НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах»

Виконав:
студент IV курсу, групи КП-61
Свинарчук Максим Владиславович _____

Керівник:
Старший викладач кафедри ПЗКС,
Гадиняк Руслан Анатолійович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Старший викладач кафедри ПМА,
Мальчиков Володимир Вікторович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Свинарчуку Максиму Владиславовичу

1. Тема проєкту «Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах», керівник проєкту Гадиняк Руслан Анатолійович, старший викладач, затверджені наказом по університету від « 25 » травня 2020 р. № 1181-с

2. Термін подання студентом проєкту « 15 » червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз предметної області та існуючих рішень;
   - аналіз мов програмування, технологій розроблення веб-додатків та систем для виявлення програмного плагіату;
   - структурна організація розроблених програмних засобів;
   - аналіз реалізації програмного забезпечення.

5. Перелік обов'язкового графічного матеріалу:
   - функціональність веб-додатка (креслення);
   - структура бази даних (креслення);
   - алгоритм роботи веб-додатка для виявлення програмного плагіату (плакат);
   - структурна схема веб-додатка (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання « 31 » жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	16.11.2019	
2.	Розроблення та узгодження технічного завдання	09.12.2019	
3.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
4.	Розроблення структури веб-додатка	16.01.2020	
5.	Підготовка матеріалів другого розділу дипломного проєкту	01.03.2020	
6.	Розроблення дизайну сторінок та графічних елементів	22.03.2020	
7.	Програмна реалізація та тестування веб-додатка	12.04.2020	
8.	Підготовка третього розділу дипломного проєкту	25.04.2020	
9.	Підготовка четвертого розділу дипломного проєкту	02.05.2020	
10.	Підготовка графічної частини дипломного проєкту	19.05.2020	
11.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Максим СВИНАРЧУК

Керівник проєкту Руслан ГАДИНЯК

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню веб-додатка для автоматизації виявлення плагіату в навчальних програмних проєктах.

У роботі виконано аналіз особливостей виявлення програмного плагіату, обґрунтовано ідею автоматизації даного процесу для перевірки навчальних робіт. Виконано пошук та порівняння наявних аналогів за критеріями на основі вимог до програмного продукту дипломного проєкту.

Веб-додаток призначений для викладачів курсів з програмування, як допоміжний інструмент з метою попередження плагіату коду в програмних проєктах студентів. Для автоматизації виявлення плагіату в програмних проєктах використано систему JPlag, налаштовано взаємодію з веб-сервісами для розміщення програмних проєктів на базі VCS, такими як GitHub і Bitbucket. Інформаційна безпека веб-додатка реалізована за допомогою розподілу прав доступу – процедури реєстрації та авторизації. Гостьовий доступ передбачає тільки можливість виявлення плагіату серед власноруч завантажених проєктів. Зареєстрований користувач має доступ до функцій, таких як створення курсів, груп, додавання інформації про студентів з посиланням на їх репозиторії, створення завдань на перевірку. Процес виявлення плагіату здійснюється автоматично в заданий користувачем час. Результатом роботи веб-додатка є статистика збігу програмного коду з виділенням ділянок, підозрюваних на плагіат.

У даному дипломному проєкті розроблено: архітектуру серверної та клієнтської частини веб-додатка, структуру бази даних, модуль взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS, модуль взаємодії із системою для виявлення плагіату в програмних проєктах, модуль для організації робочого процесу користувача, реалізовано графічні елементи та дизайн веб-сторінок.

ABSTRACT

This diploma project is dedicated to creating a web application to automate the detection of plagiarism in educational software projects.

The work contains features analysis of detecting software plagiarism, substantiates the idea of automation of this process to check educational works. Search and comparison of available analogs by criteria based on requirements for the diploma project software were performed.

The web application is intended for teachers of programming courses, as an auxiliary tool to prevent plagiarism in students' software projects. JPlag system was used to automate plagiarism detection in software projects, interaction with web services was configured to host software projects based on VCS, such as GitHub and Bitbucket. The information security of the web application is implemented using the distribution of access rights – registration and authorization procedures. Guest access provides only the possibility of detecting plagiarism among personally loaded projects. A registered user has access to features, such as creating courses, groups, adding information about students with a link to their repositories, and creating tasks for verification. The process of detecting plagiarism is carried out automatically at a user-specified time. The result of the web application is statistics of code similarity, with highlighting areas suspected of plagiarism.

The following artefacts have been developed in this diploma project: the architecture of the server and client part of the web application, the database structure, the interaction module with a web services for hosting projects based on VCS, the interaction module with system for detecting software plagiarism, workflow organization module, implemented graphic elements and web pages designs.

ДП.045440-01-90 Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-додаток для	5	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Технічне завдання		
ДП.045440-03-81	Веб-додаток для	78	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-додаток для	5	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-додаток для	16	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-додаток для	1	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Функціональність		
	веб-додатка.		
	UML-діаграма		
	прецедентів		
ДП.045440-07-99	Веб-додаток для	1	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Структура бази даних.		
	ER-діаграма		
ДП.045440-08-98	Веб-додаток для	1	
	автоматизації виявлення		
	плагіату в навчальних		
	програмних проєктах.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ПЛАГІАТУ В НАВЧАЛЬНИХ ПРОГРАМНИХ ПРОЄКТАХ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СВИНАРЧУК

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для викладачів інформатики та фахівців ІТ-галузі, які проводять навчальні курси, як допоміжний інструмент з метою попередження плагіату коду в програмних проєктах для якісного оцінювання студентів.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-додаток повинен забезпечувати такі основні функції:

1. Підтримка наступних ролей: Гість та Користувач.
2. Можливість реєстрації нового Користувача, процедуру автентифікації та авторизації.
3. Організація Користувачем навчальних курсів та окремих груп в рамках кожного курсу.
4. Створення завдань на реалізацію програмних проєктів окремо для кожної групи.

5. Автоматизація завантаження програмних проєктів для створених Користувачем груп із веб-сервісів для хостингу проєктів на базі систем керування версіями.
6. Користувач має можливість виявлення плагіату серед множини програмних проєктів в рамках створеної задачі для конкретної групи, цілого курсу або незалежно власноруч завантажених проєктів.
7. Гість має можливість виявлення плагіату лише серед множини власноруч завантажених програмних проєктів без збереження результату.
8. Можливість запуску процесу виявлення програмного плагіату для створеної Користувачем групи як власноруч, так і автоматично в конкретно заданий час.
9. Вивід та збереження результатів процесу виявлення плагіату в інформативних графіках, діаграмах та таблицях.
10. Можливість переглядати та порівнювати підозрюваний на плагіат вихідний код програм двох проєктів одночасно.

Розроблення серверної частини виконати на мові програмування Java, клієнтської частини – використовуючи бібліотеку React для забезпечення інтерактивності та динамічності веб-сторінки.

Додаткові вимоги:

1. Локалізація веб-додатка українською мовою.
2. Наявність адаптивного дизайну для пристроїв із різною роздільною здатністю екрану.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1. Пояснювальна записка.

2. Програма та методика тестування.

3. Керівництво користувача.

4. Креслення:

– «Функціональність веб-додатка. UML-діаграма прецедентів»;

– «Структура бази даних. ER-діаграма».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту..... 16.11.2019

Розроблення та узгодження технічного завдання 09.12.2019

Підготовка матеріалів першого розділу дипломного проєкту..... 30.12.2019

Розроблення структури веб-додатка 16.01.2020

Підготовка матеріалів другого розділу дипломного проєкту..... 01.03.2020

Розроблення дизайну сторінок та графічних елементів..... 22.03.2020

Програмна реалізація та тестування веб-додатка..... 12.04.2020

Підготовка третього розділу дипломного проєкту..... 25.04.2020

Підготовка четвертого розділу дипломного проєкту..... 02.05.2020

Підготовка графічної частини дипломного проєкту..... 19.05.2020

Оформлення документації дипломного проєкту..... 26.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ПЛАГІАТУ В НАВЧАЛЬНИХ ПРОГРАМНИХ ПРОЄКТАХ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СВИНАРЧУК

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Токенізація – це процес перетворення послідовності символів в послідовність токенів. Токен – об'єкт, що утворюється із лексеми, залежно від її типу. Наприклад, вираз «int i = 3 + 7» може бути перетворений в наступний набір токенів – «I @ = num + num».

VCS – Version Control System (укр. система керування версіями) – це програмне забезпечення для полегшення роботи зі змінною інформацією, дозволяє зберігати кілька версій одного і того ж документа, при необхідності повертатися до попередніх версій, визначати, хто і коли зробив ту чи іншу зміну.

Обфускатори – це програмне забезпечення, яке навмисно виконує заплутування коду, тобто змінює код програми таким чином, щоб зберегти його функціональність, але зробити майже неможливим аналіз алгоритму роботи для людини.

Фреймворк – це програмна платформа, що забезпечує загальні функціональні можливості, може вибірково змінюватися за допомогою додаткового написаного користувачем коду. Надає стандартний спосіб створення і розгортання додатків і являє собою універсальне програмне середовище багаторазового використання.

MVVM – Model-View-ViewModel – спосіб проєктування архітектури додатка, що розділяє його на 3 частини: Model – основна логіка програми, View – абстракція представлення, інтерфейс користувача, ViewModel – модель представлення, взаємодіє з View за допомогою механізму зв’язування. Використовується в ситуації, коли можливо зв'язування даних без введення спеціальних інтерфейсів View.

MVC – Model-View-Controller – спосіб проєктування архітектури додатка, використовується в ситуації, коли зв'язок між уявленням і іншими частинами програми неможливий.

CLI – Command Line Interface – різновид текстових інтерфейсів для взаємодії користувача з програмним забезпеченням.

DI – Dependency Injection (укр. впровадження залежності) – шаблон проєктування, в якому створенням зовнішніх залежностей кожного компонента виконується спеціальним механізмом без участі користувача.

СУБД – система управління базами даних.

ORM – Object-Relational Mapping (укр. Об'єктно-реляційна проєкція) – це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між сховищем даних і об'єктами програмування.

ER (Entity-relationship) діаграма – схема, яка показує відношення сутностей, що зберігаються в базі даних.

ACID – Atomicity (атомарність), Consistency (узгодженість), Isolation (ізольованість), Durability (довговічність) – вимоги до транзакційної моделі системи (наприклад, СУБД).

JPA (Java Persistence API) – це специфікація Java EE і Java SE, що описує систему управління збереженням Java-об'єктів в таблиці реляційних баз даних в зручному вигляді.

JSX – розширений синтаксис JavaScript, який використовує React для опису інтерфейсу користувача.

Токен авторизації – засіб авторизації, спеціально згенерована на секретному ключі послідовність символів.

API – Application Programming Interface (укр. прикладний програмний інтерфейс) – опис способів взаємодії з програмним забезпеченням.

RESTful API – це загальні принципи організації взаємодії з програмним забезпеченням за допомогою протоколу HTTP через методи GET, POST, PUT та DELETE.

URL – Uniform Resource Locator (укр. уніфікований локатор ресурсів) – стандартизована адреса ресурсу в мережі Інтернет.

DTO – Data Transfer Object (укр. об’єкт для передачі даних) – шаблон проєктування, який використовується для передачі даних між додатками.

DAO – Data Access Object (укр. об'єкт доступу до даних) – абстрактний інтерфейс до типу бази даних або механізму зберігання.

JRE – Java Runtime Environment (укр. середовище виконання для Java) – мінімальна реалізація віртуальної машини Java, необхідна для виконання Java-додатків.

CRUD – Create, Read, Update, Delete – базові функції управління даними.

CSRF – Cross-Site Request Forgery (укр. міжсайтова підробка запиту) – вид атак на відвідувачів веб-сайтів, які використовують недоліки протоколу HTTP.

ВСТУП

Головне завдання кожного викладача – це поділитися власними знаннями та досвідом зі студентами. Не менш важливою складовою навчального процесу є практичне застосування навичок – написання творів, розв'язання математичних задач, створення програм. Перевіряючи дані роботи, викладач оцінює те, як студенти засвоїли викладений матеріал. Але для якісного оцінювання важлива не лише наявність роботи та правильність її виконання, а й розуміння того, що вона сумлінно зроблена силами студента без запозичень. Проблема плагіату набула широкого поширення не тільки в літературі та мистецтві, а й в програмному забезпеченні, особливо в освітній сфері.

Виявлення плагіату в програмних проєктах – це окрема задача, яка вимагає індивідуального підходу до реалізації. Це пов’язано з особливістю програмних структур у порівнянні зі звичайним текстом. Існує невелика кількість систем, які здатні виявляти програмний плагіат. У своїй основі вони використовують різні алгоритми та надають результати, переглядаючи які користувач повинен самостійно визначити факт плагіату.

Головний недолік систем для виявлення програмного плагіату – необхідність виконувати багато ручної роботи, постійно збираючи проєкти студентів та завантажувати їх в систему для перевірки. Вирішити цю проблему допомагають різні веб-сервіси для розміщення програмних проєктів на базі VCS. Вони є обов’язковим інструментом в роботі будь-якого IT-спеціаліста та вже використовуються багатьма студентами в навчальних цілях.

Створення програмного продукту, який дозволить викладачам в зручний спосіб перевіряти роботи студентів на плагіат без необхідності маніпуляцій сотнями файлів, є актуальною задачею. Адже навчальних курсів, пов’язаних з програмуванням, з часом стає все більше, а наявні

сервіси для виявлення плагіату не пропонують вирішення проблеми ручного завантаження робіт студентів.

Даний дипломний проєкт присвячений створенню веб-додатка для автоматизації виявлення плагіату в навчальних програмних проєктах з використанням системи для виявлення програмного плагіату та веб-сервісів для розміщення програмних проєктів на базі VCS.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз особливостей виявлення програмного плагіату

Плагіат коду – це несанкціоноване повторне використання програмної структури та її реалізації певною мовою програмування.

Однозначно визначити програмний плагіат вкрай складно. Під даним терміном слід розуміти випадок, коли між вихідними кодами двох програм є істотна, на рівні мови програмування, схожа частина. При цьому похідна програма отримується з оригінальної за допомогою різних технік приховування факту плагіату, характерних для програмних структур. Для вирішення даної проблеми необхідно застосовувати зовсім інші підходи – відмінні від тих, які використовуються для виявлення текстового плагіату.

Також варто відмітити, що система виявлення плагіату не може бути повністю автоматичною. Іноді програмну структуру можна реалізувати тільки декількома способами. У таких випадках не коректно вважати це плагіатом. Тому кінцеве рішення має залишатися за відповідальною особою. При цьому система повинна максимально полегшити задачу пошуку дублікатів коду, автоматизуючи процес перевірки та надаючи достатню кількість інформації для прийняття остаточного рішення.

1.1.1. *Варіанти модифікації програмного коду*

Можна виділити чотири типи модифікації програмного коду для приховування факту плагіату:

1. Два фрагменти коду є повністю ідентичними, або змінені коментарі, пробільні символи чи відступи (лістинг 1.1).

Лістинг 1.1. Модифікація програмного коду першого типу

```
if (i <= j)
   x = z + j; // Коментар 1
else
   x = z - i; // Коментар 2
```

```
if (i<=j)  // Змінений Коментар 1
   x=z+j;
} else     // Змінений Коментар 2
   x=z-i;
```

2. Структурно / синтаксично два фрагмента коду ідентичні, за винятком імен ідентифікаторів, літералів, типів змінних, розмітки (лістинг 1.2). Допускаються всі зміни з пункту 1.

Лістинг 1.2. Модифікація програмного коду другого типу

```
if (i <= j)
   x = z + j; // Коментар 1
   x = z + 3;
else
   x = z - i; // Коментар 2
```

```
if (t <= k)
{              // Змінений Коментар 1
   a = b + k;
   a = b + 7; // Новий Коментар 3
} else
{
   a = b - t; // Змінений Коментар 2
}
```

3. Один фрагмент коду був отриманий шляхом копіювання іншого фрагмента коду, але з додаванням і/або видаленням операторів мови (лістинг 1.3). При цьому зберігаються всі зміни властиві типу 2.

Лістинг 1.3. Модифікація програмного коду третього типу

```
if (i <= j)
   x = z + j; // Коментар 1
   x = z + 3;
else
   x = z - i; // Коментар 2
```

```
if (t <= k)
{              // Змінений Коментар 1
   a = b + k;
   r = 100;    // Новий Вираз
   a = b + 7; // Новий Коментар 3
} else
{
   a = b - t; // Змінений Коментар 2
}
```

4. Два фрагмента синтаксично реалізовані по-різному, але виконують однакові дії (лістинг 1.4).

Лістинг 1.4. Модифікація програмного коду четвертого типу

```
int res, i=1;
for (res=1; i<=VALUE; i++){
   res=res*i;
}
```

```
int factorial(int n) {
   if (n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

Проаналізувавши описані вище методи модифікації програмного коду, можна сказати, що перший і другий типи належать до незначних змін. Третій тип розпізнати складніше, зазвичай до нього можна віднести ділянки коду, які спеціально намагалися замаскувати, вносячи додатковий «шум». Четвертий тип майже не зустрічається в навчальних проєктах в силу складності своєї реалізації. До четвертого типу також можна віднести переписування коду з однієї мови програмування в іншу. Такий тип видозміни необов'язковий для виявлення і не розглядається у даній роботі.

1.1.2. *Класифікація існуючих підходів для виявлення програмного плагіату [1, 2]*

Системи для виявлення програмного плагіату використовують різні підходи та мають відповідну класифікацію.

Виявлення плагіату методом текстового порівняння

Використовується для виявлення звичайного текстового плагіату. Вихідний код розглядається «як є». Документи порівнюються дослівно з можливим перекриттям тексту. Очевидно, що такий підхід неефективний для виявлення програмного плагіату. Мінімальна модифікація коду першого типу, що не впливає на результат роботи програми, забезпечує оригінальність при перевірці в системах, які використовують даний підхід.

Виявлення плагіату на основі атрибутів коду

Схожість коду визначається на основі атрибутів. Порівнюється кількість певних атрибутів коду (операторів, операндів, команд присвоєння, змінних, ключових слів), при цьому вважається, що подібні коди програм мають аналогічні показники. Системи на основі атрибутів вразливі до додавання і видалення коду. Третього типу модифікації достатньо, щоб уникнути виявлення плагіату (лістинг 1.5). Відповідно, системи на основі атрибутів коду, ефективні тільки тоді, коли використовуються близькі копії текстів.

Лістинг 1.5. Приклад виявлення плагіату на основі атрибутів коду. Програма 1 (ліворуч): операторів – 2; операцій – 7; операндів – 11. Програма 2 (праворуч): операторів – 2; операцій – 8; операндів – 13.

```
if (i <= j)
   x = z + j; // Коментар 1
   x = z + 3;
else
   x = z - i; // Коментар 2
```

```
if (t <= k)
{ // Змінений Коментар 1
   r = y + k;
   q = 33; // Новий вираз
   r = y + 6; // Новий Коментар 3
} else
{
   r = y - t; // Змінений Коментар 2
}
```

Виявлення плагіату на основі структури коду

Схожість коду визначається на основі його структури. Код програми попередньо обробляється з виділенням токенів та ігноруванням тих властивостей, що легко модифікуються, таких як порожні символи, коментарі і назви ідентифікаторів, що ніяк не впливають на виконання програми (лістинг 1.6). Після токенізації застосовуються алгоритми для знаходження подібних сегментів коду.

Системи на основі структури коду працюють краще за системи на основі атрибутів.

Лістинг 1.6. Приклад токенізації виконаний системою SIM

```
01: #include <stdio.h>
02: int main(void){
03:    for (int i = 0; i < 10; i++) {
04:       printf("Hello World!");
05:    }
06:    return 0;
07: }
```

```
01:
02: I @ ( ){
03: F (I @ = @ @ < @ @ + + ) {
04: @ ( " )
05: }
06: r @
07: }
```

1.2. Обґрунтування ідеї автоматизації виявлення програмного плагіату

Існуючі системи для виявлення плагіату, текстового чи програмного, потребують великої кількості ручної роботи. Відповідальна особа повинна самостійно завантажувати програмні проєкти чи тексти в систему. Якщо

змоделювати навчальний процес певної дисципліни в університеті, де викладач повинен регулярно перевіряти завдання в студентів декількох груп одночасно, то відразу виникає декілька проблем:

1. Складність отримання доступу до робіт студентів:
   – роботи необхідно класифікувати по завданням та групам;
   – якщо студенти копіюють файли в загальну папку – існує ризик перезаписати чужу роботу, при цьому відсутня можливість здати роботу за межами класу;
   – якщо використовувати електронну пошту для здачі робіт – є шанс, що повідомлення потрапить в спам або загубиться серед вхідних повідомлень, також витрачається додатковий час на їх завантаження.
2. Незручність використання систем для виявлення плагіату:
   – необхідно власноруч завантажувати роботи студентів в систему;
   – для різних ситуацій потрібні різні налаштування системи для виявлення плагіату (наявність шаблонного тексту чи файлів, які необхідно виключити з перевірки; потреба у виявлені плагіату лише серед множини завантажених робіт, а не глобально) – користувач повинен постійно змінювати налаштування системи для кожного конкретного випадку.

Існуючі системи для виявлення плагіату не надають функціональності, яка може повністю вирішити дані проблеми. Основна причина для цього – відсутність єдиного рішення для автоматизації завантаження завдань, що потребують перевірки. Загалом принцип роботи більшості систем полягає в збиранні готових завдань у внутрішній базі даних. Відповідно, перевірка на плагіат відбувається серед множини нових завантажених завдань та тих, що були завантажені раніше. Але проблема залишається – користувач такої системи змушений виконувати багато ручної роботи.

1.2.1. *Автоматизація завантаження програмних проєктів*

Якщо розглянути плагіат лише серед навчальних програмних проєктів, то для рішення проблеми автоматизації завантаження завдань можна використати можливості веб-сервісів для хостингу проєктів на базі систем керування версіями (VCS). Дані веб-сервіси популярні серед фахівців IT-сфери (GitHub, Bitbucket), а інтегровані в середовища розробки VCS дозволяють повністю використовувати їх функціональність. При цьому вони надають відкрите API, що дає можливість завантажувати програмні проєкти з цих веб-сервісів автоматично за допомогою програмного забезпечення.

1.2.2. *Автоматизація виявлення програмного забезпечення*

Для виявлення плагіату серед завантажених начальних проєктів передбачається використання спеціалізованої програмної системи. Користувач веб-додатка повинен лише організувати навчальний процес:

1. Створити курс.
2. Додати групи до курсу.
3. Додати студентів до груп з посиланнями на їхні репозиторії на веб-сервісах для хостингу проєктів.

Після цього користувач може створювати завдання для кожної групи чи курсу окремо, налаштовувати дату здачі та особливості перевірки плагіату для конкретного завдання. Веб-додаток, після настання дати здачі, автоматично завантажує програмні проєкти, запускає перевірку на плагіат. Користувачу залишається лише переглянути результати перевірки.

1.3. Аналіз існуючих програмних рішень

Для підтвердження цінності й унікальності розробки веб-додатка в рамках дипломного проєкту було виконано пошук та аналіз існуючих програмних рішень, які частково або повністю вирішують проблему автоматизації виявлення плагіату в навчальних програмних проєктах.

До досліджуваних системи були висунуті такі основні критерії як: можливість організації навчального процесу, автоматизація завантаження рішень та виявлення програмного плагіату.

Пропонується розглянути наступні системи: веб-додаток Google Classroom, веб-додаток Codequiry та десктопний додаток JPlag. Кожне з розглянутих рішень має свої недоліки, але аналіз їх особливостей дозволить покращити програмну розробку дипломного проєкту.

1.3.1. *Google Classroom [3]*

Google Classroom реалізовано як веб-додаток, який можна використовувати в якості системи управління навчанням. Головна особливість даного програмного рішення – інтеграція з сервісами Google, такими як Google Drive для створення і обміну завданнями, Google Docs, Slides і Sheets для виконання робіт, Google Calendar для розкладу та Gmail для спілкування між учасниками навчального процесу.

Основним елементом системи управління навчанням Google Classroom є Курс. Викладач створює Курс, запрошує студентів за допомогою приватного коду або додає їх власноруч, вказавши електронну пошту Gmail. Для кожного курсу створюється окрема папку на Google Drive власника курсу, де зберігаються всі матеріали курсу, завдання, роботи студентів та результати їх оцінювання.

Перелік головних особливостей Google Classroom:

1. Зручне додавання студентів. Для кожного курсу створюється унікальний код. Інші користувачі використовують даний код для приєднання до курсу. Також викладач може запросити на створений курс за допомогою електронної пошти Gmail.
2. Робота з декількома курсами. Викладач може використовувати оголошення, завдання та форми опитувань з інших курсів, відправляти записи відразу в кілька курсів та архівувати курси.

3. Додаткові матеріали. Викладач може додавати до завдань відео з хостинового сервісу YouTube, форми Google, PDF-файли і інші ресурси з Google Drive.
4. Автоматизоване створення копій документа. При створенні завдання у форматі Google-документа, сервіс поширює копії документа для кожного студента курсу окремо.
5. Налаштування завдань. Викладач може призначати для створених завдань категорії оцінок, терміни здачі, бали за відповіді і теми.
6. Відстеження завдань для студентів. Google Classroom створює для кожного курсу Google Calendar, оновлює в ньому завдання і терміни їх виконання.
7. Відстеження завдань для викладачів. Викладач може переглядати роботи учнів, в тому числі завдання, питання, оцінки і попередні коментарі.

З усіх проаналізованих програмних рішень Google Classroom надає найбільш широку функціональність для системи управління навчанням. На відміну від програмної розробки дипломного проєкту та інших аналогів, розглянутих нижче, він представляє собою платформу для двосторонньої взаємодії між викладачем та студентами та позиціонує себе не як помічник в навчальному процесі, а повноцінну систему, яка може замінити заняття в класі.

1.3.2. *Codequiry [4]*

Codequiry – це платний веб-додаток, який позиціонує себе як сучасне рішення для виявлення програмного плагіату. В основі системи лежить алгоритм відбитків документа, що використовується системою Measure Of Software Similarity (див. пункт 2.5.3), але з додатковими покращеннями.

Codequiry дозволяє швидко приступити до роботи, без попередньої підготовки. Користувач має створити директорії, які дають можливість

виконати перевірку плагіату окремо один від одного. Далі необхідно завантажити програмні проєкти в директорію та виконати перевірку.

Веб-додаток пропонує два способи перевірки вихідного коду на:

1. Peer Check – найбільш повний тест для перевірки плагіату серед завантажених в систему проєктів. Порівнює унікальний відбиток однієї кодової бази з іншою, виявляючи логічну схожість і випадки подібності коду серед множини проєктів.
2. Web Check – тест на плагіат, який порівнює код з більш ніж 100 мільйонами джерел з репозиторіїв веб-сервісу GitHub, а також з більш ніж 2 мільярдами примірників вихідного коду в мережі.

Основною перевагою Codequiry над конкурентами є спосіб представлення результатів виявлення плагіату, щоб користувач остаточно визначився чи є програмний проєкт плагіатом.

Інформаційна сторінка

Показує загальні дані результату аналізу програмних проєктів на плагіат та впорядковує їх за відсотком подібності (рис. 1.1). Надає інформацію про склад співпадінь – вказує джерела співпадінь коду та загальну дисперсію. Діаграма дисперсії показує рівень дисперсії результатів на основі Web Check та Peer Check.

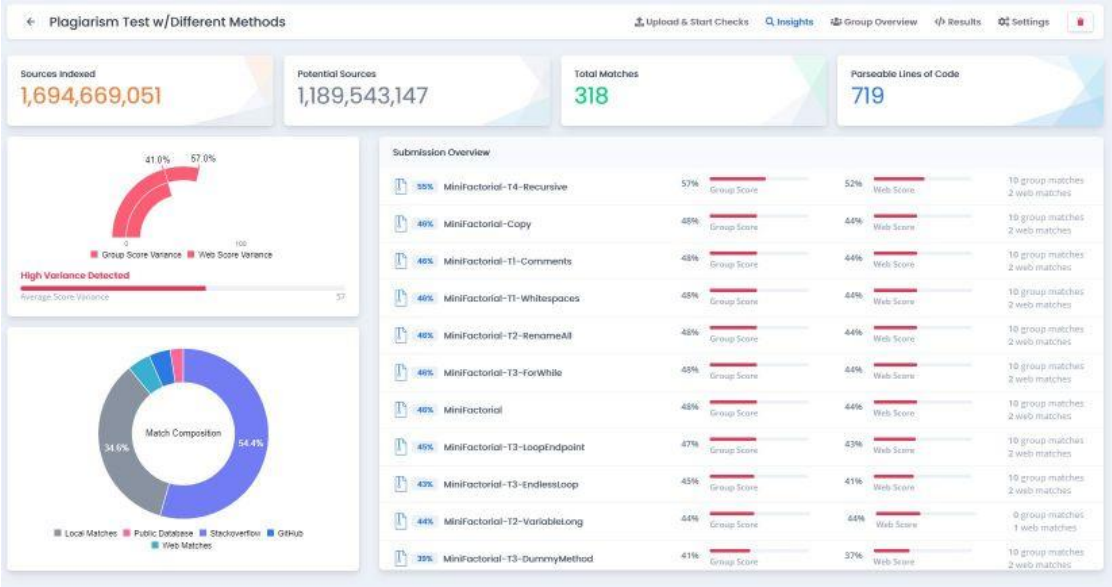


Рис. 1.1. Інформаційна сторінка плагіату веб-додатка Codequiry

Оглядова сторінка

Показує результати Peer Check перевірки між завантаженими користувачем програмними проєктами (рис. 1.2). Більшу частину сторінки займає граф кластера, який представляє візуальне уявлення рівня плагіату між проєктами. Відстань між складовими кластера пропорційна їхній подібності. Гістограма результатів показує всі програмні проєкти із вказанням їх найбільшої оцінки рівня плагіату. Таблиця подібності показує відсоток плагіату між кожним проєктом попарно. Її можна впорядкувати за найбільшим відсотком, щоб побачити потенційний код-плагіат.

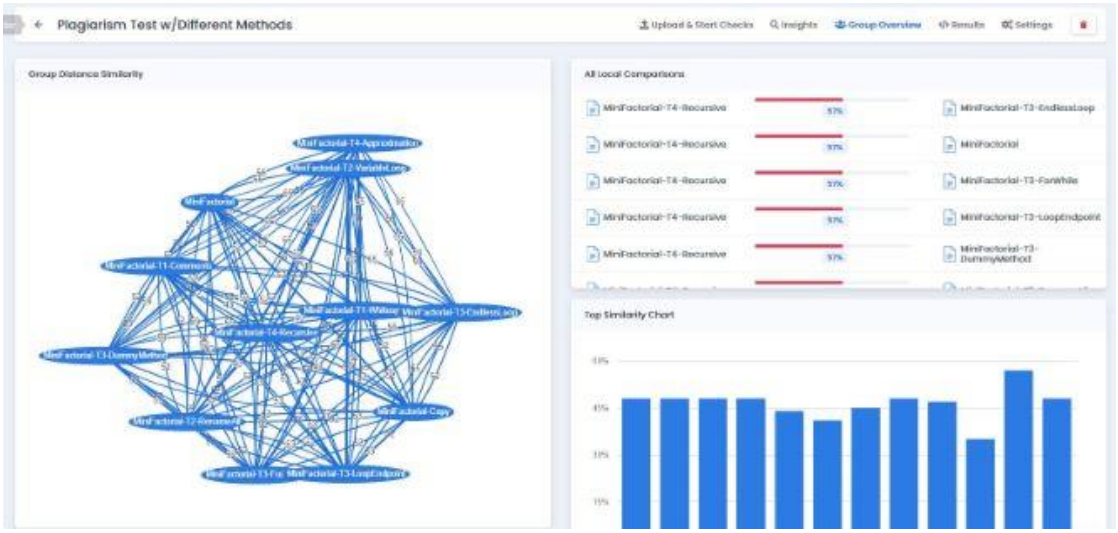


Рис. 1.2. Оглядова сторінка веб-додатка Codequiry

Сторінка результатів

Показує файли вихідного коду з виділеними ділянками, які підозрювані на плагіат (рис. 1.3).

Аналіз збігів описує всі співпадіння, які були знайдені для обраного файлу. Окремо показуються перевірка Peer Check та Web Check. Натискання на ділянку плагіату відкриє файл поруч зі знайденим співпадінням, якщо це Web Check, буде надано URL-адресу веб-сторінки.

Оцінка плагіату показує ймовірністю того, що обраний проєкт є плагіатом. Розбита на три основні розділи: оцінка між завантаженими проєктами, оцінка мережі і середня оцінка.

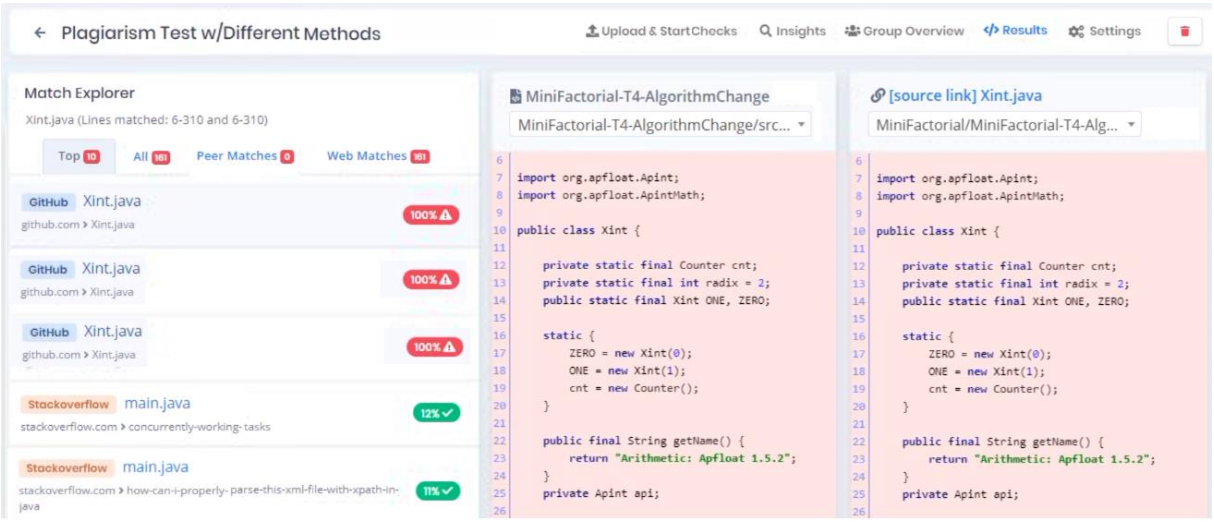


Рис. 1.3. Сторінка результатів плагіату веб-додатка Codequiry

У порівнянні з іншими проаналізованими програмними рішеннями, веб-додаток Codequiry дає можливість найбільш точно дослідити програмні проєкти на плагіат. Результати, згенеровані даною системою, дозволяють викладачу однозначно дати оцінку стосовно рівня плагіату кожного студента окремо.

Головним недоліком Codequiry є необхідність ручного завантаження робіт студентів в систему, а також потреба щомісячно платити значну ціну за використання функціональності веб-додатка.

1.3.3. *JPlag [5]*

JPlag – це додаток, який знаходить подібність між наборами файлів вихідного коду, що дає можливість виявляти програмний плагіат. JPlag не просто порівнює байти тексту, але знає синтаксис мови програмування та структуру програми, тому є стійким до багатьох технік приховування факту плагіату.

JPlag має потужний графічний інтерфейс для представлення своїх результатів. Головна сторінка розбита на 4 секції (рис. 1.4):

1. Зверху вказана загальна інформація по даному аналізу програмних проєктів на плагіат, така як назви програмних проєктів, мова програмування, кількість програмних проєктів,

кількість порівнянь, дата аналізу, мінімальна довжина токена для співпадіння, та розширення файлів, що порівнюються.

2. Секція розподілу показує скільки проєктів мають відповідне значення відсотку плагіату між собою.
3. Співпадіння, відсортовані за середньою схожістю. Середня схожість визначається як середня величина обох програмних покриттів. Це схожість за замовчуванням, яка працює в більшості випадків: збіги з високою середньою схожістю вказують на те, що програми працюють дуже схожим чином.

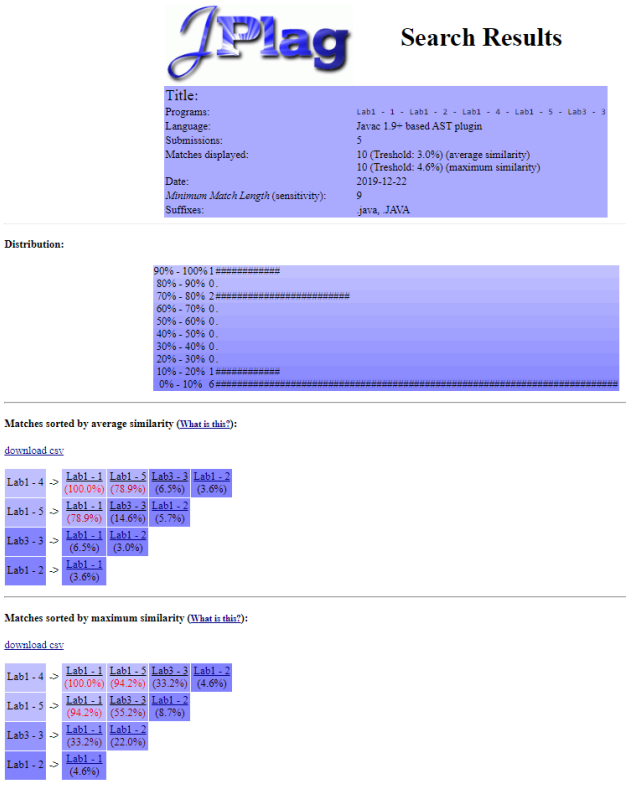


Рис. 1.4. Загальна сторінка результатів JPlag

4. Співпадіння, відсортовані за максимальною схожістю. Максимальна схожість визначається як максимум обох програмних покриттів. Цей рейтинг особливо корисний, якщо програми сильно відрізняються за розміром. Це може статися, коли було додано надлишковий код або використано обфускатори, щоб замаскувати походження плагіату програми.

Користувач може переглянути виявлені ділянки співпадінь між парами проєктів окремо (рис. 1.5). Сторінка розділена на чотири частини:

1. Дві секції внизу містять лістинг коду порівнюваних програмних проєктів, в яких подібні уривки позначені одним кольором.
2. Ліва секція у верхній частині показує відсоток подібності.
3. Права секція містить таблицю всіх уривків, які збігаються. Кожен рядок вказує діапазон номерів рядків з лістингів кожного проєкту окремо, які вважаються однаковими, разом із розміром збігу в лексемах.

Рис. 1.5. Перегляд співпадінь між вихідним кодом двох програм

Також JPlag дозволяє попередньо налаштовувати процес виявлення плагіату. Користувач може самостійно вказати перелік розширень файлів, що будуть порівнюватися; назви файлів, що будуть виключені з перевірки; кількість лексем для аналізу мінімального збігу; мінімальний відсоток збігу між проєктами для генерації звіту; вказати шаблонний код, який не буде рахуватися системою як плагіат.

Роблячи висновки щодо розглянутої системи можна сказати, що JPlag добре справляється з задачею виявлення плагіату в навчальних програмних проєктах. Результати аналізу дозволяють викладачу переглянути вихідний код, підозрюваний на плагіат. А можливість гнучкого налаштування системи – отримати більш точний результат в кожному окремому випадку.

Головним недоліком JPlag є необхідність ручного завантаження робіт студентів в систему. Також на відміну від веб-додатка Codequiry, JPlag не здійснює пошук подібності коду в мережі Інтернет.

1.4. Результати аналізу

Для більш якісного аналізу розглянутих вище програмних рішень пропонується сформувати критерії порівняння на основі вимог до програмного продукту даного дипломного проєкту:

1. *Організація навчального процесу.* Система дозволяє певним чином структурувати робочий простір для користувача.
2. *Створення завдань на реалізацію програмних проєктів.* Система надає можливість створювати завдання.
3. *Автоматизація завантаження робіт студентів.* Система для кожного окремого завдання автоматично завантажує роботи студентів для подальшого їх аналізу.
4. *Можливість виявлення програмного плагіату*. Система дозволяє перевірити на плагіат роботи студентів в рамках кожного завдання окремо.
5. *Налаштування процесу виявлення програмного плагіату.* Для виявлення плагіату система надає користувачу можливість гнучкого налаштування окремих критеріїв, таких як дату та час перевірки на плагіат, виключення шаблонного коду, спосіб формування результатів та інші.

6. *Різні методи виявлення програмного плагіату.* Система може проводити аналіз програмних проєктів на наявність плагіату на основі тільки конкретно вказаних проєктів, з різних джерел коду в мережі Інтернет або поєднувати ці підходи.
7. *Аналітика результатів виявлення програмного плагіату.* Система генерує інформативні графіки, діаграми, таблиці з результатами, дозволяє переглядати підозрюваний на плагіат вихідний код програм двох проєктів одночасно.

Результат порівняння вище описаних програмних рішень за вказаними критеріями наведено у табл. 1.1.

Таблиця 1.1

Порівняльна характеристика існуючих рішень

Критерій	Аналоги		
	Google Classroom	Codequiry	JPlag
Організація навчального процесу	+	+	–
Створення завдань на реалізацію програмних проєктів	+	–	–
Автоматизація завантаження робіт студентів	+	–	–
Можливість виявлення програмного плагіату	–	+	+
Налаштування процесу виявлення програмного плагіату	–	–	+
Різні методи виявлення програмного плагіату	–	+	–
Аналітика результатів виявлення програмного плагіату	–	+	+

З порівняльної таблиці можна зробити висновок, що веб-додаток Google Classroom з реалізованою системою управління навчанням надає користувачем найбільше переваг для організації навчального процесу. Також це єдина з порівнюваних система, яка частково автоматизує процес завантаження рішень та структурує їх (студенти завантажують роботи окремо по кожному завданню). Але головним недолік Google Classroom є відсутність функціональності для виявлення плагіату серед робіт студентів. Викладачу необхідно власноруч викачати роботи студентів та виконати перевірку на плагіат в іншій системі.

Веб-додаток Codequiry пропонує невеликі можливості для організації навчального простору, ніяким чином не вирішує проблему автоматизації завантаження програмних проєктів, але при цьому – представляє собою потужний інструмент для виявлення програмного плагіату. Головна його перевага – аналітика результатів перевірки на плагіат. Також, це єдина система з розглянутих, яка дозволяє порівнювати завантажені користувачем програмні проєкти з джерелами коду в мережі Інтернет.

Десктопний додаток JPlag представляє собою зручний інструмент для виявлення плагіату серед навчальних програмних проєктів. З порівнюваних систем, лише JPlag дозволяє користувачу налаштовувати процес виявлення плагіату для отримання більш точних та інформативних результатів. Натомість він не надає користувачу жодних інструментів, щоб організувати навчальний процес або автоматизувати завантаження робіт студентів.

За результатами аналізу існуючих рішень проблеми автоматизації виявлення плагіату в навчальних програмних проєктів можна відзначити, що жодне з них цілком не відповідає поставленим вимогам до дипломного проєкту. Таким чином, було прийнято рішення про розроблення веб-додатка, який реалізує поставлені вимоги, а також буде враховувати переваги та недоліки кожної з вище описаних систем.

2. АНАЛІЗ МОВ ПРОГРАМУВАННЯ, ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ ВЕБ-ДОДАТКІВ ТА СИСТЕМ ДЛЯ ВИЯВЛЕННЯ ПРОГРАМНОГО ПЛАГІАТУ

2.1. Переваги веб-додатка над іншими типами програмного забезпечення у вирішенні поставленої задачі

Сьогодні програмне забезпечення розробляється для широкого спектру електронних пристроїв, які можуть значно відрізнятися продуктивністю, габаритами, функціональним призначенням та іншими характеристиками. Кожний з них вимагає індивідуального підходу до розробки програмного забезпечення. З найбільш популярних видів додатків можна виділити десктопний, мобільний та веб-додаток. Пропонується окремо розглянути переваги і недоліки кожного типу.

2.1.1. *Десктопний додаток*

Десктопний додаток – це програмне забезпечення, яке встановлюється на комп’ютер (ноутбук або ПК) під конкретну операційну систему, а в деяких випадках – під конкретне апаратне забезпечення.

Переваги десктопних додатків:

1. *Не залежать від мережі Інтернет* (якщо вони не розраховані на взаємодію з іншими користувачами).
2. *Безпека.* Вся інформація користувача зберігається на власному комп’ютері, а не в мережі Інтернет.
3. *Максимальне використання ресурсів комп’ютера для роботи*. Розроблення програми під конкретне апаратне забезпечення дозволяє максимально використовувати потенціал комп'ютера.
4. *Необов’язковість оновлення. Можливість встановлення старіших версій програми.* Десктопні додатки не припиняють роботу з виходом нової версії та не змушують користувача виконати оновлення без попередньої згоди.

Недоліки десктопних додатків:

1. *Відсутність портативності*. Всі дані та налаштування додатка прив'язані до комп'ютера на якому були встановлені.
2. *Необхідне встановлення.* Десктопні додатки вимагають ручного встановлення, а також займають місце на жорсткому диску.
3. *Складність оновлення десктопних додатків, необхідність підтримки старих версій.*
4. *Залежність від апаратного та програмного забезпечення.*

2.1.2. *Мобільний додаток*

Мобільний додаток – програмне забезпечення призначене для роботи на мобільному пристрої. За своєю суттю мобільні додатки аналогічні до десктопних, але розраховані на більш портативні типи пристроїв.

Мобільні додатки мають такі ж переваги і недоліки як і десктопні, а також додаткові, що пов'язані з особливостями мобільних пристроїв.

Переваги мобільних додатків (додатково до десктопних):

1. *Використання можливостей мобільних пристроїв*. Мобільні пристрої оснащені різними інструментами (камери, GPS та NFC модулі, акселерометр, різні сканери та інше), можливості яких доступні для використовувати мобільними додатками.
2. *Push-сповіщення.*
3. *Зручна система встановлення та оновлення.* Мобільні додатки поширюються через спеціальні магазини (Play Market, Apple Store). Вони дозволяють користувачам легко інсталювати та оновлювати мобільні додатки.

Недоліки мобільних додатків (додатково до десктопних):

1. *Обмежені ресурси мобільних пристроїв*. Додатки виконуються на пристроях з значними ресурсними обмеженнями.
2. *Менша роздільна здатність екранів*. Менший розмір екранів мобільних пристроїв ускладнює показ контенту додатків.

2.1.3. *Веб-додаток*

Веб-додаток – це програмне забезпечення, що зберігається на віддаленому сервері та керується через мережу Інтернет за допомогою веб-браузера.

Переваги веб-додатків:

1. *Не потребують встановлення та не займають додаткового місця на жорсткому диску.*
2. *Не потребують оновлення.* Оновлення веб-додатків відбувається на стороні сервера незалежно від користувачів.
3. *Незалежність від платформи.* Оскільки веб-додатки використовуються за допомогою браузерів, то при їх розробці не потрібно підлаштовуватися під конкретну операційну систему.
4. *Вільний доступ з різних пристроїв.* Для доступу до програми потрібен будь-який пристрій з встановленим браузером та доступом до мережі Інтернет.
5. *Пристосовані до збільшення робочого навантаження.* Не потрібно потужне обладнання зі сторони клієнтів, оскільки вся обробка ведеться на стороні серверів. Коли навантаження на систему зростає, розробники просто додають нові сервери.

Недоліки веб-додатків:

1. *Безпека.* Веб-додатки зазвичай стикаються з більшими ризиками для безпеки, ніж десктопні чи мобільні. Користувач ніяким чином не контролює методи захисту конфіденційної інформації.
2. *Залежність від мережі Інтернет та повільніша швидкість роботи.* Неважливо наскільки потужний сервер чи пристрій клієнта. Якщо швидкість мережі не висока, веб-додатки працюватимуть повільніше, ніж десктопні чи мобільні аналоги.
3. *Сумісність з браузером.* При розробці веб-додатків потрібно враховувати їх сумісність з різними браузерами та їх версіями.

2.1.4. *Результати аналізу різних видів додатків*

Проаналізувавши описані вище переваги та недоліки, для розробки дипломного проєкту в якості типу програмного забезпечення було обрано веб-додаток. Головна причина такого вибору полягає в самому призначенні програмного рішення, що розробляється.

Система для виявлення програмного плагіату повинна виступати в ролі допоміжного інструмента для викладача. В її основі лежить взаємодія з веб-сервісами для хостингу проєктів на базі систем керування версіями, а, отже, підключення до мережі Інтернет є обов'язковим для роботи. Це нівелює деякі переваги десктопних та мобільних застосунків над веб-додатками. Оскільки система буде надавати результати аналізу коду на плагіат, необхідний достатній розмір екрану для зручного користування нею. Водночас вона повинна залишатися портативною і не бути прив'язаною до конкретного пристрою, на якому встановлена. Враховуючи вищеописані вимоги стає очевидним вибір веб-додатка.

2.2. Вибір мови програмування та технологій для розроблення серверної частини веб-додатка

Основні задачі серверної частина веб-додатка полягають у обробці запитів від клієнтської частини, виконанні бізнес-логіки всієї програми та роботи з базою даних. Для розробки серверної частини веб-додатків існує багато різних архітектурних підходів на різних мовах програмування. Пропонується проаналізувати та зробити вибір серед таких претендентів як Java, Python та JavaScript.

2.2.1. *Java [6]*

Java одночасно є як мовою програмування, так і платформою. Найпопулярнішою технологією для розробки веб-додатків на Java є Spring projects. Це набір фреймворків для вирішення широкого кола задач, від побудови веб-додатків, до роботи з БД, Big Data, тощо.

Мова програмування Java має наступні характеристики:

1. *Об'єктно-орієнтованість.* Java забезпечує платформу розробки додатків на основі об'єктів (екземплярів класів).
2. *Архітектурна незалежність (крос-платформеність)*. Компілятор генерує об'єктні файли (байт-код), формат яких не залежить від архітектури комп'ютера.
3. *Надійність*. Java забезпечує перевірку часу компіляції і перевірку часу виконання.
4. *Безпечність*. Java дозволяє створювати додатки, в які не можна вторгнутися ззовні.
5. *Висока продуктивність*. Байт-код скомпільованої програми транслюється під час виконання програми в машинний код.
6. *Багатопотоковість*. Java підтримує програми, які виконують операції в різних потоках одночасно, що сприяє високій інтерактивності для кінцевого користувача.
7. *Динамічність.* Java базується на роботі з бібліотеками, що робить мову легко адаптованою до мінливого середовища.

2.2.2. *JavaScript [7]*

Мова інтерфейсу, яка використовується для розроблення веб-сайтів, десктопних додатків та ігор, підтримується всіма браузера. Node.js перетворила JavaScript на мову загального використання. Разом з різними фреймворками вона пропонує можливість зручного написання веб-серверів. Одним з найпопулярніших є Express.js – це простий та швидкий фреймворк Node.js, який використовується в якості обробника для управління серверами.

Особливості JavaScript:

1. *Асинхронна модель виконання*. В JavaScript відсутня багатопотоковість. Натомість JavaScript пропонує асинхронність як аналог.

2. *Швидке серверне рішення.* Вбудована в JavaScript асинхронність дозволяє створювати додатки, які максимізують використання ресурсів сервера (крім процесора).
3. *Одна мова на клієнтській і серверній частині.* Використовуючи Node.js на сервері та фреймворків для створення клієнтської частини веб-додатків дозволяє отримати всі переваги JavaScript.
4. *Гнучкість*. Node.js не має строгих правил до написання програм. Розробники самостійно обирають архітектуру.
5. *Динамічна типізація.*

2.2.3. *Python [8]*

Високорівнева інтерпретована мова програмування загального призначення. Одним з популярних фреймворків для розробки веб-додатків на Python є Django, який пропонує готове рішення багатьох задач (адміністрування, авторизація, ORM, тощо). Flask – інший фреймворк на Python. Головний вибір у випадках розробки невеликих проєктів, для яких не потрібна вся функціональність, що пропонує Django. Дозволяє швидко розгорнути додаток.

Особливості Python:

1. *Синтаксис і семантика.* Python володіє послідовним та чітким синтаксисом, властивостями модульності і масштабованості.
2. *Інтерпретованість.* На відміну від Java не здійснює попередню компіляцію програми, а виконує код послідовно рядок за рядком.
3. *Крос-платформеність.* Інтерпретатор написаний під різні системи і платформи. Це дозволяє виконувати код на різних пристроях. Водночас Python часто напряму використовує системні бібліотеки. Очевидно, що такий код не виконається на платформі з іншою конфігурацією.
4. *Динамічна типізація*.

2.2.4. *Результати аналізу*

Провівши аналіз обраних мов для створення серверної частини веб-додатків було обрано мову програмування Java. Кожна з вищеописаних мов має свої особливості і переваги в написанні веб-додатків. При цьому тільки Java з самого початку позиціонувала себе як мову для написання серверного програмного забезпечення.

Як основна технологія для розробки на мові програмування Java обрано Spring Boot [9] – фреймворк з сімейства Spring projects, що дозволяє максимально пришвидшити розробку, позбавляючи програміста від необхідності власноруч конфігурувати багато процесів, необхідних для роботи сервера. В якості сервера використано Apache Tomcat 9.

2.3. Вибір технологій для розроблення клієнтської частини

Клієнтська частина веб-додатка відповідальна за взаємодію користувача зі всією системою загалом. Саме на клієнтській частині вирішується як саме буде показуватися вміст веб-сайту, буде він статичним чи динамічним. Основними технологіям для розробки клієнтської частини є HTML, CSS та JavaScript.

HTML – HyperText Markup Language – мова розмітки гіпертекстових документів, за допомогою якої розробники веб-додатків будують структуру веб-сторінок. Головними елементами HTML є теги – будівельні блоки сторінок. Вони задають блоки, параграфи, таблиці, форми, зображення, аудіо та відео, тощо.

CSS – Cascading Style Sheets – мова опису зовнішнього вигляду документа, створеного з використанням HTML. Вона дає розробнику можливість модифікувати зовнішній вигляд сторінки, в тому числі, враховуючи роздільну здатність екрану. Якщо HTML відповідає за те, що розмістити і в якому форматі, то CSS контролює який вигляд, дизайн це буде мати.

UI-фреймворки [10] – це набір інструментів, бібліотек із відкритим кодом для розробки за допомогою HTML, CSS та JS. Вони дозволяють значно пришвидшити розробку інтерфейсу веб-сторінок та додати інтерактивних елементів без написання власних скриптів. Прикладами UI-фреймворків є: Bootstrap, Bulma, Foundation, Semantic UI, Materialize.

Головною тенденцією в розробці клієнтської частини веб-додатків є *JavaScript-фреймворки*. Вони дозволяють значно полегшити розробку динамічних веб-сторінок, взаємодію з веб-серверами, збільшити інтерактивність додатка та пришвидшити його роботу зі сторони клієнта. Існує багато JavaScript-фреймворків. Пропонується проаналізувати та зробити вибір серед наступних претендентів [11]: Angular, React, Vue.js.

2.3.1. Angular

Це MVVM JavaScript фреймворк, що відмінно підходить для створення високоінтерактивних веб-додатків.

Переваги Angular:

1. Нові функції, такі як генерація бібліотек npm на основі Angular з CLI, генерація і розроблення веб-компонентів на основі Angular.
2. Одностороння прив'язка даних забезпечує виняткову поведінку додатка та зводить до мінімуму ризик можливих помилок.
3. Впровадження залежностей (DI) від компонентів, пов'язаних з модулями і модульність в цілому.
4. Структура і архітектура створені таким чином, щоб обов’язково підтримувати можливість масштабованості проєкту.

Недоліки *Angular:*

1. Різноманітність структурних елементів. Це значно ускладнює вивчення фреймворку в порівняння з React і Vue.js, які мають тільки один складовий елемент – Component.
2. Відносно низька продуктивність за різними показниками.

2.3.2. React

Це JavaScript-бібліотека, яка добре підходить для створення односторінкових додатків різних розмірів та масштабів.

Переваги React:

1. Легкий в освоєнні завдяки простому дизайну та використання JSX (синтаксису схожого на HTML) для створення шаблонів.
2. Для використання необхідно писати код на сучасному JavaScript і менше вдаватися в особливості даного фреймворку.
3. Дуже швидкий, завдяки реалізації React Virtual DOM.
4. Одностороння прив'язка даних, як і в Angular.
5. Redux – найпопулярніша платформа для управління станом додатків в React, проста в освоєнні і використанні.
6. React реалізує концепції функціонального програмування, дозволяючи створювати багаторазово використовуваний та простий в тестуванні код.
7. Проста міграція між версіями: розробник надає «кодові модулі» для автоматизації більшої частини процесу.

Недоліки React:

1. Змішування логіки з шаблонами (JSX) є нестандартним рішенням і може бути незвичним для розробників.

2.3.3. Vue.js

Це JavaScript-фреймворк для створення веб-інтерфейсів. Легко інтегрується в проєкти з використанням інших JavaScript-бібліотек.

Переваги Vue.js:

1. Розширений HTML. Vue.js має багато подібних характеристик з Angular, це може допомогти оптимізувати обробку HTML-блоків із використанням різних компонентів.

2. Дозволяє швидко перейти від інших фреймворків до Vue.js через схожість з React і Angular з точки зору архітектури та дизайну.
3. Потужні можливості інтеграції. Інтерактивні компоненти можуть бути легко інтегровані в існуючу інфраструктуру без негативного впливу на всю систему.
4. Висока масштабованість. Vue.js дозволяє створювати досить великі шаблони для багаторазового використання.

Недоліки Vue.js:

1. Нестача інформаційних ресурсів. Vue.js має досить невелику частку ринку порівняно з React або Angular, а отже і меншу спільноту, навчальних посібників та прикладів.
2. Ризик перевищення гнучкості. Іноді у Vue.js можуть виникнути проблеми під час інтеграції у великі проєкти.

2.3.4. Результати аналізу технологій для розроблення клієнтської частини веб-додатка

Проаналізувавши все вищесказане можна сформулювати список технологій, що будуть використовуватися для розроблення клієнтської частини веб-додатка.

Обов'язковими до застосування є HTML 5 та CSS – це єдині загальноприйняті технологій для створення та стилізації веб-сторінок.

У якості UI-фреймворку було обрано Bootstrap 4. Він надає широкі можливості для побудови інтерфейсу сторінок та дозволяє легко забезпечити адаптивний дизайн.

Було прийнято рішення розробляти односторінковий додаток. Для цього обрано JavaScript-фреймворк, а саме – React. Він дозволяє швидко почати розробку додатків та не потребує додаткових налаштувань, як Angular із TypeScript.

2.4. Вибір СУБД та технологій взаємодії з обраною мовою програмування

2.4.1. *Різниця між SQL та NoSQL базами даних*

Бази даних SQL використовують структуровану мову запитів для визначення та обробки даних. Це забезпечує високу продуктивність. Але вимагає попереднього проєктування схеми бази даних, оскільки модифікація вже існуючої має вплив на всі збережені дані.

Бази даних NoSQL мають динамічні схеми для неструктурованих даних, які, до того ж, можуть зберігатися різними способами: документні, ключ-значення, графові та стовпчиком. Така гнучкість дозволяє створювати документи, попередньо не визначаючи їх структуру та додавати поля до них за необхідності.

2.4.2. *MySQL*

Переваги та сильні сторони MySQL:

1. *Зрілість*. MySQL – надзвичайно поширена база даних, а отже вона стабільна та добре протестована на реальних проєктах.
2. *Повна підтримка ACID властивостей.*
3. *Підтримка реплікації.* Базу даних MySQL можна реплікувати на декілька вузлів. Це дозволяє збільшити масштабованість та доступність бази даних і забезпечити її відмовостійкість.
4. *Підтримка шардування.* Хоча це неможливо забезпечити на більшості баз даних SQL, це можна зробити на серверах MySQL.

2.4.3. *PostgreSQL*

PostgreSQL – це гібридна система баз даних SQL/NoSQL, яка пропонує такі переваги:

1. *Великі можливості управління транзакціями.*

2. *Об'єктно-орієнтована система управління базами даних (ORDBMS).* PostgreSQL є поєднанням строгої реляційної моделі (SQL) і об'єктно-орієнтованою моделі (NoSQL).
3. *Повна підтримка ACID властивостей.*
4. *Чистий SQL*. PostgreSQL використовує одну з найчистіших форм SQL, на відміну від інших баз даних, які часто мають унікальні особливості.

2.4.4. *MongoDB*

Нижче наведено деякі переваги та сильні сторони MongoDB:

1. *Динамічна схема.* Дає вам можливість змінювати схему даних без зміни будь-яких вже існуючих даних в базі.
2. *Керованість*. База даних не потребує адміністратора.
3. *Висока ефективність для простих запитів.*
4. *Підтримка реплікації та шардування.* MongoDB не потребує додаткового налаштування для масштабування бази даних.

2.4.5. *Результати аналізу*

Проаналізувавши різні види баз даних та їх представників, для розробки дипломного проєкту було обрано SQL базу даних, а саме PostgreSQL. Такий вибір пояснюється тим, що остаточна схема бази даних буде спроєктована ще на етапі розробки, а зберігатися будуть заздалегідь описані структури даних. Також база даних повинна забезпечувати транзакційність операцій. PostgreSQL базується на тому, що ця база даних пропонує таку ж функціональність що і MySQL і навіть більше.

Що стосується технологій, які будуть використовуватися з обраною мовою програмування, то було обрано Spring Data JPA – одна з технологій фреймворку Spring Data, яка реалізує стандарт JPA та використовує ORM фреймворк Hibernate.

2.5. Вибір системи для виявлення плагіату в навчальних програмних проєктах

Для виявлення плагіату в навчальних програмних проєктах передбачається використання готового рішення для аналізу вихідних кодів програм. Створено окремий модуль, ціллю якого є взаємодія веб-додатка з даною системою. Цей додатковий рівень абстракції мінімізує залежність програмного рішення від конкретної системи виявлення програмного плагіату, що дозволить за необхідності замінити використовувану систему на будь-яку іншу або створити власну.

Вибір буде здійснюватися серед наступних рішень: SIM, JPlag, MOSS.

2.5.1. *SIM [12]*

Для виявлення плагіату SIM спочатку токенізує вихідний код, потім створює загальну таблиця символів (для зберігання динамічно назначених токенів ідентифікаторів) та порівнює токенізовані рядки програм за допомогою *алгоритму локального вирівнювання рядків [13]* (рис. 2.1). SIM підтримує код C, C++, Java, Pascal та текст на природній мові.

Вирівнювання двох рядків відбувається за допомогою вставки в них пробілів таким чином, щоб їх довжини стали однаковими. Нехай A^* і B^* – нові рядки, отримані після вирівнювання вхідних рядків *a* і *b*, відповідно. Розглянемо A_i^* і B_i^*: вартість їх збігу – *s*; якщо тільки один із символів пропуск, то – *t;* вартість невідповідності – *d;* де *s* – додатне, *t* і *d* – невід'ємні. Ціна вирівнювання – це сума індивідуальних вартостей всіх пар A_i^* і B_i^*, найбільше значення цієї цільової функції для всіх *i* та *j*, таких що $i \le j \le \left|A_i^*\right| = \left|B_i^*\right|$ на рядках $A_i^*\left[i..j\right]$ і $B_i^*\left[i..j\right]$ – величина вирівнювання.

Оптимальне вирівнювання між двома рядками – максимальне значення цільової функції серед усіх комбінацій вирівнювання. Це значення може бути обчислено за допомогою динамічного програмування.

Застосування алгоритму в системі SIM виглядає наступним чином:

1. Отримуються токенізовані рядки *A* і *B* двох програм.
2. *B* ділиться на секції, кожна з яких представляє модуль програми.
3. Для кожної секції і рядка *A* отримуються значення оптимального локального вирівнювання.
4. Результати комбінуються.

Такий підхід дозволяє системі коректно обробляти перестановки модулів вихідної програми.

Рис. 2.1. Приклад локального вирівнювання токенізованого представлення коду програм

2.5.2. *JPlag [5]*

JPlag – система виявлення плагіату з потужним графічним інтерфейсом для представлення результатів. Підтримує код Java, C#-1.2, C, C++, Python 3, Scheme та текст на природній мові.

Дана система розглянута як існуюче програмне рішення для проблеми автоматизації виявлення плагіату (див. підрозділи 1.3-1.4). Зроблено висновок, що вона добре справляється з задачею виявлення плагіату, але жодним чином не дозволяє автоматизувати даний процес. Натомість представлення JPlag в якості десктопного додатка та відкритий код від розробників дає можливість використати його при розробці модуля

для взаємодії веб-додатка з системою виявлення плагіату. Тому буде цілком логічно розглянути дане рішення як претендента для вибору.

JPlag перетворює програми в рядки токенів, які представляють структуру програми. Наступний етап – порівняння двох рядків токенів за допомогою алгоритму *жадібного рядкового замощення (The Greedy String Tiling [13, 14])*, з додатковими оптимізаціями.

Алгоритм використовує дві евристики:

1. Більш довгі послідовні збіги при порівнянні рядків токенів – це краще, ніж набір менших і непослідовних збігів.
2. Алгоритм ігнорує збіги, які менші порогового значення.

Друга евристика сприяє фільтрації шумів, тобто невеликих ділянок коду, які випадково збігаються в перевірених вихідних кодах.

Результатом застосування жадібного алгоритму для рядків токенів буде набір їх спільних підрядків, що не перетинаються. Підрядок, що входить у цей набір називається тайлом. Чим більша нормалізована сума довжин отриманих тайлів, тим більше схожість даних підрядків. Функцію подібності можна представити у наступному вигляді:

$$sim(A,B) = \frac{2 \cdot |tiles|}{|A| + |B|}, \qquad (2.1)$$

де $|tiles|$ – потужність (кількість символів) знайденого набору тайлів; $|A|$, $|B|$ – відповідні потужності рядків, що порівнюються.

Псевдокод алгоритму представлений на рис 2.3.

У структурі алгоритму можна виділити 4 цикли:

1. Зовнішній цикл. Його результатом є додавання у набір чергового тайлу. Тайли додаються у порядку зменшення.
2. Внутрішній цикл по всім символам першого рядку.
3. Внутрішній цикл по всім символам другого рядку.
4. Цикл по співпадаючим символам, починаючи від першого співпадіння поточних символів даних рядків.

```
0   Greedy-String-Tiling(String A, String B) {
1       tiles = {};
2       do {
3           maxmatch = MinimumMatchLength;
4           matches = {};
5           Forall unmarked tokens A_a in A {
6               Forall unmarked tokens B_b in B {
7                   j = 0;
8                   while (A_{a+j} == B_{b+j} &&
9                           unmarked(A_{a+j}) && unmarked(B_{b+j}))
10                      j++;
11                  if (j == maxmatch)
12                      matches = matches ⊕ match(a, b, j);
13                  else if (j > maxmatch) {
14                      matches = {match(a, b, j)};
15                      maxmatch = j;
16                  }
17              }
18          }
19          Forall match(a, b, maxmatch) ∈ matches {
20              For j = 0 ... (maxmatch − 1) {
21                  mark(A_{a+j});
22                  mark(B_{b+j});
23              }
24              tiles = tiles ∪ match(a, b, maxmatch);
25          }
26      } while (maxmatch > MinimumMatchLength);
27      return tiles;
28  }
```

Рис. 2.3. Псевдокод алгоритму жадібного рядкового замощення

Оптимізації алгоритму жадібного рядкового замощення

Використовується ідея алгоритму Рабіна-Карпа – порівняння підрядків за їх хеш-значеннями:

1. Хеш-значення обчислюються для всіх підрядків довжини *MinimumMatchLength* в *A* і *B*.
2. Кожне значення хеша з рядка *A* потім порівнюється з кожним з *B*. Якщо два значення однакові, то знайдено можливий збіг двох підрядків, що починаються з цього токена.
3. Використовується хеш-таблиця для розміщення підрядків з *B*, які мають те саме значення хешу, що й заданий підрядок з *A*. Це зменшує обчислення до лінійної кількості кроків.

2.5.3. *MOSS [15]*

Measure Of Software Similarity – веб-сервіс для виявлення плагіату коду, використовує *алгоритм відбитків документа, а саме – алгоритм просіювання (Winnowing [16])*. Підтримує до 23 мов програмування.

У алгоритмі відбитків токенізована програма представляється у вигляді набору міток так, щоб ці набори для схожих програм перетиналися. Метод відбитків можна представити у наступному вигляді:

1. Послідовно хешуються підрядки токенізованої програми *P* довжини *k* (фіксований параметр).
2. Виділяється деяка підмножина їх хеш-значень, що добре характеризує *P*. Ті ж кроки повторюються для програм $T_1, T_2 ... T_n$, їх обрані хеш-значення зберігаються в таблицю.
3. За допомогою таблиці (бази) отримується набір ділянок рядку *P*, підозрілих на плагіат.
4. Аналізуються отримані на попередньому кроці дані.

Змістовна частина алгоритму – це вибір значення *k* та другий крок. Число *k* обмежує довжину найменшого підрядка, з якої може працювати даний алгоритм. Невелике його значення сприяє ігноруванню «шумів», в іншому випадку алгоритм може пропустити випадок плагіату.

Для виділення підмножини серед хеш-значень токенізованої програми система MOSS використовує *алгоритм просіювання***.**

Хід алгоритму просіювання:

1. Обирається значення *k* і $k \le t$.
2. Вхідний рядок розбивається на підрядки довжиною *k*, так звані *k*-грами. Розраховуються хеш-значення підрядків: $h_1, h_2 ... h_n$.
3. Вздовж послідовності $h_1, h_2 ... h_n$ просувається вікно розміром: $w = t - k + 1$.
4. На кожному кроці вікно рухається на одну позицію вправо. Міткою призначається мінімальне h_j у вікні. Якщо мінімальних елементів декілька, то призначається крайній правий.

Якщо в послідовних вікнах хеш-значення від одного і того ж підрядка є мінімальними, то міткою призначається тільки одна з них (немає сенсу зберігати послідовні абсолютно однакові мітки).

2.5.4. *Порівняння систем для виявлення програмного плагіату*

Для вибору системи пропонується виконати порівняння за наступними критеріями (табл. 2.3):

1. *Підтримка різних мов програмування (кількість)*.
2. *Основний алгоритм за яким система виявляє плагіат.*
3. *Виключення шаблонного коду* – чи дозволяє система обрати частини програми, які необхідно виключити з перевірки.
4. *Виключення невеликих файлів* – чи дозволяє система виключити з перевірки файли вихідного коду до заданого розміру.
5. *Історичні порівняння* – чи дозволяє система порівнювати новий набір програмного коду із попередніми.
6. *Загальний рейтинг або на основі файлів* – чи система оцінює плагіат за кожним файлом або за програмним кодом в цілому.
7. *Відкритий код* – чи вихідний код системи (або її алгоритмів) доступний для вільного користування.

На основі порівняння систем за вказаними критеріями можна зробити висновок, що системи JPlag та MOSS показали себе як найбільш ефективні. Вони надають користувачу схожу функціональність, а алгоритми, які лежать в їх основі забезпечують приблизно однакові результати при виявленні плагіату.

При цьому з розглянутих систем лише MOSS з її методом просіювання дозволяє організувати базу програмних кодів, відносно якої можна шукати плагіат в будь-якому іншому вихідному коді за прийнятний час, а не перевіряти на плагіат зразок з кожним елементом бази. Це можливо завдяки тому, що метод використовує хеш-таблицю.

Натомість, у свою чергу, JPlag представлений як десктопний додаток та має відкритий код. Це дає можливість модифікувати його під власні потреби та вільно використовувати у власних цілях на сервері без необхідності бути залежним від сторонніх веб-сервісів, як у випадку з MOSS.

Система SIM використовує алгоритм, що оснований на вирівнюванні рядків, тому на невеликих програмах він може давати помилкові дані.

В результаті, для виявлення плагіату в навчальних програмних проєктах було обрано JPlag, як найбільш компромісне рішення серед розглянутих аналогів. Відкритий код системи дозволяє легко інтегрувати її в проєкт, а наданої функціональності буде досить для проведення аналізу та формування результатів виявлення плагіату.

Таблиця 2.3

Порівняння систем для виявлення плагіату

	JPlag	MOSS	SIM
Підтримка різних мов програмування	7	23	5
Основний алгоритм	Токенізація, жадібне рядкове замощення, оптимізації	Токенізація, техніка просіювання (варіант алгоритму відбитків)	Токенізація, локальне вирівнювання рядків
Виключення шаблонного коду	+	–	–
Виключення невеликих файлів	+	+	–
Історичні порівняння	–	–	+
Загальний рейтинг або на основі файлів	Загальний	Загальний	Файл
Відкритий код	+	–	+

3. СТРУКТУРНА ОРГАНІЗАЦІЯ РОЗРОБЛЕНИХ ПРОГРАМНИХ ЗАСОБІВ

3.1. Загальна структура програмного забезпечення

Програмне забезпечення дипломного проєкту реалізоване у вигляді веб-додатка з використанням архітектурного шаблону MVC. Програмну систему можна розділити на дві частини:

1. Серверна частина, яка містить в собі всю бізнес-логіку, відповідальна за збереження та обробку даних.
2. Клієнтська частина, що представляє зовнішній вигляд веб-додатка та дає користувачу можливість повноцінно користуватися розроблюваним програмним забезпеченням.

Основні елементи даних частин та зв’язки між ними зображено на структурній схемі веб-додатка (див. Додаток 1. Структурна схема веб-додатка). Клієнтська частина взаємодіє із серверною через інтерфейс RESTful API. Такий підхід разом з обраними технологіями дозволяє розробляти елементи програмної системи незалежно один від одного. Клієнтській частині не потрібно знати особливості реалізації серверної частини, а серверній – клієнтської. Для цього попередньо формується API – адреса (URL), метод та тіло запиту, яке приймає або повертає сервер у певному форматі. Знаючи даний API клієнтська частина може розроблятися незалежно, навіть за умови, що сервер ще не було реалізовано та розгорнуто в мережі.

Пропонується окремо більш детально розглянути структуру серверної та клієнтської частини веб-додатка.

3.1.1. *Структура серверної частини веб-додатка*

Функції серверної частини веб-додатка:

1. Реалізує процедуру реєстрації та авторизації.
2. Отримує, обробляє запити від клієнта та повертає відповідь.

3. Взаємодіє з базою даних, виконуючи операції створення, зчитування, оновлення та видалення (CRUD) для сутностей системи (спроєктовані сутності описано в підрозділ 3.2).
4. Реалізує та використовує модуль взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS.
5. Реалізує та використовує модуль взаємодії із системою для виявлення плагіату в програмних проєктах.
6. Виконує роль планувальника задач для автоматичної перевірки навчальних програмних проєктів на плагіат.

Як було описано вище веб-додаток реалізовано у вигляді MVC шаблону. Серверна частина у цій зв’язці виконує роль Моделі (M) та Контроллера (C). Вона реалізує всю логіку роботи системи, створюючи її Модель, та надає інтерфейс для взаємодії з нею – Контроллер. Для виконання поставлених задач сервер має власну архітектуру, в даному разі було обрано багаторівневу [17], де зв’язки між компонентами залежать від рівня на якому вони розміщуються. Компоненти організовані в горизонтальні шари, кожен рівень виконує конкретну роль та несе певну відповідальність у застосуванні. Рівні архітектури веб сервера (рис. 3.1):

1. *Рівень подання (presentation layer)*. Розміщуються фільтри, контроллери. Відповідальний за обробку запитів від клієнта та перевірку вхідних даних. Описує API веб-сервера.

   Серверна частина реалізована за допомогою фреймворку Spring Boot**.** Він реалізує шаблон Front Controller [18], що об'єднує обробку всіх запитів шляхом їх направлення через один об'єкт – Dispatcher Servlet, який, використовуючи екземпляр класу HandlerMapping, делегує запити на відповідні контроллери. Такий підхід дозволяє позбутися дублювання коду, об’єднуючи загальну поведінку програми (авторизація, інтернаціоналізація) в один об’єкт, а також дозволяє змінювати цю поведінку динамічно під час виконання.

2. *Рівень фасадів (facade layer)*. Розміщуються класи фасадів та конвертори. Головне призначення – конвертація об’єктів DTO з запитів в об’єкти моделей та навпаки. Моделі використовуються на рівні бізнес-логіки та зберігаються в базі даних. DTO необхідні для передачі даних між клієнтською і серверною частиною. Структура цих об’єктів не обов’язково повинна бути однаковою.
3. *Рівень бізнес-логіки (service layer).* Розміщуються сервіси, які містять всю логіку роботи веб-додатка. Окремо варто відмітити спеціальний сервіс – Scheduler. В ньому реалізовані задачі, які повинні регулярно за певним таймером виконуватися в системі. Головна його ціль – аналіз бази даних на наявність завдань для перевірки програмних проєктів на плагіат. Якщо такі є, то сервіс створює нові потоки для виконання перевірки.
4. *Рівень зберігання та доступу до даних (persistence layer)*. Розміщуються окремі модулі системи, які взаємодіють з різними зовнішніми сервісами для отримання, зберігання та обробки даних.

   DAO – інтерфейс для доступу до бази даних. Приховує від користувача деталі реалізації, особливості бази даних чи технологій для взаємодії з нею.

   File system – об’єкти, що дають можливість взаємодіяти з файловою системою сервера, а саме створювати чи видаляти файли, директорії, записувати в них дані, зберігати та розпаковувати архіви у форматі .zip.

   VCS – модуль взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS (див. підрозділ 3.3).

   Plagiarism – модуль взаємодії із системою для виявлення плагіату в програмних проєктах (див. підрозділ 3.4).

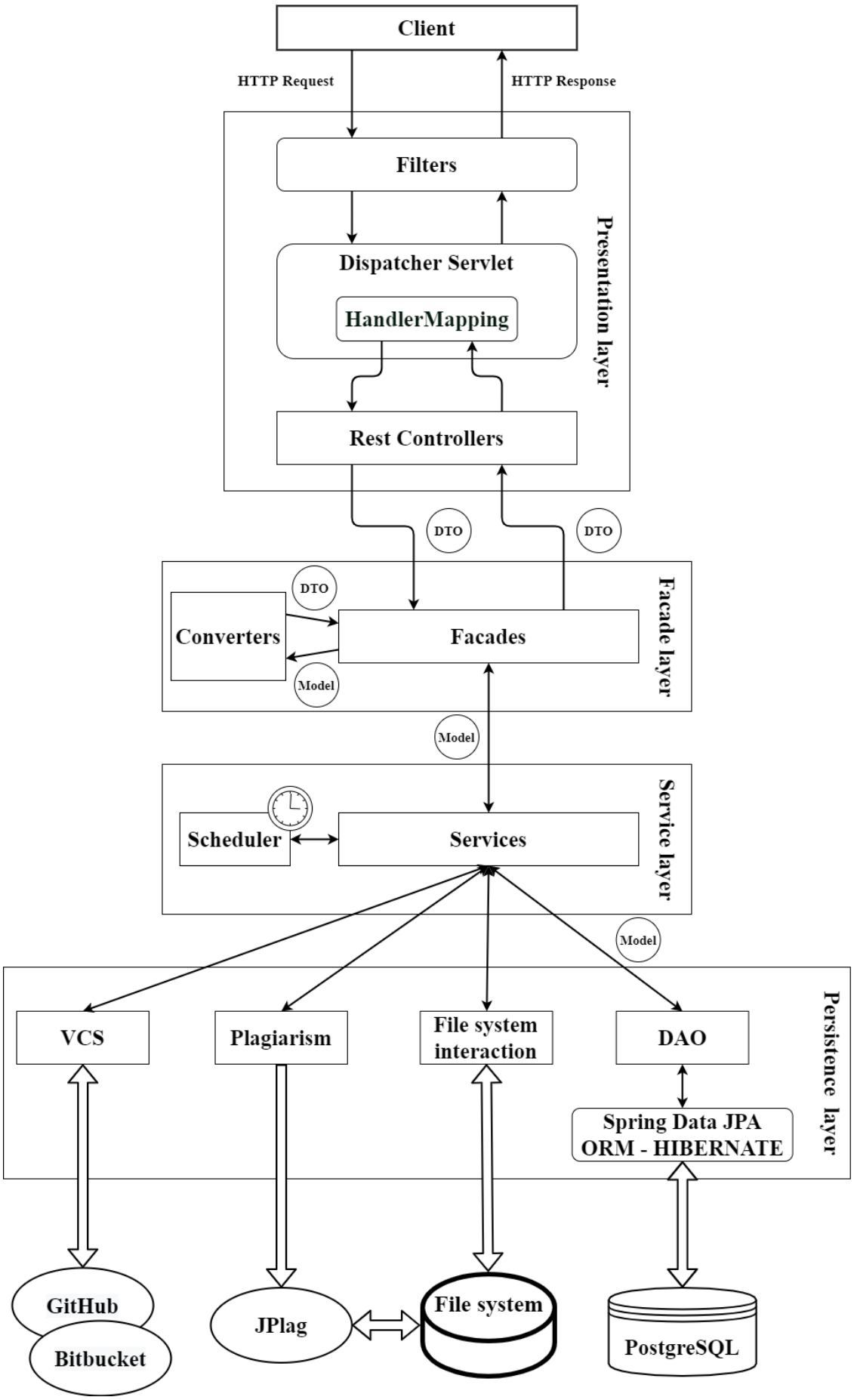


Рис. 3.1. Структура серверної частини веб-додатка

3.1.2. *Структура клієнтська частини веб-додатка*

Функції клієнтської частини веб-додатка:

1. Реалізує інтерфейс для взаємодії системи з користувачем.
2. Виконує запити до сервера для оновлення даних на веб-сторінці.
3. Забезпечує коректність відтворення дизайну інтерфейсу веб-додатка на пристроях із різною роздільною здатністю.

Клієнтська частина веб-додатка виконує роль Представлення (V) у шаблоні MVC. Використання React як технології для розробки клієнтської частини дозволяє розробити односторінковий додаток (SPA [19]) – використовується єдиний HTML-документ як основа для всіх веб-сторінок, а взаємодія з користувачем організовується через динамічне завантаження HTML, CSS, JavaScript. Такий підхід надає користувачу можливість взаємодіяти з веб-додатком без перезавантаження сторінок.

Архітектура клієнтської частини містить наступні елементи (рис. 3.2):

1. Actions – це об'єкти (Action) створені для зберігання інформації пов'язаної з дією користувача (натискання кнопки на інтерфейсі, введення або видалення тексту). Включають таку інформацію як: вид дії, місце виникнення, час, що і в який стан змінюється.
2. Middlewares – надають спосіб взаємодії з об'єктами Action, що були відправлені до сховища (Store). Перед тим, як вони відправляються до Reducer у Middlewares виконуються такі процеси як запис журналів логів, оповіщення про помилки, створення асинхронних запитів до сервера, або відправка нових об’єктів Action.
3. Dispatcher – єдина точка в додатку (клієнтської частини) для отримання об'єктів Action, що потребують обробки.
4. State – стан додатка, набір об’єктів JavaScript з інформацією, яка визначає що і як показувати користувачу.
5. Store – зберігає та керує станом додатка (State).

6. Reducers – функція (або кілька функцій), яка отримує об’єкт Action і залежно від логіки змінює стан Store.
7. View – компоненти React, які містять розмітку веб-сайту та логіку її формування залежно від State.

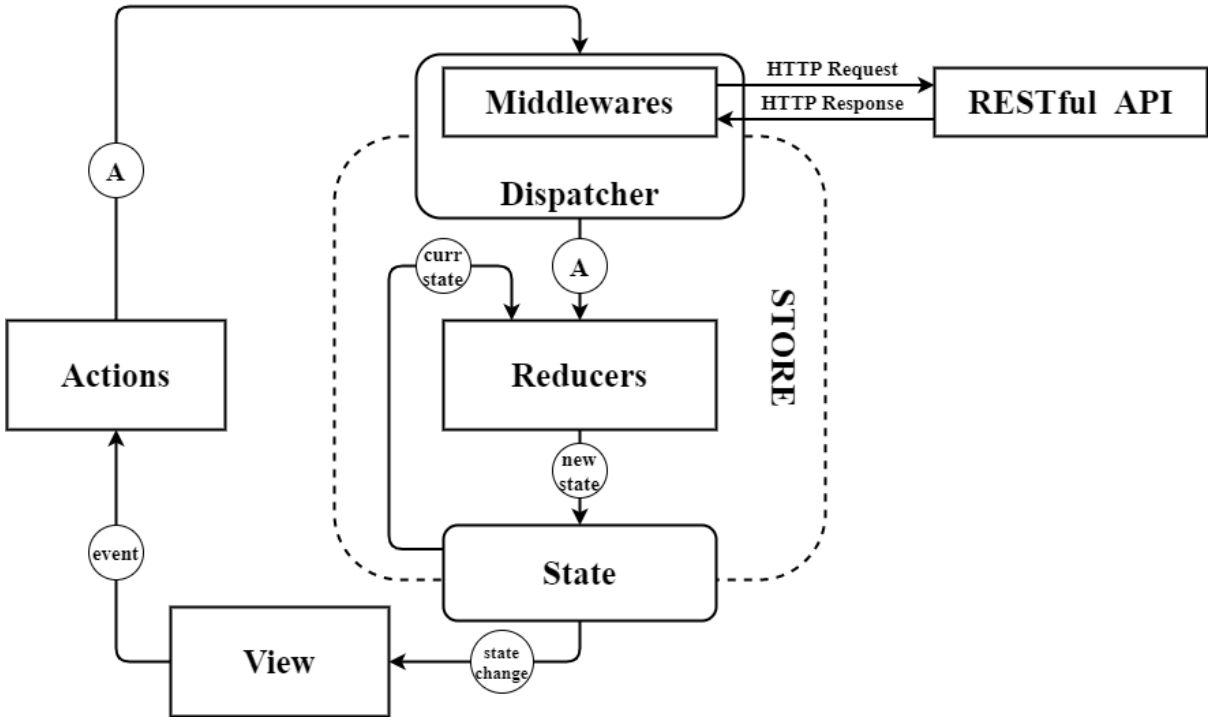


Рис. 3.2. Структура клієнтської частини веб-додатка

Уся розмітка і логіка формування клієнтської частини веб-додатка розміщується в компонентах React. Для створення цих компонентів використовується шаблон Container/Component. Він пропонує розділити відповідальність за виконання різних завдань між окремими компонентами. Це дозволяє поліпшити структуру програми та полегшує процес розробки. Шаблон містить наступні елементи:

1. Container – React компонент відповідальний за зберігання стану і описує методи для управління станом.
2. Component – React компонент, що містить логіку формування інтерфейсу. Відповідає за отримання від компонента Container властивостей і за правильне формування інтерфейсу.

3.2. Структура бази даних

Для розробки веб-додатка з урахуванням поставлених вимог до організації навчального процесу і перевірки програмних проєктів на плагіат було спроєктовано схему реляційної бази даних. Таблиці бази даних, зв'язки між ними, зовнішні і первинні ключі та типи полів зображено у вигляді ER-діаграми (див. Додаток 1. Структура бази даних). Усі таблиці нормалізовані у першу, другу та третю нормальні форми [20].

Пропонується окремо детальніше розглянути сутності спроєктованої ER-діаграми з описом їх призначення та атрибутів.

Користувач (табл. users) – профіль зареєстрованого користувача:

1. Унікальний ідентифікатор користувача (user_id) – первинний ключ.
2. Унікальний ідентифікатор ролі (role_id) – зовнішній ключ.
3. Ім'я користувача (first_name).
4. Прізвище користувача (last_name).
5. Електронна пошта (email) – унікальне значення, використовується для авторизації користувача.
6. Пароль (password) – зберігається в зашифрованому вигляді, використовується для авторизації користувача.

Роль (табл. roles) – роль користувача в системі. Необхідна для розмежування доступу до функціональності веб-додатка:

1. Унікальний ідентифікатор ролі (role_id) – первинний ключ.
2. Назва ролі (name).

Токен авторизації (табл. access_tokens) – токен авторизації, що дає можливість веб-додатку переглядати та завантажувати репозиторії з веб-сервісів для розміщення програмних проєктів на базі VCS:

1. Унікальний ідентифікатор токена авторизації (token_id) – первинний ключ.
2. Унікальний ідентифікатор користувача (user_id) – зовнішній ключ.

3. Токен авторизації (access_token_string).
4. Тип токена авторизації (token_type).
5. Права доступу токена авторизації (scope).
6. Токен оновлення (refresh_token).
7. Веб-сервіс, що надає токен (authorization_provider).

Курс (табл. courses):

1. Унікальний ідентифікатор курсу (course_id) – первинний ключ.
2. Унікальний ідентифікатор користувача (creator_id) – зовнішній ключ.
3. Назва курсу (name).
4. Опис курсу (description).
5. Дата створення курсу (creation_date).

Група (табл. groups):

1. Унікальний ідентифікатор групи (group_id) – первинний ключ.
2. Унікальний ідентифікатор курсу (course_id) – зовнішній ключ.
3. Назва групи (name).
4. Дата створення групи (creation_date).

Студент (табл. students):

1. Унікальний ідентифікатор студента (student_id) – первинний ключ.
2. Унікальний ідентифікатор користувача (creator_id) – зовнішній ключ.
3. Повне ім’я студента (full_name) – унікальне значення.

Студент-Групи (табл. student_group) – асоціативна сутність для зв’язку «багато до багатьох» між сутностями Студент та Група. Комбінація зовнішніх ключів course_id та student_id унікальна, щоб не дозволити додавати студента до різних груп в рамках одного курсу. Ідентифікатор (первинний ключ) представляється унікальною комбінацію зовнішніх ключів group_id та student_id:

1. Унікальний ідентифікатор групи (group_id) – зовнішній ключ.

2. Унікальний ідентифікатор студента (student_id) – зовнішній ключ.
3. Унікальний ідентифікатор курсу (course_id) – зовнішній ключ.
4. Посилання на репозиторій студента (vcs_repository_link).

Завдання (табл. tasks):

1. Унікальний ідентифікатор завдання (task_id) – первинний ключ.
2. Унікальний ідентифікатор курсу (course_id) – зовнішній ключ
3. Назва завдання (name).
4. Префікс шляху (repository_prefix_path) – шлях директорії в репозиторію студента, яка буде завантажена для перевірки на плагіат.
5. Опис завдання (description).

Завдання-Групи (табл. task_group) – асоціативна сутність для зв'язку «багато до багатьох» між сутностями Завдання та Група. Містить додаткову інформацію. Ідентифікатор (первинний ключ) представляється унікальною комбінацію зовнішніх ключів:

1. Унікальний ідентифікатор групи (group_id) – зовнішній ключ.
2. Унікальний ідентифікатор завдання (task_id) – зовнішній ключ.
3. Унікальний ідентифікатор результату виявлення плагіату (plagiarism_detection_result_id) – зовнішній ключ.
4. Унікальний ідентифікатор налаштування виявлення плагіату (plagiarism_detection_setting_id) – зовнішній ключ.
5. Статус виконання виявлення плагіату (plagiarism_ detection_status) – можливі 4 стана: в очікуванні – PENDING, в процесі виконання – IN_PROCESS, успішно виконано – DONE, не далося виконати – FAILED.
6. Дата створення задачі для групи (creation_date).
7. Дата закінчення виконання завдання (expiry_date) – по настанню даної дати веб-додаток завантажує репозиторії студентів та виконує їх перевірку на плагіат.

Налаштування системи JPlag (табл. plagiarism_detection_settings):

1. Унікальний ідентифікатор налаштування виявлення плагіату (plagiarism_detection_setting_id) – первинний ключ.
2. Унікальний ідентифікатор мови програмування (programming_ language_id) – зовнішній ключ.
3. Чутливість порівняння (comparison_sensitivity).
4. Мінімальнтй відсоток подібності (minimum_similarity_percent).
5. Шлях до директорії з шаблонним кодом (base_code_path).
6. Тип перевірки на плагіат (detection_type) – має 2 варіанти: серед студентів групи (GROUP) та серед всіх студентів курсу (COURSE).
7. Шлях на файлової системи до репозиторіїв (data_path).
8. Шлях на файловій системі для збереження результату (save_result_path).

Мова програмування (табл. programming_languages):

1. Унікальний ідентифікатор мови програмування (programming_ language_id) – первинний ключ.
2. Назва мови програмування (name).
3. Стандартна чутливість порівняння для обраної мови, що використовується JPlag (default_comparison_sensitivity).
4. Перелік розширень файлів, підтримуваних системою JPlag для обраної мови (file_types_support).

Результати виявлення плагіату (табл. plagiarism_detection_results):

1. Унікальний ідентифікатор результату виявлення плагіату (plagiarism_detection_result_id) – первинний ключ.
2. Опис загального результату (result_message).
3. Журнал виконання системи JPlag (log).
4. Процес закінчився успішно чи ні (is_successful).
5. URL шлях для отримання результатів (result_path).
6. Дата виконання перевірки на плагіат (date).

Результат-Студента (табл. result_student) – асоціативна сутність для забезпечення зв'язку «багато до багатьох» між сутностями Результати виявлення плагіату та Студент. Містить додаткову інформацію. Створюється лише у випадку, коли веб-додаток не зміг завантажити репозиторій студента. Ідентифікатор (первинний ключ) представляється унікальною комбінацію зовнішніх ключів:

1. Унікальний ідентифікатор студента (student_id) – зовнішній ключ.
2. Унікальний ідентифікатор результату виявлення плагіату (plagiarism_detection_result_id) – зовнішній ключ.
3. Повідомлення про помилку (log_message) – причина, чому веб-додаток не зміг завантажити репозиторій студента.

3.3. Модуль взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS

Однією з головних особливостей дипломного проєкту є автоматичне завантаження програмних проєктів студентів. Це можливо лише, якщо студенти будуть розміщувати свої проєкти в мережі Інтернет. Але використання будь-яких хостингів не є коректним вирішенням поставленої задачі, оскільки потребує від студента додаткової роботи (якщо він раніше не користувався даними сервісами), або хостинг не надає API, щоб програмно завантажувати файли. Також важливим фактором є можливість розмежовувати доступ до проєктів, щоб студенти ще на етапі виконання завдання не могли безперешкодно завантажити чуже рішення. Для досягнення поставлених цілей найкращим вибором буде використання веб-сервісів для хостингу проєктів на базі систем керування версіями (VCS), таких як GitHub та Bitbucket. Вони широко поширені серед спеціалістів IT-галузі, дозволяють гнучко налаштовувати доступ до репозиторіїв та надають API з доступом до всіх можливостей, які має веб-інтерфейс.

Модуль взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS розроблений у веб-додатку підтримує два сервіси – GitHub та Bitbucket. Кожний з них має свої особливості в API та роботі з ним. Взаємодія з даними сервісами виконується за допомогою стандарту авторизації OAuth 2 [21]. Алгоритм авторизації схожий за винятком деяких відмінностей і має наступний вигляд:

1. Було зареєстровано новий OAuth додаток в системі – обов'язковими атрибутами є назва та Callback URL – URL, на який буде переадресовано користувача після успішної авторизації.
2. Згенеровані спеціальні два токени Client Id та Client Secret, які будуть використовуватися для авторизації користувачів до зареєстрованого OAuth додатка.
3. Для авторизації користувач має перейти за спеціальним URL з вказаним Client Id (в параметрі або тілі запиту).
4. Користувач переглядає запит на доступ до конкретних даних його профілю. Якщо він погодиться з ним та авторизується у веб-сервісі його перенаправить на раніше вказаний Callback URL з згенерованим кодом в параметрі URL.
5. Використовуючи отриманий код і Client Secret, сервер робить запит на сервіс та у відповідь отримує токен авторизації для даного користувача. Важливо відмітити, що токен Client Secret – приватний, він зберігається на сервері і не повинен бути у відкритому доступі.
6. Маючи токен авторизації, веб-додаток може робити запити до API сервісу та отримувати приватну інформацію з профілю користувача.

Більш детально про створення OAuth додатка та алгоритм авторизації з особливостями для кожного сервісу розглянуто в пунктах 4.1.1-4.1.2.

Після процедури авторизації веб-сервісів для розміщення програмних проєктів на базі VCS, користувач може додавати посилання на репозиторії до системи. Важливо, щоб користувач мав доступ на читання вказаних ним репозиторіїв (якщо вони приватні). Для цього їх власники мають власноруч надати доступ в налаштуваннях відповідного веб-сервісу, тільки тоді веб-додаток зможе взаємодіяти з даними репозиторіями, використовуючи токен авторизації користувача.

Модуль взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS дає можливість системі переглядати інформацію про репозиторії, завантажувати вихідні файли програмних проєктів. Також він має наступні налаштування для більш гнучкого завантаження (можлива їх комбінація):

1. Дата останнього оновлення – модуль буде завантажувати лише ті файли та їх версії, які були доступні до вказаної дати включно. Ця опція дозволяє користувачу виконати перевірку завдань вже після закінчення терміну їх виконання.
2. Префікс шляху – модуль завантажить не цілий репозиторій, а лише ту його частину, яка буде мати вказаний префікс (шлях директорій). Ця опція дозволяє використовувати один репозиторій для зберігання різних завдань в рамках одного курсу.

3.4. Модуль взаємодії із системою для виявлення плагіату в програмних проєктах

Дипломний проєкт передбачає можливість перевірки навчальних програмних проєктів на плагіат. В якості системи, яка буде аналізувати роботи студентів було обрано JPlag (див. підрозділ 2.5). Даний модуль реалізує взаємозв’язок веб-додатка з цією системою.

JPlag виконує перевірку програмних проєктів, розміщених на файловій системі. Необхідно попередньо завантажити роботи студентів в

директорії на сервері. Також варто відмітити, шо система JPlag не здатна опрацьовувати файли в яких присутні символи кирилиці. Тому перед записом репозиторіїв на файлову систему виконується пошук на наявні недопустимі символи, які потім видаляються.

Оскільки JPlag – це самостійна програма, то взаємодія з нею відбувається шляхом створення окремого процесу операційної системи. Такий підхід має свої переваги і недоліки. Головною проблемою в такому випадку є залежність JPlag від налаштувань операційної системи, а саме наявність JRE. Але даний недолік не є критичним, оскільки мова розробки серверної частини – Java, а отже необхідне JRE буде обов'язково присутнім в операційній системі, де розгорнутий сервер. Перевага такого підходу це те, що окремий процес керується операційною системою та ніяким чином не впливає на роботу веб-сервера.

Модуль взаємодії із системою для виявлення плагіату в програмних проєктах реалізований таким чином, щоб використовувати різні конфігурації налаштування JPlag:

1. Вибір мови програмування для аналізу (обов'язкова опція), доступні Java, C#-1.2, C, C++, Python 3, Scheme та текст на природній мові.
2. Чутливість порівняння – менше значення підвищує чутливість.
3. Мінімальний відсоток – необхідний мінімальний відсоток співпадіння двох порівнюваних програм для збереження факту співпадіння в результатах.
4. Архів з шаблонним кодом, який буде виключено з порівняння.

Користувач при створенні завдання на виявлення плагіату задає доступні налаштування у веб-інтерфейсі. Веб-додаток зберігає їх до бази даних і коли буде необхідно по таймеру виконати перевірку, відповідні налаштування будуть передані до модуля, який запустить систему JPlag з заданими опціями.

3.5. Модуль для організації робочого процесу користувача

Модуль для організації робочого процесу користувача фактично складається з двох незалежних частин:

1. На стороні веб-сервера відповідальний за виконання CRUD операцій з всіма сутностями системи.
2. На стороні клієнта – побудова інтерфейсу для виклику цих операцій, показу їх результатів.

Модуль є сполучною ланкою в роботі всієї системи та надає наступні можливості користувачу (див. Додаток 1. Функціональність веб-додатка):

1. Створення, редагування та видалення курсів.
2. Додавання груп до курсів, їх редагування та видалення.
3. Додавання студентів до системи (не в ролі користувача, а лише як представлення студента, репозиторій якого буде перевірятися на плагіат), їх видалення.
4. Додавання студентів до групи, обираючи зі списку уже створених та вказуючи посилання на репозиторій, який буде використовуватися студентом для збереження завдань в рамках даного курсу. Студент може належати до декількох груп одночасно, але обов’язково на різних курсах. Видалення студентів з групи. Система не дозволить видалити двох останніх студентів з групи, якщо ще наявні завдання для перевірки.
5. Додавання, редагування та видалення завдань. Користувач створює завдання в рамках курсу і лише потім назначає його до певної групи даного курсу.
6. Налаштування завдання для кожної групи окремо. Користувач, назначаючи завдання, обирає час та параметри перевірки. Система не дозволить назначити завдання для групи, якщо в ній менше двох студентів. Редагування та видалення уже створених завдань групи.
7. Перегляд результатів виявлення плагіату.

4. АНАЛІЗ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Особливості реалізації програмного забезпечення

При розробці програмного забезпечення дипломного проєкту була реалізована стандартна для веб-додатка функціональність, така як: реєстрація користувача, механізми автентифікації та авторизації (використано стандарт JWT [22]), набір CRUD операцій для наявних сутностей в системі. Натомість головними особливостями веб-додатка є інтеграція з веб-сервісами GitHub, Bitbucket, системою JPlag та їх спільна взаємодія для досягнення головної мети – автоматизації виявлення плагіату в навчальних програмних проєктах. Пропонується розглянути дані особливості більш детально.

4.1.1. *Взаємодія з веб-сервісом GitHub*

Для отримання доступу до даних користувачів необхідно зареєструвати OAuth App, в якому вони будуть авторизуватися:

1. У налаштуваннях перейти в меню OAuth Apps (рис. 4.1).

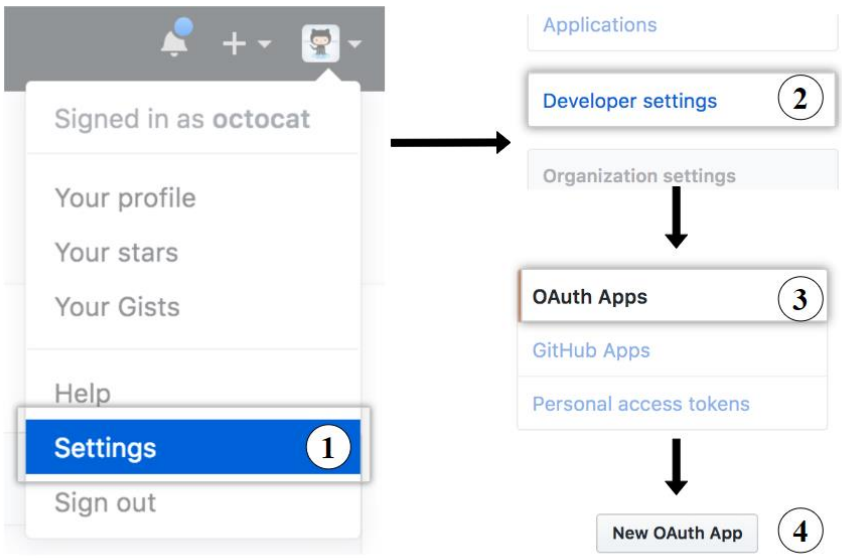


Рис. 4.1. Меню створення OAuth Apps на веб-сервісі GitHub

2. У полі «Application name» ввести назву програми. Важливо використовувати лише загальнодоступну інформацію та уникати конфіденційних даних, таких як внутрішні URL-адреси. У полі «Homepage URL» ввести повну URL-адресу веб-додатка. За бажанням заповнити поле «Application description». У полі «Authorization callback URL» ввести URL-адресу на яку сервіс після авторизації користувача відправить запит разом з параметром «code». Зареєструвати додаток (рис. 4.2).

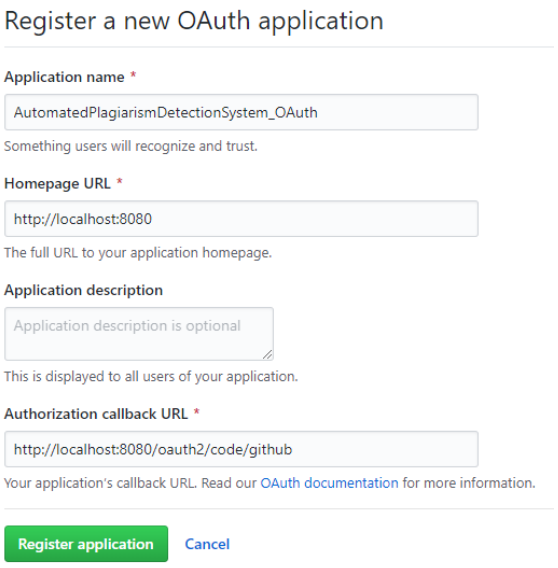


Рис. 4.2. Створення нового OAuth App на веб-сервісі GitHub

3. Отримані токени Client Id та Client Secret записати в конфігурацію сервера для використання в процедурі авторизації (рис. 4.3).

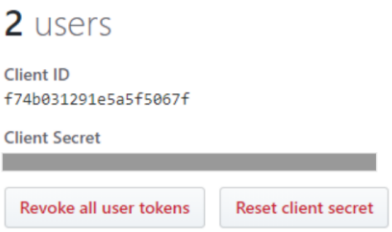


Рис. 4.3. Токени Client Id та Client Secret зареєстрованого OAuth App

Алгоритм авторизації користувача:

1. Для авторизації сервісу GitHub користувач веб-додатка натискає на посилання – спеціальним чином сформоване на сервері URL. Шаблон посилання: https://github.com/login/oauth/authorize. Детальний опис необхідних параметрів наведено у табл. 4.1.

Таблиця 4.1

Опис параметрів URL для авторизації сервісу GitHub

client_id	токен Client Id, отриманий під час реєстрації OAuth App
scope	«repo»– запит прав доступу до репозиторіїв користувача
state	випадково згенерований рядок, який має повернутися у відповідь (для захисту від CSRF)
redirect_uri	розширена адреса «Authorization callback URL» з вказаним ідентифікатором користувача

2. У веб-браузері відкривається нова веб-сторінка, де користувачу необхідно авторизуватися через GitHub та підтвердити запит доступу до даних (рис. 4.4).

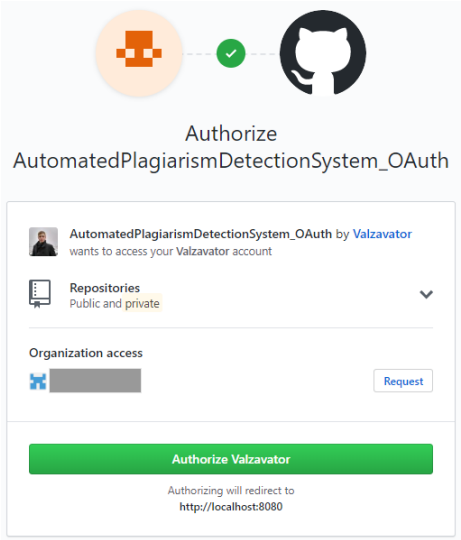


Рис. 4.4. Запит доступу до даних профілю GitHub

3. Якщо користувач приймає запит, GitHub виконує GET запит на «redirect_uri» з параметром «code» – тимчасовим кодом, а також зі «state» значення якого було задане на попередньому етапі. Термін дії тимчасового коду закінчується через 10 хвилин.
4. Сервер опрацьовує даний запит, запам’ятовує ідентифікатор користувача, перевіряє чи співпадає параметр «state». Якщо отримане значення не збігається з відправленим, запит був створений третьою стороною, процес переривається. У разі співпадіння виконується POST запит за нижче вказаним шаблоном посилання.

   Шаблон посилання: https://github.com/login/oauth/authorize.

   Детальний опис необхідних параметрів наведено в табл. 4.2.

Таблиця 4.2

Опис параметрів URL для отримання токена авторизації GitHub

client_id	токен Client Id, отриманий під час реєстрації OAuth App
client_secret	токен Client Secret, отриманий під час реєстрації OAuth App
code	тимчасовий код, отриманий на 3 кроці
state	випадково згенерований рядок, який має повернутися у відповідь (для захисту від CSRF)

5. Сервер отримує токен авторизації (лістинг 4.1) та перевіряє його на валідність, а також на наявність доступу до репозиторіїв. Якщо всі перевірки виконуються, сервер прив’язує даний токен до користувача веб-додатка за отриманим раніше ідентифікатором. Токен авторизації сервісу GitHub є безстроковий, тобто його не потрібно оновлювати з часом.

Лістинг 4.1. Приклад токена авторизації GitHub

```
{
   "access_token":"e72e16c7e42f292c6912e7710c838347ae178b4b",
   "scope":"repo",
   "token_type":"bearer"
}
```

4.1.2. *Взаємодія з веб-сервісом Bitbucket*

Для отримання доступу до даних користувачів необхідно зареєструвати OAuth consumer, в якому вони будуть авторизуватися:

1. Перейти в меню OAuth consumers (рис. 4.5).

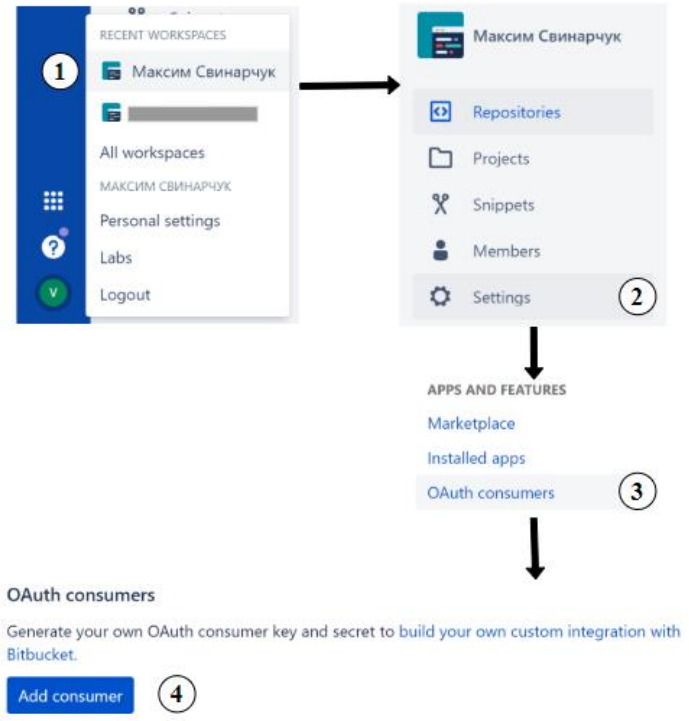


Рис. 4.5. Меню створення OAuth consumer на веб-сервісі Bitbucket

2. У полі «Name» ввести назву програми. У полі «Callback URL» ввести URL-адресу на яку буде переадресовано користувача з параметром «code». Відмітити поле «This is a private consumer». В налаштуваннях «Permissions» відмітити лише права доступу на читання репозиторіїв. На відміну від GitHub, Bitbucket не дає можливість динамічно визначати права доступу під час процесу авторизації. Зареєструвати додаток (рис. 4.6).

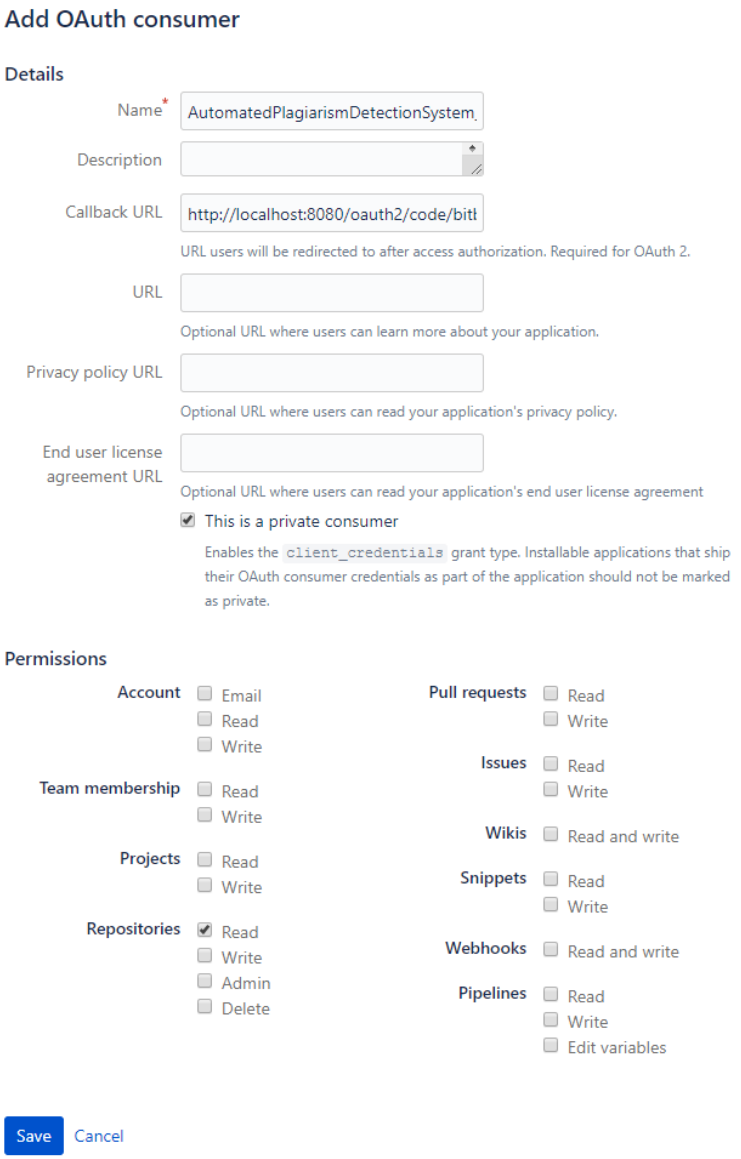


Рис. 4.6. Створення нового OAuth consumer на веб-сервісі Bitbucket

3. Отримані токени Key та Secret записати в конфігурації сервера для використання в процедурі авторизації (рис. 4.7).

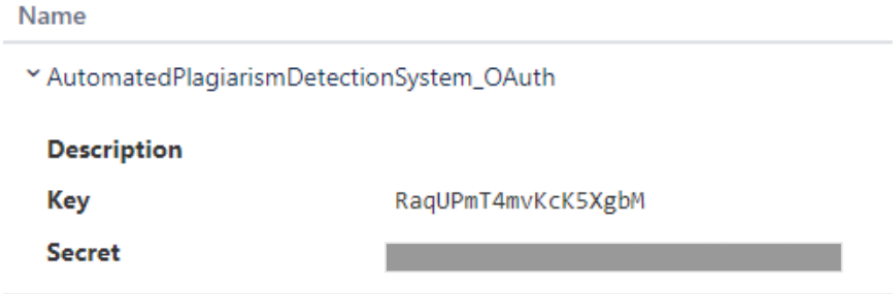


Рис. 4.7. Токени Key та Secret зареєстрованого OAuth consumer

Алгоритм авторизації:

1. Для авторизації сервісу Bitbucket користувач веб-додатка натискає на посилання – URL, яке було сформоване на сервері спеціальним чином.

   Шаблон посилання: https://bitbucket.org/site/oauth2/authorize.

   Детальний опис необхідних параметрів наведено в табл. 4.3.

Таблиця 4.3

Опис параметрів URL для авторизації сервісу Bitbucket

client_id	токен Key, отриманий під час реєстрації OAuth consumer
response_type	«code» – тип відповіді від сервісу
redirect_uri	розширена адреса «Callback URL» з вказаним ідентифікатором користувача

2. У веб-браузері відкривається нова веб-сторінка, де користувачу необхідно авторизуватися через Bitbucket та підтвердити запит на доступ до даних (рис. 4.8).

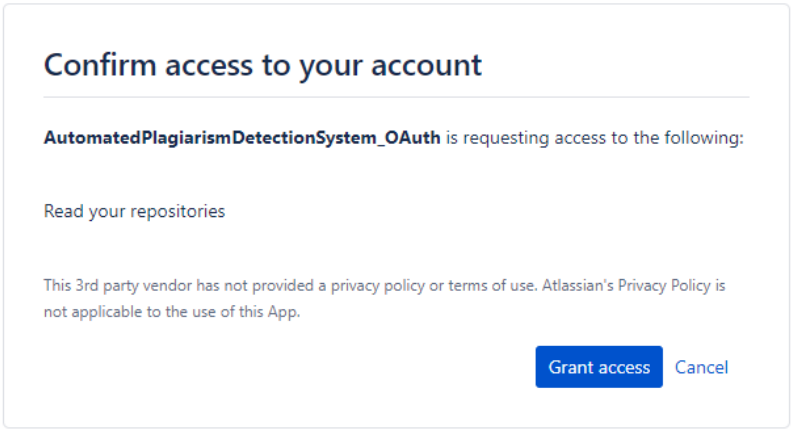


Рис. 4.8. Запит на доступ до даних профілю Bitbucket

3. Якщо користувач приймає запит, Bitbucket виконує GET запит на «redirect_uri» з параметром «code» – тимчасовим кодом.

4. Сервер опрацьовує отриману інформацію, запам’ятовує ідентифікатор користувача та виконує POST запит з тілом за нижче вказаним шаблоном посилання.
   Шаблон посилання: https://bitbucket.org/site/oauth2/access_token.
   Детальний опис полів тіла запиту наведено в табл. 4.4.

Таблиця 4.4

Опис тіла POST запиту для отримання токена авторизації

client_id	токен Key, отриманий під час реєстрації OAuth consumer
client_secret	токен Secret, отриманий під час реєстрації OAuth consumer
code	код, отриманий на 3 кроці
grant_type	тип даних для авторизації від сервісу, повинно дорівнювати «authorization_code»
redirect_uri	значення вказане на кроці 1, використовується сервісом для захисту від CSRF

5. Сервер отримує токен авторизації (лістинг 4.2). та перевіряє його на валідність, а також на наявність доступу до репозиторіїв. Якщо всі перевірки виконуються, сервер прив’язує даний токен до користувача веб-додатка за отриманим раніше ідентифікатором. Токен авторизації сервісу Bitbucket є тимчасовим і потребує оновлення з часом. Цю процедуру автоматично виконує веб-додаток.

Лістинг 4.2. Приклад токена авторизації Bitbucket

```
{   "access_token": "16jJLMy20muiBcgbJYp4hWjT2Sd7v9IXuAtGuJdTDkOI-UACr",
    "scopes": "repository",
    "expires_in": 7200,
    "refresh_token": "xhXDcMgBdQ9dBpskT5",
    "token_type": "bearer"
}
```

4.1.3. *Реалізація процесу виявлення програмного плагіату*

Користувачу веб-додатка доступні два різних підходи для виявлення програмного плагіату. Відрізняються вони лише тим, як система отримує та опрацьовує програмні проєкти перед використанням JPlag для їх перевірки (див. Додаток 1. Алгоритм роботи веб-додатка для виявлення програмного плагіату).

Користувач завантажує архів з програмними проєктами

Веб-додаток на стороні клієнта і сервера перевіряє завантажений архів на відповідність певним вимогам (див. підрозділ 4.3). Далі проводиться його розархівування на файловій системі сервера, водночас вміст перевіряється на наявність невідповідних символів, які заважають коректній роботі системі JPlag (див. підрозділ 3.4). Після цього для виявлення плагіату в окремому процесі запускається система JPlag з обраними користувачем налаштуваннями. Результат перевірки зберігається на файловій системі у вигляді HTML-документів. У базу даних записується інформація про успішність перевірки, журнал виконання системи JPlag та URL, за яким користувач може переглянути згенеровані HTML-документи. Ці ж дані повертаються користувачу веб-додатка.

Користувач налаштовує курси, груп, та завдання для автоматизації перевірки програмних проєктів

Особливість даного способу в тому, що користувач не займається завантаженням проєктів, а вказує системі коли та з якого веб-сервісу їх отримати. Детальніше налаштування робочого процесу описано в документі Керівництво користувача.

На стороні сервера безперервно в окремому потоці виконується спеціальний сервіс *Scheduler*. Він за заданим таймером (кожні 30 секунд) робить запит до бази даних, щоб отримати завдання, час перевірки яких вже настав. Якщо такі наявні, створюється окремий потік *Control thread* для їх обробки, а сервіс Scheduler далі продовжує роботу.

Control thread створює чергу задач для виконання перевірки за кожним завданням. Задача включає в себе три етапи виконання: 1) завантажити репозиторії з програмними проєктами до файлової системи сервера; 2) виконати перевірку на плагіат за допомогою системи JPlag; 3) зберегти результати до бази даних. Черга задач передається до Thread Pool Executor, який створює визначену кількість потоків (залежно від потужності процесора сервера). Кожний з цих потоків отримує задачу, виконує її та переключається на іншу (рис. 4.9).

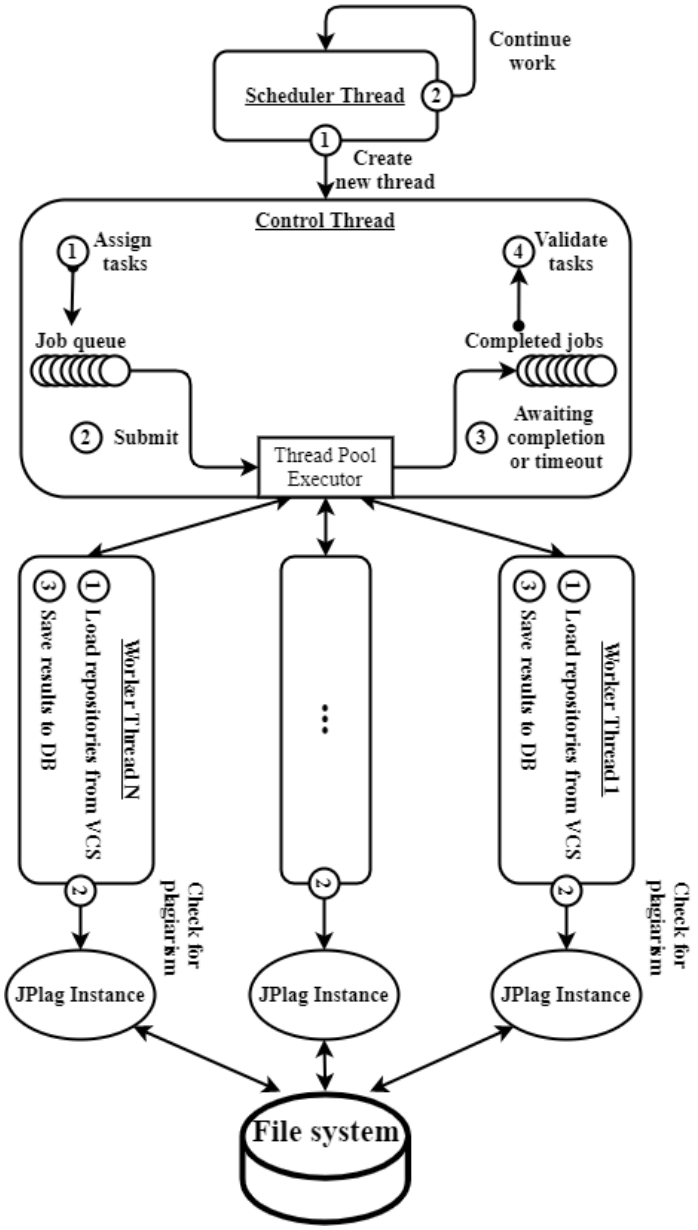


Рис. 4.9. Схема використання багатопотоковості для оптимізації процесу автоматизованої перевірки програмних проєктів

Control thread чекає виконання всіх задач, після чого перевіряє коректність завершення кожної з них. Якщо час виконання задачі перевищує встановлений таймер, потік буде перервано.

Гнучке налаштування багатопотоковості інструментами Java дозволяє досягти максимального використання ресурсів сервера та значно зменшити час виконання перевірки програмних проєктів за рахунок одночасного виконання множини таких задач.

4.2. Тестування веб-додатка

Для одночасного тестування клієнтської та серверної частин веб-додатка було використано методику ручного тестування для моделювання дій користувача. Усі операції виконувалися за допомогою інтерфейса. Таким чином було протестовано основну функціональність системи: реєстрація і авторизація користувача, створення, редагування та видалення різних сутностей системи (курси, групи, завдання, студенти).

Більш детально варто розглянути тестування особливостей розроблюваного веб-додатка, а саме операцій пов'язаних з автоматизацією виявлення програмного плагіату. Тестування було виконано відповідно до техніки сірого ящику, крім очікуваного результату в інтерфейсі програми, також перевірялися коректність записів збережених у базі даних та на файловій системі, а також записи в журналі роботи веб-додатка.

4.2.1. *Тестування ручної перевірки програмних проєктів на плагіат*

Опис тестового випадку №1: 1) користувач на сторінці разової перевірки завантажує архів з програмними проєктами; 2) налаштовує опції для системи JPlag та розпочинає перевірку.

Результат тестового випадку №1: 1) користувач отримав результати перевірки – журнал виконання системи JPlag та HTML-документи, де можна переглянути плагіат програмного

коду; 2) на файловій системі серверу зберігається вміст завантаженого архіву та згенеровані системою JPlag HTML-документи, в базі даних збережено загальний результат.

4.2.2. *Тестування авторизації користувача у веб-сервісах GitHub, Bitbucket*

Передумови: 1) користувач авторизований в системі; 2) на сторінці профілю показано, що він ще не прив’язав жодний сервіс.

Опис тестового випадку №1: 1) у профілі користувач натискає на кнопку авторизації навпроти будь-якого сервісу; 2) авторизується на відкритій сторінці даного сервісу.

Результат тестового випадку №1: 1) у профілі користувача авторизовано обраний сервіс; 2) в базі даних до користувача прив’язаний токен авторизації.

Опис тестового випадку №2: 1) користувач натискає на кнопку авторизації навпроти GitHub; 2) в URL адресі сторінки змінює параметр «scope» на будь-який інший або видаляє його (це завадить веб-додатку отримати доступ до інформації); 3) авторизується на оновленій сторінці.

Результат тестового випадку №2: 1) в профілі користувача GitHub не було авторизовано; 2) в базі даних отриманий токен авторизації не був прив’язаний до користувача; 3) в журналі роботи веб-додатка присутній запис про спробу додати некоректний токен авторизації.

4.2.3. *Тестування автоматизованої перевірки програмних проєктів на плагіат*

Передумови: 1) користувач авторизувався у веб-сервісах GitHub та Bitbucket; 2) було створено курс, в курсі створено групу та завдання; 3) студенти надали користувачу посилання на власні репозиторії GitHub чи Bitbucket, які будуть використовуватися ними в рамках даного курсу.

Опис тестового випадку №1: 1) користувач додає студента до групи та вказує його репозиторій, до якого студент не відкрив доступ для викладача.

Результат тестового випадку №1: 1) студента не було додано, користувач отримав повідомлення про відсутність доступу до репозиторію.

Опис тестового випадку №2: 1) користувач додає студентів до групи та вказує їх репозиторії, до яких має доступ; 2) назначає створене в рамках курсу завдання для даної групи, вказуючи час перевірки та опції для системи JPlag; 3) студенти виконують завдання та завантажують код програм до репозиторіїв в директорію з назвою завдання.

Результат тестового випадку №2: 1) до виконання перевірки завдання в групі отримали статус «Очікування»; 2) в заданий час сервер автоматично розпочав процес перевірки, завдання змінили статус на «В процесі», на файлову систему були завантажені директорії з репозиторіїв студентів за назвою завдання та згенеровані системою JPlag HTML-документи, в базі даних збережено загальний результат перевірки на плагіат та результат по кожному студенту окремо; 2) користувачу стали доступні результати: якщо системі JPlag не вдалося виконати перевірку – загальний статус помічено як «Невдача», або «Успіх» в іншому випадку; якщо веб-додаток не зміг завантажити репозиторій студента – навпроти нього буде вказано причину.

4.2.4. *Тестування багатопотоковості для оптимізації процесу автоматизованої перевірки програмних проєктів*

Передумови: в конфігурації сервера виділено 10 потоків на виконання процесу автоматизованої перевірки програмних проєктів.

Опис тестового випадку №1: 1) користувач налаштовує робочий процес та створює 15 завдань на перевірку в однаковий час.

Результат тестового випадку №1: 1) в заданий час система одночасно розпочала перевірку 10 завдань, їх статус змінився з

«Очікування» на «В процесі», решта залишилася в статусі «Очікування»; 2) як тільки будь-яке з 10 завдань отримало статус «Невдача» або «Успіх», завдання зі статусом «Очікування» перейшло в статус «В процесі»; 3) в результаті всі 15 завдань отримали статус «Невдача» або «Успіх».

4.3. Рекомендації щодо використання розробленого програмного забезпечення

Для правильного використання можливостей веб-додатка користувачам варто пам’ятати декілька правил. Необхідно уникати використання кирилиці в програмному коді. Система JPlag не здатна аналізувати програмні проєкти, де зустрічається кирилиця у файлах. Хоча веб-додаток попередньо видаляє заборонені символи, але це позначається на часі виконання перевірки. Також в назвах фалів і директорій варто використовувати лише латинські символи, це дозволить уникнути ситуацій, коли система JPlag в результатах покаже некоректні символи в назвах проєктів. Студентам варто створювати окремі репозиторії для кожного курсу, це попередить випадки коли в різних курсах є завдання з однаковими назвами.

Щодо налаштувань опцій використання системи JPlag:

1. Користувач обов’язково повинен вказати мову програмування, яка використовується в програмних проєктах, лише одну. Веб-додаток не здатний аналізувати вміст завантажених файлів програмного коду для самостійного вибору. Також варто звернути увагу, які формати файлів будуть аналізуватися системою JPlag для кожної мови програмування.
2. Бажано використовувати стандартне значення чутливості порівняння, яке пропонує веб-додаток для кожної мови окремо. Зменшення даного значення може призвести до хибних співпадінь, а збільшення – пропустити схожі ділянки коду.

3. Архів з шаблонним кодом та архів з проєктами в разовій перевірці повинні бути у форматі .zip розміром не більше 5 MB, без додаткових налаштувань архівації та встановленого пароля. Також архівувати потрібно самі файли/програмні проєкти, а не директорію в якій вони знаходяться.

4.4. Рекомендації щодо подальшого вдосконалення веб-додатка

Архітектура веб-додатка була спроєктована таким чином, щоб дозволити в майбутньому без особливих труднощів розширити функціональність системи або повністю змінити реалізацію деяких компонентів. Під час розробки програмного забезпечення дипломного проєкту було виділено декілька напрямків, щодо його вдосконалення.

4.4.1. *Розширення взаємодії з веб-сервісами для розміщення програмних проєктів на базі VCS*

У даній версії веб-додатка користувач може лише додавати посилання на репозиторії студентів, з яких система буде завантажувати код програм. Пропонується додати нову функціональність таку як, збір статистики по репозиторію студента, перегляд історії виконання завдання та перегляд власне самого програмного коду. Така інформація дозволить користувачу більш точно оцінити роботу кожного студента окремо.

4.4.2. Інтеграція нових веб-сервісів для розміщення програмних проєктів на базі VCS

У даній версії веб-додатка підтримується лише GitHub та Bitbucket. Пропонується розширити список доступних веб-сервісів, щоб дати більш широкий вибір для студентів. Головні вимоги до таких веб-сервісів – це можливість створювати приватні репозиторії, фіксування історії комітів та наявне API з повним доступом до функціональності веб-сервісу з підтримкою OAuth2.

4.4.3. Інтеграція нових систем для виявлення плагіату, можливість обирати яку саме використовувати.

JPlag не єдина система, яку можна без особливих проблем інтегрувати в даний програмний продукт. Маючи її як основну та безкоштовну можна додати підтримку платних аналогів, які надають більше можливостей для виявлення плагіату. Гарним претендентом є веб-додаток Codequiry (див. пункт 1.3.2). Він дозволяє додатково виконувати перевірку плагіату серед різних ресурсів мережі Інтернет. В такому випадку потрібно ще передбачити інтеграцію з платіжними системами.

4.4.4. Додавання студентів як повноцінних користувачів системи.

У даній версії веб-додатка студент є лише сутністю, створеною викладачем для прив'язки репозиторіїв. Додавання студентів як повноцінних користувачів системи дозволить полегшити роботу викладачів. Наприклад, викладачу більше не потрібно буде самостійно створювати цілі списки студентів для використання їх в системі. Можна запозичити ідею Google Classroom – для кожного курсу генерується унікальний код, студенти реєструються в системі та за даним кодом приєднуються до курсів. Також вони самостійно вказують репозиторії.

Створення акаунту для студента дозволить різним викладачам одночасно використовувати його у своїх курсах. Більш того, до акаунту буде прив'язана статистика студента по всім курсам, в яких він зареєстрований.

ВИСНОВКИ

Метою дипломного проєкту було створення веб-додатка для автоматизації виявлення плагіату в навчальних програмних проєктах.

Перед початком розроблення досліджено предметну область, проаналізовано існуючі аналоги, представлено ідею вирішення поставленої задачі. Аналіз різних типів програмного забезпечення та засобів їх розроблення, попередньо виконаний в дипломному проєкті, показав доцільність створення веб-додатка з використанням мови програмування Java та фреймворку Spring Boot на стороні сервера, JavaScript-фреймворку React та UI-фреймворку Bootstrap на стороні клієнта. В якості системи керування базами даних було обрано PostgreSQL, системи для виявлення програмного плагіату в навчальних програмних проєктах – JPlag.

Розроблений веб-додаток:

1. Забезпечує розмежування доступу до функціональності, реалізуючи процедури реєстрації та авторизації.
2. Дозволяє організувати робочий процес для користувача, створюючи курси, групи, завдання.
3. Взаємодіє з веб-сервісами GitHub та Bitbucket для завантаження програмних проєктів студентів.
4. Перевіряє завантажені проєкти студентів на плагіат та надає результати перевірки.

Розробка виконана у повному обсязі згідно з положенням Технічного завдання, тестування продукту проведено відповідно до затвердженої програми та методики тестування. Представлені напрямки подальшого вдосконалення веб-додатка.

Використання даної програмної розробки дозволить зменшити час, який витрачає викладач на монотонну роботу для перевірки програмних проєктів студентів на плагіат.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

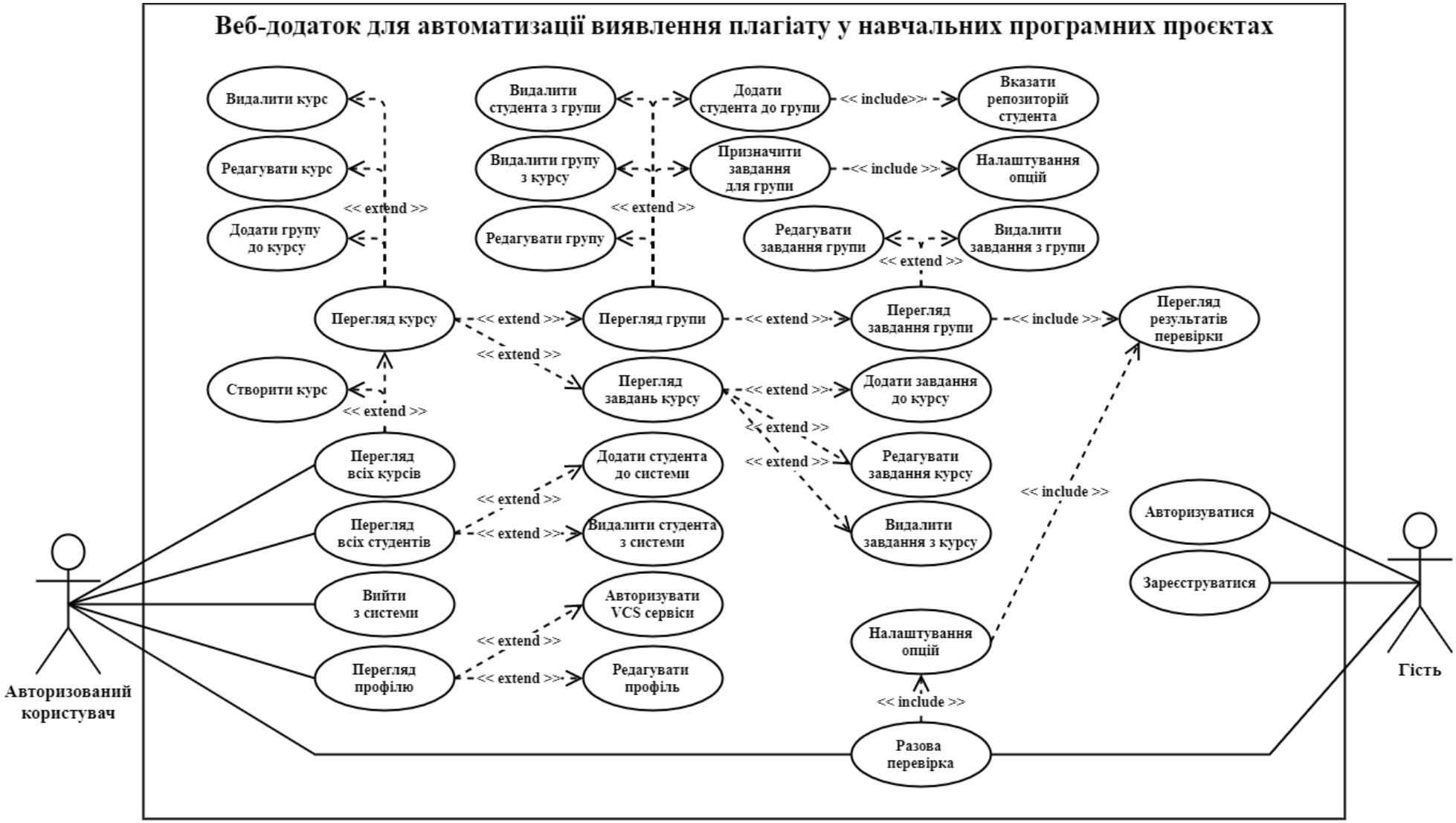
1. Chen, X. Shared information and program plagiarism detection [Text] / Xin Chen; Brent Francia; Ming Li, Brian McKinnon, Amit Seker. // IEEE Transactions on Information Theory. – 2004. – Vol. 50, № 7. – P. 1545-1551 – ISSN: 0018-9448.
2. Menai, M Similarity detection in Java programming assignments [Text] / Mohamed El Bachir Menai, Nailah Salah Al-Hassoun // 5th International Conference on Computer Science and Education. – 2010. – P. 356–361 – ISBN: 978-1-4244-6002-1.
3. About Classroom [Електронний ресурс]. – Режим доступу: https://bit.ly/2Z9w5f1
4. Codequiry Usage Guide [Електронний ресурс]. – Режим доступу: https://codequiry.com/usage/docs
5. JPlag - Detecting Software Plagiarism [Електронний ресурс]. – Режим доступу: https://jplag.ipd.kit.edu
6. Features of Java [Електронний ресурс]. – Режим доступу: https://www.javatpoint.com/features-of-java
7. JavaScript [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
8. Features of Python – Review the irresistible Python attributes [Електронний ресурс]. – Режим доступу: https://bit.ly/2zFATy2
9. Spring Boot [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-boot#overview
10. Top 12 best CSS frameworks for web development 2020 [Електронний ресурс]. – Режим доступу: https://bit.ly/2ZatMZh
11. React vs Angular vs Vue.js — What to choose in 2020? [Електронний ресурс]. – Режим доступу: https://bit.ly/2STUrW7
12. The software and text similarity tester SIM [Електронний ресурс]. – Режим доступу: https://dickgrune.com/Programs/similarity_tester

13. Евтифеева, О. Анализ алгоритмов поиска плагиата в исходных кодах программ [Текст] / Евтифеева О.А., Красс А.Л., Лакунин М.А., Лысенко Е.А., Счастливцев Р.Р. // Научно-технический вестник информационных технологий, механики и оптики. – 2007. – Том 07, № 5. – С. 188-196 – ISSN 2226-1494.
14. Prechelt, L JPlag: Finding plagiarisms among a set of programs [Text] / Lutz Prechelt, Guido Malpohl, Michael Philippsen // Journal of Universal Computer Science. – 2002. – Vol. 8, № 11 – P. 1016-1038 – ISNN: 0948-695X.
15. Aiken, A. MOSS (Measure of software similarity) plagiarism detection system [Електронний ресурс]. – Режим доступу: https://theory.stanford.edu/~aiken/moss
16. Schleimer, S. Winnowing: local algorithms for document fingerprinting [Text] / Saul David Schleimer, Daniel S. Wilkerson, Alex Aiken // Proceedings of the ACM SIGMOD international conference on Management of data. – 2003. – P. 76-85 – ISBN: 978-1-58113-634-0.
17. Richards, M. Software Architecture Patterns [Text] / Mark Richards – O'Reilly Media, Inc., 2015 – 46 p. – ISBN: 978-1-491-92424-2.
18. Front Controller [Електронний ресурс]. – Режим доступу: https://martinfowler.com/eaaCatalog/frontController.html
19. What is a Single Page Application? [Електронний ресурс]. – Режим доступу: https://bit.ly/2LeiPgP
20. Database Normalization Explained [Електронний ресурс]. – Режим доступу: https://bit.ly/35KelZ0
21. The OAuth 2.0 Authorization Framework [Електронний ресурс]. – Режим доступу: https://tools.ietf.org/html/rfc6749
22. Introduction to JSON Web Tokens [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction

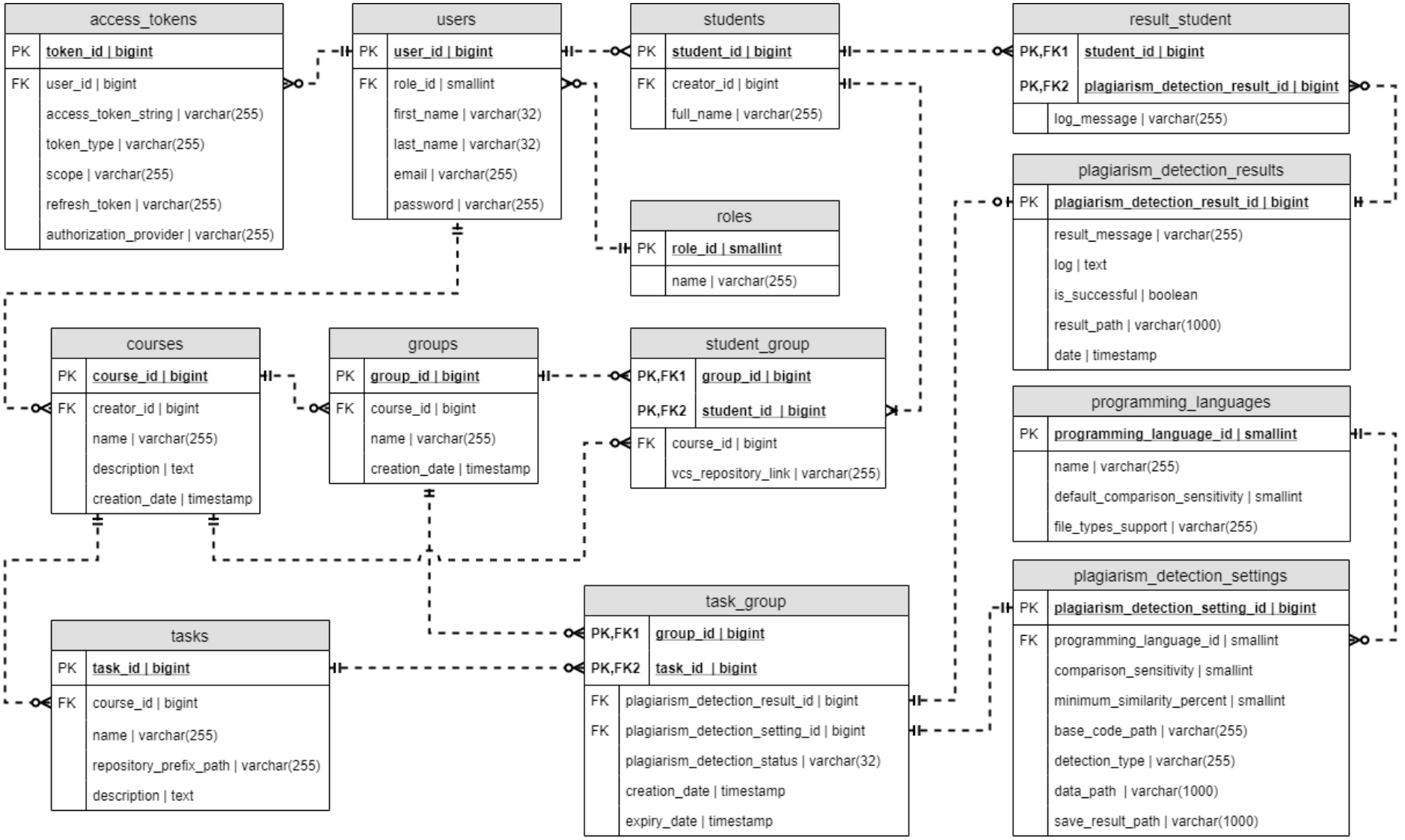
ДОДАТКИ

Додаток 1

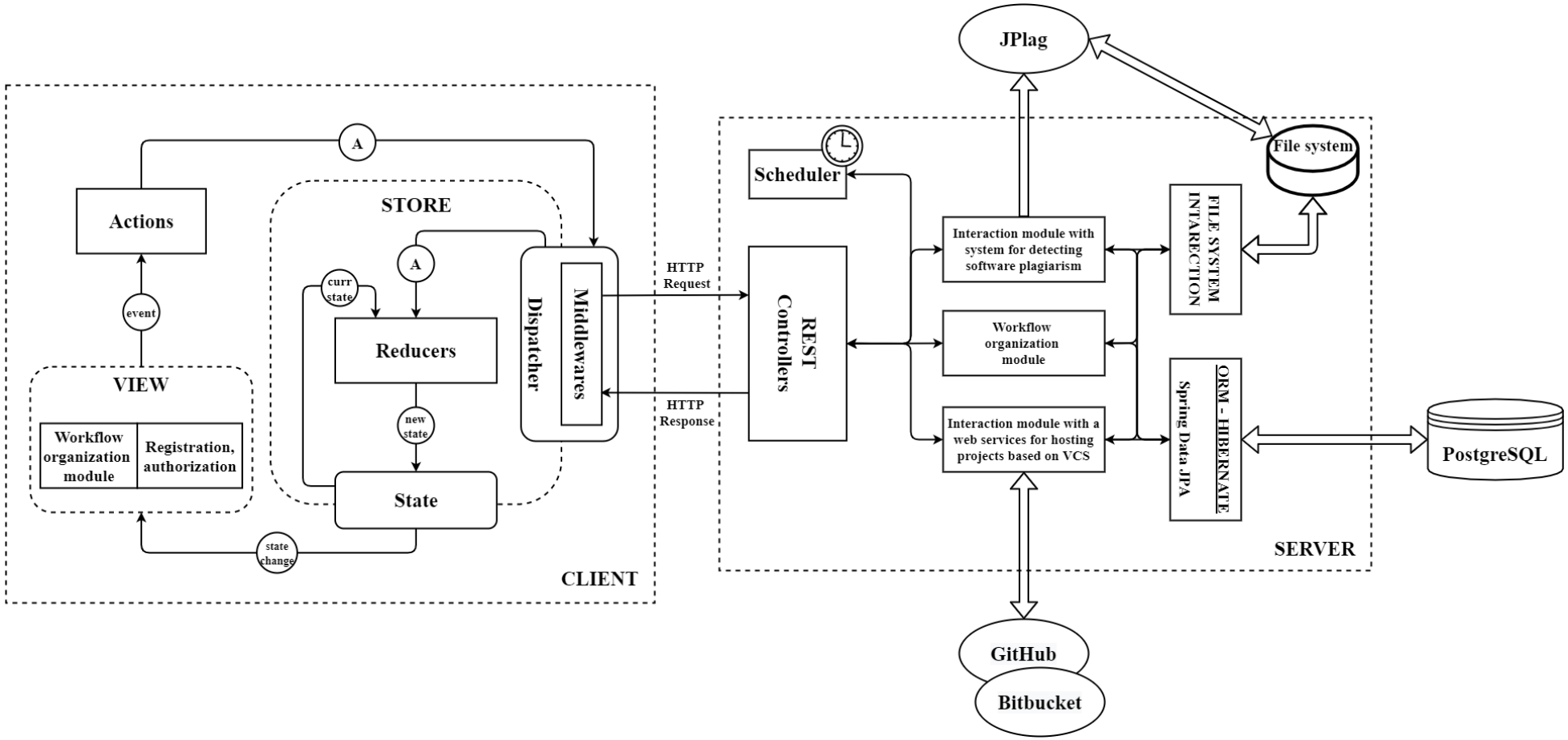
Копії графічних матеріалів



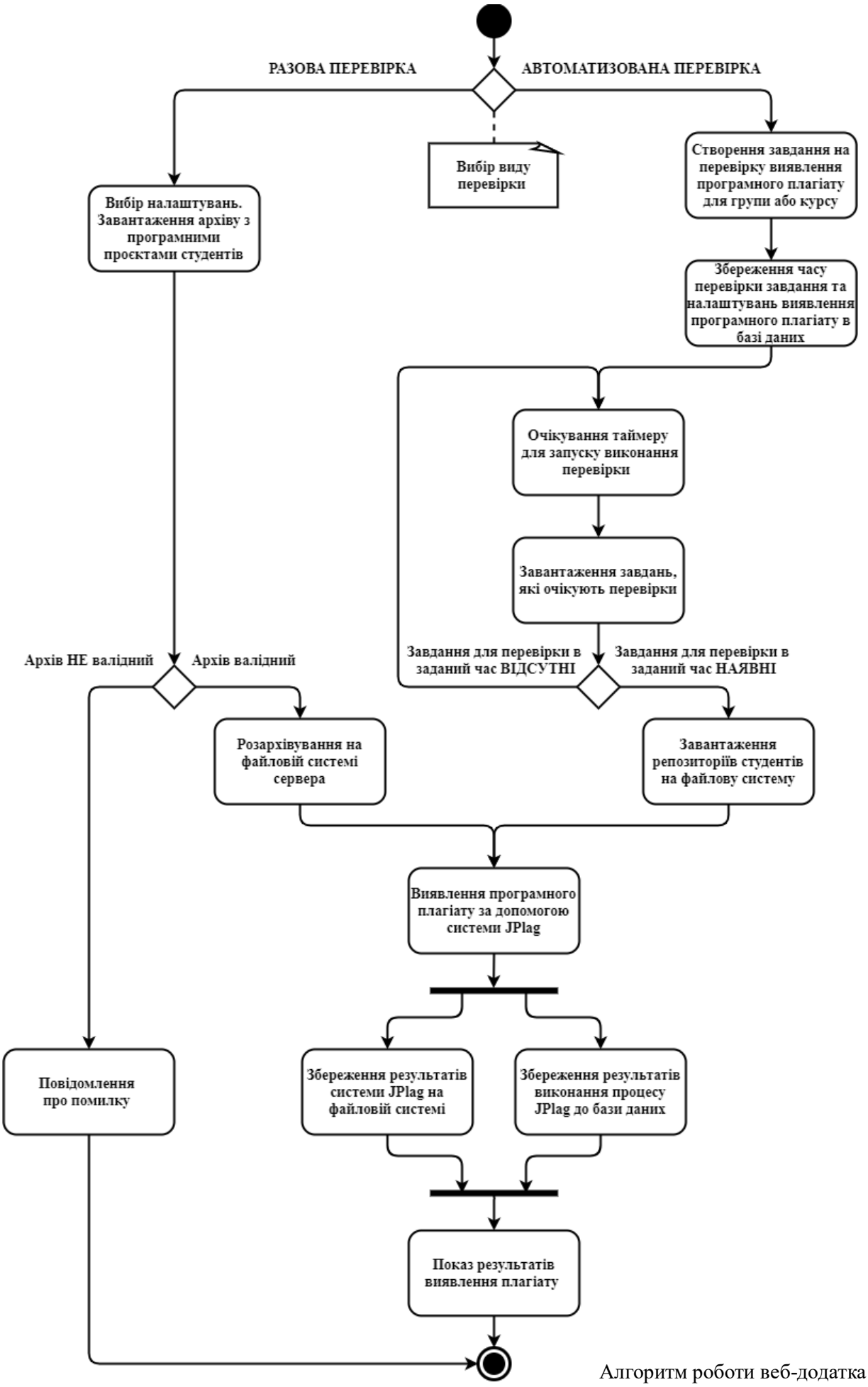
ДП.045440-06-99
Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах. Функціональність веб-додатка. UML-діаграма прецедентів



ДП.045440-07-99
Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах. Структура бази даних. ER-діаграма



Структурна схема веб-додатка
Свинарчук М.В., група КП-61



Алгоритм роботи веб-додатка для виявлення програмного плагіату
Свинарчук М.В., група КП-61

Додаток 2

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ПЛАГІАТУ В НАВЧАЛЬНИХ ПРОГРАМНИХ ПРОЄКТАХ

Виконав: студент групи КП-61 Свинарчук Максим Владиславович

Керівник: старший викладач кафедри ПЗКС Гадиняк Руслан Анатолійович

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити програмне забезпечення для автоматизації процесу виявлення плагіату в програмних навчальних проєктах.

① Аналіз

- проаналізувати існуючі аналоги;
- описати ідею автоматизації виявлення програмного плагіату.

② Розроблення

- обрати засоби реалізації;
- розробити всі складові ПЗ.

③ Результати

- порівняти з розглянутими аналогами;
- описати напрямки подальшого вдосконалення розробленого ПЗ.

АКТУАЛЬНІСТЬ

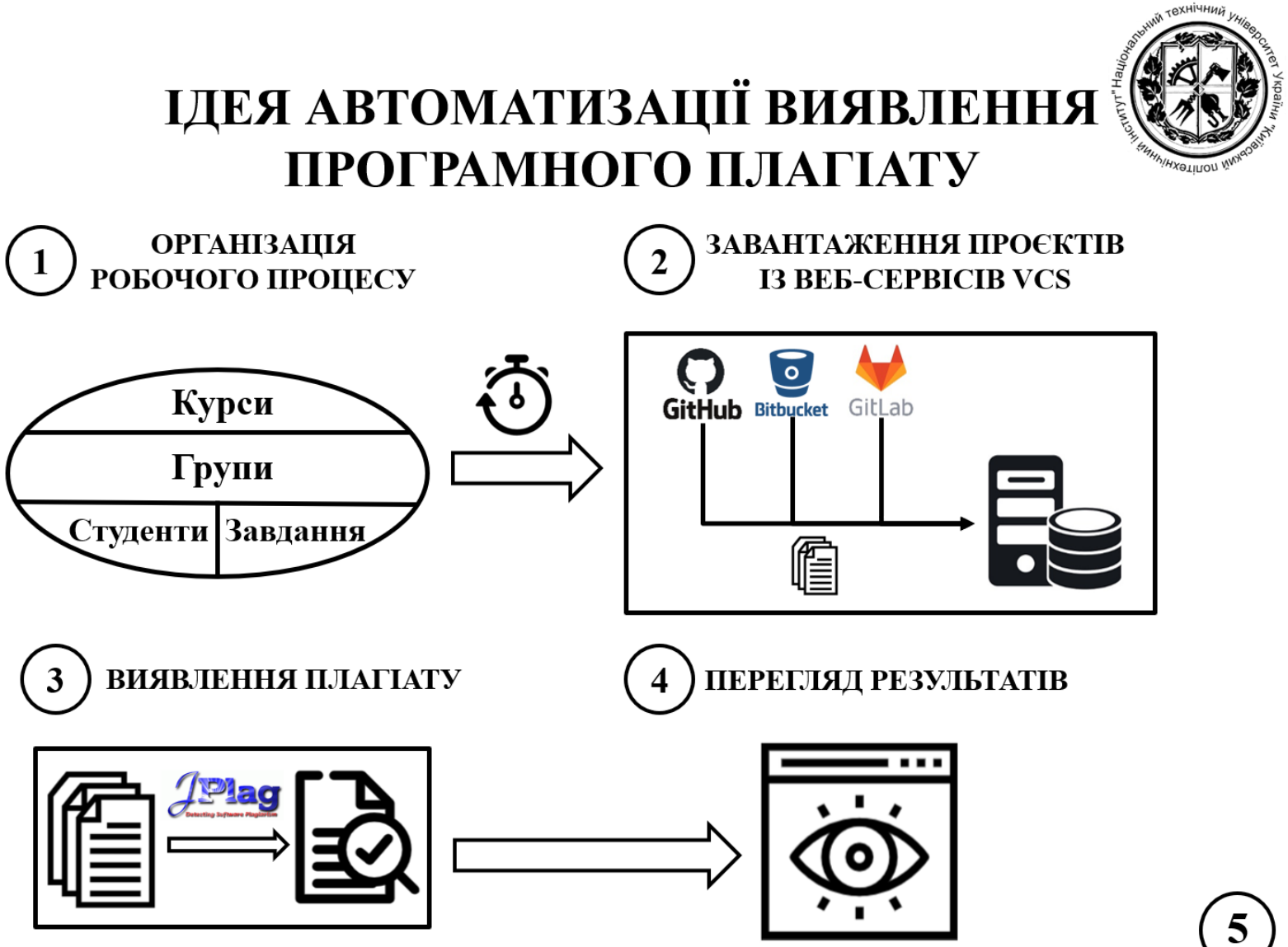
1. Велика кількість курсів з програмування.
2. Складність виявлення програмного плагіату.
3. Недосконаліть наявних рішень:
   - відсутня можливість організації навчального процесу;
   - необхідність власноруч завантажувати програмні проєкти до системи.

ІСНУЮЧІ АНАЛОГИ

SIM

MOSS

ІДЕЯ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ПРОГРАМНОГО ПЛАГІАТУ



ЗАСОБИ РЕАЛІЗАЦІЇ: ЗАГАЛЬНІ ВІДОМОСТІ

Тип ПЗ – **веб-додаток**.

Мова програмування (сервер) – **Java**.

Мова програмування (клієнт) – **JavaScript**.

База даних – **PostgreSQL 12**.

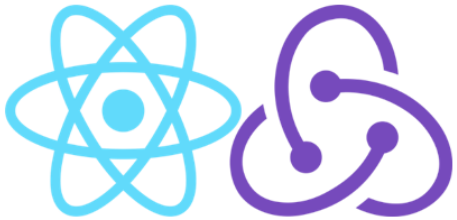
Система для виявлення плагіату – **JPlag**.

ЗАСОБИ РЕАЛІЗАЦІЇ: ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ

Серверна частина:

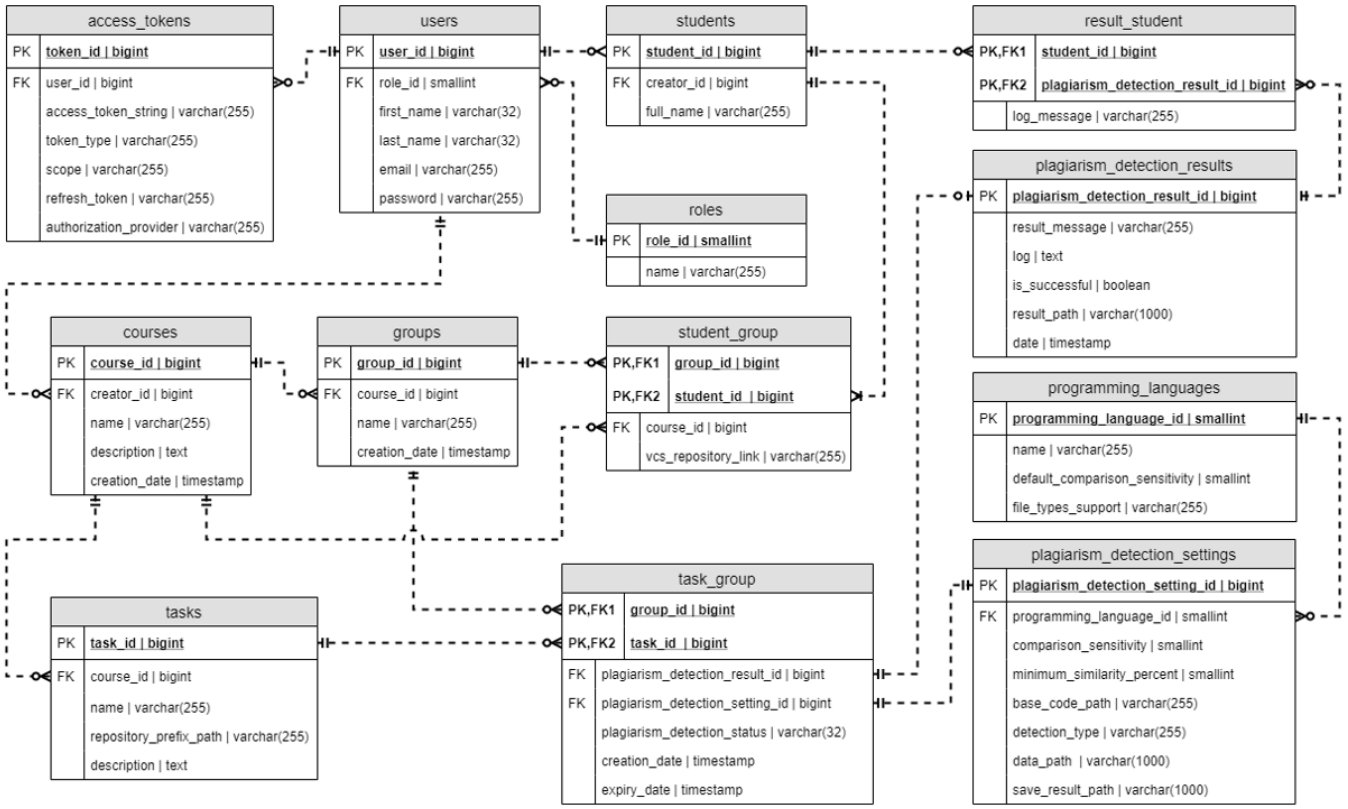
- веб-фреймворк – **Spring Boot**;
- контроль доступу – **Spring Security**;
- ORM для роботи с БД – **Spring Data JPA (Hibernate)**.

Клієнтська частина:

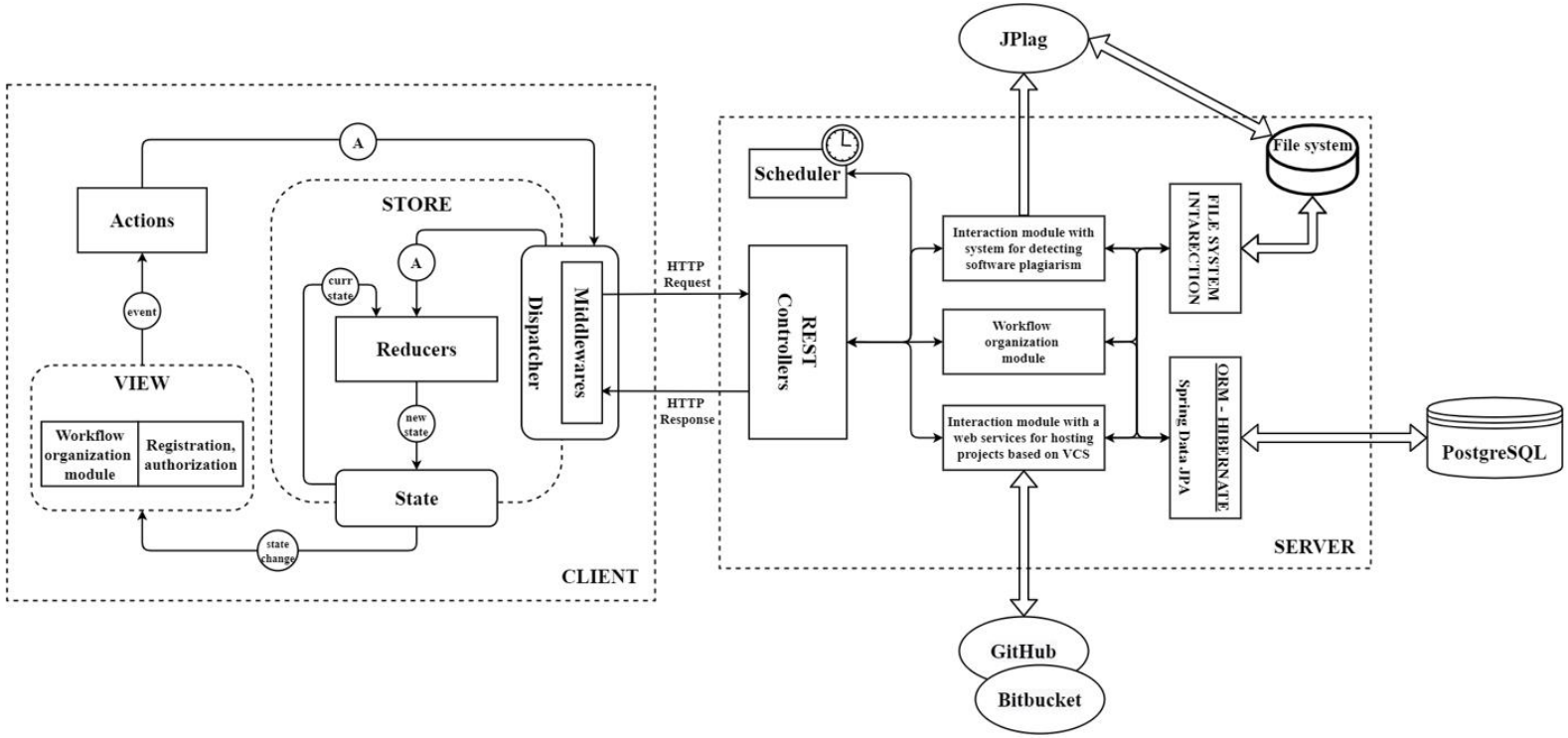


- бібліотеки – **React + Redux**;
- UI-фреймворк – **Bootstrap 4**.

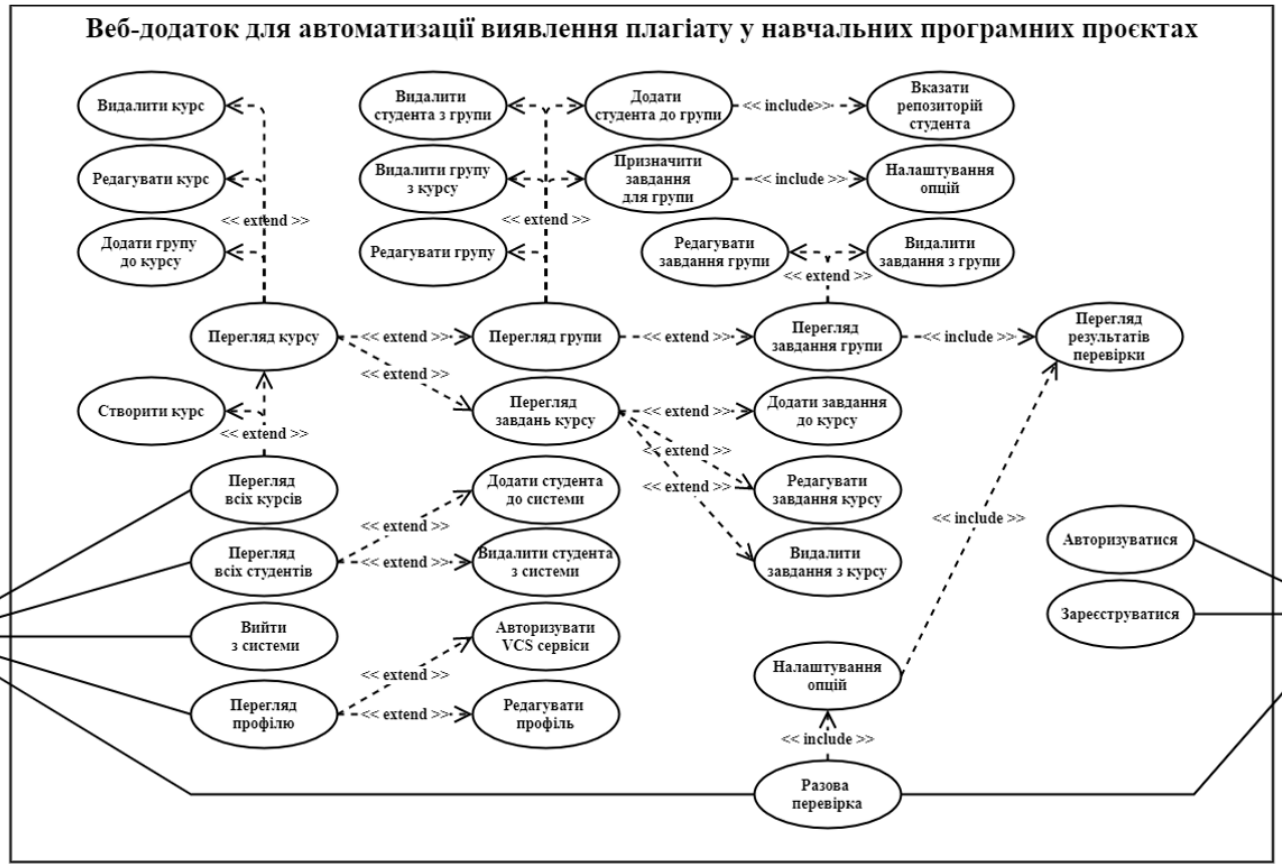
СТРУКТУРА БАЗИ ДАНИХ



АРХІТЕКТУРА СИСТЕМИ



ФУНКЦІОНАЛЬНІСТЬ СИСТЕМИ



ІНТЕГРАЦІЯ З ВЕБ-СЕРВІСАМИ VCS

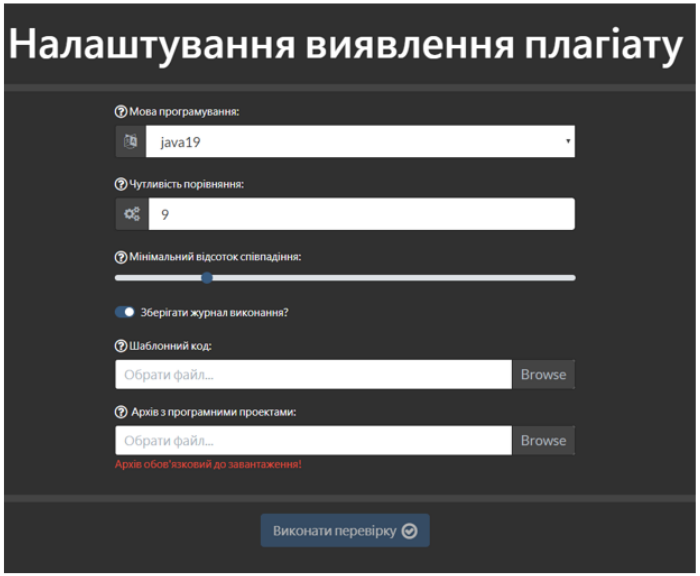
OAuth App **OAuth Consumer**

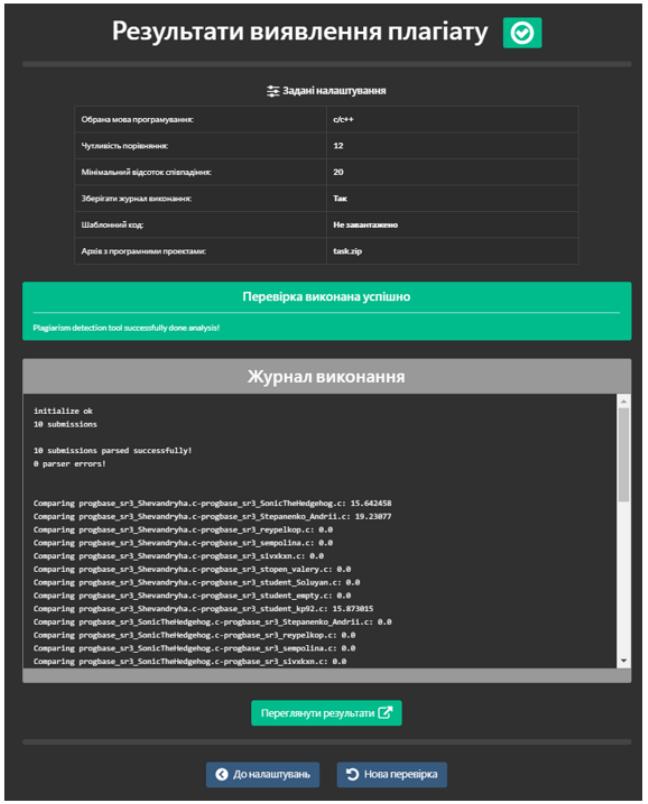
ІНТЕГРАЦІЯ З СИСТЕМОЮ JPLAG ДЛЯ ВИЯВЛЕННЯ ПЛАГІАТУ

Налаштування JPlag:

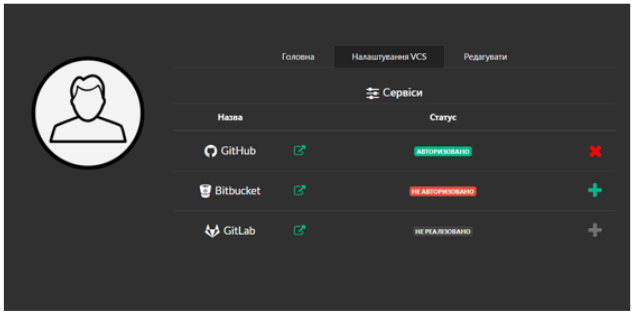
- мова програмування: Java, C#-1.2, C, C++, Python 3, Scheme та текст на природній мові;
- чутливість порівняння;
- мінімальний відсоток співпадіння;
- архів з шаблонним кодом.

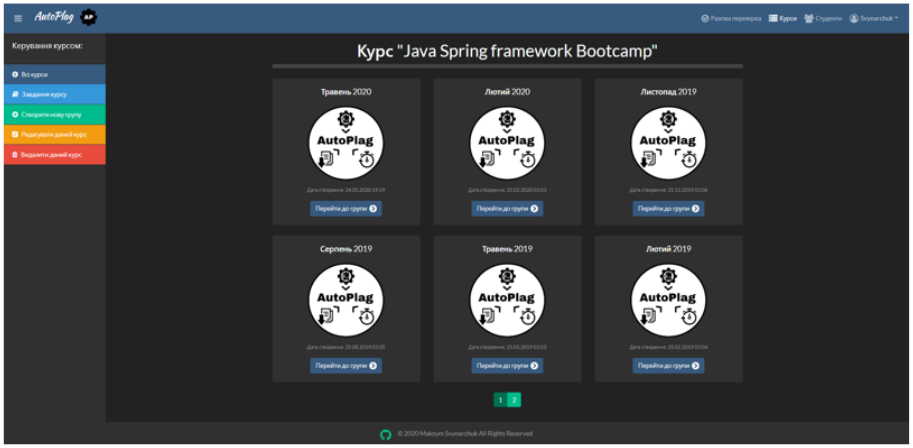
ПРИКЛАД РОБОТИ ПРОГРАМИ. РАЗОВА ПЕРЕВІРКА

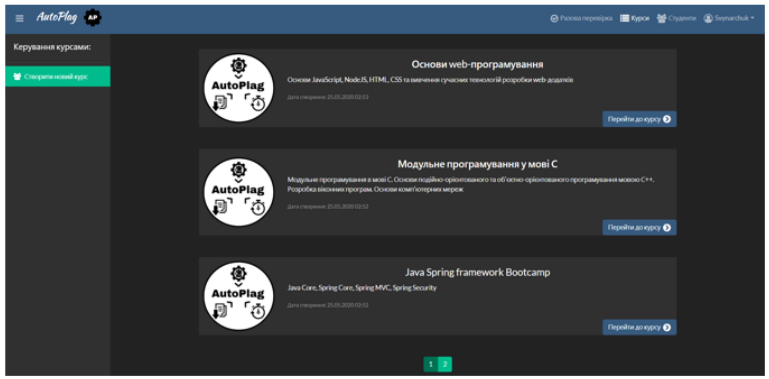


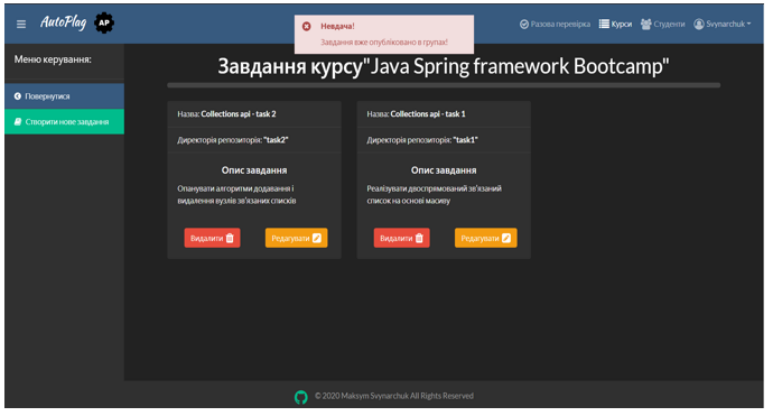


ПРИКЛАД РОБОТИ ПРОГРАМИ. АВТОМАТИЗОВАНА ПЕРЕВІРКА

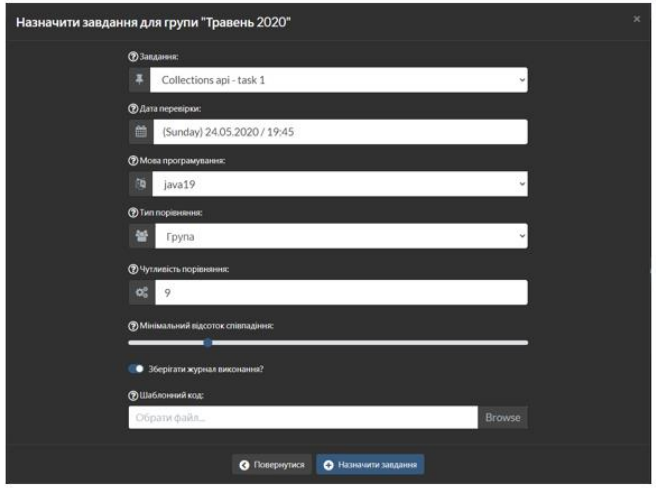


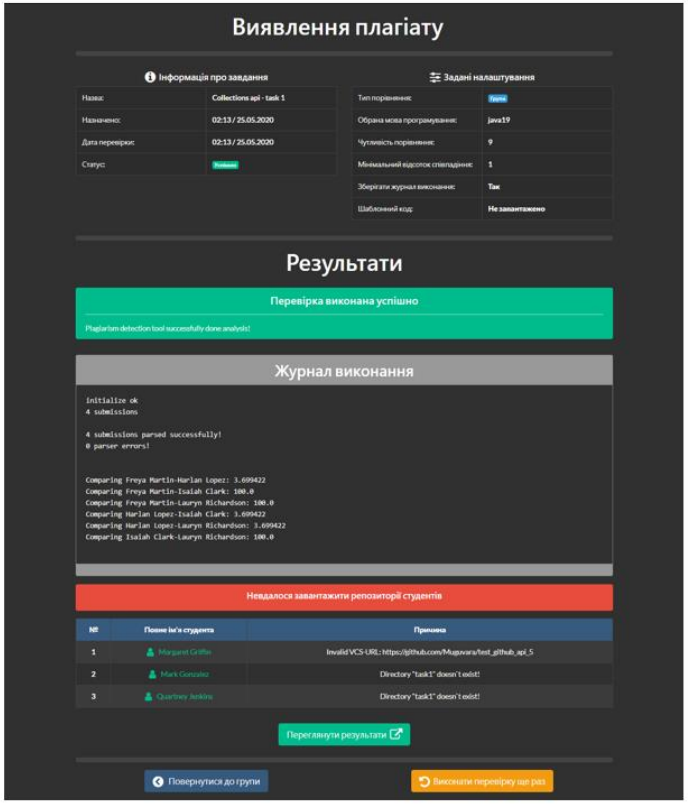


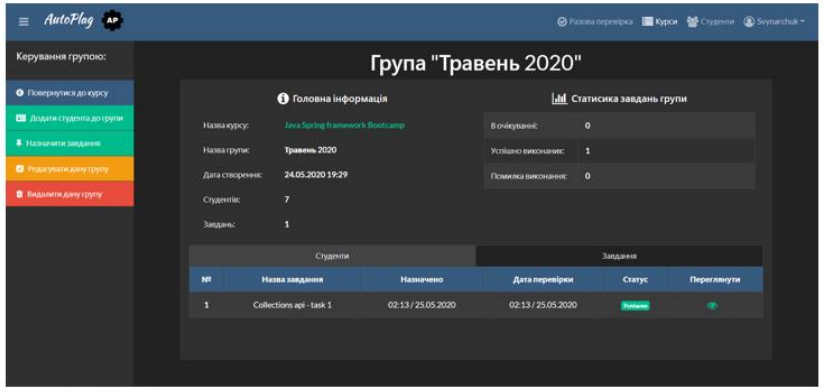




ПРИКЛАД РОБОТИ ПРОГРАМИ. АВТОМАТИЗОВАНА ПЕРЕВІРКА







ПРИКЛАД РОБОТИ ПРОГРАМИ. ПЕРЕГЛЯД РЕЗУЛЬТАТІВ

JPlag

Search Results

Title:	
Programs:	progbase_sr3_Shevandryha.c - progbase_sr3_SonicTheHedgehog.c - progbase_sr3_Stepanenko_Andrii.c - progbase_sr3_reypelkop.c - progbase_sr3_sempolina.c - progbase_sr3_sivxkxn.c - progbase_sr3_stopen_valery.c - progbase_sr3_student_Soluyan.c - progbase_sr3_student_empty.c - progbase_sr3_student_kp92.c - progbase_sr3_valerystudent.c
Language:	C/C++ Scanner [basic markup]
Submissions:	11
Matches displayed:	8 (Treshold: 21.1%) (average similarity) 12 (Treshold: 20.5%) (maximum similarity)
Date:	2020-05-31
Minimum Match Length (sensitivity):	12
Suffixes:	.cpp, .CPP, .cxx, .CXX, .c++, .C++, .c, .C, .cc, .CC, .h, .H, .hpp, .HPP, .hh, .HH

Distribution:

90% - 100%	0
80% - 90%	1 ##
70% - 80%	0
60% - 70%	0
50% - 60%	0
40% - 50%	4 ########
30% - 40%	0
20% - 30%	3 ######
10% - 20%	10 ####################
0% - 10%	37 ##

Matches sorted by average similarity (What is this?):

progbase_sr3_valerystudent.c	->	progbase_sr3_Stepanenko_Andrii.c (85.0%)	progbase_sr3_stopen_valery.c (46.9%)	progbase_sr3_student_kp92.c (45.4%)
progbase_sr3_Stepanenko_Andrii.c	->	progbase_sr3_student_kp92.c (44.3%)	progbase_sr3_stopen_valery.c (41.7%)	progbase_sr3_student_empty.c (27.7%)
progbase_sr3_student_empty.c	->	progbase_sr3_stopen_valery.c (21.1%)		

Matches sorted by maximum similarity (What is this?):

progbase_sr3_valerystudent.c	->	progbase_sr3_Stepanenko_Andrii.c (87.0%)	progbase_sr3_student_kp92.c (52.8%)	progbase_sr3_stopen_valery.c (52.0%)
progbase_sr3_Stepanenko_Andrii.c	->	progbase_sr3_student_kp92.c (52.9%)	progbase_sr3_stopen_valery.c (45.2%)	progbase_sr3_student_empty.c (28.2%)
progbase_sr3_stopen_valery.c	->	progbase_sr3_student_empty.c (23.2%)	progbase_sr3_student_kp92.c (20.5%)	
progbase_sr3_student_kp92.c	->	progbase_sr3_Shevandryha.c (21.1%)		

Matches for progbase_sr3_valerystudent.c & progbase_sr3_Stepanenko_Andrii.c

85.0%

progbase_sr3_valerystudent.c (83.146065%)	progbase_sr3_Stepanenko_Andrii.c (87.05882%)	Tokens
progbase_sr3_valerystudent.c(7-12)	progbase_sr3_Stepanenko_Andrii.c(6-11)	15
progbase_sr3_valerystudent.c(17-66)	progbase_sr3_Stepanenko_Andrii.c(14-80)	59

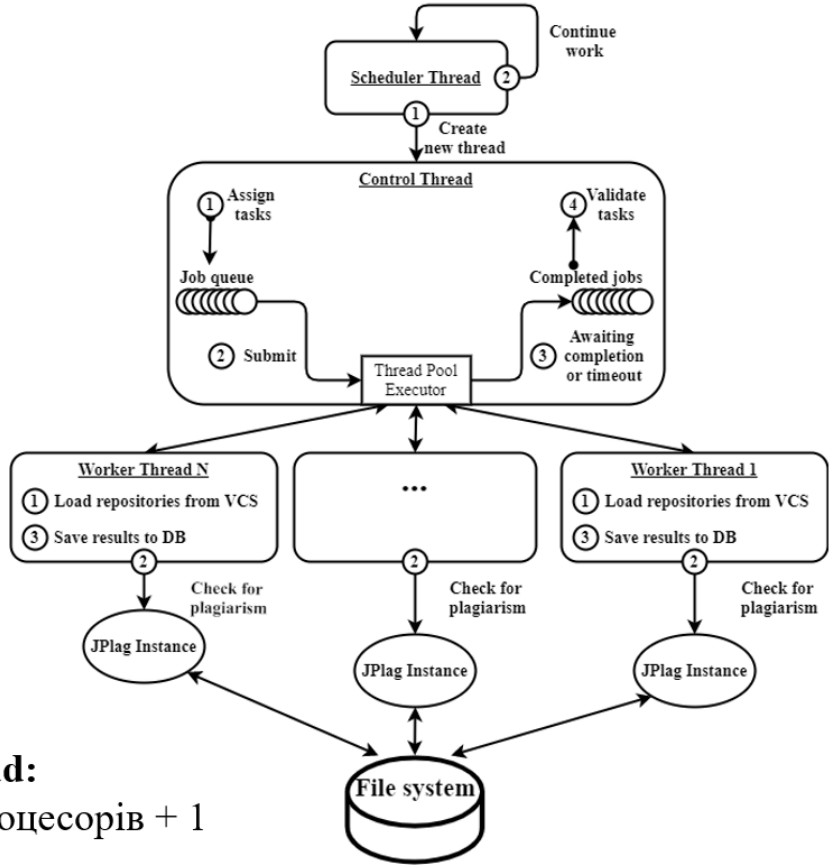
progbase_sr3_valerystudent.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>

char *getSpecialString(char *buffer, size_t bufferLength, const ch
{
    int first_space = 0;
    int last_space = 0;
    int score = 0;
    int count = 0;
    int count2=0;


    char arr1[100] = {};
    char arr2[100] = {};
    for (int i = 0; str[i] != '\0'; i++)
    {
        if (isspace(str[i]) != 0)
        {
            first_space = i;
            break;
        }
    }
    for (int i = first_space + 1; str[i] != '\0'; i++)
    {
        if (isspace(str[i]) != 0)
        {
            last_space = i;
        }
    }


    strncpy(arr1, str + first_space, last_space - first_space + 1);
    arr1[last_space - first_space + 1] = '\0';
   // puts(arr1);

    for (int i = 0; arr1[i] != '\0'; i++)
    {
        for (int j = 0; str2[j] != '\0'; j++)
        {
            if (arr1[i] != str2[j])
            {
                count++;
            }
        }
        if (count != 0)
        {
            arr2[score] = arr1[i];
            score++;
        }
        count=0;
    }
    if (strlen(arr2) > bufferLength)
    {
```

progbase_sr3_Stepanenko_Andrii.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

char * getSpecialString(char * buffer, size_t bufferLength, const ch
    int first_index = 0;
    int last_index = 0;
    int counter = 0;
    int count = 0;

    char temp1[1000];
    char temp_str[1000] = {};

    for (int i = 0; str[i] != '\0'; i++) {

        if (isspace(str[i]) != 0) {

            first_index = i;
            break;

        }

    }

    for (int i = first_index + 1; str[i] != '\0'; i++) {

        if (isspace(str[i]) != 0) {

            last_index = i;

        }

    }

    strncpy(temp1, str + first_index, last_index - first_index + 1);

    temp1[last_index - first_index + 1] = '\0';

    for (int i = 0; temp1[i] != '\0'; i++) {

        for (int j = 0; str2[j] != '\0'; j++) {

            if ((temp1[i] == str2[j]) && (isalnum(temp1[i])!= 0)) {

                counter++;

            }

        }

        if (counter != 0) {

            temp_str[count] = temp1[i];
            count++;
```

ОПТИМІЗАЦІЯ АВТОМАТИЗОВАНОЇ ПЕРЕВІРКИ



Кількість Worker Thread:
2 * кількість логічних процесорів + 1

ПОРІВНЯННЯ З АНАЛОГАМИ

Критерій	Google Classroom	Codequiry	JPlag	**Розроблене ПЗ**
Організація навчального процесу	+	+	–	+
Створення завдань на реалізацію програмних проєктів	+	–	–	+
Автоматизація завантаження робіт студентів	+	–	–	+
Можливість виявлення програмного плагіату	–	+	+	+
Налаштування процесу виявлення програмного плагіату	–	–	+	+
Різні методи виявлення програмного плагіату	–	+	–	–
Аналітика результатів виявлення програмного плагіату	–	+	+	+

НАПРЯМКИ РОЗВИТКУ

- інтеграція нових веб-сервісів VCS;
- розширення взаємодії.

- інтеграція нових систем для виявлення плагіату;
- можливість вибору.

- нова роль «Студент»;
- розширення можливостей керування навчальним процесом.

ВИСНОВКИ

1. Проаналізовано існуючі рішення.
2. Висунуто ідею розробки.
3. Спроєктовано архітектуру системи та структуру бази даних.
4. Налаштовано взаємодію з веб-сервісами GitHub та Bitbucket.
5. Налаштовано взаємодію з системою JPlag.
6. Реалізовано функціональність для організації робочого процесу.
7. Представлені напрямки подальшого вдосконалення системи.

Розробка виконана у повному обсязі згідно з положенням Технічного завдання, тестування продукту проведено відповідно до затвердженої програми та методики тестування.

ПЕРЕВІРКА НА ПЛАГІАТ

Submission author:
Гадиняк Руслан Анатолійович

Check ID:
1003888621

Check date:
08.06.2020 23:57:46 EEST

Check type:
Doc vs Internet + Library

Report date:
09.06.2020 00:05:59 EEST

User ID:
100000126

File name: **Свинарчук_ПЗ**

File ID: **1003902930** Page count: **69** Word count: **12094** Character count: **91061** File size: **623.84 KB**

1.98% Matches

Highest match: **0.26%** with library source. File ID: **1000068007**

0.38% Internet Matches	3	Page 71
1.69% Library matches	45	Page 71

0.62% Quotes

Quotes	5	Page 72

Exclude references is off

0% Exclusions

No exclusions found

Replacement

No replaced characters found

Дякую за увагу!

Додаток З

Лістинг програми

VcsOAuthGitHubDao.java

```
package com.autoplag.persistence.vcs.impl.github;

import com.autoplag.config.constant.VCS;
import com.autoplag.persistence.domain.type.AuthorizationProvider;
import com.autoplag.persistence.domain.vcs.AccessToken;
import com.autoplag.persistence.exception.oauth.InvalidVcsUrlException;
import com.autoplag.persistence.exception.oauth.OAuthIllegalTokenException;
import com.autoplag.persistence.vcs.VcsOAuthDao;

@Repository("vcsOAuthGitHubDao")
public class VcsOAuthGitHubDao implements VcsOAuthDao {

	@Override
	public String getAuthorizeOAuthUrl(String redirectUrl) {
		String redirectParameter = null;
		if (Objects.nonNull(redirectUrl) && redirectUrl.length() > 0) {
			if
(!redirectUrl.startsWith(VCS.GITHUB_AUTHORIZE_OAUTH_CALLBACK_URL)) {
				throw new InvalidVcsUrlException(redirectUrl);
			}
			redirectParameter = "redirect_uri" + "=" + redirectUrl;
		}
		return VCS.GITHUB_AUTHORIZE_OAUTH_URL +"?" + "client_id" + "="
+ VCS.GITHUB_AUTHORIZE_OAUTH_CLIENT_ID +"&" + "scope" + "=" +
VCS.GITHUB_AUTHORIZE_OAUTH_SCOPE +"&" + "state" + "=" +
VCS.GITHUB_AUTHORIZE_OAUTH_STATE +(Objects.nonNull(redirectParameter) ? "&"
+ redirectParameter : "");
	}

	@Override
	public AccessToken getAuthorizeOAuthToken(String code, String
returnedState) {
		if (!VCS.GITHUB_AUTHORIZE_OAUTH_STATE.equals(returnedState)) {
			throw new OAuthIllegalAuthorizeStateException();
		}

		AccessToken accessToken =
Unirest.post(VCS.GITHUB_AUTHORIZE_OAUTH_TOKEN_URL)
		.queryString("client_id", VCS.GITHUB_AUTHORIZE_OAUTH_CLIENT_ID)
		.queryString("client_secret",
VCS.GITHUB_AUTHORIZE_OAUTH_CLIENT_SECRET)
		.queryString("code", code)
		.queryString("state", VCS.GITHUB_AUTHORIZE_OAUTH_STATE)
		.header("Accept", "application/json")
		.asObject(AccessToken.class)
		.getBody();

	accessToken.setAuthorizationProvider(AuthorizationProvider.GITHUB);

		validateAccessToken(accessToken);

		return accessToken;
	}

	/**
	 * @param expiredAccessToken OAuth2 token for GitHub
	 * @throws OAuthIllegalTokenException invalid token initially,
	 *    GitHub hasn't time limit for oauth tokens
	 */
```

```
	@Override
	public AccessToken getRefreshedOAuthToken(AccessToken expiredToken) {
		throw new OAuthIllegalTokenException(expiredToken);
	}

	private void validateAccessToken(AccessToken accessToken) {
		if (Objects.isNull(accessToken) ||
				Objects.isNull(accessToken.getAccessToken()) ||
				Objects.isNull(accessToken.getTokenType())) {
			throw new OAuthIllegalTokenException(accessToken);
		}
		if (Objects.isNull(accessToken.getScope()) ||
	!accessToken.getScope().equals(VCS.GITHUB_AUTHORIZE_OAUTH_SCOPE))
			throw new OAuthIllegalTokenScopeException(accessToken);
	}
}
```

VcsRepositoryGitHubDao.java

```
package com.autoplag.persistence.vcs.impl.github;

import com.autoplag.config.constant.VCS;
import com.autoplag.persistence.domain.type.AuthorizationProvider;
import com.autoplag.persistence.domain.vcs.AccessToken;
import com.autoplag.persistence.domain.vcs.RepositoryFileInfo;
import com.autoplag.persistence.domain.vcs.RepositoryInfo;
import com.autoplag.persistence.exception.oauth.InvalidVcsUrlException;
import com.autoplag.persistence.exception.oauth.OAuthIllegalTokenException;
import com.autoplag.persistence.vcs.VcsRepositoryDao;
import com.autoplag.persistence.vcs.impl.github.dto.*;

@Repository("vcsRepositoryGitHubDao")
@AllArgsConstructor
public class VcsRepositoryGitHubDao implements VcsRepositoryDao {
	public static final String AUTHORIZATION = "Authorization";
	public static final String ACCEPT = "Accept";
	private final Gson gson;

	@Override
	public boolean checkAccess(AccessToken accessToken, String
repositoryUrl) {
		return Objects.nonNull(getGitHubRepositoryInfo(accessToken,
repositoryUrl));
	}

	@Override
	public RepositoryInfo getSubDirectoryRepositoryInfo(AccessToken
accessToken, String repositoryUrl, String prefixPath, Date lastCommitDate){
		GitHubRepositoryInfo gitHubRepositoryInfo =
getGitHubRepositoryInfo(accessToken, repositoryUrl);
		Optional<GitHubCommit> lastCommitOpt =
getLastCommit(accessToken, gitHubRepositoryInfo, lastCommitDate);
		if (lastCommitOpt.isEmpty()) {
			return createRepositoryInfo(gitHubRepositoryInfo,
List.of(), prefixPath);
		}
		GitHubTree tree = getTreeOfCommit(accessToken,
lastCommitOpt.get());
		return createRepositoryInfo(gitHubRepositoryInfo,
tree.getTree(), prefixPath);
	}
```

```
	@Override
	public String getRawFileContent(AccessToken accessToken,
RepositoryFileInfo fileInfo) {
		return Unirest.get(fileInfo.getUrl())
				.header(AUTHORIZATION,
accessToken.getAccessToken())
				.header(ACCEPT, VCS.GITHUB_API_RAW_ACCEPT_FORMAT)
				.asString()
				.getBody();
	}

	private RepositoryInfo createRepositoryInfo(GitHubRepositoryInfo
gitHubRepositoryInfo,List<GitHubBlob> filesInfo, String prefixPath) {
		List<RepositoryFileInfo> filteredFilesInfo = filesInfo.stream()
				.filter(blob -> blob.getType().equals("blob")
					&& blob.getPath().startsWith(prefixPath))
				.map(blob -> RepositoryFileInfo.builder()
						.path(blob.getPath())
						.url(blob.getUrl())
						.size(blob.getSize()).build())
				.collect(Collectors.toList());

		return RepositoryInfo.builder()
			.authorizationProvider(AuthorizationProvider.GITHUB)
				.name(gitHubRepositoryInfo.getName())
				.apiUrl(gitHubRepositoryInfo.getApiUrl())
				.websiteUrl(gitHubRepositoryInfo.getWebsiteUrl())
				.isPrivate(gitHubRepositoryInfo.isPrivate())
				.prefixPath(prefixPath)
				.filesInfo(filteredFilesInfo)
				.build();
	}

	private GitHubRepositoryInfo getGitHubRepositoryInfo(AccessToken
accessToken, String repositoryUrl) {
		String apiRepositoryUrl = getRepositoryUrl(repositoryUrl);
		return executeGetRequest(
				accessToken,
				apiRepositoryUrl,
				GitHubRepositoryInfo.class);
	}

	private Optional<GitHubCommit> getLastCommit(AccessToken accessToken,
GitHubRepositoryInfo repositoryInfo, Date lastCommitDate) {
		String repositoryCommitsUrl =
getRepositoryCommitsUrl(repositoryInfo.getApiUrl(), lastCommitDate);
		JSONArray jsonArray = Unirest.get(repositoryCommitsUrl)
				.header(AUTHORIZATION,accessToken.getAccessToken())
				.header(ACCEPT, VCS.GITHUB_API_JSON_ACCEPT_FORMAT)
				.asJson()
				.ifFailure(errorHandler(accessToken))
				.getBody()
				.getArray();
		if (jsonArray.length() == 0)
			return Optional.empty();
		try {
			JSONObject jsonObject = jsonArray.getJSONObject(0)
					.getJSONObject("commit");
			return Optional.of(gson.fromJson(jsonObject.toString(),
GitHubCommit.class));
		} catch (JSONException ex) {
			throw new InvalidVcsUrlException(ex);
		}
	}
```

```
      private GitHubTree getTreeOfCommit(AccessToken accessToken,
GitHubCommit commit) {
            return executeGetRequest(
                        accessToken,
                        commit.getTree().getUrl() + "?recursive=true",
                        GitHubTree.class);
      }

      private <T> T executeGetRequest(AccessToken accessToken,
                                    String url,
                                    Class<T> entityType) {
            HttpResponse<T> response = Unirest.get(url)
                        .header(AUTHORIZATION,
accessToken.getAccessToken())
                        .header(ACCEPT, VCS.GITHUB_API_JSON_ACCEPT_FORMAT)
                        .asObject(entityType)
                        .ifFailure(errorHandler(accessToken));

            return response.getBody();
      }

      private <T> Consumer<HttpResponse<T>> errorHandler(AccessToken
accessToken) {
            return response -> {
                  HttpStatus httpStatus =
HttpStatus.valueOf(response.getStatus());
                  if (httpStatus.is2xxSuccessful()) {
                        response.getParsingError().ifPresent(ex -> {
                              throw new InvalidVcsUrlException(ex);
                        });
                  }
                  GitHubErrorResponse errorResponse =
response.mapError(GitHubErrorResponse.class);
                  if (Objects.nonNull(errorResponse)) {
                        errorResponse.setStatus(response.getStatus());

      errorResponse.setStatusText(response.getStatusText());
                        if (httpStatus == HttpStatus.UNAUTHORIZED) {
                              throw new
OAuthIllegalTokenException(errorResponse.toString(), accessToken);
                        }
                        throw new
InvalidVcsUrlException(errorResponse.toString());
                  }
            };
      }

      private String getRepositoryUrl(String repositoryUrl) {
            return
AuthorizationProvider.GITHUB.getApiRepositoryUrl(repositoryUrl);
      }

      private String getRepositoryCommitsUrl(String repositoryUrl, Date
lastCommitDate) {
            repositoryUrl += VCS.GITHUB_API_COMMITS_SUFFIX_ENDPOINT +
"?page=1&per_page=1";
            if (Objects.nonNull(lastCommitDate)) {
                  repositoryUrl += "&until=" + lastCommitDate.toString();
            }
            return repositoryUrl;
      }
}
```

VcsOAuthBitbucketDao.java

```
package com.autoplag.persistence.vcs.impl.bitbucket;
import com.autoplag.config.constant.VCS;
import com.autoplag.persistence.domain.type.AuthorizationProvider;
import com.autoplag.persistence.domain.vcs.AccessToken;
import com.autoplag.persistence.exception.oauth.InvalidVcsUrlException;
import com.autoplag.persistence.exception.oauth.OAuthIllegalTokenException;

@Repository("vcsOAuthBitbucketDao")
public class VcsOAuthBitbucketDao implements VcsOAuthDao {

	@Override
	public String getAuthorizeOAuthUrl(String redirectUrl) {
		String redirectParameter = null;
		if (Objects.nonNull(redirectUrl) && redirectUrl.length() > 0) {
			if
(!redirectUrl.startsWith(VCS.BITBUCKET_AUTHORIZE_OAUTH_CALLBACK_URL)) {
				throw new InvalidVcsUrlException(redirectUrl);
			}
			redirectParameter = "redirect_uri" + "=" + redirectUrl;
		}

		return VCS.BITBUCKET_AUTHORIZE_OAUTH_URL +
				"?" + "client_id" + "=" +
VCS.BITBUCKET_AUTHORIZE_OAUTH_CLIENT_ID +
				"&" + "response_type=code" +
				(Objects.nonNull(redirectParameter) ? "&" +
redirectParameter : "");
	}

	@Override
	public AccessToken getAuthorizeOAuthToken(String code, String
returnedState) {
		AccessToken accessToken =
Unirest.post(VCS.BITBUCKET_AUTHORIZE_OAUTH_TOKEN_URL)

	.field("client_id",VCS.BITBUCKET_AUTHORIZE_OAUTH_CLIENT_ID)
				.field("client_secret",
VCS.BITBUCKET_AUTHORIZE_OAUTH_CLIENT_SECRET)
				.field("grant_type", AUTHORIZATION_CODE)
				.field("code", code)
				.field("redirect_uri", returnedState)
				.asObject(AccessToken.class)
				.getBody();
		return processAccessToken(accessToken);
	}

	@Override
	public AccessToken getRefreshedOAuthToken(AccessToken
expiredAccessToken) {
		AccessToken accessToken =
Unirest.post(VCS.BITBUCKET_AUTHORIZE_OAUTH_TOKEN_URL)
		.field("client_id",VCS.BITBUCKET_AUTHORIZE_OAUTH_CLIENT_ID)
		.field("client_secret",
VCS.BITBUCKET_AUTHORIZE_OAUTH_CLIENT_SECRET)
		.field("grant_type", REFRESH_TOKEN)
		.field("refresh_token",expiredAccessToken.getRefreshToken())
		.asObject(AccessToken.class)
		.getBody();
		return processAccessToken(accessToken);
	}
```

```
	private AccessToken processAccessToken(AccessToken accessToken) {
		accessToken.setAuthorizationProvider(
				AuthorizationProvider.BITBUCKET);
		validateAccessToken(accessToken);
		accessToken.setTokenType(
				capitalizeFirstLetter(accessToken.getTokenType()));
		return accessToken;
	}

	private void validateAccessToken(AccessToken accessToken) {
		if (Objects.isNull(accessToken) ||
				Objects.isNull(accessToken.getAccessToken()) ||
				Objects.isNull(accessToken.getTokenType())) {
			throw new OAuthIllegalTokenException(accessToken);
		}
		if (Objects.isNull(accessToken.getScope()) ||
	!accessToken.getScope().equals(VCS.BITBUCKET_AUTHORIZE_OAUTH_SCOPE)){
			throw new OAuthIllegalTokenScopeException(accessToken);
		}
	}

	private String capitalizeFirstLetter(String str) {
		return str.substring(0, 1).toUpperCase() + str.substring(1);
	}
}
```

VcsRepositoryBitbucketDao.java

```
package com.autoplag.persistence.vcs.impl.bitbucket;

import com.autoplag.config.constant.VCS;
import com.autoplag.persistence.domain.type.AuthorizationProvider;
import com.autoplag.persistence.domain.vcs.*;
import com.autoplag.persistence.exception.oauth.*;
import com.autoplag.persistence.vcs.VcsRepositoryDao;
import com.autoplag.persistence.vcs.impl.bitbucket.dto.*;

@Repository("vcsRepositoryBitbucketDao")
@AllArgsConstructor
public class VcsRepositoryBitbucketDao implements VcsRepositoryDao {
	public static final String AUTHORIZATION = "Authorization";
	private final static String REPOSITORY_INFO_FIELDS_PARAM =
"?fields=name,is_private,links.self,links.source,links.html";
	private final static String COMMITS_INFO_FIELDS_PARAM =
"?pagelen=50&fields=values.date,values.hash,next";
	private final static String BLOB_INFO_FIELDS_PARAM =
"?pagelen=50&fields=values.path,values.type,values.links,values.size,next";
	private final VcsOAuthBitbucketDao vcsOAuthBitbucketDao;

	@Override
	public boolean checkAccess(AccessToken accessToken, String
repositoryUrl) {
		try {
			return
Objects.nonNull(getBitbucketRepositoryInfo(accessToken, repositoryUrl));
		} catch (OAuthIllegalTokenException ex) {
			accessToken =
vcsOAuthBitbucketDao.getRefreshedOAuthToken(accessToken);
			return
Objects.nonNull(getBitbucketRepositoryInfo(accessToken, repositoryUrl));
		}
	}
```

```
	@Override
	public RepositoryInfo getSubDirectoryRepositoryInfo(AccessToken
accessToken, String repositoryUrl, String prefixPath, Date lastCommitDate){
		BitbucketRepositoryInfo bitbucketRepositoryInfo =
getBitbucketRepositoryInfo(accessToken, repositoryUrl);
		Optional<BitbucketCommit> lastCommitOpt = getLastCommit(
				accessToken,
				bitbucketRepositoryInfo,
				prefixPath,
				lastCommitDate);
		if (lastCommitOpt.isEmpty()) {
			return createRepositoryInfo(bitbucketRepositoryInfo,
List.of(), prefixPath);
		}
		List<BitbucketBlob> filesInfo = getFilesInfo(
				accessToken,
				bitbucketRepositoryInfo,
				lastCommitOpt.get(),
				prefixPath);
		return createRepositoryInfo(bitbucketRepositoryInfo, filesInfo,
prefixPath);
	}

	@Override
	public String getRawFileContent(AccessToken accessToken,
RepositoryFileInfo fileInfo) {
		return Unirest.get(fileInfo.getUrl())
				.header(AUTHORIZATION,accessToken.getAccessToken())
				.asString()
				.getBody();
	}

	private RepositoryInfo createRepositoryInfo(BitbucketRepositoryInfo
bitbucketRepositoryInfo, List<BitbucketBlob> filesInfo, String prefixPath){
		List<RepositoryFileInfo> filteredFilesInfo = filesInfo.stream()
				.filter(blob ->
					blob.getType().equals("commit_file")
					&& blob.getPath().startsWith(prefixPath))
				.map(blob -> RepositoryFileInfo.builder()
						.path(blob.getPath())
						.url(blob.getSelfUrl())
						.size(blob.getSize())
						.build())
				.collect(Collectors.toList());
		return RepositoryInfo.builder()
			.authorizationProvider(AuthorizationProvider.BITBUCKET)
				.name(bitbucketRepositoryInfo.getName())
				.apiUrl(bitbucketRepositoryInfo.getApiUrl())
			.websiteUrl(bitbucketRepositoryInfo.getWebsiteUrl())
				.isPrivate(bitbucketRepositoryInfo.isPrivate())
				.prefixPath(prefixPath)
				.filesInfo(filteredFilesInfo)
				.build();
	}

	private BitbucketRepositoryInfo
getBitbucketRepositoryInfo(AccessToken accessToken, String repositoryUrl) {
		String apiRepositoryUrl = getRepositoryUrl(repositoryUrl);
		return Unirest.get(apiRepositoryUrl)
				.header(AUTHORIZATION,accessToken.getAccessToken())
				.asObject(BitbucketRepositoryInfo.class)
				.ifFailure(errorHandler(accessToken))
				.getBody();
	}
```

```
	private Optional<BitbucketCommit> getLastCommit(AccessToken
accessToken, BitbucketRepositoryInfo repositoryInfo, String prefixPath,
Date lastCommitDate) {
		String repositoryCommitsUrl =
getRepositoryCommitsUrl(repositoryInfo.getApiUrl(), prefixPath);
		BitbucketPagination<BitbucketCommit> commits =
BitbucketPagination.<BitbucketCommit>builder()
				.next(repositoryCommitsUrl)
				.build();
		Optional<BitbucketCommit> lastCommit = Optional.empty();
		while (Objects.nonNull(commits.getNext())) {
			commits = Unirest.get(commits.getNext())
					.header(AUTHORIZATION,
accessToken.getAccessToken())
					.asObject(new
GenericType<BitbucketPagination<BitbucketCommit>>() {})
					.ifFailure(errorHandler(accessToken))
					.getBody();
			lastCommit = commits.getValues().stream()
					.filter(c ->
c.getDate().compareTo(lastCommitDate) <= 0)
					.max(Comparator.comparing(
BitbucketCommit::getDate));
		}
		return lastCommit;
	}

	private List<BitbucketBlob> getFilesInfo(AccessToken accessToken,
BitbucketRepositoryInfo bitbucketRepositoryInfo, BitbucketCommit commit,
String prefixPath) {
		String repositorySrcUrl =
bitbucketRepositoryInfo.getSourceUrl() + "/" + commit.getHash() + "/" +
prefixPath + BLOB_INFO_FIELDS_PARAM;
		return getSubDirFilesInfo(accessToken, repositorySrcUrl);
	}

	private List<BitbucketBlob> getSubDirFilesInfo(AccessToken
accessToken, String srcUrl) {
		BitbucketPagination<BitbucketBlob> filesInfoPage =
BitbucketPagination.<BitbucketBlob>builder()
				.next(srcUrl)
				.build();
		List<BitbucketBlob> filesInfo = new ArrayList<>();
		while (Objects.nonNull(filesInfoPage.getNext())) {
			filesInfoPage = Unirest.get(filesInfoPage.getNext())
					.header(AUTHORIZATION,
accessToken.getAccessToken())
					.asObject(new
GenericType<BitbucketPagination<BitbucketBlob>>() {})
					.ifFailure(errorHandler(accessToken))
					.getBody();
			filesInfo.addAll(filesInfoPage.getValues());
		}
		List<BitbucketBlob> toReturn = new ArrayList<>(filesInfo);
		for (BitbucketBlob blob : filesInfo) {
			if (blob.getType().equals("commit_directory")) {
				List<BitbucketBlob> subDirFilesInfo =
						getSubDirFilesInfo(accessToken,
blob.getSelfUrl() + BLOB_INFO_FIELDS_PARAM);
				toReturn.addAll(subDirFilesInfo);
			}
		}
		return toReturn;
	}
```

```
      private <T> Consumer<HttpResponse<T>> errorHandler(AccessToken
accessToken) {
            return response -> {
                  HttpStatus httpStatus =
HttpStatus.valueOf(response.getStatus());
                  if (httpStatus.is2xxSuccessful())
                        response.getParsingError().ifPresent(ex -> {
                              throw new InvalidVcsUrlException(ex);
                        });
                  if (httpStatus == HttpStatus.FORBIDDEN) {
                        BitbucketErrorResponse errorResponse = new
BitbucketErrorResponse();
                        errorResponse.setStatus(response.getStatus());
                        errorResponse.setStatusText(
response.getStatusText());
                        errorResponse.setError("Access denied. You must
have write or admin access.");
throw new OAuthIllegalTokenException(errorResponse.toString(),accessToken);
                  }
                  BitbucketErrorResponse errorResponse =
response.mapError(BitbucketErrorResponse.class);
                  if (Objects.nonNull(errorResponse)) {
                        errorResponse.setStatus(response.getStatus());
                        errorResponse.setStatusText(
response.getStatusText());
                        if (httpStatus == HttpStatus.UNAUTHORIZED)
            throw new OAuthIllegalTokenException(errorResponse.toString(),
accessToken);
            throw new InvalidVcsUrlException(errorResponse.toString());
                  }
            };
      }

      private String getRepositoryUrl(String repositoryUrl) {
            return AuthorizationProvider.BITBUCKET.getApiRepositoryUrl(
repositoryUrl)+ REPOSITORY_INFO_FIELDS_PARAM;
      }
      private String getRepositoryCommitsUrl(String repositoryUrl, String
prefixPath) {
            repositoryUrl += VCS.GITHUB_API_COMMITS_SUFFIX_ENDPOINT +
COMMITS_INFO_FIELDS_PARAM;
            if (Objects.nonNull(prefixPath) && !prefixPath.isBlank()) {
                  repositoryUrl += "&path=" + prefixPath;
            }
            return repositoryUrl;
      }
}
```

SoftwarePlagDetectionToolImpl.java

```
package com.autoplag.persistence.plagiarism.jplag;

import com.autoplag.config.constant.JPlag;
import com.autoplag.config.constant.Paths;
import com.autoplag.persistence.domain.PlagDetectionResult;
import com.autoplag.persistence.domain.PlagDetectionSettings;
import com.autoplag.persistence.plagiarism.SoftwarePlagDetectionTool;

@Slf4j
@Repository
public class SoftwarePlagDetectionToolImpl implements
SoftwarePlagDetectionTool {
```

```
	public static final String INDEX_FILE = "index.html";
	public static final String BAD_BASECODE_ERROR = "Error: Bad basecode
submission";

	@Override
	public PlagDetectionResult generateHtmlResult(PlagDetectionSettings
settings) {
		if (!validateSettings(settings)) {
			return PlagDetectionResult.failed("Invalid settings");
		}
		String jplagLog = null;
		ProcessBuilder processBuilder =
configureJplagProcess(settings);
		boolean isSuccessful = false;
		try {
			Process process;
			if (settings.getSaveLog()) {
				process = processBuilder.start();
				jplagLog = getJPlagLog(process.getInputStream());
			} else {
				process = processBuilder.inheritIO().start();
			}
			isSuccessful = process.waitFor() == 0;
		} catch (IOException | InterruptedException ex) {
			log.error(ex.toString());
		}
		PlagDetectionResult result = PlagDetectionResult.builder()
				.date(new Date())
				.isSuccessful(isSuccessful)
				.log(jplagLog)
				.build();
		if (isSuccessful) {
			result.setResultMessage("Plagiarism detection tool
successfully done analysis!");
			result.setResultPath(
					generateResultPath(settings.getResultPath()));
		} else {
			if (Objects.nonNull(jplagLog) &&
jplagLog.contains(BAD_BASECODE_ERROR)) {
				result.setResultMessage("Archive with BASECODE must
contain files with allowed format " +
						"for the selected programming
language!");
			} else {
				result.setResultMessage("Plagiarism detection tool
was interrupted (see logs)!");
			}
		}
		return result;
	}

	private ProcessBuilder configureJplagProcess(PlagDetectionSettings
setting) {
		ProcessBuilder builder = new ProcessBuilder();
		List<String> commands = new LinkedList<>();
		// execute jar file
		commands.add(JPlag.JAVA_PATH);
		commands.add("-jar");
		commands.add(JPlag.JPlAG_PATH);
		// data directory path
		commands.add("-s");
		commands.add(setting.getDataPath());
		// result directory path
		commands.add("-r");
```

```
            commands.add(setting.getResultPath());
            // programming language type
            commands.add("-l");
            commands.add(setting.getProgrammingLanguage().getName());
            // sensitivity of the comparison
            if (Objects.nonNull(setting.getComparisonSensitivity())) {
                  commands.add("-t");

      commands.add(setting.getComparisonSensitivity().toString());
            }
            // min similarity percent
            if (Objects.nonNull(setting.getMinimumSimilarityPercent())) {
                  commands.add("-m");

      commands.add(setting.getMinimumSimilarityPercent().toString() + "%");
            }
            // base code directory path
            if (Objects.nonNull(setting.getBaseCodePath())) {
                  commands.add("-bc");
                  commands.add(JPlag.BASE_CODE_DIRECTORY);
            }

            return builder.command(commands);
      }

      private boolean validateSettings(PlagDetectionSettings setting) {
            return Objects.nonNull(setting) &&
                        Objects.nonNull(setting.getDataPath()) &&
                        Objects.nonNull(setting.getResultPath()) &&
                        Objects.nonNull(setting.getProgrammingLanguage());
      }

      private String generateResultPath(String settingResultPathStr) {
            Path settingResultPath = Path.of(settingResultPathStr);
            Path staticFolderPath = Path.of(Paths.STATIC_FOLDER);
            String urlPath = staticFolderPath.relativize(settingResultPath)
                        .toString()
                        .replace("\\", "/");
            return Paths.URL_HOST + "/" + urlPath + "/" + INDEX_FILE;
      }

      private String getJPlagLog(InputStream stream) throws IOException {
            byte[] buffer = new byte[1024];
            int length;
            ByteArrayOutputStream result = new ByteArrayOutputStream();
            while ((length = stream.read(buffer)) != -1) {
                  result.write(buffer, 0, length);
            }
            return
removeSensitiveData(result.toString(StandardCharsets.UTF_8));
      }

      private String removeSensitiveData(String data) {
            int startIndex = data.indexOf("initialize ok");
            int endIndex = data.indexOf("Writing results to:");
            if (startIndex == -1) {
                  return data;
            }
            if (endIndex == -1) {
                  return data.substring(startIndex);
            }
            return data.substring(startIndex, endIndex);
      }
}
```

AutomatedPlagiarismDetectionScheduler.java

```
package com.autoplag.service.cronjob;
import com.autoplag.persistence.domain.TaskGroup;
import com.autoplag.service.PlagDetectionService;
import com.autoplag.service.TaskGroupService;

@Component
@AllArgsConstructor
public class AutomatedPlagiarismDetectionScheduler {
	private final PlagDetectionService plagDetectionService;
	private final TaskGroupService taskGroupService;
	private static final int SCHEDULER_TIMEOUT_IN_MS = 30000;

	@Scheduled(fixedRate = SCHEDULER_TIMEOUT_IN_MS)
	public void scheduleTaskWithFixedRate() {
		List<TaskGroup> taskGroups =
taskGroupService.getAllExpiredTaskGroupsWithPendingStatus();
		if (!taskGroups.isEmpty()) {
			new AutomatedPlagiarismDetectionJob(taskGroupService,
plagDetectionService, taskGroups)
				.start();
		}
	}
}
```

AutomatedPlagiarismDetectionJob.java

```
package com.autoplag.service.cronjob;

import com.autoplag.config.constant.Detection;
import com.autoplag.persistence.domain.PlagDetectionResult;
import com.autoplag.persistence.domain.TaskGroup;
import com.autoplag.persistence.domain.type.PlagDetectionStatus;
import com.autoplag.service.PlagDetectionService;
import com.autoplag.service.TaskGroupService;

@Slf4j
public class AutomatedPlagiarismDetectionJob extends Thread {
	private final List<TaskGroup> taskGroups;
	private final PlagDetectionService plagDetectionService;
	private final TaskGroupService taskGroupService;

	public AutomatedPlagiarismDetectionJob(TaskGroupService
taskGroupService, PlagDetectionService plagiarismDetectionService,
List<TaskGroup> taskGroups) {
		super("AutomatedPlagiarismDetectionJob with taskGroups.size=" +
taskGroups.size());
		this.taskGroupService = taskGroupService;
		this.plagDetectionService = plagiarismDetectionService;
		this.taskGroups = taskGroups;
	}
	@Override
	public void run() {
		log.debug("Start ThreadPoolExecutor for process {} taskGroups",
taskGroups.size());
		ThreadPoolExecutor threadPool = (ThreadPoolExecutor)
Executors.newFixedThreadPool(Detection.DEFAULT_POOL_SIZE);
		List<Future<PlagDetectionResult>> results = new
ArrayList<>(taskGroups.size());
		for (TaskGroup taskGroup : taskGroups) {
```

```
                  PlagDetectionResultCallable callable = new
PlagDetectionResultCallable(  taskGroupService, plagDetectionService,
taskGroup);
                  results.add(threadPool.submit(callable));
            }
            threadPool.shutdown();
            try {
                  if
(!threadPool.awaitTermination(Detection.EXECUTION_TIMEOUT_IN_MINUTES,
TimeUnit.MINUTES)) {
                        threadPool.shutdownNow();
                        if (!threadPool.awaitTermination(
Detection.TASK_CANCEL_RESPONSE_TIMEOUT_IN_SECONDS, TimeUnit.SECONDS)) {
                              log.error("Thread pool did not terminate");
                        }
                  }
                  for (Future<PlagDetectionResult> result : results) {
                        if (!result.isDone()) {
                              result.cancel(true);
                        }
                        try {
                              log.info("Result of plagiarism detection: {}",
result.get().toString());
                        } catch (Exception ex) {
                              log.error("Plagiarism detection failed!",ex);
                        }
                  }
            } catch (InterruptedException ex) {
                  log.error("AutomatedPlagiarismDetectionJob failed", ex);
                  checkTaskGroupStatus(taskGroups);
                  threadPool.shutdownNow();
                  Thread.currentThread().interrupt();
            }
      }
      private void checkTaskGroupStatus(List<TaskGroup> taskGroups) {
            for (TaskGroup taskGroup : taskGroups) {
                  Optional<TaskGroup> taskGroupOpt =
      taskGroupService.getTaskGroupByIdAndStatus(taskGroup.getId(),
PlagDetectionStatus.IN_PROCESS);
                  if (taskGroupOpt.isPresent()) {
                        TaskGroup completedTaskGroup = taskGroupOpt.get();
                        PlagDetectionResult result =
PlagDetectionResult.builder()
                                    .isSuccessful(false)
                                    .date(new Date())
                                    .resultMessage("Failed detect plagiarism
for task!")
                                    .build();
                        completedTaskGroup.setPlagDetectionResult(result);
      completedTaskGroup.setPlagDetectionStatus(PlagDetectionStatus.FAILED)
;
                        taskGroupService.saveTaskGroup(taskGroup);
                  }
            }
      }
      static class PlagDetectionResultCallable implements
Callable<PlagDetectionResult> {
            private final TaskGroupService taskGroupService;
            private final PlagDetectionService plagiarismDetectionService;
            private final TaskGroup taskGroup;
            public PlagDetectionResultCallable(TaskGroupService
taskGroupService, PlagDetectionService plagiarismDetectionService,
TaskGroup taskGroup) {
                  this.taskGroupService = taskGroupService;
```

```
                  this.plagiarismDetectionService =
plagiarismDetectionService;
                  this.taskGroup = taskGroup;
            }
            @Override
            public PlagDetectionResult call() {
                  try {
      taskGroup.setPlagDetectionStatus(PlagDetectionStatus.IN_PROCESS);
                        taskGroupService.saveTaskGroup(taskGroup);
                        PlagDetectionResult result =
plagiarismDetectionService.processForTaskGroup(taskGroup);
                        taskGroup.setPlagDetectionResult(result);
                        taskGroup.setPlagDetectionStatus(
                                    result.getIsSuccessful()
                                                ? PlagDetectionStatus.DONE
                                                : PlagDetectionStatus.FAILED
                        );
                        return result;
                  } catch (Exception ex) {
                        log.error("", ex);
                        PlagDetectionResult result =
PlagDetectionResult.failed("Failed detect plagiarism for task!");

      taskGroup.setPlagDetectionStatus(PlagDetectionStatus.FAILED);
                        taskGroup.setPlagDetectionResult(result);
                        throw ex;
                  } finally {
                        taskGroupService.saveTaskGroup(taskGroup);
                  }
            }
      }
}
```

PlagDetectionServiceImpl.java

```
package com.autoplag.service.impl;

import com.autoplag.persistence.dao.StudentGroupDao;
import com.autoplag.persistence.domain.*;
import com.autoplag.persistence.domain.type.DetectionType;
import com.autoplag.persistence.domain.vcs.AccessToken;
import com.autoplag.persistence.domain.vcs.RepositoryInfo;
import com.autoplag.persistence.exception.oauth.InvalidVcsUrlException;
import com.autoplag.persistence.exception.oauth.OAuthIllegalTokenException;
import com.autoplag.persistence.plagiarism.SoftwarePlagDetectionTool;
import com.autoplag.service.PlagDetectionResultService;
import com.autoplag.service.PlagDetectionService;
import com.autoplag.service.exception.FailedToWriteToFileSystemException;
import com.autoplag.service.vcs.VcsDownloadService;
import com.autoplag.util.FileSystemWriter;

@Service
@AllArgsConstructor
@Slf4j
public class PlagDetectionServiceImpl implements PlagDetectionService {
      private final VcsDownloadService vcsDownloadService;
      private final SoftwarePlagDetectionTool softwarePlagDetectionTool;
      private final StudentGroupDao studentGroupDao;
      private final PlagDetectionResultService plagDetectionResultService;
      private final FileSystemWriter fileSystemWriter;
```

```
	@Transactional
	@Override
	public PlagDetectionResult processForTaskGroup(TaskGroup taskGroup) {
		log.debug("Attempt to process taskGroup ({})",
taskGroup.getId());
		User user = taskGroup.getGroup().getCourse().getCreator();
		Task task = taskGroup.getTask();
		PlagDetectionSettings setting =
taskGroup.getPlagDetectionSettings();
		Set<StudentGroup> studentGroups = loadStudents(taskGroup);
		boolean wasReloaded =
Objects.nonNull(setting.getBaseCodePath())?
fileSystemWriter.reloadDirectory(setting.getDataPath(),
setting.getBaseCodePath()):
fileSystemWriter.reloadDirectory(setting.getDataPath());
		if (!wasReloaded)
			return PlagDetectionResult.failed("Server error!");
		Set<ResultStudent> resultStudents = new HashSet<>();
		for (StudentGroup studentGroup : studentGroups) {
			String errorLogMessage = null;
			String studentFullName =
studentGroup.getStudent().getFullName();
			String repositoryDataPath =
Path.of(setting.getDataPath(), studentFullName).toString();
			String vcsRepositoryUrl =
studentGroup.getVcsRepositoryUrl();
			AccessToken accessToken =
user.getAccessToken(vcsRepositoryUrl);
			try {
				RepositoryInfo repositoryInfo =
vcsDownloadService.downloadRepository(
								accessToken,
								vcsRepositoryUrl,
								task.getRepositoryPrefixPath(),
								taskGroup.getExpiryDate());
				if (repositoryInfo.isEmptyRepository()) {
					errorLogMessage = "Directory \"" +
task.getRepositoryPrefixPath() + "\" doesn`t exist!";
				} else {
	vcsDownloadService.downloadAndSaveRawContentOfFiles(
									accessToken,
									repositoryInfo,
									repositoryDataPath);
				}
			} catch (InvalidVcsUrlException ex) {
				errorLogMessage = "Invalid VCS-URL: " +
studentGroup.getVcsRepositoryUrl();
			} catch (OAuthIllegalTokenException ex) {
				errorLogMessage = "No access to student repository:
" + studentGroup.getVcsRepositoryUrl();
			} catch (FailedToWriteToFileSystemException ex) {
				errorLogMessage = "Failed to save student
repository: " + studentGroup.getVcsRepositoryUrl();
			}
			if (Objects.nonNull(errorLogMessage)) {
				resultStudents.add(ResultStudent.builder()
								.student(studentGroup.getStudent())
								.logMessage(errorLogMessage)
								.build());
			}
		}
		PlagDetectionResult result =
softwarePlagDetectionTool.generateHtmlResult(setting);
```

```
            result =
plagDetectionResultService.savePlagDetectionResult(result);
            if (!resultStudents.isEmpty()) {
                  result.addResultStudents(resultStudents);
            }
            return
plagDetectionResultService.savePlagDetectionResult(result);
      }

      @Override
      public PlagDetectionResult processForSingleTask(PlagDetectionSettings
setting) {
            return softwarePlagDetectionTool.generateHtmlResult(setting);
      }
      private Set<StudentGroup> loadStudents(TaskGroup taskGroup) {
            DetectionType detectionType =
taskGroup.getPlagDetectionSettings().getDetectionType();
            if (DetectionType.GROUP == detectionType) {
                  return taskGroup.getGroup().getStudentGroups();
            } else if (DetectionType.COURSE == detectionType) {
                  return
studentGroupDao.findAllWhoHaveTask(taskGroup.getTask());
            }
            throw new IllegalArgumentException(detectionType.toString());
      }
}
```

VcsDownloadServiceImpl.java

```
package com.autoplag.service.vcs.impl;

import com.autoplag.config.constant.VCS;
import com.autoplag.persistence.domain.type.AuthorizationProvider;
import com.autoplag.persistence.domain.vcs.AccessToken;
import com.autoplag.persistence.domain.vcs.RepositoryFileInfo;
import com.autoplag.persistence.domain.vcs.RepositoryInfo;
import com.autoplag.persistence.exception.oauth.InvalidVcsUrlException;
import com.autoplag.persistence.exception.oauth.OAuthIllegalTokenException;
import com.autoplag.persistence.exception.oauth.VCSException;
import com.autoplag.persistence.vcs.VcsRepositoryDao;
import com.autoplag.service.exception.FailedToWriteToFileSystemException;
import com.autoplag.service.vcs.VcsDownloadService;
import com.autoplag.service.vcs.VcsOAuthService;
import com.autoplag.util.FileSystemWriter;

@Service
@AllArgsConstructor
@Slf4j
public class VcsDownloadServiceImpl implements VcsDownloadService {
      public static final String ILLEGAL_AUTHORIZATION_PROVIDER = "Such
authorization provider doesn't exist!";
      private final VcsRepositoryDao vcsRepositoryBitbucketDao;
      private final VcsRepositoryDao vcsRepositoryGitHubDao;
      private final VcsOAuthService vcsOAuthBitbucketService;
      private final VcsOAuthService vcsOAuthGitHubService;
      private final FileSystemWriter fileSystemWriter;
      private final Pattern rootRepositoryPattern = Pattern.compile("(" +
VCS.BITBUCKET_WEBSITE_REPOSITORY_PREFIX_ENDPOINT + "(/[^\\s/]+){2}|" +
                  VCS.GITHUB_WEBSITE_REPOSITORY_PREFIX_ENDPOINT +
"(/[^\\s/]+){2})");
```

```
	@Override
	public String getRootRepositoryUrl(String repositoryUrl) {
		Matcher matcher = rootRepositoryPattern.matcher(repositoryUrl);
		if (!matcher.find()) {
			throw new InvalidVcsUrlException("Invalid url for VCS
services (GitHub or Bitbucket)");
		}
		return matcher.group();
	}

	@Override
	public void checkAccessToRepository(AccessToken userAccessToken,
String repositoryUrl) {
		log.debug("Check access to repository ({})", repositoryUrl);
		AuthorizationProvider authorizationProvider =
AuthorizationProvider.recognizeFromUrl(repositoryUrl);
		try {
			checkAccessToken(userAccessToken, authorizationProvider);
			VcsRepositoryDao vcsRepositoryDao =
getVcsRepositoryDaoForAuthorizationProvider(authorizationProvider);
			vcsRepositoryDao.checkAccess(userAccessToken,
repositoryUrl);
		} catch (VCSException ex) {
			log.debug(ex.toString());
			throw new VCSException("Репозиторій не існує або Ви не
отримали доступ до нього!", ex);
		}
	}

	@Override
	public RepositoryInfo downloadRepository(AccessToken userAccessToken,
String repositoryUrl, String prefixPath, Date lastDateCommit) {
		log.debug("Attempt to download repository ({})",
repositoryUrl);
		AuthorizationProvider authorizationProvider =
AuthorizationProvider.recognizeFromUrl(repositoryUrl);
		checkAccessToken(userAccessToken, authorizationProvider);
		VcsRepositoryDao vcsRepositoryDao =
getVcsRepositoryDaoForAuthorizationProvider(authorizationProvider);
		try {
			RepositoryInfo repositoryInfo =
vcsRepositoryDao.getSubDirectoryRepositoryInfo(
					userAccessToken,
					repositoryUrl,
					prefixPath,
					lastDateCommit);
			log.debug("Attempt to download SUCCESSFUL - repository
({}) ", repositoryUrl);
			return repositoryInfo;
		} catch (OAuthIllegalTokenException ex) {
			log.debug(ex.toString());
			AccessToken newAccessToken =
tryToRefreshToken(userAccessToken);
			RepositoryInfo repositoryInfo =
vcsRepositoryDao.getSubDirectoryRepositoryInfo(
					newAccessToken,
					repositoryUrl,
					prefixPath,
					lastDateCommit);
			log.debug("Attempt to download SUCCESSFUL - repository
({}) ", repositoryUrl);
			return repositoryInfo;
		}
	}
```

```
	@Override
	public boolean downloadAndSaveRawContentOfFiles(AccessToken
userAccessToken, RepositoryInfo repositoryInfo, String repositoryDataPath){
		checkAccessToken(userAccessToken,
repositoryInfo.getAuthorizationProvider());
		if (repositoryInfo.isEmptyRepository())
			return false;
		VcsRepositoryDao vcsRepositoryDao =
getVcsRepositoryDaoForAuthorizationProvider(
				repositoryInfo.getAuthorizationProvider());
	fileSystemWriter.deleteDirectory(repositoryDataPath);
	for (RepositoryFileInfo fileInfo : repositoryInfo.getFilesInfo()) {
			String fileData;
			try {
				fileData =
vcsRepositoryDao.getRawFileContent(userAccessToken, fileInfo);
			} catch (OAuthIllegalTokenException ex) {
			userAccessToken = tryToRefreshToken(userAccessToken);
			fileData =
vcsRepositoryDao.getRawFileContent(userAccessToken, fileInfo);
			}
			try {
				fileSystemWriter.writeStringData(
					Path.of(repositoryDataPath,
fileInfo.getPath()).toString(),fileData);
			} catch (IllegalStateException ex) {
				throw new FailedToWriteToFileSystemException(ex);
			}
		}
		return true;
	}

	private VcsRepositoryDao getVcsRepositoryDaoForAuthorizationProvider(
			AuthorizationProvider authorizationProvider) {
		if (authorizationProvider == AuthorizationProvider.GITHUB) {
			return vcsRepositoryGitHubDao;
		} else if (authorizationProvider ==
AuthorizationProvider.BITBUCKET) {
			return vcsRepositoryBitbucketDao;
		}
	throw new IllegalStateException(ILLEGAL_AUTHORIZATION_PROVIDER);
	}

	private AccessToken tryToRefreshToken(AccessToken accessToken) {
		if (AuthorizationProvider.BITBUCKET ==
accessToken.getAuthorizationProvider()) {
	return vcsOAuthBitbucketService.getRefreshedOAuthToken(accessToken);
		} else if (AuthorizationProvider.GITHUB ==
accessToken.getAuthorizationProvider()) {
	return vcsOAuthGitHubService.getRefreshedOAuthToken(accessToken);
		} else
			throw new OAuthIllegalTokenException(accessToken);
	}

	private void checkAccessToken(AccessToken accessToken,
AuthorizationProvider authorizationProvider) {
		if (Objects.isNull(accessToken) ||authorizationProvider !=
accessToken.getAuthorizationProvider()) {
			throw new OAuthIllegalTokenException("Invalid
authorization for " + authorizationProvider + " service",accessToken);
		}
	}
}
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ПЛАГІАТУ В НАВЧАЛЬНИХ ПРОГРАМНИХ ПРОЄКТАХ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СВИНАРЧУК

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах, серверна частина якого написана мовою програмування Java з використанням веб-фреймворку Spring Framework, а клієнтська частина – мовою програмування JavaScript із використанням бібліотеки для розробки користувацького інтерфейсу React.

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних елементів:

1. Працездатність функцій реєстрації та автентифікації.
2. Попередження неавторизованого доступу.
3. Функціональна працездатність елементів сторінок веб-додатка.
4. Доступ до бази даних через основну серверну частину.
5. Коректна робота функціоналу створення курсів, груп, завдань на перевірку плагіату програмних проєктів.
6. Робота сервера з модулем, що відповідає за інтеграцію інструмента для виявлення програмного плагіату в веб-додаток.
7. Робота сервера з модулем, що відповідає за автоматизацію завантаження програмних проєктів із веб-сервісів для хостингу проєктів на базі систем керування версіями.
8. Забезпечення коректної обробки REST запитів та відповідей на них.
9. Відповідність дизайну вимогам програмного забезпечення.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується відповідно до техніки сірого ящику або Gray Box Testing.

Існують ще два схожі підходи до тестування:

1. Black Box Testing – тестування відбувається без будь-яких знань внутрішньої структури системи. Тестувальник має доступ до програми тільки через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, через які інший комп'ютера або інший процес може підключитися до системи.
2. White Box Testing – тестування здійснюється на основі внутрішнього функціонування і логіки роботи коду. Тестувальнику відомо все про систему, щоб точно розпізнати частину коду, яка має помилку.

Gray Box Testing є середнім між двома вищеописаними підходами тестування. Тестувальник в даному разі має обмежені знання про внутрішні деталі системи й перевіряє як вихідний код, так і сам програмний продукт на відповідність існуючим програмним функціональним та нефункціональним вимогам.

Використовуються такі методи тестування:

1. Функціональне тестування.
2. Ручне тестування інтерфейсу.
3. Тестування продуктивності веб-додатка.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Повний процес тестування проходить у такому порядку:

1. Статичне тестування коду.
2. Динамічне ручне тестування полів вводу на граничні та неможливі значення.
3. Динамічне ручне тестування на відповідність функціональним вимогам.
4. Тестування інтерфейсу при різній роздільній здатності екрану.

5. Тестування інтерфейсу в різних браузерах: Chrome, Safari, Firefox.
6. Тестування при конкурентному навантажені на систему з боку декількох користувачів.
7. Тестування стабільності роботи при різних зовнішніх умовах (максимальному навантаженні на оперативну пам’ять та процесор, низька швидкість мережі Інтернет).

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ АВТОМАТИЗАЦІЇ ВИЯВЛЕННЯ ПЛАГІАТУ В НАВЧАЛЬНИХ ПРОГРАМНИХ ПРОЄКТАХ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Максим СВИНАРЧУК

2020

ЗМІСТ

1. СТРУКТУРА ВЕБ-ДОДАТКА

Веб-додаток для автоматизації виявлення плагіату в навчальних програмних проєктах складається лише з динамічних веб-сторінок. Головна мова інтерфейсу – українська. Англійська мова використовується для показу результатів виявлення плагіату системою JPlag.

Веб-сторінки системи: головна сторінка, сторінка реєстрації, сторінка авторизації, сторінка разової перевірки, сторінка профілю, сторінка всіх студентів, сторінка всіх курсів, сторінка обраного курсу, сторінка всіх завдань обраного курсу, сторінка обраної групи, сторінка обраного завдання для конкретної групи.

Всі веб-сторінки використовують один шаблон (рис. 1.1): вгорі розміщується навігаційна панель, внизу нижній колонтитул, також для авторизованих користувачів з'являється додаткова бокова панель для керування системою. Вміст даних елементів змінюється в залежності від того, авторизований користувач чи ні та на якій веб-сторінці він знаходиться.

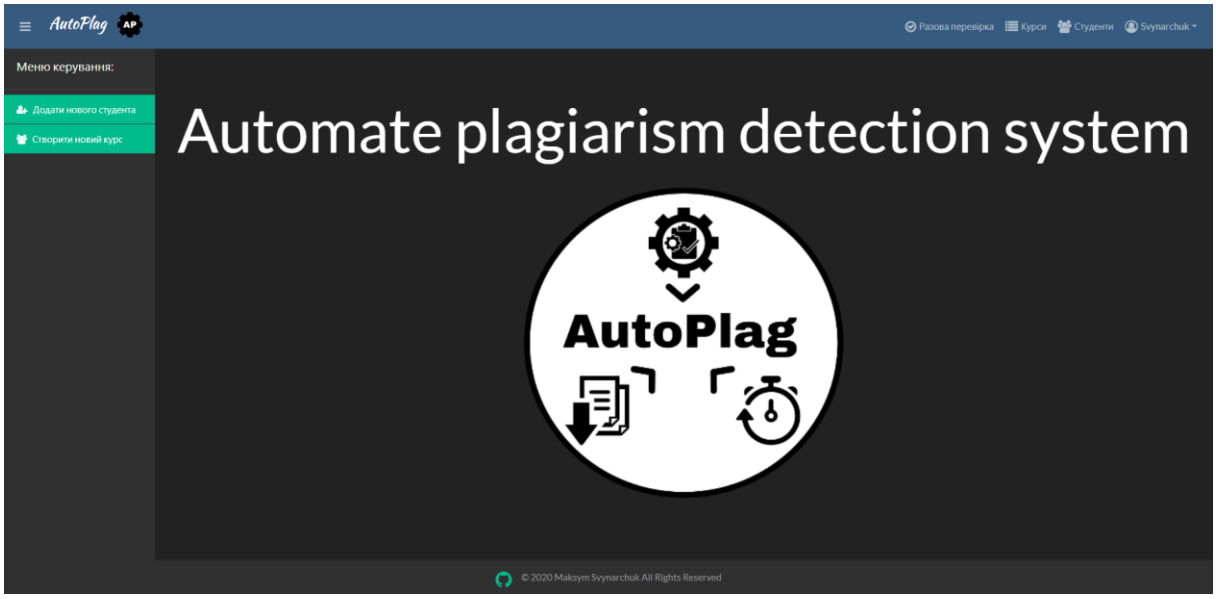


Рис. 1.1. Головна сторінка веб-додатка для авторизованого користувача

2. ПРОЦЕДУРА РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ

Для повноцінного використання можливостей веб-додатка користувач повинен пройти процедуру реєстрації та авторизації. Форма реєстрації має наступні поля для вводу: електронна пошта користувача, ім’я та прізвища (дозволяються тільки букви), пароль (не менше 6 символів). Електронна пошта унікальна для кожного користувача. У випадку співпадіння з’явиться помилка реєстрації (рис. 2.1).

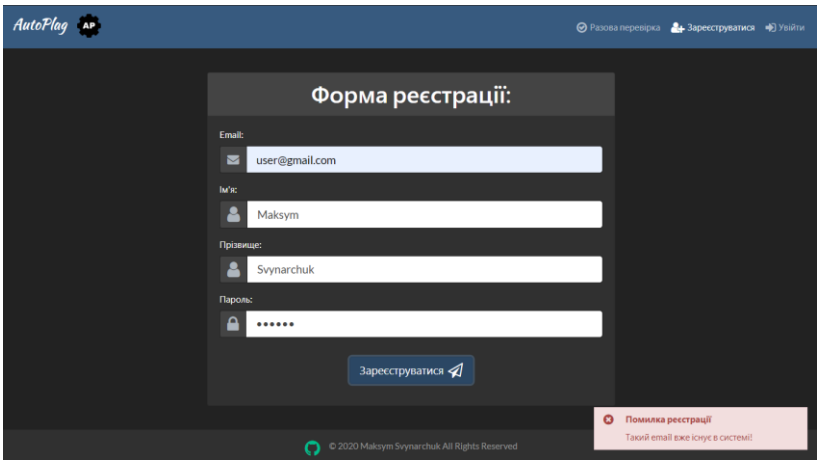


Рис. 2.1. Сторінка реєстрації

Для входу в систему користувач заповнює поля електронної пошти та пароля у формі авторизації. Якщо дані не співпадають, з’явиться помилка авторизації (рис. 2.2).

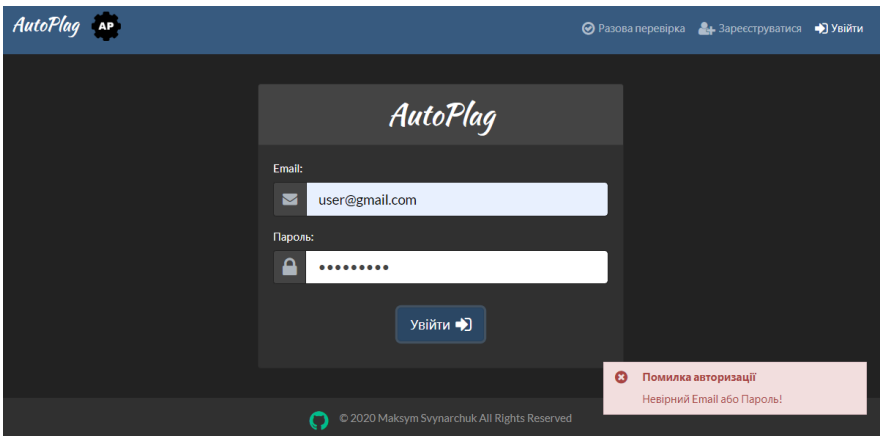


Рис. 2.2 Сторінка авторизації

3. ВИДИ ПЕРЕВІРОК ТА НАЛАШТУВАННЯ ЇХ ОПЦІЙ

Для виявлення плагіату веб-додаток надає два види перевірок.

3.1. Разова перевірка

Дозволяє користувачу виконати перевірку на плагіат серед власноруч завантажених проєктів. Форма налаштувань має 6 опцій, підказки до кожної можна переглянути якщо навести курсор миші на знак запитання біля назви опції (рис. 3.1):

1. Мова програмування для якої буде виконуватися перевірка.
2. Чутливість порівняння – менше значення підвищує чутливість.
3. Мінімальний відсоток співпадіння двох порівнюваних програм для збереження факту співпадіння в результатах.
4. Чи зберігати журнал виконання системи JPlag в результатах.
5. Архів з шаблонним кодом, який буде виключено з порівняння, не обов’язкова опція.
6. Архів з проєктами для яких буде виконана перевірка.

Архіви з програмним кодом обов’язково повинні бути у форматі .zip та розміром не більше 5 МВ.

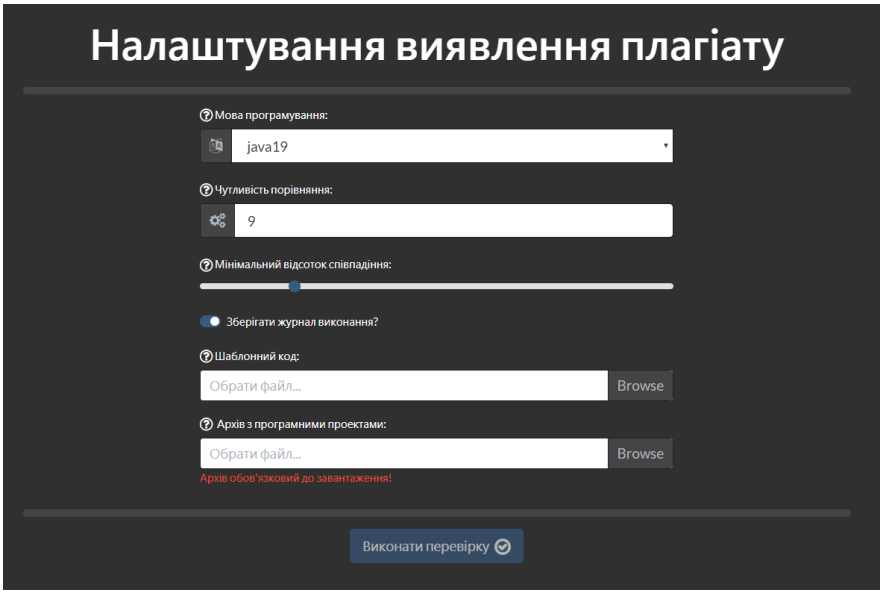


Рис. 3.1. Меню налаштування разової перевірки

3.2. Автоматизована перевірка (лише для авторизованих користувачів)

Дозволяє користувачу виконати перевірку на плагіат у заданий час серед студентів групи для конкретного завдання. Форма налаштувань має 8 опцій (рис. 3.2):

1. Завдання курсу для кого буде виконуватися перевірка.
2. Час виконання автоматичної перевірки.
3. Мова програмування для якої буде виконуватися перевірка.
4. Тип перевірки: Група – серед студентів даної групи; Курс – серед усіх студентів курсу, які виконували обране завдання.
5. Чутливість порівняння – менше значення підвищує чутливість.
6. Мінімальний відсоток співпадіння двох порівнюваних програм для збереження факту співпадіння в результатах.
7. Чи зберігати журнал виконання системи JPlag в результатах.
8. Архів з шаблонним кодом не більше 5 MB у форматі .zip, який буде виключено з порівняння, не обов’язкова опція.

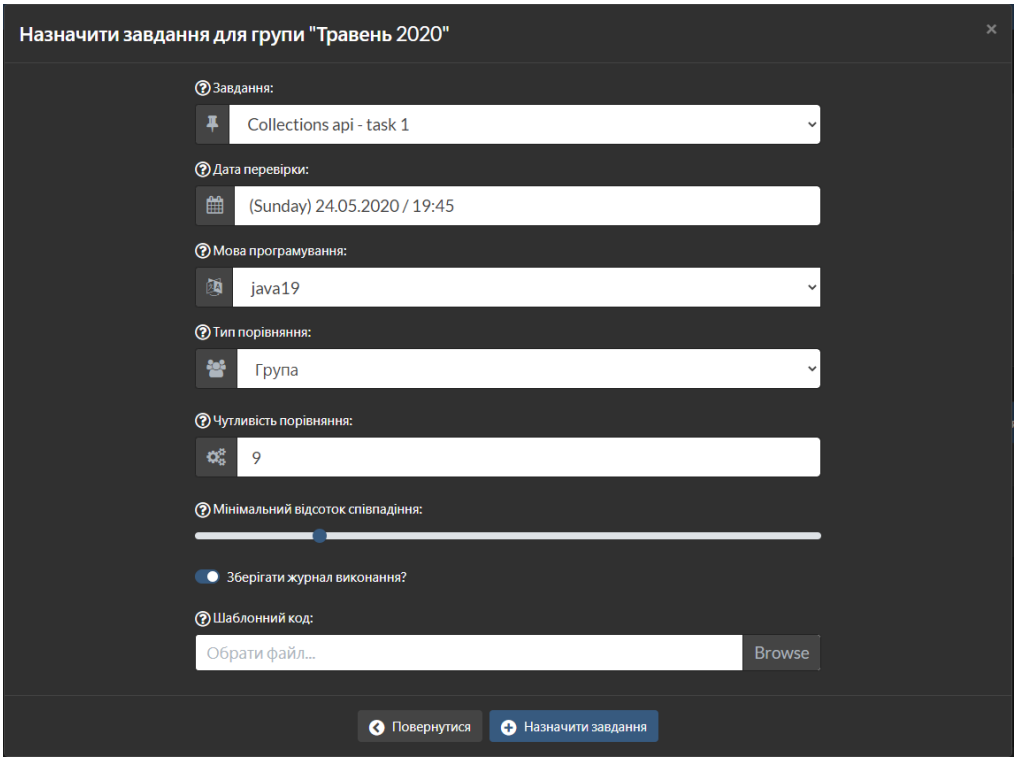


Рис. 3.2. Меню налаштування автоматизованої перевірки

4. НАЛАШТУВАННЯ РОБОЧОГО ПРОЦЕСУ ДЛЯ ВИКОРИСТАННЯ АВТОМАТИЗОВАНОЇ ПЕРЕВІРКИ

Перед налаштуванням процесу автоматизованої перевірки користувач повинен на сторінці профілю в меню «Налаштування VCS» авторизуватися у веб-сервісах для розміщення програмних проєктів на базі VCS (рис. 4.1). На даний момент підтримується GitHub та Bitbucket.

Після успішної авторизації система зможе завантажувати доступні для користувача репозиторії. Важливо, щоб студенти відкрили доступ до власних репозиторіїв будь-якому з авторизованих користувачем аккаунтів веб-сервісів.

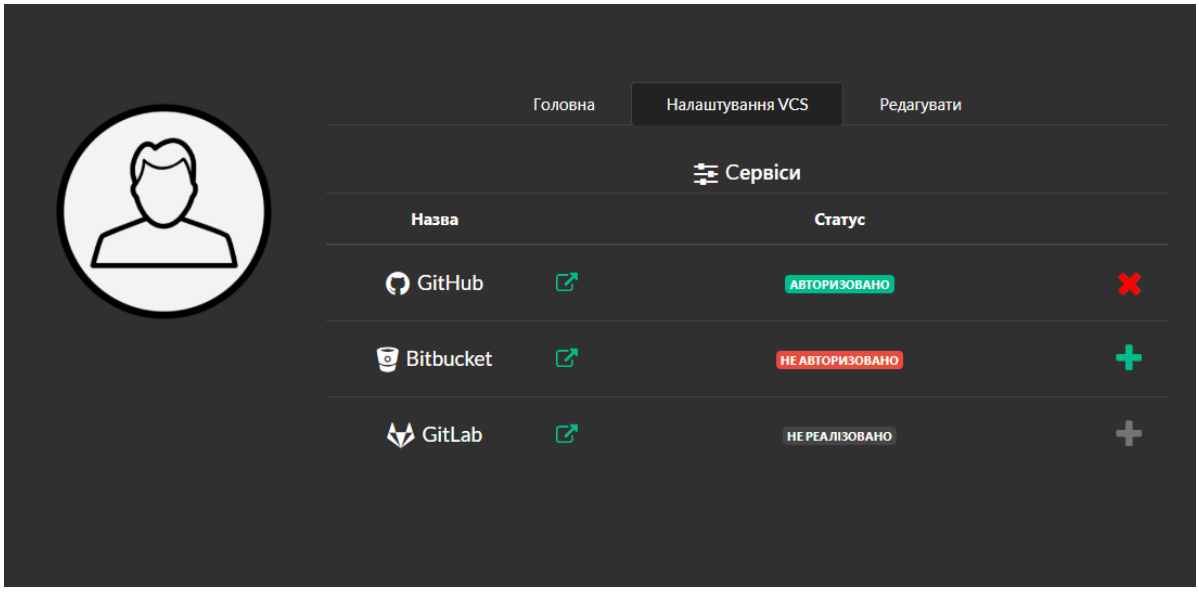


Рис. 4.1. Меню «Налаштування VCS» на сторінці профілю

Наступним необхідним кроком є додавання студентів до системи. Користувач на сторінці «Студенти» (рис. 4.2) на боковій панелі натискає на кнопку «Додати нового студенти», з'являється меню з полем для вводу повного ім'я студента. Важливо, щоб дане ім'я було унікальне серед всіх студентів, в іншому випадку – система не дозволить додати студента (рис. 4.3). Також система не дозволить видалити студента, якщо він включений хоча б до однієї групи.

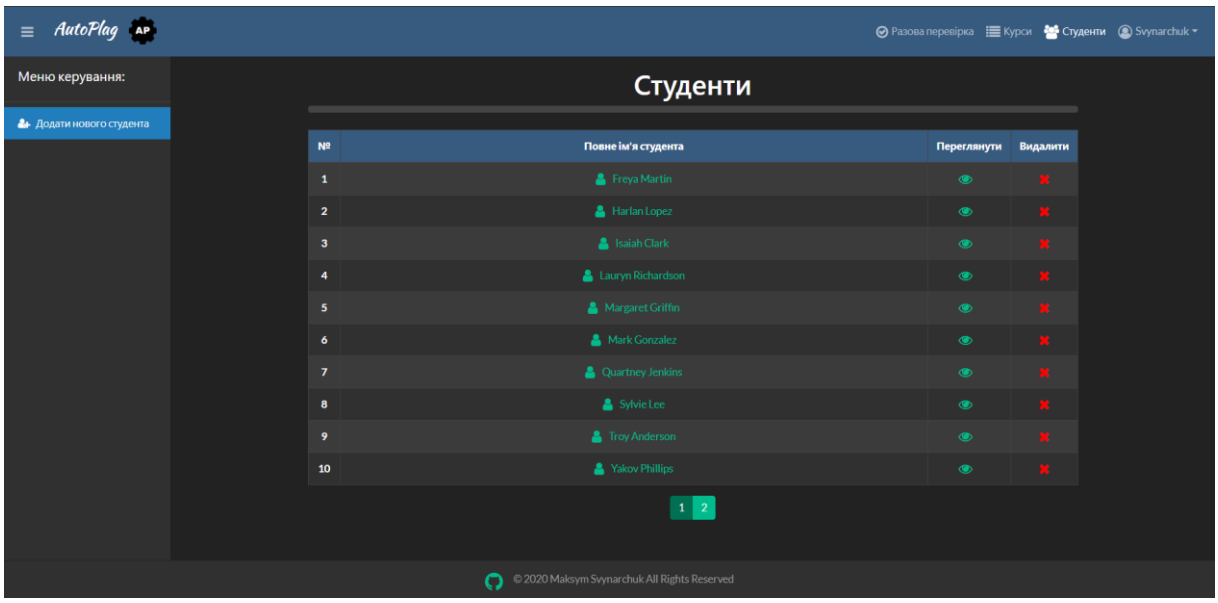


Рис. 4.2. Сторінка перегляду всіх студентів

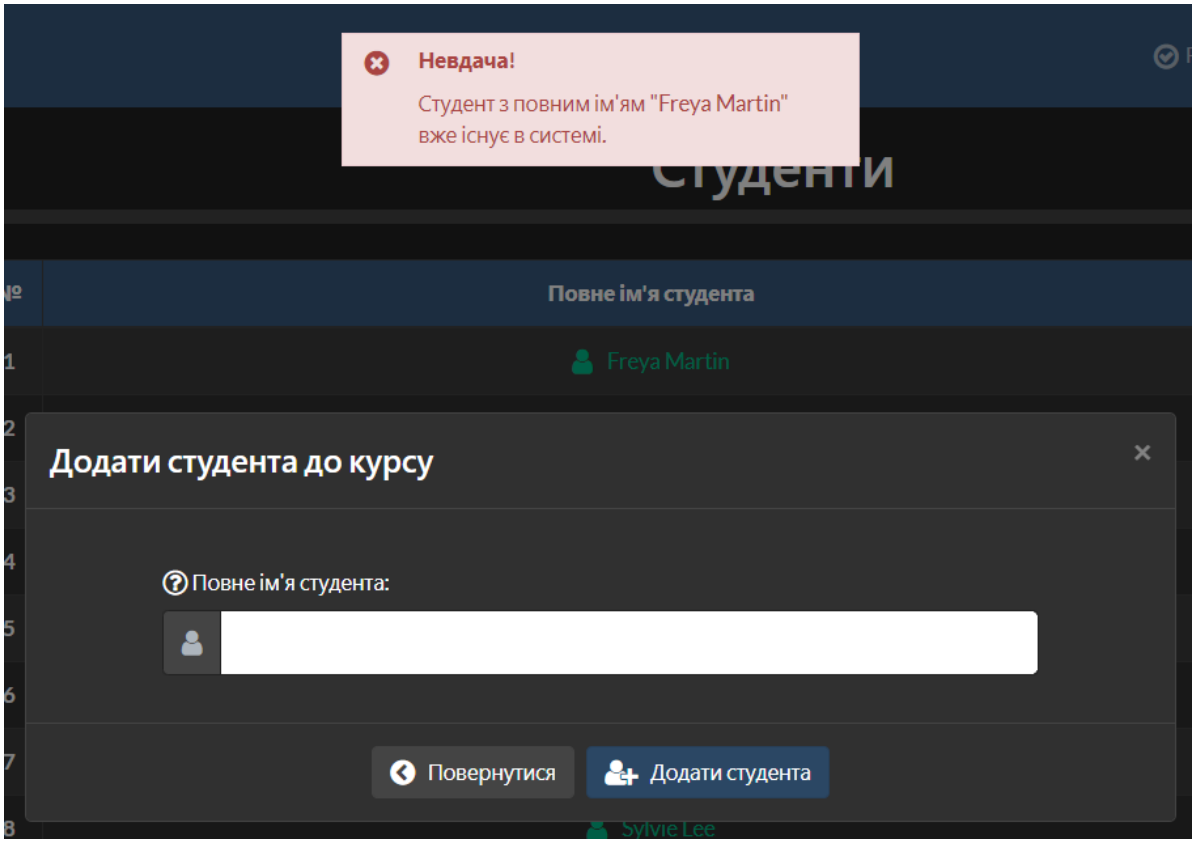


Рис. 4.3. Спроба додати вже існуючого студента в систему

Далі користувач повинен на сторінці «Курси» (рис. 4.4) на боковій панелі обрати «Створити новий курс» та у відкритому меню заповнити необхідні поля (рис. 4.5).

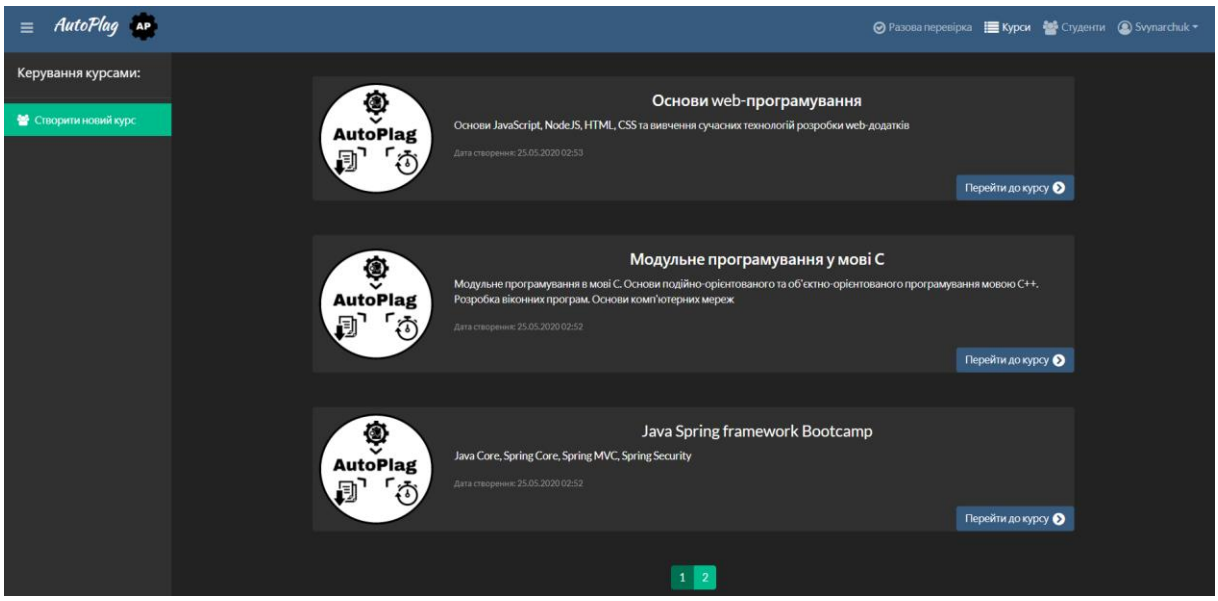


Рис. 4.4. Сторінка перегляду всіх курсів

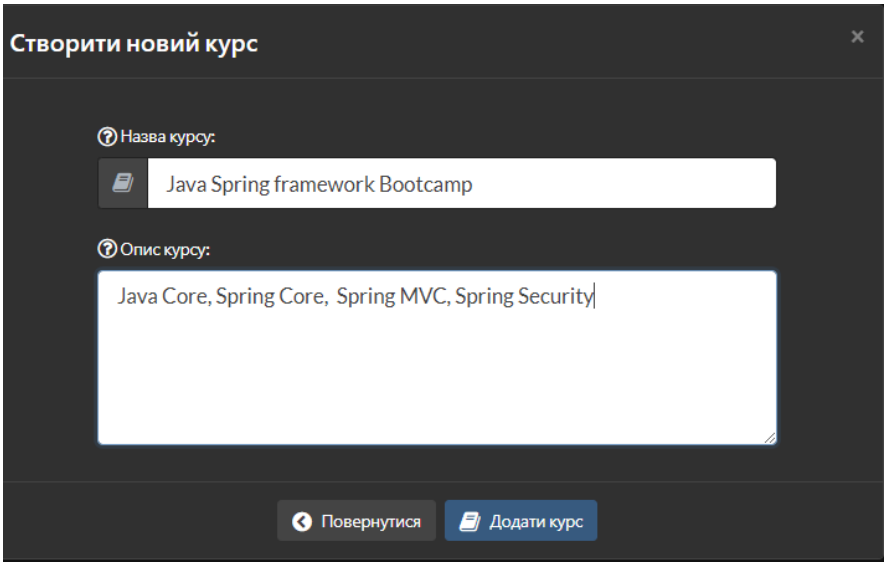


Рис. 4.5. Меню створення нового курсу

Після цього користувача буде переадресовано на сторінку перегляду новоствореного курсу (рис. 4.6). На боковій панелі стануть доступними меню видалення та редагуванні курсу, створення нової групи та посилання на сторінку всіх завдань курсу.

На сторінці завдань обраного курсу користувач може додавати нові завдання, редагувати та видаляти існуючі. Система не дозволить видалити завдання, які вже опубліковані в групах (рис. 4.7).

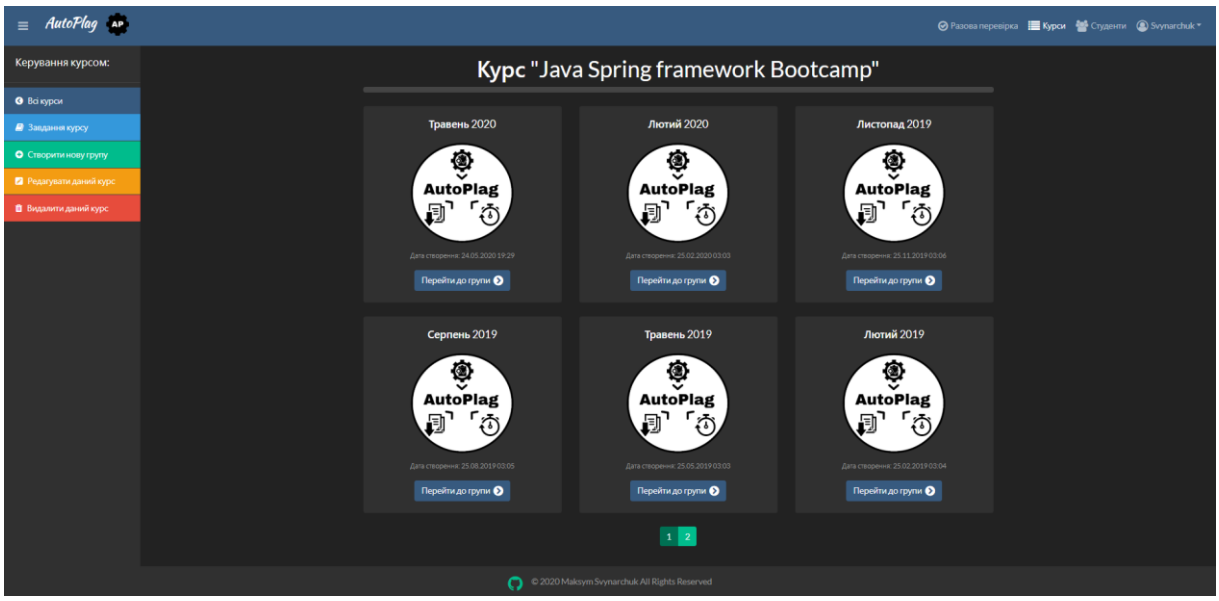


Рис. 4.6. Сторінка перегляду обраного курсу

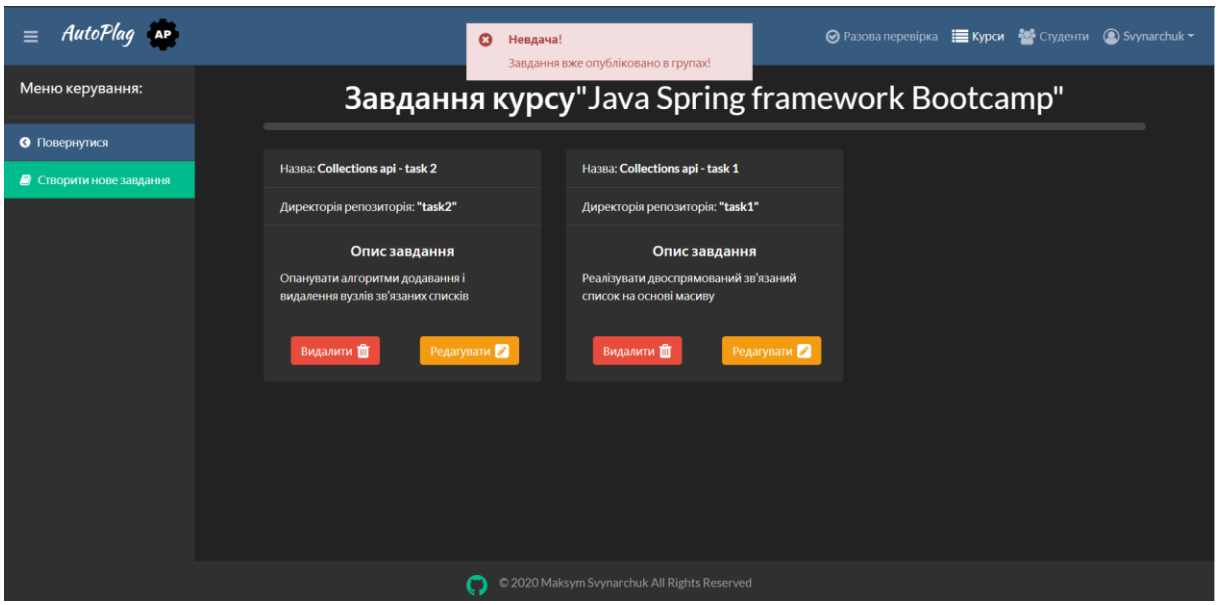


Рис. 4.7. Спроба видалити вже опубліковане завдання курсу

У меню створення нового завдання користувач обов’язково повинен вказати назву та шлях до кінцевої директорії на репозиторії, за яким система завантажить рішення студента (рис. 4.8). Після цього користувач може назначати завдання для наявних в курсі груп.

На сторінці обраної групи користувачу доступні різні можливості для управління нею (рис. 4.9). Користувач може редагувати чи видалити групу, додавати та видаляти студентів, назначати чи видаляти завдання.

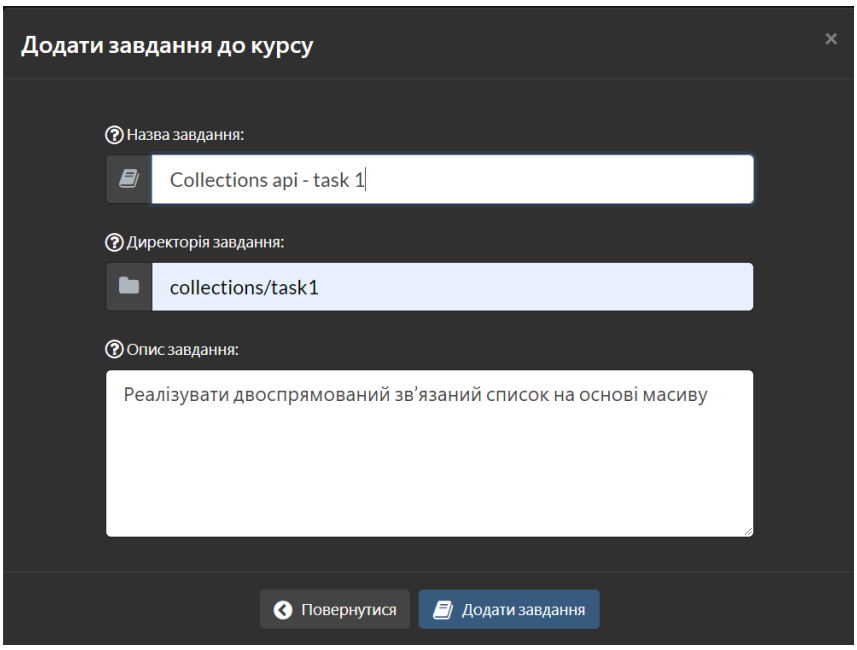


Рис. 4.8. Меню створення нового завдання

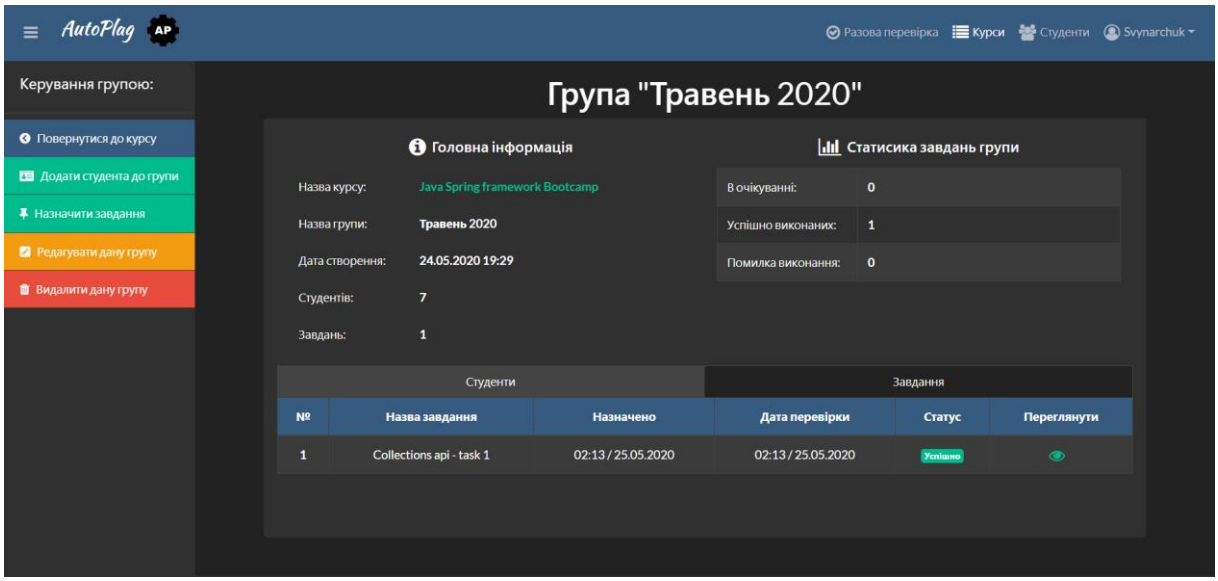


Рис. 4.9. Сторінка перегляду обраної групи

У меню «Додати студента до групи» доступні лише ті студенти, які не включені до жодної з груп даного курсу. Користувач повинен вказати посилання на репозиторій студента. Система перевіряє чи існує такий репозиторій та чи має користувач до нього доступ. Якщо всі перевірки виконані, студент стає учасником групи (рис. 4.10). Користувач також може видаляти студентів з групи, але система не дозволить видалити двох останніх, якщо ще наявні завдання для перевірки.

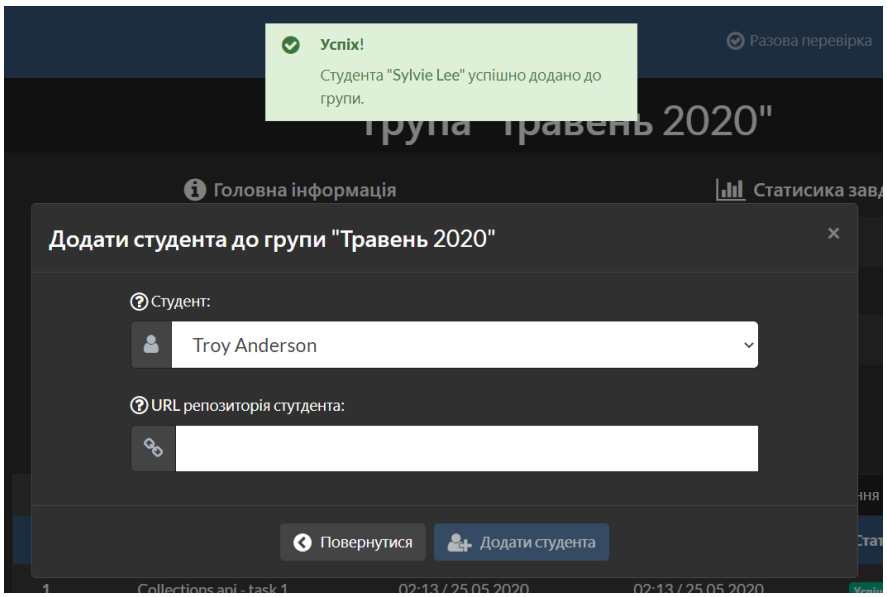


Рис. 4.10. Спроба додати студента до групи

У меню «Назначити завдання» користувач налаштовує процес автоматизованої перевірки для конкретного завдання (див. розділ 3). Система не дозволить назначити завдання для групи, якщо в ній менше двох студентів.

Після налаштування завдання для групи стає доступною сторінка з детальною інформацією по ньому (див. розділ 5). На цій сторінці користувач може редагувати чи видалити завдання з групи, а також розпочати перевірку негайно чи повторити перевірку, якщо вона вже була виконана.

5. ПЕРЕГЛЯД РЕЗУЛЬТАТІВ ВИЯВЛЕННЯ ПЛАГІАТУ

Після закінчення будь-якої з двох типів перевірок для виявлення плагіату веб-додаток на окремій веб-сторінці дає можливість користувачу переглянути інформацію, де буде вказано обрані користувачем налаштування, успішно чи ні завершилася перевірка, журнал виконання (якщо користувач обрав необхідну опцію для його збереження) та посилання на перегляд статичних веб-сторінок з результатами виявлення плагіату, згенерованих системою JPlag (рис. 5.1).

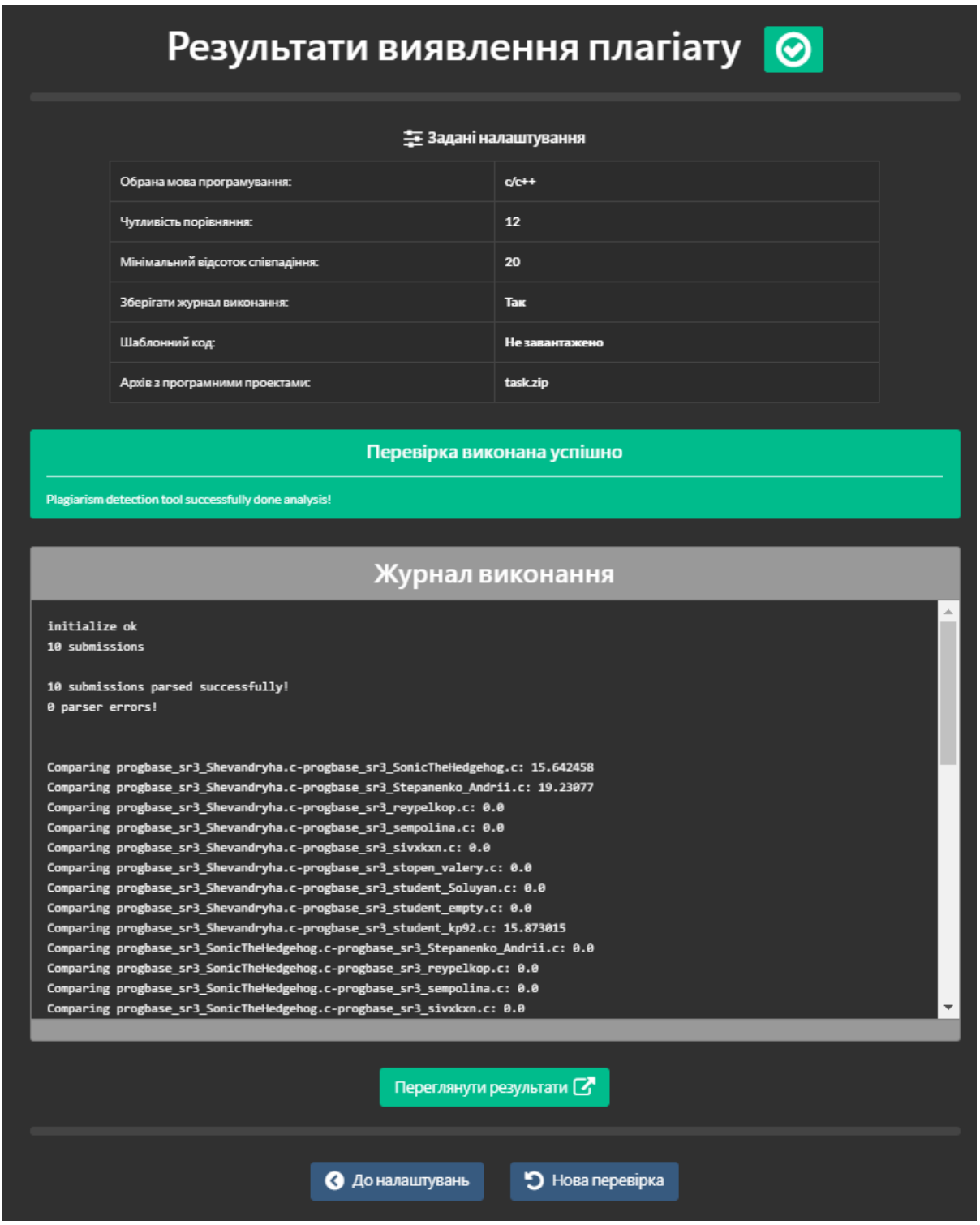


Рис. 5.1. Успішний результат разової перевірки

На сторінки результатів автоматизованої перевірки також додатково вказана інформація про завдання, що перевіряються та помилки при завантаженні репозиторіїв студентів, якщо такі є (рис. 5.2).

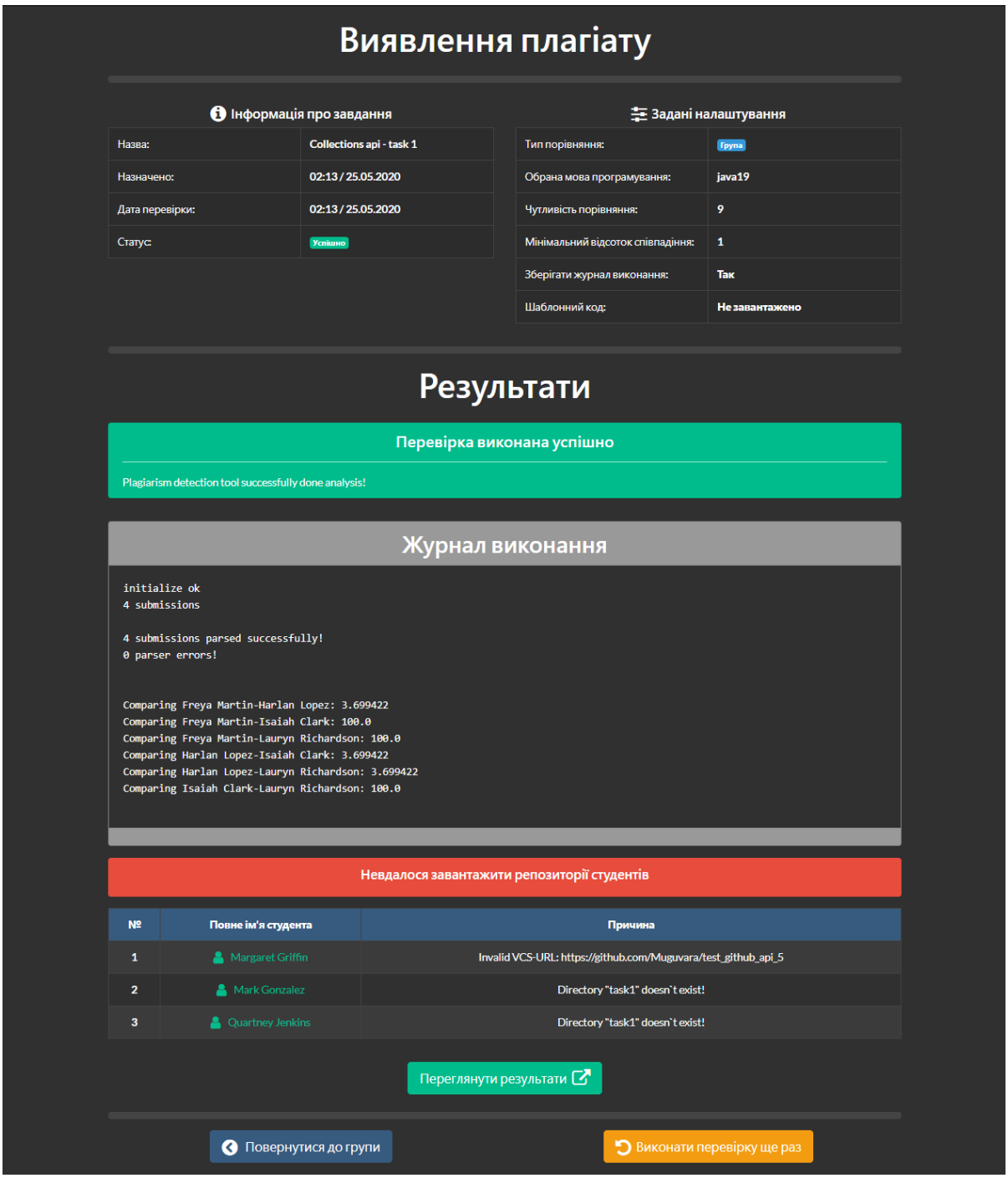


Рис. 5.2. Успішний результат автоматизованої перевірки

Головна сторінка результатів виявлення плагіату (рис. 5.3) розбита на 4 секції:

1. Зверху вказана загальна інформація, така як назви програмних проєктів, мова програмування, кількість програмних проєктів, кількість порівнянь, дата аналізу, мінімальна довжина токена для співпадіння, та розширення файлів, що порівнюються.

2. Секція розподілу показує скільки проєктів мають відповідне значення відсотку плагіату між собою.
3. Співпадіння, відсортовані за середньою схожістю.
4. Співпадіння, відсортовані за максимальною схожістю.

JPlag

Search Results

Title:	
Programs:	progbase_sr3_Shevandryha.c - progbase_sr3_SonicTheHedgehog.c - progbase_sr3_Stepanenko_Andrii.c - progbase_sr3_reypelkop.c - progbase_sr3_sempolina.c - progbase_sr3_sivxkxn.c - progbase_sr3_stopen_valery.c - progbase_sr3_student_Soluyan.c - progbase_sr3_student_empty.c - progbase_sr3_student_kp92.c - progbase_sr3_valerystudent.c
Language:	C/C++ Scanner [basic markup]
Submissions:	11
Matches displayed:	8 (Treshold: 21.1%) (average similarity) 12 (Treshold: 20.5%) (maximum similarity)
Date:	2020-05-31
Minimum Match Length (sensitivity):	12
Suffixes:	.cpp, .CPP, .cxx, .CXX, .c++, .C++, .c, .C, .cc, .CC, .h, .H, .hpp, .HPP, .hh, .HH

Distribution:

90% - 100%	0	.
80% - 90%	1	##
70% - 80%	0	.
60% - 70%	0	.
50% - 60%	0	.
40% - 50%	4	########
30% - 40%	0	.
20% - 30%	3	######
10% - 20%	10	##########
0% - 10%	37	##########

Matches sorted by average similarity (What is this?):

progbase_sr3_valerystudent.c	->	progbase_sr3_Stepanenko_Andrii.c (85.0%)	progbase_sr3_stopen_valery.c (46.9%)	progbase_sr3_student_kp92.c (45.4%)
progbase_sr3_Stepanenko_Andrii.c	->	progbase_sr3_student_kp92.c (44.3%)	progbase_sr3_stopen_valery.c (41.7%)	progbase_sr3_student_empty.c (27.7%)
progbase_sr3_student_empty.c	->	progbase_sr3_stopen_valery.c (21.1%)		

Matches sorted by maximum similarity (What is this?):

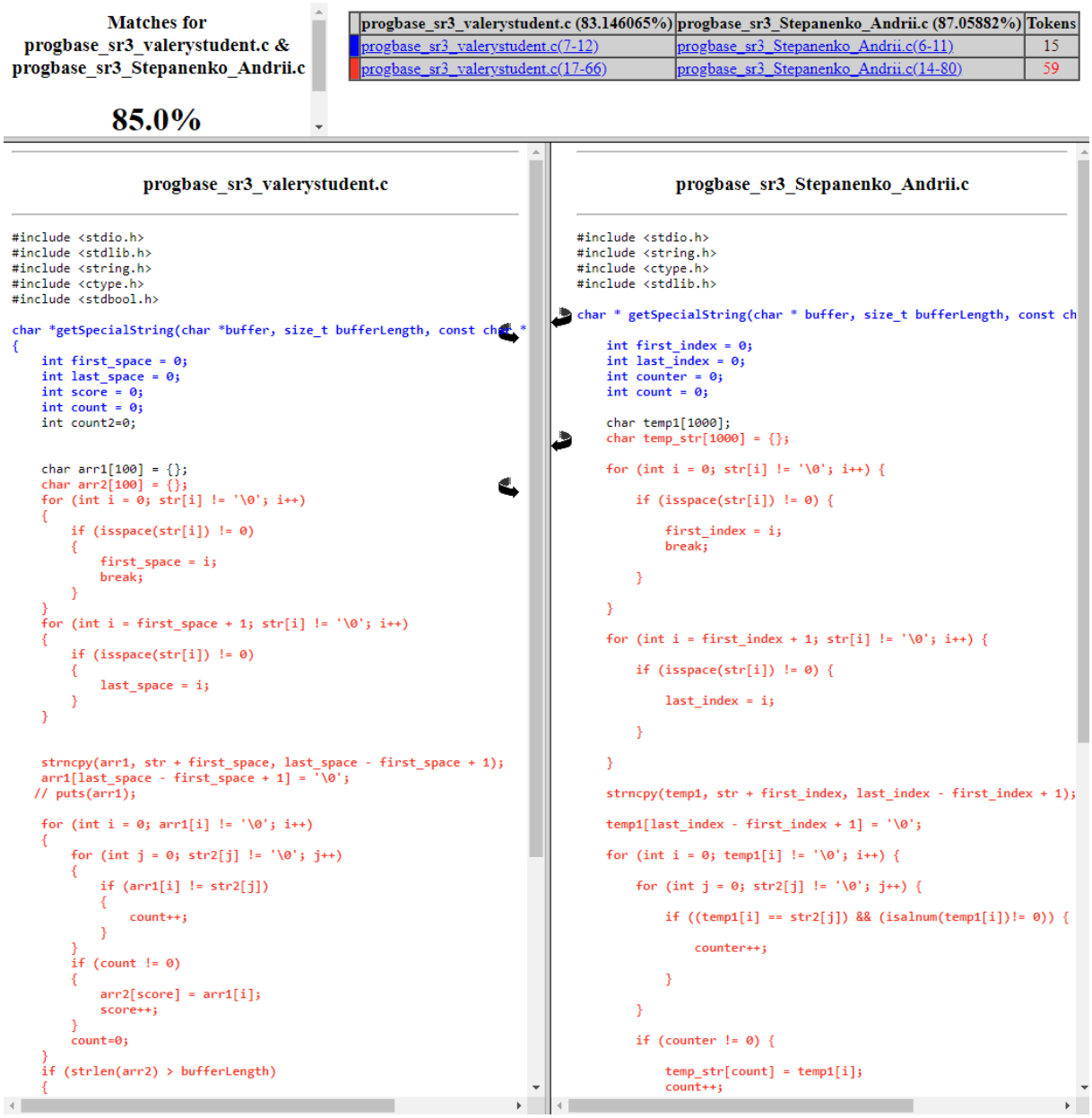
progbase_sr3_valerystudent.c	->	progbase_sr3_Stepanenko_Andrii.c (87.0%)	progbase_sr3_student_kp92.c (52.8%)	progbase_sr3_stopen_valery.c (52.0%)
progbase_sr3_Stepanenko_Andrii.c	->	progbase_sr3_student_kp92.c (52.9%)	progbase_sr3_stopen_valery.c (45.2%)	progbase_sr3_student_empty.c (28.2%)
progbase_sr3_stopen_valery.c	->	progbase_sr3_student_empty.c (23.2%)	progbase_sr3_student_kp92.c (20.5%)	
progbase_sr3_student_kp92.c	->	progbase_sr3_Shevandryha.c (21.1%)		

Рис. 5.3. Загальна сторінка результатів JPlag

Користувач може переглянути виявлені ділянки співпадінь між парами проєктів окремо (рис. 5.4). Сторінка розділена на чотири частини:

1. Дві великі секції внизу містять лістинг вихідного коду порівнюваних програмних проєктів, в яких подібні уривки позначені одним кольором.

2. Ліва секція у верхній частині показує відсоток подібності.
3. Права секція містить таблицю всіх уривків, які збігаються. Кожен рядок вказує діапазон номерів рядків з лістингів кожного проєкту окремо, які вважаються однаковими, разом із розміром збігу в лексемах.



Matches for progbase_sr3_valerystudent.c & progbase_sr3_Stepanenko_Andrii.c

85.0%

progbase_sr3_valerystudent.c (83.146065%)	progbase_sr3_Stepanenko_Andrii.c (87.05882%)	Tokens
progbase_sr3_valerystudent.c(7-12)	progbase_sr3_Stepanenko_Andrii.c(6-11)	15
progbase_sr3_valerystudent.c(17-66)	progbase_sr3_Stepanenko_Andrii.c(14-80)	59

progbase_sr3_valerystudent.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdbool.h>

char *getSpecialString(char *buffer, size_t bufferLength, const ch
{
    int first_space = 0;
    int last_space = 0;
    int score = 0;
    int count = 0;
    int count2=0;


    char arr1[100] = {};
    char arr2[100] = {};
    for (int i = 0; str[i] != '\0'; i++)
    {
        if (isspace(str[i]) != 0)
        {
            first_space = i;
            break;
        }
    }
    for (int i = first_space + 1; str[i] != '\0'; i++)
    {
        if (isspace(str[i]) != 0)
        {
            last_space = i;
        }
    }


    strncpy(arr1, str + first_space, last_space - first_space + 1);
    arr1[last_space - first_space + 1] = '\0';
   // puts(arr1);

    for (int i = 0; arr1[i] != '\0'; i++)
    {
        for (int j = 0; str2[j] != '\0'; j++)
        {
            if (arr1[i] != str2[j])
            {
                count++;
            }
        }
        if (count != 0)
        {
            arr2[score] = arr1[i];
            score++;
        }
        count=0;
    }
    if (strlen(arr2) > bufferLength)
    {
```

progbase_sr3_Stepanenko_Andrii.c

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

char * getSpecialString(char * buffer, size_t bufferLength, const ch

    int first_index = 0;
    int last_index = 0;
    int counter = 0;
    int count = 0;

    char temp1[1000];
    char temp_str[1000] = {};

    for (int i = 0; str[i] != '\0'; i++) {

        if (isspace(str[i]) != 0) {

            first_index = i;
            break;

        }

    }

    for (int i = first_index + 1; str[i] != '\0'; i++) {

        if (isspace(str[i]) != 0) {

            last_index = i;

        }

    }

    strncpy(temp1, str + first_index, last_index - first_index + 1);

    temp1[last_index - first_index + 1] = '\0';

    for (int i = 0; temp1[i] != '\0'; i++) {

        for (int j = 0; str2[j] != '\0'; j++) {

            if ((temp1[i] == str2[j]) && (isalnum(temp1[i])!= 0)) {

                counter++;

            }

        }

        if (counter != 0) {

            temp_str[count] = temp1[i];
            count++;
```

Рис. 5.4. Перегляд співпадінь між вихідним кодом двох програмних проєктів