**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
« » червня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня бакалавр**

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система автоматизованого тестування з підвищеними вимогами до безпеки»
здобувача групи ІН - 01 Бондаря Владислава Валерійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Владислав БОНДАР
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук, к.т.н., доцент Борис КУЗІКОВ
(підпис)

**Суми – 2024**

**Сумський державний університет**
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня бакалавра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Бондаря Владислава Валерійовича

1. Тема роботи: «Інформаційна система автоматизованого тестування з підвищеними вимогами до безпеки» затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 29 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд реалізованих рішень, що дозволяють тестувати програмні продукти. 3) Розробка UI компонентів та архітектури інформаційної системи. 4) Розробка програмного забезпечення для автоматизації процесу тестування з акцентом на забезпечення високого рівня безпеки 5) Аналіз результатів роботи.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____

_____ _____
(підпис) (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Аналіз проблеми предметної області, постановка й формування завдань дослідження* | | |
| *2* | *Огляд реалізованих рішень для тестування програмних продуктів* | | |
| *3* | *Розробка UI компонентів та архітектури інформаційної системи* | | |
| *4* | *Розробка програмного забезпечення для автоматизації процесу тестування з акцентом на забезпечення високого рівня безпеки* | | |
| *5* | *Аналіз результатів роботи* | | |
| *6* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | | |

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

# АНОТАЦІЯ

**Записка**: 62 стр., 26 рис., 1 додаток, 23 використаних джерел.

**Обґрунтування актуальності теми роботи** – Тема кваліфікаційної роботи є актуальною, оскільки присвячена створенню ізольованих тестевих середовищ для тестування потенційно небезпечних додатків.

**Об'єкт дослідження** — процес тестування програмних додатків.

**Мета роботи** — розробка інформаційної системи автоматизованого тестування java додатків із підвищеними вимогами до безпеки середовища тестування.

**Методи дослідження** — віртуалізація та контейнеризація програмних додатків.

**Результати** — розроблено інформаційну систему, яка отримує tar-архів з програмним додатком, створює Docker контейнер з ним, тестує додаток всередині системи та надсилає результат тестування користувачу.

ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗОВАЦІЯ ТЕСТУВАННЯ, JAVA, DOCKER, SPRING BOOT, MAVEN, THYMELEAF, ВІРТУАЛІЗАЦІЯ.

**Зміст**

# Вступ

**Актуальність.** У сучасному IT-середовищі безпека вважається одним із найважливіших аспектів розробки програмного забезпечення. Однак, жодна операційна система не є абсолютно безпечною, що створює певний ризик для комп'ютерних систем. Багато розробників і фахівців з безпеки зацікавлені у тому, щоб мати можливість тестувати будь-які додатки максимально безпечно. Проте у випадку загроз зі сторони невідомих або малодосліджених програм залишати усі небезпеки ізольованими в рамках якогось середовище, що не має впливу на основну систему. Таким чином, розробка інформаційно системи для автоматизованого тестування додатків з підвищеними вимогами до безпеки буде мати позитивний вплив на розвиток систем, що слугують ізольованим середовищем для тестування коду та є резилієнтними.

**Об’єкт дослідження.** Процес тестування програмних додатків.

**Предмет дослідження.** Створення Docker контейнерів засобами Java Docker API Client та запуск Java додатків в них.

**Гіпотеза.** Створення ізольованого середовище, яке є безпечним для тестування додатків з підвищеними вимогами до безпеки, можна досягнути шляхом створення Docker контейнерів з використанням Java Docker API Client.

**Новизна**. На відміну від існуючих аналогів інформаційних систем, описане у даній роботі програмне рішення дозволить швидко тестувати заархівовані Java додатки та отримувати детальну статистику по кожному з тестових кейсів без необхідності розархівації і повного їх перенесення у середовище, де проводиться тестування.

**Структура.** Дане робота складається зі вступу, аналізу реалізованих рішень, постановки задачі, проектування UI та архітектури додатку, програмної реалізації, висновків, списку використаних джерел та додатків.

# 1. ІНФОРМАЦІЙНИЙ ОГЛЯД

## 1.1. Роль інформаційної безпеки при тестуванні додатків

Сучасний світ інформаційних технологій стрімко розвивається, що призводить до зростання кількості програмних додатків. Однак, разом з цим і зростає ризик для безпеки даних й інформаційних систем в цілому. Програмне забезпечення не є досконалими та досі містить багато вразливих місць, які є чудовими мішенями для зловмисників. Тому ретельне і всебічне тестування додатків є вкрай важливим. Ресурсні обмеження не завжди дозволяють провести таке тестування на комп'ютері розробника. При використанні зовнішнього тестового оточення для тестування маємо протиріччя між економічною доцільністю використання його для тестування виключно одного додатку та ризиками витоку даних при тестуванні кількох додатків в одному оточенні [21].

У сучасному світі мало кому можна довіряти, особливо при роботі з програмним забезпеченням, оскільки навіть зараз існує дуже багато вразливостей, через які зловмисники можуть отримати доступ до конфіденційної інформації або навіть управляти системою. Сучасні операційні системи, такі як Windows та Linux, можуть бути легко скомпрометовані досвідченими кіберзлочинцями, які легко можуть отримати доступ до усіх можливостей комп'ютера за допомогою лише одного скрипту, який буде виконано інколи навіть не помітно для користувача.

Інколи виникають ситуації, коли потрібно запускати код, автору якого не можна повністю довіряти, оскільки потенційно будь-який код може містити загрозу. Зазвичай це може бути потрібно тоді, коли потрібно щось протестувати, наприклад, необхідно впевнитися, що код взагалі запуститься і виконує корисну роботу, проте виконання самого коду на головну систему не вплине. Проблема збереження конфіденційності даних в таких випадках є критичною. Якщо програмне забезпечення неналежно обробляє конфіденційні дані, це може призвести до їх неправомірного доступу або витоку. Це особливо небезпечно на

тих системах, де користувач використовує фінансову інформацію, оскільки зловмисник потенційно можуть отримати до неї доступ. Шкідливе програмне забезпечення може навіть впливати на роботу інших програм або навіть змінювати налаштування операційної системи з метою отримання додаткових прав або доступу. Небезпека втрати контролю над системою також велика. Зловмисники можуть намагатися встановити «backdoor» або інші шкідливі програми на комп'ютері з метою виконання подальших атак або викрадення даних.

Спектр можливих наслідків для системи є майже безкінечним. Шкідливе програмне забезпечення може набувати характер «інтернет-троля» та просто вимикати комп’ютер у будь-який момент часу. Також можливий сценарій, коли зловмисники зашифрують жорсткий диск, а потім будуть вимагати гроші за його розшифрування, подібно до комп’ютерного вірусу «Petya».

Під час проведення тестування програмного продукту, основна мета – це перевірити, чи працює логіка так, як очікується, тобто, чи відповідає вона функціональність вимогам та чи правильно вона обробляє дані. Однак, під час цього процесу можна натрапити на небезпечний код у програмі, який може потенційно спричинити проблеми з безпекою чи стабільністю системи.

Наприклад, якщо у програмі є уразливості, такі як недостатньо перевірені вхідні дані чи можливість виконання вірусного коду, це може призвести до порушень безпеки. Також, недоліки у програмному коді можуть призвести до некоректної роботи програми, що може виявитися під час її експлуатації.

Незважаючи на це, при тестуванні програмного продукту, ми робимо це в ізольованому середовищі. Це означає, що навіть якщо в програмі є «поганий» код, який може призвести до проблем, ці проблеми не вплинуть на роботу системи в цілому. Ізольоване середовище дозволяє проводити тестування безпечно, навіть якщо виявляються проблеми у програмі.

Такий підхід забезпечує, що навіть при виявленні недоліків у програмному коді під час тестування, наша система залишається стійкою та безпечною для користувачів.

## 1.2. Огляд існуючих рішень

HackerRank — це платформа для практики програмування та вирішення алгоритмічних задач. На платформі новачки в програмуванні можуть попрактикуватись в розв’язуванні алгоритмічних задач. Користувач обирає задачу, читає її умову та починає писати код на мові програмування, яку він обрав. У переліку доступних мов багато популярних варіантів: Python, C++, Java, JavaScript. Після завершення вирішення задачі, сайт запустить виконання та порівнює результати з тими, які є всередині системи, щоб переконатися, що завдання було виконано вірно.

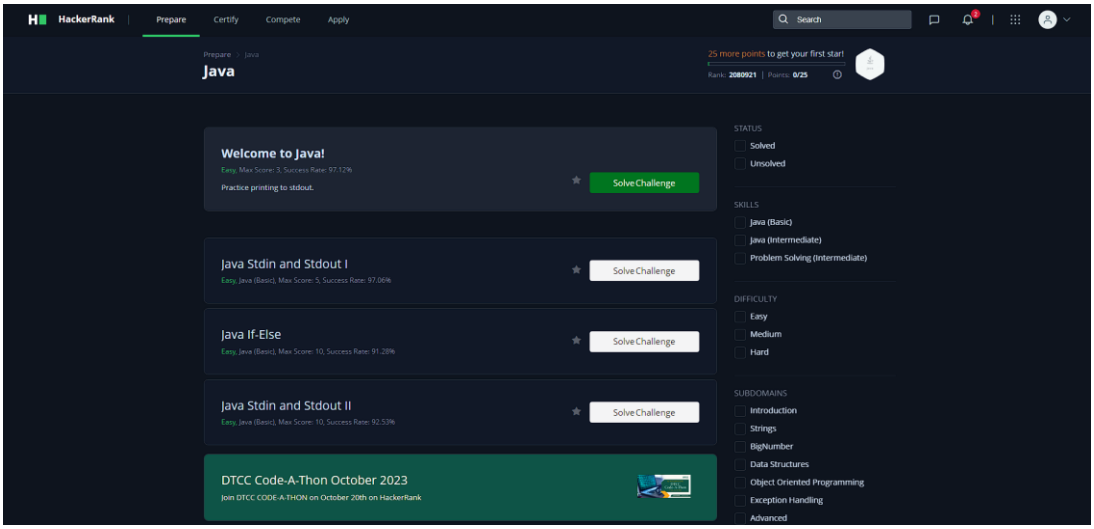


Рисунок 1.1 – Перелік задач на HackerRank

HackerRank також дає вам можливість переглядати приклади коду та використовувати оцінку системи для отримання відгуків про якість вашого рішення.

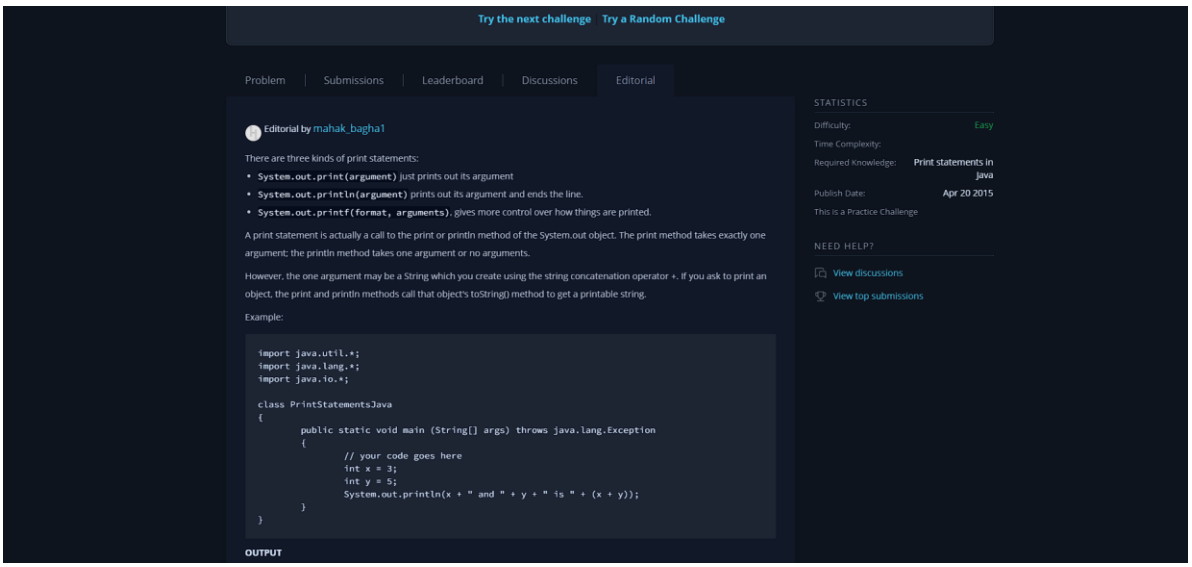


Рисунок 1.2 - Розбір задачі з HackerRank

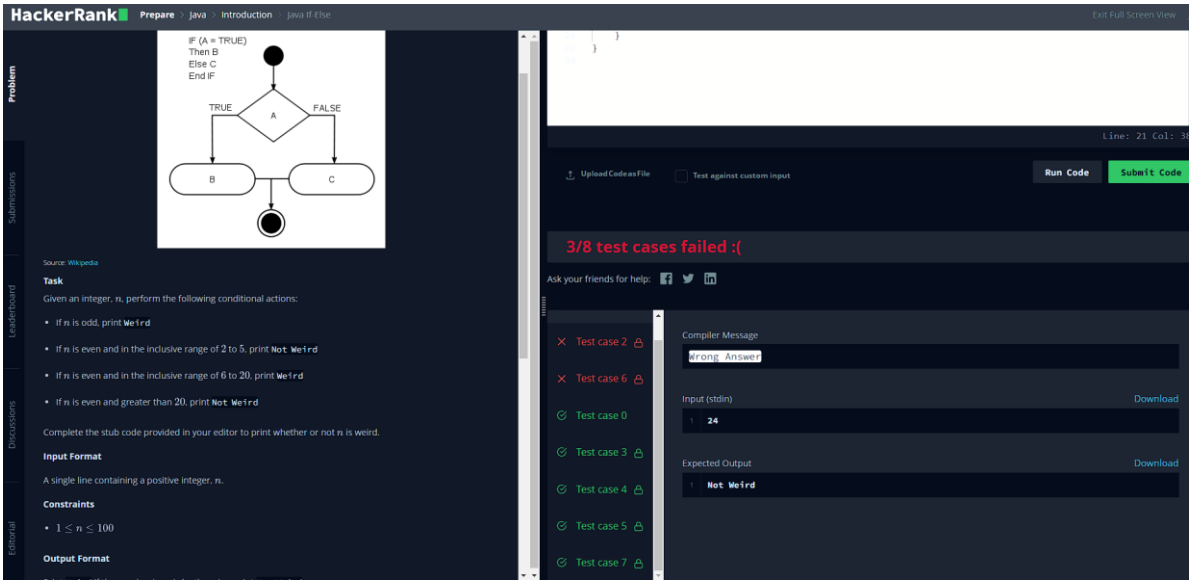


Рисунок 1.3 - Оцінка якості коду

Replit — це платформа розробки веб-додатків, яка підтримує різні мови програмування. Платформа надає можливості створювати проекти, писати код, компілювати та запускати його разом у веб-браузері. Такі сервіси часто мають назву «sandbox», оскільки в них можна виконувати будь-яку логіку при цьому бути впевненим, що основна користувацька система не зазнає шкоди. Такі веб-додатки користуються популярністю серед користувачів, яким потрібно швидко розгорнути і перевірити проект, але через безпекові причини вони не можуть це зробити на власному комп’ютері.

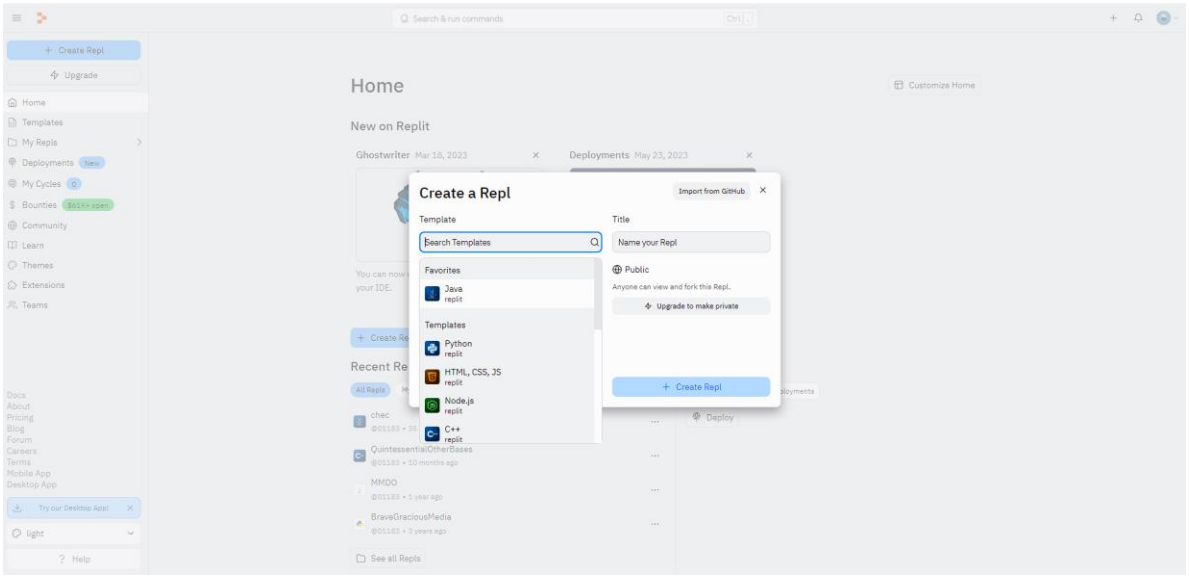


Рисунок 1.4 - платформа розробки веб-додатків Replit

Replit надає вбудоване середовище розробки з можливістю редагувати код, запускати програму та спостерігати за результатами.

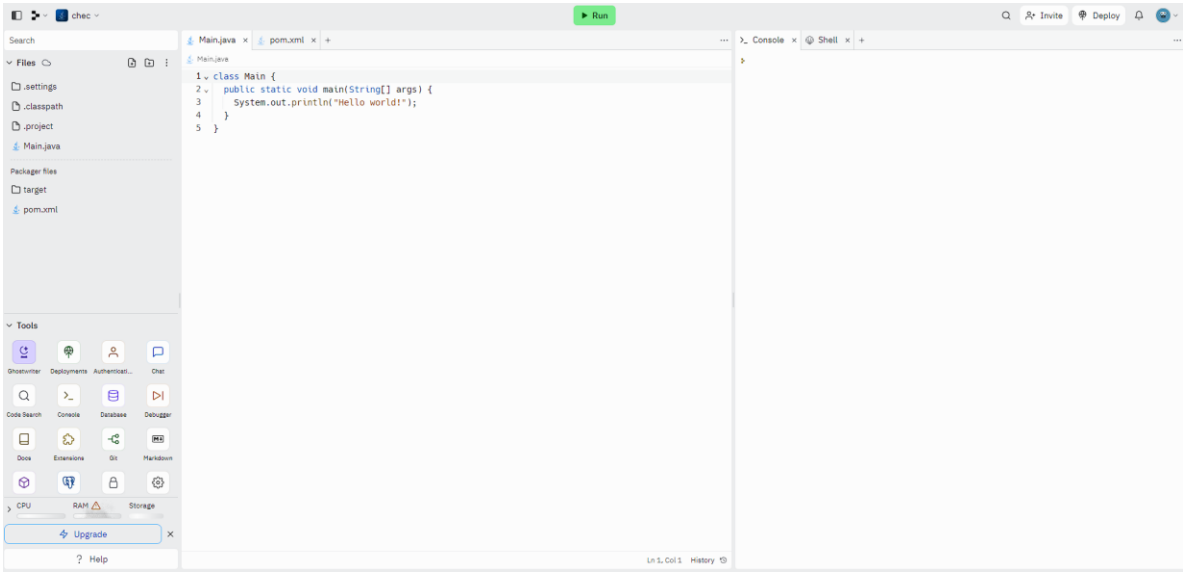


Рисунок 1.5 - вбудоване середовище розробки Replit

Користувач також можете створювати власні тести, щоб перевірити функціональність свого коду.

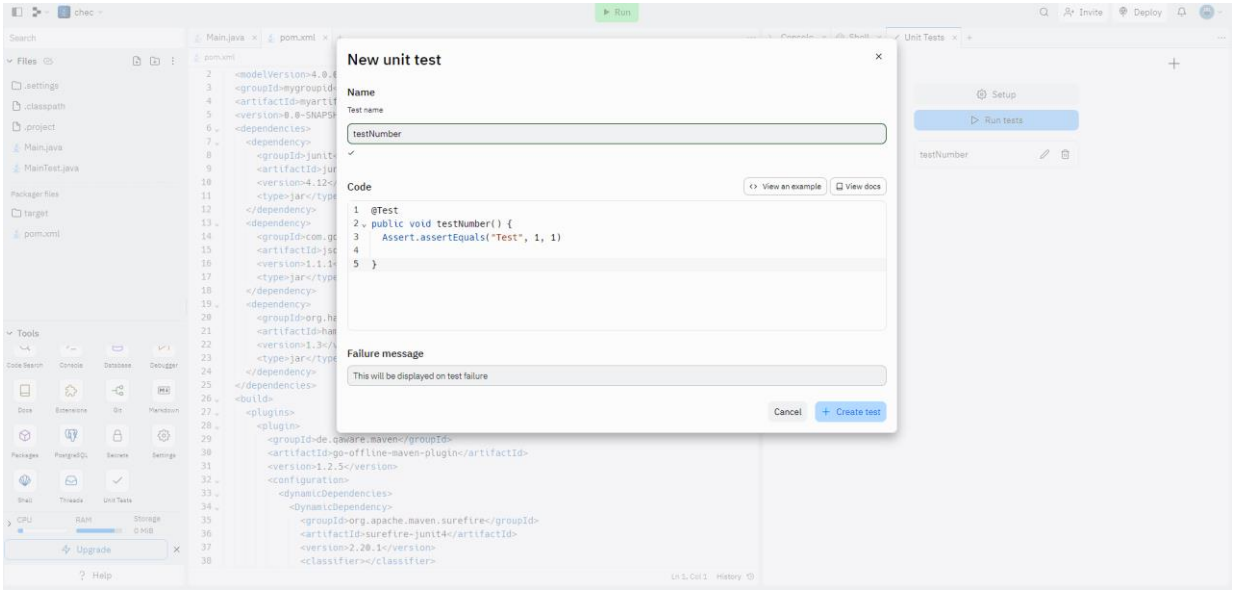


Рисунок 1.6 - Створення власних тестів в Replit

Replit також має функцію спільної роботи над проектом, яка дозволяє легко співпрацювати з іншими розробниками.

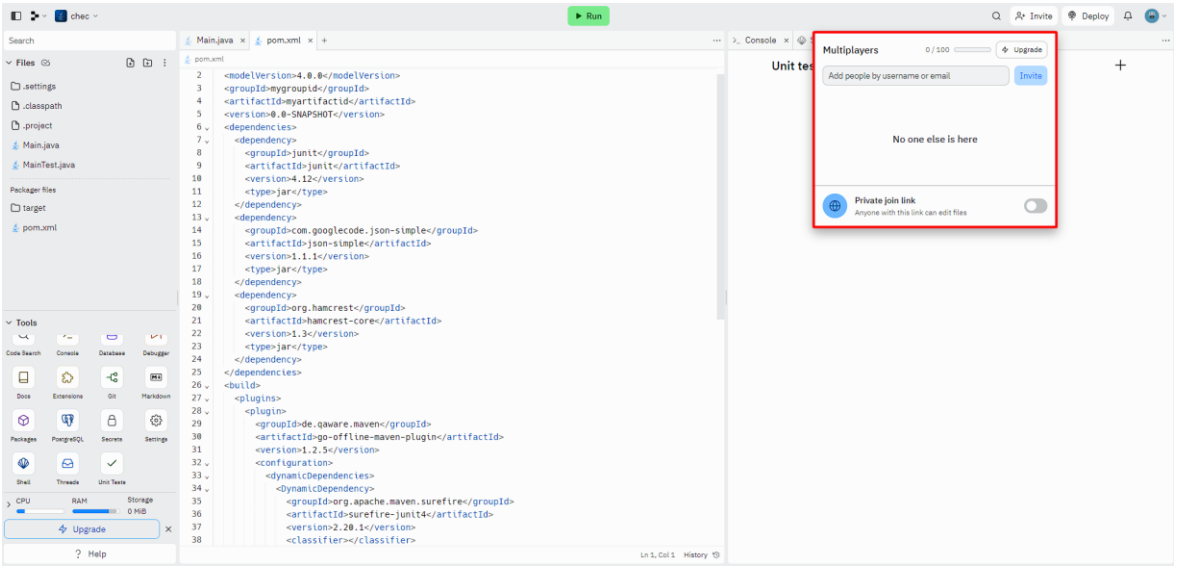


Рисунок 1.7 - Діалогове вікно для додавання нових членів команди в Replit

LeetCode — це популярна онлайн-платформа, яка пропонує набір завдань для кодування, щоб допомогти людям потренуватися та вдосконалити свої навички програмування. Сервіс набув своє популярності через те, що дозволяє користувачам підготуватися до співбесід у великих IT-компаніях, бо багато задач на цьому сервісі зустрічаються на технічних інтерв'ю. Перевірка задач відбувається майже моментально, а кількість тестових кейсів велика, що дозволяє швидко та якісно перевірити код.

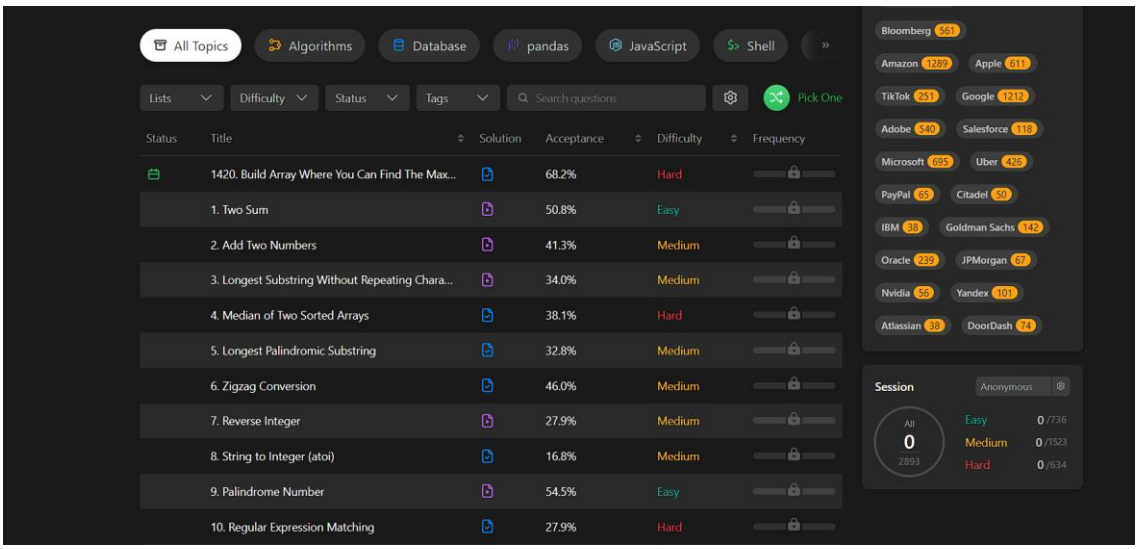


Рисунок 1.8 - Список задач з LeetCode

Платформа дозволяє позмагатися з іншими учасниками або імітувати технічну співбесіду, вирішуючи проблеми протягом певного часу. Це може допомогти вам попрактикуватися в часових обмеженнях для справжньої технічної співбесіди.

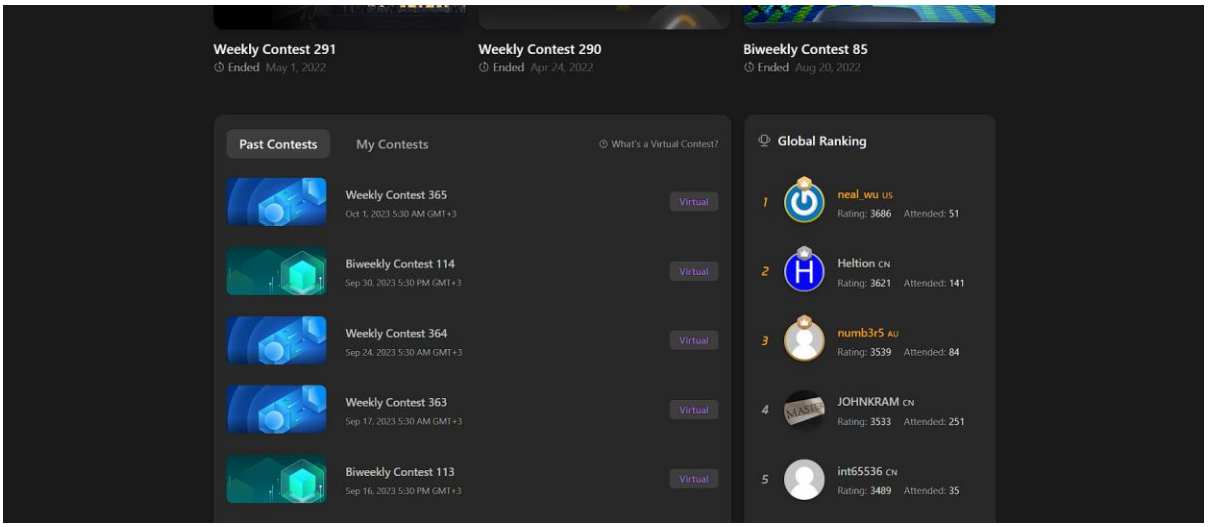


Рисунок 1.9 - Змагання в вирішені задач на LeetCode

LeetCode містить завдання, які зазвичай ставлять під час технічних співбесід у конкретних компаніях. Більшість таких завдань є платними, про те навіть у такому випадку користувачі купують підписки, оскільки якість та роз’яснення цих задач дійсно допомагають у реальному житті. Багато великих компаній звертають уваги на досягнення на цьому сайті, бо вважають його дійсним хорошим тренажером для новачків.

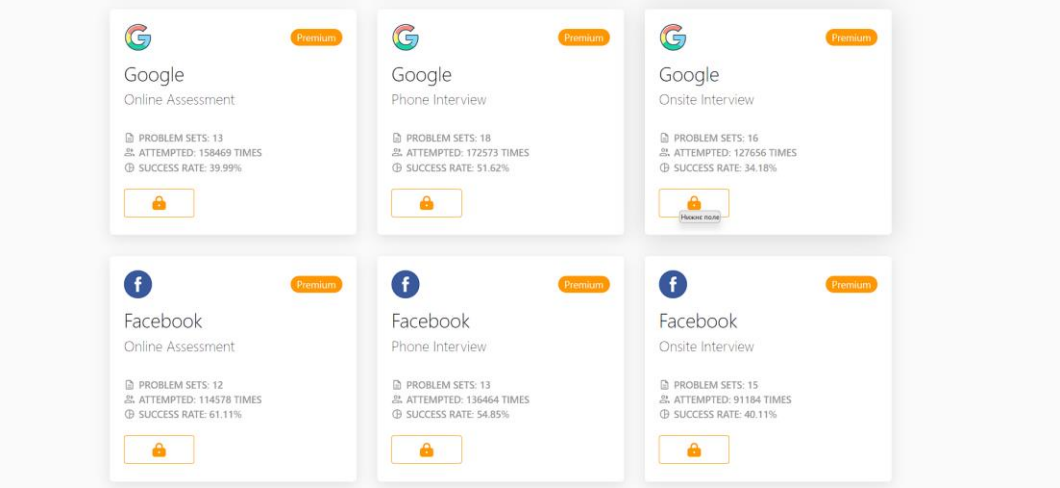


Рисунок 1.10 - Варіанти проходження технічних співбесід на LeetCode

Eolymp – це автоматизована система оцінювання. Щоразу, під час проведення змагання з програмування, викладання або вивчення алгоритму, Eolymp надає користувачу масштабовану систему оцінювання, освітню платформу та підтримку світового рівня [19].

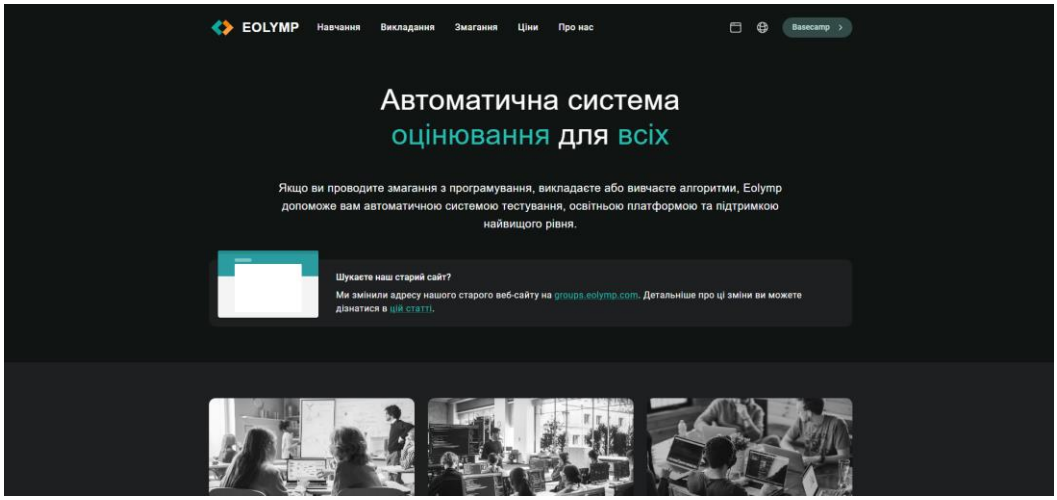


Рисунок 1.11 - Головна сторінка Eolymp

Даний сервіс надає можливість новачкам і не тільки розвинути власні навички з програмування, що дозволяє покращити алгоритмічне мислення та успішно пройти співбесіду. Сервіс користується популярністю серед університетів. Багато викладачів дають розв’язувати задачі своїм студентам саме на цьому ресурсі, щоб здобувачі освіти могли швидко отримати результати роботи здатності свого коду у різних ситуаціях.

Розробники попіклувалися, щоб перелік доступних мов для програмування був максимально широким. Веб-додаток підтримує такі мови програмування, як Java, JavaScript, C# та багато інших.

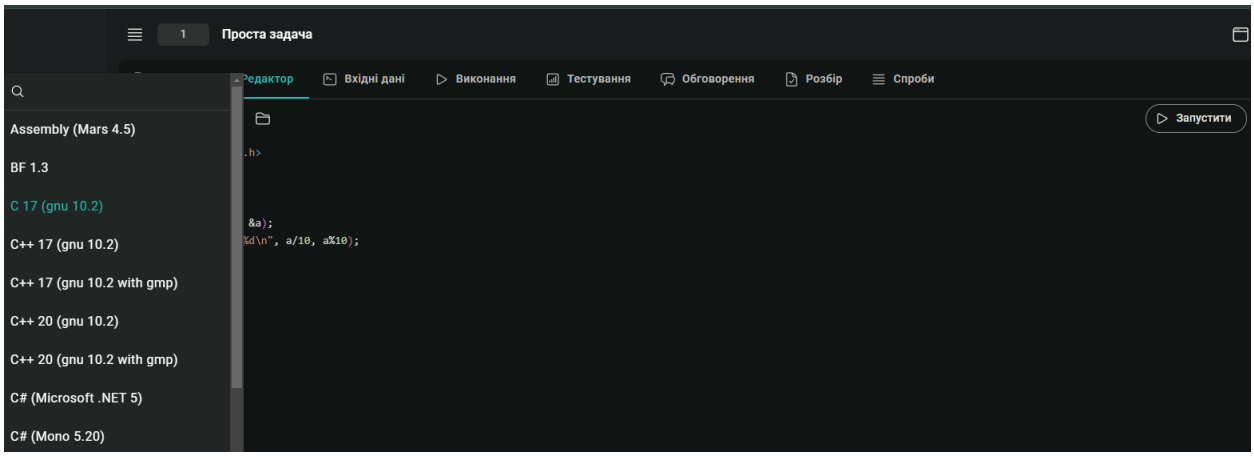


Рисунок 1.12 - Список доступних мов програмування

Ресурс також надає можливість змагання з іншими учасниками. Головними критеріями оцінювання коду є час виконання і об’єм використаної оперативної пам’яті. Також дуже важливо, щоб алгоритм був не тільки швидкий і займав мало оперативної пам’яті, а і був стійким до максимальної великої кількості тест-кейсів.

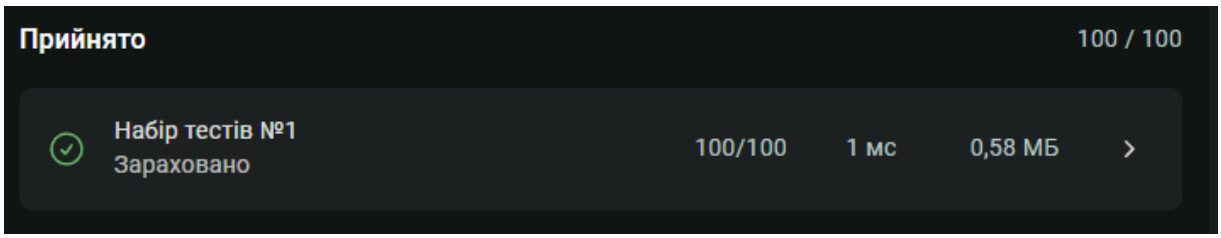


Рисунок 1.13 - Результати виконаної задачі

На жаль, не всі функції сайту є безкоштовними. Деякі змагання, які являють собою набір задач, є платними. Кількість змагання в яких можна прийняти участь напряму залежить від тарифу.

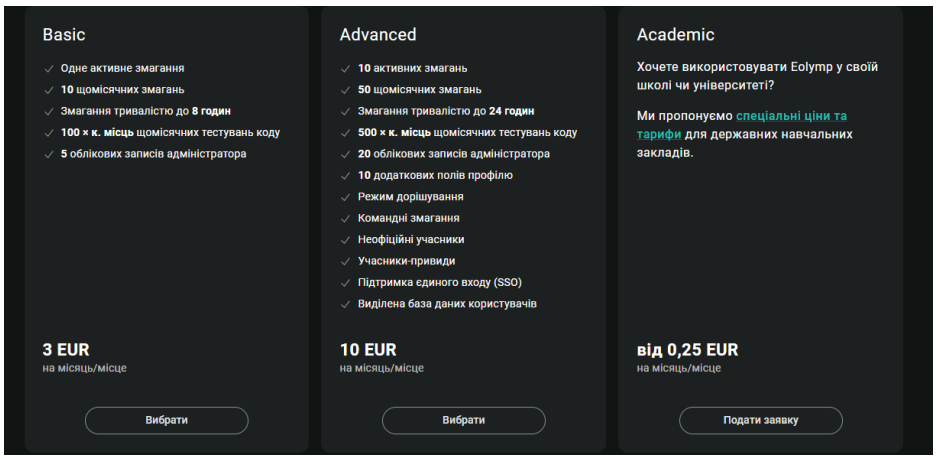


Рисунок 1.14 - тарифи Eolymp

Для створення тест-кейсів потрібно передати лише вхідні дані і очікувані результати, які є рядки або наборами рядків. Такий підхід до створення тестів, унеможливлює створення таких тестів, які глибоко зможуть перевіряти код та його структуру.

IDE One - це ще один інструмент, заснований на глибокому програмуванні та розробці програмного забезпечення. Їх онлайн-редактор підтримує підсвічування синтаксису для деяких дуже відомих мов. До них відносяться Objective-C, Java, C #, VB.NET, SQL і десятки інших. Що хорошого в їх додатку, так це те, що ви можете швидко налагоджувати безліч різних мов програмування на одній сторінці. Ви також можете зберегти вихідний код за допомогою унікальної URL-адреси для загального доступу в Інтернеті [20].

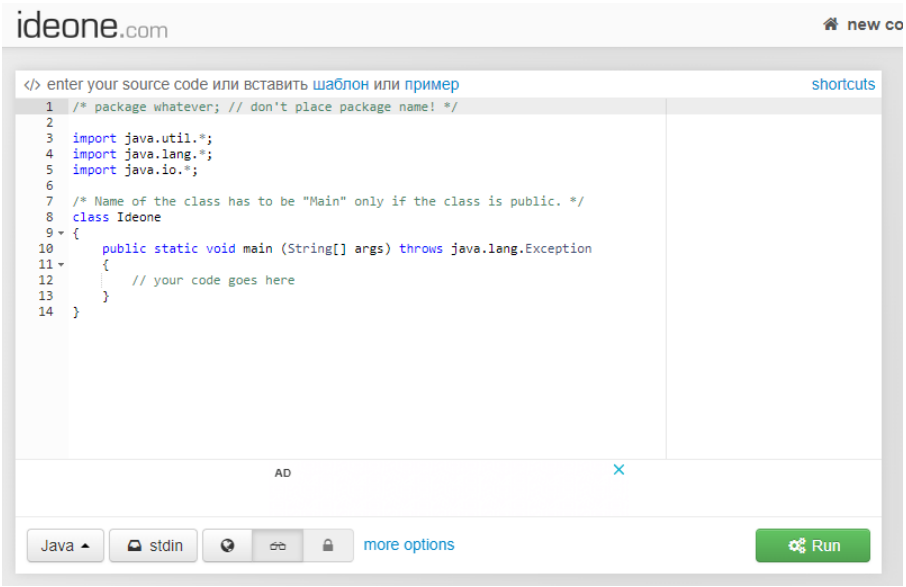


Рисунок 1.15 – Головна сторінка IDE One

Сервіс є повністю безкоштовним, про те дуже обмеженим у функціях. Користувачі не можуть створювати власні unit-тести або додавати зовнішні залежності, коли потрібно протестувати код зі складною логікою.

Час виконання програми також обмежений і залежить від конфігурації. Максимальний час виконання програми становить 15 секунд.

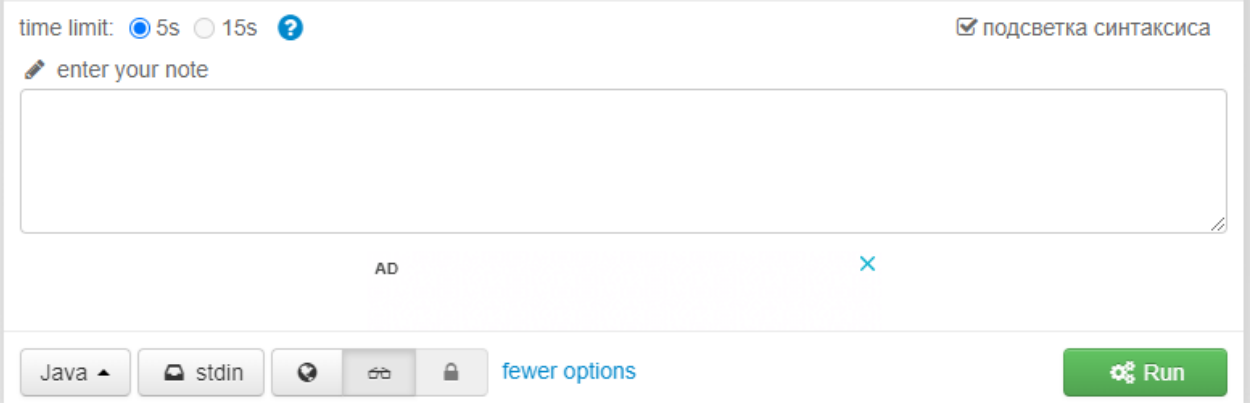


Рисунок 1.16 - Розширені налаштування сайту

Сайт є повністю безкоштовним та функціонує за рахунок реклами.

**Результати аналізу**

Таблиця 1.1 – Порівняльна таблиця

| **Назва сервісу** | **Переваги** | **Недоліки** |
|---|---|---|
| **1** | **2** | **3** |
| HackerRank | 1. Можливість підготуватися до технічної співбесіди.<br>2. Можливість перегляду рішень інших учасників.<br>3. Можливість прийняття участі в змаганнях. | 1. Високі ціни.<br>2. Перегляд деяких тест-кейсів платний. |

Продовження табл. 1.1

| 1 | 2 | 3 |
|---|---|---|
| Replit | 1. Можливість створення проекту на багатьох мовах програмування.<br>2. Можливість написання власних авто-тестів.<br>3. Можливість запрошення інших людей для роботи над проектом.<br>4. Велика кількість інтеграцій і плагінів. | 1. Високі ціни. |
| LeetCode | 1. Можливість підготуватися під конкретне інтерв'ю в певній компанії.<br>2. Можливість розвинути алгоритмічну логіку (сайт пропонує різні курси).<br>3. Можливість прийняття участі в змаганнях.<br>4. Ціни нижче ніж у конкурентів. | 1. Відсутність «intelligence» під час написання коду (Немає підказок). |

Продовження табл. 1.1

| 1 | 2 | 3 |
|---|---|---|
| Eolymp | 1. Можливість створення власних тест-кейсів.<br>2. Можливість створення власних змагань для певного кола осіб.<br>3. Велика кількість підтримуваних мов програмування, а також їх версій.<br>4. Можливість участь у безкоштовних змаганнях. | 1. Не всі змагання безкоштовні.<br>2. Безкоштовне «intelligence» дуже примітивне і майже не допомагає при розробці. |
| IDE One | 1. Повністю безкоштовний.<br>2. Можливість запустити код на 60 різних мова програмування.<br>3. Можливість відправити свій код URL-посилання. | 1. На сайті присутня реклама.<br>2. Відсутність «intelligence» під час написання коду (Немає підказок).<br>3. Відсутність можливості створення unit-тестів.<br>4. Відсутність можливості додати стороні бібліотеки. |

## 1.3. Основні поняття

Віртуалізація – це технологія, що забезпечує представлення фізичних елементів комп’ютера, таких як процесор, жорсткий диск, оперативна пам’ять та інших, на програмному рівні. Дана технології може використати реальний фізичний ресурс для створення декількох віртуальних машин, що будуть існувати: окремо один від одної та не впливати на поведінку головної (хостової) системи. Це дозволить ефективніше використовувати реальний ресурс, спростить управління та знизить витрати на інфраструктуру.

Віртуальну машину можна налаштувати самостійно, навіть до встановлення операційної системи. Це дозволяє будь-якому користувачеві вибирати обчислювальні ресурси відповідно до своїх потреб та фінансових можливостей. Вже існує ряд рішень під назвою «хостинг», які полегшують розгортання рішень. Одним з найпопулярніших рішень є Amazon Web Services (AWS), хмарна платформа, яка пропонує широкий спектр хмарних сервісів, включаючи обчислення, сховища, бази даних та засоби розробки мереж.

Віртуалізація використовується в багатьох програмних продуктах. На її основі можна створювати власні середовище для тестування коду або самі звичайні середовище для запуску коду без шкоди для системи, яка цей код запускає. Багато розробників використовують багато віртуальних машин, щоб відтворити ситуації, коли у нас є багато різних сервісів, які мають фізично знаходитися на різних машинах, проте для локального тестування якогось функціоналу потрібно відтворити той факт, що вони знаходиться на різних фізичних машинах [5, с. 3-4].

Віртуальна машина (Virtual Machine) — повна апаратна віртуалізація, яка дозволяє запускати кілька операційних систем і додатків на одному фізичному сервері. Віртуальна машина емулює апаратне забезпечення, включаючи процесор, пам’ять, диск і мережеві інтерфейси, і кожна віртуальна машина

працює як окремий ізольований екземпляр операційної системи [1]. Це дозволяє розробникам запускати різні операційні системи та програми на одному сервері без взаємного впливу та конфліктів.

Віртуальні машини симулюють операційне середовище, використовуючи технологію, яка називається «Гіпервізор (англ. Hypervisor)». Ця технологія дозволяє розподілити велику фізичну машину на кілька менших віртуальних машин, кожна з яких призначена для певного застосування. Це зробило революцію в обчислювальній техніці інфраструктури протягом майже двох десятиліть і все ще використовується сьогодні [2, с. 7].

Приклад архітектури віртуальної машини зображено на рисунку 1.17 [23].

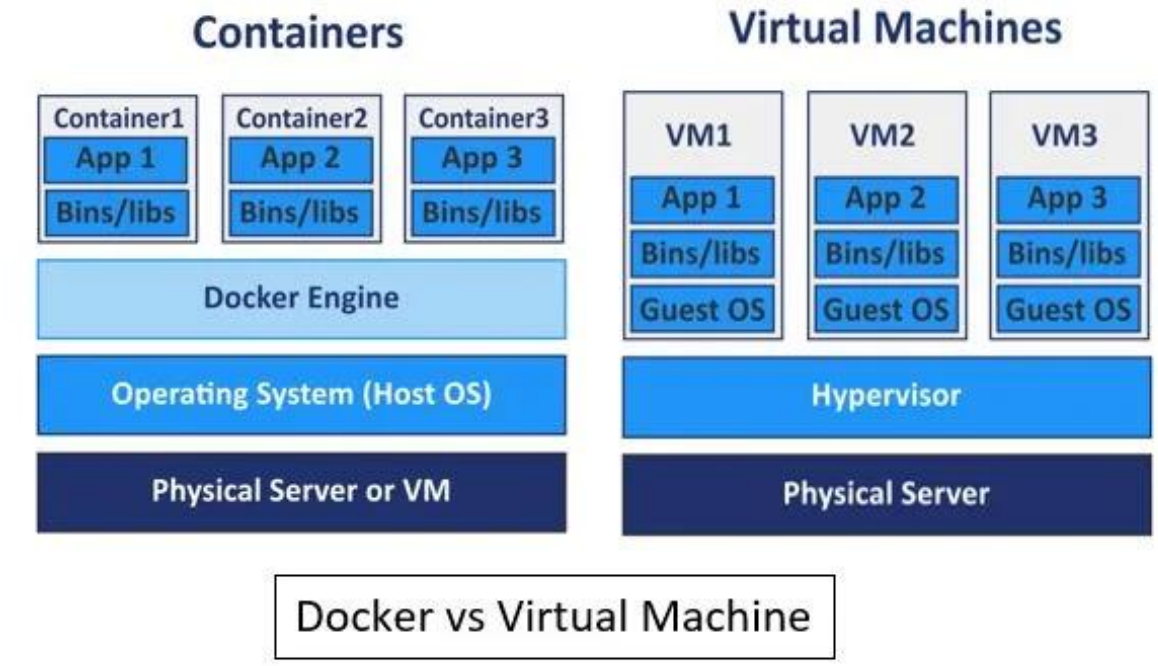


Рисунок 1.17 – Архітектура Docker контейнера та віртуальної машини

Як можна бачити на рисунку 1.17, кожна віртуальна машина має свою власну операційну систему, залежності, бібліотеки та додаток, який працює всередині віртуальної машини.

Головною ідеєю Docker є усуспільнення операційної системи. Контейнери Docker використовують концепцію операційної системи на рівні контейнера. Контейнери — це легкі та ізольовані середовища, які використовують спільне ядро операційної системи. Кожен контейнер містить усі необхідні залежності та

бібліотеки для запуску певної програми. Контейнери Docker поділяють ядро хостової операційної системи [2, с. 9-11], що дозволяє їм бути більш ефективними порівняно з віртуальними машинами. Вони також забезпечують послідовність і переносимість додатків, оскільки контейнери містять усе, що їм потрібно для роботи, і можуть працювати в будь-якій системі, яка підтримує Docker.

Як можна зрозуміти з рисунку 1.17, контейнерами керує спеціальна програма, яка називається Docker Engine. Саме вона буде надавати доступ контейнерам до ресурсів хостової операційної системи, що дозволить зменшити використання ресурсів на зберігання та обробку кожної операційної системи окремо.

Важливо додати, що існують і інші рішення, які імплементують концепцію контейнера, але вони є не настільки популярними, як Docker контейнери.

Основна відмінність між віртуальною машиною та контейнером Docker - це рівень віртуалізації. Хоча віртуальні машини імітують апаратне забезпечення та потребують додаткових ресурсів для запуску кожної операційної системи, контейнери є легкими та ефективними, оскільки вони використовують загальне ядро операційної системи. Контейнери забезпечують узгоджене та ізольоване середовище для запуску додатків, що дозволяє їм узгоджено працювати в різних обчислювальних середовищах [22, с. 19]. На жаль, у цьому рішенні є деякі підводні камені. Основна проблема цього рішення полягає в тому, що контейнери, які потребують Linux, не можуть бути запущені безпосередньо з системи Windows. Це пов'язано з тим, що Windows має значні відмінності у способі роботи ядра.

Таким чином, віртуальні машини забезпечують повну ізоляцію та гнучкість, але вимагають більше ресурсів, тоді як контейнери Docker дозволяють легко розгортати та масштабувати програми, зберігаючи ефективність використання ресурсів [3]. Вибір між цими двома підходами залежить від конкретних вимог і контексту проекту.

## 1.4. Переваги контейнеризації при тестуванні коду

У сучасному світі код який створений іншими розробниками може містити дуже багато небезпек. Взаємодія з чужим кодом може бути корисною, особливо у випадках використання бібліотек та залежностей від інших розробників, але також це може призвести до потенційних проблем із безпекою та стабільністю системи. Контейнеризація стала незамінним інструментом у таких випадках, коли важливо забезпечити ізоляцію та безпеку виконання чужого коду при цьому, коли це потрібно зробити швидко з мінімальним використанням ресурсів.

До переваг такого контейнеризації належать [4, с. 4-5]:

- **Ізоляція ресурсів**: контейнери дозволяють створювати окремі віртуальні середовища, відокремлені від хост-системи та інших контейнерів. Це забезпечує ізоляцію ресурсів і запобігає можливим конфліктам між програмами, що працюють у різних середовищах.
- **Безпека**: код третьої сторони може містити потенційно небезпечні або зловмисні фрагменти, які можуть пошкодити хост-систему або поставити під загрозу безпеку даних. Запуск цього коду в ізольованому віртуальному середовищі допомагає обмежити можливий збій або інцидент і захищає хост-систему від потенційних загроз.
- **Сумісність**: контейнери можуть працювати на різних операційних системах, незалежно від операційної системи хост-системи. Це дозволяє запускати код, створений для інших платформ, без необхідності змінювати базову операційну систему.
- **Легка реконфігурація**: контейнери дозволяють швидко створювати та копіювати нові контейнери або створювати резервні копії стану існуючих. Це дозволяє відновити середовище до попереднього стану в разі проблем або видалення шкідливого коду.
- **Тестування та налагодження**: запуск чужого коду в ізольованому віртуальному середовищі спрощує процес тестування та налагодження,

оскільки ви можете аналізувати та впливати на виконання програми, не впливаючи на хост-систему.

## 1.5. Постановка задачі

Метою роботи є розробка програмного забезпечення для тестування додатків з підвищеними вимогами до безпеки. Дане дослідження виконується в рамках в рамках кваліфікаційної роботи бакалавра за спеціальністю 122 «Комп’ютерні науки».

Програмний продукт має задовольняти наступні вимоги:

- Компілювати та запускати додатки написані на мові програмування Java 17 версії.
- Додатки, які тестуються не можуть впливати на систему, де працює сам програмний продукт.
- Завантажувати зовнішні залежності з мережі інтернет.
- Запускати тестові методи за допомогою Maven.
- Формувати тестовий звіт.
- Завантаження додатків у систему через tar-архів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. виконати аналіз проблемної області та визначити актуальність;
2. провести аналіз реалізованих рішень;
3. провести порівняння методів віртуалізації для тестування коду в ізольованому середовищі;
4. виконати проектування архітектури додатку та UI компонентів;
5. розробити програмну реалізацію інформаційної системи для тестування коду з підвищеними вимогами до безпеки.

# 2. ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

## 2.1. Огляд архітектури програмного додатку

Головна проблема, яку вирішує програмний додаток, це створення Docker контейнеру, який буде в собі компілювати та виконувати тести іншого програмного продукту.

Діаграми потоків даних (DFD) широко використовуються для структурованого аналізу та проектування програмного забезпечення. Вона також широко розповсюджена у сфері ділового адміністрування. Діаграми потоків даних це візуальний інструмент для зображення логічних моделей і виражає перетворення даних в системі. DFD включає в себе механізм моделювання потоку даних. Він підтримує декомпозицію для ілюстрації деталей потоків даних і функцій. DFD не може представити інформацію про послідовність операцій. Тому вона не є методом моделювання процесів або процедур [13, с. 85].

Діаграми потоків даних (див. рис. 2.1 та 2.2) є ефективним засобом візуалізації взаємодії компонентів програми, що дозволяє розуміти та оптимізувати роботу системи з точки зору ефективності та продуктивності. Основні принципи перетворення даних, викладені на рисунку 2.1, включають процес, у якому користувач надсилає tar-архів, що містить програмний додаток, що підлягає тестуванню, разом з тестовими методами. Після цього програма створює середовище для тестування і надсилає програмний додаток для проведення тестів разом з параметрами для створення контейнера. В результаті цього процесу отримується xml-файли з результатами тестування, які піддаються додатковій обробці даних. Після цієї обробки користувач отримує відформатований тестовий звіт.


Рисунок 2.1 DFD діаграма 0 рівня

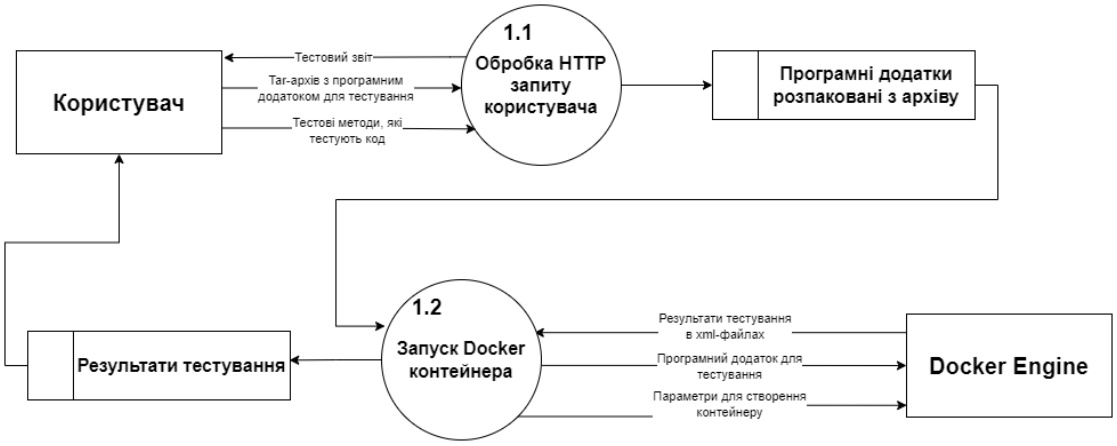


Рисунок 2.2 DFD діаграма 1 рівня

На рисунку 2.2 зображена DFD, яка має два сховище : «Результати тестування» та «Програмні додатки розпаковані з архіву». Перше сховище зберігає результати тестування, а саме який тест був виконаний та який результат роботи цього тестового методу, також може зберігати мета інформацію, яка була надана Docker Engine. Друге середовище зберігає розпаковані архіви програмних продуктів, які потрібно перевірити. Воно потрібно щоб зручно вносити інформацію в Docker контейнер. Всі програмні реалізації, що тестуються, повинні бути попередньо заархівовані. Для запуску Docker контейнера потрібно програмний додаток для тестування.

**Модель «сутність-зв'язок»** (або ER-модель) описує взаємопов'язані об'єкти, що представляють інтерес у певній галузі знань. Базова ER-модель складається з типів сутностей, які класифікують об'єкти, і визначає зв'язки, які

можуть існувати між сутностями (екземплярами цих типів сутностей) [6]. Поширеним методом візуалізації ER-діаграм є UML нотація.

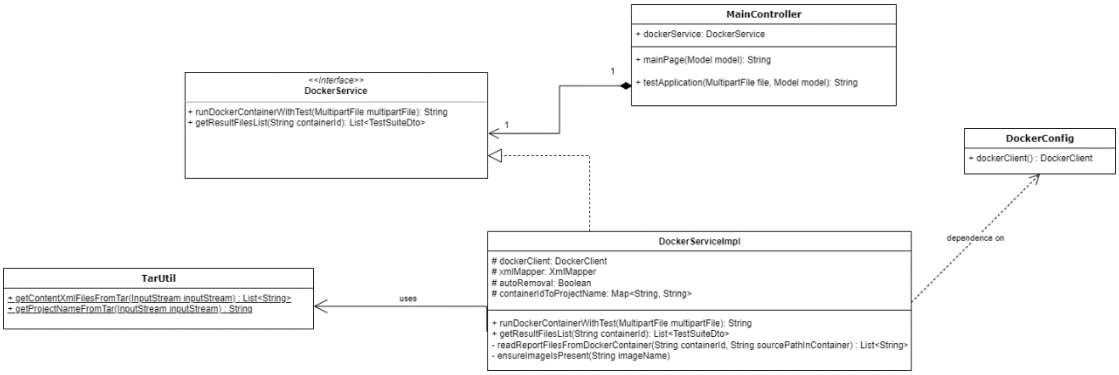


Рисунок 2.3 Entity Relationship діаграма

**DockerServiceImpl:** Цей клас є основним сервісом, який використовується для взаємодії з Docker. Він містить методи для створення та запуску Docker контейнерів, копіювання архівів до контейнера, отримання результатів з контейнера та переконання, що потрібний Docker образ присутній. Він використовує DockerClient для взаємодії з Docker API та TarUtil для обробки tar-архівів. Сучасна ідеологія Spring додатків базується на тому, що всі екземпляри класів мають належати Spring Context, де будуть створюватися і залежності від конфігурації зберігатися, тому DockerServiceImpl має залежність «dependence on» DockerConfig, оскільки саме DockerConfig створює DockerClient, який необхідний для роботи DockerServiceImpl. Також DockerServiceImpl відповідає за очікування завершення роботи контейнера та отримання результатів тестування з контейнера.

**MainController:** Цей клас є контролером Spring MVC, який обробляє HTTP-запити. Він використовує DockerService для запуску Docker контейнера з тестами та отримання результатів тестування. MainController рендерить html-сторінки в браузері за допомогою Thymeleaf. Thymeleaf - це серверний движок шаблонів Java, який можна використовувати як у веб-середовищі, так і в автономному режимі. В основному використовується для HTML, але також може

бути використаний для XML, JavaScript, CSS або простого тексту [5, с. 15]. Рисунок 2.4 показує, як працює рендеринг на стороні сервера:

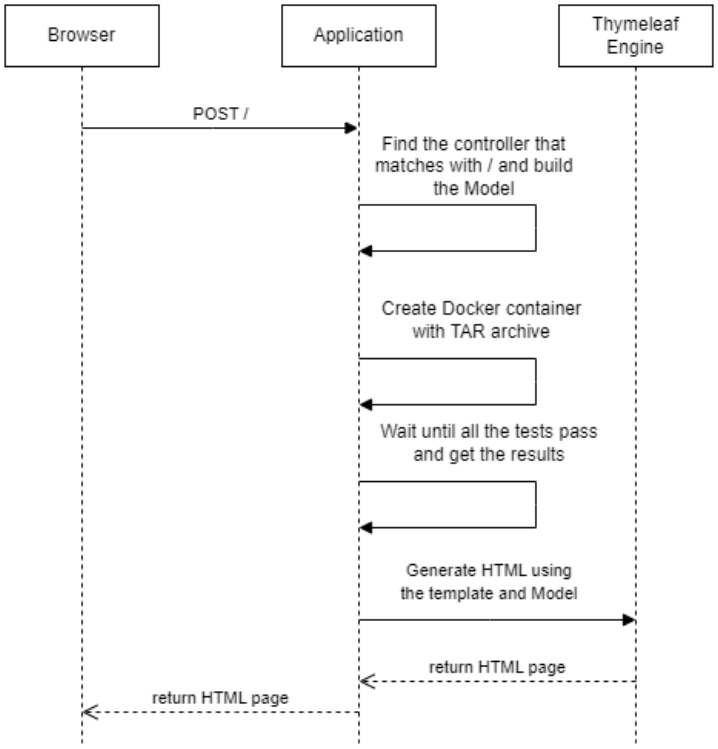


Рисунок 2.4 - Рендеринг HTML сторінок за допомогою Thymeleaf

**TarUtil:** Цей клас є утилітою, яка використовується для обробки tar-архівів. Він містить метод для отримання вмісту з tar-архіву. Цей клас використовується DockerServiceImpl для обробки tar-архівів, які отримані з Docker контейнера, оскільки для того, щоб дістати інформацію з Docker контейнера Java Docker API Client архівує дані в tar-архівів. TarUtil використовує одну з найпопулярніших бібліотек для роботи з tar-архівами, а саме Apache Commons Compress. Apache Commons Compress надає API для багатьох форматів архівних файлів. Існує три типи форматів архівних файлів: лише архівація, лише стиснення, архівація та стиснення. У Commons Compress архівація здійснюється за допомогою архіваторів. Архіватори працюють з архівами, що містять структурований вміст. Як архіватори підтримуються формати файлів: ar, cpio, dump, tar та zip [8].

**DockerConfig:** Цей клас є конфігурацією Spring, яка визначає DockerClient як Spring Bean, що створюється фабричним методом dockerClient. Фабричний

метод - це метод з анотацією @Bean, який буде використовуватися для реєстрації бінів у контексті програми. За замовчуванням, ім'я біна збігається з ім'ям методу [9, с. 19]. Цей клас відповідає за створення та конфігурацію DockerClient, який використовується DockerServiceImpl для взаємодії з Docker API.

Це є основні класи, які представлені в роботі даної програми, але існують ще три додаткових класи DTO (Data transfer object), вони не містять в собі жодної бізнес логіки і використовуються лише з метою легкого парсингу xml-файлів з результатами тестування. Поля цих класів просто змаплені з полями, які представленні в xml-файлах (див. рис. 2.5).

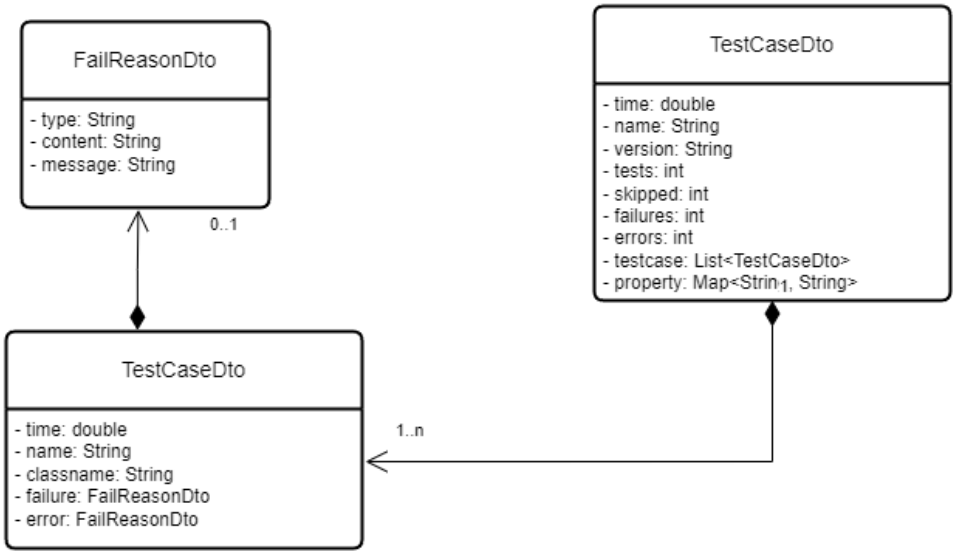


Рисунок 2.5 - ER діаграма для класів DTO

## 2.2. UI компоненти

Для розробки дизайну веб-додатку використано Figma. Даний сервіс дозволяє швидко розробляти і впроваджувати макети різних додатків.

Першим кроком до створення будь-якого додатку є розроблення саме макету цього додатку, оскільки його розробка значно простіша і, отже, дешевша, що дозволяє заощадити час та гроші замовника, а також вирішити конфлікти візії продукту ще на початкових етапах його створення.

Правильне налаштування UI компонентів дуже важливе для сучасних додатків, оскільки саме за допомогою них користувач взаємодіє з сервісом. При

неправильному налаштуванні у користувача можуть виникати певні незручності, що врешті-решт змусить його шукати альтернативи.

Для бакалаврської роботи були використані наступні 3 фрейми макету (рисунок 2.6, 2.8 та 2.9).

На рисунку 2.6 зображено стартову сторінку додатку на ній представлені два графічні елементи: поле для надсилання файлів на сервер для перевірки та привітання з описом того, що система готова до використання.

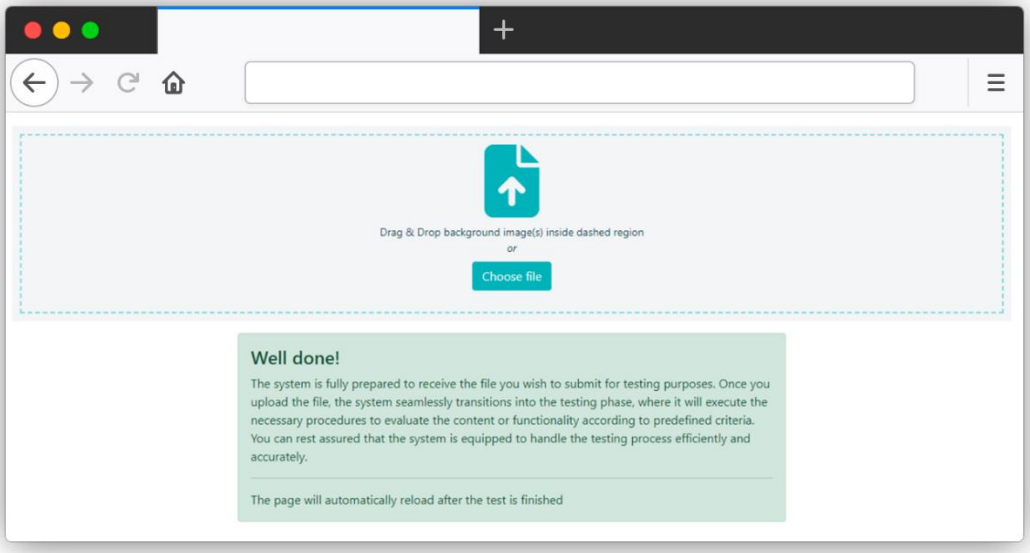


Рисунок 2.6 – Стартова сторінка додатку

При натисканні на «Choose file» з’являється модальне вікно в якому потрібно обрати шлях до tar-архіву (рисунок 2.7):

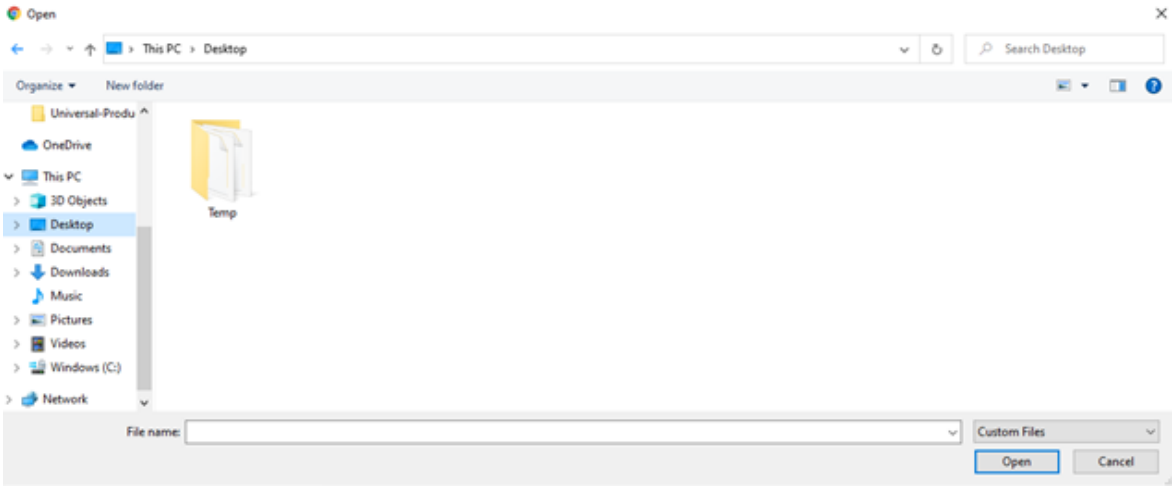


Рисунок 2.7 – Модальне вікно для завантаження tar-архіву

На рисунку 2.8 зображено додаток після тестування коду користувача. Після завершення тестування користувача перенаправляє на сторінку, де він має змогу побачити тестовий звіт. Тестовий звіт розділяється на класи, які в свою чергу поділяються на методи. Користувач має змогу переглядати детальну інформацію за кожним пунктом - просто натискаючи кнопку на той клас або метод, який його цікавить.

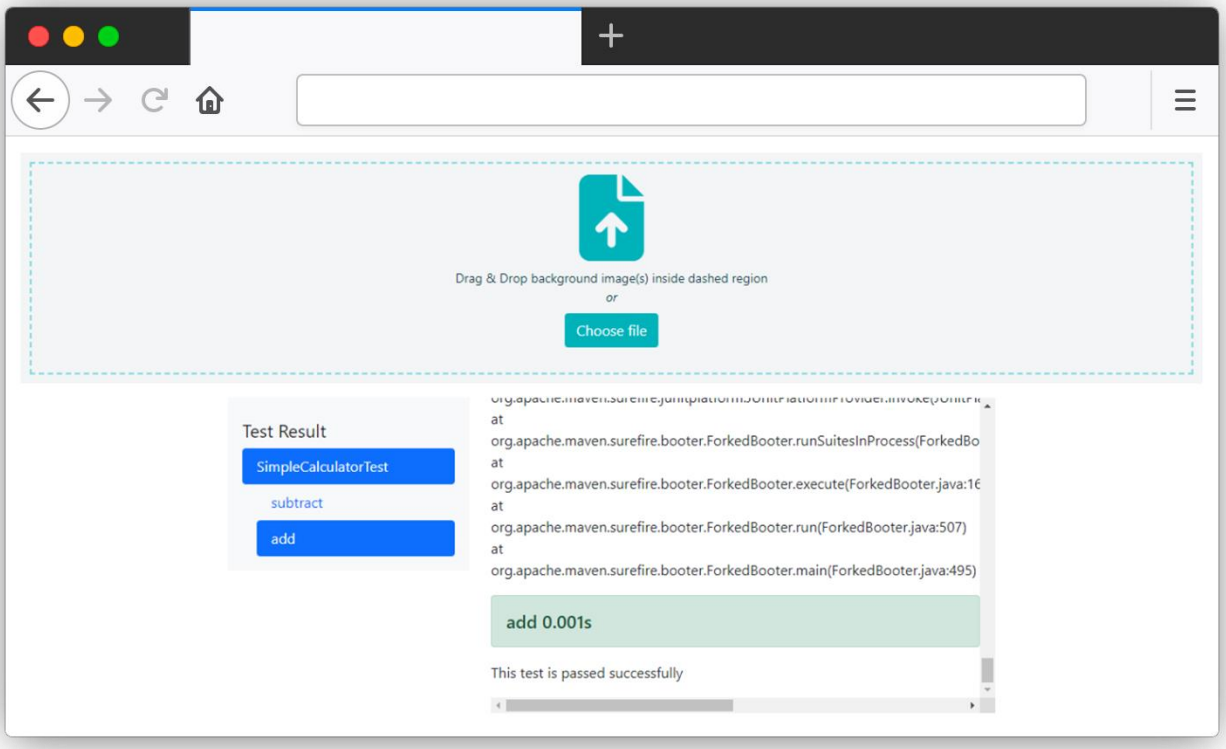


Рисунок 2.8 – Сторінка додатку після тестування

Аналізуючи дані кожного методу, можна виявити розподіл методів, які були успішними та ті, що зазнали невдачі, зафарбованими відповідними кольорами. Методи, що пройшли успішно, позначаються зеленим кольором, тоді як невдалим присвоюється червоний колір. Такий підхід сприяє покращенню користувацького досвіду використання додатку.

Для кожного методу представлена інформація:

- Клас до якого належить тестовий метод.
- Ім’я тестового методу.
- Час виконання тестового методу.

Для методів, що зазнали невдачі відображається ще додаткова інформація:

- Трасування виклику.
- Повідомлення про причину помилки.

На рисунку 2.9 зображено додаток користувача, який система не змогла повністю коректно перевірити. Наприклад, у випадку, коли вірусний код намагався взяти під контроль середовище виконання, але в нього це не вдалося і контейнер припинив свою роботу.

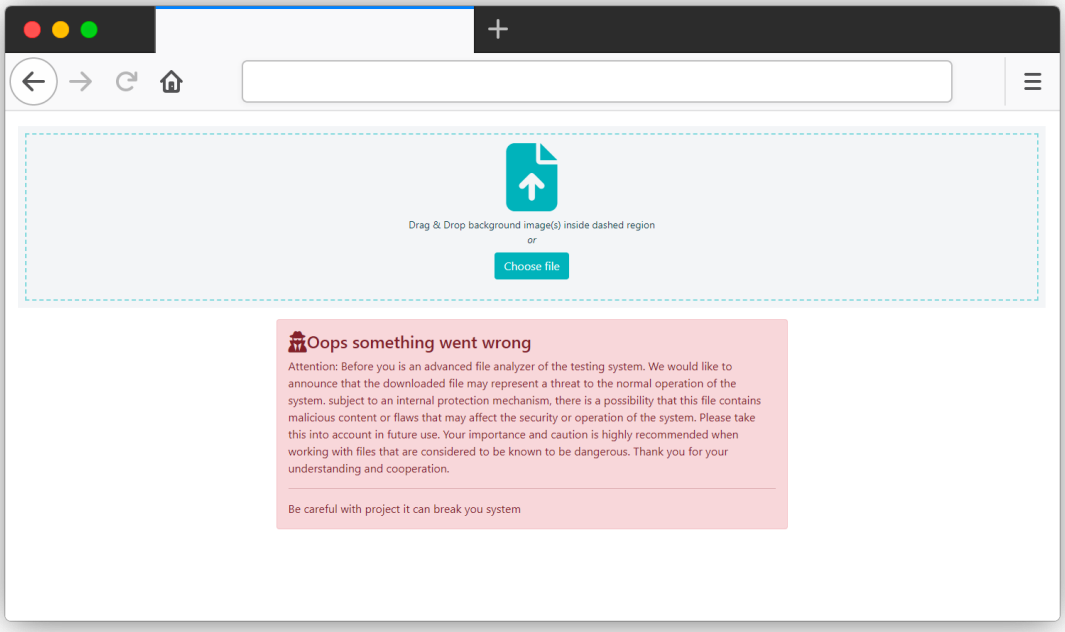


Рисунок 2.9 – Макет додатку при шкідливому коді користувача

# 3. ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

## 3.1. Налаштування Docker

Першим кроком до створення повноцінного Docker середовища є встановлення Docker. Для завантаження файлу з Docker-ом потрібно відкрити офіційний сайт Docker https://www.docker.com/get-started/ та натиснути кнопку скачати «Download for Windows». Docker можна інсталювати і на інші операційні системи, крім Windows.

Також потрібно інсталювати WSL (підсистема Windows для Linux), оскільки спочатку свого створення Docker не передбачав можливості роботи на операційних системах Windows, тому раніше створювали віртуальні машини Linux на операційних системах Windows, щоб мати можливість якось працювати з Docker. Будь-який комп'ютер з Windows, на якому працює WSL 2 також може запускати контейнери Linux. Це робить Windows 10 і 11 чудовими платформами для розробки і тестування контейнерів Windows і Linux [10, с. 9].

Для інсталяції WSL офіційна документація рекомендує виконати наступну команду в режимі адміністратора в PowerShell [11]:

```
wsl --install
```

Після успішного завантаження файлу з сайту та інсталяції WSL, необхідно інсталювати Docker на комп’ютері. Під час інсталяції Docker-а потрібно буде погодитися з налаштуванням WSL 2, що дозволяє використовувати Docker-у Windows Subsystem for Linux.

Після інсталяції Docker необхідно перезавантажити комп’ютер.

Перевірити працездатність Docker можна за допомогою команди:

```
docker run hello-world
```

## 3.2. Аналіз тестового звіту

Тестовий звіт створюється плагіном Maven Surefire, інструментом, який використовується для виконання модульних тестів у проектах Apache Maven. Звіт створюється у форматі XML і містить інформацію про виконані тести, наприклад кількість тестів, час, який знадобився для виконання, а також будь-які виявлені помилки чи збої.

Елемент testsuite є кореневим елементом звіту та містить такі атрибути, як назва набору тестів, час, який знадобився для виконання тестів, а також кількість тестів, помилок і невдач. Елемент властивості містить список системних властивостей, встановлених під час виконання тесту, наприклад використовувану версію Java, операційну систему та шлях до класу.

Елемент testcase представляє окремий тестовий приклад і включає такі атрибути:

- name: ім’я тестового методу.
- classname: ім’я тестового класу яке включає в себе і ім’я пакету.
- time: час виконання в секундах.

Якщо тест провалився і не був успішно виконаний, то елемент testcase мати вкладений елемент failure з наступними атрибутами:

- message: повідомлення з описом причини падіння. Значення цього атрибута залежить від тестового методу, оскільки саме там буде описане значення в випадку падіння тесту.
- type: клас до якого належить виняток. Виняток належить до класу java.lang.Exception, який характеризує користувацькі помилки, цей клас імплементує інтерфейс java.lang.Throwable, тому всі об'єкти виключення мають спільну поведінку. Вони містять інформацію про причину виникнення виключної ситуації та місце її виникнення (номер рядка вихідного коду) [12, с. 54].

Елемент failure має вложений текст трасування виклику, який надає можливість користувачу відслідкувати, де помилка з’явилася та виклик яких методів призвів до цієї помилки.

Якщо тест провалився по причині, яка пов’язана більше з помилкою системи, а не користувача, то виникає «помилка», яка є елементом error з наступними атрибутами:

- message: повідомлення, яке описує помилку. Цей атрибут напряму залежить від тої помилка, яка виникла, оскільки там міститься інформація про її причину.
- type: клас до якого належить помилка. Цей клас має бути наслідником класу java.lang.Error, який імплементує інтерфейс java.lang.Throwable.

Елемент error, як і елемент failure, має вложений текст трасування виклику.

Також у файлові присутні елементи property. Ці елементи містять мета інформацію:

- java.version: версія Java, яка використовується для запуску тестів.
- java.vendor: Постачальник віртуальної машини Java, яка використовується для запуску тестів.
- os.name: назва операційної системи, яка використовується для запуску тестів.
- os.arch: архітектура операційної системи, що використовується для виконання тестів (наприклад, x86, amd64).
- java.class.path: шлях до класів, який використовується для запуску тестів, тобто список каталогів і файлів JAR, які містять класи та ресурси, необхідні для тестів.
- user.dir: поточний робочий каталог під час виконання тестів.
- basedir: базовий каталог проекту Maven.
- surefire.test.class.path: шлях до класу, який використовується плагіном Surefire для запуску тестів.

## 3.3. Опис програмної реалізації

Задля забезпечення виконання вимог до розробки програмного продукту були використані наступні технології:

- Spring Boot - це мікрофреймворк з відкритим вихідним кодом від Pivotal. Це фреймворк корпоративного рівня для розробників для створення автономних додатків на віртуальних машинах Java (JVM). Його основна мета - скоротити код, щоб вам було легше запускати ваш додаток [14, с. 4].
- Apache Maven - це інструмент для управління та розуміння програмних проектів. Заснований на концепції об'єктної моделі проекту (POM), Maven може керувати збіркою, звітністю та документацією проекту з центральної частини інформації [15].
- Docker - це програмне забезпечення, яке дозволяє автоматизувати процеси розгортання та управління програмними контейнерами. Docker контейнери інкапсулюють в собі програмний код, залежності, а також середовище виконання.
- Apache Commons Compress - це бібліотека для мови програмування Java. Вона була розроблена Apache Software Foundation та надає можливості стиснення і розпакування різних форматів архівів. Бібліотека підтримує багато форматів архівів: ZIP, TAR, GZIP, BZIP2, XZ.
- Thymeleaf - це бібліотека Java і шаблонний генератор, який використовується для розбору і відображення даних, що генеруються додатком, у файли шаблонів - таким чином забезпечуючи трансформацію. Генератор подібний до HTML, але має більше атрибутів для роботи з даними, що рендеряться. Thymeleaf дозволяє кешувати дані/файл, що аналізуються, для підвищення ефективності під час виробництва. Типи шаблонів, які він може обробляти: HTML, JAVASCRIPT, CSS, XML, TEXT і RAW [16].

В рамках реалізації створено клас Main, який буде початковою точкою для запуску всього Java додатку. В цьому класі розміщується функція main, яка приймає в якості параметру масив аргументів. Тіло цієї функції складається лише з строчки коду, яка запускає увесь Spring Boot додаток.

Також клас Main служить, як і конфігураційний клас. Він створює клас XmlMapper, який необхідний для аналізу тестових звітів з xml-файлів.

Створено клас DockerConfig з методом dockerClient. Анотація @Bean на методом означає, що метод буде фабричним і буде створювати екземпляр класу DockerClient, коли такий буде потрібний Spring Context. Фабричні методи в Spring анотуються за допомогою @Bean. Методи з анотацією @Bean викликаються автоматично, і вони повертають саме той бін (екземпляр класу під управлінням Spring), який потрапляє в контекст як керований Spring-Bean [18, с. 156].

Основним завдання класу MainController є обробка HTTP-запитів, які поступають від клієнту на сервер. Для цього у Spring MVC є спеціальні анотації, які можуть викликати методи над якими вони розміщені. Так при виклику HTTP-запиту з методом «GET» буде викликаний метод mainPage. Після виклику цього методу клієнт отримувати відповідь у вигляді HTML-сторінки на якій буде форма для надсилання tar-архіву з кодом, який потрібно протестувати, а також привітання користувача і повідомлення про те, що інформаційна система готова до використання. Інший метод класу буде приймати «POST» запит в яких у тілові буде файл з tar-архів. Після виклику такого методу буде викликаний dockerService, який спочатку запустить контейнер на виконання, потім буде очікувати завершення роботи контейнера, щоб дістати результати тестування, після чого клієнт отримує у відповіді HTML-сторінку, де буде розміщуватися, ще одне поле для надсилання нового файлу на тестування, а також тестовий звіт.

Клас DockerServiceImpl реалізує інтерфейс DockerService. Головною метою цього класу є створення Docker контейнерів, а також отримання

результатів з нього. Цей клас помічений анотацією @Service, яка вказує на те, що Spring має створити екземпляр даного класу і використовувати його.

Для реалізації вище зазначеного функціоналу було розроблено 4 методи:

1. runDockerContainerWithTest(MultipartFile multipartFile): Цей метод запускає Docker контейнер з тестами. Метод приймає файл у форматі MultipartFile як вхідний параметр. Спочатку метод переконується, що потрібний Docker образ присутній, потім створює Docker контейнер з використанням цього образу та параметрами для запуску тестів. Після цього метод копіює вхідний файл до контейнера та запускає контейнер. Метод повертає ID створеного контейнера.
2. getResultTestSuite(String containerId): Цей метод отримує список файлів з результатами тестування. Він приймає ID контейнера та назву проекту як вхідні параметри. Спочатку метод очікує завершення роботи контейнера, потім отримує вміст папки з результатами тестування з контейнера та перетворює ці файли у Java об’єкти для більш зручної подальшої роботи.
3. readReportFilesFromDockerContainer(String containerId, String sourcePathInContainer): Цей приватний метод читає файли звітів з Docker контейнера. Метод приймає ID контейнера та шлях до файлів у контейнері як вхідні параметри. Спочатку метод перевіряє, чи існує контейнер, потім отримує вміст папки з контейнера та повертає список XML-файлів з цієї папки.
4. ensureImageIsPresent(String imageName): Цей приватний метод переконується, що потрібний Docker образ присутній. Метод приймає назву образу як вхідний параметр. Спочатку метод перевіряє, чи існує образ. Якщо образ не існує, він завантажує образ.

Перші два методи помічені анотацією @SneakyThrows, що означають що усі виключення, які будуть викидатися під час роботи цього методу будуть обгортатися в клас RuntimeException. Такий підхід спрощує написання коду, оскільки не потрібно перехоплювати «checked exception» на вищих рівнях стеку.

Інтерфейс DockerService має на меті описати головні методи, які обов’язково має включати клас, який реалізовує цей інтерфейс. В фреймворку Spring – це правило хорошого тону, коли кожен сервіс має інтерфейс, який реалізовує.

Створення класу TarUtil має на меті відокремити логіку роботи з tar-архівами в окремий клас. TarUtil має 2 статичні методи:

1. getProjectNameFromTar(InputStream inputStream): цей метод приймає вхідний потік з tar-архівом, після чого починає аналізувати цей архів. Даний метод знаходить першу директорію у цьому архіві та повертає її назву. Потім ця директорія буде вважатися тією, в якій зберігається проект який потрібно протестувати.
2. getContentXmlFilesFromTar(InputStream inputStream): метод, який аналізує директорію «target/surefire-reports» та збирає вміст усіх рядків файлу в один великий, щоб потім покласти в колекцію та повернути її як результат. Даний підхід дозволяє швидко дістати всю необхідну інформацію про результати тестування з Docker контейнеру.

## Висновки

В результаті виконання роботи було створено інформаційну систему автоматизованого тестування з підвищеними вимогами до безпеки. Було розглянуто декілька реалізованих рішень на основі, яких була створена інформаційна система. Також під час розробки проекту було враховано усі вимоги до програмного продукту.

У ході виконання кваліфікаційної роботи бакалавра було виконано наступні завдання:

1. Виконано аналіз проблемної області та визначено потенційні проблеми, з якими може зустрітися користувач при запуску невідомого коду.
2. Проведено аналіз HackerRank, Replit, LeetCode, Eolymp і IDE One та створено порівняльну таблицю. Таблиця містить переваги та недоліки по кожному з реалізованих рішень.
3. Проведено ґрунтовний аналіз літератури з тематики віртуалізації.
4. Проведено порівняння методів віртуалізації для тестування коду. Виділено основні переваги, які надає контейнеризація додатків, а саме ізоляція ресурсів, безпека, сумісність, легка реконфігурація, тестування та налагодження;
5. Виконано проектування архітектури та UI компонентів додатку. Розроблено ER-діаграму в нотації UML, яка демонструє взаємодію класів всередині системи. Створено три макети додатку, що відображають його стан під час запуску, після завершення тестування, а також у випадку виникнення помилок, які можуть з'явитися в процесі тестування.
6. Розроблено програмну реалізацію інформаційної системи для тестування коду з підвищеними вимогами до безпеки, використовуючи наступні технології: Java, Spring Boot, Docker, Maven, Thymeleaf;

За результатами роботи інформаційної системи, можна зробити висновок, що розроблена система може спростити процес тестування для Java-розробників, а також захистити від проблем пов’язаних з безпекою.

## Список використаних джерел


1. Віртуальні машини [Електронний ресурс] // КомпБест – інтернет-магазин брендових ПК з Європи. – Режим доступу: https://compbest.com.ua/ua/virtualnye-mashiny/#:~:text=Віртуальна%20машина%20(Virtual%20Machine)%20-,Д (дата звернення: 01.05.2024). – Назва з екрана.
2. Agarwal G. Modern devops practices: implement and secure devops in the public cloud with cutting-edge tools, tips, tricks, and techniques / Gaurav Agarwal. – [Б. м.] : Packt Publishing, Limited, 2021. – 530 с.
3. TI40: Відмінності між віртуальними машинами та контейнерами [Електронний ресурс] // Школа автоматики. – Режим доступу: http://edu.asu.in.ua/mod/page/view.php?id=127 (дата звернення: 03.05.2024). – Назва з екрана.
4. Telang T. Beginning cloud native development with microprofile, jakarta EE, and kubernetes [Електронний ресурс] / Tarun Telang. – Berkeley, CA : Apress, 2023. – Режим доступу: https://doi.org/10.1007/978-1-4842-8832-0 (дата звернення: 03.05.2024). – Назва з екрана.
5. Luksa M. Kubernetes in action / Marko Luksa. – [Б. м.] : Manning Publications, 2018. – 624 с.
6. Contributors to Wikimedia projects. Entity–relationship model - Wikipedia [Електронний ресурс] / Contributors to Wikimedia projects // Wikipedia, the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Entity–relationship_model (дата звернення: 05.05.2024). – Назва з екрана.
7. Deblauwe W. Taming thymeleaf: practical guide to building a webapplication with spring boot and thymeleaf / Wim Deblauwe. – [Б. м.] : Lulu Press, Inc., 2021.
8. Chou C.-H. Java tools for developers : навч. посіб. / Cheng-Hung Chou. – 2-ге вид. – [Б. м. : б. в.], 2016. – 411 с.

9. Deinum M. Spring boot 3 recipes [Електронний ресурс] / Marten Deinum. – Berkeley, CA : Apress, 2024. – Режим доступу: https://doi.org/10.1007/979-8-8688-0113-6 (дата звернення: 06.05.2024). – Назва з екрана.
10. Poulton N. Docker deep dive: zero to docker in a single book (mastering containers 1) : підручник / Nigel Poulton. – [Б. м. : б. в.], 2016. – 375 с.
11. Install WSL [Електронний ресурс] // Microsoft Learn: Build skills that open doors in your career. – Режим доступу: https://learn.microsoft.com/en-us/windows/wsl/install (дата звернення: 10.05.2024). – Назва з екрана.
12. Samoylov N. Learn java 17 programming: learn the fundamentals of java programming with this updated guide with the latest features / Nick Samoylov. – [Б. м.] : Packt Publishing, Limited, 2022.
13. Li Q. Modeling and Analysis of Enterprise and Information Systems [Електронний ресурс] / Qing Li, Yu-Liu Chen. – Berlin, Heidelberg : Springer Berlin Heidelberg, 2009. – Режим доступу: https://doi.org/10.1007/978-3-540-89556-5 (дата звернення: 12.05.2024). – Назва з екрана.
14. Villafranca S. Full-Stack web development with spring boot and angular: a practical guide to building your full-stack web application / Seiji Villafranca, Devlin Basilan Duldulao. – [Б. м.] : Packt Publishing, Limited, 2022.
15. Maven - welcome to apache maven [Електронний ресурс] // Maven - Welcome to Apache Maven. – Режим доступу: https://maven.apache.org (дата звернення: 15.05.2024). – Назва з екрана.
16. Spring boot - how thymeleaf works? - geeksforgeeks [Електронний ресурс] // GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/spring-boot-how-thymeleaf-works/ (дата звернення: 16.05.2024). – Назва з екрана.
17. Carnell J. Spring Microservices in Action / John Carnell. – [Б. м.] : Manning Publications, 2017. – 384 с.
18. Ullenboom C. Spring boot 3 and spring framework 6 / Christian Ullenboom. – [Б. м.] : Rheinwerk Computing, 2023. – 934 с.

19. Eolymp. – Режим доступу: https://eolymp.com/ (дата звернення: 17.05.2024). – Назва з екрана.
20. Rud A. 10 веб-сайтів для тестування коду онлайн | блог hyperhost.ua [Електронний ресурс] / Alla Rud // Український хостинг провайдер HyperHost. – Режим доступу: https://hyperhost.ua/info/uk/10-veb-saytiv-dlya-testuvannya-kodu-onlayn (дата звернення: 17.05.2024). – Назва з екрана.
21. Бондар В. Перевірка додатків із підвищеними вимогами до безпеки / Владислав Бондар // Інформатика, математика, автоматика : Міжнар. наук. конф, Суми, 22 квіт. 2024 р. – Суми, 2024. – С. 104–105.
22. Якименко І. О. Інформаційне та програмне забезпечення системи тестового контролю знань: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 122 - комп’ютерні науки [Бакалаврська робота] / І. О. Якименко ; наук. кер. І. В. Шелехов. — Суми: Сумський державний університет, 2023. — 117
23. Docker vs Virtual Machine: what are the differences [Електронний ресурс] // UrClouds. – Режим доступу: https://urclouds.com/2020/07/04/docker-vs-virtual-machine-what-are-the-differences/ (дата звернення: 18.05.2024). – Назва з екрана.

# Додаток А. Програмна реалізація

## MainController:

```
import lombok.RequiredArgsConstructor;
import org.example.dto.TestSuiteDto;
import org.example.service.DockerService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.multipart.MultipartFile;

import java.util.Collections;
import java.util.List;

@Controller // Анотація, яка вказує, що цей клас є контролером Spring MVC
@RequiredArgsConstructor // Анотація Lombok, яка генерує конструктор для всіх final полів
public class MainController {

    private final DockerService dockerService; // Сервіс для взаємодії з Docker

    @GetMapping // Анотація, яка вказує, що цей метод обробляє GET-запити
    public String mainPage(Model model) {
        model.addAttribute("testSuites", Collections.emptyList()); // Додаємо атрибут до моделі
        return "index"; // Повертаємо назву відображення
    }
    @PostMapping // Анотація, яка вказує, що цей метод обробляє POST-запити
    public String testApplication(MultipartFile file, Model model) {
        try {
            // Запускаємо Docker контейнер з тестами
            String containerId = dockerService.runDockerContainerWithTest(file);

            // Отримуємо список файлів з результатами тестування
            List<TestSuiteDto> testSuites = dockerService.getResultTestSuite(containerId);
            model.addAttribute("testSuites", testSuites); // Додаємо атрибут до моделі
        } catch (Exception e) {
            model.addAttribute("error"); // Додаємо атрибут до моделі
        }

        return "index"; // Повертаємо назву відображення
    }
}
```

## DockerConfig:

```
import com.github.dockerjava.api.DockerClient;
import com.github.dockerjava.core.DefaultDockerClientConfig;
import com.github.dockerjava.core.DockerClientBuilder;
import com.github.dockerjava.httpclient5.ApacheDockerHttpClient;
import com.github.dockerjava.transport.DockerHttpClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration // Анотація, яка вказує, що цей клас є джерелом бінов для контексту Spring
public class DockerConfig {
    @Bean // Анотація, яка вказує, що метод створює, налаштовує та повертає об'єкт, який потім керує Spring
    public DockerClient dockerClient() {
        // Створення конфігурації за замовчуванням для DockerClient
        DefaultDockerClientConfig config = DefaultDockerClientConfig.createDefaultConfigBuilder()
                .build();
        // Створення HTTP-клієнта для DockerClient, використовуючи конфігурацію
        DockerHttpClient httpClient = new ApacheDockerHttpClient.Builder()
                .dockerHost(config.getDockerHost()) // Встановлюємо адресу Docker хоста
                .sslConfig(config.getSSLConfig()) // Встановлюємо конфігурацію SSL
                .build();
        // Створення DockerClient, використовуючи конфігурацію та HTTP-клієнт
        return DockerClientBuilder.getInstance(config)
                .withDockerHttpClient(httpClient)
                .build();
    }
}
```

## FailReasonDto:

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlText;
import lombok.Data;
@Data
@JsonIgnoreProperties(ignoreUnknown = true)
public class FailReasonDto {
    @JacksonXmlProperty(isAttribute = true)
    private String message;
    @JacksonXmlProperty(isAttribute = true)
    private String type;
    @JacksonXmlText
    private String content;
}
```

## TestCaseDto:

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlRootElement;
import lombok.Data;

@Data
@JacksonXmlRootElement(localName = "testcase")
@JsonIgnoreProperties(ignoreUnknown = true)
public class TestCaseDto {
    @JacksonXmlProperty(isAttribute = true)
    private String name;
    @JacksonXmlProperty(isAttribute = true)
    private String classname;
    @JacksonXmlProperty(isAttribute = true)
    private double time;
    private FailReasonDto failure;
    private FailReasonDto error;

    public FailReasonDto getFailure() {
        if (failure == null) {
            return error;
        }

        return failure;
    }
}
```

## TestSuiteDto:

```
@Data
@JacksonXmlRootElement(localName = "testsuite")
@JsonIgnoreProperties(ignoreUnknown = true)
public class TestSuiteDto extends TestCaseDto {
    @JacksonXmlProperty(isAttribute = true)
    private String version;

    @JacksonXmlProperty(isAttribute = true)
    private String name;

    @JacksonXmlProperty(isAttribute = true)
    private double time;

    @JacksonXmlProperty(isAttribute = true)
    private int tests;
```

```
    @JacksonXmlProperty(isAttribute = true)
    private int errors;

    @JacksonXmlProperty(isAttribute = true)
    private int skipped;

    @JacksonXmlProperty(isAttribute = true)
    private int failures;

    @JacksonXmlElementWrapper(useWrapping = false)
    private Map<String, String> property;

    @JacksonXmlElementWrapper(useWrapping = false)
    private List<TestCaseDto> testcase;

    public String getName() {
        return name.split("\\.")[name.split("\\.").length - 1];
    }
}
```

## DockerService:

```
import org.example.dto.TestSuiteDto;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

/**
 * DockerService is an interface that defines the contract for a service that interacts with
Docker.
 * It declares two methods: runDockerContainerWithTest and getResultFilesList.
 */
public interface DockerService {

    /**
     * This method is expected to take a MultipartFile as an argument, which is a representation
of an uploaded file received in a multipart HTTP request.
     * The method is expected to run a Docker container with the test, presumably using the
content of the provided file.
     * The method returns a String, which is likely the ID of the Docker container that was
started.
     *
     * @param multipartFile The file to be tested in the Docker container.
     * @return The ID of the Docker container that was started.
     */
```

```
    String runDockerContainerWithTest(MultipartFile multipartFile);

    /**
     * This method takes two String arguments: containerId and projectName.
     * The containerId is likely the ID of a Docker container.
     * This method is expected to return a List<String>, which could be a list of file names or
paths representing the result files from the Docker container.
     *
     * @param containerId The ID of the Docker container.
     * @return A list of file names or paths representing the result files from the Docker
container.
     */
    List<TestSuiteDto> getResultTestSuite(String containerId);

}
```

## DockerServiceImpl:

```
import com.fasterxml.jackson.dataformat.xml.XmlMapper;
import com.github.dockerjava.api.DockerClient;
import com.github.dockerjava.api.command.CreateContainerCmd;
import com.github.dockerjava.api.command.InspectContainerResponse;
import com.github.dockerjava.api.command.PullImageResultCallback;
import com.github.dockerjava.api.exception.NotFoundException;
import com.github.dockerjava.api.model.HostConfig;
import com.github.dockerjava.api.model.Image;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.example.dto.TestSuiteDto;
import org.example.service.DockerService;
import org.example.util.TarUtil;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service // Анотація, яка вказує, що цей клас є компонентом Spring і може бути виявлений в
процесі сканування класів
@RequiredArgsConstructor // Анотація Lombok, яка генерує конструктор для всіх final полів
public class DockerServiceImpl implements DockerService {
```

```
    protected final DockerClient dockerClient; // Клієнт Docker, який використовується для
взаємодії з Docker
    protected final XmlMapper xmlMapper; // Об'єкт, який використовується для парсингу XML-файлів

    @Value("${docker.auto-removal:false}") // Значення, яке витягується з конфігурації Spring
    protected Boolean autoRemoval;

    protected Map<String, String> containerIdToProjectName = new HashMap<>();

    @SneakyThrows
    public String runDockerContainerWithTest(MultipartFile multipartFile) {
        String javaImageName = "maven:3.8.3-openjdk-17"; // Назва Docker образу, який
використовується для запуску тестів
        // Назва проекту, витягнута з назви вхідного файлу
        String projectName = TarUtil.getProjectNameFromTar(multipartFile.getInputStream());

        ensureImageIsPresent(javaImageName); // Переконуємося, що потрібний Docker образ
присутній

        // Створюємо Docker контейнер з використанням образу та команди для запуску тестів
        CreateContainerCmd createContainerCmd = dockerClient.createContainerCmd(javaImageName)
                .withCmd("sh", "-c",
                        "cd ../%s && ".formatted(projectName) +
                                "mvn clean test")
                .withHostConfig(HostConfig.newHostConfig()
                        .withAutoRemove(false)
                );

        // Зберігаємо ID створеного контейнера
        String containerID = createContainerCmd.exec().getId();
        containerIdToProjectName.put(containerID, projectName);

        // Копіюємо вхідний файл до контейнера
        dockerClient.copyArchiveToContainerCmd(containerID)
                .withTarInputStream(multipartFile.getInputStream())
                .exec();

        // Запускаємо контейнер
        dockerClient.startContainerCmd(containerID).exec();

        return containerID; // Повертаємо ID контейнера
    }

    @SneakyThrows
```

```
    public List<TestSuiteDto> getResultTestSuite(String containerId) {
        // Очікуємо завершення роботи контейнера
        dockerClient.waitContainerCmd(containerId).start().awaitCompletion();

        // Визначаємо шлях до файлів з результатами тестування в контейнері
        String pathToFilesInDocker = "/%s/target/surefire-
reports".formatted(containerIdToProjectName.get(containerId));
        containerIdToProjectName.remove(containerId);

        // Отримуємо список файлів з результатами тестування
        List<String> files = readReportFilesFromDockerContainer(containerId,
pathToFilesInDocker);

        // Парсимо файли з результатами тестування
        return files.stream()
                .map(file -> {
                    try {
                        return xmlMapper.readValue(file, TestSuiteDto.class);
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                })
                .toList();
    }

    private List<String> readReportFilesFromDockerContainer(String containerId, String
sourcePathInContainer) {
        // Отримуємо інформацію про контейнер
        InspectContainerResponse container =
dockerClient.inspectContainerCmd(containerId).exec();

        // Перевіряємо, чи існує контейнер
        if (container == null) {
            throw new NotFoundException(String.format("Container (ID %s) not found",
containerId));
        }

        try {
            // Отримуємо вміст папки з контейнера
            InputStream inputStream = dockerClient.copyArchiveFromContainerCmd(containerId,
sourcePathInContainer)
                    .exec();

            // Повертаємо список XML-файлів з цієї папки
            return TarUtil.getContentXmlFilesFromTar(inputStream);
```

```
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private void ensureImageIsPresent(String imageName) throws InterruptedException {
        // Отримуємо список всіх Docker образів
        List<Image> images = dockerClient.listImagesCmd().exec();

        // Перевіряємо, чи існує потрібний Docker образ
        boolean imageExists = images.stream()
                .anyMatch(image -> Arrays.asList(image.getRepoTags()).contains(imageName));

        // Якщо образ не існує, завантажуємо його
        if (!imageExists) {
            dockerClient.pullImageCmd(imageName)
                    .exec(new PullImageResultCallback())
                    .awaitCompletion();
        }
    }

}
```

**TarUtil:**

```
import org.apache.commons.compress.archivers.tar.TarArchiveEntry;
import org.apache.commons.compress.archivers.tar.TarArchiveInputStream;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

// Клас TarUtil використовується для роботи з TAR-архівами
public class TarUtil {

    // Метод для отримання назви проекту з TAR-архіву
    public static String getProjectNameFromTar(InputStream inputStream) throws IOException {
        // Створюємо TarArchiveInputStream з вхідного потоку
        try (TarArchiveInputStream tarInput = new TarArchiveInputStream(inputStream)) {
            TarArchiveEntry entry;
            // Проходимо по всіх записах в TAR-архіві
            while ((entry = tarInput.getNextTarEntry()) != null) {
                // Якщо запис є каталогом і його ім'я відповідає шаблону "^./.+$"
                if (entry.isDirectory() && entry.getName().matches("^./.+$")) {
                    // Якщо воно відповідає, повертаємо ім'я каталогу після видалення "./" з
початку та "/" з кінця
```

```
                    return entry.getName()
                            .replaceAll("^./", "")
                            .replaceAll("/$", "");
                }
            }
        }
        // Якщо не знайдено відповідного каталогу, повертаємо null
        return null;
    }

    // Метод для отримання вмісту XML-файлів з TAR-архіву
    public static List<String> getContentXmlFilesFromTar(InputStream inputStream) throws
IOException {
        // Створюємо TarArchiveInputStream з вхідного потоку
        try (TarArchiveInputStream tarInput = new TarArchiveInputStream(inputStream)) {
            TarArchiveEntry entry;
            List<String> files = new ArrayList<>();
            // Проходимо по всіх записах в TAR-архіві
            while ((entry = tarInput.getNextTarEntry()) != null) {
                // Якщо запис не є каталогом і є XML-файлом
                if (!entry.isDirectory()) {
                    String fileName = entry.getName().replace("surefire-reports/", "");
                    if(!fileName.endsWith(".xml")) {
                        continue;
                    }

                    long fileSize = entry.getSize();
                    byte[] content = new byte[(int) fileSize];
                    int bytesRead = 0;
                    // Читаємо вміст файлу в масив байтів
                    while (bytesRead < fileSize) {
                        int result = tarInput.read(content, bytesRead, (int) fileSize -
bytesRead);
                        if (result == -1) break; // end of stream reached
                        bytesRead += result;
                    }
                    // Додаємо рядок, який представляє вміст файлу, до списку файлів
                    files.add(new String(content));
                }
            }
            // Повертаємо список рядків, які представляють вміст XML-файлів з TAR-архіву
            return files;
        }
    }
}
```

## Main:

```
import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;


@SpringBootApplication

public class Main {


    public static void main(String[] args) {

        SpringApplication.run(Main.class, args);

    }


    @Bean

    public XmlMapper xmlMapper() {

        return new XmlMapper();

    }

}
```

## index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Test your application</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
          rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
          crossorigin="anonymous">
    <link rel="stylesheet"
          href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>
</head>
<style>
    :root {
```

```
    --colorPrimaryNormal: #00b3bb;
    --colorPrimaryDark: #00979f;
    --colorPrimaryGlare: #00cdd7;
    --colorPrimaryHalf: #80d9dd;
    --colorPrimaryQuarter: #bfecee;
    --colorPrimaryEighth: #dff5f7;
    --colorPrimaryPale: #f3f5f7;
    --colorPrimarySeparator: #f3f5f7;
    --colorPrimaryOutline: #dff5f7;
    --colorButtonNormal: #00b3bb;
    --colorButtonHover: #00cdd7;
    --colorLinkNormal: #00979f;
    --colorLinkHover: #00cdd7;
}

body {
    margin: 24px;
}


.upload_dropZone {
    color: #0f3c4b;
    background-color: var(--colorPrimaryPale, #c8dadf);
    outline: 2px dashed var(--colorPrimaryHalf, #c1ddef);
    outline-offset: -12px;
    transition: outline-offset 0.2s ease-out,
    outline-color 0.3s ease-in-out,
    background-color 0.2s ease-out;
}

.upload_dropZone.highlight {
    outline-offset: -4px;
    outline-color: var(--colorPrimaryNormal, #0576bd);
    background-color: var(--colorPrimaryEighth, #c8dadf);
}

.upload_svg {
    fill: var(--colorPrimaryNormal, #0576bd);
}

.btn-upload {
    color: #fff;
    background-color: var(--colorPrimaryNormal);
}
```

```
    .btn-upload:hover,
    .btn-upload:focus {
        color: #fff;
        background-color: var(--colorPrimaryGlare);
    }

    .scrollspy {
        position: relative;
        height: 350px;
        overflow: auto;
    }
</style>
<body>

<form action="/" method="post" enctype="multipart/form-data" id="upload-file-form">

    <fieldset class="upload_dropZone text-center mb-3 p-4">
        <legend class="visually-hidden">Image uploader</legend>
        <i class="fa-solid fa-file-arrow-up fa-6x" style="color: #00b3bb;"></i>
        <p class="small my-2">Drag &amp;amp; Drop background image(s) inside dashed
region<br><i>or</i></p>
        <input id="file" th:data-post-url="@{/}"
               name="file"
               class="position-absolute invisible"
               type="file" multiple accept="application/x-tar"/>
        <label class="btn btn-upload mb-3" for="file">Choose file</label>
        <div class="upload_gallery d-flex flex-wrap justify-content-center gap-3 mb-0"></div>
    </fieldset>
</form>

<div th:if="${error == null && (testSuites == null || testSuites.isEmpty())}" class="d-flex
justify-content-center">
    <div class="alert alert-success" role="alert" style="width: 720px">
        <h4 class="alert-heading">Well done!</h4>
        <p>The system is fully prepared to receive the file you wish to submit for testing
purposes. Once you upload the
            file, the system seamlessly transitions into the testing phase, where it will execute
the necessary
            procedures
            to evaluate the content or functionality according to predefined criteria. You can
rest assured that the
            system
            is equipped to handle the testing process efficiently and accurately.</p>
        <hr>
        <p class="mb-0">The page will automatically reload after the test is finished</p>
```

```
    </div>
</div>

<div th:if="${error != null}" class="d-flex justify-content-center">
    <div class="alert alert-danger" role="alert" style="width: 720px">
        <h4 class="alert-heading"><i class="fa-solid fa-user-secret fa-lg"></i>Oops something
went wrong</h4>
        <p>Attention: Before you is an advanced file analyzer of the testing system. We would
like to announce that the
            downloaded file may represent a threat to the normal operation of the system. subject
to an internal protection mechanism,
            there is a possibility that this file contains malicious content or flaws that may
affect the security or operation of the system.
            Please take this into account in future use. Your importance and caution is highly
            recommended when working with files that are considered to be known to be dangerous.
Thank you for your understanding and cooperation.</p>
        <hr>
        <p class="mb-0">Be careful with project it can break you system</p>
    </div>
</div>

<div th:if="${error == null && (testSuites != null && !testSuites.isEmpty())}" class="row d-flex
justify-content-center">
    <div class="row" style="width: 900px">
        <div class="col-4">
            <nav id="navbar-example3" class="navbar navbar-light bg-light flex-column align-
items-stretch p-3">
                <a class="navbar-brand" href="#">Test Result</a>
                <nav th:each="testSuite : ${testSuites}" class="nav nav-pills flex-column">
                    <a class="nav-link" th:href="'#item-' + ${testSuite.getName()}"
                       th:text="${testSuite.getName()}"></a>
                    <nav th:each="test : ${testSuite.getTestcase()}"
                         class="nav nav-pills flex-column ms-3">
                        <a class="nav-link" th:href="'#item-' + ${testSuite.getName()} + '-' +
${test.getName()}"
                           th:text="${test.getName()}"></a>
                    </nav>
                </nav>
            </nav>
        </div>
        <div class="col-8">
            <div data-bs-spy="scroll" data-bs-target="#navbar-example3" data-bs-offset="0"
class="scrollspy"
                 tabindex="0">
                <div th:each="testSuite : ${testSuites}" th:id="'item-' +
```

```
${testSuite.getName()}">
                    <h4 th:text="${testSuite.getName()} + ' ' + ${testSuite.getTime()} +
's'"></h4>
                    <div th:each="test : ${testSuite.getTestcase()}"
                         th:id="'item-' + ${testSuite.getName()} + '-' + ${test.getName()}">
                        <h5 th:text="${test.getName()} + ' ' + ${test.getTime()} + 's'"
                            th:class="${test.getFailure() == null ? 'alert alert-success' :
'alert alert-danger'}"
                            role="alert"></h5>
                        <p th:text="${test.getFailure() == null ? 'This test is passed
successfully' : test.getFailure().getContent()}"></p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

</body>
<script>
    console.clear();
    ('use strict');

    (function () {

        'use strict';

        // Four objects of interest: drop zones, input elements, gallery elements, and the files.
        // dataRefs = {files: [tar files], input: element ref, gallery: element ref}

        const preventDefaults = event => {
            event.preventDefault();
            event.stopPropagation();
        };

        const highlight = event =>
            event.target.classList.add('highlight');

        const unhighlight = event =>
            event.target.classList.remove('highlight');

        const getInputAndGalleryRefs = element => {
            const zone = element.closest('.upload_dropZone') || false;
            const gallery = zone.querySelector('.upload_gallery') || false;
            const input = zone.querySelector('input[type="file"]') || false;
```

```
    return {input: input, gallery: gallery};
}

const handleDrop = event => {
    const dataRefs = getInputAndGalleryRefs(event.target);
    dataRefs.files = event.dataTransfer.files;
    handleFiles(dataRefs);
}


const eventHandlers = zone => {

    const dataRefs = getInputAndGalleryRefs(zone);

    if (!dataRefs.input) return;

    // Prevent default drag behaviors
    ;['dragenter', 'dragover', 'dragleave', 'drop'].forEach(event => {
        zone.addEventListener(event, preventDefaults, false);
        document.body.addEventListener(event, preventDefaults, false);
    });

    // Highlighting drop area when item is dragged over it
    ;['dragenter', 'dragover'].forEach(event => {
        zone.addEventListener(event, highlight, false);
    });
    ;['dragleave', 'drop'].forEach(event => {
        zone.addEventListener(event, unhighlight, false);
    });

    // Handle dropped files
    zone.addEventListener('drop', handleDrop, false);

    // Handle browse selected files
    dataRefs.input.addEventListener('change', event => {
        dataRefs.files = event.target.files;
        handleFiles(dataRefs);
    }, false);

}

// Initialise ALL dropzones
const dropZones = document.querySelectorAll('.upload_dropZone');
for (const zone of dropZones) {
    eventHandlers(zone);
```

```
        }

        const isTarFile = file =>
            ['application/x-tar'].includes(file.type);


        function previewFiles(dataRefs) {
            if (!dataRefs.gallery) return;
            for (const file of dataRefs.files) {
                let reader = new FileReader();
                reader.readAsDataURL(file);
                reader.onloadend = function () {
                    let p = document.createElement('p');
                    p.innerHTML = '<i class="fas fa-file-archive"></i> ' + file.name + " (TAR
archive)";
                    dataRefs.gallery.appendChild(p);
                }
            }
        }


        // Handle both selected and dropped files
        const handleFiles = dataRefs => {

            let files = [...dataRefs.files];

            // Remove unaccepted file types
            files = files.filter(item => {
                if (!isTarFile(item)) {
                    console.log('Not a tar file, ', item.type);
                }
                return isTarFile(item) ? item : null;
            });

            if (!files.length) return;
            dataRefs.files = files;

            previewFiles(dataRefs);
            document.getElementById('upload-file-form').submit();
        }

    })();

</script>
</html>
```

**pom.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>bachelor-work</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
    </parent>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <docker.client.version>3.3.6</docker.client.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
```

```
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.32</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.github.docker-java</groupId>
            <artifactId>docker-java</artifactId>
            <version>${docker.client.version}</version>
        </dependency>
        <dependency>
            <groupId>com.github.docker-java</groupId>
            <artifactId>docker-java-transport-httpclient5</artifactId>
            <version>${docker.client.version}</version>
        </dependency>
    </dependencies>
</project>
```