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

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«На правах рукопису»
УДК 004.054

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інтегровані інформаційні системи»
зі спеціальності 126 «Інформаційні системи та технології»
на тему: «Система збору та представлення даних про виконання автоматизованих тестових сценаріїв»

Виконав (-ла):
студент (-ка) VI курсу, групи ІА-з92мп
Вознюк Оксана Олегівна _____

Керівник:
к.т.н., доцент кафедри АУТС
Букасов Максим Михайлович _____

Рецензент:
_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – другий (магістерський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інтегровані інформаційні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Вознюк Оксани Олегівні

1. Тема дисертації «Система збору та представлення даних про виконання автоматизованих тестових сценаріїв», науковий керівник дисертації Букасов Максим Михайлович, к.т.н., доцент кафедри АУТС, затверджені наказом по університету від «26» 10 2020 р. №3132-с

2. Термін подання студентом дисертації _____

3. Об’єкт дослідження: автоматизоване тестування програмного забезпечення.

4. Вихідні дані

5. Перелік завдань, які потрібно розробити: розробка системи, що відтворює збір даних про автоматизовані тестові сценарії та повертає їх, структуруючи зі заданими метриками.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу : структурна схема, діаграма прецедентів, діаграми класів, діаграма сутність-зв’язок, діаграма послідовності, схеми алгоритмів роботи системи.

7. Орієнтовний перелік публікацій

8. Дата видачі завдання _____01.09.2020_____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Дослідження предметної області та існуючих рішень.	14.09.2020	
2.	Визначення функціональних вимог до системи	18.09.2020	
3.	Визначення технологій розробки	01.10.2020	
4.	Побудова архітектури системи	22.10.2020	
5.	Тестування системи	29.10.2020	
6.	Розробка стартап-проекту	11.10.2020	
7.	Оформлення документації	27.11.2020	

Студент Вознюк О.О.

Науковий керівник Букасов М.М.

РЕФЕРАТ

Магістерська дисертація: 110 с., 33 рис., 42 табл., 9 додатків, 17 джерел.

Актуальність. Для успішного запуску автоматизованих тестових сценаріїв для тестування програмного забезпечення необхідне виконання ряду умов, але часто не всі умови є підконтрольні тестувальнику, окрім цього «людський фактор» також має своє місце при створені тестових сценаріїв. Як наслідок – актуальна проблема нестабільних автоматизованих тестових сценаріїв, яка не має універсального рішення – унікальність тестових сценаріїв та їх тісний зв’язок зі специфікою продукту, що тестується на дають побудувати єдиний підхід. Тому не менш актуальними є інструменти що призначенні для пришвидшення та полегшення процесу стабілізації.

Реалізована система дозволяє користувачу отримувати актуальні дані про результати виконання автоматизованих тестових сценаріїв, у вигляді звіту, що формується за вказаною користувачем метрикою. Такий звіт містить всю необхідну для початку стабілізації інформацію, що надає прозору картину стану тестових сценаріїв.

Метою роботи виступає пришвидшення процесу стабілізації тестових сценаріїв шляхом надання користувачу статистики виконання автоматизованих тестових сценаріїв за заданими метриками.

Об’єктом дослідження є процес автоматизованого тестування програмного забезпечення.

Предметом дослідження виступає обробка результатів виконання автоматизованих тестових сценаріїв.

Ключові слова: Автоматизоване тестування, виведення статистики, збір статистики, автоматизація, результат тестування.

ABSTRACT

The dissertation: 110 pages, 33 drawings, 42 tables., 9 appendixes, 17 sources.

Relevance. Successful launch of automated test scenarios for software testing requires a number of conditions, but often not all of the conditions are under tester’s control. In addition, so called "human factor" also has a place in the creation of test scenarios. As a result, there is a problem of unstable automated test scenarios, which does not have a universal solution, the uniqueness of test scenarios and their close connection with the specifics of the product being tested do not allow to build a unified approach. Therefore, no less relevant are the tools designed to speed up and facilitate the stabilization process.

The implemented system allows the user to obtain up-to-date data on the results of automated test scenarios, in the form of a report generated by the metric specified by the user. This report contains all the information needed to start stabilizing, which provides a transparent picture of the status of test scenarios.

The purpose of the work is to accelerate the process of stabilization of test scenarios by providing the user with statistics on the execution of automated test scenarios according to the specified metrics.

The object of research is the process of automated software testing.

The subject is processing the result of automated test scenarios run.

Keywords: Automated testing, statistics output, statistics collection, automation, test result.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

ООП – Об’єктно-Орієнтовне Програмування

ПЗ – Програмне забезпечення

СУБД – Система управління базою даних

API – Application Programming Interface

CI/CD – Continuous Integration/ Continuous Delivery

DTO – Data Transfer Object

JSON – JavaScript Object Notation

JRE – Java Runtime Environment

JVM – Java Virtual Machine

MVP – Minimal Value Product

POJO – Plain Old Java Object

UML – Unified Modeling Language

XML – Extensible Markup Language

XSS – Cross-Site Scripting

SOLID – Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion

SWOT – Strength, Weakness, Opportunities, Threats

ВСТУП

Сфера інформаційних технологій оточує сучасну людину на протязі всього життя. Зі стрімким розвитком технологічного прогресу, використання різноманітного програмного забезпечення стало невід'ємною частиною буденного життя, що, як наслідок, породжує обсяжний ринок сервісів, призначених для задоволення навіть найбільш специфічних та вузькоспеціалізованих потреб.

Найчастіше для задоволення буденних прагнень людини вже було створено відповідне програмне забезпечення. Наприклад, у відповідь на прагнення підтримувати швидкий асинхронний зв'язок з близькими людьми було створено велику кількість, так званих, «месенжерів», найпопулярніші з яких нараховують активність до 2 млрд. користувачів в місяць. Користувач має можливість обрати найбільш підходяще та просте у використанні рішення за власним смаком, що, враховуючи кількість доступних рішень, спонукає виробників програмного забезпечення підтримувати свій продукт на високому рівні якості.

Як наслідок, можна виділити мінімальні загальні критерії, яким має відповідати програмне забезпечення щоб утримувати свої позиції на ринку – високий рівень захисту інформації, повноцінне функціональне оснащення, стабільність продукту, тощо.

Тенденції сучасного ринку розробки програмного забезпечення все більше підвищують вагомість фактору стабільності, що включає в себе мінімізацію вірогідності появи дефектів продукту, відсутність критичних дефектів основного функціоналу, що має виконуватись з кожним новим оновленням продукту. Для реалізації даних цілей команда розробників, як правило, включає тестувальників, які забезпечують якість ПЗ на кожному етапі його розроблення.

Для тестування програмного забезпечення, зокрема, за допомогою автоматизованих тестових скриптів, існує багато сформованих методологій, шаблонів тестування, критеріїв оцінки, тощо. Для досягнення найвищої продуктивності автоматизованого тестування, тестові скрипти мають відповідати ряду вимог, серед яких: стабільність, атомарність, гнучкість в розширенні,

оптимальність, відповідність вимогам бізнесу, та інші.

Як інструкції для виконання більшості вимог існує набір правил та концепцій побудови проектів автоматизованого тестування, однак для забезпечення вимоги стабільності скриптів практично неможливо створити загальні правила, через занадто велику вагу фактору специфіки продукту, що тестується.

Стабільним можна назвати тестовий скрипт, який в результаті виконання не буде відмічений помилково пройденим, або помилково порушеним, а, незалежно від кількості запусків, при незмінних зовнішніх умовах тестування, буде відмічений завжди одним і тим самим результатом.

Через відсутність загальноприйнятих підходів підтримки стабільності тестів, це питання вирішується для кожного продукту індивідуально та часто займає забагато часу, оскільки перш за все необхідно вирішити, які саме тести потрібно стабілізувати – проблема нестабільності скриптів зазвичай стає помітною тільки протягом певного часу регулярних спостережень за результатами проходжень тестових сценаріїв. Далі потрібно визначити, в чому полягає проблема, які неточності коду провокують хибну поведінку скрипту, та насамкінець, знайти рішення без втрати функціональної цінності тестового сценарію.

Метою даної роботи стала розробка додатку, що зможе в автоматичному режимі збирати дані про виконання автоматизованих тестових сценаріїв, зберігати їх, та на вимогу відображати, відповідно до заданих метрик, що має суттєво пришвидшити процес стабілізації тестових скриптів та, як наслідок, забезпечити прискорити та спростити етап тестування в процесі розробки ПЗ.

Завдання, що винесені на дану роботу, наступні:

- аналіз подібних рішень на ринку, їх переваг та недоліків. Визначення, на основі цього аналізу, основних вимог до користувацького функціоналу та інтерфейсу додатку;
- вибір технологій для розробки програмного забезпечення, які дозволять якнайкраще виконати поставлені вимоги;
- розробка логіки та написання програмного коду застосунка;
- тестування та відлагодження додатку.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

1.1.1 Опис процесу розробки програмного забезпечення

Процес розробки програмного забезпечення не дозволяє виокремити чітко задану послідовність кроків до виконання, оскільки велику вагу має фактор індивідуальності продукту, що розробляється. Однак можна виділити загальні етапи процесу розробки, що в тій чи іншій мірі справедливі практично для кожного продукту:

- аналіз та формування вимог до нового, або існуючого функціоналу;
- планування дати початку роботи над ним;
- імплементація поставленої задачі;
- тестування розробленого функціоналу;
- інтеграційні роботи, впровадження програмного коду та налаштувань;
- підтримка розробленого функціоналу.

Існує деяка кількість узагальнених моделей, які оперують даними етапами, однак встановлюють різні критерії виконання поставленої задачі.[1] Моделі досить точно описані, однак повне слідування ним недосяжне, вони суто теоретичні.

Наприклад, модель «Водоспад» (також відома як «Каскадна» модель) передбачає суворо послідовне виконання кожного з зазначених етапів, всі вимоги мають бути задокументовані та незмінні, а кожен етап вважається виконаним тільки коли він відповідає всім вимогам. Використовуючи таку модель, можна оцінити час виконання задачі та її складність ще на перших етапах процесу розробки. Однак, така модель зручна у використанні тільки для невеликих систем, в іншому випадку – значний ризик невідповідності поставленим термінам виконання задачі.

«Ітеративна» модель передбачає розбиття масивної задачі на деяку кількість невеликих, кожна з яких буде представляти невеликий проект. Завершення кожної ітерації супроводжується доставленням коду. Для такої моделі характерна відсутність або неточність оцінки складності поставленої задачі на перших ітераціях розробки.

Інша модель – «Спіральна» пропонує розділити весь процес розробки

функціоналу на невеликі ітерації зазначених етапів що повторюються, таким чином процес полегшується, з'являється можливість швидше отримати відгуки користувачів та вжити необхідні заходи. Візуалізація спіральної моделі розробки зображено на Рисунку 1.1. Хоча дана модель є більш гнучкою з точки зору кінцевого результату, вона також має недоліки – поставлене технічне завдання може змінюватись із кожною ітерацією, що може знижувати якість програмного коду.

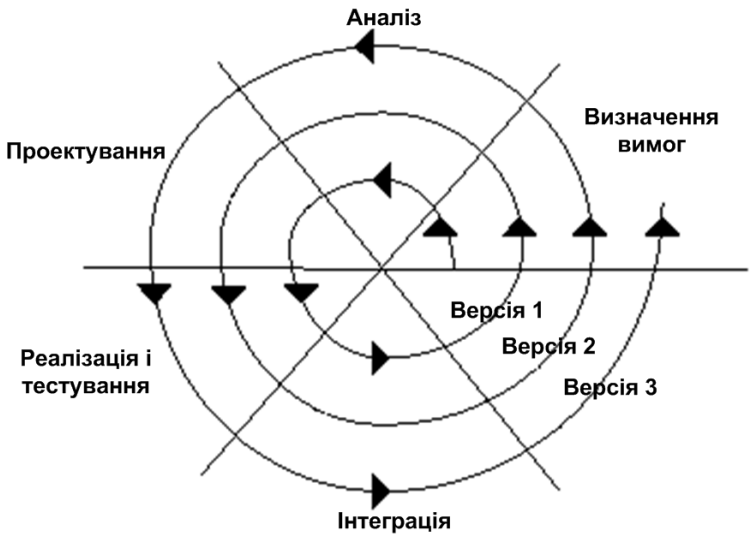

Рисунок 1.1 – Спіральна модель розробки ПЗ

Для проекту, який вже має основний функціонал та ведеться робота над його покращенням або нарощуванням нового виділяють загальні етапи розробки. Першочергово встановлюється, який функціонал необхідний продукту для успішного розвитку. Для цього потрібно проаналізувати ринок та рішення конкурентних продуктів, встановити користувацькі потреби, тощо. Даний етап зазвичай виконується бізнес-аналітиками та менеджерами. Як тільки бажана функціональність сформульована розпочинається підготовка її специфікації – документів із детальним описом, враховуючи максимальну варіативність виключних випадків. В цьому також бере участь команда розробників та тестувальників, оскільки вони також можуть помітити неточності із точки зору імплементації задачі. Одночасно із цим також можуть вирішуватись загальні алгоритми та архітектурні рішення проекту. В результаті такий документ виступає технічним завданням на яке спирається команда

розробників в процесі імплементації та команда тестувальників визначаючи ступінь відповідності актуального результату із заданими вимогами. Окрім цього, команда тестувальників також починає формувати тестову документацію – опис процесу тестування – тест-плани, тест-кейси, чек-списки, тощо.

Коли технічне завдання сформоване команда розробників розпочинає роботу над імплементацією поставленої задачі. Часто одночасно із цим тестувальники-автоматизатори також оформлюють каркас автоматизованих тестових сценаріїв, які будуть завершені після закінчення розробки заданого функціоналу. Такий підхід значно пришвидшує процес тестування та полегшує подальшу підтримку функціоналу.

Після того як завершується етап розробки – команда тестувальників розпочинає мануальне та автоматизоване тестування нового функціоналу. При появі дефектів нового функціоналу або створеного раніше (такі дефекти називають регресійними) команда розробників має їх вирішити на даному етапі. Часто на етапі тестування відбувається верифікація продукту бізнес-аналітиками або менеджерами проекту на відповідність основним вимогам.

Наступним кроком виступають інтеграційні роботи – нова кодова база не повинна пошкодити існуючу, відслідковується, що програмний код нового функціоналу та будь-які інші пов'язані із ним конфігураційні налаштування (bpmn-файли, змінні оточення в хмарних сховищах, тощо) не конфліктують із існуючими раніше. Також відбувається процес доставки програмного коду на оточення де із ним взаємодіють користувачі продукту. До цього етапу поставлена задача вже має мати відповідні автоматизовані тестові сценарії, що будуть верифікувати відповідність функціоналу поставленим вимогам протягом подальшої розробки.

1.1.2 Опис процесу тестування програмного забезпечення

Процес тестування програмного забезпечення включає широкий комплекс робіт, що виконуються для верифікації стану продукту, визначення його відповідності поставленим вимогам. Існує багато класів та підкласів методологій

тестування, що об'єднуються за характером процесу що тестується.[2] До найчастіше типів, що найчастіше використовуються можна віднести наступні:

- функціональне тестування;
- smoke-тестування;
- тестування продуктивності;
- тестування безпеки;
- тестування локалізації;
- інтеграційне тестування, тестування сумісності.

Функціональне тестування зазвичай є базовим класом тестування, оскільки воно передбачає перевірку відповідності роботи функціоналу ПЗ поставленим вимогам. Воно може проводитись на різних рівнях системи – наприклад, на модульному (компонентному) рівні, коли кожен сервіс тестується ізольовано від інших, що підвищує надійність результатів тестування, та, в противагу, тестування на рівні системи, коли всі сервіси тестуються одночасно у зв'язку між собою.

Smoke-тестування – окремий специфічний підклас функціональних тестів, мета яких – найшвидшим чином перевірити базовий функціонал продукту. Зазвичай до такого типу тестування прибігають відразу після релізу продукту для найшвидшої перевірки працеспроможності системи.

Тестування продуктивності дозволяє перевірити здатність системи витримувати передбачене навантаження, навантаження може вимірюватись різними метриками, в залежності від специфіки продукту – наприклад, кількістю унікальних користувачів, що одночасно користуються сервісом, кількістю одночасно отриманих запитів сервером, тощо. Тестування продуктивності включає два типи – тестування навантаження та стрес-тестування.

Тестування навантаженням проводиться для оцінки часу реакції системи при очікуваному рівні навантаження, в якому перебуває система в продакшн-оточенні. Для емуляції такого навантаження існує ряд допоміжного ПЗ та методололгій. Мета стрес-тестування – встановити характеристики системи та її працеспроможність під час критичного рівня навантаження, або визначення такого рівня.

Тестування безпеки полягає у визначені ризиків продукту, що пов’язані із

захистом системи від вірусного ПЗ, хакерських атак, несанкціонованого доступу до персональних даних користувачів, тощо. Це також включає тестування на предмет відсутності вразливостей системи, наприклад XSS, ін’єкції коду, тощо. Тестування локалізації менш розповсюджений тип тестування у порівнянні із попередніми. Він полягає у врахуванні відмінностей мов, що підтримуються системою та стилю оформлення додатків різних країн, наприклад, розміщення тексту не зліва направо, а зверху вниз.В процесі інтеграційного тестування перевіряється обернена сумісність версій системи (у разі появи критичних дефектів, встановлення попередньої версії продукту має бути без втрати фунціоналу). Також перевіряється коректний зв’язок між внутрішніми сервісами системи, та взаємодію системи зі стороннім ПЗ. Окрім цього, можна виділити дві категорії тестування в які входять всі класи, в залежності від того чи використовуються автоматизовані тестові сценарії – мануальне та автоматизоване тестування.

Автоматизоване тестування передбачає написання автоматизованих тестових сценаріїв використовуючи мову програмування. Деякі види тестування доцільно виконувати як мануально так і автоматизовано, наприклад, функціональні тести (написані скрипти виступають регресійним тестуванням), smoke-тести також часто автоматизують, що дозволяє зручно їх викликати в автоматичному режимі системами CI/CD – Continius Integration / Continius Development.Для таких цілей можна легко використовувати практично будь-яку мову програмування, однак найчастіше намагаються використовувати мову, на якій написаний програмний код продукту – це дозволяє зручно та консистентно перевикористовувати програмні блоки. Також, прийнято використовувати і звичайні практики програмування – як принципи SOLID, імплементація шаблонів проектування при необхідності – існують шаблони саме для автоматизованого тестування, як Page Object Pattern, окрім них часто використовуються і стандартні.

Автоматизовані тестові сценарії часто розділяються на три основні категорії:

- тестування користувацького інтерфейсу;
- тестування серверної частини продукту;
- тестування продуктивності.

Таке розділення дозволяє спроектувати тести атомарно та ізольовано, що підвищує їх стабільність та коректність. Також, це дозволяє розділити тести на різні проекти, що підвищує читабельність коду, зменшує ймовірність некоректного використання сервісів при написані тестових сценаріїв. Однак однією із проблем автоматизованого тестування виступає проблема нестабільності тестових сценаріїв. Вона може постати через різноманітні причини, як брак часу на коректне написання або налаштування тестового сценарію, через що обирається швидке рішення яке може бути не ідеальним, нестабільність оточення для виконання тестових сценаріїв, некоректність перевірок всередині тестових сценаріїв, та інші. Якою б не була причина появи нестабільності завжди постає питання її вирішення. Часто при цьому приймається рішення вимикання або не запуск тестових сценаріїв, оскільки не всі їх результати коректні, що підвищує ризики появи дефектів продукту, підвищує навантаження на команду тестувальників – так як в такому випадку регресійне тестування проводиться мануально, що посилює так званий «людський фактор».В процесі мануального тестування також може використовуватись програмне забезпечення для спрощення деяких дій. Наприклад, відсилання REST-запиту, це можна зробити без стороннього ПЗ через командний рядок консолі або браузер, однак набагато швидше та простіше це можна зробити через Postman – додаток, основна мета якого – спростити таку дію.

Окрім безпосереднього тестування існує група документів, які використовується в процесі тестування, або відображають його результати. До такої групи належать: тест-план (test-plan), чек-список (check-list), тест-сценарій (test-scenario), тест-випадок (test-case), звіт дефекту (bug report).

Зазвичай така документація створюється та заповнюється тестувальниками та описує процес тестування, що саме підлягає перевірці та які кроки для тестування необхідно виконати. Хоча, на разі детальна підтримка всієї тестової документації – надлишкова робота і на практиці вона займає занадто багато часу, і активно підтримуються тільки звіти дефектів.

1.2 Аналіз існуючих рішень

На сьогоднішній день на ринку представлено багато систем, що мають деякий функціонал, необхідний для збору та відображення інформації про виконання тестових сценаріїв. Однак наявні рішення зазвичай призначені для вирішення сторонніх задач, є складними в інтеграції та незручними у використанні через велику кількість надлишкового функціоналу.

1.2.1 Системи безперервної інтеграції

Найближчою за функціональними можливостями групою програмного забезпечення виступають системи безперервної інтеграції (більше відомі як Continuous Integration, або CI). До найпопулярніших систем даного сегменту відносять Teamcity CI, Jenkins, Travis CI, Gitlab CI, Circle CI та інші. Їх призначення та функціональне оснащення практично однакове, а суттєва різниця помітна переважно в користувацькому інтерфейсі та особливостях інтеграції із технологіями існуючого проекту, тому огляд, переваги та недоліки будь-якої із зазначених систем будуть справедливими і для інших. Розглянемо можливості TeamCity CI як рішення поставленої задачі.[3]

Основна ціль використання систем безперервної інтеграції – постійна перевірка компонентів коду на предмет взаємодії між собою, наскільки вона коректна, при кожній перевірці можливий запуск автоматизованих тестових сценаріїв та функціональний контроль якості продукту на кожному етапі його розробки. Тому CI-системи повинні мати можливість зв’язку та взаємодії з системами контролю версій, віддаленими серверами або хмарними сховищами, та інше. Однак, функціонал Teamcity CI також частково задовольняє потребу в зборі інформації відносно виконання автоматизованих тестових сценаріїв – при правильній конфігурації, для кожного збирання проекту запускаються авто-тести, інформація про збірку також в доступі, та можна побачити кількість пройдених та впавших тестів для кожного запуску, визначити скільки разів конкретний тест завершувався помилкою в

перевірках за останні 10 запусків.

Інтерфейс Teamcity CI досить простий та зрозумілий, легко підтримує сторонні плагіни, як, наприклад, Allure, що формує та відображає звіти про виконання тестових сценаріїв. Інтерфейс Teamcity CI зображено на Рисунку 1.2.

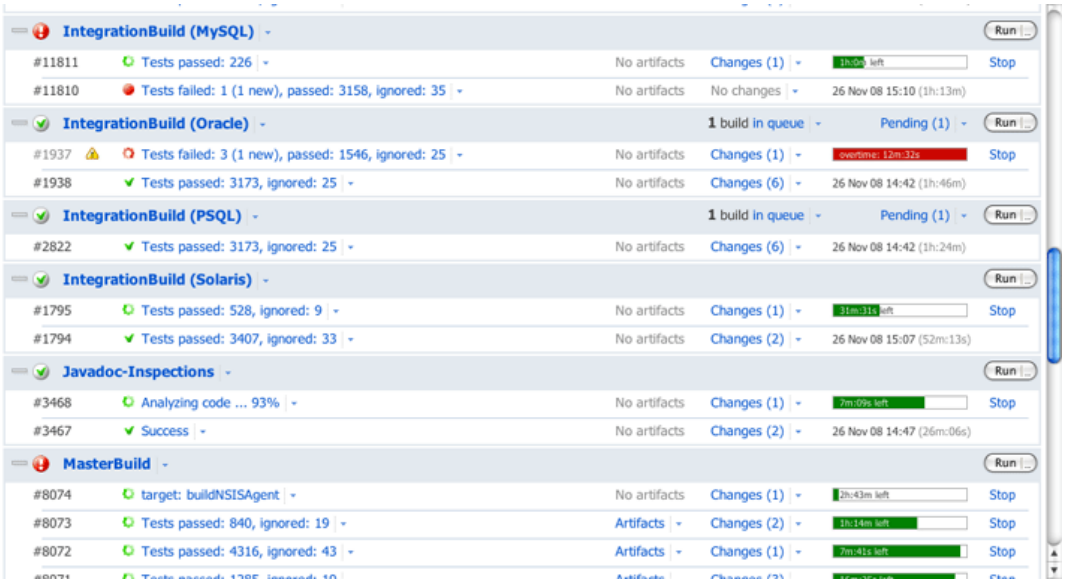

Рисунок 1.2 – Інтерфейс Teamcity CI

Переваги використання TeamCity CI як інструменту збору статистики про виконання тестів:

- можливість використання додаткового плагіну, який підключає Teamcity CI до середовища розробки, що дозволяє переглядати частину зібраної статистики відносно конкретного тесту в процесі розробки;
- доступна документація, широке коло користувачів;
- статистика формується в автоматичному режимі, потребує мінімальних зусиль користувача;
- системи безперервної інтеграції, як правило, використовуються в процесі розробки, їх використання в цілях збору статистики авто-тестів найчастіше звільняє від необхідності використовувати стороннє ПЗ.

Серед недоліків можна виділити:

- використання систем CI сугубо для збору статистики проходження тестів потребують багато часу на налаштування ті інтеграцію системи в бізнес-процеси, що стає дорогим процесом з економічної точки зору;
- сформована статистика не є повноцінною, оскільки представляє данні тільки стосовно проходження тестів тільки на одній конфігурації (яких може бути як завгодно багато), загальної статистики по всім конфігураціям – оточення для виконання тестів, гілка репозиторію, тощо – система не надає;
- можливість роботи тільки в режимі онлайн;
- збір статистики не є основним функціоналом Teamcity CI, зручного інтерфейсу для її перегляду система не надає, відсутня можливість користувацьких налаштувань.

Отже, системи безперервної інтеграції не виступають повноцінним рішенням поставленої задачі, через незручність користувацького інтерфейсу та відсутність можливості користувацьких налаштувань параметрів збору статистики та її відображення.

1.2.2 Інструменти управління тестуванням

В цілях підвищити якість тестування в процесі розробки часто використовують стороннє програмне забезпечення, серед них є група інструментів управління тестуванням. Такі інструменти зазвичай використовують як єдине сховище всієї тестової документації, що дозволяє легко її впорядковувати, візуалізувати, тощо. Такі інструменти часто мають можливість тісної інтеграції із системами CI/CD, трекінговими системами як Jira, що підвищує якість та швидкість синхронізації результатів тестування, автоматизує процес їх збереження.

Один із найбільш популярних інструментів даної групи – TestRail.[4] Основний функціонал, що надає система – інструменти зручного управління тест-кейсами, тест-планами, сценаріями. Описані кроки мануальних сценаріїв співвідносяться із автоматизованими тестовими сценаріями, статус їх виконання також відмічається в системі (є можливість мануально відмітити та автоматично, при відповідних

конфігураціях для сторонніх систем, як CI/CD, системи контролю версій, тощо), надання інформації про процес тестування в режимі реального часу. Інтерфейс системи досить громіздкий, має багато рівнів вкладеності через велику кількість додаткового функціоналу. Інтерфейс TestRail зображено на Рисунку 1.3.

Перевагами системи виділяють:

– популярне рішення, має широку спільноту користувачів та детальну документацію;
– багато засобів візуалізації процесу та результатів тестування;
– широкий вибір користувацьких налаштувань;
– глибока інтеграція із популярними інструментами, що використовуються в процесі розробки;
– запуск виконання тестових сценаріїв в сторонніх системах із користувацького інтерфейсу TestRail;
– відкритий API, що дозволяє клієнтам самостійно конфігурувати власні інтеграції.

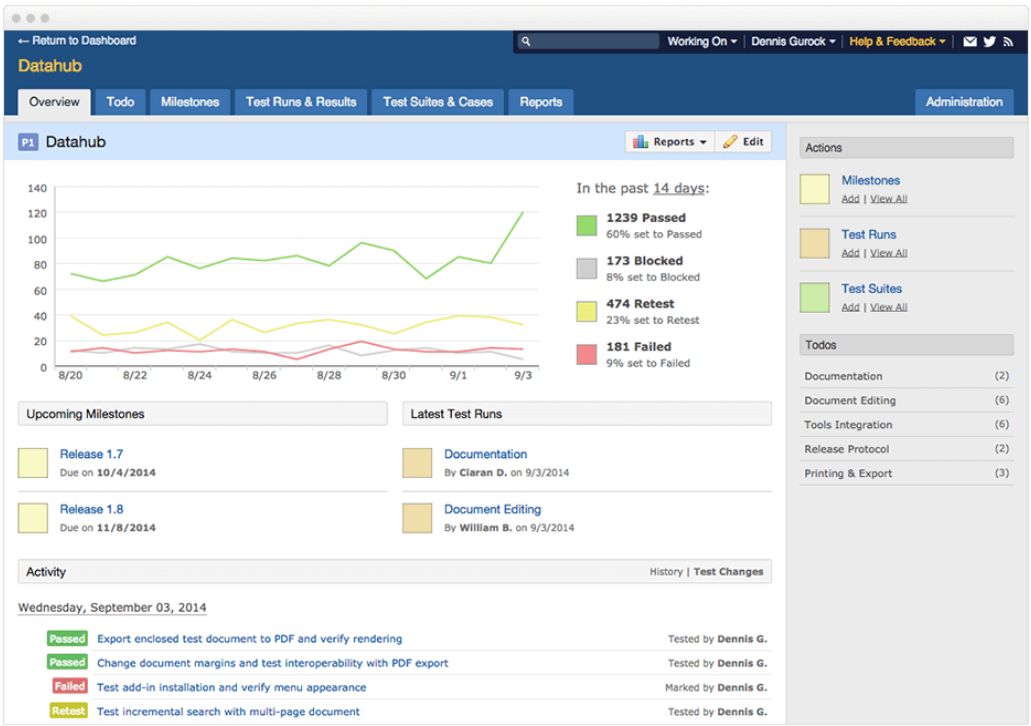

Рисунок 1.3 – Інтерфейс TestRail

При цьому система надає деякий функціонал, що необхідний для рішення поставленої задачі. Одне з її призначень – збір та формування звітності про тестування. Як видно на Рисунку 1.3, система здатна візуалізувати результати тестів для кожного запуску та для загальної статистики.

Однак, надається небагато варіантів відображення звітності, тільки декілька простих метрик. Окрім цього, використання даної системи тільки в цілях збору та відображення статистики виконання тестових сценаріїв потребує дуже багато часу та зусиль для налаштування системи, її інтеграції із проектом з автоматизованими тестовими сценаріями, системою CI/CD. Інтерфейс системи дуже перевантажений стороннім функціоналом, що робить її використання складним та незручним. Також, це платне рішення, а враховуючи, що буде необхідна лише невелика частина його функціоналу, дане рішення виступає економічно невигідним. Тому інструмент TestRail не може бути використаним як повноцінне рішення поставленої задачі.

1.2.3 Статичні аналізатори коду

Функціями збору статистики виконання тестових сценаріїв також володіють статичні аналізатори коду. Основна мета використання подібних додатків – уникнення очевидних помилок в програмному коді, які можна виділити ще на етапі компіляції програмного забезпечення та найшвидше їх вирішення. Такі системи також легко інтегруються із середовищами розробки та CI системами, що робить їх використання легким.

Одна з найпопулярніших таких систем – Sonarqube,[5] Окрім свого безпосереднього функціоналу зберігає статистику виконання модульних тестових скриптів, та надає підказки, які частини коду потребують тестового покриття.

До переваг Sonarqube відносять детальну документацію, код додатку в вільному доступі, його використання безкоштовне. Окрім цього, основний функціонал додатку може допомагати й при написанні проекту автоматизованого тестування. Система також легко інтегрується із популярними оточеннями для розробки та системами безперервної інтеграції.

Основним недоліком виступає незручність у використанні з функціональними тестовими сценаріями, на відміну від модульних тестів. Також користувач не має можливості налаштовувати параметри збору та перегляду статистики.

Висновки до розділу 1

Було розглянуто предметну область – процес розробки програмного забезпечення, методології та основні етапи процесу. Також більш детально розглянуто процес тестування ПЗ, види тестування, як вони пов’язані між собою, тестову документацію.

Було оцінено наявні рішення – наразі на ринку представлено багато рішень, які в тій чи іншій мірі здатні вирішити поставлену задачу. Було розглянуто три групи програмного забезпечення, що виступають наявними аналогами. Аналіз їх переваг та недоліків виявив, що дані системи призначені для вирішення сторонніх задач. Вони володіють деяким функціоналом, що може використовуватись для збору та відображення інформації про виконання автоматизованих тестових сценаріїв, та вони часто використовуються в процесі розробки програмного забезпечення з перших етапів (до етапу тестування та виникнення проблеми стабілізації тестів), тому тестувальник не має потреби їх налаштовувати та конфігурувати.

Однак даний функціонал таких систем не є повноцінним для рішення поставленої задачі, користувацький інтерфейс не розрахований на такі потреби, що робить їх використання складним та незручним.

2 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ

Встановимо ролі користувачів, та які функції система їм надаватиме. Було виділено трьох акторів – Користувач, Адміністратор та Система Автоматизації Складання. Адміністратор конфігурує систему, виконує процес її інтеграції із проектом з автоматизованими тестовими сценаріями. Користувач – актор, що працює із основним функціоналом системи, запитує статистику про виконання тестових сценаріїв. Проект взаємодіє із системою тільки запускаючи процес збереження актуальних даних із останнього звіту.

2.1 Use case діаграма для Користувача

Основний функціонал, яким користується актор Користувач – запит різноманітної статистики стосовно виконання тестових сценаріїв, даний сценарій зображено на Рисунку 2.1. Система може потребувати різні аргументи та в результаті групує статистику за відповідними правилами.

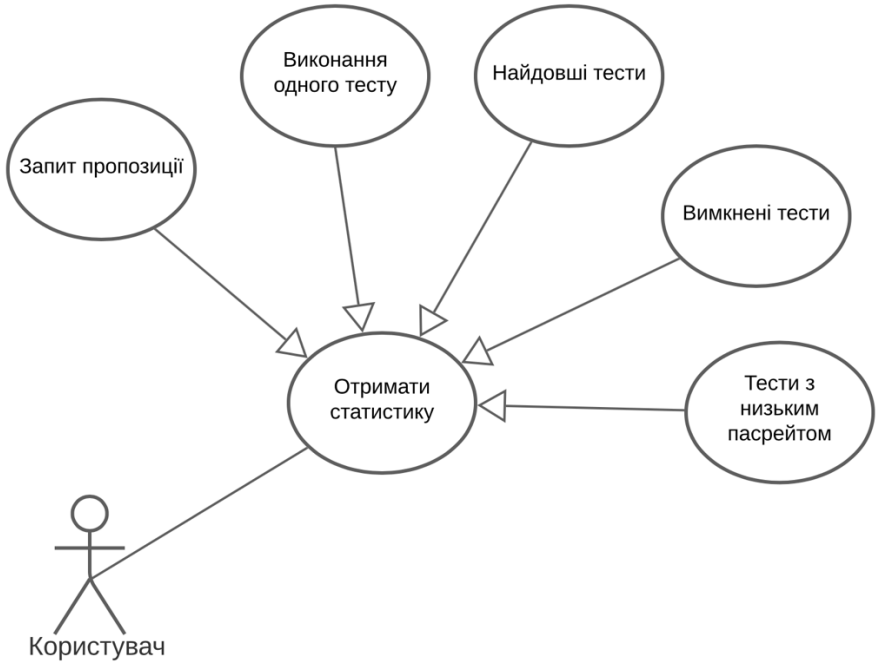

Рисунок 2.1 – Варіанти використання для користувача

Більш детально прецедент «Отримати статистику» описаний в таблицях 2.1 – 2.7. Користувач може запитати статистику виконання тестового сценарію за вказаним ім’ям за заданим користувачем, або заданим за замовчуванням час. Результатом виступатиме масив повідомлень формату JSON про вказаний тест із додатковими атрибутами, окрім статусу.

Таблиця 2.1 – Прецедент «Виконання одного тесту»

Назва	Виконання одного тесту
Актор	Користувач
Опис	Надіслати запит про статистику виконання одного тестового сценарію за вказаний період
Передумови	Наявність записів із вказаним ім’ям тесту
Результат	Користувачу надані результати пошуку в форматі JSON

Користувач може запитати статистику виконання тестових сценаріїв (бажану кількість вказує користувач), які виконувались найдовше, за заданий користувачем або за замовчуванням час. Результатом виступатиме масив повідомлень формату JSON про знайдені тестові сценарії.

Таблиця 2.2 – Прецедент «Найдовші тести»

Назва	Найдовші тести
Актор	Користувач
Опис	Надіслати запит про статистику тестів, що виконувались найдовше за вказаний період
Передумови	Наявність записів про результати тестів
Результат	Користувачу надані результати пошуку в форматі JSON

Користувач може запитати статистику про вимкнені тестових сценаріїї (бажану кількість вказуе користувач), які вимкнені (мають статус SKIPED або IGNORED) за заданий користувачем або за замовчуванням час. Результатом виступатиме масив повідомлень формату JSON про знайдені тестові сценарії.

Таблиця 2.3 – Прецедент «Вимкнені тести»

Назва	Вимкнені тести
Актор	Користувач
Опис	Надіслати запит про статистику вимкнених тестів за вказаний період
Передумови	Наявність записів про результати тестів
Результат	Користувачу надані результати пошуку в форматі JSON

Користувач може запитати статистику про тестові сценарії (бажану кількість вказує користувач), які мають відсоток успішного виконання нижче заданого користувачем, за заданий користувачем або за замовчуванням час. Результатом виступатиме масив повідомлень формату JSON про знайдені тестові сценарії.

Таблиця 2.4 – Прецедент «Тести із низьким пасрейтом»

Назва	Тести із низьким пасрейтом
Актор	Користувач
Опис	Надіслати запит про статистику тестів, які мали процент проходження нижче за вказаний за вказаний період
Передумови	Наявність записів про результати тестів
Результат	Користувачу надані результати пошуку в форматі JSON

Користувач може запитати пропозицію системи про тестові сценарії щодо їх поліпшення. Результатом виступатиме масив повідомлень формату JSON про тестові

сценарії які необхідно стабілізувати, або повідомлення, що результати якими оперує система – задовільні

Таблиця 2.5 – Прецедент «Запит пропозиції»

Назва	Запит пропозиції
Актор	Користувач
Опис	Надіслати запит на оцінку системою результатів проходження тестів за вказаний період часу
Передумови	Наявність записів про результати тестів
Результат	Користувачу надана пропозиція про покращення тестових сценаріїв, або повідомлення про задовільні результати

2.2 Use case діаграма для Адміністратора

Основний функціонал, яким користується актор Адміністратор – конфігурування параметрів системи в процесі її інтеграції із існуючим проектом автоматизованого тестування. Він взаємодіє із чотирма прецедентами.

Більш детально прецедент «Інтеграція системи» описаний в таблицях 2.2.1 – 2.1.4. Адміністратор може явно вказати шлях до звіту про виконання тестових скриптів. Результатом буде оновлення змінної всередині системи, яка відповідає за шлях до файлу та виведення відповідного повідомлення про статус запиту.

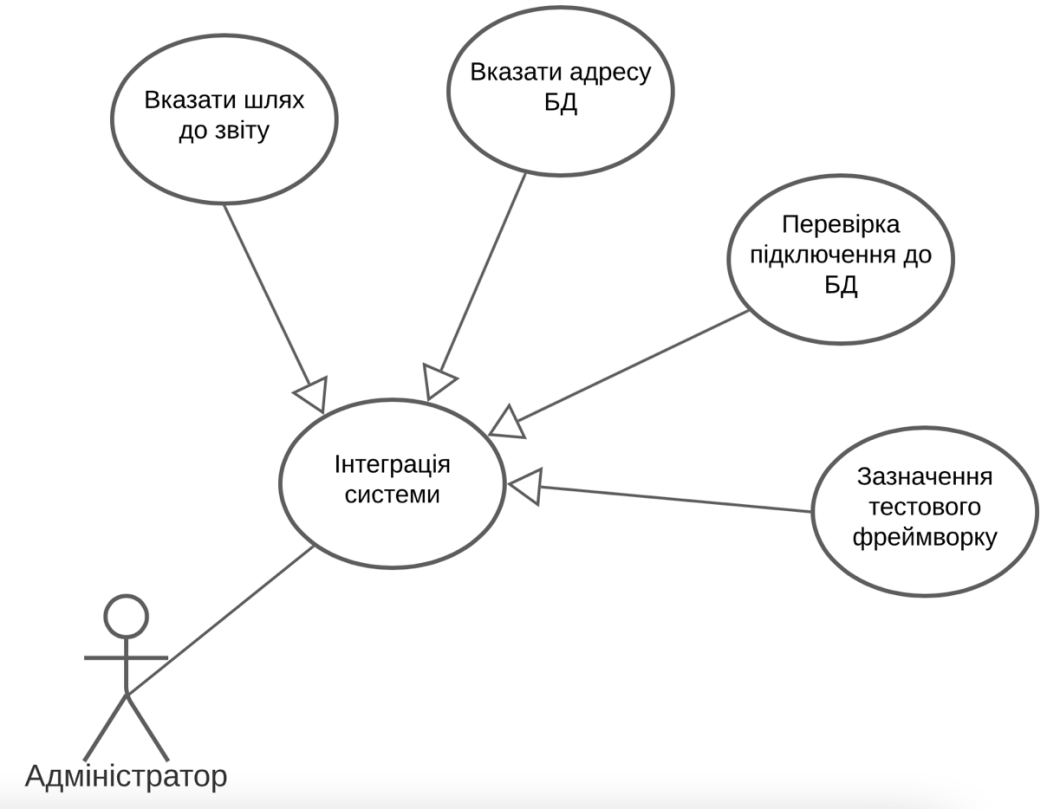

Рисунок 2.2 – Варіанти використання для Адміністратора

Таблиця 2.6 – Прецедент «Вказати шлях до звіту»

Назва	Вказати шлях до звіту
Актор	Адміністратор
Опис	Надіслати запит про вказання нового шляху до автоматично генерованого звіту
Передумови	Наявність файлу за заданим шляхом
Результат	Адміністратору надане повідомлення про статус виконання запиту – позитивний чи негативний

Адміністратор може явно вказати адресу підключення до БД, де система буде зберігати звіти про виконання тестових скриптів. Результатом буде оновлення змінної всередині системи, яка відповідає за адресу БД та виведення відповідного повідомлення про статус запиту.

Таблиця 2.7– Прецедент «Вказати адресу БД»

Назва	Вказати адресу БД
Актор	Адміністратор
Опис	Надіслати запит про вказання нової адреси БД – хост та порт
Передумови	Запущена БД MongoDB за вказаною адресою
Результат	Адміністратору надане повідомлення про статус виконання запиту – позитивний чи негативний

Адміністратор може явно вказати тестовий фреймворк, який використовується проектом із автоматизованими тестами. Результатом буде оновлення змінної всередині системи, яка відповідає за фреймворк, що використовується та виведення відповідного повідомлення про статус запиту.

Таблиця 2.8 – Прецедент «Зазначення тестового фреймоворку»

Назва	Зазначення тестового фреймоворку
Актор	Адміністратор
Опис	Надіслати запит про вказання імені тестового фреймворку
Передумови	Фрейморк підтримується системою
Результат	Адміністратору надане повідомлення про статус виконання запиту – позитивний чи негативний

Адміністратор може перевірити підключення до бази даних. Результатом буде спроба підключення до БД, створення та зчитування об’єктів в ній, та виведення відповідного повідомлення про статус запиту.

Таблиця 2.9 – Прецедент «Перевірка підключення до БД»

Назва	Перевірка підключення до БД
Актор	Адміністратор
Опис	Надіслати запит про перевірку підключення до БД, перевірити можливість операцій запису та зчитування
Передумови	Запущена БД MongoDB за вказаною адресою
Результат	Адміністратору надане повідомлення про статус виконання запиту – позитивний чи негативний

2.3 Use case діаграма для актора Система автоматизації складання

Вся взаємодія проекту з системою – тригер події захоплення нового звіту про виконання тестових сценаріїв.

Рисунок 2.3 – Варіанти використання для Системи збірки

Опис прецеденту для Система Автоматизації Складання зазначено в таблиці 2.10. Після завершення виконання тестових сценаріїв фреймворки для автоматизованого тестування автоматично генерують файли звітів про результати тестів. Після цього система збірки використовуючи команду «grep» взаємодіє із системою, анонсуючи новий актуальний звіт, який система має опрацювати.

Таблиця 2.10 – Прецедент «Анонсувати новий звіт»

Назва	Анонсувати новий звіт
Актор	Система Автоматизації Складання
Опис	Тригер події системи, яка має завантажити актуальний звіт після завершення виконання тестових сценаріїв
Передумови	Проект інтегрований із системою
Результат	Актуальний звіт був зчитаний та дані записані в БД

Висновки до розділу 2

В даному розділі було детально розглянуто всі сценарії використання системи. Для цього було побудовано діаграму прецедентів. Було виділено трьох акторів, які взаємодіють із системою – Користувач, Адміністратор та Система Автоматизації Складання.

Для Адміністратора присутні функції налаштування системи, інтеграція системи із існуючим проектом, перевірка статусу підключення. Для Користувача присутні функції запиту різної статистики та інформації про проходження тестових сценаріїв та пропозицій, згенерованих системою. Системі Автоматизації Складання доступна функція запуску події зчитування системою актуального звіту після завершення виконання тестових сценаріїв. Всі описані прецеденти виступають функціональними вимогами до системи.

3 ВИЗНАЧЕННЯ ДЖЕРЕЛА ІНФОРМАЦІЇ ТА СТРУКТУРИ ЗВІТУ

3.1 Визначення метрик виведення інформації

Для того щоб система була найбільш гнучкою, була корисною в процесі розслідування різноманітних недоліків створених автоматизованих тестових сценаріїв, пов’язаних із вимогою стабільності, необхідно виділити основну інформацію, яка необхідна тестувальнику та найбільш зручний спосіб її представлення. Для визначення бажаного набору даних розглянемо найчастіші передумови для нього. Основними недоліками тестів, що свідчать про необхідність стабілізації виступають хибно-позитивні та хибно-негативні результати виконання.

Хибно-позитивні результати означають, що тестовий скрипт нічого не тестує, завжди повертаючи позитивний результат, проблема може бути в некоректній підготовці даних, некоректно визначеному сценарію для тестування конкретного функціоналу продукту, тощо.

Хибно-негативні результати також можуть виникати із тих самих причин, окрім цього можливі й проблеми із нестабільним оточенням для виконання тестів, нестабільною швидкістю Інтернету, тощо. Такі результати виступають менш критичною проблемою, ніж хибно-позитивні, оскільки тестувальник має з’ясувати причину падіння тестів та пройти мануально сценарій написаного скрипта, що мінімізує вірогідність не помітити дефект продукту, на відміну від хибно-позитивних результатів тесту. Однак, мануальне тестування некоректних скриптів також потребує витрат часу та зусиль, яких можна уникнути при стабільних автоматизованих тестах. Окрім цього, одна з основних проблем у процесі стабілізації тестів – визначення та відтворення дефекту, що призводить до нестабільності сценарію. Часто, особливо у випадку хибно-негативних результатів, результати неоднозначні, наприклад, 9 запусків тест повернув позитивний результат і 1 запуск – негативний.

Тому, найбільш корисними даними в процесі стабілізації виступають дані про деяку кількість останніх запусків виконання всіх тестів, або конкретного тесту, що

може допомогти виділити нестабільні скрипти та представить додаткову інформацію, що допоможе в розслідуванні дефекту. Такі дані можуть надаватись системою у формі звіту, який формується за відповідними правилами, в залежності від обраної команди. Враховуючи, дефекти, що свідчать про нестабільність тестів, основними виступатимуть звіти за наступними метриками:

- тести, кількість яких задається (за замовчуванням – 10), які найчастіше мали негативний статус виконання за заданий час (за замовчуванням – місяць);
- тести, кількість яких задається, які найчастіше мали позитивний статус виконання за заданий час;
- результати виконання обраного тесту за заданий час;
- результати виконання обраної групи тестів за заданий час;
- тести, кількість яких задається, які мають найдовшу тривалість виконання за заданий час;
- тести, кількість яких задається, які мають найкоротшу тривалість виконання за заданий час;

Звіт має включати в себе інформацію про кожен тестовий сценарій, який відповідає обраній метриці. До необхідних атрибутів тестового сценарію в звіті можна віднести:

- назва автоматизованого тестового скрипта;
- дата та час виконання;
- тривалість виконання;
- статус проходження;
- помилка (в разі завершення з помилкою);

Також можна виділити додаткові атрибути, такі як ідентифікатор оточення, де виконувався тестовий скрипт, група, до якої відноситься тест (популярні тестові фреймворки надають можливість об’єднувати тести в групи та набори (Suite) та запускати тести по одному, весь тестовий клас, всю групу, або набір), тощо. Такі атрибути є більш специфічними ніж наведені, однак також використовуються, тож можуть бути імплементованими в подальшій підтримці системи.

3.2 Визначення джерела інформації

Для того щоб система володіла актуальними та повними даними про виконання тестових скриптів необхідно визначити джерело даних, та вирішити сумісну із цим проблему – спосіб зчитування даних. Можливі декілька варіантів вирішення даного питання:

- користувач самостійно вносить в систему дані про виконання тестових сценаріїв;
- система взаємодіє із популярним програмним забезпеченням, що використовується для формування звітів про виконання тестів (як, наприклад, Allure або Report Portal);
- система взаємодіє із фремворком, який використовується для написання тестів (наприклад, JUnit або TestNG) або із системою автоматизації складання проекту (як, наприклад, Maven або Gradle).

Перший варіант виступає найменш зручним із користувацької точки зору, оскільки він потребує регулярної кропіткої взаємодії із системою. Враховуючи те, що задля коректних результатів роботи системи вона має оперувати консистентними даними, це накладає на користувача великий набір правил, яким введені дані повинні відповідати та підвищує складність реалізації системи (оскільки система також має посилено валідувати введену інформацію для уникнення помилок пов’язаних із людським фактором).

Здобувати дані за допомогою стороннього програмного забезпечення набагато більш зручний спосіб з точки зору користувацького інтерфейсу. Він гарантує, що дані зі всіх запусків тестових скриптів будуть зібрані без змін, таким самим чином, як вони збираються для користувацького звіту. Однак значним недоліком такого підходу виступає наявність проміжного програмного забезпечення. Воно також може мати дефекти, які можуть виявитись критичними для коректної роботи системи. Популярне репортингове ПЗ зараз також підтримується, додатки продовжують нарощувати функціонал, а існуючий – не має гарантій незмінності. Тому, використовуючи його як основне джерело даних, система має завжди підтримувати

ряд найпопулярніших версій обраного ПЗ, та швидко адаптуватись до змін в його функціональних можливостях.

Врешті решт, використовувати популярні тестові фреймворки та системи автоматизації складання проекту (вони часто тісно інтегровані між собою) для збору даних виступає найбільш гнучким з точки зору розробки системи та легким із користувацької точки зору способом. До його переваг можна віднести:

- не потребує додаткових дій користувача для збору даних;
- мінімізація людського фактору впливу на дані;
- популярних тестових фреймворків набагато менше ніж репортингового ПЗ, що зробить систему підходящою для більшої кількості користувачів;
- інформація отримується безпосередньо із ПЗ, де вона була зібрана, вона представлена в найбільш повному вигляді із усіх можливих джерел;
- популярні тестові фреймворки мають безкоштовну ліцензію;
- велика кількість документів, що генеруються автоматично в заданій за замовчуванням директорії.

Однак, недолік цього способу – як і в попередньому випадку, проблема підтримки системи для роботи із новими версіями тестових фреймворків, але вона менш значна, оскільки тестові фреймворки мають великий діапазон в часі між оновленнями (наприклад, JUnit 4.12 був представлений в грудні 2014 року, а наступна версія JUnit 4.13 – в січні 2020).

В ході виконання автоматизованих тестових сценаріїв, як правило, тестові фреймворки з системами автоматизації складання проектів генерують ряд мета-документів із додатковою детальною інформацією стосовно виконання тестів. Для конкретності, розглянемо документи, які були автоматично згенеровані під час запуску проекту з автоматизованими тестовими скриптами використовуючи технології JUnit та Maven. Вміст директорії із такими файлами зображено на Рисунку 3.1.

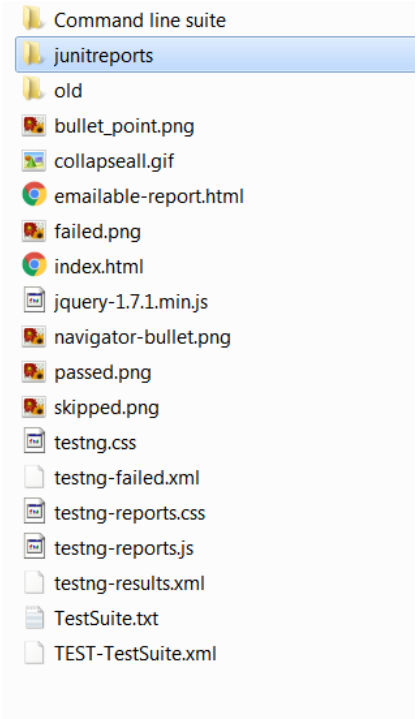

Рисунок 3.1 – Автоматично згенеровані документи

Деякі зі згенерованих документів мають однаковий вміст, але різний формат файлу, що надає можливість обрати найзручніший документ як джерело даних. Оскільки документ буде підлягати парсингу системою, найзручніший формат із представлених – XML. Вміст звіту TestNG в форматі XML зображено на Рисунку 3.2., відкрито файл testing-results.xml.

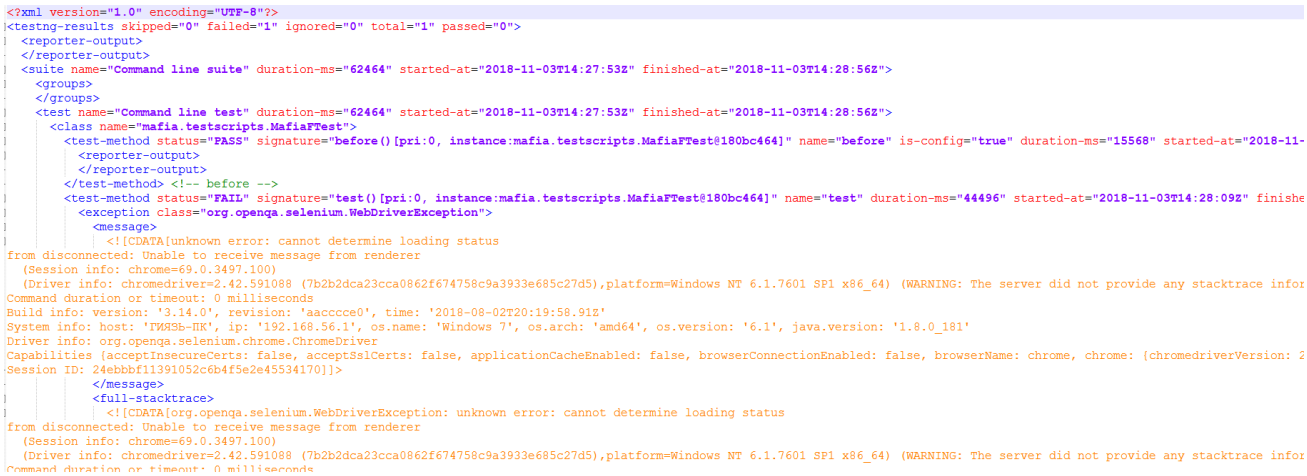

Рисунок 3.2 – Вміст звіту TestNG в форматі XML

Як видно, всі необхідні та опціональні дані для системи присутні у звіті, тому його доцільно й легко можна використовувати як джерело даних. На Рисунку 3.3 представлено приклад звіту JUnit в форматі XML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<testsuites disabled="" errors="" failures="" name="" tests="" time="">
    <testsuite disabled="" errors="" failures="" hostname="" id=""
               name="" package="" skipped="" tests="" time="" timestamp="">
        <properties>
            <property name="" value=""/>
        </properties>
        <testcase assertions="" classname="" name="" status="" time="">
            <skipped/>
            <error message="" type=""/>
            <failure message="" type=""/>
            <system-out/>
            <system-err/>
        </testcase>
        <system-out/>
        <system-err/>
    </testsuite>
</testsuites>
```

Рисунок 3.3 – Приклад звіту JUnit в форматі XML

Всі необхідні поля для системи також в наявності, як в звіті згенерованому TestNG. Тому система може спиратись на файли звітів, що генеруються автоматично тестовими фреймворками.

Висновки до розділу 3

Для забезпечення гнучкості та зручності у використанні системи було визначено набір даних який має надавати система, що буде найбільш необхідним для тестувальника в процесі стабілізації автоматизованих тестових сценаріїв. Для цього було розглянуто основні ознаки дефекти тестів, що приводять до їх нестабільності – хибно-позитивні та хибно-негативні результати, причини їх появи.

В результаті, було виділено важливу інформацію про тестові сценарії, що має оформлюватись в звіт окремо для кожного сценарію. Також було встановлено метрики виведення інформації, які будуть інформативними для багатьох проблем

пов’язаних зі стабілізацією тестів, які має підтримувати система.

Окрім цього, було визначено й додаткові поля звіту, які можуть бути корисними при вирішенні специфічних проблем тестових сценаріїв, що дає простір для розширення системи та задоволення потреб більшої кількості користувачів.

Було розглянуто різні способи отримання системою інформації про виконання тестових сценаріїв. Розглянувши їх переваги та недоліки, було визначено, що найбільш стабільний та достовірний спосіб – взаємодія із фреймворком для написання тестів або системою автоматизації складання проекту. Основні переваги на його користь перед іншими варіантами – можливість збирати дані в автоматичному режимі (як зручність із користувацької точки зору, так і мінімізація людського фактору), повнота інформації та універсальність. Також присутній несуттєвий недолік – необхідність підтримувати оновлення тестового фреймворку.

Для вибору джерела даних було розглянуто документи які генеруються автоматично тестовим фреймоврком. Не всі із них мали повну інформацію про тестові сценарії, що необхідна системі, деякі мали еквівалентні дані, однак у різних форматах файлу. В результаті, було обрано файл, згенерований тестовим фреймворком, формату XML.

4 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ

Для побудови гнучкої архітектури системи необхідно обрати підходящі технології, які володіють достатнім функціоналом для досягнення мети.

4.1 Платформа

Платформою розробки була обрана об’єктно-орієнтовна мова програмування Java. Оскільки передбачається, що система буде встановлюватись та використовуватись, в основному, локально вона має мати можливість інсталяції на різні операційні системи без втрати функціоналу та зручності користувацького інтерфейсу. Класичним рішенням виступає використання мови Java – програмний Java-код компілюється в байт-код, який може виконуватись на іншій платформі, де встановлена віртуальна машина JVM.

Для розробки було обрано OpenJDK одинадцятої версії. JDK – Java Development Kit, стандартний набір пакетів із утилітами за замовчуванням, документацією та середою виконання JRE – Java Runtime Environment. Безкоштовна версія призначена для некомерційної розробки, та надає достатній функціонал для використання в процесі розробки системи. Історія Java почалась з офіційного випуску ще в 1996 року, з того часу й до сьогодні мова активно підтримується та вдосконалюється, має масштабну спільноту користувачів.

До особливостей Java можна віднести принцип, яким керувались розробники при її створенні – Write once Run anywhere, тобто, платформо-незалежність. Після запуску програмного коду, він компілюється в байт-код, який виконує JVM. Оскільки JVM входить в ряд сервісів, які надає JRE, а також виконання додатку не залежить від платформи (де має бути встановлена JVM), то Java-код може виконуватись в будь-якому середовищі де встановлена JRE.

Окрім цього, Java – об’єктно-орієнтовна мова програмування, хоча, починаючи із 8 версії, в мові з’явились конструкти функціонального програмування, основна сутність, якою оперує програмний код – об’єкт. Це також дозволяє слідувати

принципам ООП та SOLID в процесі проектування архітектури системи.[6]

SOLID – п’ять принципів проектування, запропоновані Робертом Мартіном, призначені для створення гнучкої та легкої у підтримці та масштабуванні архітектури системи. Single responsibility – принцип єдиної відповідальності, пропонує кожному об’єкту надавати тільки одне зобов’язання, яке має бути повністю приховане в класі. Open-closed – принцип передбачає, що члени класу відкриті до розширення функціоналу, однак їх неможна змінювати. Liskov substitution – принцип підстановки, використовуючи базовий тип має бути можливість використовувати його підтип без змін коду. Interface segregation – розбиття інтерфейсів таким чином, щоб вони не залежали від сторонніх функцій інших інтерфейсів. Dependency inversion – високорівневі модулі не мають залежати від низькорівневих, вони взаємодіють через прошарок абстракції.

4.2 Система автоматизації складання проекту

В процесі розробки та підтримки проекту перед розробником постає ряд питань такі як підтримка великої кількості залежностей від сторонніх утиліт, менеджмент всіх необхідних кроків в процесі збирання проекту, оптимізація коду, тощо. Із таким рангом питань стає у нагоді підключення системи автоматизації складання проекту, який бере таку роботу на себе.

Класичним вибором системи автоматизації складання проекту для Java проекту виступають системи автоматизації складання Maven та Gradle. В основному функціоналі вони дуже подібні, однак Gradle виграє набагато більш лаконічним синтаксисом та в більш вузькоспеціалізованих можливостях, тому було обрано саме його.[7]

До основних переваг Gradle можна віднести наступні:

– скрипт збірки описується мовою Groovy, синтаксис мови легкий у читанні та не потребує глибокого розуміння мови, лаконічний, в файлі легко орієнтуватись (на відміну від скриптів Maven, які в форматі XML, та набагато масивніші);

– швидке виконання команд (в порівнянні із Maven, команда clean build відпрацьовує за 0.685с, в противагу 25.852с);

– можливість встановити залежність від конкретного файлу або директорії іншого модулю, а не всього зібраного модулю формату JAR ;

– можливість версіонування проекту, окремих модулів;

– можливість генерації стандартного проекту, із стандартними файлами залежностей та структурою директорій.

На Рисунку 4.1 зображено синтаксис файлу build.gradle. Для Рисунку було взято файл одного із модулів системи. Хоча Gradle окремий інструмент, його реалізація не пов’язана із Maven, він оперує, в основному, бібліотеками, що знаходяться в Maven-репозиторії, що відображається в блоці repositories.

```
plugins {
    id 'java'
}

version '1.0-SNAPSHOT'

sourceCompatibility = 1.11

repositories {
    mavenCentral()
}

dependencies {
    implementation project(":mongo-service")
    implementation project(":parser")
    compile group: 'org.testng', name: 'testng', version: '7.0.0'
    compile 'org.projectlombok:lombok:1.18.14'
    annotationProcessor 'org.projectlombok:lombok:1.18.14'
}
```

Рисунок 4.1 – Синтаксис файлу build.gradle

В блоці dependencies вказуються залежності, які необхідні для роботи модулю, або проекту. Ключові слова, як «compile» або «runtime» перед адресою вихідного коду бібліотеки вказують на фазу збірки проекту, де йому знадобляться ті чи інші

залежності. Коректне використання ключових слів можуть оптимізувати збірку проекту за часом та ресурсами.

4.3 Тестовий фреймворк

В процесі розробки додатку, окрім написання коду програми необхідно й відлагоджувати його, забезпечувати стабільність існуючого функціоналу. Для даної мети можна використовувати автоматизовані модульні та інтеграційні тестові сценарії. Для цього одні із найвідоміших фреймворків – TestNG та JUnit.[8]

Загалом, за основним функціоналом та механізмом використання дані фреймворки дуже подібні, однак вони мають відмінності в деяких додаткових функціях. Наприклад, TestNG має набагато більше анотацій, які використовуються в процесі тестування, вони не мають обмеження, що методи мають бути тільки статичними членами класу, в порівнянні із JUnit.

Окрім цього, TestNG надає можливість створювати XML документи для групування тестів в різні категорії, наприклад, за функціоналом, або типом (модульні, інтеграційні, навантаження, тощо). Це дуже влучна функція, вона дозволяє уникнути перевантаження кодової бази тестових класів надлишковою інформацією, залишаючи в них тільки бізнес-логіку.

Також TestNG відразу надає можливість багатопокового запуску тестових сценаріїв, для цього просто необхідно додати параметри в XML файл. JUnit таким функціоналом не володіє, для цього необхідно підключати сторонні утиліти та окремо їх налаштовувати. Вони також конфліктують із деякими версіями фреймворку, тому процес налаштування може бути довгим та складним.

Враховуючи дані можливості фреймворків було обрано TestNG. Для поточних цілей проекту його функціоналу точно вистачить для тестування в процесі розробки, та на додачу він має потенціал для написання функціональних автоматизованих тестів в майбутньому при розширенні та підтримці системи.

4.4 Додаткові утиліти

Так як система буде отримувати інформацію із файлів з форматами XML та JSON необхідно встановити зручний та повноцінний спосіб роботи із ними. Java надає власні утиліти для роботи із такими документами, однак вони мають досить обмежений функціонал та громіздкий синтаксис, що робить програмний код складний у читанні та підтримці.

Для даних цілей було обрано одну із найпопулярніших бібліотек напрямку – Jackson. Для роботи із великими та середніми за розмірами файлами вона проявила себе найшвидшою серед аналогів – JSON.simple, GSON та JSONP.[9] Функціонал бібліотеки охоплює багато форматів документів, що може стати в нагоді про розширені системи. Для кожного формату синтаксис практично однаковий, що робить процес розробки швидким, а імплементовані сервіси з допомогою Jackson консистентними між собою.

Окрім цього, також було обрано додаткові утиліти – Lombok та Sl4j. Призначення бібліотеки Lombok полягає в спрощені описання багатьох базових конструкцій мови програмування Java, таких як методи-аксесори, стандартні конструктори класу, методи equals та hashCode, тощо. Оскільки часто такі конструкції необхідні в стандартному вигляді, Lombok дозволяє їх додавати анотаціями для членів класу, або всього класу (наприклад, для геттерів, це буде означати що для кожного поля класу є публічний метод get). Це суттєво зменшує захаращеність програмного коду, пришвидшує навігування та процес розробки.

Бібліотека Sl4j призначена для універсального виведення логів роботи системи. Вона має багато налаштувань та можливостей та єдиний лаконічний інтерфейс, що робить її легкою та непомітною у використанні.

4.5 База даних

В якості бази даних, де будуть зберігатись результати проходження тестових сценаріїв, було обрано документо-орієнтовану систему управління базами даних

MongoDB.[10] Основні характеристики MongoDB, за якими було здійснено вибір даної СУБД:

– гнучкий JSON формат документів, що особливо зручно враховуючи, що в системі вже буде мапер JSON документів і можливе повторне використання коду;

– не потрібні навички володіння мовою SQL, операції з БД легко проводяться за допомогою програмного коду із проекту;

– СУБД розрахована та оптимізована в основному для простих запитів, що відповідає вимогам системи, не прогнозуються складні операції із базою та колекціями;

– система легка в масштабуванні, що необхідно при розширені додатку для підтримки нових фреймовркв, тощо;

– відкрита безкоштовна версія, масштабна спільнота користувачів, багато навчальних матеріалів.

Окрім цього, розробниками MongoDB також було створено утиліту mongo-java-driver, що можна використовувати з мовою Java Вона має легкий для розуміння та лаконічний синтаксис, дозволяє всі необхідні операції виконувати програмним кодом, виконуючи при цьому роль абстракції.

4.6 Середовище розробки

Середовище розробки – програмне забезпечення, що містить в собі налаштований текстовий редактор, засоби запуску та відлагодження програмного коду та додатковий функціонал. Для розробки програмного забезпечення середовище розробки не є необхідним, програмний код можливо написати в звичайному текстовому редакторі, компілювати його та запускати через командний рядок. Даний підхід потребує багато часу та уваги розробника, а призначення середовища розробки – спростити процес написання коду наскільки це можливо автоматизованими засобами. До переваг що надають такі середовища відносять :

– автоматичне доповнення коду, створення стандартних конструкцій;

– запуск компілятора по мірі написання коду, керування компіляцією, збірками,

діями із репозиторієм через користувацький інтерфейс;

- глибока інтеграція із системами збірки, автоматичне оновлення зовнішніх залежностей;
- можливість додаткового підключення плагінів для взаємодії зі стороннім програмними забезпеченням.

Тому було обрано використовувати IntelliJ IDEA. Існує її близький конкурент – Eclipse IDE, однак він має менш інтуїтивно-зрозумілий користувацький інтерфейс.

Висновки до розділу 4

В даному розділі було детально описано вибір технологій, що необхідні для розробки системи. Для цього було розглянуто найпопулярніші технології, які використовуються для рішення подібних задач, та серед них було обрано ті, що найкраще підходять вимогам системи.

В якості платформи розробки було обрано мову програмування Java завдяки її об’єктно-орієнтованості, платформонезалежності та ряду інших переваг. Для складання проекту, менеджменту залежностей та оптимізації кінцевого результату було обрано Gradle. Для проведення модульного та інтеграційного тестування системи в процесі її розробки та підтримки було обрано фреймворк TestNG. Також було обрано додаткові утиліти, що будуть пришвидшувати та спрощувати процес розробки. В якості СУБД було обрано MongoDB завдяки її простоті у використанні та легкості інтеграції із системою.

Всі обрані технології є безкоштовними та з програмним кодом у вільному доступі. Широка спільнота користувачів для кожної із них спростить розслідування причин появи проблем в процесі імплементації, у разі їх появи.

5 АЛГОРИТМ ВЗАЄМОДІЇ ІЗ КОРИСТУВАЧЕМ

Розглянемо один із основних сценаріїв роботи з системою – коли користувач запитує деяку статистику стосовно результатів виконання тестових сценаріїв. Для більш детального огляду побудуємо схему алгоритму роботи системи при введені такого запиту та діаграму послідовності для сценарію, коли система інтегрована та підключена до проекту і користувач запускає тести і після їх виконання запитує деяку статистику.

5.1 Алгоритм обробки запиту статистики

Розглянемо алгоритм роботи системи після того як система була встановлена, сконфігурована, та підключена до проекту. Користувач вже запускав деякі тестові сценарії та їх результати вже були збережені в базі даних системи. Алгоритм обробки запиту статистики зображено на Рисунку 5.1.

Перш за все система очікує введення користувачем команди, що вказує на те, які саме дані йому необхідні. Як було зазначено в розділі 3, це може бути запит про статистику виконання конкретного тесту за певний час, або запит результатів деякої групи тестів, тощо.

Будь яка введена команда із аргументами підлягає валідації і тільки в разі коректно введеної команди та набору аргументів (враховуючи, що деякі із них – опціональні, як, наприклад, часовий діапазон за який необхідна статистика, стандартне значення, задане системою – 30 днів) система продовжує обробку команди. В іншому випадку – на консоль виводиться повідомлення про помилку.

Деякі додаткові перевірки також відбуваються ще на етапі валідації – наприклад, при запиті статистики щодо одного тесту – відразу відбувається перевірка на наявність відповідних записів в колекціях БД. Оскільки назва тесту може мати достатньо багато символів, в процесі вводу команди легко помилитись, такі перевірки можуть суттєво підвищити якість користувацького досвіду, пришвидшуючи швидкодію системи.

Після валідації відбувається перевірка опціональних параметрів. У випадку коли вони не задані – встановлюються задані за замовчуванням значення. Наразі – це значення періоду, за який необхідно зібрати статистику, а значення за замовчуванням для нього – 30 днів.

Далі визначається які саме дані система повинна повернути, формується та виконується відповідний запит до БД. Отримані дані також підлягають обробці – їх частина не необхідна в презентації користувачу (технічна мета-інформація, як id сутності тест-звіту), вони також форматуються в зручний для читання вигляд – текст формату JSON та виводиться на консоль.

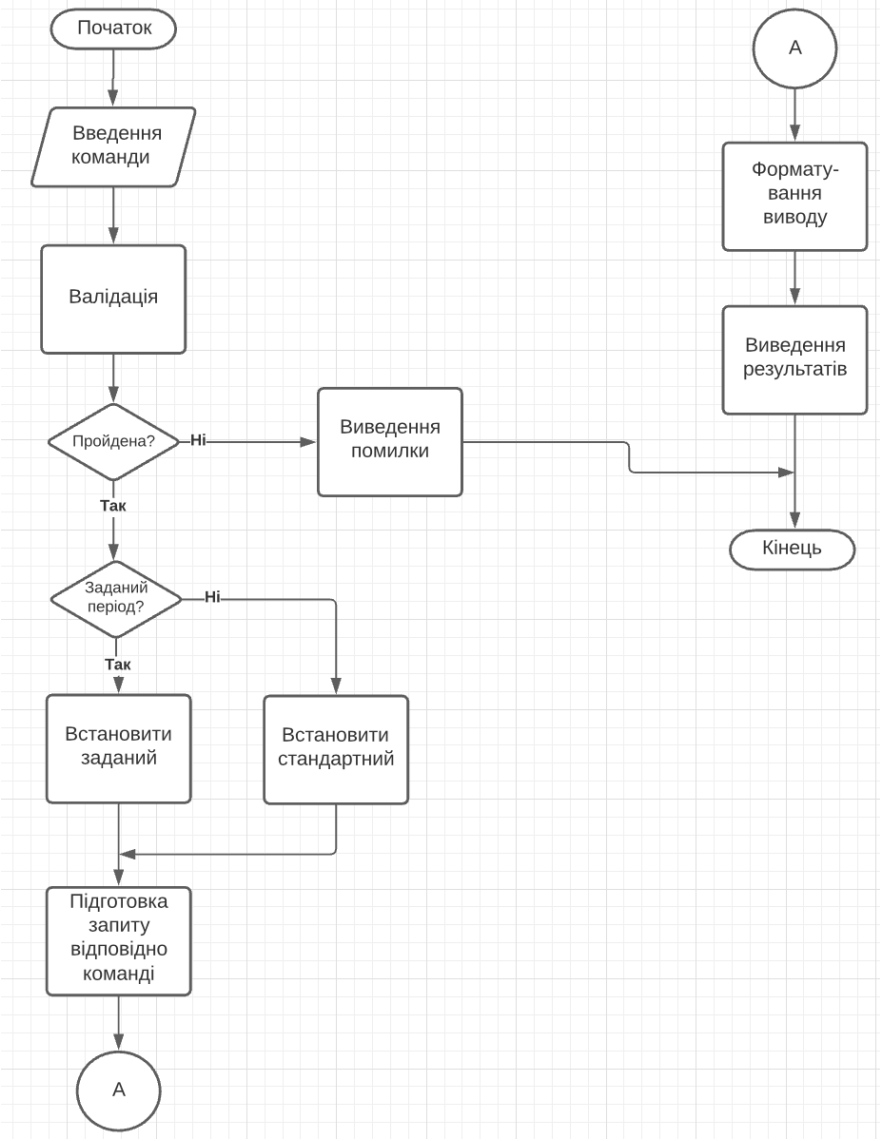

Рисунок 5.1 – Алгоритм обробки запиту статистики

Описаний алгоритм роботи системи подібний для багатьох сценаріїв обробки користувацьких команд. Значні відмінності присутні для алгоритму обробки конфігураційних команд, оскільки їх виконання практично не передбачає запитів до бази даних. Відрізняється також алгоритм обробки запиту пропозиції системи стосовно покращення тестових сценаріїв. Алгоритм обробки запиту пропозиції зображено на Рисунку 5.2.

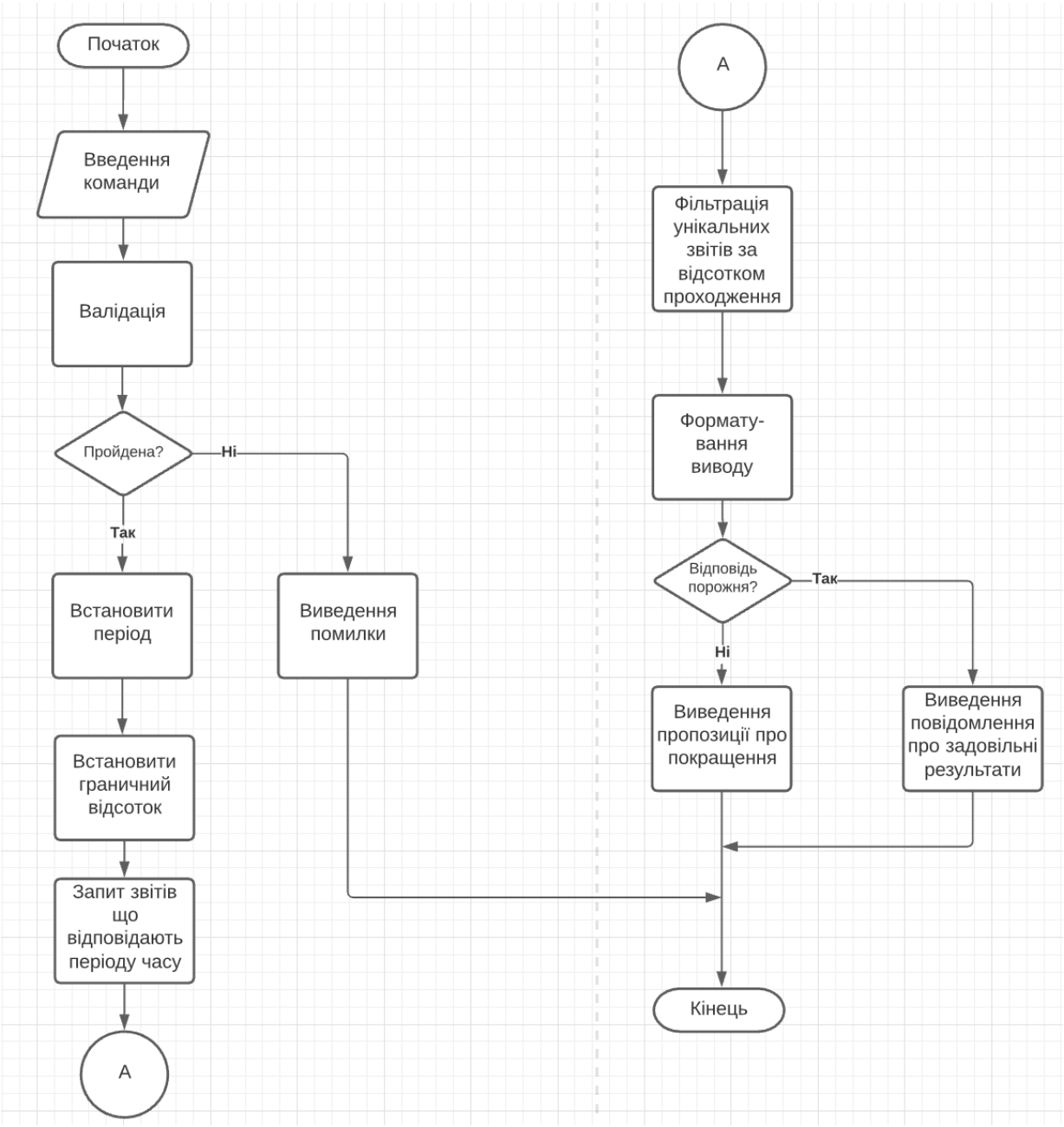

Рисунок 5.2 – Алгоритм обробки запиту пропозиції про покращення тестових сценаріїв

Дана команда не передбачає аргументів. Відповідно, валідація полягає тільки у перевірці правильно написаної команди. Після цього система оцінює результати тестових сценаріїв за останні 30 днів, як найбільш актуальні та встановлює список унікальних тестів (за назвою), процент проходження яких нижче 50%. Передбачається, що такі результати свідчать про нестабільність тестового сценарію та функціональної користі він не несе. Однак при розширені системи буде доцільно додати функціонал що надає користувачу можливість налаштування даних параметрів для своїх потреб в залежності від проекту. Після цього користувачу презентується даний список із повідомленням, що дані тести потенційно нестабільні та користувач має прийняти певні заходи відносно їх. В разі, якщо такі тестові сценарії не знайдені – виведення повідомлення про задовільні результати.

Наразі така система пропозицій оцінює тільки один параметр, однак вона може мати багато додаткових перевірок у майбутньому розширенні системи, наприклад, встановлення які тести вимкнені більше ніж 30 днів та пропозиція їх увімкнути, встановлення, які тести завжди мають позитивний результат з пропозицією перевірити їх функціональність та багато інших.

5.2 Діаграма послідовності зчитування даних та обробки запиту статистики

За допомогою діаграми послідовності можна прослідкувати процес взаємодії між різними об’єктами в порядку їх появи, впорядковано за часом. Вертикальні лінії відображають плин часу, прямокутники на них – час очікування. Суцільними горизонтальними лініями відмічають запит, що прямує від однієї сутності до іншої, а пунктирними – відповідь на запит. Також для додаткової інформації часто додають текстові пояснення для горизонтальних ліній – який саме слідує запис, та що саме у відповіді. На діаграмі зображено процес налаштування підключення до бази даних та запису та зчитування даних системою. Для даної ситуації передбачається що система вже інтегрована з проектом. Відображена взаємодія двох сутностей із системою – користувач та система автоматизації складання. Діаграма послідовності зчитування даних та обробки запиту статистики зображена на Рисунку 5.3.

За даним сценарієм користувач налаштовує специфічну адресу підключення до бази даних, змінюючи системне значення за замовчуванням. Використовуючи команду «db-host» користувач вказує нову адресу. В процесі зміни адреси система виконує базові перевірки підключення та встановлює, що хост або порт, вказаний користувачем не вірний. Дана помилка формується у відповідне повідомлення та повертається користувачу, а стандартна адреса підключення не була змінена.

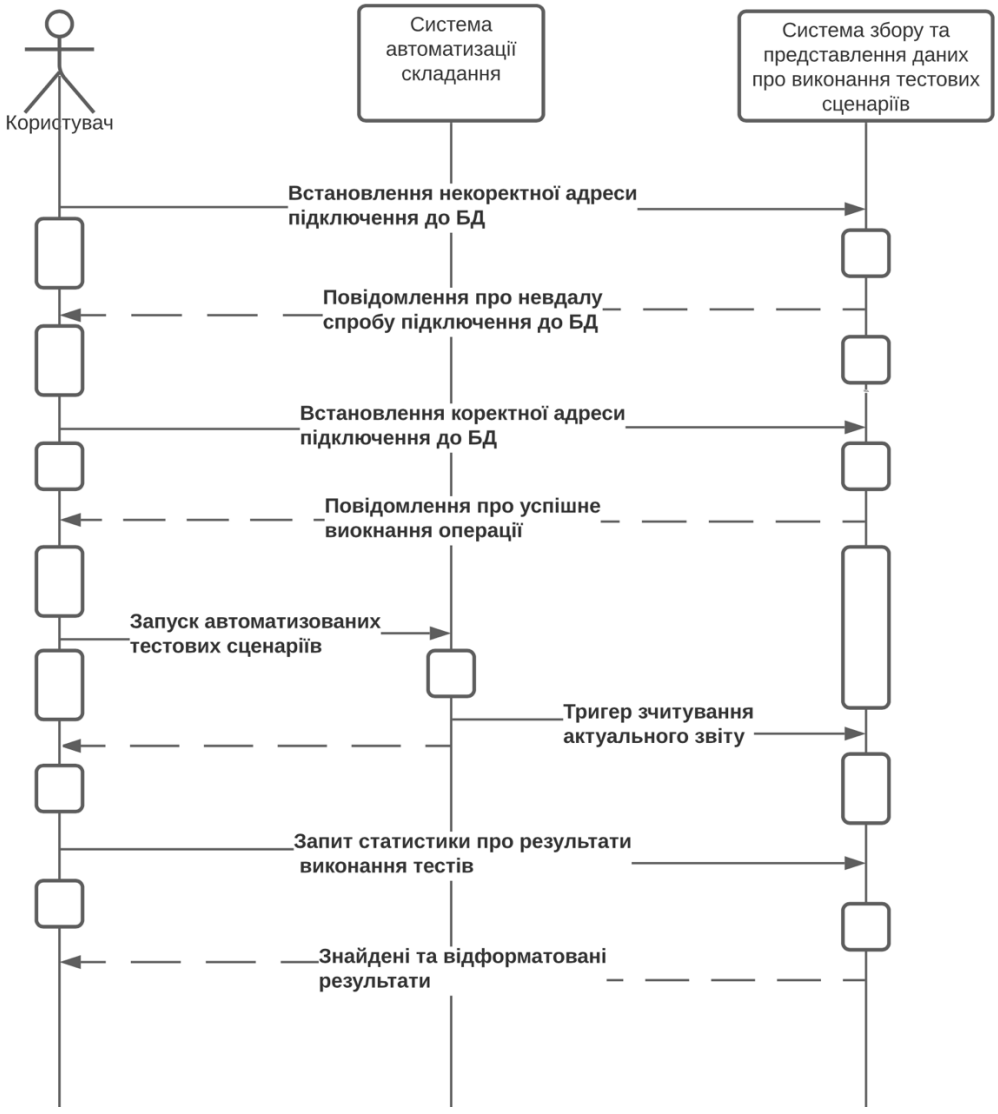

Рисунок 5.3 – Діаграма послідовності зчитування даних та обробки запиту статистики

Після цього користувач вказує коректну адресу, система її перевіряє та змінює значення відповідної конфігураційної змінної. Користувачу повертається повідомлення про успішне завершення операції.

Пісористувач запускає автоматизовані тестові сценарії, після завершення їх виконання системою автоматизованого складання викликається команда, що активує функцію зчитування актуальних звітів системою. Система зчитує звіт згенерований системою автоматизації складання, на його основі формується ряд внутрішніх об’єктів звіту та відбувається запис даних у відповідну колекцію БД (відповідність встановлюється вказанням тестового фреймворку, який використовує користувач).Коли користувач посилає запит на отримання деякої статистики виконання тестів, запит опрацьовується системою. Введені дані підлягають парсингу, валідації команди та її аргументів, на відповідність типів, чи всі необхідні вказані, та інше. Після пройденої валідації, формується відповідний запит до бази до бази даних, отримані дані підлягають фільтрації та сортуванню. Результати обробляються та формуються у зручний для читання текст, який презентується користувачу.

Як видно з діаграми, система має досить просту схему взаємозв'язків, виконуючи кожен крок послідовно та спираючись на результат виконання попереднього.
Проста структура є гнучкою та легко підлягає розширенню системи, потребує мінімальних зусиль користувача для взаємодії.

Висновки до розділу 5

В даному розділі було розглянуто алгоритм роботи системи для двох сценаріїв – запит користувача на отримання статистики та запит на пропозицію про покращення тестових сценаріїв. Для цього було побудовано схеми алгоритмів. Було детально розглянуто послідовність дій, яку має відтворити система для кожного сценарію, описано кожен із кроків. Відрізняються алгоритми роботи при конфігуруванні системи – після етапу валідації команди викликаються відповідні сервіси, які встановлюють нові значення внутрішніх змінних та перевірка валідності нових налаштувань, а результатом виступає виведення повідомлення користувачу.

Також було розглянуто діаграму послідовності для сценарію зчитування даних та обробки запиту статистики. Була розглянута взаємодія системи із користувачем та системою збірки, запити та відповіді, впорядковано за часом.

6 АРХІТЕКТУРА СИСТЕМИ

Для зручності розробки та підтримки системи програмній код, що відповідає за весь її функціонал, можна розбити на деяку кількість модулів, кожен з яких буде виконувати одну конкретну функцію. Таким чином, до основних модулів системи було віднесено:

- модуль роботи із зовнішніми документами, згенеровані фреймоворком для тестування;
- модуль зв’язку із базою даних;
- модуль відображення даних.

6.1 Структурна діаграма

Розроблена структурна схема системи представлена на Рисунку 6.1.

Рисунок 6.1 – Структурна схема

Структурна схема системи відображає систему, розбиту на 5 блоків, які об’єднуються в три основні модулі. Блок роботи із файловою системою зчитує файли, що згенеровані фреймворком для автоматизованого тестування. Система також надає можливість змінювати шлях із заданого за замовчуванням, за яким будуть зчитуватись звіти.

Файли звітів в оригінальному стані мають формат XML, однак, з точки зору проектування системи, оперувати даними у такому вигляді складно та не оптимально, тому в блоці парсингу відбувається їх трансформація у POJO – Plain Old Java Object, після цього всі дані зі звіту виступають структурою даних, якою можна легко оперувати та масштабувати.

Блок роботи із базою даних надає функціонал для запису і зчитування об’єктів із бази даних. Оскільки обрана БД оперує сутностями формату JSON, в даному модулі також знаходиться функція конвертації об’єкту Java в текст формату JSON, та навпаки.

Блок роботи із командним рядком надає можливість введення користувачем команди із атрибутами для подальшої її обробки, та виведення результуючої інформації або повідомлень про помилки назад, у вікно консолі. Даний блок тісно пов’язаний із блоком валідації, де встановлюється, чи коректна була введена команда користувачем та подальші аргументи для неї.

В блоці обробки користувацького запиту розміщений функціонал для підготування відповідного запиту в БД, обробки його результату. Також тут відбувається форматування результуючого звіту у зрозумілій та лаконічній формі.

Стрілки, що підходять до блоків роботи із файлами та обробки запиту ззовні означають зовнішній вплив. Для роботи із файлами – це тригер зчитування нових звітів, а для блоку роботи із командним рядком – це введена команда в командному рядку.

6.2 Інтерфейс взаємодії із користувачем

Для взаємодії із користувачем було розглянуто три варіанти користувацького інтерфейсу:

- взаємодія через вікно desktop-додатку;
- WEB-сторінка додатку;
- інтерфейс командного рядку.

Оформлення інтерфейсу графічно в формі desktop-додатку не виступає кращим

рішенням, оскільки зараз не передбачається надання будь-якої візуалізації знайденої статистики. Також така реалізація суттєво збільшить час на розробку дизайну та оформлення системи як додаток із графічним інтерфейсом, що практично не дасть системі переваг, навпаки, система стане менш зручною у використанні, так як зазвичай в процесі роботи тестувальника або розробника відкрито близько 10 додатків.

Оформлення інтерфейсу у вигляді WEB-сторінки також не оптимальне рішення. Зі сторони розробки – необхідна додаткова компетенція в WEB-технологіях, розгортанні додатку на віддаленому сервісі, слідкування за правильним відображенням для всіх популярних браузерах на всіх популярних операційних системах. При цьому, користувач практично не отримає користі від графічного інтерфейсу.

На сам кінець, використання системи через інтерфейс командного рядку виступає найлегшим у реалізації та найзручнішим у використанні. Всі популярні середовища розробки містять в собі термінал, із якого також можна працювати із системою, маючи на одному екрані програмний код тестових сценаріїв та статистику їх виконання одночасно.

6.3 Модуль парсингу зовнішніх документів

Для прозорішого розуміння взаємодії класів між собою було побудовано UML діаграму класів. Діаграму зображено на Рисунку 6.2

Оскільки дана діаграма відображає тільки взаємодію класів всередині модулю парисngу, на ній не зображено взаємозв’язки, що пов’язують даних модуль із іншими. В даному модулі фактично представлені класи тільки двох логічних категорій – сервісні класи, що знаходяться в лівій частині діаграми та DTO, що розміщені в лівій частині діаграми.

Класи, що відповідають DTO пов’язані між собою відношенням агрегації, та практично всі з них мають зв’язок «один до багатьох». Було встановлене дане відношення, оскільки внутрішні об’єкти DTO також використовуються програмою.

Класи, що відповідають сервісам пов’язані відношенням асоціації. Даним типом відношення відмічений процес обміну даними між сервісами.

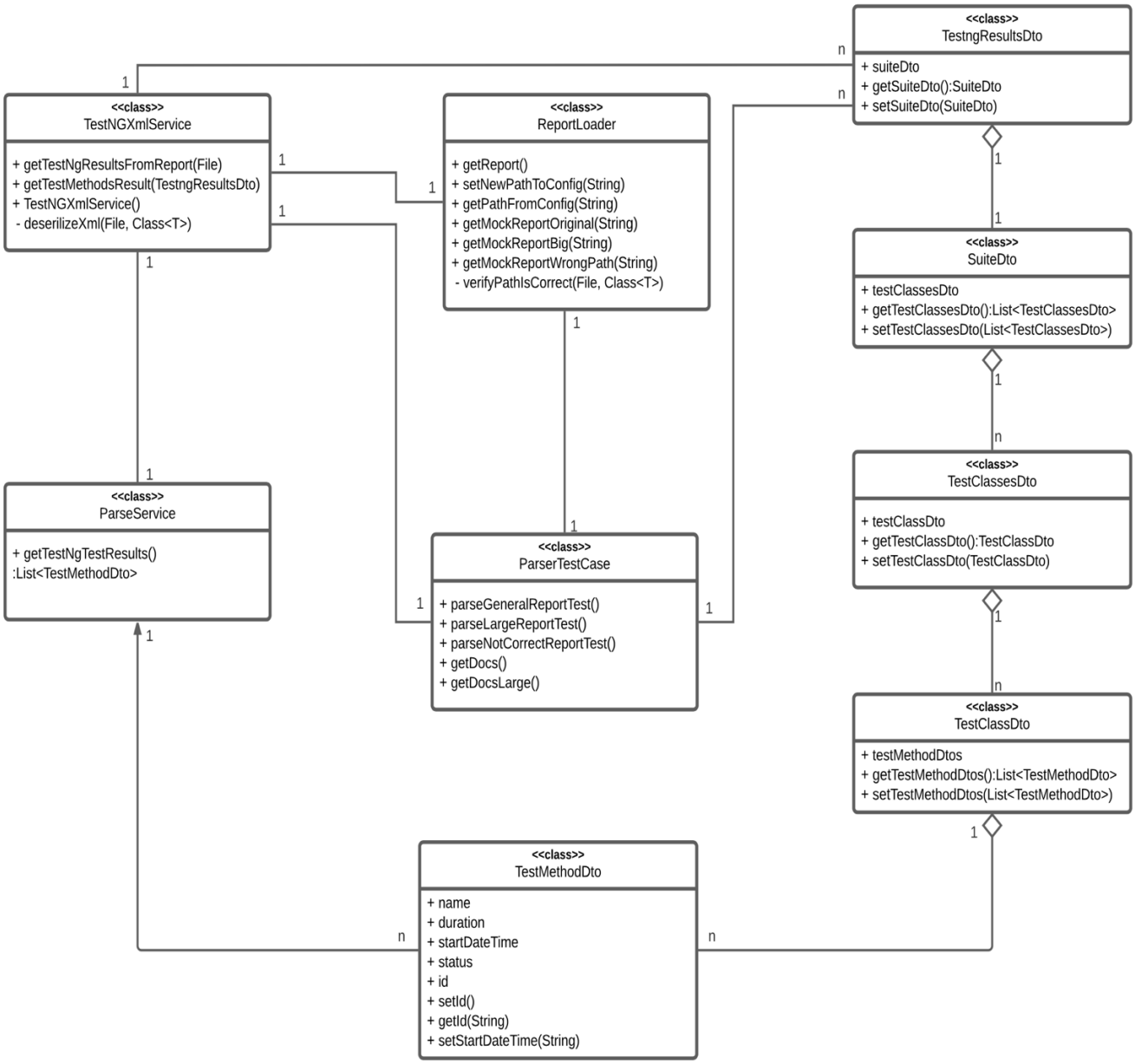

Рисунок 6.2 – Діаграма класів модулю парсингу

Окрім цього, діаграма містить одну неточність – в класі TestMethodDto перелічено поля, які він містить, вони відмічені як маючі публічні методи доступу. В реалізації дані поля приватні, однак для кожного із них впроваджено публічні методи-аксесори. Дані методи аксесори є стандартними та не несуть додаткової логіки, тому їх було опущено на діаграмі а поля відмічені публічними в цілях легшого розуміння діаграми.

Модуль роботи із зовнішніми документами, так званий parser, відповідає за парсинг відповідного документу із інформацією про виконання тестових сценаріїв та формування Java об’єктів зі здобутими даними. Такі об’єкти виступають основними сутностями, якими буде оперувати система і в інші двох модулях. За прийнятими неофіційними правилами іменування таких об’єктів в спільності Java розробників – їх ім’я має завершуватись закінченням “dto”, що означає Data transfer object. Конвенція іменування дозволяє легко виділити такі об’єкти з поміж допоміжних. Дана система має 5 таких об’єктів, які представляють дерево, відповідно до формату документу, який було представлено на Рисунку 3.2. Об’єкти в порядку вкладеності, від зовнішнього до внутрішнього:

– TestngResultDto;
– SuiteDto;
– TestClassesDto;
– TestClassDto;
– TestMethodDto.

Клас TestMethodDto (зображений на Рисунку 6.3) є найнижчім в ієрархії вкладеності, однак є одним із найважливіших, оскільки саме об’єкти його типу зберігаються в базі даних MongoDB у формат JSON. Об’єкти такого типу створюються під час парсингу звіту формату XML. Для коректного парсингу для кожного члену класу було додано анотацію, що свідчить про місцезнаходження атрибуту в дереві документу.

Для підвищення легкості читання коду та збереження єдиного стилю іменування практично для кожного поля було додано параметр localName – за таким ім’ям бібліотека Jackson орієнтується в файлі. Окім цього кожен із членів класу не є окремим полем, всі вони атрибути одного поля (ім’я поля – test-method, що вказано в аргументі анотації для класу).

Також даний тип має змінну id, якої нема в звіті, звідки заповнюються всі інші змінні. Ця змінна виступає в ролі унікального ключа для об’єкту. Для її генерації було використано вбудовану в Java утиліту – UUID, яка генерує випадкову послідовність символів.

Завдяки доцільно використаним анотаціям, що надані бібліотеками Lombok та Jackson, самі методи парсингу виглядають дуже лаконічно та інтуїтивно зрозуміло:

```
@Getter
@Setter
@JacksonXmlRootElement( localName = "test-method")
public class TestMethodDto {
    @JacksonXmlProperty( isAttribute = true)
    private String name;
    @JacksonXmlProperty(isAttribute = true, localName = "duration-ms" )
    private float duration;
    @JacksonXmlProperty(isAttribute = true, localName = "started-at")
    private LocalDateTime startDateTime;
    @JacksonXmlProperty(isAttribute = true, localName = "status")
    private String status;
    private String id;

    public TestMethodDto() { setId(); }

    private void setId() { id = UUID.randomUUID().toString(); }

    public String getId() { return id; }

    public void setStartDateTime(String dateTime){
        this.startDateTime = LocalDateTime.parse( dateTime.replace( target: "Z",  replacement: "" ) );
    }
}
```

Рисунок 6.3 – Клас TestMethodDto

Для коректного мапінгу XML документу було використано механізм ObjectMapper, наданий бібліотекою Jackson. Даний функціонал знаходиться в сервісному класі XmlService. В конструкторі класу, при створенні об’єкту маперу, додатково налаштовано ігнорування атрибутів, що не вказані у dto, оскільки не всі із них необхідні, та не використовувати десеріалізацію за замовчуванням, оскільки класи мають поля, які генеруються, а не зчитуються зі звіту.

Основний метод цього класу – публічний getTestNgResultsFromReport , який примає об’єкт типу File як аргумент та викликає приватний deserilizeXml, який приймає та повертає generic-об’єкт. В результаті, із файлу генерується DTO на найвищому рівні ієрархії – об’єкт TestngResultDto.

В даному модулі також є сервіс зчитування файлу звіту. Основний функціонал розташований в класі ReportLoader, фрагмент якого зображений на Рисунку 6.4. Шлях до цільового файлу звіту знаходиться у файлі config.properites, у вигляді

значення змінної. Мова Java має власний механізм роботи із конфігураційними файлами, що знаходиться в бібліотеці util, його й було використано.

```
public File getReport() { return Paths.get(getPathFromConfig()).toFile(); }

public boolean setNewPathToConfig(String path){
    try (InputStream inputStream = getClass().getClassLoader().getResourceAsStream(propFileName)){
        Properties properties = new Properties();
        if (inputStream != null) {
            properties.load(inputStream);
        } else {
            throw new FileNotFoundException("property file '" + propFileName + "' not found in the classpath");
        }
        properties.setProperty("TARGET_DIRECTORY", path);
        return verifyPathIsCorrect(path);
    } catch (Exception e) {
        System.out.println("Exception: " + e);
    }
    return false;
}
```

Рисунок 6.4 – Фрагмент класу ReportLoader

За замовчуванням, в конфігураційному файлі вказаний шлях до звіту всередині проекту в цілях полегшити та пришвидшити відлагодження написаного коду. Однак даний шлях користувач може легко змінити використавши команду target та вказавши новий шлях, за яким система буде шукати звіт.

При роботі із конфігураційним файлом було використано конструкт мови Java – try with resources, що дозволяє встановити ініціалізацію об’єкту типу InputStream аргументом для блоку try. Такий конструкт передбачає у будь якому випадку закриття потоку, що необхідно із точки зору ресурсу пам’яті, без додаткового описання блоку finaly, що робить програмний код лаконічним та інтуїтивно зрозумілим.

При задані нового шляху в методі setNewPathToConfig також відбувається перевірка, чи заданий шлях існує, та чи є за ним в доступі файл звіту. В разі помилки, коли система не зможе знайти відповідний шлях, або не зможе розпарсити об’єкт звіту й виведе на консоль відповідну помилку.

6.4 Модуль роботи із базою даних

Модуль зв’язку із базою даних відповідає за запис та зчитування даних із

MongoDB. В основному, він оперує тільки двома сутностями, які відображені на діаграмі Сутність-Зв’язок із використанням нотації Чена, зображеної на Рисунку 6.5.

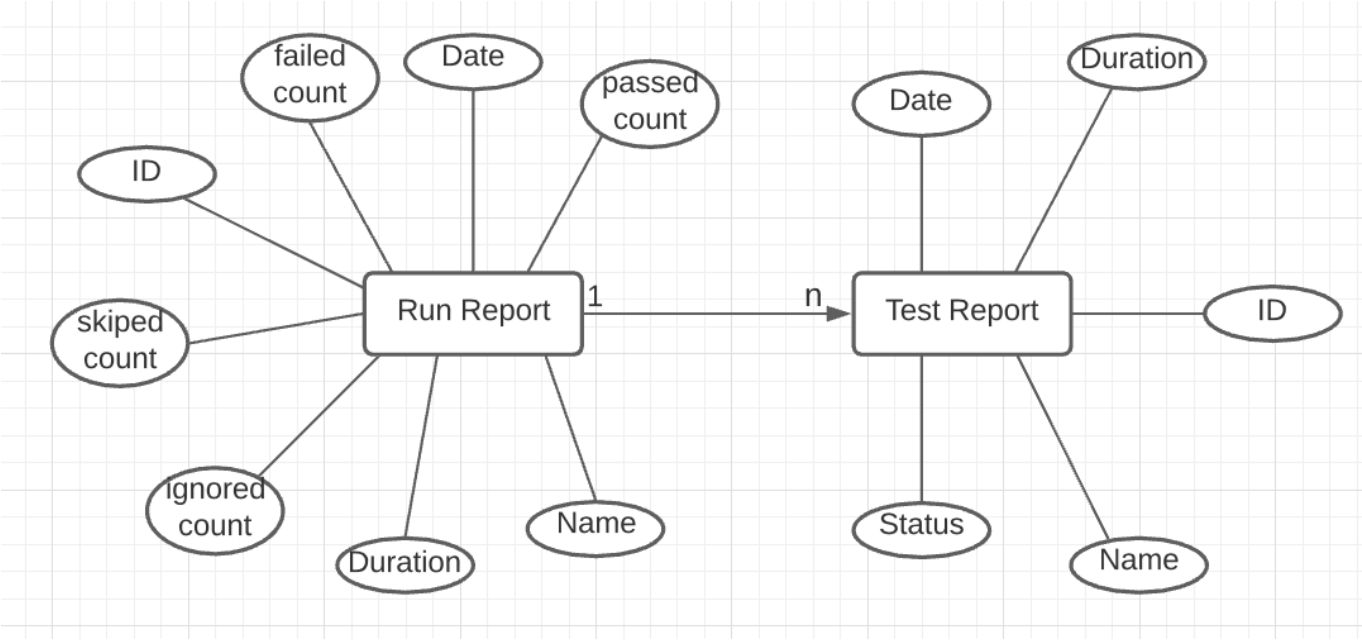

Рисунок 6.5 – ER-діаграма звіту про виконання всіх тестових сценаріїв та звіту про виконання одного тестового сценарію

Завдяки тому, що для роботи системи необхідно зберігати тільки дві пов’язані між собою сутності, стало можливим побудова простої та лаконічної моделі бази даних. Системі необхідна тільки одна база даних, яка утримує в собі набір колекцій для кожного із тестових фреймворків, які підтримує система. Для кожного із фрейморків створюється дві колекції – для сутностей звіту про запуск тестів та для сутностей результатів тесту.

Для кращого розуміння взаємодії класів між собою було побудовано UML діаграму класів для даного модулю. Діаграму класів модулю зв’язку із базою даних представлено на Рисунку 6.6.

Так само як на Рисунку 6.2, зв’язки взаємодії із іншими модулями на даній діаграмі не зображено. В нижній частині діаграми розміщено класи DTO даного модулю. Три класи мають сугубо технічну мету, це класи IdDto, ChronologyDto та DateTimeDto. Їх використання обумовлено особливостями збереження даних в MongoDB.

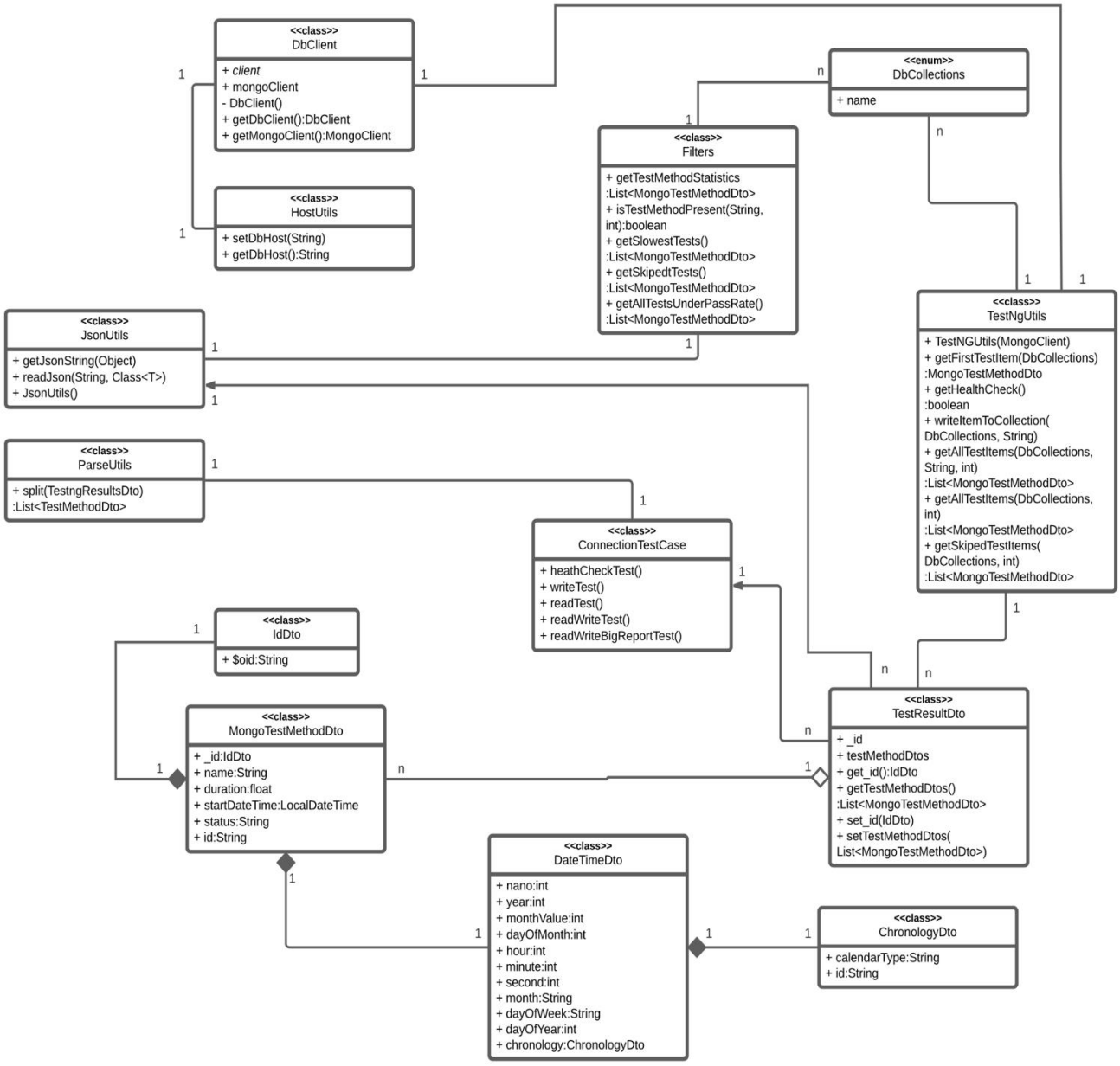

Рисунок 6.6 – Діаграма класів модулю зв’язку із базою даних

Для збереження даних функцією запису приймається аргумент типу TestMethodDto, який використовується для парсингу автоматично згенерованого звіту модулем парсингу. Даний клас містить поле типу LocalDateTime – стандартного типу для операцій із датами, що надає платформа Java. В процесі запису даного типу, механізми бібліотеки mongo-java-driver встановлюють, що даний тип вона не підтримує, та конвертує його в набір полів, що містять інформацію про дату і час.

В процесі зчитування таких звітів не можна використовувати той самий клас

TestMethodDto, так як mongo-java-driver не має механізму автоматичного зворотного парсингу набору згенерованих нею полів в об’єкт типу LocalDateTime. Тому було створено два додаткові класи ChronologyDto та DateTimeDto, які виступають проміжними в процесі отримання поля даного типу. Тому метод setStartDateTime містить додаткову логіку ручного парсингу масиву полів в тип LocalDateTime, що зображено на Рисунку 6.7.

```
public void setStartDateTime( DateTimeDto dateTime){
    startDateTime = LocalDateTime.of( dateTime.getYear(),
                                      dateTime.getMonthValue(),
                                      dateTime.getDayOfMonth(),
                                      dateTime.getHour(),
                                      dateTime.getMinute(),
                                      dateTime.getSecond(),
                                      dateTime.getNano());
}
```

Рисунок 6.7 – Діаграма сутність-зв’язок

Також, через те що тип DateTimeDto містить одинадцять приватних полів та методи-аксесори для кожного із них, для простішого розуміння моделі поля на діаграмі були відмічені як публічні, а методи-аксесори не вказані.

В даному модулі відбувається підключення до бази даних (або створюється нова, якщо вона відсутня) та відповідної колекції (також створюється нова в разі відсутності такої). Єдиною точкою підключення до бази даних виступає клас DbClient, який зображений на Рисунку 6.8.

При проектуванні даного класу, як точки підключення до бази даних, було використано шаблон проектування Singleton. Клас має лише приватний конструктор та статичний метод, який перевіряє, чи був вже створений об’єкт даного класу і, якщо так, повертає цей екземпляр, не створюючи новий. Такий підхід гарантує лише один створений екземпляр об’єкту підключення до бази даних, що суттєво зменшує вірогідність конфліктів між операціями зчитування та запису до бази при паралельному виконанні коду додатку.

```
public class DbClient {
    private static DbClient client;
    private MongoClient mongoClient;

    private DbClient(){
        mongoClient = new MongoClient( new MongoClientURI( new HostUtils().getDbHost() ) );
    }

    public static DbClient getDbClient(){
        if(client == null){
            client = new DbClient();
        }
        return client;
    }

    public MongoClient getMongoClient() { return mongoClient; }
}
```

Рисунок 6.8 – Клас DbClient

При створенні об’єкту клієнту хост та порт, де розвернута база даних зчитується із конфігураційного файлу config.properties. За замовчуванням – задана стандартна адреса та порт, де встановлюється база даних MongoDB – mongodb://localhost:27017.

В даному модулі також можна змінити адресу підключення до бази даних за допомогою функціоналу, що надає сервісний клас HostUtils. При потребі, користувач може змінити адресу та порт підключення за допомогою команди «db-host», в результаті чого викликається метод setDbHost.

Окрім цього наданий функціонал health-check підключення до бази даних, який користувач може запустити командою «healthcheck». Перевірка відбувається в методі getHealthCheck та перевіряє можливість підключення до бази даних, створення колекції, запису та зчитування інформації. Тому користувач в процесі інтеграції системи, або зміни її налаштувань завжди може швидко перевірити її працеспроможність.

Враховуючи особливість MongoDB – оперування об’єктами формату JSON, в даному модулі також є сервіс, що конвертує об’єкти в текст та навпаки – JsonUtils.

Так само, як і з парсингом XML файлів, для конвертації було використано бібліотеку Jackson. Синтаксис дуже подібний та практично не залежить від типу

документу, що підлягає конвертації, що робить такі сервіси легкими у використанні та проектуванні. При створені маперу також було додатково вказано параметри маперу, якими він має керуватись в процесі парсингу об’єкту – була відключена обробка помилок при невідомих аргументах, підключено мапінг масиву JSON об’єктів у масив Java об’єктів.

Сервіси, які безпосередньо опрацьовують запис та зчитування в базу даних знаходяться у окремій директорії, для кожного фрейморку для тестування – окремий сервіс. Хоча вони мають надавати один і той самий функціонал, в залежності від обраного користувачем фреймворку тестування, об’єкти звіту можуть мати різну форму та структуру, відповідно, відрізнятиметься й механізм роботи із ними. Фрагмент класу TestNgUtils зображено на Рисунку 6.9.

Запис та зчитування забезпечуються бібліотекою mongo-java-driver від розробників MongoDB. Бібліотека володіє широким функціоналом, та має лаконічний та легкий для читання синтаксис, вона виступає додатковим шаром абстракції перед інструментами безпосередньої роботи із БД.

Інтерфейс інструменту дозволяє оформляти запити за допомогою програмного коду, не використовуючи безпосередньо синтаксис команд MongoDB, що робить іплементацію даного механізму консистентною з іншими сервісами системи з точки зору проектування архітектури.

Функція зчитування на Рисунку 6.9 також має декілька перевантажених варіантів – функцій із тим самим ім’ям, але різними аргументами. Загалом, вони відрізняються між собою лише набором умов, які виконуються в процесі пошуку записів в колекції, в іншому – алгоритм однаковий.

Умови, в синтаксисі mongo-java-driver – фільтри, всі накладають деяке правило, за яким обраються записи. В даному випадку – необхідно відфільтрувати всі записи за датою (записи, що відповідають більш давнім тестовим звітам ніж вказана кількість днів в другому аргументі функції враховуватись не будуть після даної фільтрації), після цього знайти всі записи для звітів із вказаним ім’ям тесту.

```
public void writeItemToCollection( DbCollections collectionName, String jsonString){
    MongoCollection collection = database.getCollection( collectionName.name() );
    Document person = Document.parse( jsonString );
    collection.insertOne( person );
}

public List<MongoTestMethodDto> getAllTestItems(DbCollections collectionName, String name, int period){
    MongoCollection collection = database.getCollection( collectionName.name() );
    List<Document> documents = new ArrayList<>();
    List<MongoTestMethodDto> resultDtos = new ArrayList<>();
    int dateLimit = LocalDate.now().getDayOfYear() - period;
    MongoCursor i = collection.find( Filters.lte( fieldName: "startDateTime.dayOfMonth", dateLimit) )
            .filter(Filters.eq( fieldName: "name", name )).iterator();
    while(i.hasNext()) {
        documents.add((Document) i.next());
    }
    documents.forEach( doc-> resultDtos.add( JsonUtils.readJson( doc.toJson(), MongoTestMethodDto.class)) );
    return resultDtos;
}
```

Рисунок 6.9 – Фрагмент класу TestNgUtils

В результаті виконання запиту був отриманий об’єкт, що імплементує інтерфейс Iterable, в якому міститься список знайдених записів. Виклик функції iterator() та дії всередині циклу відображають механізм ітерування по об’єкту типу FindIterable<TDocument>. Після чого був отриманий список об’єктів типу Document – одного із типів mongo-java-driver, який можна легко трансформувати в текст, який згрупований за правилами формування JSON, звідки Jackson ObjectMapper конвертує текст в простий java об’єкт, яким системі зручно оперувати надалі.

В даному модулі також розміщений сервіс, що виконує роль абстракції між високорівневими задачами та зв’язком і підключенням для бази даних – клас Filters. Фрагмент класу Filters зображено на рисунку 6.10.

В процесі обробки користувацької команди, модуль презентації викликає сервісні функції даного класу. Клас має тільки статичні поля та методи, оскільки його призначення сугубо сервісне, нема потреби створювати об’єкт такого класу так як об’єкти не будуть мати ніякої логічної різниці між собою. Змінна testNGUtils також промаркована ключовим словом final, що робить змінну константою, а її значення незмінним та попереджує випадкові помилки при подальшому розширені системи.

Метод getSlowestTests на Рисунку 6.10 демонструє, що не всі дані було обрано сортувати можливостями фільтрів для запитів в базу даних. В даному випадку необхідно встановити деяку кількість тестів, що виконувались найдовший час за деякий період часу.

```
private static final TestNGUtils testNGUtils = new TestNGUtils( DbClient.getDbClient().getMongoClient() );

public static List<MongoTestMethodDto> getTestMethodStatistics(String testName, int period){
    return testNGUtils.getAllTestItems( DbCollections.TESTNG_RESULTS, testName , period);
}

public static boolean isTestMethodPresent(String testName, int period){
    return testNGUtils.getAllTestItems( DbCollections.TESTNG_RESULTS, testName, period ).size()>0;
}

public static List<MongoTestMethodDto> getSlowestTests(int testCount, int period){
    return testNGUtils.getAllTestItems( DbCollections.TESTNG_RESULTS, period ) List<MongoTestMethodDto>
            .stream() Stream<MongoTestMethodDto>
            .sorted(Comparator.comparingDouble(MongoTestMethodDto::getDuration))
            .limit(testCount)
            .collect(Collectors.toList());
}
```

Рисунок 6.10– Фрагмент класу Filters

Для цього було використано комбінований підхід – було знайдено всі записи, які відповідали заданому періоду часу за допомогою відповідного запиту в MongoDB, а результуючий список було відсортовано засобами мови Java, а саме – групи методів Stream API. Для цього із об’єкту List було створено потік, для його сортування було використано вбудовану реалізацію функціонального інтерфейсу Supplier, за допомогою якої було задано правило сортування знайдених об’єктів. Кількість об’єктів була обмежена заданою (заданою користувачем, або 10, за замовчуванням) і вони були сформовані у список.

Інтерфейс технології Stream API дуже лаконічний, програмний код із його використанням легкий у читанні та виглядає як логічна послідовність дій зі структурою даних.

Окрім цього, в класі Filters присутній метод getAllTestsUnderPassRate. Він виконує запит в БД, який повертає всі знайдені звіти за вказаний проміжок часу і формує список, в який входять всі унікальні тести, що мають відсоток успішного виконання нижче заданого (функція приймає це значення як аргумент).

6.5 Модуль презентації даних

Для кращого розуміння взаємодії класів між собою в модулі презентації було

побудованого діаграму класів для даного модулю. Дану діаграму зображено на Рисунку 6.11.

Даний модуль оперує невеликою кількістю класів так як він тісно взаємодіє із модулями роботи із базою даних та парсингу, він скоріше виконує транспорту функцію, перенаправляючи запити до інших модулів. В даному модулі нема потреби в реалізації класів DTO, а центральним об’єктом виступає об’єкт типу Request – сутність, що містить всі дані введеної користувачем команди, якими оперує система.

Константний тип UserCommands пов’язаний із типом Request відношенням композиції, UserCommands містить перелічення всіх відомих системі команд, а об’єкт типу Request має відповідати одній із них. Всі інші класи виступають сервісними класами, що пов’язані між собою відношенням асоціації, тобто відношенням обміну даних між собою.

Основна мета даного модулю – підтримка взаємодії з користувачем, а саме, обробка вхідних команд та презентація знайдених даних. Його функціонал можна умовно об’єднати в три групи – зчитування команди, введеної користувачем, опрацювання команди, підготовка відповідних даних, та форматування даних для повернення користувачу.

Практично всі сутності даного модулю – сервісні класи, між якими відбувається обмін даними. Окрім сервісів, представлено дві сутності – типи Request та UserCommands. Сутності містять перелічення команд, які підтримує система та саму сутність користувацького запиту, відповідно.

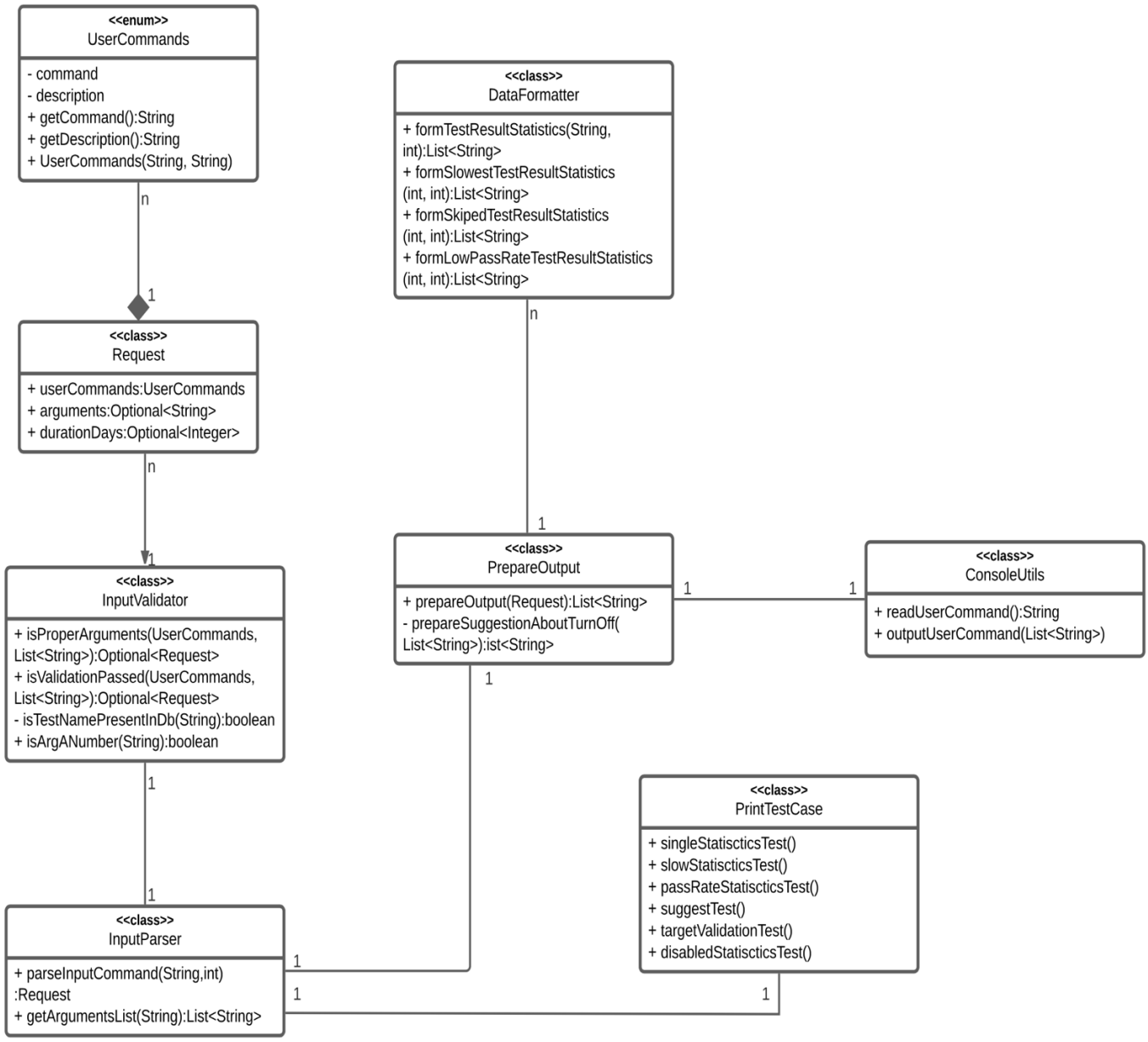

Рисунок 6.11– Діаграма класів модулю презентації

Всі команди, що підтримуються системою зазначені в перелічувальному типі UserCommands. Тип UserCommands зображений на Рисунку 6.12.

Для кожної команди вказано її ім’я, яке використовується в процесі парсингу введеної команди, та опис команди із її сигнатурою – переліком параметрів та лаконічного опису її роботи, що використовується при виклику команди «help». Вибір можливостей enum був влучним, оскільки команди можна легко згрупувати у масив і ітерувати по ньому та використовувати у конструктах таких як switch-case.

```
public enum UserCommands {

    SET_PATH_TO_RESULTS( command: "target",  description: "target {path} - Point to source report"),
    GRAB_RESULTS( command: "grab",  description: "grab - Trigger loading new report"),
    SET_DB_HOST( command: "db-host",  description: "db-host {host:port} - Point to MongoDB host and port"),
    GET_TEST_STATISTIC( command: "test", description: "test {test name} {period in days} - Get statistics about test for period"),
    GET_SLOWEST( command: "slow", description: "slow {test count} {period} - Get slowetst tests for period"),
    GET_TURNED_OFF( command: "disabled", description: "disabled {test count} {period} - Get disabled tests for period"),
    GET_TEST_LOW_PASSRATE( command: "passrate", description: "passrate {limit passrate} {period} - Get tests with lower pass rate for perio
    GET_SUGGESTION( command: "suggest", description: "suggest {period} - Get suggest about test runs"),
    HELP( command: "help", description: "help - Get all commands descriptions");

    private final String command;
    private final String description;
    UserCommands(String command, String description){
        this.command = command;
        this.description = description;
    }

    public String getCommand() { return command; }
    public String getDescription(){ return description; }
}
```

Рисунок 6.12 – Тип UserCommands

Основна сутність, якою оперує даний модуль – об’єкт класу Request відображає користувацький запит. Вона формується в процесі парсингу та валідації команди. Сутність Request зображена на Рисунку 6.13.

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Request {
    private UserCommands userCommands;
    private Optional<String> arguments;
    private Optional<Integer> durationDays;
}
```

Рисунок 6.13 – Сутність Request

Сутність має три параметри – назва команди, аргумент що для неї необхідний (тут вказується ім’я тестового сценарію, або бажана кількість тестів у кінцевому звіті), та період запуску тестів, який має бути охоплений у звіті. Для аргументу та періоду було використано тип Optional, він декларує, що значення змінної може бути пустим і також надає зручний синтаксис для перевірки цього, тому програмний код

має бути адаптований для такого сценарію роботи.

Даний тип було використано із розрахунку, що для деяких команд можуть бути використані стандартні значення – як, наприклад, у випадку, коли користувач хоче отримати вимкнені тести та вказує тільки назву команди, без бажаної кількості унікальних тестів, та періоду часу. В такому разі, будуть обрані стандартні значення – 10 унікальних тестів за останні 30 днів, що значно підвищує якість користувацького досвіду роботи із системою. В результаті, для всіх команд на разі може бути використана тільки одна сутність, яка враховує особливості різних команд.

Окрім цього, програмний код сутності виглядає максимально лаконічно. За допомогою бібліотеки Lombok та анотацій, що вона надає, замість 14 стандартних неінформативних методів, які б робили клас набагато масивнішим, було використано 4 анотації із назв яких зрозуміло, що для кожного поля є публічні методи-аксесори та клас має два публічні конструктори.

Валідація введеної команди проводиться в сервісному класі InputValidator. На даному етапі вже відома команда, яку ввів користувач та необхідно визначити, чи коректні були введені аргументи до неї.

В даній функції система перебирає відомі їй команди із типу UserCommands, та в залежності від тої, що була використана, виконується ряд перевірок. Деякі команди, як target та db-hosts потребують однакових параметрів, тому блоки перевірки для них об’єднані в один. Для команд із опціональними параметрами саме в цьому методі задаються стандартні значення.

Найскладніша валідація – для команди test, оскільки відразу відбувається перевіка на наявність тесту із зазначеним ім’ям в записах БД, в разі негативного результату – користувачу відображається повідомлення. Таке навантаження валідації також може пришвидшити час відгуку системи та покращити користувацький досвід.

Для запитів, що потребують запиту в базу даних було створено сервіс DataFormatter. Його методи взаємодіють із модулем mongo-service, отримані звідти результати підлягають переведенню в формат тексту та групуванню у список. Фрагмент класу DataFormatter зображено на Рисунку 6.14.

Клас виконує сугубо сервісну функцію, він не має полів, тільки публічні

функції. Оскільки нема потреби в інстанціюванні такого класу, всі методи класу статичні. Конвертація отриманих даних виконуються за допомогою Stream API, що виглядає послідовно та лаконічно. Дані сервіси викликаються із класу PrepareOutput, який, в залежності від команди викликає відповідний сервіс та формує кінцевий звіт, що буде виведений на консоль користувачу.

```
public class DataFormatter {

    public static List<String> formTestResultStatistics( String testName, int duration){
        return Filters.getTestMethodStatistics(testName, duration).stream()
                .map(MongoTestMethodDto::toString)
                .collect( Collectors.toList() );
    }

    public static List<String> formSlowestTestResultStatistics(int testCount, int duration){
        return Filters.getSlowestTests(testCount, duration) List<MongoTestMethodDto>
                .stream() Stream<MongoTestMethodDto>
                .map(MongoTestMethodDto::toString) Stream<String>
                .collect( Collectors.toList() );
    }

    public static List<String> formSkipedTestResultStatistics(int testCount, int duration){
        return Filters.getSkipedtTests(testCount, duration) List<MongoTestMethodDto>
                .stream() Stream<MongoTestMethodDto>
                .map(MongoTestMethodDto::toString) Stream<String>
                .collect( Collectors.toList() );
    }
```

Рисунок 6.14 – Фрагмент класу DataFormatter

В класі PrepareOutput також відбувається ітерування по відомим командам типу UserCommands. В залежності від команди, викликається відповідна функція – на оновлення змінної, запит на статистику, тощо. Для сервісних команд тут генерується повідомлення, яке буде пізніше презентовано користувачу

Передбачається, що деякі команди мають в результаті статистику до виведення, деякі – виконували тільки сервісну роботу – як встановлення шляху до звіту, або адреси бази даних, тому для них формується тільки повідомлення про успішне чи ні виконання операції. Варіант за замовчуванням – відповідає за команду help, він збирає всі описи для кожної із команд та формує із цього список.

Оскільки цінність системи полягає у наданні актуальної інформації про виконання тестових сценаріїв, постало питання розробки механізму зчитування актуальних тестових результатів. Було розглянуто декілька варіантів реалізації такого механізму – імплементація так званого listener використовуючи підхід аспектно-орієнтованого програмування та імплементація додаткової команди, яка буде слугувати тригером того, що було завершено виконання тестів та система може зчитати актуальний звіт. Оскільки стабільна робота такого механізму є критично важливою для коректної роботи системи, було розглянуто переваги та недоліки обох підходів.

Реалізація через listener полягає в тому, що в системі описаний деякий сервіс, який очікує виконання деякої події, та має задані інструкції, що мають бути виконаними до події, після неї, або за іншими заданими розробником правилами. Такого типу сервіси часто реалізують слідуванням принципам аспектно-орієнтованого програмування, де основною сутність виступає аспект – модуль або клас, що надає наскрізний функціонал.[11] Для аспекту встановлюється область видимості (наприклад, він націлений на деякий клас, метод, директорію, тощо) та тип події, яку він очікує. Його особливість та зручність в тому, що функціонал аспекту не потребує явного виклику в програмному коді, його реалізація не захаращує проект.

Для рішення механізму тригеру зчитування звіту аспект має бути націлений на файл звіту та очікувати зміни його хеш-суми, або інших подібних параметрів, що були б індикаторами оновлення файлу. Основною перевагою такого підходу виступає відсутність людського фактору, користувач не впливає на даний механізм після інтеграції системи. Основним недоліком є неточність доступних метрик, зміна яких може використовуватись тригером, розширення та поглиблення детальності перевірок будуть потребувати програмної імплементації додаткових сервісів користувачем, що погіршує користувацький досвід.

На противагу цьому було розглянуто та імплементовано інший підхід – винесення механізму тригеру в окрему команду «grab», що доступна користувачу. Передбачається, що користувач буде викликати дану команду після завершення виконання тестів. Оскільки автоматизовані тестові сценарії мають підтримуватись в

стані, коли вони можуть бути запущені через командний рядок, для цього використовуються відповідні конфігураційні файли від систем автоматизації складання проекту. В таких файлах вказується послідовність дій, що має бути використана перед запуском тестових сценаріїв та можуть бути вказані просто командні рядки, а не тільки методи, надані системою автоматизації складання. Тому такий підхід не обвантажить користувача, дана команда викликатиметься автоматично із вказаного файлу. Окрім цього, буде підвищено розуміння взаємодії із системою, чітко видно в який момент та яким чином вона отримує для обробки новий звіт.

Взаємодія із консоллю відбувається за допомогою стандартних команд, вбудовані в Java – Scanner(system.in) та System.out.println. Очевидна простота такого підходу має переваги – робота із консоллю, як точка взаємодії із користувачем, має бути стабільною та передбачуваною і використання вбудованих функцій для цього – один із найкращих варіантів, не передбачує додаткової імплементації вже відомого функціоналу.

Висновки до розділу 6

В даному розділі було розглянуто структуру системи. Для кращого розуміння було побудовано структурну схему для даної системи, на якій відображено п’ять логічних блоків системи, що в реалізації формуть три модулі.

Також було описано архітектуру системи та принципи побудови. Було детально розглянуто кожен із трьох модулів – парсингу, роботи із базою даних та презентації. Розглянуто особливості імплементації сервісів та їх взаємодії між собою, особливості взаємодії системи із користувачем. Вибір принципів та шаблонів проектування архітектури, доцільність їх використання були обґрунтовані.

Для візуалізації сутностей, які зберігаються в базі даних було створено діаграму Сутність-Зв’язок за нотацією Чена. На діаграмі, та із опису модулю роботи із базою даних, чітко помітно, що модель бази даних була створена максимально простою, реалізована взаємодія із нею легко масштабується, що може стати в нагоді при

подальшому розширенні функціоналу системи. Для кожного модулю системи також була побудована діаграма класів, яка відображає орієнтовний вміст класів модулю та їх взаємодію між собою.

В результаті було встановлено, що система володіє всім бажаним фунціоналом, необхідним для продуктивної роботи із нею. Окрім цього, її архітектура була спроектована таким чином, що дозволяє легко масштабуватись та розширювати функціонал системи завдяки слідуванню принципам SOLID та використанню підходящих шаблонів проектування. [12]

При подальшому розширені системи стане необхідним введення додаткових шарів абстракції для модулю роботи із базою даних. Розвиток системи має включати підтримку більшої кількості тестових фреймворків, збільшення кількості метрик, що підтримує система. [13]

Також потенційно корисним було б імплементація графічного інтерфейсу для візуального відображення статистики. Однак реалізація графічного інтерфейсу буде потребувати значних змін архітектури системи та необхідно буде провести додаткове дослідження для встановлення найбільш зручного способу взаємодії із системою для спеціалістів з автоматизованого тестування та розробників.

7 ТЕСТУВАННЯ СИСТЕМИ

В процесі побудови архітектури системи також виконувалось її тестування в цілях підтримки якості програмного коду та імплементованого функціоналу системи. Загальноприйнятим шаблоном тестування виступає пірамідо-подібна модель, в основі якої модульне тестування, після нього інтеграційне і на вершині – end-to-end тести.[14] Піраміда класів тестування зображена на Рисунку 7.1.

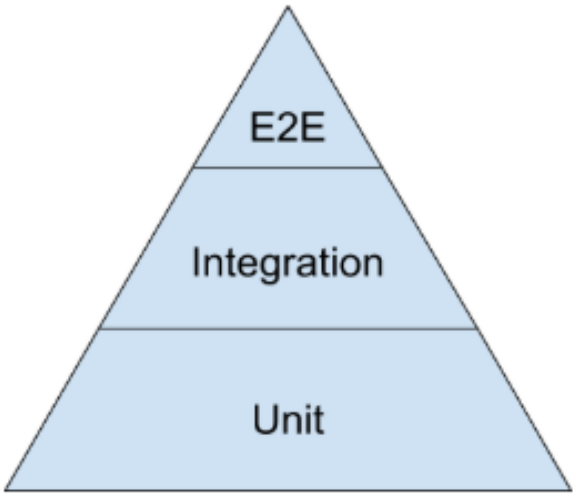

Рисунок 7.1 – Піраміда класів тестування

Така модель визначає оптимальну умовну пропорцію кількості тестових сценаріїв для кожного із видів тестування, тому часто модульних тестів найбільше, для масштабних проектів їх кількість вимірюється сотнями та тисячами, оскільки вони перевіряють найменші структури програмного коду – методи, або навіть строки коду. Інтеграційних тестів зазвичай набагато менше, їх мета в перевірці правильності взаємодії між підсистемами, модулями, мікро-сервісами, тощо. Такі два види тестів можуть бути тільки автоматизованими тестовими сценаріями. На вершині піраміди end-to-end тести, які тестують додаток у кінцевому вигляді, тобто повноцінну роботу всього функціоналу так, як він буде доступним для користувача. Такі тести також можуть бути автоматизованими однак часто вони проводяться мануально на фінальних етапах розробки.

Для тестування функціоналу системи були розроблені інтеграційні автоматизовані тестові скрипти та було проведено мануальне тестування end-to-end.

7.1 Інтеграційні тестові сценарії

Для розробки інтеграційних тестових сценаріїв обрані технології для системи також були підходящими, тому нема необхідності підтримувати проектом декілька мов програмування, велику кількість залежностей, тощо, що робить кінцеву збірку системи більш оптимізованою. Єдина додана технологія для тестування – додатково було підключено фреймворк TestNG, за допомогою його анотацій були оформлені тестові сценарії.

Фреймворк також взаємодіє зі системою автоматизації складання проекту, тому тестові сценарії можна запускати із командного рядку викликаючи Gragle задачу на ім'я test. Вона сканує проект та за замовчуванням знаходить тестові класи, що в кінці імені мають «Test» або «TestCase» (дане налаштування також можна кастомізувати).

Для кожного із програмних модулів було створено ряд тестових сценаріїв, які знаходяться у стандартній директорії для модулю src/test/java/. Звіти, що використовуються для тестових сценаріїв знаходяться в корені проекту.

Для модулю парсингу були написані три інтеграційні тести, що перевіряли основні сценарії, які він має опрацьовувати:

- щасливий шлях парсингу очікуваного звіту в очікуваній директорії;
- щасливий шлях парсингу звіти із великою кількістю тестових звітів;
- негативний сценарій при заданні невірного формату звіту.

Останній сценарій перевіряє як неправильно вказаний шлях до звіту, так і неправильно заданий формат звіту, помилки в процесі парсингу документу.

Загалом тести мають одну і ту саму структуру, що схожа на структуру API тестів «given when then». Це означає, що спершу для тестового сценарію готуються дані для перевірки функціоналу на який націлений тест. Після цього відбувається взаємодія із функціоналом та далі описується, який результат є еталонним та

відбувається порівняння еталонного із актуальним. За результатами перевірки встановлюється, чи вважається тест успішно пройденим чи ні. Приклад тесту для модулю парсингу зображено на Рисунку 7.2.

```
@Test
public static void parseLargeReportTest() {
    File report = reportLoader.getMockReportBig();
    TestngResultsDto testngResultsDto = xmlService.getTestNgResultsFromReport( report );
    List<TestMethodDto> testResults = xmlService.getTestMethodsResult( testngResultsDto );
    testResults.forEach( r-> Assert.assertFalse(r.getName().isEmpty()));
    Assert.assertEquals(testResults.size(),  expected: 44);
}
```

Рисунок 7.2 – Приклад тесту для модулю парсингу

Завдяки інкапсуляції функцій в сервісних класах сам програмний код тесту виглядає стисло та лаконічно. Зрозумілі назви методів сервісних класів та параметри що перевіряються в блоках assert дають змогу зрозуміти що тестується не цікавлячись імплементацією функцій що викликаються – в даному тесті відбувається зчитування звіту, його парсинг та верифікація, що відповідні поля заповнені та їх кількість.

Вказувати константи та літерали в тестах зазвичай вважається поганою практикою, оскільки інколи неясно чому саме такий літерал чи цифру було використано, однак зараз такий підхід допустимий, оскільки тестові дані зчитуються із одного файлу, а не генеруються тестом та їх можна легко знайти та зрозуміти чому саме таке сило було використано. Результат проходження тестів для модулю парсингу зображено на Рисунку 7.3.

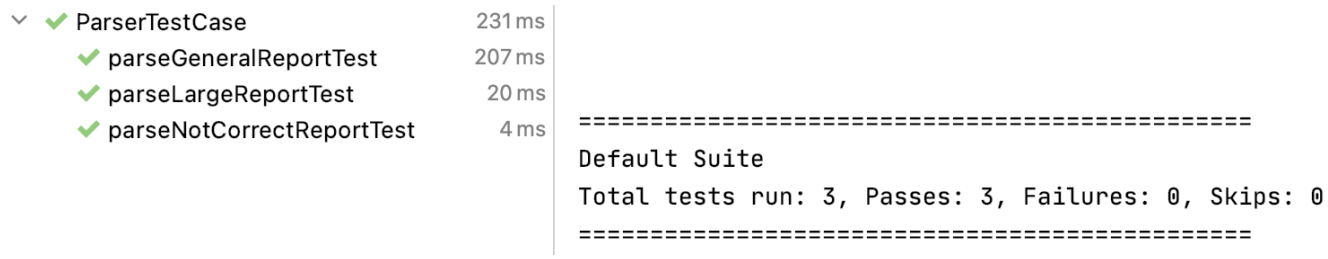

Рисунок 7.3 – Результат проходження тестів для модулю парсингу

Запустивши тестовий клас можна впевнитись, що всі тести мають позитивний результат, актуальні відповіді еквівалентні очікуваним.

Для модулю роботи із базою даних тести мають більш складний процес підготовки даних, в деяких тестах відбуваються неявні перевірки. Було імплементовано тести для перевірки можливості запису та зчитування з бази даних, перевірено роботу методу healthCheck.

Для даного тестового набору було встановлено залежність всіх тестів від одного – того, що тестує healthCheck. У випадку, які даний тест не успішний – це буде означати, що існують деякі проблеми із підключенням до бази даних, або вони виникають в процесі запису чи зчитування даних. Окрім цього, тестам було зазначено строгий порядок запуску, їх було відсортовано від мінімальних перевірок, до більш великих. Це дозволить у разі падінь тестів проаналізувати, який впав перший, можливо, саме він буде вказувати на проблему системи. Результат проходження тестів для модулю роботи з БД зображено на Рисунку 7.4.

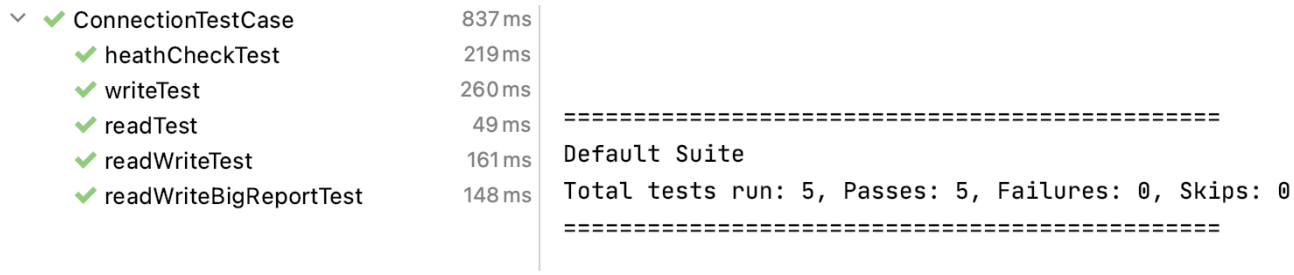

Рисунок 7.4 – Результат проходження тестів для модулю роботи з БД

Для модулю презентації було імплементовано тестові сценарії для кожної з команд, які доступні користувачу. В якості підготовки даних, що необхідні для даних тестів використовуються результати виконання тестів для модулю роботи із базою даних. В даному випадку хоча такі тести не можна вважати ізольованими, це має сенс, оскільки дефекти роботи модулю взаємодії із БД також будуть впливати і на модуль презентації і на його тести. В разі падіння через підготування даних корінь проблеми одразу зрозумілий, що суттєво зменшує час необхідний для розслідування причини падіння тесту. Приклад тесту для модулю презентації зображено на Рисунку 7.5.

```java
@Test
public void suggestTest(){
    String input = "suggest";
    Request request = new InputParser().parseInputCommand( input );
    Assert.assertEquals(request.getUserCommands(), UserCommands.GET_SUGGESTION);
    List<String> output = new PrepareOutput().prepareOutput( request );
    Assert.assertTrue( condition: output.size()>1);
    Assert.assertEquals(output.get(0),  expected: "These tests are flaky, pass rate for all of them is lower 50%");
}
```

Рисунок 7.5 – Приклад тесту для модулю презентації

Тести, так само, мають подібну один одному структуру, відрізняються тільки рядок команди, що зберігається в змінній input та строки коду, що валідують отриманий результат в кінці тесту, додаткова підготовка не потребуються. Результат проходження тестів для модулю парсингу зображено на Рисунку 7.6.

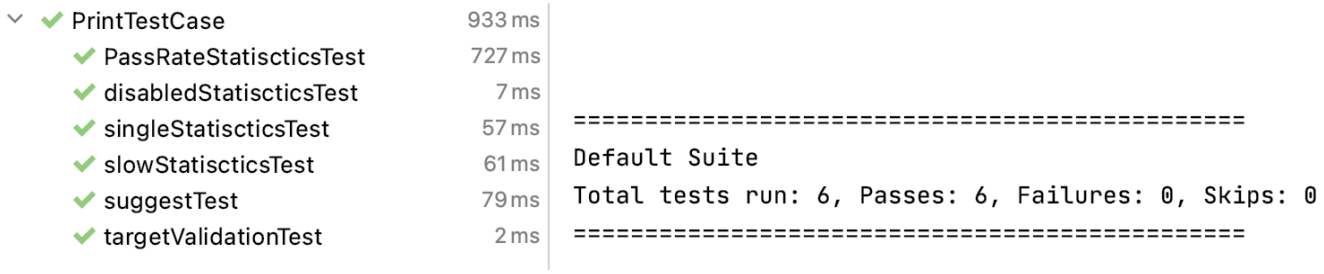

Рисунок 7.6 – Результат проходження тестів для модулю парсингу

7.2 End-to-end тестування

Для кожного варіанту використання та конфігурації системи було описано тестовий сценарій та очікуваний результат. Для того щоб повноцінно протестувати систему було написано невеликий проект із невеликою кількістю тестових сценаріїв також із використанням мови Java, фреймворку TestNg.

В таблицях 7.1 – 7.3 наведено сценарії тестування конфігураційних команд що використовуються в процесі інтеграції системи із проектом автоматизованого тестування та підключення до бази даних.

Таблиця 7.1 – Тест зміни директорії для зчитування звіту

Дія:	Зміна директорії для зчитування звіту	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена	Система готова до запису та зчитування даних	Успішно
Кроки тесту		
Введення команди «target {шлях до файлу звіту}», підставляючи коректний шлях	Значення змінної для шляху до звіту змінено	Успішно
Післяумова		
Перевірка статусу запиту	Виведення на консоль повідомлення про успішне завершення операції	Успішно

Таблиця 7.2 – Тест зміни адреси підключення до бази даних

Дія:	Зміна адреси підключення до бази даних	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена, стандартна адреса змінена вручну	Система готова до запису та зчитування даних	Успішно

Кроки тесту		
Введення команди «db-host {адреса бази даних}», підставляючи коректний шлях	Значення змінної для адреси БД змінено	Успішно
Післяумова		
Перевірка статусу запиту	Виведення на консоль повідомлення про успішне завершення операції	Успішно

Таблиця 7.3 – Тест перевірки статусу підключення до бази даних

Дія:	Перевірка статусу підключення до бази даних	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена.	Система готова до запису та зчитування даних	Успішно
Кроки тесту		
Введення команди «db-healthcheck»	Підключення до БД, створення тестової колекції, запис та зчитування із неї та видалення тестової колекції.	Успішно
Післяумова		
Перевірка статусу запиту	Виведення на консоль повідомлення про успішне завершення операції	Успішно

В таблиці 7.4 наведено сценарій тесту зчитування актуального автоматично згенерованого звіту. Хоча дана команда викликатиметься автоматично системою автоматизації складання проекту, її виконання не відрізняється від від запуску її через командний рядок, як це було протестовано.

Таблиця 7.4 – Тест зчитування актуального автоматично згенерованого звіту

Дія:	Зчитування актуального автоматично згенерованого звіту з результатами виконання тестів.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена.	Система готова до запису та зчитування даних	Успішно
Кроки тесту		
Введення команди «grab»	Пошук файлу за заданим шляхом, парсинг у масив POJO та їх запис у відповідну колекцію.	Успішно
Післяумова		
Перевірка запису в БД через консольні команди MongoDB.	Виведення на консоль повідомлення про успішне завершення операції	Успішно

В таблицях 7.5 – 7.10 наведено сценарії тестування основних команд взаємодії системи із користувачем. Для кожної команди було пройдено окремий сценарій,

оскільки вони можуть виконуватись тільки послідовно та їх результати не пов’язані між собою, вони атомарні.

Таблиця 7.5 – Тест отримання статистики для одного тестового сценарію

Дія:	Отримання статистики для одного тестового сценарію при валідних аргументах та наявності результатів шуканого тесту .	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД існують записи результатів для шуканого тесту.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно
Кроки тесту		
Введення команди «test {назва тесту} {період часу}», де вказується валідна назва тесту та період часу – 10 днів.	Запит до БД із вказаними фільтрами – за назвою та періодом. Формування отриманих результатів у читаємий звіт.	Успішно
Післяумова		
Перевірка відповіді системи.	Виведення на консоль статистики результатів виконання тестових сценаріїв у вигляді списку JSON-об’єктів.	Успішно

Для сценарію, наведеного в таблиці 7.6, не було вказано опціональні аргументи команди. Оскільки механізм обробки не необхідних параметрів один для всіх команд, в даному випадку для попередження матричного тестування, що зайняло б багато часу та не надало б практичної користі, було використано концепцію класів еквівалентності. Таким чином, використання стандартного значення для не необхідного параметру було протестовано тільки в одному тестовому сценарії, а успішні результати дають право вважати, що даний механізм виконується вірно і для інших команд.

Таблиця 7.6 – Тест отримання статистики найповільніших тестових сценаріїв

Дія:	Отримання статистики найповільніших тестових сценаріїв без вказання опціональних аргументів.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД існують записи результатів запуску тестів.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно
Кроки тесту		
Введення команди «slow» без вказання бажаної кількості тестів та періоду часу.	Встановлення стандартних значень – 10 тестів, 30 днів. Запит до БД із фільтрацією за періодом, сортування за часом виконання, пошук 10 унікальних тестів. Формування отриманих результатів у читаємий звіт.	Успішно

Післяумова		
Перевірка відповіді системи.	Виведення на консоль 10 результатів найдовшого виконання для унікальних тестів у вигляді списку JSON-об’єктів.м	Успішно

Таблиця 7.7 – Тест отримання статистики вимкнених тестових сценаріїв

Дія:	Отримання статистики вимкнених тестових сценаріїв із вказанням опціональних аргументів.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД існують записи результатів запуску тестів.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно
Кроки тесту		
Введення команди «disabled {кількість тестів} {період}» де кількість тестів – 5, період – 15 днів.	Запит до БД із фільтрацією за останні 5 днів, та статусом тесту – SKIPED або IGNORED, пошук 5 унікальних тестів. Формування отриманих результатів у читаємий звіт.	Успішно
Післяумова		

Перевірка відповіді системи.	Виведення на консоль 5 результатів вимкнених унікальних тестів за останні 15 днів у вигляді списку JSON-об’єктів.	Успішно

Таблиця 7.8 – Тест отримання статистики тестових сценаріїв з низьким відсотком проходження

Дія:	Отримання статистики тестових сценаріїв з низьким відсотком проходження.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД існують записи результатів запуску тестів.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно
Кроки тесту		
Введення команди «passrate {відсоток проходження}» де відсоток проходження – 50, а період не заданий.	Запит до БД із фільтрацією за стандартний період 30 днів, та визначення всіх тестів, які мали результати «PASS» в меньш ніж 70% запусків. Формування отриманих результатів у читаємий звіт.	Успішно
Післяумова		

Перевірка відповіді системи.	Виведення на консоль всіх встановлених тестів із проходженням менш ніж у 70% випадків за останні 30 днів у вигляді списку JSON-об’єктів.	Успішно

Таблиця 7.9 – Тест отримання пропозиції

Дія:	Отримання пропозиції відносно покращення.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД існують записи результатів запуску тестів.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно
Кроки тесту		
Введення команди «suggest {період}» де період – 7.	Запит до БД із фільтрацією за останні 7 днів, та визначення 10 унікальних тестів, які мали результати «PASS» в менш ніж 90% запусків. Формування отриманих результатів у читаємий звіт.	Успішно
Післяумова		

Перевірка відповіді системи.	Виведення на консоль повідомлення про необхідність перегляду нестабільних тестів та виведення всіх потенційно нестабільних тестів за останні 7 днів у вигляді списку JSON-об’єктів.	Успішно

Таблиця 7.10 – Тест отримання інформації про команди що підтримує система

Дія:	Отримання пропозиції відносно покращення.	
	Результат	Статус
Кроки тесту		
Введення команди «help».	Формування списку всіх доступних команд та короткого опису їх роботи. Виведення повідомлення на консоль.	Успішно
Післяумова		
Перевірка відповіді системи.	Повідомлення містить всі 10 команд та лаконічний опис для кожної із них.	Успішно

Окрім проходження так званих «happy path» – тобто перевірки, що система працює правильно при вірно заданих параметрах та очікуваних діях користувача,

також були пройдені основні негативні сценарії, що можуть виникнути у процесі взаємодії із системою. Негативні сценарії наведено в таблицях 7.11–7.13. Розглянуте підключення до бази даних із відсутністю прав запису до БД, отримання статистики результатів одного тестового сценарію, при його відсутності (або невірно вказаним ім’ям) та сценарій невдалої валідації аргументів.

Таблиця 7.11 – Негативний тест зміни адреси підключення до бази даних та перевірки статусу зв’язку

Дія:	Зміна адреси підключення до бази даних та перевірки статусу зв’язку, при відсутності прав запису в БД.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена, стандартна адреса змінена вручну. Користувачу не надані права запису в БД.	Система готова до запису та зчитування даних	Успішно
Кроки тесту		
Введення команди «db-host {адреса бази даних}», підставляючи коректний шлях.	Значення змінної для адреси БД змінено	Успішно
Введення команди «db-healthcheck».	Підключення до БД, створення тестової колекції. Отримання помилки запису, формування відповідного повідомлення.	Успішно

Післяумова		
Перевірка статусу запиту	Виведення на консоль повідомлення про негативне завершення операції, пропозиція перевірити підключення.	Успішно

Таблиця 7.12 – Негативний тест отримання статистики для одного тестового сценарію

Дія:	Отримання статистики для одного тестового сценарію при відсутності результатів шуканого тесту .	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД не було внесено записи результатів для шуканого тесту.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно
Кроки тесту		
Введення команди «test {назва тесту} {період часу}», де вказується не валідна назва тесту та період часу – 10 днів.	На етапі валідації запит до БД для перевірки наявності будь-яких результатів для вказаного імені тесту. Отримання негативного результату та формування відповідного повідомлення.	Успішно

Післяумова		
Перевірка відповіді системи.	Виведення на консоль повідомлення про відсутність результатів для вказаного тесту.	Успішно

В таблиці 7.13 наведено сценарій негативного тесту валідації аргументів користувацької команди, для якого також було використано концепцію класів еквівалентності. Перевіряючи валідацію аргументів для однієї команди, відповідно верифікується й валідація для інших команд, що забезпечується єдиним використаним механізмом. Такий підхід суттєво пришвидшив процес тестування за рахунок тестування системи як «білого ящику» – із урахуванням особливостей її реалізації.

Таблиця 7.13 – Негативний тест валідації аргументів користувацької команди

Дія:	Отримання статистики для одного тестового сценарію при невірному вказанні аргументів команди.	
	Результат	Статус
Передумова		
СУБД MongoDB встановлена та запущена. В БД не було внесено записи результатів для шуканого тесту.	Система готова до запису та зчитування даних, формування та представлення статистики.	Успішно

Кроки тесту		
Введення команди «test {назва тесту} {період часу}», де вказується валідна назва тесту та період часу – слово «десять».	На етапі валідації запиту спроба розпарсити останній аргумент як чисельний тип. Отримання негативного результату та формування відповідного повідомлення.	Успішно
Післяумова		
Перевірка відповіді системи.	Виведення на консоль повідомлення про невалідність вказаних аргументів.	Успішно

Висновки до розділу 7

Було проведено детальне тестування системи, наслідуючи загально прийняту піраміду тестування. Був опущений етап модульного тестування, однак були імплементовані інтеграційні автоматизовані тестові скрипти, які перевіряють основний функціонал системи, взаємодію модулів між собою.

Окрім цього було проведено мануальне тестування системи end-to-end. Було розглянуто сценарії взаємодії користувача із системою, та пройдено кожен із них із використанням концепції класів еквівалентності. В процесі ручного тестування була створена тестова документація – тестові сценарії, що оформлені в таблицях 7.1–7.13. При розширені функціоналу системи необхідним постане імплементація модульних тестів та тестування навантаженням, подальша оптимізація роботи системи.

8 РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

Маркетинговий аналіз стартап проекту складається із чотирьох етапів:

- визначення ідеї проекту;
- технологічний аудит ідеї проекту;
- аналіз ринкових можливостей запуску проекту;
- формування ринкової стратегії;
- формування маркетингової програми.

Було розглянуто кожен з етапів більш детально.

8.1 Ідея проекту

Було проаналізовано зміст ідеї, напрямки її застосування та потенційні вигоди для користувача, результати сформовані в Таблицю 8.1.

Таблиця 8.1 – Опис ідеї

Зміст ідеї	Напрямки застосування	Вигоди для користувача
		1. Надання чіткої картини стану автоматизованих тестових сценаріїв.
		2. Полегшення та пришвидшення стабілізації тестових сценаріїв, як наслідок – підвищення ефективності процесу тестування ПЗ.

		3. Надання механізму пропозицій про покращення тестових сценаріїв, що може вказувати на проблеми, що користувач раніше не приймав до уваги.

Від існуючих аналогів та замінників продукт відрізняється тим, що він володіє тільки вузькоспеціалізованим детально опрацьованим функціоналом для рішення однієї задачі. Продукт має максимально лаконічний інтерфейс та є легким в інтеграції.

Для визначення сильних та слабких сторін ідеї проекту, її було порівняно із конкурентними рішеннями, результат визначення техніко-економічних характеристик ідеї зображений в Таблиці 8.2.

Таблиця 8.2 – Визначення техніко-економічних характеристик ідеї

№ п/п	Техніко-економічні характеристики ідеї	Конкуренти				W	N	S
		Мій проект	TeamCity CI	Sonarqube	TestRail			
1.	Збір статистики про виконання тестових сценаріїв	+	+	+	+			+

2.	Можливість виведення статистики виконання за заданими метриками	+	-	-	-			+
3.	Графічний інтерфейс	-	+	-	+		+	
4.	Зручність взаємодії із користувачем	+	-	-	-			+
5.	Можливість надання пропозиції про покращення тестових сценаріїв	+	-	-	-	+		
6.	Інтеграція зі сторонніми системами	-	+	+	+		+	

В результаті було встановлено, що, з техніко-економічної точки зору в порівнянні із рішеннями конкурентів, ідея проекту не передбачає графічного користувацького інтерфейсу та інтеграції за сторонніми інструментами. Однак дані характеристики були визначені як нейтральні, тому ідея вважається конкурентноспроможною.

8.2 Технологічний аудит ідеї проекту

Було проаналізовано технологічну можливість реалізації ідеї проекту. Для цього було розглянуто які технології можна використати для реалізації ідеї проекту та їх доступність. Результати технологічного аудиту в Таблиці 8.3.

Таблиця 8.3 – Технологічний аудит

№ п/п	Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
1.	Мова програмування	Java	Наявна	Доступна
2.	СУБД	MondoDB	Наявна	Доступна
3.	Парсинг звіту з результатами	Jackson, власна реалізація	Бібліотека Jackson, використана за основу, наявна	Доступна
4.	Взаємодія із БД	Mongo-java-driver, власна реалізація	Бібліотека Mongo-java-driver, використана за основу, наявна	Доступна
5.	Сортування даних за заданими метриками	Власна реалізація	Власна реалізація	Власна реалізація
6.	Механізм тригеру зчитування актуального звіту	Власна реалізація	Власна реалізація	Власна реалізація

Обрана технологія реалізації проекту: Проект розроблений на платформі Java із використанням MongoDB, в процесі розробки механізмів парсингу, тригеру

зчитування звіту, формування статистики та взаємодії із БД було використано додаткові інструменти Jackson та Mongo-java-driver для спрощення імплементації.

Встановлено, що ідея проекту технологічно здійсненна, для реалізації можуть бути використані наявні технології у вільному безкоштовному доступі, однак специфіка системи також передбачає й власну реалізацію частини механізмів.

8.3 Аналіз ринкових можливостей проекту

Таблиця 8.4 – Огляд потенційних користувачів проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів
Необхідність отримання актуальної, цілісної та достовірної інформації про результати виконання автоматизованих тестових сценаріїв	Спеціалісти в автоматизованому тестуванні, розробники ПЗ	– Налаштування конфігурацій системи; – Запит статистики про тестові сценарії.	– Різноманіття метрик виведення статистики; – Легкість взаємодії з системою; – Повнота даних, що представлені у звіті.

Було встановлено основних потенційних користувачів проекту, розглянувши потреби ринку, цільову аудиторію, відмінності у поведінці різних груп клієнтів та вимоги споживачів. Результати огляду наведено в Таблиці 8.4. Таким чином, було встановлено, що цільова аудиторія – спеціалісти в автоматизованому тестуванні, розробники ПЗ.

Окрім цього, було розглянуто фактори, що ускладнюють процес виходу проекту на ринок та фактори що полегшують даний процес. Більш детально дані фактори описані в Таблиці 8.5 та Таблиці 8.6.

Серед основних загроз було визначено фактор конкуренції та технологічний фактор. Для обох факторів була сформована можлива реакція компанії. Для попередження технологічного – система має підтримуватись та вчасно оновлюватись для відповідності новим версіям фреймворків тестування, з якими вона взаємодіє. Окрім зменшення ризику появи даної загрози це забезпечить стабільну роботу продукту.

Таблиця 8.5 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Конкуренція	На ринку вже представлені рішення, які частково виступають рішеннями. Вони також мають багато стороннього функціоналу та часто використовуються в процесі розробки для рішення інших задач.	Активні рекламні заходи із акцентом на варіативність функціоналу системи, простоту роботи із нею.
2.	Технологічний	Фреймворк, який не підтримується системою може стрімко набрати популярність, велика кількість потенційних користувачів втратить інтерес до системи через неможливість її використання.	Постійний моніторинг тенденції ринку тестових фреймворків, підтримка та оновлення функціоналу системи для різних технологій.

Було виділено три основні фактори можливостей, що сприятимуть виходу проекту на ринок та сформовано потенційну реакцію компанії на них – рекламна кампанія, оновлення продукту та його документації для заохочення більш широкої

аудиторії користувачів.

Також було проведено ступеневий аналіз конкуренції на ринку для визначення особливостей конкурентного середовища та як вони впливають на продукт.[15] Результати аналізу наведено в Таблиці 8.7. Було проведено аналіз конкуренції на ринку згідно М.Портера (Таблиця 8.8).[16]

Таблиця 8.6 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Розширення ринку автоматизованого тестування	При збільшені кількості спеціалістів галузі, частіше постає питання стабілізації тестових сценаріїв, більше потенційних клієнтів продукту.	Проведення маркетингових заходів, рекламної кампанії, представлення продукту на технічних конференціях.
2.	Вдосконалення реалізації	Рефакторинг програмного коду із ціллю підвищити гнучкість проекту, оптимізувати роботу системи за часом та ресурсами, що позитивно вплине на користувацький досвід.	Ведення документації версій системи, відведення ресурсів для вдосконалення кодової бази системи.

3.	Взаємодія зі стороннім ПЗ	Взаємодія із системами контролю версій, системами CI/CD, тощо допоможе швидше та непомітніше для користувача отримувати звіти, що також підвищить зацікавленість продуктом.	Оновлення архітектури системи, проектування та документація нових функціональних можливостей.

Таблиця 8.7 – Аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії що бути конкурентноспроможною)
1.Тип конкуренції – олігополія	На ринку представлено декілька сильних конкурентів.	Наголошення на широких вузькоспеціалізованих функціональних можливостях системи, легкості її використання та прозорості роботи.

2. Рівень конкурентної боротьби – міжнародна	Рішення конкурентів не орієнтовані на країну чи деяку область, використовуються по всьому світу.	Нарощення функціоналу для підтримки більш специфічних користувацьких технологій та потреб. Використання англійської мови інтерфейсу.
3. За галузевою ознакою – внутрішньогалузева	Система представляє інтерес тільки тестувальникам та розробникам в процесі розробки програмного забезпечення.	Глибоке дослідження цільової галузі та тенденцій її розвитку. Оновлення та адаптація системи в залежності від змін потреб користувачів.
4. За видами товарів – товарно-родова конкуренція	Існує конкуренція із системами, що використовуються для рішення інших задач, однак можуть використовуватись як аналог даного продукту	Необхідність розширення функціоналу системи, акцентуація в маркетинговій кампанії на систему, як цілісне та повноцінне рішення однієї конкретної задачі.

5. За характером переваг – нецінова	При виборі рішення користувач переважно звертає увагу на функціональне оснащення системи, ціновий фактор не виступає ключовим.	Необхідність підтримки детальної документації продукту.
6. За інтенсивністю – не марочна	Марка продукту не характеризує його, не впливає на якість системи.	Не впливає.

Таблиця 8.8 – Аналіз конкуренції за М.Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Teamcity CI, TestRail, Sonarqube	Розробники програмного забезпечення, спеціалісти автоматизованого тестування.	Відсутні	Розробники програмного забезпечення, спеціалісти автоматизованого тестування.	Власна розробка

Висновки	Присутні прямі конкуренти.	За рахунок наявності широких функціональних можливостей для рішення вузькоспеціалізованої проблеми, низька ймовірність появи аналогічного продукту.	Постачальники не впливають на конкуренцію в галузі.	Велика кількість клієнтів, число продовжуватиме збільшуватись із розширенням ринку розробки та тестування ПЗ.	Власна розробка буде більш обмеженою за функціональністю та буде орієнтована на тільки на потреби одного користувача.

В результаті аналізу конкуренції на ринку, було встановлено, що в обраній галузі існують прямі конкуренти продукту та товари-замінники. Для оцінки змоги продукту позиціонувати себе як конкурентноспроможний було оцінено фактори конкурентноспроможності (Таблиця 8.9). Було визначено та проаналізовано чотири основні фактори, які вказують на те, що продукт має потенціал для виходу на ринок.

Таблиця 8.9 – Фактори конкурентноспроможності

№ п/п	Фактор конкурентноспроможності	Обґрунтування
1.	Проста взаємодія із користувачем.	Взаємодія із користувачем відбувається через максимально простий інтерфейс – командний рядок, що дозволяє взаємодіяти із системою напряму через середу розробки, на відміну від конкурентних рішень із громіздким та складним у навігації користувацьким інтерфейсом.
2.	Велика кількість метрик виведення статистики.	На відміну від конкурентних рішень, проект володіє набором форматів виведення інформації, що будуть найбільш зручними для користувачів із різними потребами.
3.	Легкість нарощування нового функціоналу	Архітектура системи побудована із використанням відомих практик, можливість її роширення була закладена ще на етапі проектування, тому система легка у підтримці та масштабуванні.
4.	Легкість інтеграції із існуючим проектом автоматизованого тестування.	Для інтеграції системи необхідно виконати тільки невеликий ряд кроків, що інтуїтивно зрозумілі для користувача та не потребують спеціальної кваліфікації.

На основі факторів також було проведено оцінку сильних та слабких сторін товарів-конкурентів в порівнянні із даним продуктом, результати наведені в Таблиці 8.10.

Таблиця 8.10 – Порівняльний аналіз слабких та сильних сторін

№ п/п	Фактор конкурентноспроможності	Рейтинг товарів-конкурентів у порівнянні із системою						
		-3	-2	-1	0	+1	+2	+3
1	Проста взаємодія із користувачем.			+				
2	Велика кількість метрик виведення статистики.		+					
3	Легкість нарощування нового функціоналу					+		
4	Легкість інтеграції із існуючим проектом автоматизованого тестування.					+		

Також було проведено визначення слабких та сильних сторін проекту за допомогою SWOT-аналізу (Таблиця 8.11).[17] Визначено сильні сторони продукту, на які необхідно робити наголос при пошуку потенційних клієнтів, слабкі сторони, які мають бути мінімізовані в майбутньому та фактори можливостей та загроз що можуть вплинути на продукт.

Таблиця 8.11 – SWOT-аналіз

Сильні стороні: – легка взаємодія із користувачем; – велика кількість метрик збору та виведення статистики; – легкість інтеграції існуючими системами; – простота в нарощенні нового функціоналу.	Слабкі стороні: – відсутність візуалізації сформованої статистики; – не всі популярні тестові фреймвоки на разі підтримуються системою.

Можливості: – імплементація можливості відображення знайденої інформації графічно; – розширення системи для підтримування більшої кількості фреймовркв тестування.	Загрози: – нарощення подібного функціоналу конкурентами.

На основі SWOT-аналіу були розроблені альтернативні комплекси заходів для виходу проекту на ринок. Альтернативи ринкового впровадження наведені в таблиці 8.12.

Таблиця 8.12 – Альтернативи ринкового впровадження

№ п/п	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Розроблення MVP продукту, вихід із ним на ринок для пошуку перших клієнтів та поступове нарощення функціоналу для розширення клієнтської бази.	Висока	3 місяці
2.	Розроблення всього необхідного функціоналу та підвищення ціни підписки на продукт.	Низька	6 місяців

В результаті аналізу альтернатив було обрано перший варіант – вихід на ринок із мінімальним функціоналом для задоволення користувацьких потреб. Рішення прийняте завдяки високій ймовірності отримання ресурсів та невеликим строкам реалізації, в порівнянні з другим варіантом.

8.4 Побудова ринкової стратегії проекту

Побудова стратегії поведінки компанії на ринку потребує визначення основних груп потенційних користувачів продукту. Для цього було проведено їх огляд, результат наведено в Таблиці 8.13.

Таблиця 8.13 – Опис цільових груп потенційних користувачів

№ п/п	Опис профілю	Готовність сприйняти продукт	Орієнтовний попит в межах сегменту	Інтенсивність конкуренції	Простота входу
1.	Спеціалісти автоматизованого тестування ПЗ	Висока	Висока	Середня	Проста
2.	Розробники ПЗ	Висока	Середня	Низька	Складна

Як цільові групи було визначено спеціалісти автоматизованого тестування та розробники програмного забезпечення. Для обох груп інтенсивність конкуренції не вища за середню, а орієнтовний попит не нижчий за середній. Компанія розраховує на задоволення потреб однієї галузі, тому обрана стратегія охоплення ринку – стратегія концентрованого маркетингу.

Таблиця 8.14 – Визначення стратегії конкурентної поведінки

Чи є прохід «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики продукту конкурентів?	Стратегія конкурентної поведінки

Ні	Компанія буде шукати нових користувачів.	Частково функціонал буде скопійованим, однак основний функціонал та підтримка різних тестовий фреймворків полягає у власній розробці.	За рахунок невеликого розміру цільового сегменту ринку та стабільності проблеми стабілізації тестових сценаріїв обрано стратегію зайняття конкурентної ніші.

Після визначення стратегій охоплення ринку та конкурентної поведінки був розроблений підхід позиціонування проекту на ринку, описані основні асоціації проекту. Визначення стратегії позиціонування наведено в Таблиці 8.15.

Таблиця 8.15 – Визначення стратегії позиціонування

Вимоги до продукту	Базова стратегія розвиту	Конкурентноспроможні позиції проекту	Вибір асоціацій, що мають сформувати комплексну позицію проекту (три ключових)

Презентація актуальних звітів із необхідними атрибутами за популярними метриками.	Стратегія позиціонування категорії	Висока кількість метрик, що необхідна для ефективного процесу стабілізації тестових сценаріїв, взаємодія із автоматично генерованими звітами, як наслідок, надання повної та достовірної інформації.	– Зручність, – Гнучкість, – Самостійність.

8.5 Побудова маркетингової програми проекту

Було сформовано маркетингову концепцію товару для даного проекту, для цього були описані потреби користувачів, вигоди від використання товару та переваги перед конкурентами. Основні переваги концепції потенційного товару наведені в Таблиці 8.16.

Таблиця 8.16 – Основні переваги концепції потенційного товару

№ п/п	Потреба	Вигода яку пропонує товар	Ключові переваги перед конкурентами
1.	Отримання актуальної та повної інформації про виконання тестових сценаріїв	Різноманіття форматів виведення статистики, зручна взаємодія із системою, що потребує мінімальних зусиль користувача.	– Зручний формат виведення статистики – Велика кількість метрик формування статистики.

2.	Отримання пропозицій про покращення тестових сценаріїв	Функція може бути корисною для початківців в галузі, звертає увагу на проблеми, які користувач міг не помітити.	– Багатократна оцінка тестових сценаріїв; – Єдиний інтерфейс взаємодії із користувачем.

Після визначення концепції товару було розглянуто трирівневу маркетингову модель товару, основа якої полягає у визначення сутності та складових товару за задумом, у реальному виконанні та з підкріпленням. Огляд трьох рівнів моделі товару було наведено в Таблиці 8.17.

Таблиця 8.17 – Три рівні моделі товару

Рівні товару	Сутність та складові
1. Товар за задумом	Система, що збирає результати виконання тестових сценаріїв, на потребу користувача повертає їх за заданими метриками. Вигоди від використання продукту: – Підвищення якості процесу стабілізації тестових сценаріїв; – Отримання чіткої картини стану тестових сценаріїв.

2. Товар у реальному виконанні	Властивості та характеристики: – Актуальність; – Гнучкість архітектури, готовність до розширення системи; – Легка взаємодія із користувачем; – Реалізовано основні метрики представлення статистики; – Механізм пропозиції для покращення стану тестових сценаріїв.
3.Товар із підкріпленням	Інтеграція із більшою кількістю фреймовкрів тестування, описання більш спецефічних метрик представлення статистики, удосконалення механізму генерації пропозицій.

Захист ідеї проекту полягає у об’єднанні властивостей та характеристик що закладені в реальному виконанні товару та в товарі із підкріпленням. Для успішної побудови бізнес-моделі також необхідно визначити особливості системи збуту та встановити оптимальну систему. Формування системи збуту наведено в таблиці 8.18.

Таблиця 8.18 – Формування системи збуту

Специфіка закупівельної поведінки цільових споживачів	Функції збуту, що має виконувати постачальник	Глибина каналу збуту	Оптимальна система збуту
Придбання підписки на продукт, за яку плата стягається раз в деякий період.	Забезпечення постійної наявності продукту, організація легкого доступу до нього.	Нульова, посередники відсутні.	Проведення збуту власними силами.

Після визначення оптимальної системи збуту даного товару необхідно визначити останню складову маркетингової програми – огляд концепції маркетингових комунікацій та визначення завдання рекламного повідомлення та концепції рекламного звернення. Концепція маркетингових комунікацій наведена в таблиці 8.19.

Таблиця 8.19 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються клієнти	Ключові позиції, обрані для позиціювання	Завдання рекламного повідомлення	Концепція рекламного звернення
Вибір продукту, спираючись на його функціональне оснащення, легкість інтеграції та взаємодії із ним.	Інтернет, технічні конференції.	Гнучкість функціоналу системи, легкість взаємодії з нею.	Знайомство потенційних користувачів із продуктом.	Наголошення на варіативності форматів виведення статистики, зручності використання та інсталяції продукту.

Висновки до розділу 8

В даному розділі було проведено розробку стартап-проекту. В рамках цього було проаналізовано ідею проекту та напрямки її застосування, визначено її техніко-економічні характеристики, но основі чого було встановлено що ідея є конкурентноспроможною. Було проведено технологічний аудит системи та

становлено, що вона є технологічно здійсненною, всі наявні технології необхідні для цього доступні.

Було проведено аналіз ринкових можливостей проекту, встановлено фактори загроз для проекту, серед основних виділено технологічний фактор та вплив конкурентів, однак були встановлені вірогідні кроки компанії для таких випадків. Також було визначено фактори можливостей та потенційна реакція компанії на них.

Було проведено ступеневий аналіз конкуренції на ринку та аналіз за М.Портером. Було встановлено, що наразі на ринку існують безпосередні конкуренти та можлива поява товарів-замінників, тому було визначено фактори конкурентноспроможності системи та на їх основі проведено оцінку слабких та сильних сторін товарів конкурентів та проведено SWOT-аналіз. В результаті було визначено, на які сильні сторони проекту слід робити акцент в маркетинговій кампанії – велика кількість метрик виведення статистики, легкість взаємодії із користувачем, та які слабкі сторони потребують доопрацювання в процесі розвитку системи – збільшення кількості фреймворків що підтримуються системою та реалізація візуального представлення сформованої статистики.

Було визначено альтернативи ринкового впровадження. За найвищою ймовірністю отримання ресурсів та найкоротшими строками реалізації було обрано стратегію розроблення MVP продукту та виходу із ним на ринок із поступовим нарощенням функціоналу та пошуком клієнтів одночасно.

Було побудовано ринкову стратегію проекту, встановлені цільові групи потенційних клієнтів, стратегій конкурентної поведінки та позиціонування товару на ринку. Була побудована маркетингова програма проекту та визначено оптимальну систему збуту та розроблена концепція рекламного звернення.

ВИСНОВКИ

В процесі виконання даної роботи була побудована система збору та представлення даних про виконання автоматизованих тестових сценаріїв. Був проведений аналіз предметної області та встановлено актуальність проблеми, що обумовлена зростаючим ринком автоматизованого тестування та розробки програмного забезпечення та невідворотністю постання проблеми стабілізації автоматизованих тестових сценаріїв.

Були проаналізовані існуючі рішення, що представляють три групи програмного забезпечення – системи безперервної інтеграції, репортингове програмне забезпечення та статичні аналізатори програмного коду. Аналіз рішень для кожної із даних груп встановив, що рішення надають деякий функціонал, що може бути використаним в процесі стабілізації тестів для збору та відображення статистики. Однак дані продукти призначені для рішення сторонніх задач, а функціонал таких систем не є повноцінним для рішення поставленої задачі, користувацький інтерфейс не розрахований на такі потреби, що робить їх використання складним та незручним.

Були визначені сценарії використання системи та побудовано UML-діаграму для їх візуалізації. Встановлено, що із системою можуть взаємодіяти три актори – користувач, адміністратор та система автоматизації складання проекту. Для користувача надається функціонал отримання відповідної статистики за різними запитами. Адміністратору доступне конфігурація параметрів системи – модифікація адреси підключення до бази даних, встановлення шляху до звіту, тощо. Системі автоматизації складання надається функція анонсування оновленого актуального звіту для його зчитування системою.

Було детально розглянуто вибір джерела інформації про результати проходження тестових сценаріїв та визначення атрибутів звіту, що повертається системою користувачу. Із наявних варіантів, в якості джерела даних було обрано автоматично згенеровані звіти системами автоматизації складання. Дане джерело було визначено як найбільш стабільне та достовірне, легке у підтримці. Спираючись

на потреби користувача, були визначені базові та додаткові метрики, за якими система має фільтрувати та сортувати дані.

Було визначено технології для розробки системи, враховуючи її специфіку. Для цього було проведено аналіз доступних технологій та обрано найбільш зручні та швидкі у використанні. Деякі інструменти було обрано сугубу для полегшення та пришвидшення процесу розроблення. Всі обрані технології є доступними та безкоштовними.

Також було описано алгоритми роботи системи для двох користувацьких запитів, побудовано відповідні схеми алгоритмів. Для опису архітектури системи було побудовано структурну схему системи, де виділено шість блоків, які формують три основні модулі системи – модулі парсингу, роботи із базою даних та презентації. Реалізація кожного модулю детально описана, прийняті технічні рішення та використані шаблони проектування були обґрунтовані. Для кожного модулю також були побудовані діаграми класів, на яких відмічена взаємодія класів між собою всередині модулю.

Було проведене мануальне тестування системи із використанням підходу тестування за класами еквівалентності. Також створено інтеграційні автоматизовані тестові сценарії для кожного модулю.

Було розроблено стартап-проект для даної системи. Сформовано ідею проекту, проведено її техніко-економічний аудит, побудована ринкова стратегія та маркетингова програма. Було встановлено, що ідея є конкурентноспроможною, сформовано можливу реакцію на фактори можливостей та загроз.

В результаті, поставлене завдання на дану роботу виконане в повному обсязі. Архітектура створеної системи є гнучкою та сприяє підтримці системи та впровадженню нового функціоналу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «6 basic SLDC methodologies» [Електронний ресурс] // Режим доступу: https://www.roberthalf.com.au/blog/employers/6-basic-sdlc-methodologies-which-one-best
2. «Принципи тестування» [Електронний ресурс] // Режим доступу: https://studwood.ru/1586516/informatika/printsipi_testuvannya
3. «TeamCity» [Електронний ресурс] // Режим доступу: https://www.jetbrains.com/help/teamcity/teamcity-documentation.html#Copyright+and+Trademark+Notice
4. «Популярні інструменти для створення тест-кейсів» [Електронний ресурс] // Режим доступу: https://training.qatestlab.com/blog/helpful-materials/popular-tools-for-creating-test-cases/
5. «Sonarqube» [Електронний ресурс] // Режим доступу: https://www.sonarqube.org/enterprise-edition/?gads_campaign=SonarQube&gads_ad_group=SonarQube&gads_keyword=sonarqube&gclid=Cj0KCQiAw_H-BRD-ARIsALQE_2Old15VLMISfm4yk6nzoFyP1g4FFCDbCWy8Zeci1hG4vpR6TGNjvtIaAqXPEALw_wcB
6. «SOLID – принципи об'єктно-орієнтованого програмування» [Електронний ресурс] // Режим доступу: https://web-creator.ru/articles/solid
7. «Gradle vs Maven comparsion» [Електронний ресурс] // Режим доступу: https://gradle.org/maven-vs-gradle/
8. «Junit vs TestNg» [Електронний ресурс] // Режим доступу: https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/uchebnik-junit/11-junit-vs-testng
9. «Порівнюємо Java-бібліотеки для роботи із JSON: Json.simple, GSON, Jackson, JSONP» [Електронний ресурс] // Режим доступу: https://tproger.ru/translations/java-json-library-comparison/
10. «MongoDB» [Електронний ресурс] // Режим доступу:

https://docs.mongodb.com/manual/

11. «Aspect Oriented Programing with Spring» [Електронний ресурс] // Режим доступу: https://docs.spring.io/spring-framework/docs/2.5.x/reference/aop.html
12. Джошуа Блох «Java – ефективне програмування», третє видання
13. Jeanne Boyarsky, Scott Selikoff «OCP study guide»
14. «The testing pyramid» [Електронний ресурс] // Режим доступу: https://medium.com/front-end-in-regions-grodno/the-testing-pyramid-2fec1c1fef91
15. «Ступеневий аналіз конкуренції» [Електронний ресурс] // Режим доступу: https://studfile.net/preview/5992853/page:11/
16. «Модель п’яти сил конкуренції М.Портера» [Електронний ресурс] // Режим доступу: https://pidru4niki.com/12980108/marketing/analiz_konkurentiv
17. «SWOT-аналіз бізнес-проекту» [Електронний ресурс] // Режим доступу: http://homestartup.ru/pub_swot.html