МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«Мікросервіс для валідації UI-інтерфейсу»

Галузь знань 12 «Інформаційні технології»

Спеціальність 122 «Комп’ютерні науки»

Освітня програма «Комп’ютерні науки»

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 42

Рожков Ярослав Ігорович

Керівник к.т.н., доцент Лещенко О.О.

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол № 11 від 06.06.2022 р.

зав. кафедри_____ доц. Іларіонов О.Є.

Київ – 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій
Кафедра інтелектуальних технологій
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних технологій
Іларіонов О.Є.
_____
«___» _____ 2022 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Рожкову Ярославу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)
«Мікросервіс для валідації UI-інтерфейсу»
затверджена протоколом засідання кафедри від « 23 » грудня 2021 р. № 4
2. Термін здачі студентом закінченого проекту (роботи) 29 травня 2022 року
3. Вихідні дані до проекту (роботи)
Заздалегідь сформовані шаблони для порівняння, зовнішня система, що звертається до сервісу
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
Аналіз предметної області, розробка структури мікросервісу, розробка інформаційного забезпечення та програмного забезпечення системи.
5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)
Об'єкт, предмет дослідження та мета розробки (1-2 слайди), актуальність розробки мікросервісу для валідації (1-2 слайди), аналіз наявних систем валідації (2 слайди), архітектурна будова системи (1-2 слайди), інформаційне забезпечення системи (1-2 слайди), розроблений мікросервіс (2-3 слайди), результати валідації (1 слайд), висновки (1 слайд).
6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються
7. Дата видачі завдання 15 лютого 2022 року

Керівник  / Лещенко О.О./
(підпис) (ПІБ)

Завдання прийняв до виконання  / Рожков Я.І. /

(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Аналітичний огляд літератури за темою досліджень	25.01.2021 – 29.01.2021	
2	Аналіз існуючих систем валідації зображень	30.01.2021 – 03.02.2021	
3	Аналіз основних процесів предметного середовища	04.02.2021 – 08.02.2021	
4	Визначення мети роботи, об’єкта та предмету дослідження	09.02.2021 – 13.02.2021	
5	Функціональний аналіз інформаційної системи	01.03.2021 – 05.03.2021	
6	Розробка IDEF0 процесу використання мікросервісу	06.03.2021 – 10.03.2021	
7	Розробка архітектури інформаційної системи	11.03.2021 – 15.03.2021	
8	Розробка технічної архітектури системи	16.03.2021 – 20.03.2021	
9	Визначення архітектурного підкходу, який буде використовуватися в програмному додатку	21.03.2021 – 25.03.2021	
10	Аналіз інформаційних потоків	26.03.2021 – 31.03.2021	
11	Розробка інформаційного забезпечення системи	01.04.2021 – 05.04.2021	
12	Опис програмних засобів	06.04.2021 – 16.04.2021	
13	Опис структури програмного забезпечення	17.04.2021 – 27.04.2021	
14	Огляд процесу тестування	28.04.2021 – 08.04.2021	
15	Оформлення випускної кваліфікаційної роботи	31.05.2021	

Студент-дипломник (підпис) / Рожков Я.І. / (ПІБ)

Керівник випускної кваліфікаційної роботи (підпис) / Лещенко О.О. / (ПІБ)

АНОТАЦІЯ

Рожков Ярослав Ігорович виконав випускну кваліфікаційну роботу на тему «Мікросервіс для валідації UI-інтерфейсу» за спеціальністю 122 – «Комп’ютерні науки».

Об'єктом дослідження кваліфікаційної роботи є система для валідації UI-інтерфейсу додатків або зображень.

Предметом дослідження кваліфікаційної роботи є система для валідації UI-інтерфейсу додатків або зображень з використанням штучного інтелекту.

Мета роботи полягає в наданих системі фото екрану з відкритим додатком або окремі зображення для формування рішення по розбіжностям з попередньо завантаженими шаблонами.

Результат роботи. У випускній кваліфікаційній роботі здійснено проектування системи валідації зображень з використанням особливостей мікросервісної архітектури, в результаті виконання роботи створений мікросервіс, що дає змогу обробляти зображення, порівнюючи актуальне зображення зі збереженим шаблоном та отримувати результат валідації через API.

Ключові слова: мікросервіс, валідація, порівняння, API, шаблон, адміністрування.

ANNOTATION

Rozhkov Yaroslav completed the final qualifying work on the topic "Microservice for UI-interface validation" in the specialty 122 - "Computer Science".

The object of study of the qualification work is a system for validation of the UI-interface of applications or images.

The subject of the qualification work is a system for validation of the UI-interface of applications or images using artificial intelligence.

The purpose of the work is to provide a system of photo screens with an open application or individual images to form a solution to the differences with pre-loaded templates.

The result of work. In the final qualifying work, the image validation system was designed using the features of microservice architecture, as a result of the work created a microservice that allows you to process images by comparing the current image with the saved template and get validation results via API.

Keywords: microservice, validation, comparison, API, template, administration.

ЗМІСТ

Перелік умовних позначень та скорочень

QA – Quality Assurence (Тестувальник).

AQA, Automation QA – інженер автоматизованого тестування

UI – User Interface (користувацький інтерфейс).

Валідація – порівняння, перевірка.

API (від англ. Application Programming Interface) - набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Бекенд (англ. Back-end) - усе, що працює на сервері, тобто «не в браузері» або «на комп'ютері, підключеному до мережі (зазвичай, до Інтернету), який відповідає на повідомлення від інших комп'ютерів».

Фронтенд (англ. Front-end) - все, що браузер може читати, виводити на екран та/або запускати. Тобто це HTML, CSS та JavaScript.

БД – база даних

ІС – інформаційна система

англ. – англійська

ВСТУП

На сьогоднішній день розробка програмного забезпечення відбувається у надзвичайно швидкому темпі, але контроль якості створених таким чином продуктів досі залишається вирішеним не до кінця.

Для покрашення швидкості тестування та валідації створеного або зміненого сервісу, додатку, сайту або будь-якого іншого продукту у більшості компаній приходять до нового витку напрямку контролю якості – створення автоматизованих тестів. Роль розробника автоматизованих тестів (далі - AQA) полягає у створенні спеціальними інструментами інструкцій, що дозволять комп’ютеру самому відтворювати дії, які до цього пройшли звичайні тестувальники (далі - QA). QA складають тестові сценарії, по яким AQA створює свої автоматизовані тести.

Але здебільшого, окрім перевірки самого функціоналу, QA до того перевіряють візуальну складову додатку та результатів його роботи, у випадку коли додаток формує документи, зображення, тощо. Тож для повноцінного контролю якості продукту AQA також робити перевірку на правильність відображення про виконанні створених ним інструкцій. Створення відповідного функціоналу для реалізації таких перевірок і є метою даної роботи.

РОЗДІЛ 1. АНАЛІЗ СИСТЕМ ВАЛІДАЦІЇ ТА ПОРІВНЯННЯ ІНТЕРФЕЙСУ ТА ЗОБРАЖЕНЬ

1.1 Аналіз області тестування UI-інтерфейсу

Областю застосування розроблюваного сервісу є сфера розробки автоматизованих тестів, які перевіряють будь-який UI або зображення на відповідність до заздалегідь створених шаблонів. Система має надати можливість передачі у неї необхідних зображень, знімку екрана тощо, що містить у собі візуальну інформацію, яку система буде порівнювати з наданими шаблонами та повертати у відповідь результати порівняння відповідно до заданих параметрів, як наприклад відсоток похибки та інші.

Зручність даної розробки буде полягати у її мікро-сервісній архітектурі, оскільки в такому випадку розробник тестів не буде залежати від обраної мови програмування, фреймворку і інших використаних у проекті технологій, оскільки взаємодія з валідатором буде відбуватися через REST API технологію, а відповідно для коректної роботи необхідно буде просто сформувати тіло відправки у сервіс та викликати метод передачі сервісу та коректно отримати відповідь.

Програмні сторонні бібліотеки, що націлені на написання автоматизованих тестів здебільшого працюють лише з цифровими даними у вигляді тексту та чисел, тож коли розробник створює інструкції для написання тестових сценаріїв, він може зазначати які кнопки у інтерфейсі необхідно натиснути, які дані ввести у необхідні поля, та яке вікно з текстом підтвердження дій необхідно дочекатися.

На таких діях будуються сучасні автоматизовані тести, що допомагають у проведенні регресійних тестів та валідуванні зміненої бізнес-логіки у продукті, що тестується. Тим не менше доволі часто під час тестування потрібно перевіряти не лише правильне функціонування елементу, наприклад кнопки на сайті, а ще й правильність її відображення та розміщення на сторінці сайту. Беручи до уваги, що більшість інструментів для пошуку елементів для їх

перевірки покладається лише на так зване DOM-дерево елементів – успішне проходження відповідного тесту не може гарантувати, що відображення елементів залишилось правильним та незмінним, оскільки незважаючи на наявність елемента у дереві елементів не вказує на його розміщення на сторінці. В такому випадку необхідно шукати інші способи тестування і саме для цього буде створений відповідний сервіс.

1.2 Теорія автоматизованого тестування

Оскільки система буде створена для покращення роботи існуючих автоматизованих тестів, необхідно узагальнити, що саме розуміє під собою автоматизоване тестування.

Автоматизоване тестування (automated Testing) – це набір технік, підходів і інструментальних елементів, які дозволяють вилучити тестувальника з виконання деяких завдань в процесі тестування.[1]

У автоматизованому тестуванні використовується три рівні:

1. Тестування на рівні коду. (Модульне тестування).
2. Функціональне тестування.
3. GUI – тестування. (Graphical user interface – Графічний інтерфейс користувача).[1]

Для професійного тестування використовуються тест-кейси, які частково або повністю покривають інструментальне середовище, однак розробка тест-кейсів, запуск, оцінка виконання тестів та опис дефектів в баг-трекінгових системах не обходиться без втручання тестувальника.

Тест-кейс – набір вхідних даних, умови виконання, очікуваний і фактичний результат, розроблений з цілю перевірки властивостей і поведінку програмного середовища.

Переваги такого виду тестування:

- Тестування навантаженості системи.
- Час виконання автотестів.
- Легкість повторної перевірки після внесення змін.

• Повторюваність.

Недоліки такого тестування:

• Великі витрати.

• Вартість інструменту автоматизації.

• Пропуск дрібних помилок.

Незважаючи на перелічені вище недоліки автоматизованого тестування, на сьогодні бізнес доволі часто витрачає кошти на це, оскільки комп'ютер зчитує, фіксує і порівнює інформацію швидше, аніж людина, до того ж він може це робити в будь-який час та скільки завгодно разів, що підвищує загальний рівень усвідомлення команди розробки продукту про його поточний стан. Автоматизація тестів – найкращий спосіб підвищити ефективність, охоплення тестом і швидкість виконання при тестуванні програмного забезпечення. [1]

Автоматизоване тестування програмного забезпечення важливо з таких причин:

• Ручне тестування всіх робочих процесів, усіх полів, усіх негативних сценаріїв вимагає часу та грошей.

• Складно перевірити багатомовні сайти вручну.

• Автоматизація тестів при тестуванні програмного забезпечення не вимагає втручання людини. Ви можете запустити тест і залишити його без нагляду.

• Автоматизація тестів збільшує швидкість виконання тесту.

• Автоматизація сприяє збільшенню охоплення тестом.

• Тестування вручну може стати нудним, схильним до помилок. Автоматизований процес тестування:

У процесі автоматизації виконуються наступні кроки (рис. 1.1)[2]:

1) Вибір інструменту тестування.

2) Визначення сфери автоматизації.

3) Планування, проєктування та розробка.

4) Виконання тесту.

5) Технічне обслуговування.

Рис. 1.1. Процеси автоматизованого тестування.

1.3 Аналіз сучасної мікросервісної архітектури

На сьогодні майже всі компанії та розробники, що працюють у IT-сфері зрозуміли, що рух до монолітної архітектури розроблюваного ПО призводить лише до проблем, серед яких – велика вартість реалізації, складність у підтримці, важка до масштабування система та таке інше. Архітектура мікросервісів прийшла на допомогу в момент, коли необхідно було знайти рішення проблем, описаних вище, при цьому не втратити нічого у грошах, продуктивності та гнучкості.

Сучасний підхід до створення додатків показує нам, що можливість робити свої розробки не атомарними, а модульними – доволі раціональне рішення. Це допомагає зменшити навантаження на окремі вузли системи, полегшити заміну бажаного модуля на новий, або просто додати у систему більше операційних вузлів. Реалізація такої архітектури через ці переваги стала домінантною на сучасній арені IT. Саме тому далі буде розглядатися реалізація саме з таким підходом до побудови системи.

Традиційна розробка програми працює за принципом моноліту: всі завдання реалізуються в одному великому додатку. Усі окремі сервіси мають доступ до великої бази даних і виводяться через інтерфейс користувача - все реалізовано в рамках однієї програми. Підхід мікросервісів заснований на модулях: кожен мікросервіс відповідає лише за виконання окремого завдання. Робочі процеси настільки ж різні, як і результати двох підходів.

Беручи до уваги область даної роботи – створення інструменту для розробників автоматизованих тестів, питання вибору архітектури вирішується одразу. Адже якщо поглянути на ідею з більш детальним вивченням проблеми, то стане зрозуміло, що автотести завжди є лише доповненням до основного продукту, а не центральною складовою, а отже немає сенсу розробляти певний функціонал або бібліотеку для прямого використання у розроблюваних тестах.

Набагато простіше реалізувати мікросервіс, до якого автотести, під час свого виконання тестових інструкцій, будуть звертатися завдяки Application Programming Interface (далі - API), що допомагає у зв’язках між сервісами у сучасній архітектурі. Адже під час того, як кожний модуль працює у своєму середовищі, виконуючи свої функції за допомогою заздалегідь розробленої логіки – в кінцевій точці будь що оперує з даними, формати яких зазвичай намагаються привести до певної стандартизації, тому налаштувавши передачу таких уніфікованих даних від одного модуля до іншого за допомогою інтерфейсу. Така реалізація дозволить уникнути певних проблем з місцем зберігання необхідного функціоналу для валідації у тому ж сховищі, що й автоматизовані тести, а також не навантажувати зайвий раз процесор, зайнятий виконанням тестових інструкцій ще й обробкою методів порівняння зображення для надання структурованої відповіді стосовно схожості або певних розбіжностей між шаблонним зображенням та фактичним, тим що згенерували тести під час свого виконання.

Звичайно таких підхід не може бути на сто відсотків складеним з переваг. В даному випадку основним недоліком можна вважати питання розгортання створеного сервісу, адже запускаючи тести з уже влаштованим функціоналом по обробці зображень, фахівцю непотрібно перейматися за те, щоб функціонал був доступний, оскільки вбудований в систему варіант напряму буде запускатися з тестами у тому ж середовищі, без додаткових потреб у налаштуванні. Але з іншого боку, вирішення питання розгортання та подальшого використання для зв’язку з сервісом інтерфесу API надає сервісу ще однієї великої переваги – можливість використовувати реалізацію з будь

яким написаним функціоналом для створення автотестів, оскільки можливість виконувати інструкції для перевірки продукту, для якого написані тести, призводить до висновку що в середовищі існує функціонал для виклику методів API, отримання та відповіді на подібні запити, що допоможе створити уніфіковане рішення для більшості сучасний фреймворків та використовувати розроблений функціонал у відповідних середовищах.

Повертаючись до питання розгортання можна зрозуміти, що разом з сучасними тенденціями на мікросервіси все більше компаній та фахівців намагалися знайти відповідь на поставлене питання – якщо мікросервіси полегшують створення складної цифрової архітектури, то як усунути питання розгортання, адже це не дозволяє впроваджувати всюди подібний підхід. В момент пошуку відповіді на це питання згадали про вже створену, але не так часто уживану технологію віртуалізації, та її подальшу гілку розвитку – контейнеризацію. Стало зрозуміло, що кількість сервісів може зростати дуже швидко, адже було дуже легко захопитися у поділі наявних процесів на окремі частини у вигляді сервісів, але тоді необхідно було розгортати під кожний сервіс своє середовище для виконання та роботи. Тому на шляху пошуку оптимального рішення було створено програмне забезпечення Docker, що використовує технологію контейнеризації чи не найкраще за усі свої аналоги. Оскільки під час проходження навчання на курсі, один з предметів вивчення був присвячений саме йому, то досвід роботи з цією технологією наявний, а отже має рацію використовувати саме її під час фінального етапу розробки функціоналу для перевірки його роботи та відповідності тим вимогам, що були визначені заздалегідь.

1.4 Мета створення мікросервісу для валідації UI-інтерфейсу

Для визначення мети необхідно мати представлення про те, чому саме в даній ситуації існує потреба у подібній реалізації.

Як було згадано вище – на сьогоднішній день існує дуже багато мов програмування, завдяки яким створюються автотести, а також багато фреймворків для покращення рівня якості написаних інструкцій для перевірки ПО. У зв’язку з цим існує багато бібліотек для різних мов, що допомагають у відповідному бажанні отримувати інформацію про зміну візуального інтерфейсу. Окрім цього, порівнювати наявний UI необхідно з якимось шаблоном, що надасть змогу порівняти та визначити чи намає певних змін з точки зору візуального сприйняття. Усі ці шаблони необхідно десь зберігати, що буде займати додаткове місце, адже з огляду на специфіку написання тестів, більшість розробників тримають необхідні ресурси у тій самій папці, що й сам проект автоматизації.

Отже на загал можна побачити ситуацію, коли є потреба у уніфікованому рішенні, яке до того ж вирішить питання розміщення еталонних шаблонів для порівняння. Перше, що спадає на думку для вирішення цього питання – мікросервіс. Цей принцип побудови додатку вирішує проблему уніфікації, адже API дозволяє поєднувати розрізненні блоки до єдиної системи, аби лише формати даних для передачі були узгодженні та з обох сторін (проекту автоматизованих тестів – з одного, та мікросервісу валідації – з іншого) була налагоджена обробка відправки запитів та прийняття відповіді та логіка роботи з ними.

На додаток до цього рішення про створення даної реаліхації включає у себе бажання зробити зручним не лише сам процес валідації зображення, але й підготовки до цього процесу. Налаштування та описання коректної бази даних допоможе вирішити питання зі зберіганням шаблонів, оскільки дані будуть доступні під час виклику сервісу, що надасть змогу використовувати одне сховище декільком розробникам тестів, а також не займати додаткове місце у локальному проекті, адже об’єм даних може сягати доволі великих значень пам’яті, що призведе до зайвого збільшення проекту автоматизованих тестів. Це не було б проблемою, якби подібний проект був основним у продукті, інколи таке допускається. Але репозиторій автотестів це додаткова розробка до

основного продукту (а відповідно і репозиторію). У випадку коли над автотестами працює декілька розробників подібний підхід допоможе зберегти час на комунікацію між ними, узгодження стеку ресурсів, що будуть використовуватися для тестів, адже маючи одну й ту саму базу даних немає проблеми у відповідності тестових даних, з бази вони однакові.

1.5 Аналіз реалізації системи

Для реалізації даного мікросервісу, який буде валідувати зображення, необхідна реалізація штучного інтелекту, що буде порівнювати зображення з шаблонним, знаходити відмінності та помічати їх для результуючої відповіді.

Необхідні кроки для реалізації мети:

1. Реалізувати модуль порівняння зображень та влаштувати його у мікросервіс;
2. Реалізувати функціонал формування результуючої відповіді;
3. Описати в системі залежність функціоналу від заданих адміністратором параметрів;
4. Створити архітектуру мікросервісу з механізмами отримання даних для валідації та надсилання у відповідь результатів перевірки;
5. Створити базу даних для зберігання шаблонів, з якими будуть порівнюватися надіслані зображення;
6. Розробити веб-додаток для адміністрування мікросервісу;

Далі зображено схематичну діаграму взаємодії користувачів з системою мікросервісу (Use case), де зображені відповідні користувачі системи та її основні функції (рис. 1.2)

Рис. 1.2 Діаграма Use case взаємодії з мікросервісом

На діаграмі можна побачити двох основних користувачів системи – самого розробника автотестів, що використовує систему для роботи, та адміністратора сервісу, що несе відповідальність за надання доступу до сервісу та його підтримання.

Окрім цього на діаграмі зображено дві додаткові функції системи, це завантаження шаблонів до системи, та налаштування користувацьких параметрів, оскільки допускається мінімальна робота з сервісом і без цих двох функцій.

1.6 Постановка задачі

Об'єктом дослідження кваліфікаційної роботи є система для валідації UI-інтерфейсу додатків або зображень.

Предметом дослідження кваліфікаційної роботи є система для валідації UI-інтерфейсу додатків або зображень з використанням штучного інтелекту.

Мета роботи полягає в наданих системі фото екрану з відкритим додатком або окремі зображення для формування рішення по розбіжностям з попередньо завантаженими шаблонами.

Внутрішні зацікавлені сторони:

- Розробник автоматизованих тестів;
- Технічні компанії;
- Тестувальники проектів;
- Власники продукту;

Зовнішні зацікавлені сторони:

- Конкурентні продукти.

Задачі дипломного проектування:

- Дослідити методи валідації та порівняння зображення
- Розробити на основі сервісної архітектури додаток для зручного і швидкого вбудовування в існуючі проекти розроблених автоматизованих тестів

Функціональні вимоги:

- Програмний додаток повинен взаємодіяти з основною системою шляхом REST API
- Система повинна бути незалежна від використовуваної мови програмування, що використовується для написання інструкцій автоматизованих тестів та мати уніфіковані методи звернення до системи
- Система повинна мати можливість для налаштування адміністраторами системи
- Розроблюваний додаток має формувати та надсилати у відповідь результати роботи порівняння та валідування зображення
- Додаток має мати можливість завантажувати заздалегідь створені шаблони для порівняння та мати базу даних для їх зберігання

Нефункціональні вимоги:

- Незалежність (система має розміщуватися на віддаленому сервері та не залежати від проектів розробників автоматизованих тестів)
- Продуктивність (програма має швидко оброблювати дані, щоб не затримувати проходження тестів по продукту, що тестується)

На вході у системи буде мати вхідні дані:

- Поточне зображення для порівняння.
- Заздалегідь створені шаблони для порівняння з поточним зображенням
- Параметри валідування, що при необхідності розробники можуть вносити до системи для корегування її роботи

Вихідні дані:

- звіт результату порівняння у вигляді зображення з відміченими зонами розбіжності поточного зображення з шаблоном;
- Рішення про співпадіння, або коефіцієнт розбіжності.

Представимо програмний модуль у вигляді «чорної скрині». (див. рис. 1.3)

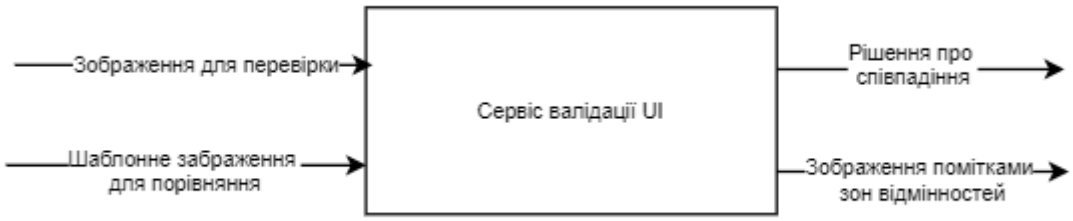


Рис. 1.3 – Програмний модуль у вигляді "чорної скрині"

Висновок до першого розділу

В результаті виконання першого розділу було проведено аналітичний огляд дипломної роботи. Була визначена мета даної роботи, її призначення та завдання, особливості роботи та область застосування.

При описанні першого розділу були виокремлені відповідні функціональні та нефункціональні вимоги до системи, що являють собою закладений в систему функціонал та визначають подальші критерії прийняття системи як завершеної по закінченню розробки. Також схематично були зображені такі діаграми, як Use case, де були зазначені основні користувачі системи, а також їх функції, окрім цього була зображена так звана схема «чорного ящика», що наявно демонструє лише зовнішні фактори, що впливають на систему.

Окрім цього були проаналізовані існуючі конкурентні рішення на ринку, виділені їх сильні та слабкі сторони для розуміння того, якою саме повинна бути розроблювана система, щоб мати можливість виділятися на фоні інших рішень.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ МІКРОСЕРВІСУ ДЛЯ ВАЛІДАЦІЇЇ UI-ІНТЕРФЕЙСУ

2.1 Функціональний аналіз

Робота з шаблонами :

- Завантаження нових шаблонів (користувач може через користувацький UI завантажити необхідні для себе шаблони та обрати ім'я для шаблону, з яким він буде збереженний до бази даних)
- Отримання збережених шаблонів (функціонал, що дозволяє отримати дані шаблонів з база даних для їх подальшого відображення)
- Перегляд нових шаблонів (після завантаження шаблонів через UI, там же користувач має змогу переглянути заздалегідь завантажені шаблони, враховуючи і ті, що він щойно завантажив)
- Видаленння (користувач може обрати ті шаблони, які йому вже не потрібні та видалити їх)

Валідація зображення :

- Отримання поточного зображення (отримання поточного зображення для перевірки від користувача)
- Отримання шаблону зі сховища (отримання заздалегідь збереженого шаблону з бази даних)
- Порівняння зображень (процес візуального порівняння поточного зображення та шаблону з бази даних)

Генерація результату :

- Генерація результуючого зображення (генерація та форматування результату валідації у вигляді зображення)
- Повернення результату валідації (функціонал віддачі користувачу результуючого зображення та рішення про співпадіння)

Поділ на функціональні підкласи зображено на рисунку (рис 2.1)

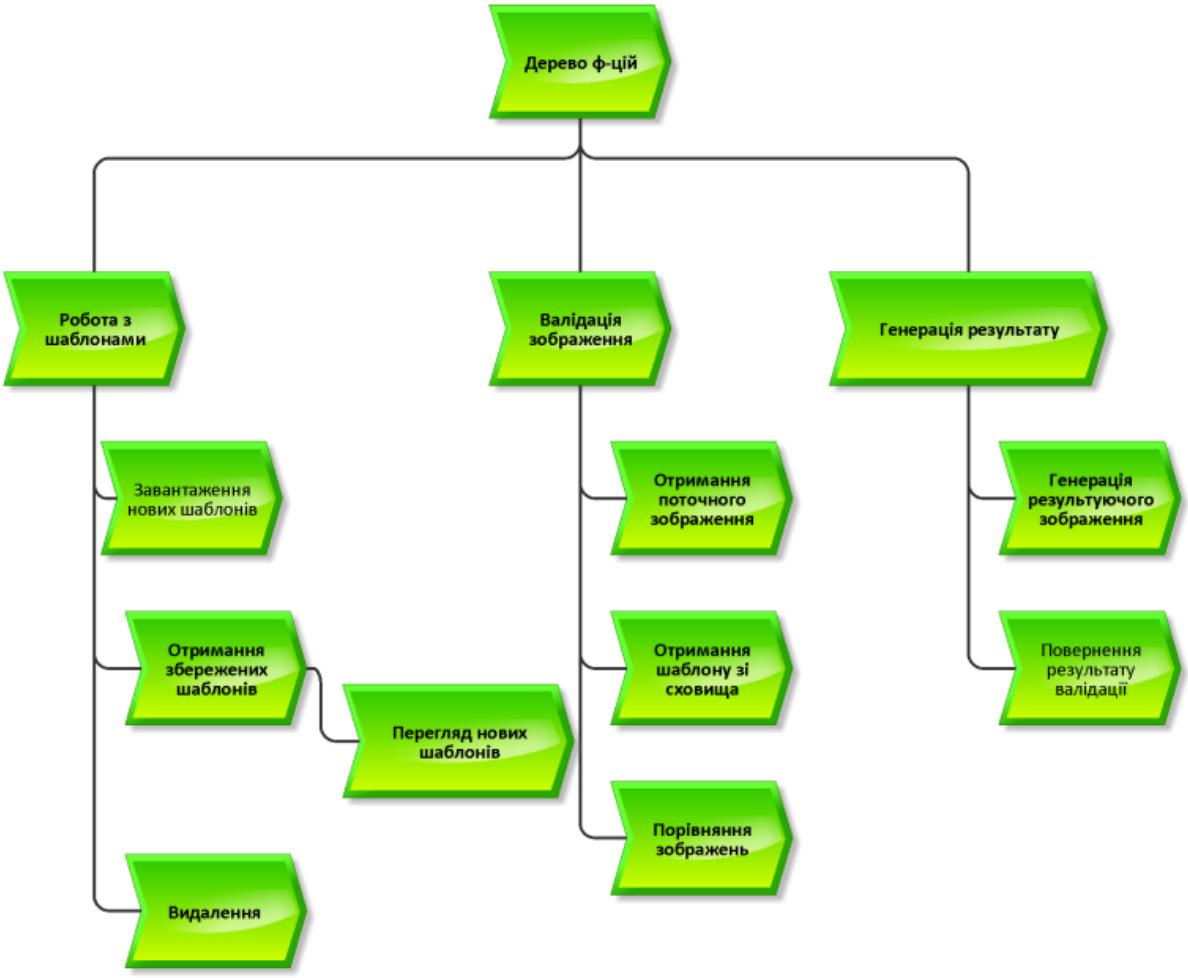


Рис. 2.1. Дерево функцій

2.2. Архітектура системи у нотації IDEF0

1. У якості Input (інформація, яка надходить до системи в момент її ініціалізації та потребується для подальшої роботи системи) на схемі виступає: зображення для перевірки, параметри валідування, шаблонне зображення для порівняння;
2. У якості Control (документи чи правила, які регулюють процес обробки інформації) виступають правила контролю доступу, виконання яких необхідно брати до уваги під час роботи сервісу для відмежування сторонніх осіб від

системи, а також правила цілісності даних, що повинні виконуватися, аби в системі не виникало артефактів та вона не зламалася.

3. Виконавцем (Mechanism) виступають розробник автоматизованих тестів, що звертається до сервісу під час необхідності порівняти певне зображення, а також адміністратор сервісу, що надає доступ користувачам.

4. У якості вихідної інформації Output (результат роботи програми) виступає зображення з помітками зон відмінностей (якщо такі є), а також рішення системи чи співпадають

Намальована за правилами діаграма у нотації IDEF0 (рис. 2.2.)

Рис. 2.2. IDEF0 процесу валідації UI-інтерефейсу

2.3 Архітектура інформаційної системи

ІС валідації UI-інтерфейсу складається з трьох підсистем :

1. Підсистема керування даними :

1.1 Модуль збереження шаблонів в базу даних (модуль, що здійснює конвертацію до необхідного формату вхідного файлу та зберігає його до бази даних)

1.2 Модуль повернення шаблонів з бази даних (модуль, що отримує необхідні дані з бази даних, конвертує до необхідного формату та в віддає його на запит отримання даних)

2. Підсистема валідації :

2.1 Модуль проведення порівняння (модуль, що здійснює візуальне порівняння між зображенням, що надав користувач, та шаблоном з бази даних);

2.2 Модуль генерації результату (модуль, що реалізує генерацію фактичного результуючого зображення, беручи за основу поточний файл користувача);

3. Підсистема адміністрування

3.1 Модуль адміністрування списком користувачів (модуль, що реалізує видавання доступу адміністратором користувачу)

3.2 Модуль адміністрування списком шаблонів (модуль, що буде наданий користувачу для керування своїм списком доступних шаблонів)

Модулі та підсистеми зображені на рисунку архітектури (рис. 2.3)

Рис. 2.3 Архітектура інформаційної системи

2.3.1 Діаграма сутностей

Опишемо діаграму сутностей (рис 2.4), за якими буде існувати предметна область. Виділимо такі об’єкти:

- користувач;
- система;
- валідатор зображення;

Рис. 2.4 Діаграма сутностей

2.3.2 Діаграма діяльності

Після того, як було визначено основні сутності опишемо їх основні життєві цикли. Опишемо основні стани життєвих циклів та їх переходи. Побудуємо діаграму діяльності системи валідації UI-інтерфейсу (рис. 2.5).

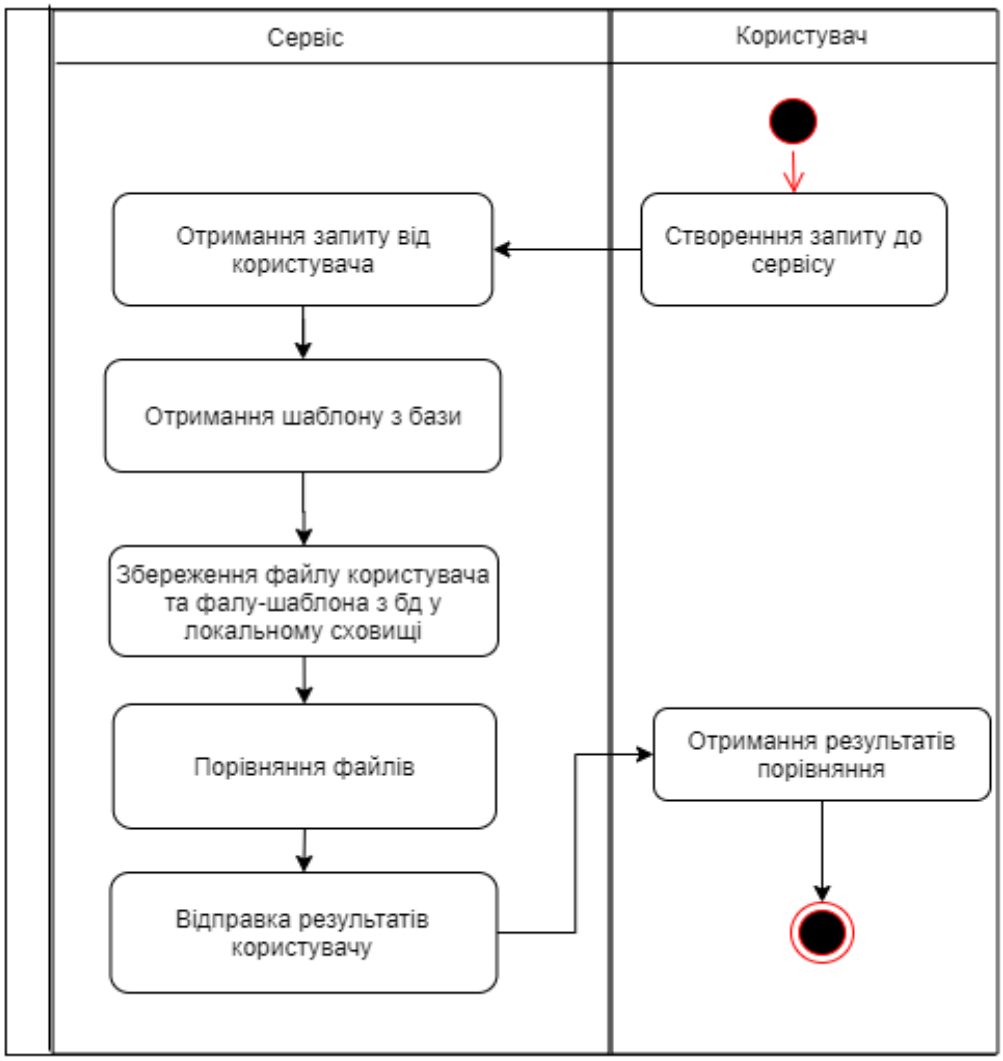


Рис. 2.5. Діаграма діяльності

Для початку роботи користувачу необхідно звернутися до сервісу, передавши йому файл для валідації та ім’я збереженного у бд шаблону для порівняння. Після ініціалізації процесу сервіс робить запит до бази даних для отримання файлу-шаблону з таким ім’ям. Після отримання відповідного файлу з бази даних сервіс запускає процес порівняння файлів, результат якого відправляє назад користувачу.

2.3.3 Діаграма розгортання

Для представлення загальної архітектури та зв’язків розробленої системи була створена діаграма розгортання (рис. 2.6) , на якій зображенні окремі компоненти системи та як вони пов’язані між собою.

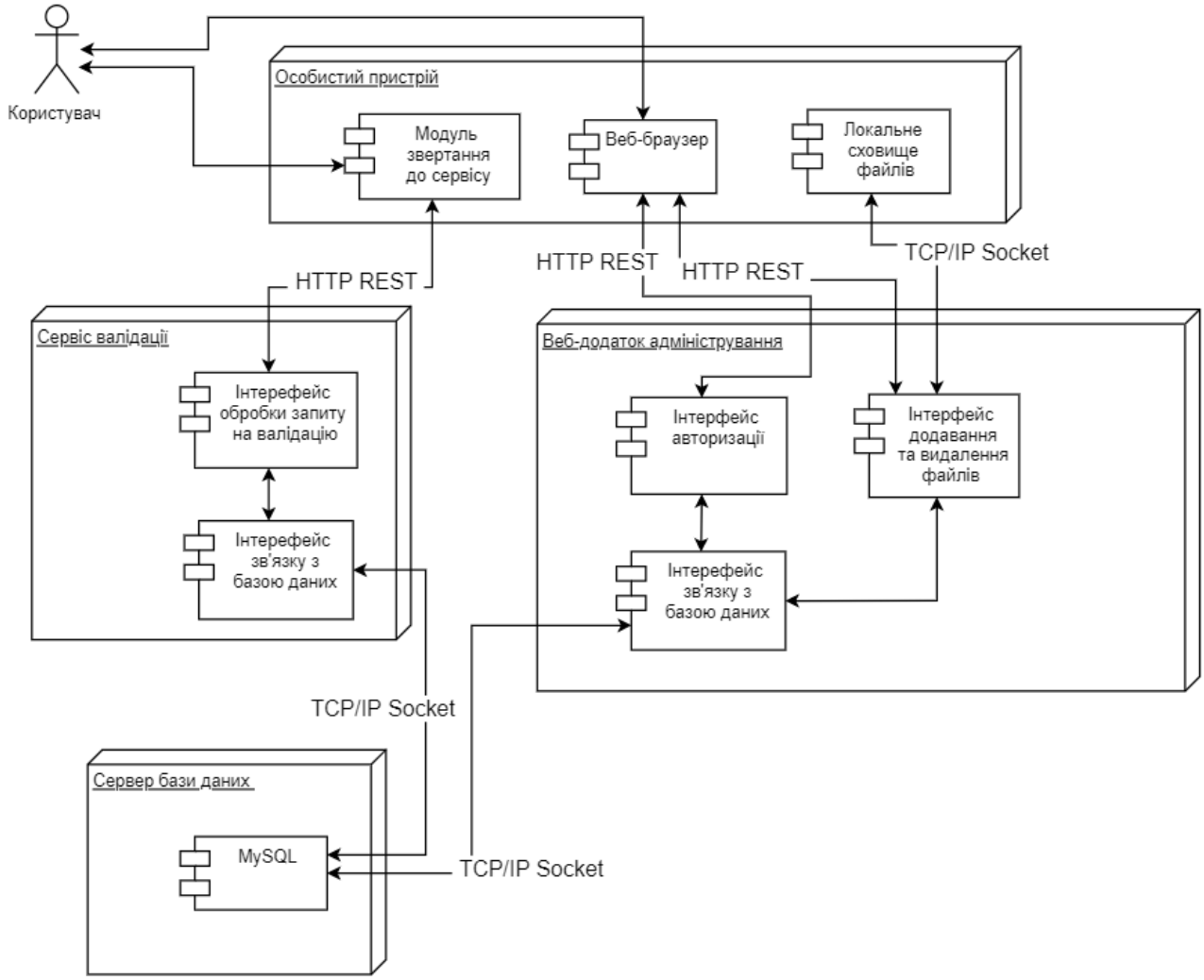


Рис. 2.6. Діаграма розгортання

На діаграмі наглядно видно, що на загал система складається з двох частин – адміністрування та самого сервісу валідації.

Частина з адмініструванням має свій фронтенд та бекенд, на яких реалізовані інтерфейси для взаємодії з користувачем та для зберігання даних.

Частина з сервісом передусім має інтерфейс для отримання запитів від користувача, а також для зв’язку з базою даних щоб отримати дані про шаблони для подальшої валідації.

2.4 Вибір середовища та інструментів реалізації

- Java – жорстко-типізована об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Sun Microsystems (надалі придбаною компанією Oracle). Розробка ведеться спільнотою, організованою

через Java Community Process; мова та основні технології, що реалізують її, поширюються за ліцензією GPL. Права на торгову марку належать корпорації Oracle.[3]
Програми Java зазвичай транслюються у спеціальний байт-код, тому вони можуть працювати на будь-якій комп'ютерній архітектурі, для якої існує реалізація віртуальної Java-машини. Дата офіційного випуску – 23 травня 1995 року. Займає високі місця у рейтингах популярності мов програмування. Історія Java почалася в 1991 році, коли група розробників компанії Sun Microsystems (яка згодом приєднається до Oracle) під керівництвом Джеймса Гослінга розпочала створення мови Oak («Дуб») для використання у цифрових побутових пристроях. Oak мав стати альтернативою C/C++ як простий і кросплатформенний мову.
Найбільше Java затребувана у сфері фінансів, де потрібні надійні веб-сайти, стійкі до високого трафіку, програми електронного документообігу та власні платформи для управління ризиками. На Java розробляють серверні додатки для обробки даних, його використовують при створенні візуального вигляду веб-сайтів за допомогою спеціальних шаблонів (фреймворків), а також на ньому пишуть ігри.
Не обходиться без Java і під час написання хмарних програм. Цю мову можна зустріти в серверній частині інтернет речей (IoT), де використовуються різні датчики та камери відеоспостереження.[4]

- Spring Framework - один з найпопулярніших фреймворків для створення веб-застосунків на Java. Spring забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники та організації, які хочуть створити інформаційну систему, що базується на платформі Java . Через широку функціональність важко визначити найбільш значущі структурні елементи, у тому числі він складається. Spring не повністю пов'язаний з платформою Java Enterprise , незважаючи на свою масштабну інтеграцію з нею, що є важливою причиною його популярності.
Spring , ймовірно, найбільш відомий як джерело розширень (features), необхідних для ефективної розробки складних бізнес-додатків поза великоваговими програмними моделями, які історично були домінуючими в промисловості. Ще одна його перевага в тому, що він ввів функціональні можливості, що раніше не використовуються, в сьогоднішні панівні методи розробки, навіть поза платформою Java.
Цей фреймворк пропонує послідовну модель і робить її застосовною до більшості типів програм, які вже створені на основі платформи Java. Вважається, що Spring реалізує модель розробки, засновану на найкращих стандартах індустрії, і робить її доступною у багатьох сферах Java.[5]

- ReactJS – це JavaScript - бібліотека з відкритим кодом для розробки інтерфейсів користувача. React розробляється та підтримується Facebook, Instagram та спільнотою окремих розробників та корпорацій.React може використовуватися для розробки односторінкових та мобільних додатків. Його мета — надати високу швидкість, простоту та масштабованість . Як бібліотека для розробки інтерфейсів користувача React часто використовується з іншими бібліотеками, такими як MobX, Redux і GraphQL.
У React присутня однонаправлена передача даних - властивості передаються від батьківських компонентів до дочірніх. Компоненти отримують властивості як безліч незмінних (англ. immutable) значень, тому компонент не може безпосередньо змінювати властивості, але може викликати зміни через callback-функції. Такий механізм називають «властивості вниз, події нагору».
React використовує віртуальний DOM (англ. virtual DOM). React створює кеш -структуру в пам'яті, що дозволяє обчислювати різницю між попереднім та поточним станом інтерфейсу для оптимального оновлення DOM браузера. Таким чином, програміст може працювати зі сторінкою, вважаючи, що вона оновлюється вся, але бібліотека самостійно вирішує, які компоненти сторінки необхідно оновити.
React Hooks дозволяють використовувати стан та інші можливості React без написання класів. Побудова хуків користувача дозволяє поміщати логіку компонента в повторно використовувані функції. [6]

Висновок другий розділ

Отже, в результаті виконання другого розділу дипломної робити було проведено функціональний аналіз та побудовано дерево функцій системи. Наведено узагальнену технічну архітектуру системи, визначено підсистеми, модулі та зв’язки між компонентами системи.

Розроблено структуру та головну функціональність мікросервісу за допомогою Java Spring Boot та адміністраторскої частини за допомогою React та Spring Boot із використання мови програмування Java, ReactJS.

На далі можна створити користувацький інтерфейс для адміністраторської частини ІС, написати документацію для користувача та провести тестування розробленого мікросервісу та проаналізувати роботу самого застосунку.

РОЗДІЛ 3. ТЕХНОЛОГІЧНІ ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО МОДУЛЯ

3.1. Реалізація програмного додатку

Під час розробки системи був розроблений функціонал, що дозволяє після розгортання мікросервісу у системі та ініціалізації початкових даних можна починати використання.

Мікросервіс складається з двох частин, це сам сервіс та частина адміністрування. Вони пов’язані між собою за допомогою API-інтерфейсу.

3.1.1 Реалізація підсистеми адміністрування

Адміністраторська частина, як було наведено у попередніх розділах, допомагає у додаванні нових шаблонів користувачу, для подальшого їх використання у процесі валідації, а також для адміністрування користувачів

Для реалізації системи адміністрування було використано дві складові. Основна частина, що відповідальна за логіку інтерфейсу адміністрування реалізована за допомогою Spring Boot , інформація про цей фреймворк була наведена у другому розділі, та бази даних, що являє собою реляційна база даних MySQL.

При ініціалізації системи автоматично створюється обліковий запис з адміністраторськими правами. Уся інформація потрапляє у таблицю у базі даних та зберігається там для подальшого використання.

При заході по посиланню на сайт користувачу виводиться вікно авторизації.(рис.3.1)

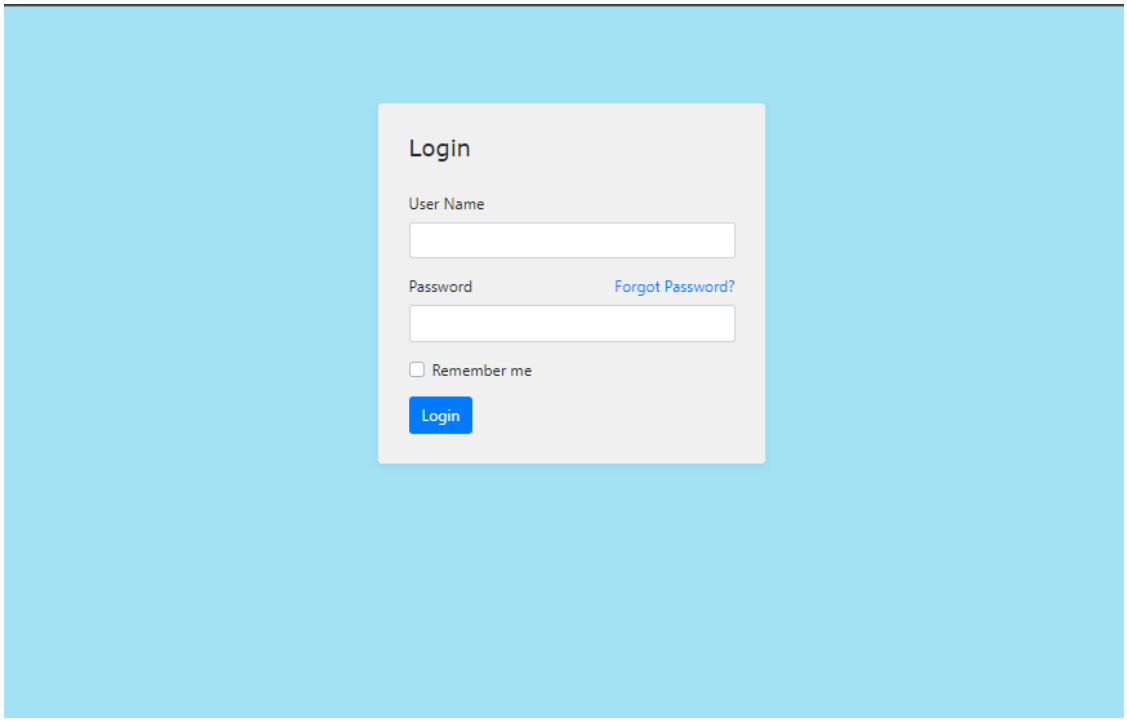


Рис. 3.1 Вікно авторизації на сайті адміністрування

На бекенді дані користувача перевіряються з даними з бази, а пароль хешується за допомогою pringframework.security.crypto.password. Ця бібліотека допомагає сховати пароль у сховищі, що б ті учасники проекту, які мали доступ до отримання даних з бази – не могли побачити реальний пароль користувача.

Під час авторизації у системі на бекенді формується токен доступу, який на далі, при виклику методів з серверної сторони, використовується, як пароль, для того щоб уникнути виклику методів поза сесією користувача (лістинг методу, що відповідає за генерацію сесії наведено у лістинг 3.1, а опис методу у таблиці 3.1).

Лістінг 3.1.

```
@PostMapping("/auth/login")
    public ResponseEntity<?> login(@RequestBody
AuthenticationRequest authenticationRequest) throws
InvalidKeySpecException, NoSuchAlgorithmException {

          final Authentication authentication =
authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                    authenticationRequest.getUserName(),
authenticationRequest.getPassword()));
```

```
    SecurityContextHolder.getContext().setAuthentication(authenti
cation);

          User user=(User)authentication.getPrincipal();
          String
jwtToken=jWTTokenHelper.generateToken(user.getUsername());

          LoginResponse response=new LoginResponse();
          response.setToken(jwtToken);


          return ResponseEntity.ok(response);
     }
```

Таблиця 3.1. Опис методу авторизації

Method name	api/v1/auth/login
Method type	POST
Request body	{username:'',password:''}
Response body	{token:''}

Для демонстрації роботи цього методу та інших буде використовуватися програмне забезпечення Postman (рис 3.1).

Після того, як були внесені всі необхідні дані, можна натиснути кнопку “Send” та впевнитися у роботі методу на серверній частині.

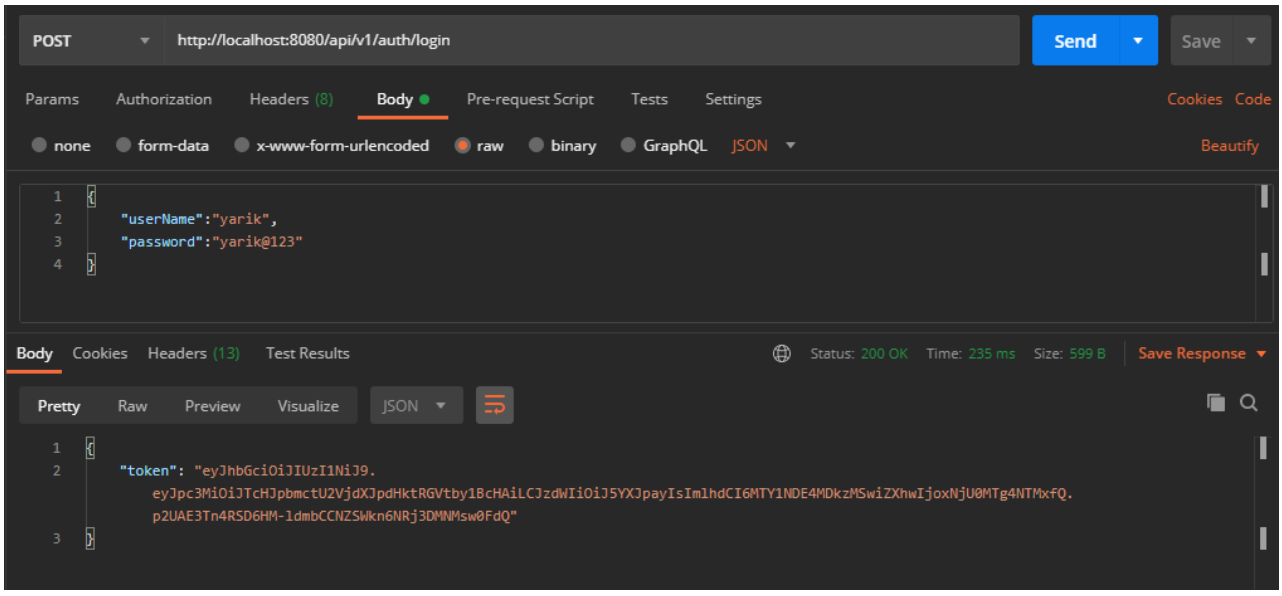


Рис. 3.1. Робота методу авторизації у програмі Postman

У випадку, коли авторизація пройшла успішно і дані співпадають з даними у базі даних на фронтенд віддаються дані користувача відповідним методом (Лістинг 3.2)

Лістінг 3.2.

```
@GetMapping("/auth/userinfo")
    public ResponseEntity<?> getUserInfo(Principal user){
        User userObj=(User)
userDetailsService.loadUserByUsername(user.getName());

        UserInfo userInfo=new UserInfo();
        userInfo.setFirstName(userObj.getFirstName());
        userInfo.setLastName(userObj.getLastName());
        userInfo.setRoles(userObj.getAuthorities().toArray());


        return ResponseEntity.ok(userInfo);

    }
```

Після потрапляння в інтерфейс додавання шаблонів користувачу виводяться дані користувача, усі його шаблони, якщо вони уже були додані (рис. 3.2)

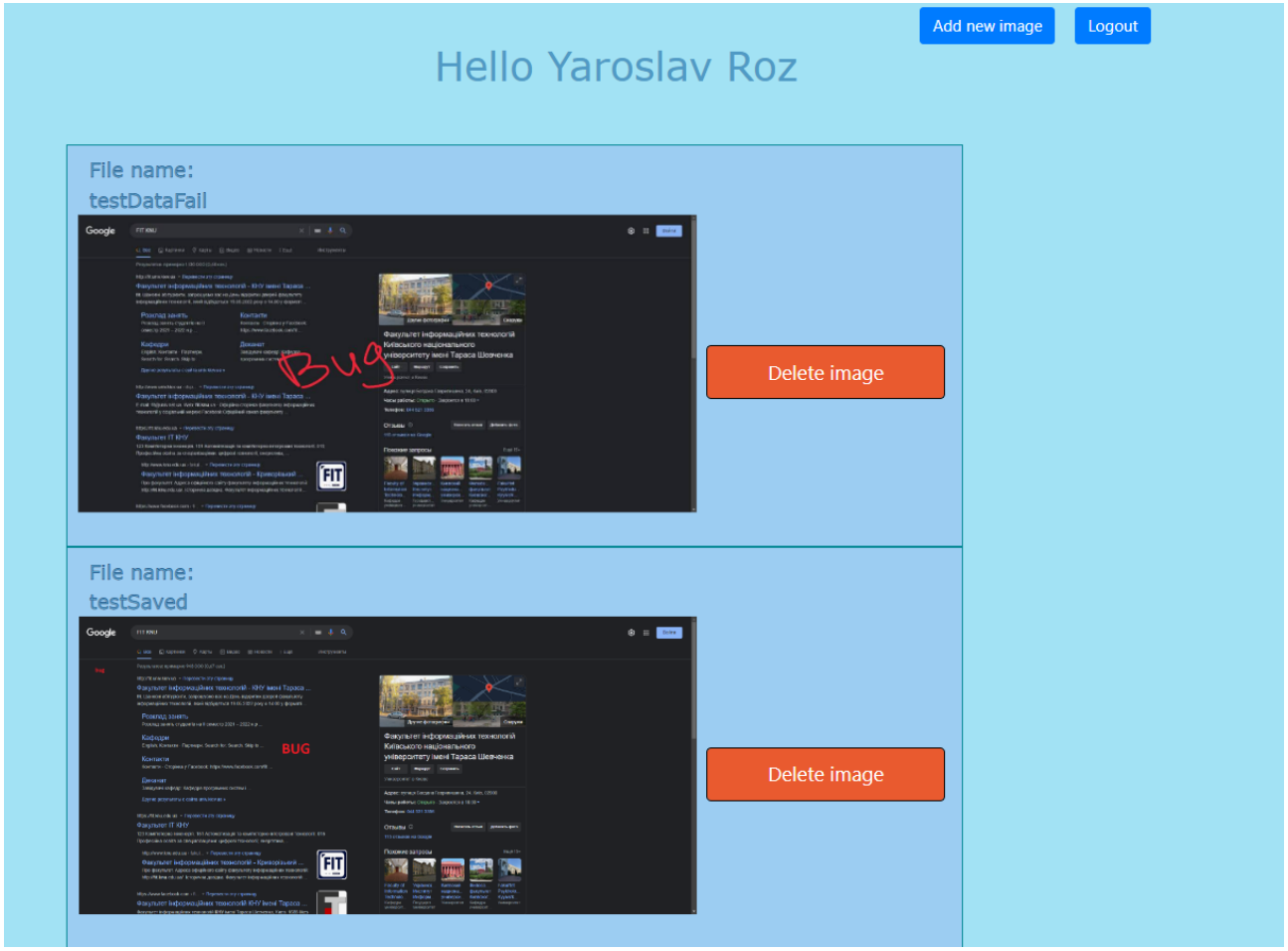


Рис.3.2 Інтерфейс користувацької сторінки у адміністраторському модулі

Дані для сторінки, такі як ім’я файлу та сам файл у вигляді мета-даних береться з відповідної таблиці *file_entity* у базі даних(рис. 3.3)

Рис.3.3 Фізичний вигляд таблиці file_entity у адміністраторі PHPMyAdmin

Дані також отримуються на серверній частині методом getAllFilesFullData (Лістинг 3.3), метод викликається на фронтенді, у проекті реалізованому за допомогою ReactJS (реалізація виклику методу наведена у лістингу 3.4)

Лістінг 3.3. Віддавання даних по файлах на бекенді

```
@GetMapping("/getAllFilesFullData")
    public List<imageBase> getAllFilesWithFullData(){
        List<String> list = fileEntityService.getFileMetadata();
        List<imageBase> listBase = new ArrayList<>();
        for(String s:list)
            listBase.add(new imageBase(s));
        return listBase;
    }
```

Лістінг 3.4. Отримання даних по файлах на фронтенді

```
export const getMetaDataAOXIS=()=>{
    return axios({
        method:'GET',

url:`${process.env.hostUrl||'http://localhost:8080'}/file/getAllFi
lesMetada`,
        headers:{
            'Authorization':'Bearer '+getTokenOut(),
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
    });

}
```

За домогою реалізації на ReactJS відповідь від серверу розбивається на об’єкти, після чого генерується візуалізація на користувацькій сторінці, наведена вище. Оскільки на запит методу повертається JSON об’єкт з масивом даних по кожному файлу, то на фронтенді за допомогою функції forEach, що виконує описану логіку для кожного об’єкта списку, формується список з web об’єктів, що включають у себе дані по файлу, конвертовані в тип Image, його назва та згенерована на кожний об’єкт кнопка видалення, аби користувач мав змогу видалити непотрібні шаблони.

Також в даному додатку для користувача реалізований функціонал додавання нових шаблонів для подальшої роботи з ними при проведенні валідації. При натисканні кнопки «Add new image» користувач потрапляє на сторінку для оновлення свого списку шаблонів, де може натиснути кнопку «Upload file», вибрати на своєму пристрої необхідний файл (рис 3.4), а також

набрати бажану назву для нового файлу, який буде збережений до бази та натиснути «Add file» (рис. 3.5).

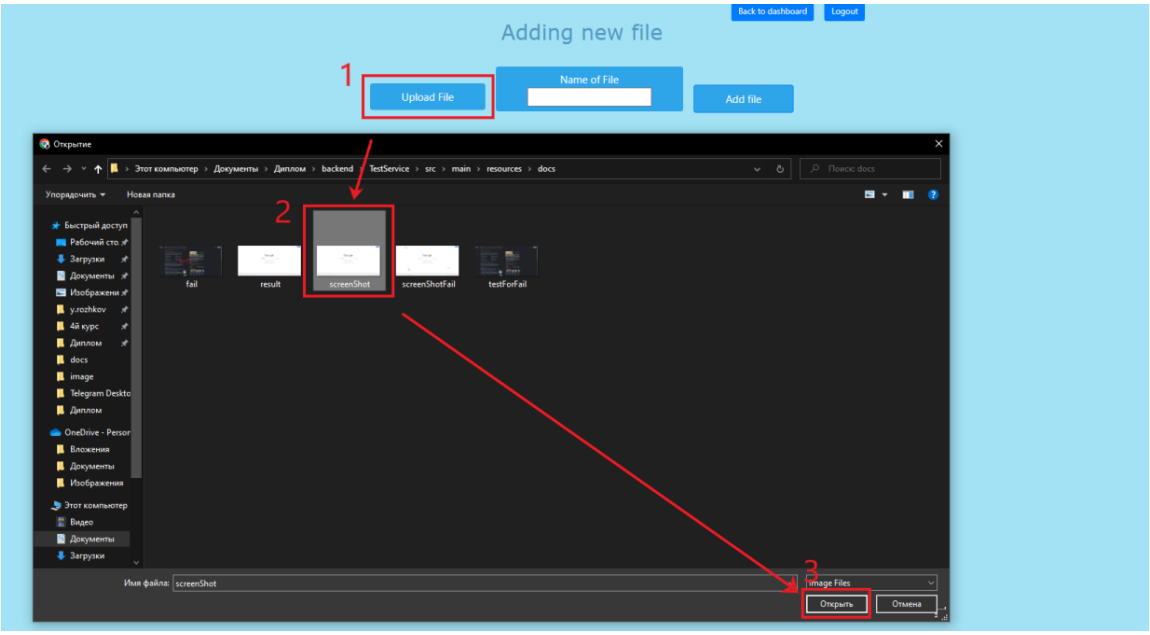


Рис.3.4 Схема додавання нового зображення у інтерфейсі

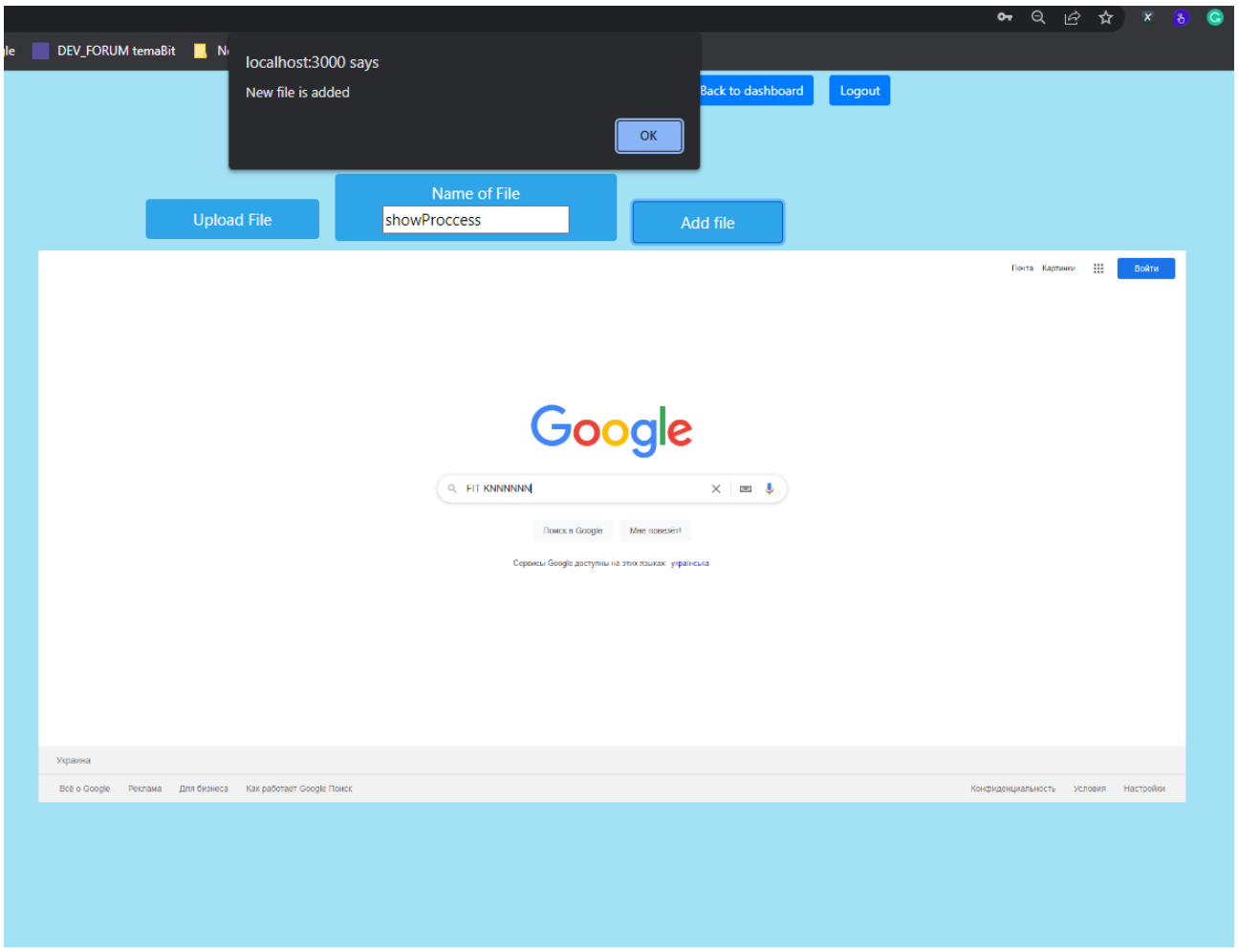


Рис.3.5 Візуалізація обраного документу та додавання

Після додавання файлу на сторінці викликається метод add (опис структури методу наведений у табл.3.2) з серверної частини для внесення мета-даних файлу до бази. Дані файлу приводяться до об’єкту для передачі у POST метод.

Таблиця 3.2. Опис методу додавання файлу

Method name	/file/add
Method type	POST
Request body	{fileName:’’,metaData:’’}
Response body	String

За назвою файлу можна визначити що він був доданий до бази даних (рис. 3.6)

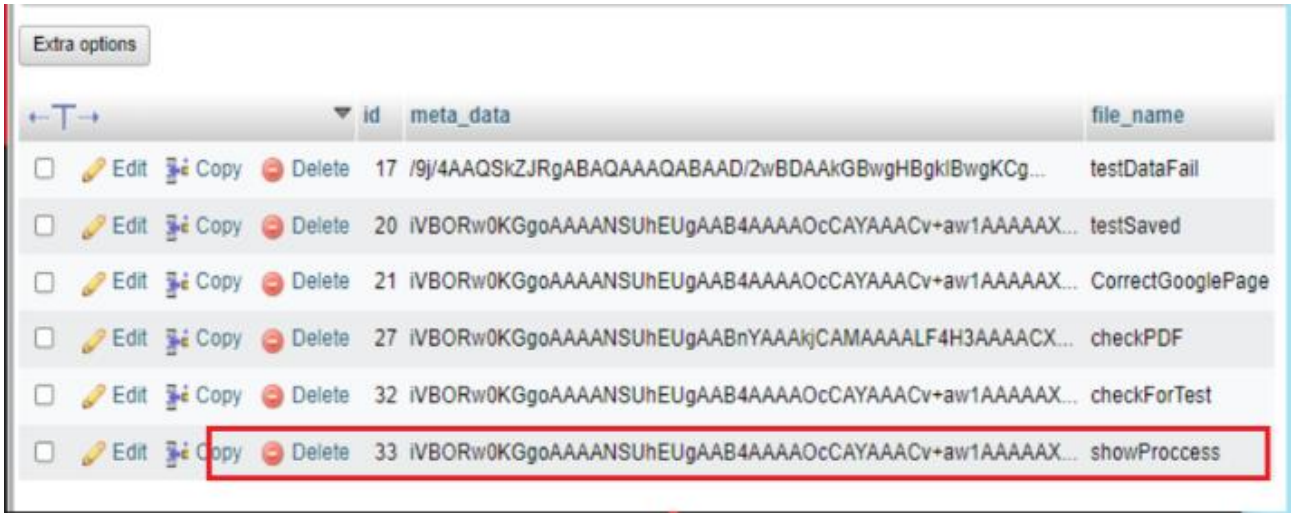


Рис. 3.6 Перевірка наявності нового запису у базі

А також можна побачити на сайті при натискані кнопки «back to dashboard», що у списку наявних шаблонів з’явився щойно доданий (рис 3.7)

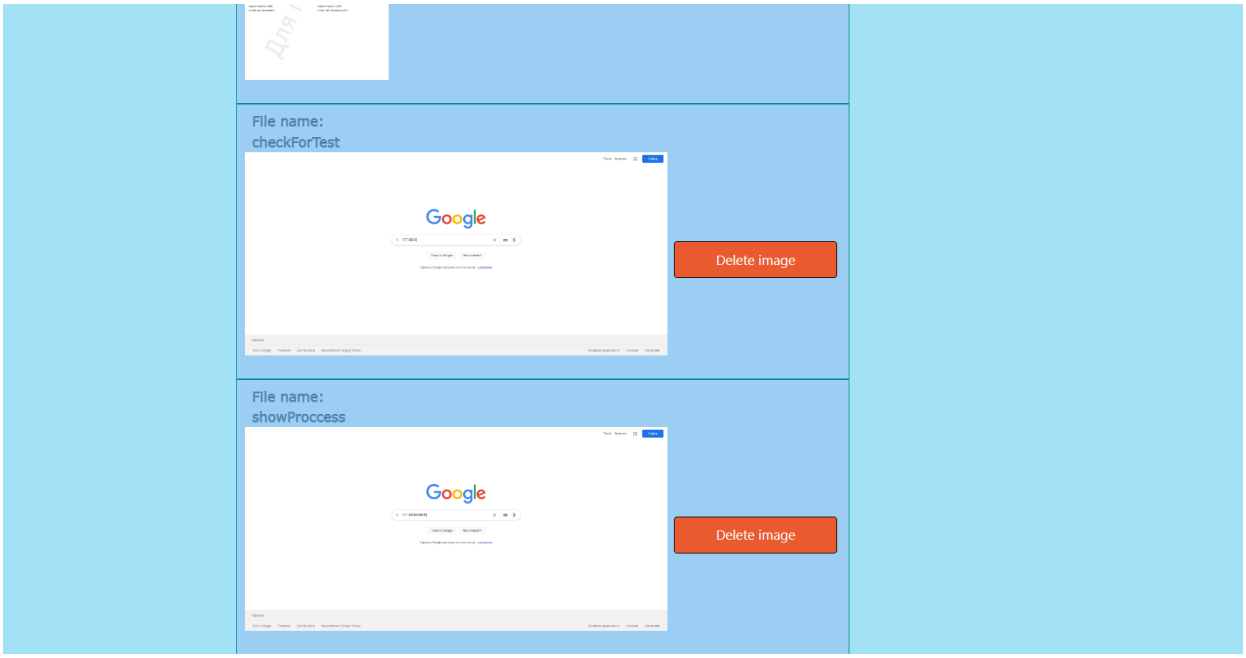


Рис. 3.7 Перевірка наявності нового шаблону на сайті

Якщо раптом користувач розуміє, що все ж таки додав не той файл, то він може натиснути кнопку «Delete image» навпроти необхідного шаблону після чого шаблон видалиться (рис 3.8 та рис. 3.9)

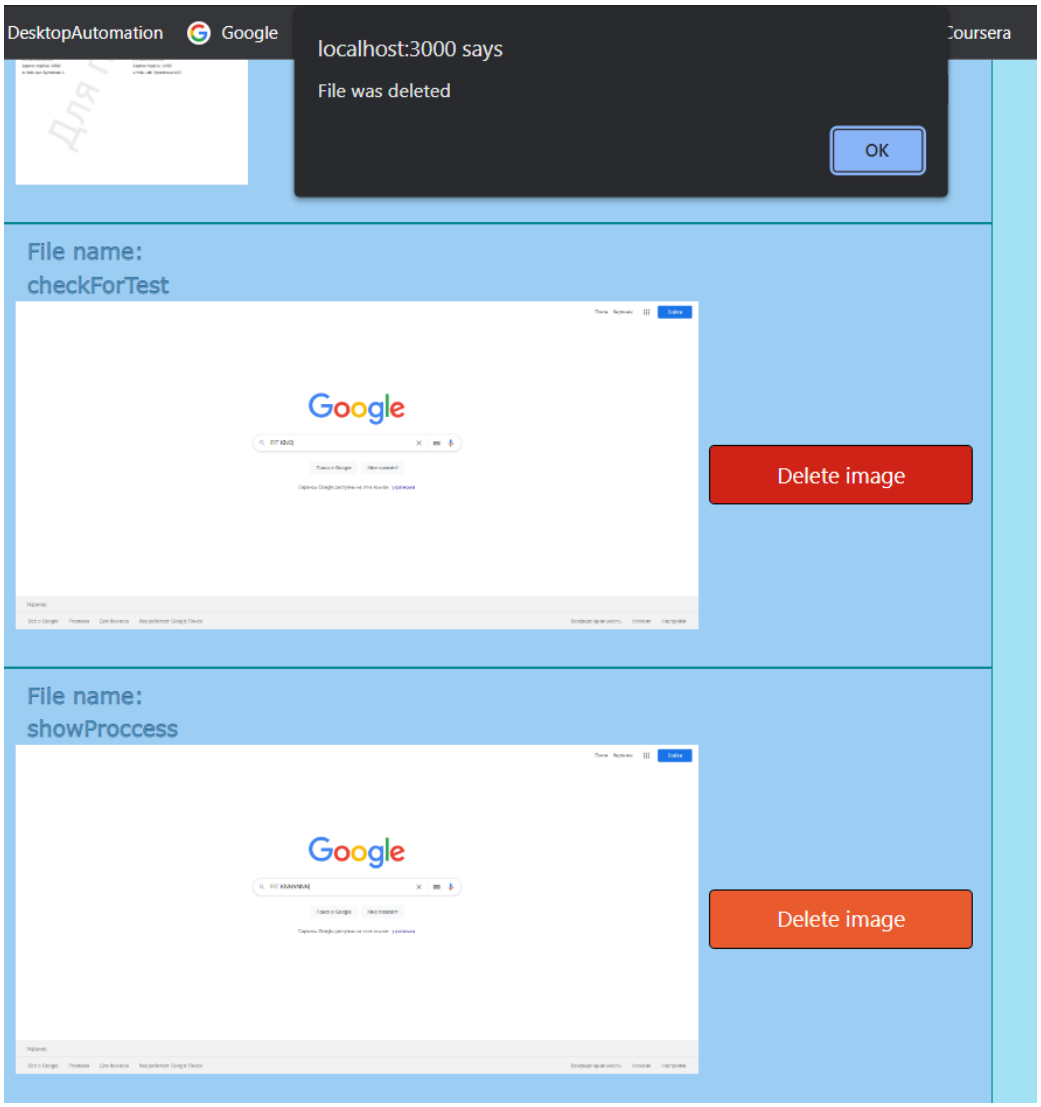


Рис. 3.8 Видалення файлу на сторінці

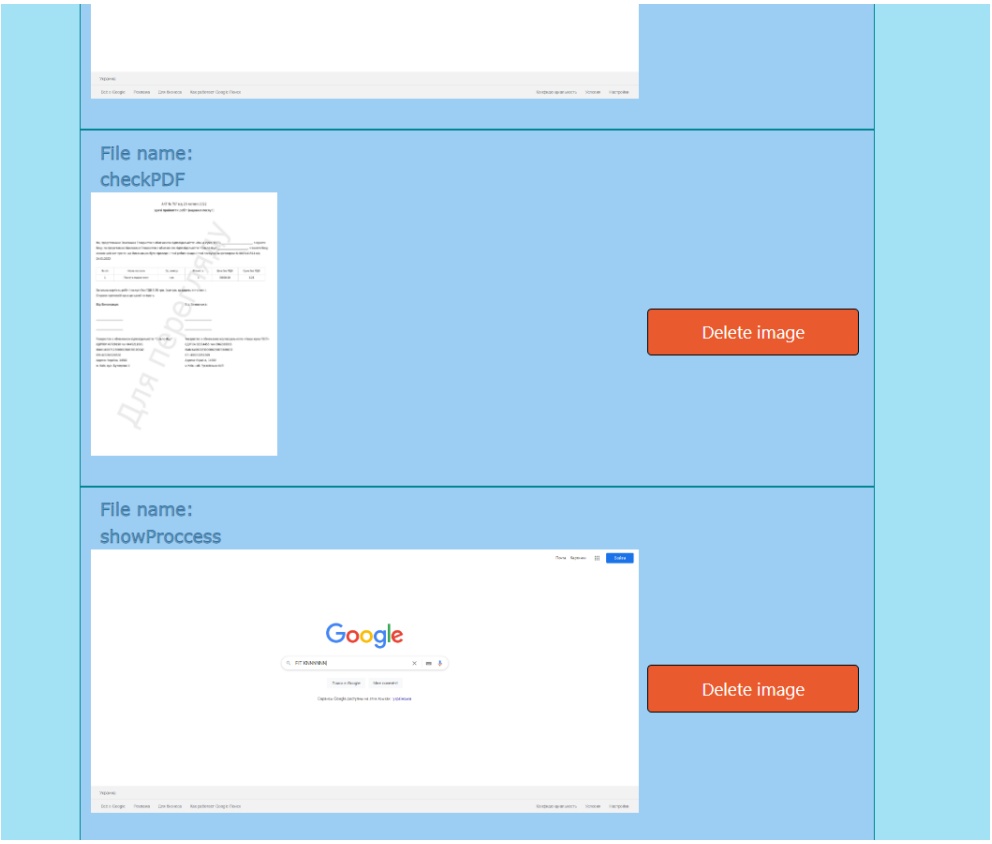


Рис. 3.9 Сторінка з видаленим файлом

Після того, як користувач сформував необхідний обсяг шаблонів для своєї роботи він може працювати з самим сервісом.

3.1.2 Реалізація підсистеми сервісу валідації

Основна частина мого проекту – сам сервіс для валідації – так само як і сайт розроблений за допомогою Spring Boot, оскільки дуже зручно описувати реалізацію для API за допомогою цього фреймворку.

Якщо брати до уваги, як саме в кінцевому результаті реалізований бекенд, то основний метод у нього один - /validation/this (опис структури методу наведений у табл.3.3).

Тіло запиту має наступні поля:

- nameOfTemplate – ім’я файлу шаблону, який користувач заздалегідь завантажив у адміністраторській частині;
- fileForCompareInBase – мета-дані файлу, що необхідно порівняти, які передають у форматі Base64

Тіло відповіді має наступні поля:

- result – результат валідації сервісом, реалізує собою речення «correct», «incorrect», «size not equal»;

- resultFileBase64– результат валідації. Сам результат представляється у вигляді зображення, а у відповіді приходять мета-дані цього зображення, щоб можна було привести ці дані до формату зображення.

Таблиця 3.3. Опис методу валідації файлу

Method name	/file/add
Method type	POST
Request body	{nameOfTemplate:'', fileForCompareInBase:''}
Response body	{result:'', resultFileBase64:''}

Як тільки сервіс отримав запит на валідацію, йому необхідно привести отримані мета-дані до вигляду зображення, щоб порівнювати саме них, для цього викликається метод generateImagefromBase (Лістинг 3.5)

Лістінг 3.5. Метод конвертації мета-даних до зображення

```
public static void generateImagefromBase(String dataInBase64,
String nameOfGeneratedFile){
        File file;
        file =  new File(nameOfGeneratedFile+".png");) {
            byte[] decoder =
Base64.getDecoder().decode(dataInBase64);

            fos.write(decoder);
            System.out.println("Image File Saved");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Після того як сервіс отримав мета-дані файлу від користувача, а також збереженного шаблону з бази даних та конвертував їх до зображення ініціалізується бібліотека валідації картинок ImageComparison [7].

Це відкрите рішення для валідації картинок у проектах, що написані на Java, тому було вирішено використовувати її для самого процесу валідації та отримання результату по ньому.

Бібліотека має чимало визначених параметрів (табл. 3.4) для роботи бібліотеки та коректної валідації. В подальших реалізаціях заплановано зробити профіль користувачу аби він мав змогу самому корегувати ці параметри, але поки вони будуть сталими на бекенді сервісу

Таблиця 3.4 Список параметрів для бібліотеки валідації

Параметр	Опис
threshold	Поріг, який означає максимальну відстань між нерівними пікселями. Можна змінити відповідно до розміру та вимог до зображення.
rectangleLineWidth	Ширина лінії, на якій зображений прямокутник.
destination	Файл призначення результату.
minimalRectangleSize	Номер мінімального прямокутника. Враховуйте як (ширина x висота). За замовчуванням це 1.
maximalRectangleCount	Максимальна кількість прямокутників, які будуть намальовані. Це означає, що отримаємо перші x найбільших прямокутників. Значення за замовчуванням -1, це означає, що будуть намальовані всі прямокутники.
pixelToleranceLevel	Рівень допуску пікселів. За замовчуванням це 0,1 -> 10% різниці. Значення можна встановити від 0,0 до 0,99.
drawExcludedRectangles	Прапорець із зазначенням малювання виключених прямокутників чи ні.
fillExcludedRectangles	Прапор із зазначенням заповнення виключених прямокутників чи ні.
percentOpacityExcludedRectangles	Бажана непрозорість заливки виключеного прямокутника.
fillDifferenceRectangles	Прапор із зазначенням заповнення різновидів прямокутників чи ні.
percentOpacityDifferenceRectangles	Бажана непрозорість заливки різницевого прямокутника.
allowingPercentOfDifferentPixels	Відсоток дозволених пікселів, які відрізняються, щоб валідація вважалася пройденою . Наприклад, відсоток пікселів, які ігнорують у порівнянні. Значення може бути від 0,0 до 100,00

differenceRectangleColor	Різниця кольору прямокутника зображення. За замовчуванням він червоний.

Після формування усіх вхідних даних, бібліотека потребує ініціалізації валідатора шляхом передачі у конструктор необхідних даних ImageComparison(expectedImage, actualImage)

expectedImage – файл шаблону, очікуваний результат

actualImage – фактичний файл, тобто той який користувач передав сервісу при запиті.

Після цього сервіс зберігає файл, що є результатом валідації н локальне сховище, викликається метод generateBaseFromImage, що тепер генерує мета-дані з файлу, та віддає тіло відповіді користувачу, куди включається мета-дані результуючого файлу, а також коментар валідатора відносно того як пройшла валідація.

3.2 Тестування та аналіз мікросервісу для валідації UI-інтерфейсу

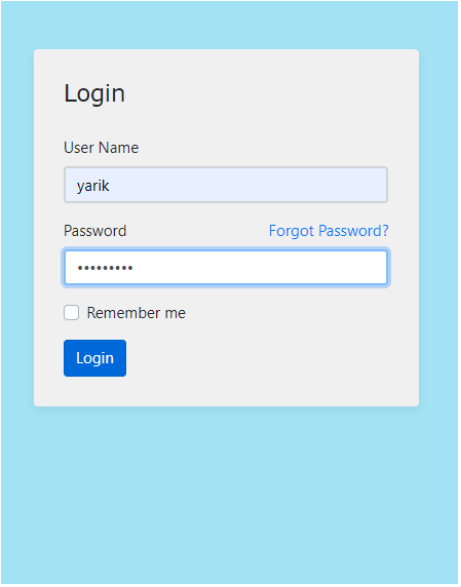
Розглянемо основні тестові випадки під час роботи мікросервісу:

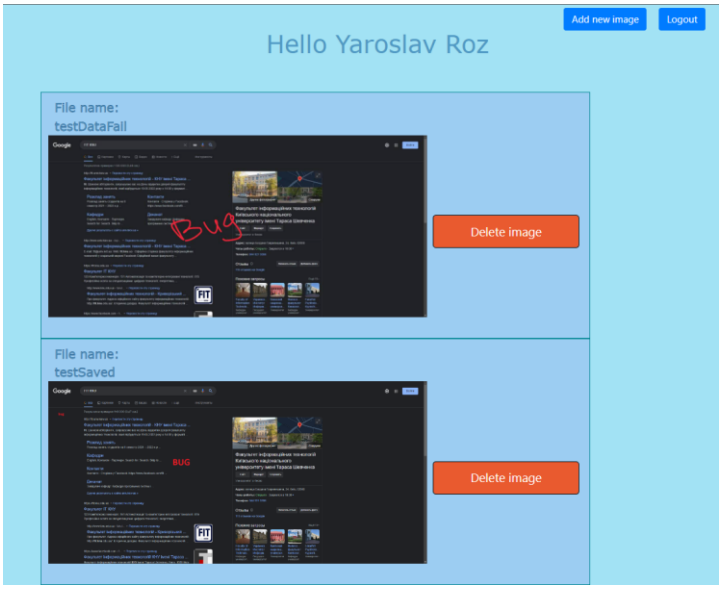
- Функціонал авторизації на сайті (таблиця 3.5);

Таблиця 3.5 Тестування функціоналу авторизації

Назва	Перевірка авторизації на сайті
Передумова	Відкрита сторінка авторизації
Кроки	1. Внести правильний логін 2. Внести правильний пароль 3. Натиснути Login
Очікуваний результат	Відкривається сторінка користувача з шаблонами.

Результат виконання авторизації наведено на рис. 3.10-3.11.





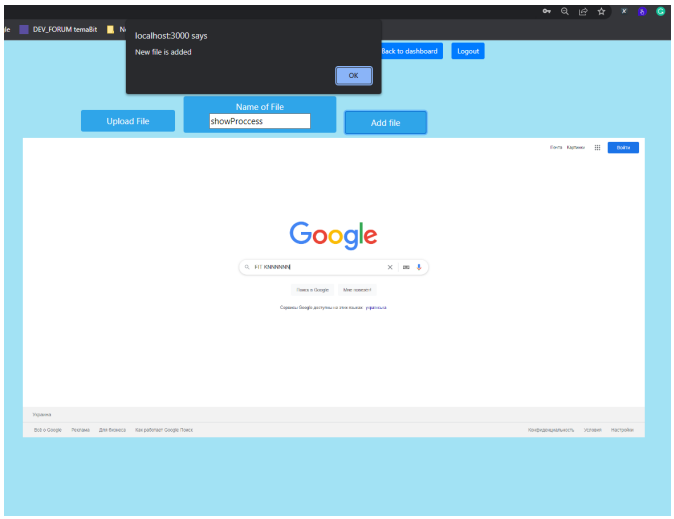
Рисунки 3.10-3.11 – Результат авторизації

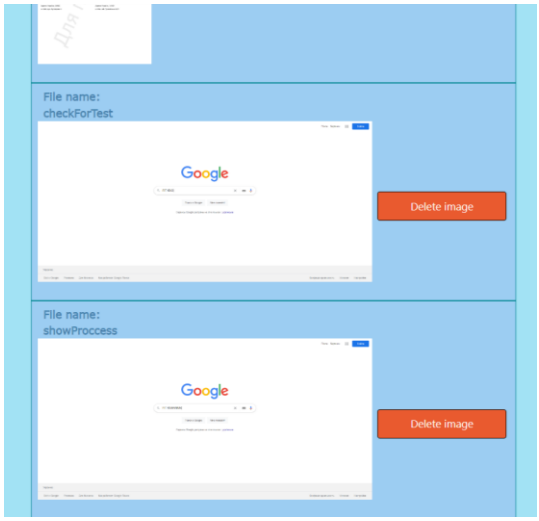
– Додавання нового файлу на сайті (таблиця 3.6);

Таблиця 3.6 Тестування функціоналу додавання файлу

Назва	Перевірка додавання файлу на сайті
Передумова	Відкрита сторінка додавання нового шаблону
Кроки	1. Натиснути «Upload file» 2. Вибрати файл на локальному сховищі 3. Вибрати ім’я для шаблону 4. Натиснути «Add file» 5. Повернутися на головну сторінку
Очікуваний результат	У кінці списку доданих шаблонів знаходится щойно доданий

Результат виконання додавання файлу наведено на рис. 3.12-3.13.





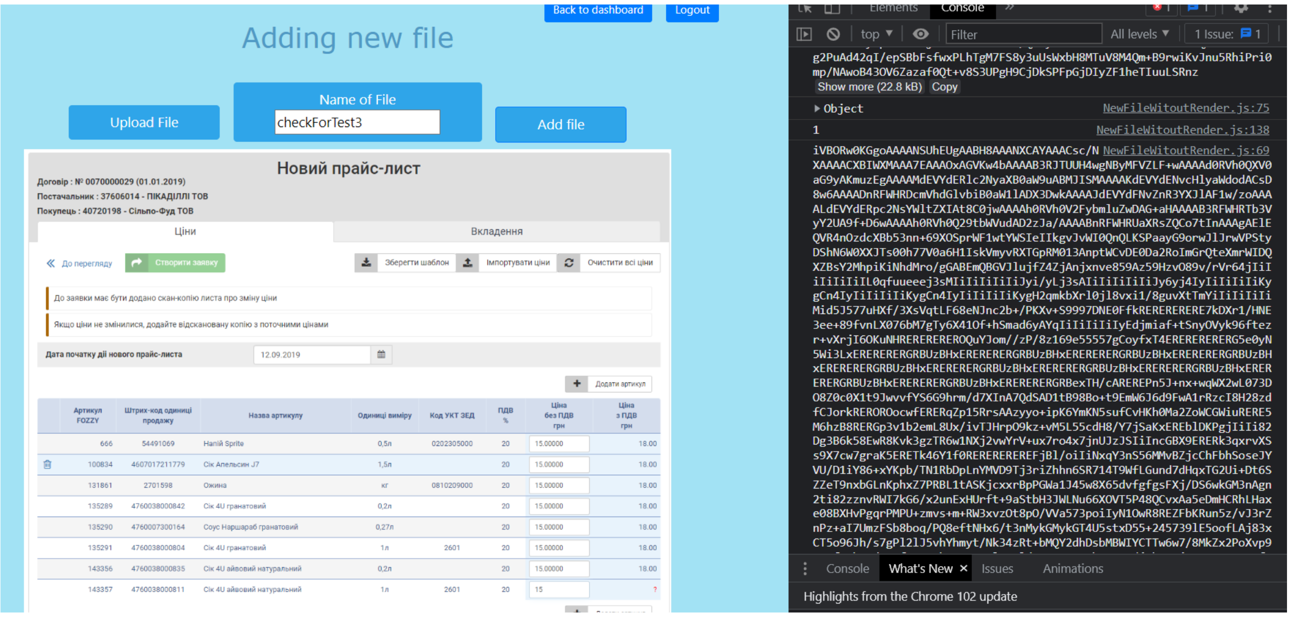
Рисунки 3.12-3.13 – Результат додавання нового файлу

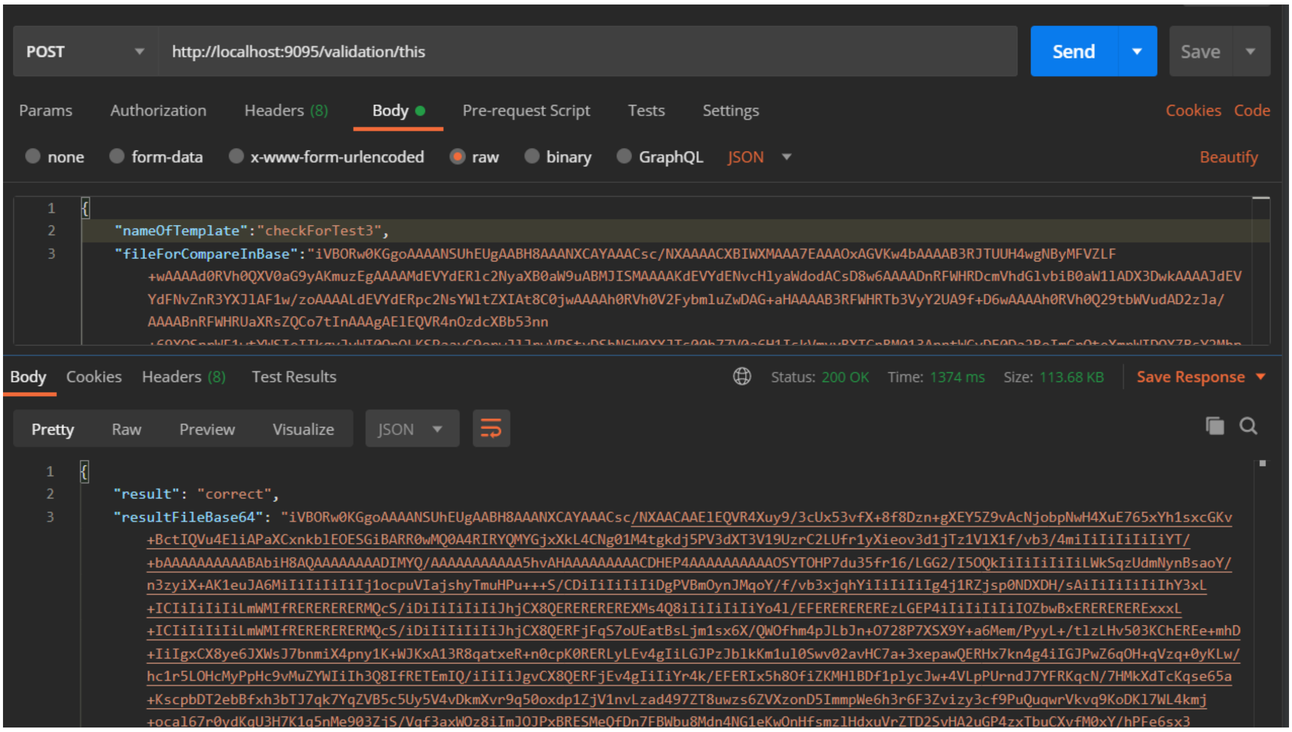
– Перевірка валідації у позитивному випадку (таблиця 3.7);

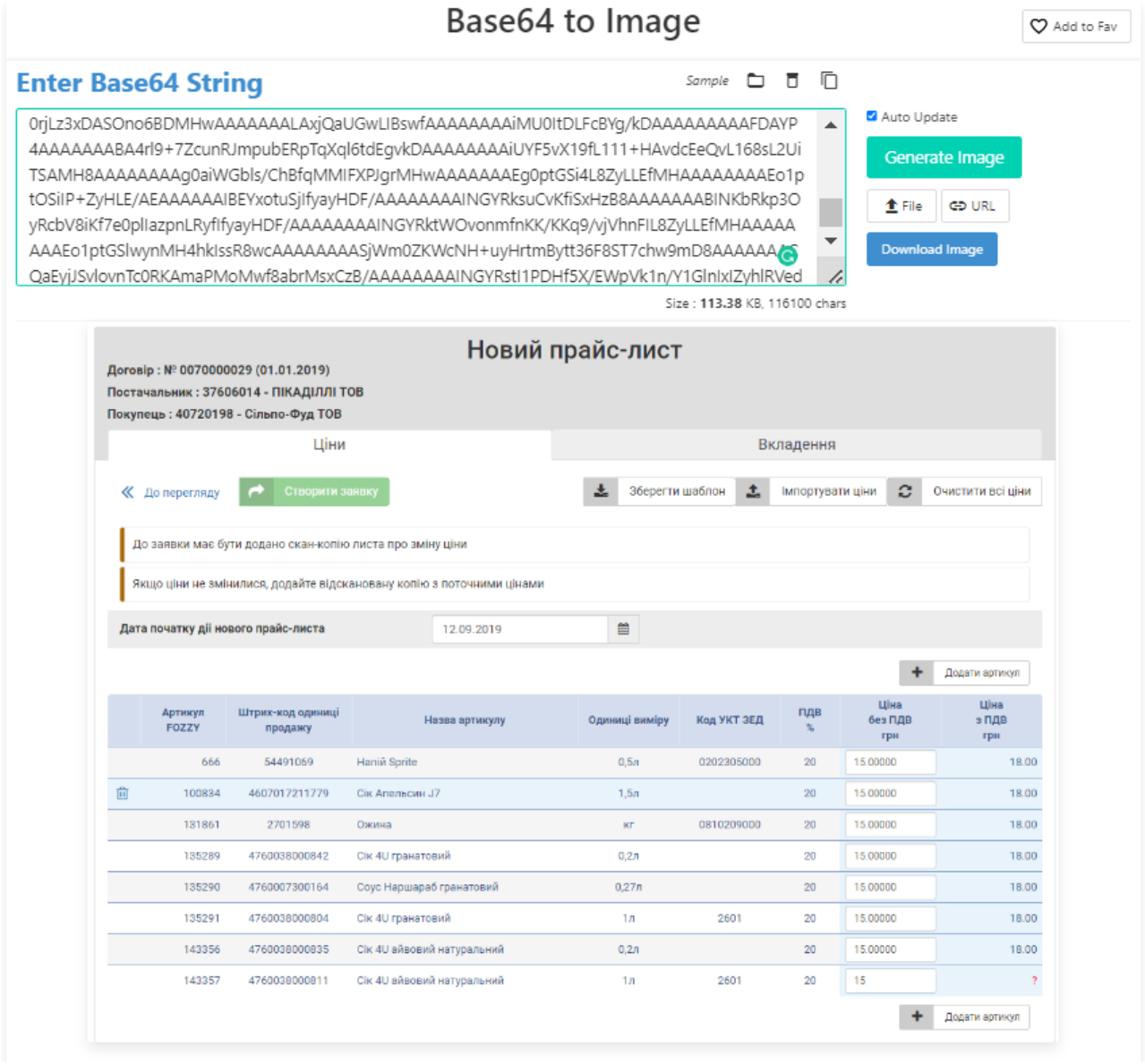
Таблиця 3.7 Тестування валідації у позитивному випадку

Назва	Перевірка валідації у позитивному випадку
Передумова	Завантажений шаблон для валідації у базу даних
Кроки	1. Обрати файл, що не відрізняється від шаблону в базі 2. Надіслати до сервісу запис на валідацію, передав мета-дані файлу та назву такого ж шаблону
Очікуваний результат	Сервіс повертає мета-дані файлу на якому немає відміток про помилки, а також поле result:”correct”

Результат виконання валідації у позитивному випадку наведено на рис. 3.14-3.16.


Back to dashboard
Logout
Adding new file
Upload File
Name of File
checkForTest3
Add file
Новий прайс-лист
Ціни
Вкладення
Object
NewFileWitoutRender.js:75
NewFileWitoutRender.js:138
NewFileWitoutRender.js:69
Show more (22.8 kB)
Copy
Console
What's New
Issues
Animations
Highlights from the Chrome 102 update


POST
http://localhost:9095/validation/this
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
"nameOfTemplate":"checkForTest3",
"fileForCompareInBase":"iVBORw0KGgoAAAANSUhEUgAABH8AAANXCAYAAACsc/NXAAACXBIWXMAAA7EAAAOxAGVKw4bAAAAB3RJTUUH4wgNByMFVZLF
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 1374 ms
Size: 113.68 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
"result": "correct",
"resultFileBase64": "iVBORw0KGgoAAAANSUhEUgAABH8AAANXCAYAAACsc/NXAACAAE1EQVR4Xuy9/3cUx53vfX+8f8Dzn+gXEY5Z9vAcNjobpNwH4XuE765xYh1sxcGkv



Рисунки 3.14-3.16 – Результат виконання валідації у позитивному випадку

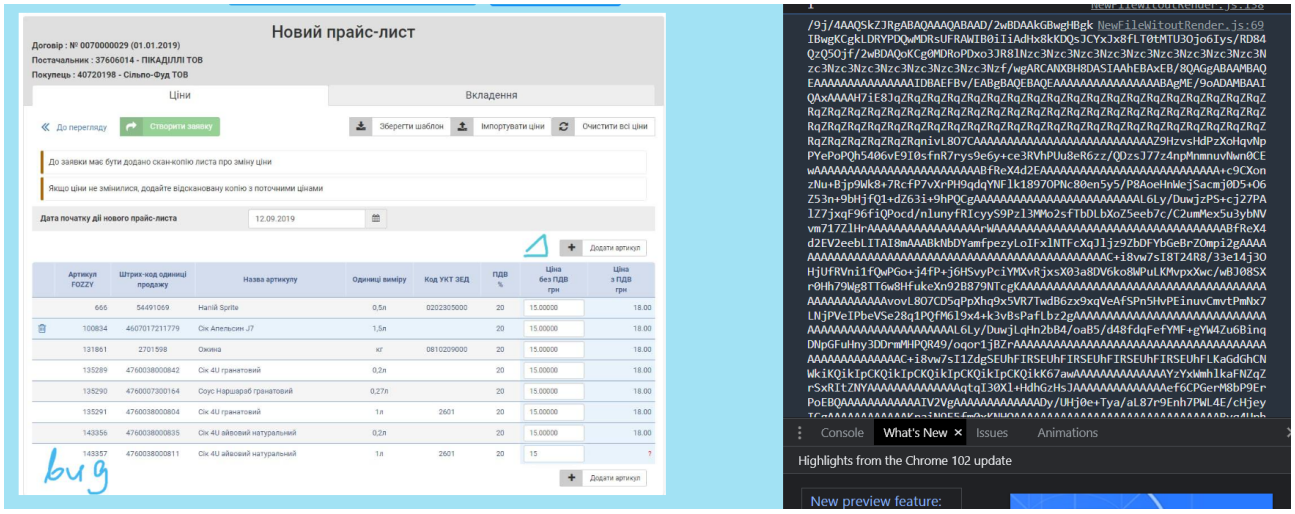
– Перевірка валідації у негативному випадку (таблиця 3.7);

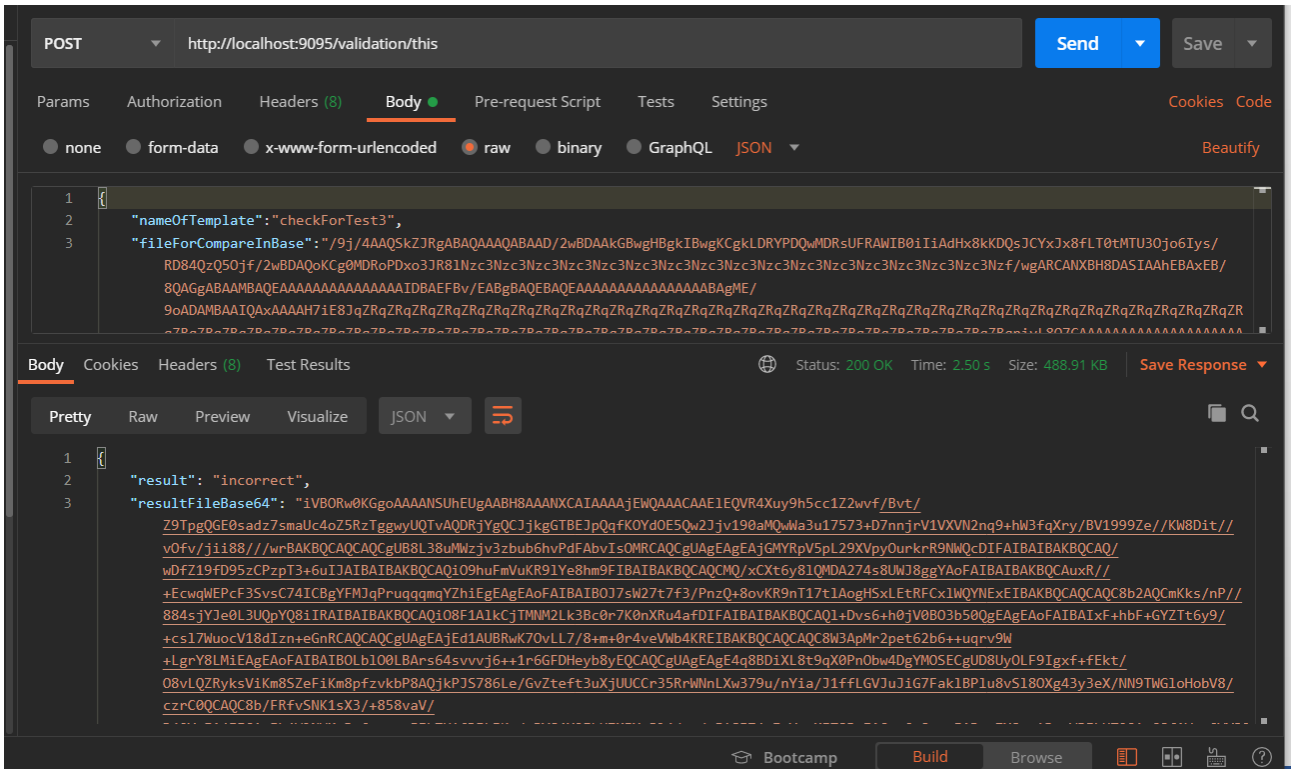
Таблиця 3.7 Тестування валідації у негативному випадку

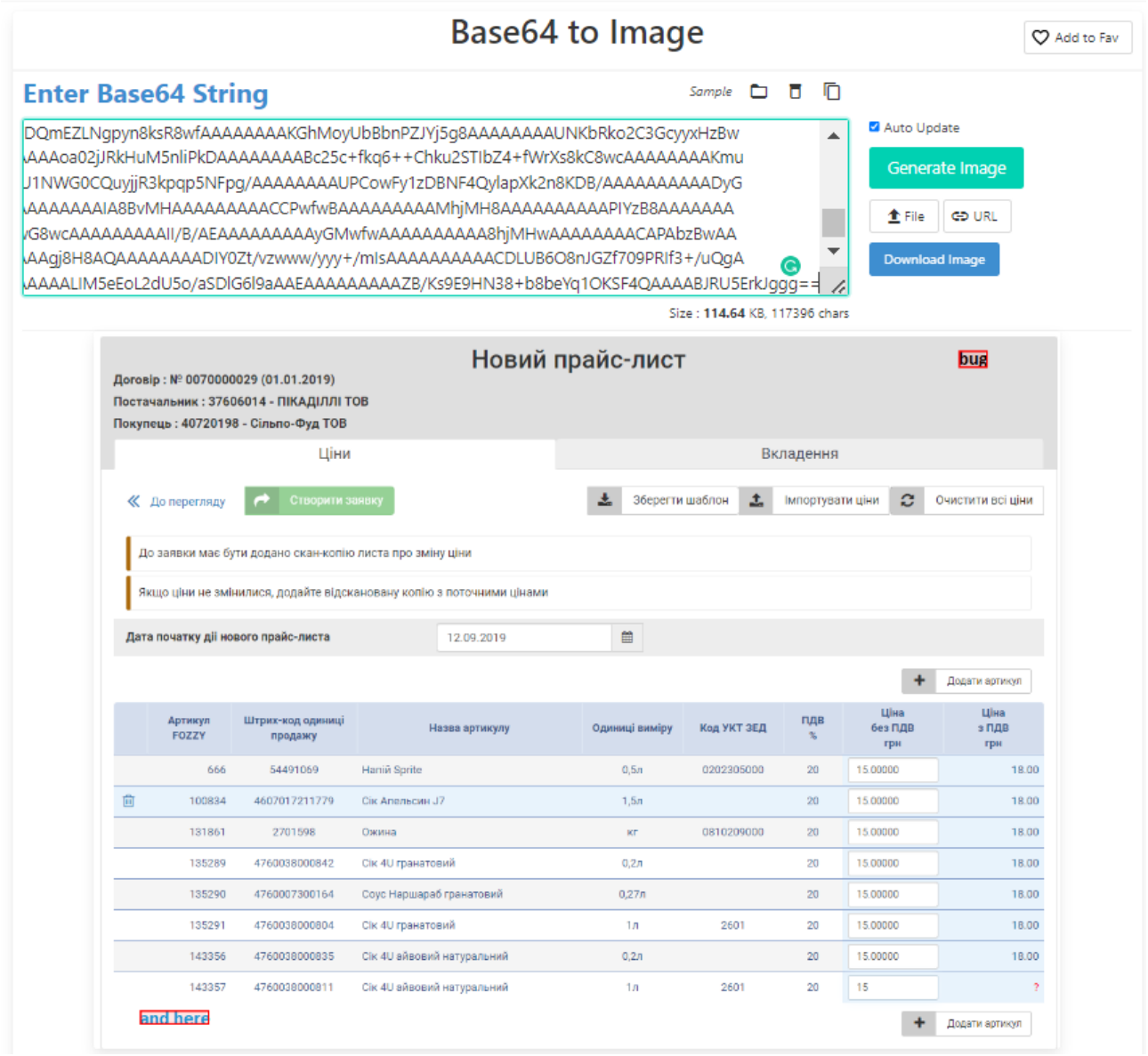
Назва	Перевірка валідації у негативному випадку
Передумова	Завантажений шаблон для валідації у базу даних
Кроки	1. Обрати файл, що відрізняється від шаблону в базі

	2. Надіслати до сервісу запис на валідацію, передав мета-дані файлу та назву такого відповідних файлу та шаблону
Очікуваний результат	Сервіс повертає мета-дані файлу на якому є відмітки про помилки, а також поле result:”incorrect”

Результат виконання валідації у негативному випадку наведено на рис. 3.17-3.19.


Новий прайс-лист
Договір : № 0070000029 (01.01.2019)
Постачальник : 37606014 - ПІКАДІЛЛІ ТОВ
Покупець : 40720198 - Сільпо-Фуд ТОВ
Ціни
Вкладення
До перегляду
Створити заявку
Зберегти шаблон
Імпортувати ціни
Очистити всі ціни
До заявки має бути додано скан-копію листа про зміну ціни
Якщо ціни не змінилися, додайте відсканованої копію з поточними цінами
Дата початку дії нового прайс-листа
12.09.2019
Додати артикул
bug
/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAkGBwgHBgk
Console
What's New
Issues
Animations
Highlights from the Chrome 102 update
New preview feature:


POST
http://localhost:9095/validation/this
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nameOfTemplate":"checkForTest3",
"fileForCompareInBase":"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU3Ojo6Iys/
RD84QzQ5Ojf/2wBDAQoKCg0MDRoPDxo3JR8lNzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzf/wgARCANXBH8DASIAAhEBAxEB/
8QAGgABAAMBAQEAAAAAAAAAAAAAAAAIDBAEFBv/EABgBAQEBAQEAAAAAAAAAAAAAAAAABAgME/
9oADAMBAAIQAxAAAAH7iE8JqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZRqZR
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 2.50 s
Size: 488.91 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"result": "incorrect",
"resultFileBase64": "iVBORw0KGgoAAAANSUhEUgAABH8AAANXCAIAAAAjEWQAAACAAElEQVR4Xuy9h5cc1Z2wvf/Bvt/
Z9TpgQGE0sadz7smaUc4oZ5RzTggwyUQTvAQDRjYgQCjkgGTBEJpQqfKOYdOE5Qw2Jjv190aMQwWa3u17573+D7nnjrV1VXVN2nq9+hW3fqXry/BV1999Ze//
/KW8Dit//
vOfv/jii88///wrBAKBQACQAgUB8L38uMWzjv3zbub6hvPdFAbvIsOMRCAQCgUAgEAjGMYRpV5pL29XVpyOurkrR9NWQcDIFAIBAIBAKBQCAQ/
wDfZ19fD95zCPzpT3+6uIJAIBAIBAKBQACQMQ7A274s8UWJ8ggYAoFAIBAKBQCAQCMQ//
+EcwqWEPcF3SvsC74ICBgYFMJqPruqqqmqYZhiEgEAgEAoFAIBAIBIBOJR9nT17tlAogHSxLEtRFCxlWQYNExEIBAKBQCAQCAQC8b2AQCkks/nP//
884sjYJe0L3UQpYQ8iIRAIBAIBAKBQCAQiO8F1AlkCjTMNM2Lk3Bc0r7K0nXRu4afDIFAIBAIBAKBQCAQl+Dvs6+h0jV0BO3b50QgEAgEAoFAIBAIxF+
+csl7WuocV18dIzn+eGnRCAQCAQCgUAgEAjEd1AUBRwK7OvLL7/8+m+0r4veVWb4KREIBAKBQCAQCAQC8W3ApMrMr2pet62b6++uqrv9W
+LgrY8LMiEAgEAoFAIBAIBLBlO0LBArs64svvvj6++1r6GFDHeyb8yEQCAQCgUAgEAgE4q8BDiXL8t9qX0PnObw4DgYMOSECgUD8UyOLF9Igxf+fEkt/
O8vLQZRyksViKm8SZeFiKm8pfzvkbP8AQjkPJS786Le/GvZteft3uXjUUCCr35RrWNnLXw379u/nYia/J1ffLGVJuJiG7FaklBPlu8vSl8OXg43y3eX/NN9TWGloHobV8/
czrC0QCAQC8b/FRfvSNK1sX3/+858vaV/



Рисунки 3.14-3.16 – Результат виконання валідації у негативному випадку

Висновок до третього розділу

Отже, в результаті виконання третього розділу дипломної роботи були подані написані для програми функції. Із поясненням роботи їх логіки, результатів роботи, наведений їх лістинг. Пояснене використання функції сторонніх бібліотек, їх синтаксис. Наведені приклади використання написаних функції у роботі програми, та як їх результат впливає на валідацію файлів.

Написані тест-кейси для створеного ПЗ, проведено тестування та аналіз сервісу валідації.

Висновок

У результаті виконання дипломної роботи було отримано мікросервіс, що дозволяє порівнювати свої файли з шаблонами, що були заздалегідь занесені в базу даних.

Було проведено аналітичний огляд дипломної роботи. Проаналізовано існуючі підходи, рішення та результати, що досягнуто на даний момент часу. Проаналізовані існуючі рішення щодо використання систем валідації у автоматизованих тестах.

Проведено функціональний аналіз, визначено вимоги до застосунку. Розроблено архітектуру застосунку, визначено підсистеми, модулі та зв’язки між компонентами системи. Розроблено застосунок із використанням мов програмування Java та ReactJS, та створені тест-кейси для тестування.

Отже, цей застосунок можна буде в подальшому використовувати для валідації інтерфейсів додатків, які покриваються автоматизованими тестами, оскільки один раз зафіксувавши стан кожної сторінки та завантаживши її до бази даних в подальшому буде надзвичайно просто валідувати ті самі сторінки, просто роблячи на кожній по знімку екрана автоматизованим тестом та відправляючи цей знімок до сервісу з назвою шаблону.

Хоча деякі з існуючих рішень можуть запропонувати більш широкий спектр функцій – більшість з них обмежена мовою, для якої рішення розроблялося. У моєму ж випадку, реалізацією мікросервісу може користуватися будь який продукт, що має реалізацію звертання до REST API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Куликов Святослав Тестирование программного обеспечения. Базовый курс/ Святослав Куликов., 2021. – Т. 3 : – 298 с.

2. Quality Assurance and Software Testing: A Brief History: [Електронний ресурс] // Режим доступу: https://saucelabs.com/blog/quality-assurance-and-software-testing-a-brief-history

3. Java | Definition & Facts: [Електронний ресурс] // Режим доступу: https://www.britannica.com/technology/Java-computer-programming-language

4. Java Programming Language - Baeldung: [Електронний ресурс]. Режим доступу: https://www.baeldung.com/java-history

5. Spring Boot: [Електронний ресурс]. Режим доступу: https://spring.io/why-spring

6. React | A JavaScript library for building user interfaces: [Електронний ресурс]. Режим доступу: https://reactjs.org/

7. Github.com | romankh3 | image-comparison : [Електронний ресурс]. Режим доступу: https://github.com/romankh3/image-comparison

Додаток А

Лістинг dashBoardControllData

```
import { render, shouldComponentUpdate } from '@testing-library/react';
import {getTokenOut} from './authenticationService';
import {b64toBlob} from './testOneMore';
import { memo, useState } from "react";
import React from "react";
import ReactDOM from 'react-dom';

import App from "../App";



import axios from 'axios';
import { Container } from 'react-bootstrap';

const url = 'http://localhost:8080/file/getAllFilesMetada';

//const [data,setData]=useState({});




export function Data() {
   //return
   const imageMap =
    getData(url).then(data =>
      data.forEach(element => {
      const contentType = 'image/png';

      const blob = b64toBlob(element, contentType);
      const blobUrl = URL.createObjectURL(blob);
```

```
        const img = document.createElement('img');
        img.style.width = "470px";
        img.style.height = "290px";
        img.alt = "wind icon";
        img.className ="enlarge-onhover"
        img.src = blobUrl;

        let parent = document.querySelector('#imagePlace');
        //let p = document.createElement(img);//document.createElement('p');
        //p.innerHTML = '!';// document.createElement(img)
        //parent.appendChild(p);
        parent.appendChild(img);
    }));
    console.log(imageMap);

}



export function DataAOXISFull(arr) {
    const arrOfPromises = new Map(Object.entries(arr));

    let parent = document.querySelector('#imagePlace');
    if(parent != null)
        parent.innerHTML = '';

    arrOfPromises.forEach(function(value, key) {
         const contentType = 'image/png';
```

```
const blob = b64toBlob(value.metaData, contentType);
const blobUrl = URL.createObjectURL(blob);

const img = document.createElement('img');
img.alt = "wind icon";
img.className ="portraitImg";
img.src = blobUrl;

const i = document.createElement('img');
i.src = blobUrl;

var modal = document.getElementById("myModal");
var modalImg = document.getElementById("img01");
var captionText = document.getElementById("caption");


img.onclick = function(){
    modal.style.display = "block";
    modalImg.src = this.src;
    modalImg.className ="portraitModal";
    captionText.innerHTML = value.fileName;
    modalImg.height = i.naturalHeight*0.93;
    modalImg.width = i.naturalWidth*0.9;
  }
    var span = document.getElementsByClassName("close")[0];
    span.onclick = function() {
    modal.style.display = "none";
    }

let parent = document.querySelector('#imagePlace');
```

```
let p = document.createElement('div');
let newdiv = document.createElement('div');

let btn = document.createElement('button');
btn.textContent = 'Delete image'

let ic = document.createElement('i');
ic.className = "fas fa-car";

btn.onclick =  function(e){deleteImage(value.id)};

let br = document.createElement('br');

btn.id = 'delimg';
btn.setAttribute('imageName',key);
p.className = "img-line";

var lab = document.createElement('label');
lab.className='FileNameLable';
lab.textContent= 'File name: ' + value.fileName;
lab.setAttribute('imageId',value.id);

p.id = "iline";

p.appendChild(lab);
newdiv.appendChild(br);
newdiv.appendChild(img);

newdiv.appendChild(btn);
```

```
        p.appendChild(newdiv);
        p.appendChild(br);
        parent.appendChild(p);

      })

}


export function deleteImage(imageId){
    deleteFile(imageId).then((response)=>{
        alert(response.data);
        document.location.reload();
    })
}

function houldComponentUpdate(){
    return false;

  }

async function getData(url){
  const response = await fetch(url
  , {
      method: 'GET',
      headers : {
          'Authorization':'Bearer '+getTokenOut(),
          'Content-Type': 'application/json',
          'Accept': 'application/json'
         }
```

```
    });
  const data = await response.json();
  const arrOfPromises = data.map(item => item.imageData
  );
  return Promise.all(arrOfPromises);
}



export const getMetaDataAOXIS=()=>{
    return axios({
        method:'GET',
        url:`${process.env.hostUrl||'http://localhost:8080'}/file/getAllFilesMetada`,
        headers:{
            'Authorization':'Bearer '+getTokenOut(),
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
    });

}

export const getFullDataAOXIS=()=>{
    return axios({
        method:'GET',
        url:`${process.env.hostUrl||'http://localhost:8080'}/file/getAll`,
        headers:{
            'Authorization':'Bearer '+getTokenOut(),
            'Content-Type': 'application/json',
            'Accept': 'application/json'
```

```
        }
    });

}

export const getMetaDataAOXISByUser=()=>{
    return axios({
        method:'GET',
        url:`${process.env.hostUrl||'http://localhost:8080'}/file/getAllFilesMetada`,
        headers:{
            'Authorization':'Bearer '+getTokenOut(),
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
    });

}

export const postAddNewFile=(fileData)=>{
    return axios({
        'method':'POST',
        'url':`${process.env.hostUrl||'http://localhost:8080'}/file/add`,
        headers:{
            'Authorization':'Bearer '+getTokenOut(),
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        },
        'data':fileData
    })
}
```

```
export const deleteFile=(fileId)=>{
    return axios({
        'method':'POST',
        'url':`${process.env.hostUrl||'http://localhost:8080'}/file/delete/`+fileId,
        headers:{
            'Authorization':'Bearer '+getTokenOut(),
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        }
    })
}
```