОВКИ

Метою роботи було розроблення програмного забезпечення для проектування, відладки та тестування API.

Для досягнення поставленої мети роботи був сформульований наступний перелік задач:

1) аналіз існуючих методологій та інструментів. Провести огляд та аналіз існуючих методологій, інструментів та підходів, що використовуються у розробці, налагодженні та тестуванні API. Вивчити їх переваги та недоліки, а також їх застосування для різних типів API;

2) дослідження вимог щодо розробки API. Вивчити вимоги та очікування розробників API, а також потреби та обмеження, з якими вони стикаються в процесі роботи. Визначити основні функціональні та нефункціональні вимоги до програмного забезпечення, яке буде розроблено;

3) проектування інтерфейсу користувача. Розробити інтуїтивно зрозумілий і зручний інтерфейс користувача, що дозволяє розробникам проектувати API і визначати його структуру, функціональність і параметри. Врахувати важливість чіткості та легкості розуміння інтерфейсу для підвищення продуктивності розробників;

4) створення інструментів налагодження та тестування. Розробити інструменти, які допоможуть розробникам виявляти та виправляти помилки в API, а також перевіряти його працездатність та продуктивність. Включити функції автоматичного тестування та перевірки відповідності API стандартам та специфікаціям;

5) інтеграція засобів розробки. Забезпечити інтеграцію розробленого програмного забезпечення з існуючими засобами розробки та популярними інструментами, такими як інтегровані середовища розробки (IDE), системи контролю версій та інструменти безперервної інтеграції та доставки (CI/CD). Це дозволить розробникам легко інтегрувати створювані API у існуючі проекти та системи;

6) розробка документації та версіонування API. Розробити механізми для документування та версіонування API, щоб забезпечити ясність, стабільність та зворотну сумісність інтерфейсу для розробників-споживачів. Включити можливість автоматичного створення документації на основі визначень API;

7) тестування та оцінка результатів. Провести тестування розробленого програмного забезпечення, оцінити його ефективність та надійність, а також провести порівняльний аналіз з існуючими інструментами та методами розробки API. Виправити виявлені проблеми та покращити функціональність за результатами тестування.

Ці завдання допомогли у досягненні мети роботи, а саме розробці програмного забезпечення, здатного полегшити процес проектування, налагодження та тестування API, підвищити якість інтерфейсів, що розробляються, і покращити досвід розробників.

Об'єктом дослідження були процеси роботи програмного забезпечення для проектування, відладки та тестування API.

Предметом дослідження було апаратно-програмне забезпечення для розроблення програмного забезпечення для проектування, відладки та тестування API.

Практичне значення одержаних результатів полягає у підвищенні якості проектування, відладки та тестування API.

Результатами роботи є програмне забезпечення для автоматизації проектування, відладки та тестування API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schildt, H. Java: The Complete Reference, 2019. 1376 p.

2. Schildt, H. Swing: A Beginner's Guide, 2006. 608 p.

3. Zukowski, J. Java AWT Reference, 1997. 486 p.

4. Friesen, J. Java I/O, NIO, and NIO.2, 2011. 832 p.

5. Harold, E. R. Java Network Programming, 2017. 708 p.

6. Marinacci, J., Adamson, C. Swing Hacks: Tips and Tools for Killer GUIs, 2005. 542 p.

7. Bloch, J. Effective Java, 3rd edition, 2018. 416 p. (Chapter 8: "General Programming" covers Java Collections Framework)

8. Lafore, R. Data Structures and Algorithms in Java, 2017. 800 p. (Chapter 8: "Collections" covers Java Collections Framework)

9. Eckel, B. Thinking in Java, 4th edition, 2006. 1150 p.

10. Sierra, K., Bates, B. Head First Java, 2nd edition, 2005. 688 p.

11. Horstmann, C. S., Cornell, G. Core Java Volume I - Fundamentals, 11th edition, 2018. 928 p.

12. Horstmann, C. S., Cornell, G. Core Java Volume II - Advanced Features, 11th edition, 2018. 1040 p.

13. Lea, D. Concurrent Programming in Java: Design Principles and Patterns, 2nd edition, 2008. 432 p.

14. Bloch, J. Java Concurrency in Practice, 2006. 432 p.

15. Lea, D. Data Structures and Algorithms in Java, 2nd edition, 1998. 684 p.

16. Flanagan, D. Java in a Nutshell, 7th edition, 2019. 1194 p.

17. Bloch, J. Effective Java, 3rd edition, 2018. 416 p.

18. Eckel, B. Thinking in Java, 4th edition, 2006. 1150 p.

19. Sierra, K., Bates, B. Head First Java, 2nd edition, 2005. 688 p.

20. Horstmann, C. S., Cornell, G. Core Java Volume I - Fundamentals, 11th edition, 2018. 928 p.

21. Horstmann, C. S., Cornell, G. Core Java Volume II - Advanced Features, 11th edition, 2018. 1040 p.

22. Lea, D. Concurrent Programming in Java: Design Principles and Patterns, 2nd edition, 2008. 432 p.

23. Ries, E. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, 2011. 336 p.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;


public class APITester extends JFrame
{
   private JTextField urlTextField;
   private JComboBox<String> methodComboBox;
   private JTextArea headersTextArea;
   private JTextArea parametersTextArea;
   private JTextArea bodyTextArea;
   private JTextArea responseTextArea;
   private JButton sendButton;
   private JButton loadFileButton;
   private JButton saveFileButton;

   public APITester()
   {
      setTitle("API Tester");
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 400);
        setLayout(new BorderLayout());

        // Створення панелі верхньої частини вікна
        JPanel topPanel = new JPanel(new GridLayout(5, 2));
        topPanel.add(new JLabel("API URL:"));

        urlTextField = new JTextField();
        topPanel.add(urlTextField);
        topPanel.add(new JLabel("Method:"));

        methodComboBox = new JComboBox<>(new String[]{"GET", "POST",
"PUT", "DELETE"});
        topPanel.add(methodComboBox);
        topPanel.add(new JLabel("Headers:"));

        headersTextArea = new JTextArea();
        topPanel.add(new JScrollPane(headersTextArea));
        topPanel.add(new JLabel("Parameters:"));

        parametersTextArea = new JTextArea();
        topPanel.add(new JScrollPane(parametersTextArea));
        topPanel.add(new JLabel("Body:"));

        bodyTextArea = new JTextArea();
        topPanel.add(new JScrollPane(bodyTextArea));

        add(topPanel, BorderLayout.NORTH);
```

```
        // Створення панелі нижньої частини вікна
        JPanel bottomPanel = new JPanel(new BorderLayout());
        bottomPanel.add(new JLabel("Response:"), BorderLayout.NORTH);

        responseTextArea = new JTextArea();
        bottomPanel.add(new JScrollPane(responseTextArea),
BorderLayout.CENTER);

        sendButton = new JButton("Send Request");
        bottomPanel.add(sendButton, BorderLayout.SOUTH);

        add(bottomPanel, BorderLayout.CENTER);

        // Створення кнопок для завантаження та збереження файлів
        loadFileButton = new JButton("Load File");
        saveFileButton = new JButton("Save File");
        JPanel fileButtonPanel = new JPanel(new FlowLayout());

        fileButtonPanel.add(loadFileButton);
        fileButtonPanel.add(saveFileButton);

        add(fileButtonPanel, BorderLayout.SOUTH);

        // Додавання обробників подій для кнопок

        sendButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
```

```
        // Отримання введених користувачем даних
        String url = urlTextField.getText();
        String method = (String) methodComboBox.getSelectedItem();
        String headers = headersTextArea.getText();
        String parameters = parametersTextArea.getText();
        String body = bodyTextArea.getText();

        // Відправлення запиту та отримання відповіді
        String response = sendRequest(url, method, headers, parameters, body);

        // Відображення отриманої відповіді у вікні
        responseTextArea.setText(response);
    }
});

loadFileButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Створення діалогу вибору файлу
        JFileChooser fileChooser = new JFileChooser();
        int returnValue = fileChooser.showOpenDialog(null);

        if (returnValue == JFileChooser.APPROVE_OPTION)
        {
            File selectedFile = fileChooser.getSelectedFile();

            try
            {
```

```
            // Завантаження вмісту файлу
            FileReader fileReader = new FileReader(selectedFile);
            BufferedReader bufferedReader = new BufferedReader(fileReader);
            StringBuilder fileContent = new StringBuilder();
            String line;

            while ((line = bufferedReader.readLine()) != null)
            {
               fileContent.append(line).append("\n");
            }
            bufferedReader.close();

            // Встановлення вмісту файлу у вікні
            bodyTextArea.setText(fileContent.toString());
         } catch (IOException ex)
         {
            ex.printStackTrace();
         }
      }
   }
});

saveFileButton.addActionListener(new ActionListener()
{
   @Override
   public void actionPerformed(ActionEvent e)
   {
      // Створення діалогу збереження файлу
      JFileChooser fileChooser = new JFileChooser();
      int returnValue = fileChooser.showSaveDialog(null);
```

```
            if (returnValue == JFileChooser.APPROVE_OPTION)
            {
                File selectedFile = fileChooser.getSelectedFile();

                try
                {
                    // Запис вмісту з поля у файл
                    FileWriter fileWriter = new FileWriter(selectedFile);
                    BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);

                    bufferedWriter.write(bodyTextArea.getText());
                    bufferedWriter.close();
                } catch (IOException ex)
                {
                    ex.printStackTrace();
                }
            }
        }
    });
}

private String sendRequest(String url, String method, String headers, String
parameters, String body)
{
    try
    {
        // Створення URL-об'єкту та відкриття з'єднання
        URL apiUrl = new URL(url);
```

```
        HttpURLConnection connection = (HttpURLConnection)
apiUrl.openConnection();
        connection.setRequestMethod(method);

        // Налаштування заголовків
        String[] headerLines = headers.split("\n");

        for (String line : headerLines)
        {
           String[] parts = line.split(":");

           if (parts.length == 2)
           {
              String key = parts[0].trim();
              String value = parts[1].trim();

              connection.setRequestProperty(key, value);
           }
        }

        // Обробка параметрів
        Map<String, String> parametersMap = new HashMap<>();
        String[] parameterLines = parameters.split("\n");

        for (String line : parameterLines)
        {
           String[] parts = line.split("=");

           if (parts.length == 2)
           {
```

```
                String key = parts[0].trim();
                String value = parts[1].trim();

                parametersMap.put(key, value);
            }
        }

        // Додавання параметрів до URL-адреси
        if (!parametersMap.isEmpty())
        {
            StringBuilder parametersBuilder = new StringBuilder();

            for (String key : parametersMap.keySet())
            {
                if (parametersBuilder.length() > 0)
                {
                    parametersBuilder.append("&");
                }
                parametersBuilder.append(key).append("=").append(parametersMap.g
et(key));
            }

            String query = parametersBuilder.toString();
            url = url + "?" + query;
            apiUrl = new URL(url);
            connection = (HttpURLConnection) apiUrl.openConnection();
            connection.setRequestMethod(method);

            // Налаштування заголовків повторно
            for (String line : headerLines)
```

```
            {
                String[] parts = line.split(":");
                if (parts.length == 2)
                {
                    String key = parts[0].trim();
                    String value = parts[1].trim();

                    connection.setRequestProperty(key, value);
                }
            }
        }

        // Відправлення тіла запиту, якщо воно присутнє
        if (!body.isEmpty())
        {
            connection.setDoOutput(true);
            connection.getOutputStream().write(body.getBytes());
        }

        // Отримання коду статусу та відповіді з сервера
        int responseCode = connection.getResponseCode();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        StringBuilder response = new StringBuilder();
        String line;

        while ((line = reader.readLine()) != null)
        {
            response.append(line);
            response.append("\n");
```

```
            }
            reader.close();

            // Повернення результату відправлення запиту
            return "Status Code: " + responseCode + "\n\n" + response.toString();
        } catch (IOException e)
        {
            e.printStackTrace();

            return "An error occurred: " + e.getMessage();
        }
    }

    public static void main(String[] args)
    {
        SwingUtilities.invokeLater(new Runnable()
        {
            @Override
            public void run()
            {
                // Створення і відображення головного вікна програми
                APITester apiTester = new APITester();
                apiTester.setVisible(true);
            }
        });
    }
}
```