Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Стешенко Анни Андріївни*
	(ПІБ)
академічної групи	*122М-21-1*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*«122 Комп'ютерні науки»*
	(назва освітньої програми)
на тему:	*Розробка програмного забезпечення для автоматизації зміни стану ресурсів тестування*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*проф. Мещеряков Л.І.*	98	відмінно	

Рецензент				

Нормоконтролер	*проф. Лактіонов І. С.*			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 22 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122М-21-1* *Стешенко Анні Андріївні*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка програмного забезпечення для автоматизації зміни стану ресурсів тестування*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 20.10.2022 р. № 2127 -л

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес автоматизації управління об'єктами для тестування за допомогою мови програмування Java.

Предмет досліджень – моделі, методи та інформаційні технології серверної та UI складових для забезпечення якості інструмента автоматизації зміни стану ресурсів тестування.

Мета НДР – підвищення ефективності, швидкості та складності сценаріїв для повного тестування інформаційних систем.

Вихідні дані для проведення роботи – теоретичні та експериментальні дослідження для вирішення задач проектування інформаційної системи застосунку та підходи до вибору технологій реалізації.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень полягає в розробці та застосуванні сучасних методів реалізації програмного забезпечення для удосконалення існуючих засобів зміни стану ресурсів тестування за рахунок збільшення ступеня автоматизації та безпеки використання ПЗ з прямим доступом до бази даних.

Практична цінність результатів полягає у тому, що отримані в ході дослідження результати роботи можуть застосовуватися як і для переведення ручного тестування інформаційної системи в більш авторматичний режим, так і у візуалізації стану наповнення бази даних для пришвидшення етапу перевірки програмного забезпезпечення через збільшення автономності інженера з тестування.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень повинні бути подані у вигляді, який дозволяє побачити та оцінити безпосереднє використання технологій серверної та UI сторін розробки. В результаті роботи повинен бути розроблений програмний комплекс для забезпечення автоматизації зміни стану ресурсів тестування.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	05.09.2022-24.09.2022
Збір, дослідження та систематизація інформації щодо проектування та реалізації безпечних функціональних застосунків на Java	25.09.2022-11.10.2022
Розробка і тестування автоматизованої системи для вирішення задачі забезпечення автоматизації управління об'єктами для тестування	12.10.2022-07.11.2022

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки адаптивності даної системи до процесу тестування програмного продукту будь-якої установи, таким чином зменшити затрати на заробітну плату людям, які виконують мануальне тестування.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки можливості вивчення та використання розробленої моделі ІС для вирішення задач проектування застосунків управління ресурсами тестування в корпоративних системах. Створена на нових технологіях архітектура інформаційної системи, що має стабільну та перевірену систему авторизації та безпеки, є ефективною та дозволить легко масштабувати та інтегрувати новий функціонал.

Завдання видав _____ *Мещеряков Л.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Стешенко А.А.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *05.09.2022 р.*

Термін подання кваліфікаційної роботи до ЕК *09.11.2022*

РЕФЕРАТ

Пояснювальна записка: 88 с., 30 рис., 3 дод., 52 джерела.

Об'єкт розробки: процес автоматизації управління об'єктами для тестування за допомогою мови програмування Java.

Предмет дослідження: методи, схеми та засоби автоматизації зміни стану ресурсів тестування.

Мета кваліфікаційної роботи: підвищення ефективності, швидкості та складності сценаріїв для повного тестування інформаційних систем.

Методи дослідження. Для виконання поставлених завдань були використані методи: об'єктно-орієнтоване програмування, криптографічні методи захисту інформації, система управління збереженням java об'єктів у таблиці реляційних баз даних, механізм шаблонів Java/HTML5.

Наукова новизна полягає в збільшенні рівня автоматизації та безпеки методів і засобів для зміни стану ресурсів тестування.

Практичне значення роботи. Отримані в ході дослідження результати роботи можуть застосовуватися як і для переведення ручного тестування інформаційної системи в більш авторматичний режим, так і у візуалізації стану наповнення бази даних для пришвидшення етапу перевірки програмного забезпезпечення через збільшення автономності інженера з тестування.

Список ключових слів: REST API, веб-застосунок, фреймворк, сервер, інтерфейс, Spring Framework, http-запит, http-відповідь, інформаційна система, база даних, криптографія, тестування в мануальному та автоматичному режимі.

ABSTRACT

Explanatory note: 88 pages, 30 pictures, 3 appendices, 52 sources.

Object of research: process of automating the management of objects for testing using the Java programming language.

Subject of research: methods, schemes and means of automation of changing the state of testing resources.

Purpose of Master's thesis: increasing the efficiency, speed and complexity of scenarios for full testing of the information system.

Research methods. The following methods were used to perform the tasks: object-oriented programming, cryptographic methods of information security, management system for storing java objects in relational database tables, Java/HTML5 template engine.

Originality of research is to increase the level of automation and security of methods and means for changing the state of testing resources.

Practical value of the results. The results obtained in the study can be used both for the transfer of manual testing of the information system in automatic mode, and in visualizing the state of the database to speed up the software testing phase by increasing the autonomy of the test engineer.

Keywords: REST API, web application, framework, server, interface, Spring Framework, http-request, http-response, information system, database, cryptography, manual and automatic testing.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

СУБД – система управління базою даних;

UI – user interface;

GUI – graphical user interface;

СУБД – система управління базами-даних;

SQL – structured query language;

ПК – персональний комп’ютер;

ОС – операційна система;

HTTP – hyper text transfer protocol;

API – application programming interface;

ПЗ – програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність дослідження. Поява перших комп'ютерів спричинила появу їх логічних супутників — програмних продуктів. Як і будь-яка людська діяльність, програмування не може існувати без помилок. На ранніх етапах розвитку сфери процес перевірки та налагодження програм через їхню достатню простоту не був важким, але в даний час проблема стала дуже актуальною.

Як відомо, не дивлячись на те як добре був реалізований програмний продукт спочатку, у процесі розробки всі програми зазнають змін. Вони можуть бути обумовлені необхідністю виправлення існуючих помилок, виявлених у процесі їх виконання, або бажанням внести до програми додаткові зміни. Також при розробці та тестуванні програмного забезпечення варто враховувати розширення областей застосування старих програм, що призводить до появи не врахованих раніше нових функціональних вимог.

Сутність вищевикладеного зводиться до того, що контроль за своєчасними змінами інформаційних систем набув статусу критично важливого чинника для збереження корисності програм, тобто фактора, що став за ступенем значущості домінуючим та привернув увагу фахівців даної сфери.

У зв'язку з постійно зростаючою конкуренцією в галузі інформаційних технологій на ринку програмного забезпечення, з'явилась низка продуктів, націлених на підвищення зручності та швидкості процесу. Проте, проведене дослідження виявило недостатню ефективність інформаційних методів, засобів і технологій автоматизації, що існують у цій галузі, особливо у застосуванні їх до проектів вузькоспеціалізованих областей.

У кваліфікаційній роботі на основі аналізу останніх перевірених тенденцій розвитку технологій та існуючих концепцій, засобів та методів повного тестування запропоновано новий розширений метод проведення тестування та створено необхідні засоби програмної підтримки. Розроблену програму відрізняє висока кастомізованість середовища тестування, універсальність у застосуванні

та відносно низька трудомісткість процесу впровадження.

Мета дослідження: Підвищення ефективності, швидкості та складності сценаріїв для повного тестування інформаційної системи.

Завдання дослідження: Для досягнення поставленої мети в роботі були сформульовані та вирішені наступні завдання:

1. Викласти принципи процесу перевірки програмного продукту;

2. Дослідити особливості реалізації та захисту програм тестування;

3. Проаналізувати доступні інструменти підключення до сховища даних та захисту комунікації, які доступні в IntelliJ IDEA;

4. Спроектувати та розробити відповідне програмне забезпечення, яке буде здатним до розгортання на різних конфігураціях ОС;

5. Перевірити ефективність системи та зробити висновки щодо доцільності її створення.

Об’єкт дослідження: Процес автоматизації управління об'єктами для тестування за допомогою мови програмування Java.

Предмет дослідження: Методи, схеми та засоби автоматизації зміни стану ресурсів тестування.

Методи дослідження. Для виконання поставлених завдань були використані наступні методи та інструменти: об'єктно-орієнтоване програмування, криптографічні методи захисту інформації, система управління збереженням java об'єктів у таблиці реляційних баз даних, механізм шаблонів Java/HTML5.

Наукова новизна полягає в удосконаленні методів та засобів автоматизації зміни стану ресурсів тестування.

Практичне значення. Проведене дослідження буде корисно для розробників, тестувальників, аспірантів та студентів, що спеціалізуються або цікавляться проблематикою повного та ефективного тестування інформаційних систем.

Результати дослідження можуть застосовуватися як в практиці розробки

програмного забезпечення, так і поглиблення рівня самостійного опанування галузі.

Особистий внесок автора:

1. Наукові результати роботи отримані автором самостійно.

2. Вибір методів досліджень і технологій реалізації;

3. Розробка теоретичної частини роботи з дослідження і систематизування знань про існуючі підходи до зміни стану об'єктів в процесі тестування;

4.Реалізація засобів програмної підтримки для автоматизації зміни стану ресурсів тестування;

5. Оцінка отриманих результатів.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить 88 сторінок, в тому числі 68 сторінок тексту основної частини з 30 рисунками, списку використаних джерел з 52 найменуваннями на 4 сторінках, 3 додатка на 20 сторінках.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Тестування в життєвому циклі програмного забезпечення

Момент від появи ідеї створення деякого програмного продукту до моменту завершення його підтримки називають життєвим циклом програмного забезпечення. Зручншіше всього уявляти його структуру, що містить процеси дії та завдання, що здійснюються в ході розробки, використання та супроводу програмного продукту, у вигляді моделей.

Такі моделі можна узагальнено розділити на 3 групи:

1. Інженерний підхід
2. З урахуванням специфіки задачі
3. Сучасні технології швидкої розробки

Одним з перших стандартів для розробки програм стала каскадна модель життєвого циклу, також звана моделлю «waterfall» (рис. 1.1). Вона була розроблена ще у 1980-х роках і характеризується тим, що етапи суворо послідовні та перехід між ними необоротний.

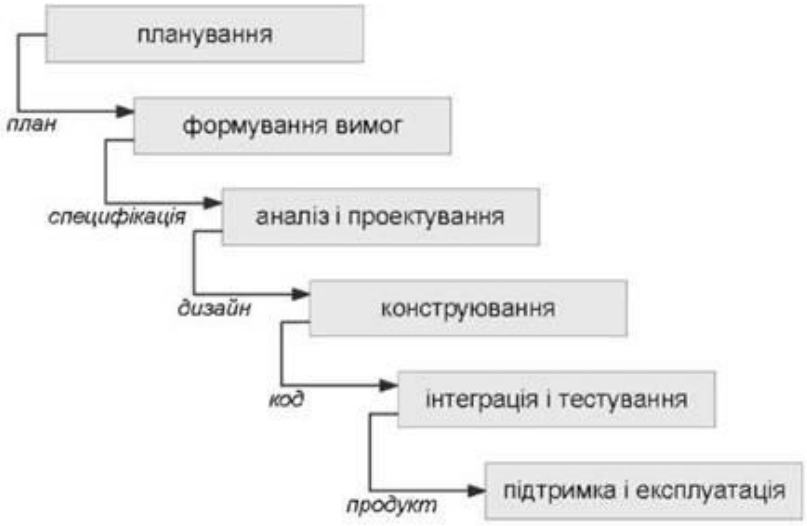

Рис. 1.1. Водоспадна модель життєвого циклу програмного забезпечення

Модель «водоспаду» була застосована однією з перших, і головним її фактором є можливість переходу до наступного етапу тільки після завершення попереднього. Наприклад, після початку тестування вже неможливо повернутися до розробки, якщо з'явилися нові функціональні вимоги. Це дає головний плюс підходу – чітке планування термінів та вартості кожного етапу. Однак набагато відчутнішим є мінус нереалістичності даної моделі, коли на практиці розробка системи майже ніколи не проходить строго відповідно до негнучкої, заздалегідь продуманої схеми. Належить до першої групи моделей.

Удосконаленою версією класичної каскадної моделі стала V-модель, що є розробкою через тестування (рис 1.2).

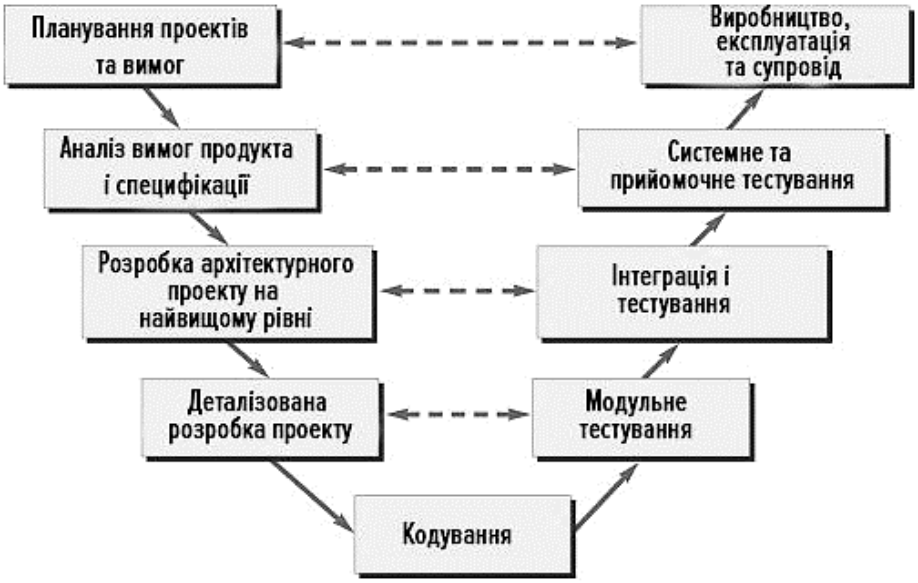

Рис. 1.2. V-подібна модель життєвого циклу програмного забезпечення

У моделі процес розробки представлений низхідною послідовністю у лівій частині умовної літери V, а стадії тестування – її правому ребрі. Відповідність етапів розробки та тестування показано горизонтальними лініями.

На кожному етапі відбувається контроль поточного процесу із метою переконатися у можливості переходу на наступний рівень. Тестування починається зі стадії написання вимог, і для кожного з етапів передбачений свій рівень тестового покриття. Окремий тест-план із зазначенням критеріїв входу та

виходу розробляється під кожен рівень тестування, а отже під час роботи над поточним рівнем також опрацьовується стратегія тестування наступного.

Таким чином, V-модель має більш наближений до сучасних методів алгоритм, але все ще містить ряд недоліків. Її використання актуальне в проектах, які передбачають ширше тестове покриття. Належить до основних практик екстремального програмування та третьої групи моделей.

При комбінації моделі водоспаду та каскадної вийшла спіральна модель – розробка програмного забезпечення з урахуванням ризиків. Кожна фаза спіральної моделі у розробці програмного забезпечення починається з визначення мети проектування та закінчується переглядом прогресу (рис. 1.3).

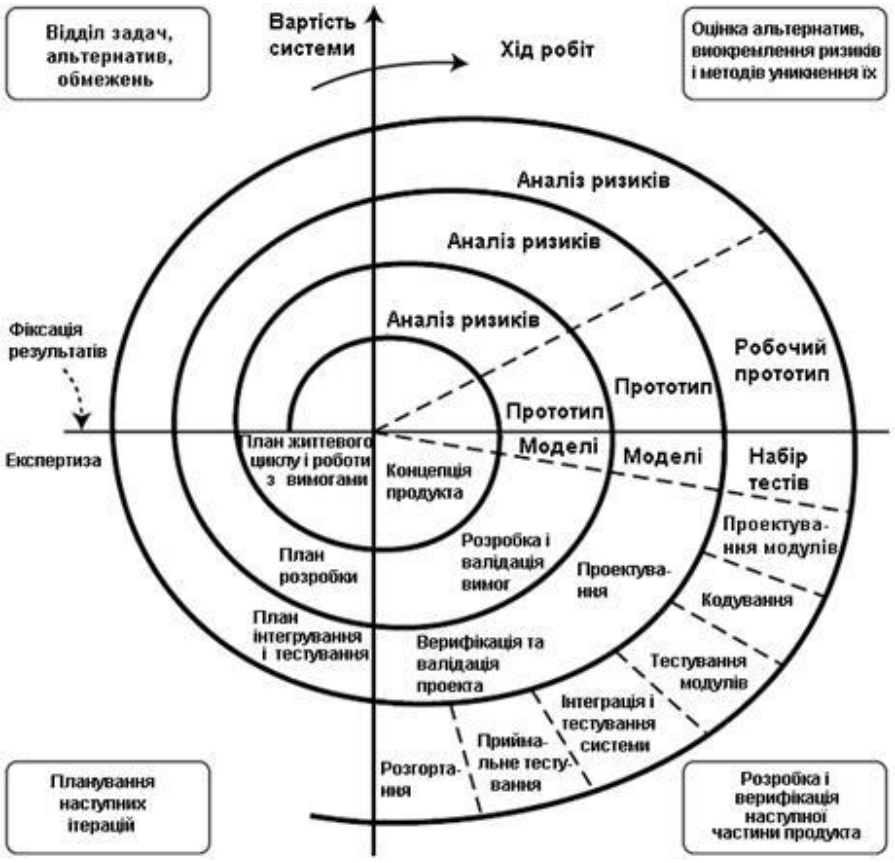

Рис. 1.3. Спіральна модель життєвого циклу програмного забезпечення

Дана модель належить до третьої групи та має переваги у швидкому отриманні результату, підвищенню конкурентоспроможності та змінюваності.

Однією з найбільш гнучких моделей розробки програмного забезпечення є Rational Unified Process (RUP). Її життєвий цикл складається з чотирьох фаз: початку, розвитку, конструювання та передачі. Кожна з фаз має обов'язкові цілі. Крім того, команди розробки та тестування за необхідності можуть проводити додаткові ітерації доти, доки вимоги не будуть задоволені. У середньому, повноцінне створення програмного забезпечення складається з 10-15 ітерацій, кожна з яких займає від 2 до 6 тижнів.

Більшість видів тестування із залученням самих різних ресурсів відбувається під час етапу конструювання. Команда розробників розпочинає безпосереднє створення архітектури системи, а фахівці із забезпечення якості та тестувальники вивчають функціональність рішення та особливості інтерфейсу. Розробка відлагодженого та перевіреного операційного програмного забезпечення – кінцева мета етапу конструювання.

Також не можна не сказати про методологію scrum – це ітеративно-інкрементальна технологія. Поняття «ітеративна» означає, що технологія розбивається на рівні за тривалістю проміжки часу – спринти, які тривають від одного до чотирьох тижнів. А слово «інкрементальна» – що в результаті ітерації виходить новий, потенційно робочий продукт, який вирішує бізнес-проблему.

Однак для процесу тестування ця модель має вагомі мінуси. Насамперед часто ТЗ не складають через його швидке старіння і, отже, неможливість одночасно скласти й актуальний набір автоматизованих тест-кейсів. Якщо програма взаємодіє з іншими додатками чи сервісами, складність його тестування та управління ресурсами підвищується.

Зі сказаного раніше випливає, що тестування програмного забезпечення – це обов'язковий процес за будь-якої моделі розробки продукту. Саме воно як справжнє розслідування надає зацікавленим сторонам інформацію про якість програмного продукту або послуги, що тестується. Воно також допомагає отримати об'єктивне, незалежне уявлення про програмне забезпечення, щоб дозволити бізнесу оцінити та зрозуміти ризики його впровадження.

Методи випробувань включають процес виконання програми за допомогою різноманітних ресурсів та зміни їх стану за описаними сценаріями з метою виявлення програмних помилок та перевірки того, що програмний продукт придатний для використання.

1.2 Актуальність мануального тестування

Мануальне тестування можна розглядати як взаємодію професійного тестувальника і ПЗ з метою пошуку помилок і дефектів. Взаємодіючи з додатком безпосередньо, тестувальник може порівнювати очікуваний результат із реальним та залишати рекомендації.

Особливо актуальним є ручне тестування для невеликих проектів з великою кількістю функціоналу. Насамперед тому, що автоматизація — це дорого. Окрім великих часових та фінансових витрат на автотести на етапі створення, вони вимагають і значних ресурсів для підтримки актуальності після кожної зміни програми. Такі витрати повинні виправдовуватися та окупатися, тому не завжди є виправданими.

Крім того, автоматизація досить складно впроваджується. На відміну від класичного ручного тестування, яке завжди буде популярним через свою простоту і покриття будь-якого користувача сценарію.

Автоматизоване тестування вбирає в себе лише виконання монотонних та однотипних завдань, оскільки автотести можна застосувати лише у частинах ПЗ, де відомий очікуваний результат і ресурси тестування залишаються постійними.

Автотести не призначені для «нових і незвіданих територій», нестандартно і хаотично діяти це прерогатива людини. Усе сказане вище підводить до висновку, що автоматизоване тестування призначене лише для частини від загального обсягу роботи на етапі перевірки ПЗ, але звільняє багато ресурсів за рахунок скорочення часу на просту перевірку працездатності продукту.

Саме мануальний тестувальник визначає доцільність автотестів, розподіляючи при цьому де вигідно автоматизувати, а де дешевше та ефективніше пройти кейс вручну.

Можемо дійти висновку, що дисципліна тестування проходить етап стрімкого розвитку і класичне мануальне тестування ще трансформується, роблячи своє місце у сфері вагомішим.

1.3 Існуючі підходи до управління ресурсами

Особливістю ручного тестування є труднодоступність і довготривалість створення точних ресурсів з метою відтворення сценарію дій користувача. Саме на цьому етапі витрачається неоправдано багато сил і знижується ефективність роботи команди тестування. Найпростішим і найпоширенішим способом для створення потрібного об'єкта тестування є складання документа про таку необхідність, який згодом затверджується менеджером, а потім внесення необхідних змін до бази даних розробниками відповідно до заведеної для них задачі. Автоматизація даного шляху значно покращила б якість та швидкість етапу тестування програмного продукту.

Популярним рішенням можна назвати DataGrip від JetBrains (рис. 1.4). Це середовище керування базами даних для розробників або тестувальників, що призначенене для запитів, створення та керування базами даних. Бази даних можуть працювати локально, на сервері або в хмарі. Підтримує MySQL, PostgreSQL, Microsoft SQL Server, Oracle тощо.

Схожою за функціоналом є Aqua Data Studio (рис. 1.5) — універсальне інтегроване середовище розробки (IDE) для баз даних і візуальної аналітики. Вона дозволяє розробникам баз даних, адміністраторам баз даних, а також розробникам і тестувальникам працювати зі багатоплатформенними базами даних і даними, що містяться в них.

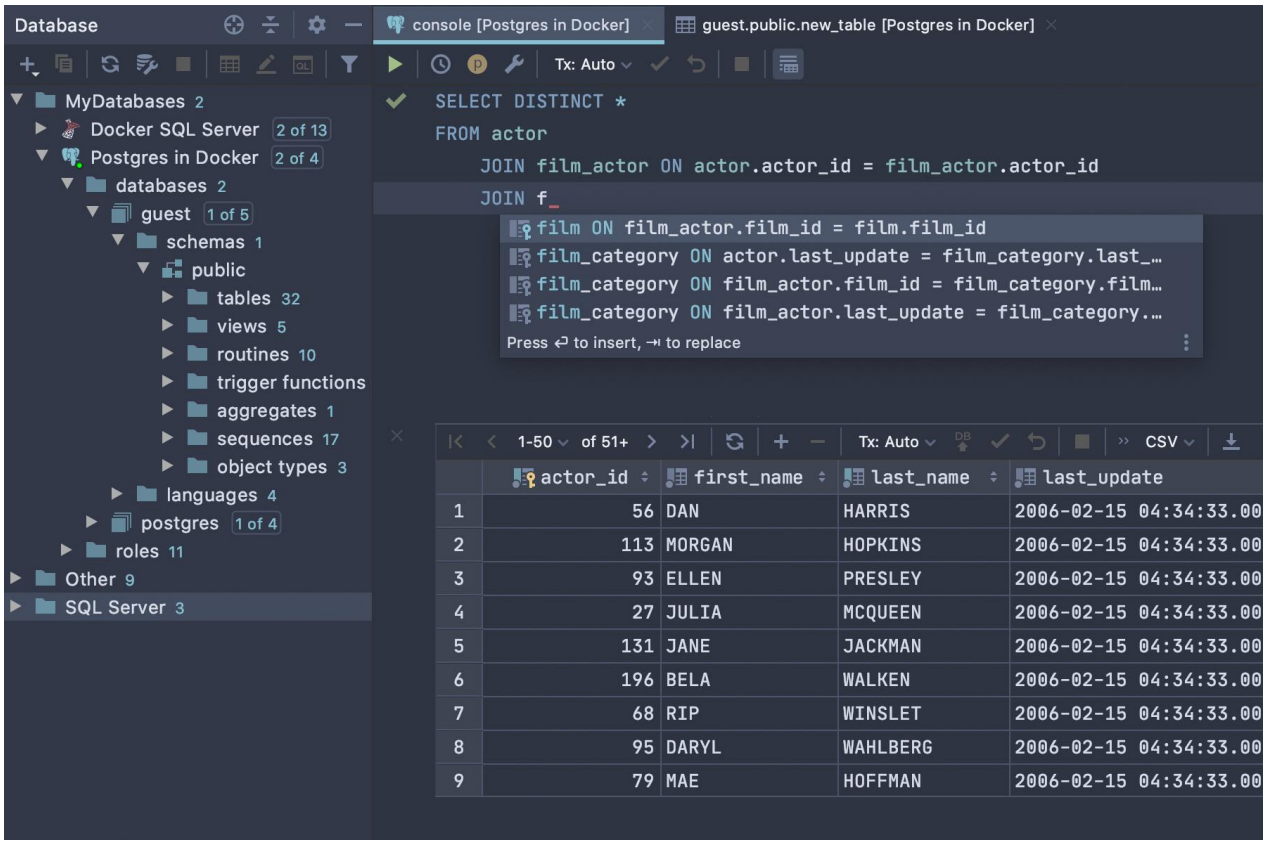

Рис. 1.4. Інтерфейс DataGrip

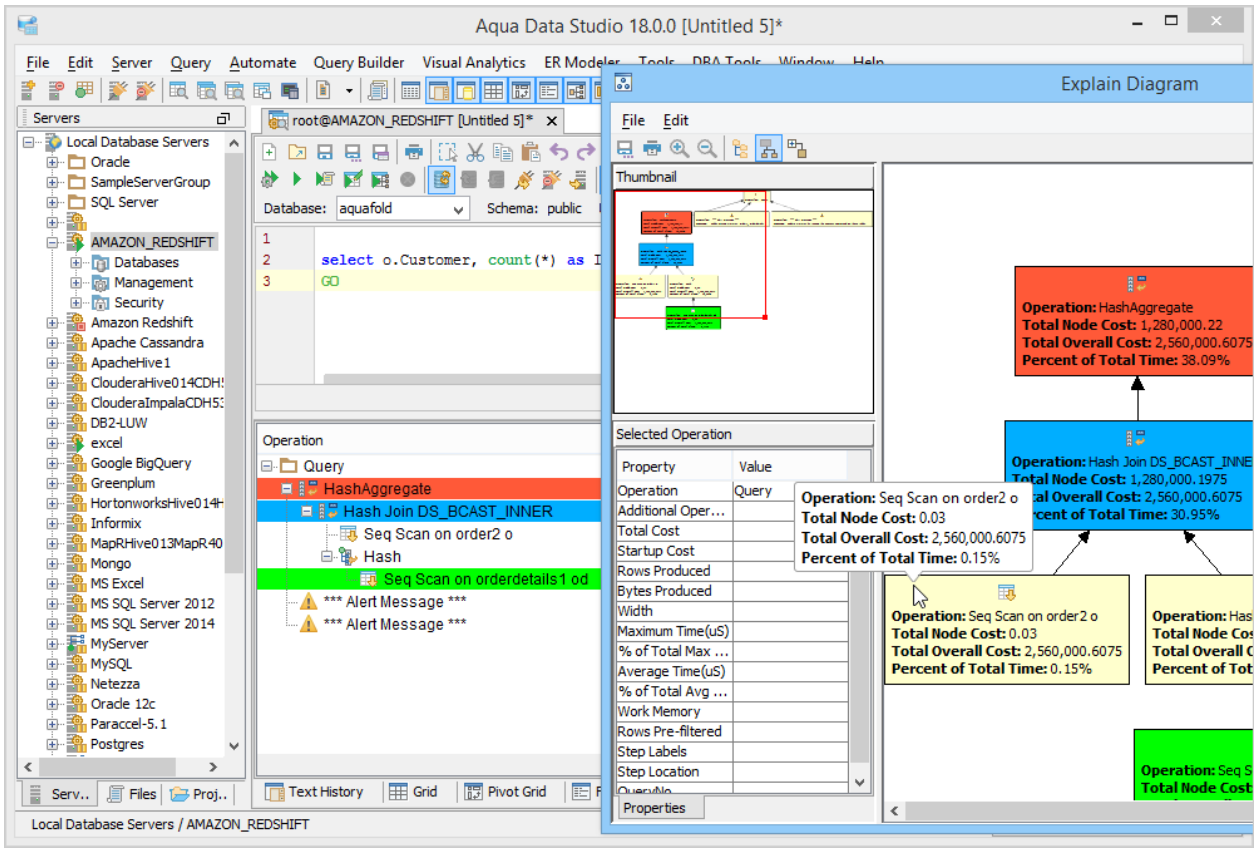

Рис. 1.5. Інтерфейс Aqua Data Studio

Перевагою Aqua Studio є чудовий функціонал з планування виконання SQL скриптів через певні проміжки часу, визначені користувачем в ОС Windows, та можливість формування цінних візуальних звітів для аналітика.

DataGrip використовує концепції ролей користувачів для керування дозволами у своїх базах даних, проте не підтримує заплановані завдання.

Спільним обтяжуючим фактором подібних середовищ керування базами даних є необхідність користувача знати спеціальну мову SQL, яка використовується в програмуванні та призначена для маніпулювання даними в системах керування реляційною БД, для управління ресурсами тестування. Такий підхід збільшує список вимог до тестувальника та людський фактор помилок при перевірці програмного забезпечення.

1.4 Порівняльний аналіз засобів досягнення ефективності, масштабованості та швидкості роботи додатка

Архітектура програмного забезпечення – дуже важлива тема програмної інженерії останніх років. На практиці реалізація додатків із чистою архітектурою часто викликає значні труднощі.

Почнемо з аналізу схеми архітектури додатку (рис 1.6). На рисунку є чотири кола, кожен з яких представляє окремий рівень ПЗ, але потрібно пам’ятати, що вони лише схематичні, і залежно від архітектурного рішення їх кількість може збільшитись або зменшитись.

Схема містить наступні атрибути:

- об'єкти: це найстабільніша частина коду програми, яка не повинна зазнавати зовнішніх змін, наприклад, методи або структури даних;
- сценарії використання: зона для запровадження інкапсуляції та впровадження бізнес-логіки;
- адаптери інтерфейсу: в цій частині відбувається перетворення та подання даних у сценарії використання;

- фреймворки та драйвери: ця область містить фреймворки та інструменти для запуску програми.

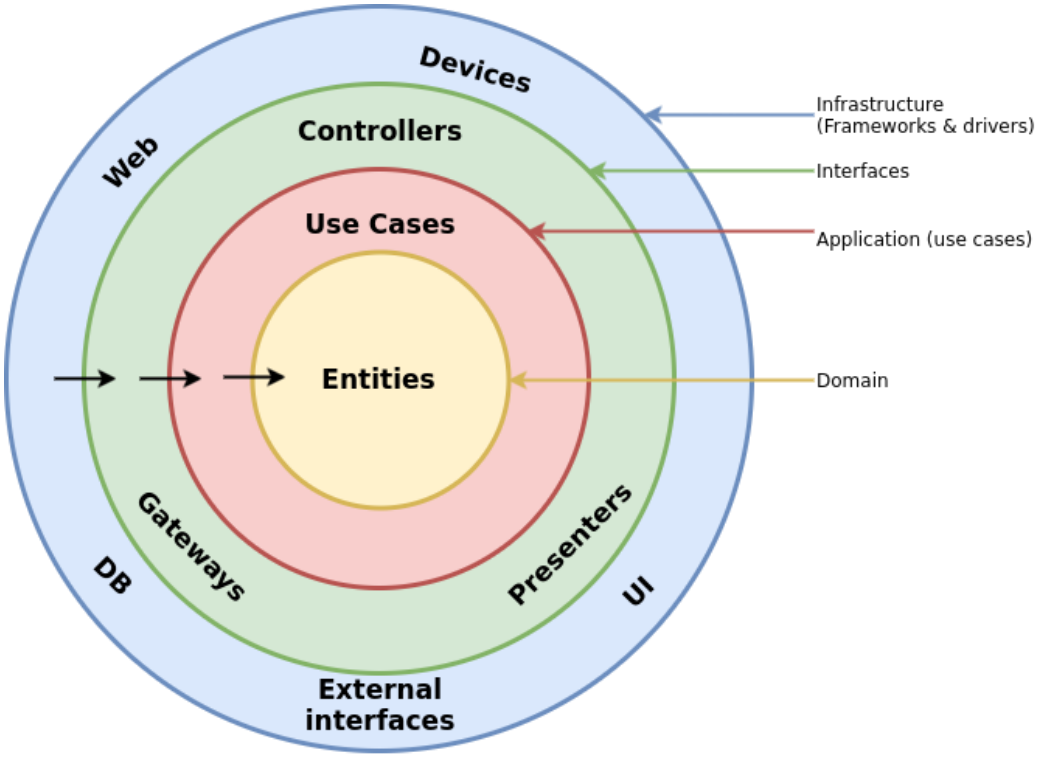

Рис. 1.6. Схема для "чистої" архітекури

Фундаментальним правилом роботи даної архітектури є «Правило залежності» (відображено стрілками на зображенні). Залежність є саме внутрішньою, а не зовнішньою, це означає, що зовнішні кола знають про внутрішні та взаємодіють з ними, але внутрішні не знають про зовнішні. Для того щоб відокремити клієнтський код від об’єктів, які потребують змін, використовується ін’єкція залежностей.

Коли справа доходить до тестування, дотримання принципу інверсії залежностей також є великою підмогою, оскільки це дозволяє залежати від абстракцій, а не реалізацій. А отже, макети можна надавати тестам подвійно.

Для реалізації інверсії залежностей існує декілька популяних технологій, наприклад Java EE та Spring. Java EE – це платформа з надання API та середовища виконання для розробки та запуску великомасштабних, масштабованих, багаторівневих, надійних та безпечних програм.

У додатків Java EE є структура, що має дві основні якості:

- багаторівневість. У Java EE програми розділені за функціональним принципом на ізольовані модулі;
- вкладеність. Усередині Java EE сервера розміщуються контейнери компонентів.

Поділ ПЗ за рівнями є загальновідомим:

- клієнтський;
- середній рівень;
- рівень доступу до даних.

На клієнтському рівні програма запитує дані у Java EE сервера, який після цього обробляє запит клієнта та повертає йому відповідь. Наприклад, клієнтським додатком може бути браузер.

Середній рівень ділиться на web та бізнес-логіку. На веб-рівні за допомогою технологій: Java Server Faces technology (JSF), Java Server Pages (JSP), Expression Language (EL), Servlets та Contexts and Dependency Injection for Java EE (CDI) забезпечується взаємодія між клінтами та рівнем бізнес-логіки.

Рівень бізнес-логіки можна вважати ядром всієї системи і складається він з наступних компонентів: Enterprise JavaBeans, JAX-RS RESTful web services, Java Persistence API entities та Java Message Service. В них реалізована вся бізнес-логіка програми, яка забезпечує функції для покриття потреб певної конкретної сфери бізнесу (наприклад електронна комерція або банківська справа).

Рівень доступу до даних іноді називають рівнем корпоративних інформаційних систем (Enterprise Information Systems – EIS). Він складається з різних серверів баз даних, систем планування ресурсів (Enterprise Resource Planning – ERP) та інших джерел даних. Рівень бізнес-логіки використовує звернення до цього рівня за інформацією про дані з БД.

Технології рівня EIS: Java Database Connectivity API (JDBC), Java Persistence API, Java EE Connector Architecture, Java Transaction API (JTA).

На рисунку 1.7 наведено схему з найпростішим Java EE додатком.

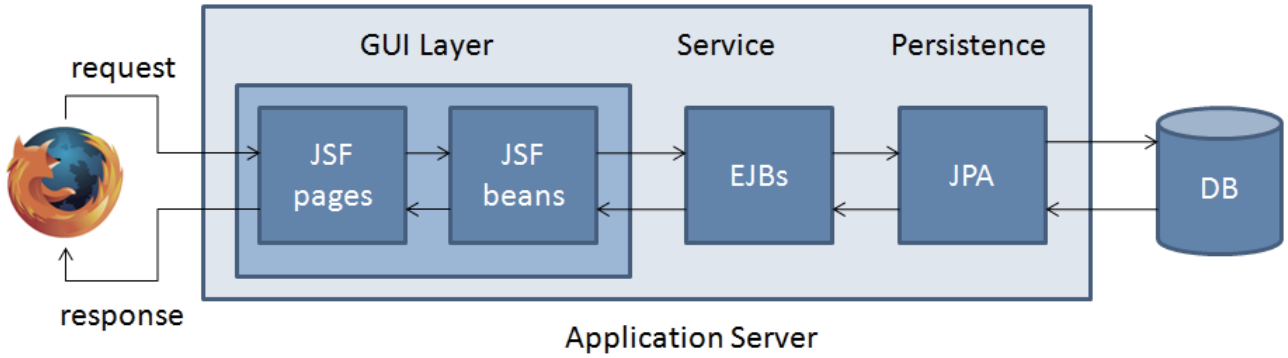

Рис. 1.7. Java EE додаток

Спочатку запит обробляється JSF сторінкою, а потім JSF біном. Далі управління йде на бізнес-логіку в EJB, яка працює з базою даних через JPA.

Головною перевагою є властивість масштабованості, тобто якщо навантаження на цей додаток збільшиться на кілька порядків, його схема не зміниться. Додаток треба буде встановити на кластер з кількох потужних нод, і все працюватиме в кластерному оточенні без жодних правок. Слабкими сторонами Java EE є досить складне середовище розробки додатків та остаточна вартість проекту, яка включаючи розробку, розгортання та тестування програми може виявитися непомірно високою.

Конкурентним Java EE вважається Spring Framework.

Розвиток даних двох платформ був досить цікавим. У Java EE перші версії було створено за участю IBM. Вони вийшли дуже корисними та функціональними, але досить великоваговими та не дуже зручними у використанні. Давався взнаки той факт, що розробникам було важко підтримувати велику кількість конфігураційних файлів. Spring IoC пройшов реліз у той же час. Це була невелика, зручна при впровадженні та приємна у користуванні бібліотека. Конфігураційний файл у ній також використовувався, але на відміну Java EE, він був один. Таким чином простота Spring призвела до того, що практично всі почали використовувати цей фреймворк у своїх проектах.

Spring відомий своїми системами впровадження залежностей та інверсії управління (IoC). Вони дозволяють з легкістю створювати великомасштабні та

слабо пов'язані між собою додатки за рахунок використання IoC та AOP. Фреймворк Spring особливо підходить для фінансових та корпоративних програм завдяки своїй безпеці, швидкості та легкості побудови транзакційних систем. На даний момент головними роботодавцями розробників Spring є різноманітні банки, eBay, Visa та інші корпоративні гіганти.

У порівнянні з Java EE, простий додаток на Spring не сильно відрізняється за своїм принципом роботи (рис. 1.8).

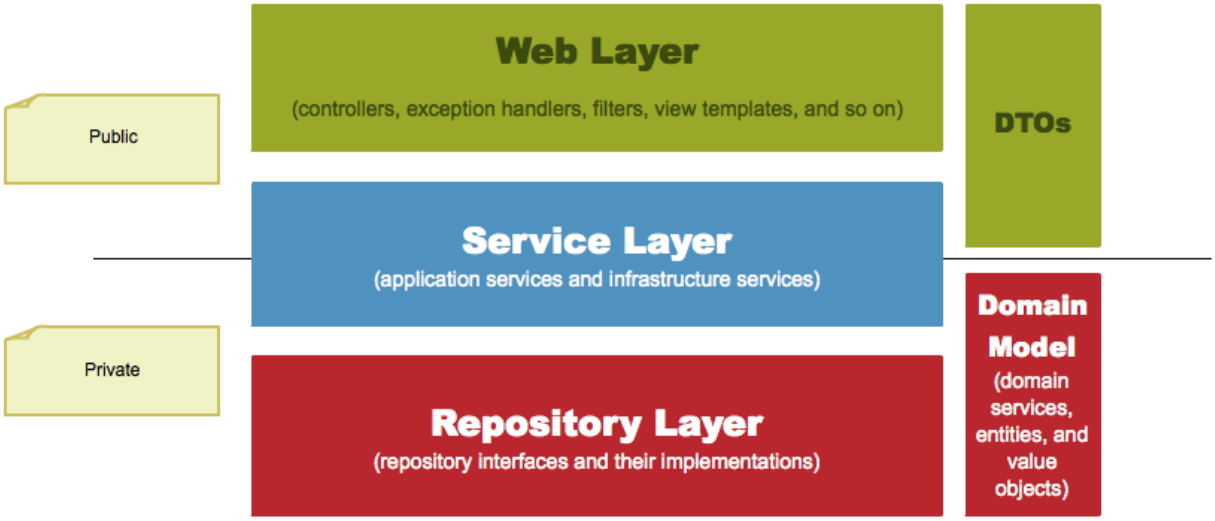

Рис. 1.8. Spring додаток

Spring спрощує процес розробки програми за рахунок консолідації системи залежностей та зниження навантаження на окремі класи. Ця технологія називається інверсія управління, вона перекладає відповідальність за залежності від об'єктів сам фреймворк. В IoC (інверсії управління) об'єкт вказує на необхідність певної залежності, і платформа налаштовує та керує нею автоматично. Процес, що Spring використовує для автоматичного управління залежностями bean-компонентів називається впровадження залежностей. В ході нього Spring перевіряє, які саме складові bean-компоненти необхідні для роботи обраного розробником bean-а, і впроваджує їх автоматично як залежність.

Особливістю фреймворку Spring є підтримка веб-архітектури MVC, яка розділяє функціональні можливості між рівнями моделі, представлення та

контролера. Приємним бонусом стала обробка помилок, що включає обробку винятків JDBC із системою їх ієрархії.

Високий рівень популярності обумовлений великою кількістю переваг Spring:

- легкість: Spring використовує звичайні старі об'єкти Java (POJO), а не сервери або корпоративні контейнери;
- ліцензія з відкритим кодом;
- модульність: IoC та MVC дозволяють повторно використовувати компоненти у додатку без ручного керування залежностями кожного з них;
- сильна підтримка екосистеми Java: Spring новаторськи використовує вже існуючі технології, такі як ORM-фреймворки, JEE та таймери JDK;
- масштабування транзакцій: Spring пропонує послідовний, масштабований інтерфейс управління транзакціями як для локальних і глобальних транзакцій;
- безпека: простота впровадження готових модулів безпеки з функціями автентифікації;
- гнучкі конфігурації: є можливість використовувати інструкції на основі Java або конфігурацію XML;
- простота: Spring Boot сильно спрощує первинне налаштування програми, а також код програми Spring простий для тестування.

До мінусів можна віднести недостатність документації та паралельні механізми. Широкий спектр можливостей Spring означає, що завдання можна виконати декількома способами. Отже для вибору ідеального рішення необхідно досконале знання доступних інструментів, щоб уникнути плутанини між командами. При цьому в документації Spring відсутні чіткі рекомендації з низки тем, особливо щодо методів кібербезпеки.

У результаті проведеного аналізу, можемо зробити висновок, що за основним функціоналом Spring і Java EE поділяють паритет та є найкращими інструментами для побудови корпоративних мережевих додатків на мові Java.

Однак при виборі технології необхідно враховувати, що Java EE може

працювати лише в рамках Enterprise Application Server, до яких не належить популярний Tomcat. У той час як додаток побудований на технологіях фреймворку Spring може працювати на будь-якому сервері, і навіть взагалі без нього, адже може запустити сервер у собі самостійно.

Описаний функціонал робить Spring найкращим інструментом для розробки невеликих програм з GUI на Front-end або для мікросервісної архітектури, коли Java EE навпаки добре підходить для реалізації масштабованого монолітного кластерного додатку.

Можемо зробити висновок, що для побудови відносно невеликого додатку з автоматизації зміни стану ресурсів тестування Spring є найкращим варіантом. Крім того, додаток на Spring буде легше у підтримці на перспективі, адже дана технологія є більш популярною на ринку розробки ПО, тому більшість розробників працюють саме з нею. І також є хороший шанс, що через п’ять років він буде активно підтримуватися, чого не можна сказати про інші фреймворки.

1.5 Аутентифікація та авторизація

Існування будь-якого веб-додатоку практично неможливио без авторизації та аутентифікації, оскільки скрізь потрібно авторизуватися. У випадку програмного продукту з прямим доступом до цінної та захищеної бази даних цей крок стає в рази важливішим.

Для забезпечення безпеки створюваного додатку зі стеком технологій Spring буде актуальним використання Spring Security – це платформа для додатків Java, яка надає спеціалізовані механізми для побудови систем авторизації та аутентифікації. Вона являє собою систему, що налаштовується для перевірки автентичності та контролю доступу до ресурсів додатків, а також захисту корпоративних додатків, створених за допомогою Spring (рис. 1.9).

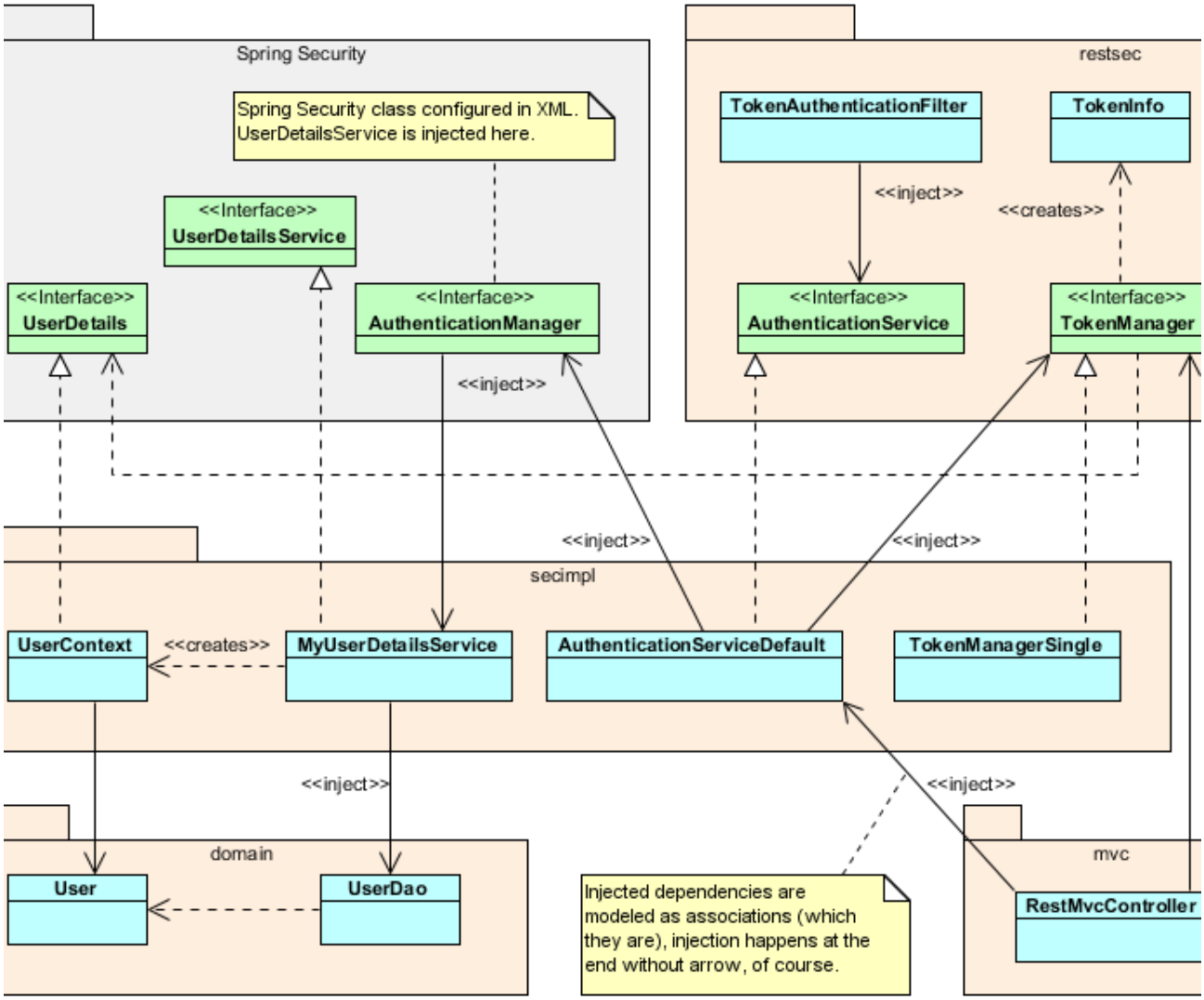

Рис. 1.9. Схематичне зображення реалізації Spring Security

Ключовим об'єктом є SecurityContextHolder, він зберігає інформацію про поточний контекст безпеки певної програми, включаючи приватну інформацію про самого користувача, який працює зараз із додатком. Усередині SecurityContext міститься об'єкт Authentication. Spring Security використовує цей об'єкт для надання інформації системи безпеки, яка пов'язана із запитами користувача. Сервіс UserDetails є представленням самого користувача, але в більш розширеному вигляді та з урахуванням специфіки програми Spring Security. У разі успішної аутентифікації, UserDetails використовується для створення Authentication об'єкта, який зберігається в SecurityContextHolder.

Широкі можливості авторизації в Spring Security є однією з найбільш

вагомих причин її популярності. Розподіл програми на частини дозволяє відокремити доступ до певних зон окремим користувачам через функціонал GrantedAuthority. Масив об'єктів GrantedAuthority складають певні повноваження, які надаються користувачу, наприклад роль адміністратора або тестувальника, які пізніше налаштовуються для веб-авторизації, а також авторизації методів. Даний розподіл зон доступу стає можливим завдяки широкій області видимості для дозволів, в яку входить вся програма.

Для створення безпечного додатка з автоматизації зміни стану ресурсів тестування знадобиться впровадження управління доступом до програми на основі ролей і привілеїв користувачів.

1.6 Висновки

Автоматизоване тестування звільнило час від точкової і регулярної превірки технічного стану продукту, підвищив цінність роботи з масштабними процесами, такими як імплементування контролю якості на всіх рівнях управління проектом та найбільш творчій і складній частині розробки ПЗ – тестування нового функціоналу та пошуку дефектів при повторенні проблемних сценаріїв користувачів.

Для мануального тестування постає проблема підготовки ресурсів та періодичної зміни їх стану задля досягнення відповідності тестових умов до реальної послідовності дій користувача та оновлення його даних у БД додатка.

В існуючих програмах із створення запитів та керування базами даних було виявлено ряд пунктів, які мають потенціал до вдосконалення та підвищення ефективності процесу тестування, наприклад створення функціоналу зміни даних таблиць за допомогою зрозумілого графічного інтерфейсу без необхідності створення SQL-скриптів. Для впровадження оновлення інформації необхідно зробити декілька видів таймерів під різні потреби тестування – відкладений запуск змін та їх застосування у визначений час, дату тощо.

Розробка за допомогою фреймворків дає змогу отримати швидший,

якісніший та економічно кращий результат, ніж створення проекту без використання будь-яких платформ. Широкий вибір готових реалізацій найуживаніших функціональних можливостей скорочує витрати часу і грошей на розробку ПЗ, і також допомагає досягти вищої стабільності програми за рахунок гарного рівня тестового покриття та постійної підтримки рівня захисту від нових загроз. В процесі порівняльного аналізу доступних технологій для створення додатку було обрано Spring Framework, що було обумовлено наявністю великої кодової бази, розвиненою спільнотою розробників, гарною продуктивністю і тим, що Spring показав себе як зручна та продуктивна технологія для створення складних бекендів – досвід Amazon, Google, Udemy, Trivago та інших успішних компаній це підтверджує.

Для забезпечення належного рівня захисту системи та запровадження рольової схеми доступу було обрано Spring Security, що є потужною структурою автентифікації та контролю доступу, яка легко налаштовується під конкретний проект. Важливою особливістю Spring Security є надання захисту від атак, таких як фіксація сесії, викрадення кліків, підробка міжсайтових запитів тощо. Де-факто це стандарт для захисту програм на основі Spring Framework, який зосереджений на забезпеченні як автентифікації, так і авторизації програм Java.

РОЗДІЛ 2
ЗБІР, СИСТЕМАТИЗАЦІЯ ТА ДОСЛІДЖЕННЯ ІНФОРМАЦІЇ ПРО СУЧАСНІ МОДЕЛІ ІНФОРМАЦІЙНИХ СИСТЕМ НА БАЗІ СУКУПНОСТІ ТЕХНОЛОГІЙ SPRING

2.1 Основи побудови IC у Spring Framework

До появи Enterprise Java Beans (EJB) для створення веб-додатків розробникам Java потрібно було використовувати JavaBeans. Хоча JavaBeans допомагали в розробці компонентів користувальницького інтерфейсу (UI), вони не могли забезпечити управління транзакціями та безпеку, що є вкрай необхідним для розробки надійних і стабільних корпоративних систем. Поява EJB розглядалась як вирішення цієї проблеми, вона розширила веб і корпоративні компоненти Java. Проте розробка додатку за допомогою EJB залишалась складною, оскільки розробнику необхідно було виконувати надлишкові завдання, такі як створення інтерфейсів Home і Remote та реалізацію методів зворотного виклику життєвого циклу.

Фреймворк Spring з'явився як рішення даних проблем. Для розробки корпоративних додатків він використовує різномантіні нові методи, такі як Аспектно-орієнтоване програмування (AOP), Plain Old Java Object (POJO) та ін'єкція залежностей (DI), тим самим усуваючи складності, пов'язані з розробкою ПЗ з використанням EJB. Spring зосереджений на наданні способів керування бізнес-об'єктами, чим значно спростив розробку веб-додатків у порівнянні з класичними фреймворками Java та інтерфейсами прикладного програмування (API), такими як Java Server Pages, підключення до баз даних Java та Java Servlet.

Фреймворк Spring зручно розглядати як набір підфреймворків, які також називаються шарами, таких як Spring Web MVC, Spring AOP, Spring Object-Relational Mapping (Spring ORM) та Spring Web Flow тощо. При створенні веб-додатку є можливість використання будь-якого з цих модулів окремо, а також

модулі також можуть бути згруповані разом, щоб забезпечити кращу функціональність веб-додатку. Унікальним Spring роблять такі особливості фреймворку, як IoC, AOP та управління транзакціями.

При створенні ПЗ по зміні управління стану ресурсів будуть використовуватись наступні функціональні можливості фреймворку:

- контейнер IoC;

Spring надає два пакети, а саме org.springframework.beans та org.springframework.context, які допомагають у забезпеченні функціональності IoC. Він відноситься до основного контейнера, який використовує патерн DI або IoC для неявного надання посилання на об'єкт в класі під час виконання.

- фреймворк доступу до даних;

Дозволяє розробникам використовувати API персистентності, такі як JDBC та Hibernate, для зберігання персистентних даних в БД. Це допоможе у вирішенні різних проблем розробки: взаємодія з підключенням до бази даних, переконання у закритті з'єднання, робота з з винятками і реалізація управління транзакціями. Загалом дана частина Spring дозволить легко писати код для доступу до персистентних даних у всьому додатку.

- рівень абстракції JDBC;

Допомагає в обробці помилок у ефективній і простій формі.

- фреймворк Spring MVC;

Дозволяє створювати додатки на основі архітектури MVC, тобто всі запити від користувача спочатку проходять через контролер, а потім відправляються у візуальну обробку, такі як JSP-сторінки або сервлети.

- Spring Web Service;

Генерує кінцеві контексти та визначення вебсервісів на основі класів Java.

- Spring Security;

Цей фреймворк забезпечить різні функції безпеки, такі як: автентифікація та авторизація для створення захищеної програми Java Enterprise.

2.2 Архітектура Spring Framework

Фреймворк Spring складається з семи модулів, які показані на рисунку 2.1.

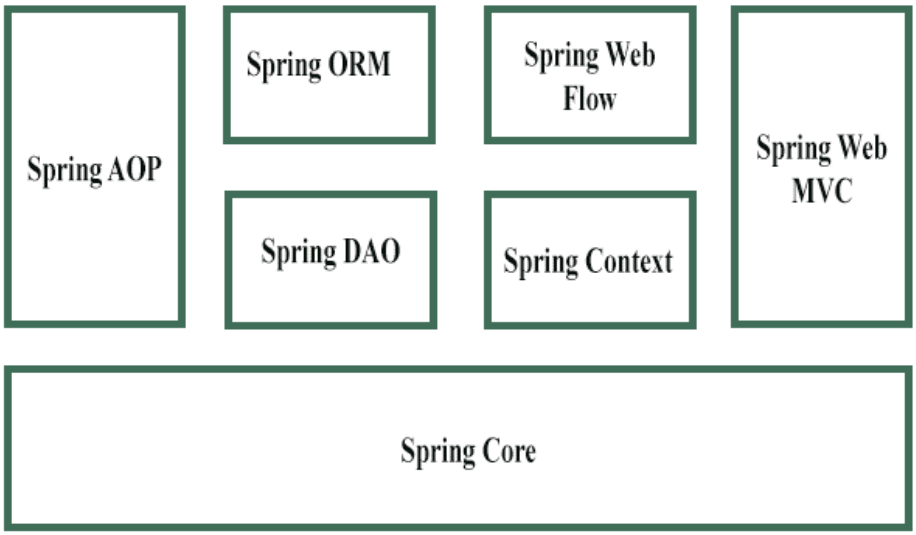

Рис. 2.1. Модулі Spring Framework

Це Spring Core, Spring AOP, Spring Web MVC, Spring DAO, Spring ORM, контекст Spring та Spring Web Flow. Кожний модуль надає різні платформи для розробки корпоративних додатків; наприклад, модуль Spring Web MVC призначений для розробки додатків на основі патерну Model–view–controller.

Головним компонентом фреймворку Spring є модуль Spring Core, що надає контейнер IoC. Існує два типи реалізацій контейнера Spring, а саме: bean factory та application context. Перший визначається за допомогою інтерфейсу BeanFactory та виступає в якості контейнера для бінів, що дозволяє відокремити конфігурацію та специфікацію залежностей від програмної логіки. Дана фабрика постає в якості центрального IoC-контейнера та відповідає за екземпляри об'єктів, також вона конфігурує і збирає залежності між цими об'єктами.

Подібно до об'єктно-орієнтованого програмування, що створює в додатках ієрархію об'єктів, AOP розбиває програми на аспекти або проблеми. Модуль Spring AOP дозволяє реалізувати ці проблеми або аспекти в Spring-

додатку. За допомогою анотації @Aspect звичайні класи або біни стають аспектами в Spring AO та допомагають у моніторингу збоїв в роботі програми, управлінні транзакціями і веденні журналів. Найпоширенішим прикладом використання управління транзакціями можна вважати банківські операції, такі як переказ суми з одного рахунку на інший.

Модуль Spring ORM використовується в додатку для доступу до інформації з баз даних. Він надає функціонал для маніпулювання БД за допомогою Java Data Objects, Hibernate та iBatis. Spring ORM підтримує DAO, що забезпечує зручний спосіб побудови таких ORM-рішень на основі DAO як просте декларативне управління транзакціями, прозора обробка виключень, потоко-безпечні легкі шаблонні класи, класи підтримки DAO.

Web-MVC модуль фреймворку реалізує архітектуру Model-View-Controller для створення веб-додатку. Він розділяє компоненти програми на моделі, контроллери та представлення. За принципом роботи Spring MVC при формуванні запиту від браузера він потрапляє спочатку до класу DispatcherServlet, який за допомогою відображень-обробників відправляє запит до контролера. Останній дістає та обробляє інформацію із запиту та надсилає результат у вигляді об'єкту моделі до класу DispatcherServlet, який використовує класи ViewResolver для відправки результатів до представлення.

Розширенням модуля Spring Web MVC став Spring Web Flow. Якщо Spring Web MVC реалізує контролери форм для реалізації попередньо визначеного робочого процесу, наприклад клас SimpleFormController, то Spring Web Flow допомагає у створенні Java-класу або XML-файлу, що дозволяє керуати робочим процесом між різними сторінками веб-додатку.

Пакет фреймворку Spring Web DAO забезпечує підтримку структурного шаблону, який дозволяє ізолювати прикладний/бізнес-рівень від рівня персистентності (зазвичай це реляційна база даних) за допомогою абстрактного API. Даний модуль вводить рівень абстракції JDBC та надає програмні та декларативні класи управління транзакціями.

Модуль контексту у Spring-програмі базується на головному модулі Core. Контекст додатку ApplicationContext є інтерфейсом BeanFactory, тому цей модуль отримує свої можливості з пакету org.springframework.beans та підтримує такі популярні функціональні можливості як інтернаціоналізація (I18N), валідація та завантаження ресурсів.

2.3 Структура діяльності автономного додатку

Spring Boot представляє собою автоматизовану та спрощену версію Spring Framework, яка доступна для розширення іншим функцоналом фреймворку за необхідності. Саме тому кожен проект, у тому числі додаток для автоматизації зміни стану ресурсів тестування, починається з розробки основи на цій частині фреймворку. Spring Boot реалізован на основі багаторівневої архітектури, в якій кожен шар взаємодіє з іншими, незалежно від того, вище або нижче вони в ієрархічному порядку.

Основною метою Spring Boot є видалення з програми незручних та громіздких налаштувань конфігурації на основі XML та частини анотацій. Поряд з цим Spring Boot забезпечує такі переваги, як мінливість (можливість змін конфігурації), автономність і готовність до запуску додатка з ранніх етапів розробки. Spring Boot скадається з чотирьох компонентів (рис. 2.2).

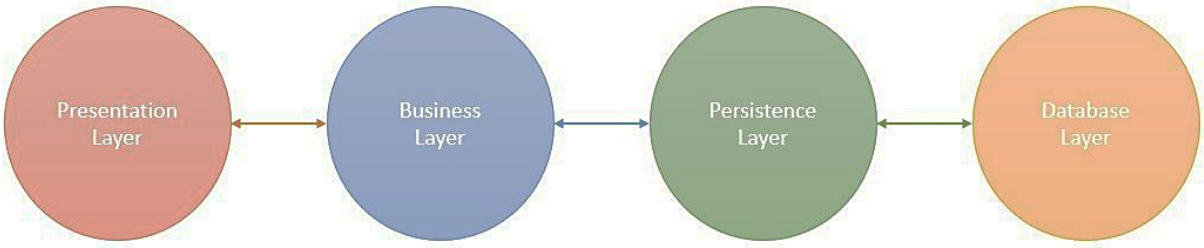

Рис. 2.2. Компоненти Spring Boot

1. Рівень представлення

Перший презентаційний рівень є верхнім шаром архітектури Spring Boot. Він складається з представлень (views) – інтерфейсної частини програми. Рівень

представлення обробляє HTTP-запити та виконує аутентифікацію, відповідаючи за розшифровку параметра поля JSON в Java об'єкти та навпаки. Після виконання аутентифікації запиту, він передає його на наступний бізнес-рівень.

2. Бізнес-рівень

Бізнес-рівень відповідає за всю бізнес-логіку. Він складається з класів сервісів та виконує валідацію та авторизацію.

3. Рівень збереження

Рівень персистентності утримує логіку зберігання бази даних, відповідаючи за перетворення бізнес-об'єктів Java в рядки бази даних та навпаки.

4. Рівень бази даних

Останній даних містить усі бази даних додатку, такі як MySql, MongoDB тощо. Зручною є можливість оперувати декількома базами даних. Рівень бази даних відповідає за виконання CRUD-операцій.

Базовою структурою застосунка по зміні стану ресурсів тестування буде слугувати впорядкована структура діяльності Spring Boot, яка організовує ресурси в процеси (рис. 2.3).

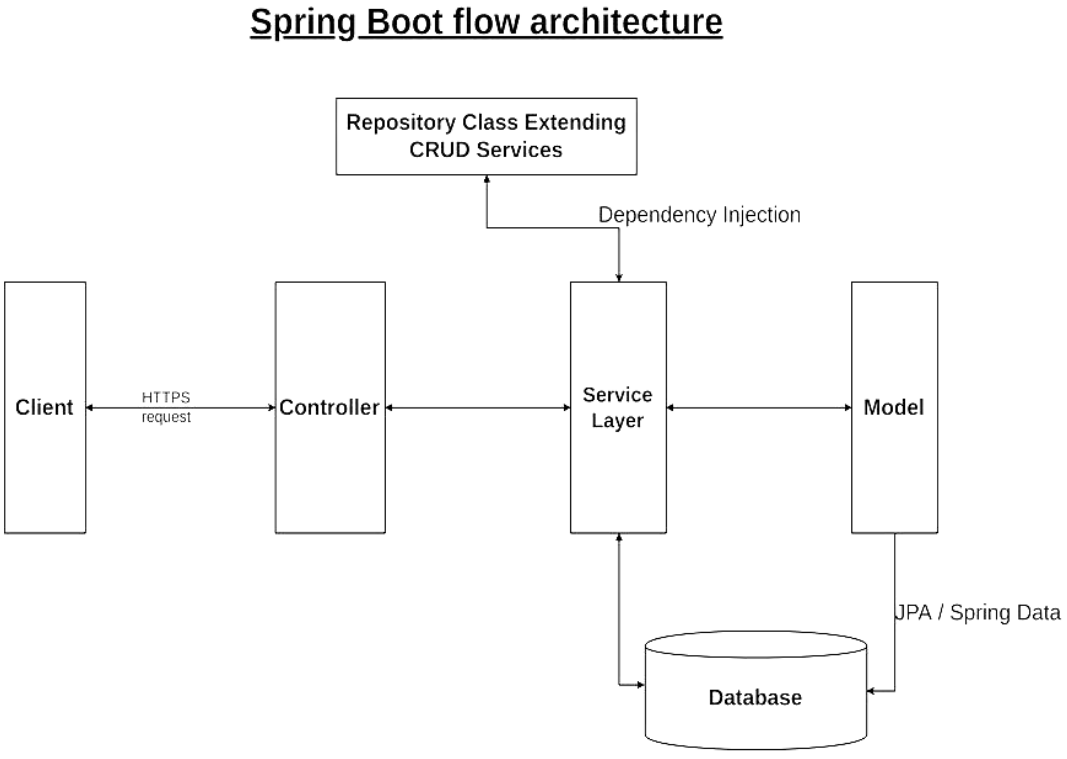

Рис. 2.3. Структура діяльності додатку

Згідно з наведеною схемою, спершу клієнт робить HTTP-запит (може бути будь-який: GET, PUT, POST тощо). Даний HTTP-запит пересилається на контролер, що відображає його, обробляє дескриптори та викликає виконання логіки з сервера. На сервісному рівні виконується вся бізнес-логіка. Дані з БД відображаються в клас моделі Spring Boot та обробляються за допомогою бібліотеки Java Persistence Library (JPA). У кінці сторінка JavaServer Pages (JSP) повертається як відповідь від контролера.

Архітектура Spring Boot спирається на фреймворк Spring. Отже, значною мірою в своїй діяльності він застосовує всі функції та модулі Spring, такі як Spring MVC, Spring Core тощо.

2.4 Обґрунтування рівня залежності та зв'язаності

В об'єктно-орієнтованому програмуванні залежність відноситься до ступеня прямого знання, яке один елемент має про інший, тобто як часто зміни в класі A викликають відповідні зміни в класі B.

Існує два типи такої залежності, зображених на рис.2.4:

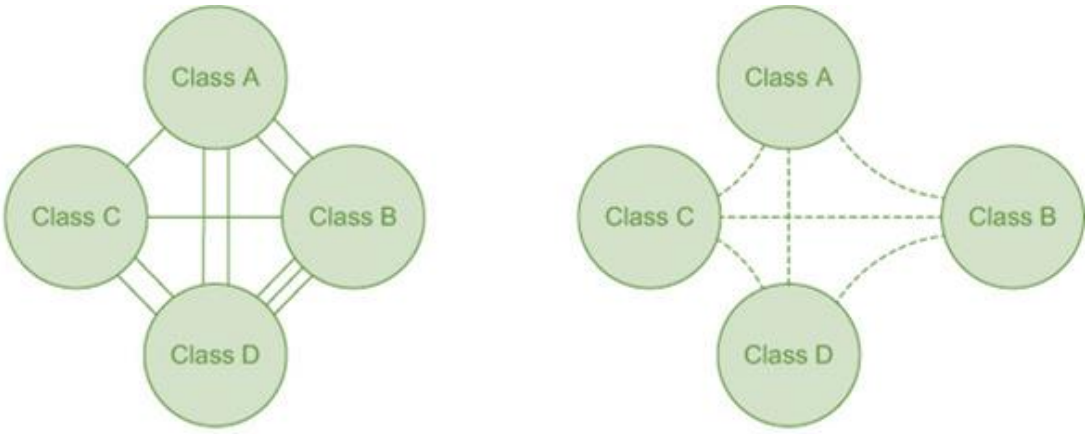

Рис. 2.3. Сильна та слабка залежність частин додатку

- тісна: у даному випадку два класи часто змінюються разом;

Клас A знає більше, ніж вкрай необхідно, про те, яким чином було реалізовано B, тому A і B тісно пов'язані між собою.

- слабка: такий зв'язок означає, що в основному класи є незалежними один від одного.

Класи вважаються слабко зв'язаними, якщо єдине знання, яке клас А має про клас В, це розкрите через інтерфейс. Spring Framework використовує механізм ін'єкції залежностей для того, щоб подолати проблеми жорсткого зв'язку між об'єктами додатку.

Для розробки додатку з управління станом ресурсів тестування потрібно визначитись з перевагами та недоліками кожного виду залежності, а саме:

- сильна залежність сповільнює тестування, в той час як слабка покращує здатність продукту до нього;
- при жорсткій залежності важко поміняти місцями функціонал між двома класами, але набагато простіше оперувати фрагментами з кодом, модулями, об'єктами чи компонентами при слабкій залежності складових;
- сильна залежність є швидшою і дешевшою в розробці невеликих додатків, проте не в підтримці;

Зробивши аналіз переваг, можемо бачити, що для створення додатку жорсткий зв'язок є поганим принципом, тому що він перешкоджає здібності продукту до тестування, зменшує повторне використання коду, гнучкість та робить зміни системи набагато складнішими. Слабка залежність є кращим вибором, адже такий підхід відкритий до оновлення та масштабування ПЗ, тому що зміна вимог вплине на менші частини програми.

Якщо звести принцип слабкої залежності в абсолют, то скоро можна прийти до розміщення взагалі всієї функціональності в єдиному класі. Таким чином, залежності від інших класів не буде взагалі, але вочевидь в цей клас потрапить абсолютно непов'язана між собою бізнес-логіка. Цей висновок призвів принципу об'єктно-орієнтованого програмування, який полягає у тому, що клас розроблений з єдиною, добре сфокусованою метою. І чим більш цілеспрямованим є клас, тим більшою є його зв'язність.

Головною перевагою високої згуртованості є легкість підтримки такого додатку через рідші зміни та легший процес тестування. Ще однією перевагою високої зв'язності є більша придатність для багаторазового використання, а отже швидша розробка та зниження витрат на підтримку проекту.

Низька залежність та висока логічна зв'язність є золотим стандартом розробки системи, до якого буде прагнути додаток з автоматизації зміни стану ресурсів тестування. Його суть можна об'єднати так: ПЗ має складатися з слабко пов'язаних між собою класів, які містять виокремлену бізнес-логіку. Дотримання цього принципу дасть змогу перевикористовувати створені класи, зберігаючи розуміння про їхню зону відповідальності.

2.5 Технології Spring для побудови додатка

2.5.1 Платформа Spring Boot

Щоб оптимізувати процес управління залежностями, Spring Boot неявно пакує необхідні сторонні залежності для кожного типу програми на основі Spring і надає їх розробнику за допомогою так званих starter-пакетів (spring-boot-starter-web, spring-boot-starter-data-jpa і т.д. .д.). Для розробки додатку по автоматизації зміни стану тестових ресурсів будуть необхідні наступні пакети:

- «spring-boot-starter-web» для створення Spring web-додатку, додасть у проект такі популярні бібіотеки, як spring-webmvc, jackson-json, validation-api та Tomcat;
- «spring-boot-starter-data-jpa» для доступу до бази даних без пошуку сумісніх драйверів;
- «spring-boot-starter-thymeleaf» – стартер для створення веб-додатків MVC з використанням представлень Thymeleaf;
- «spring-boot-starter-security» є середовищем для аутентифікації та авторизації користувачів.

Отже, при використанні Spring Boot спеціальний XML-файл pom.xml, що впроваджує інформацію про проект та різні деталі конфігурації, містить набагато менше рядків, ніж при використанні його в звичайних Spring-додатках.

Другою чудовою здатністю Spring Boot є автоматична конфігурація програми, яка може бути повністю перевизначена в будь-який момент за допомогою налаштувань користувача.

Після вибору відповідного starter-пакету, Spring Boot спробує автоматично налаштувати Spring-додаток на основі доданих jar-залежностей. Наприклад, для обраного пакету spring-boot-starter-web, Spring Boot автоматично конфігурує такі зареєстровані біни, як DispatcherServlet та ResourceHandlers.

Першою складовою створюваного Spring Boot-додатку будуть різні моделі програми організовані в пакеті ua.testerossa.model, їх DTO (або об'єкти передачі даних), що присутні в пакеті dto. Існують різні думки про необхідність використання DTO. В ПЗ по автоматизації зміни стану ресурсів тестування було прийнято рішення їх впровадження для послаблення зв’язку між шаром моделі програми з шаром інтерфейсу користувача.

DTO дозволяють передавати тільки ті дані, якими потрібно поділитися з користувальницьким інтерфейсом, а не весь об'єкт моделі, в процесі розробки ПЗ було агреговано за допомогою декількох підоб'єктів і збережено в базі даних.

Наступною складовою додатку будуть сервіси та об'єкти доступу до даних. Об'єкти доступу до даних (DAO) присутні в пакеті репозиторію ua.testerossa.repository. Всі вони є розширеннями інтерфейсу JpaRepository. Методи інтерфейсу є базовими, що зображені на рисунку 2.4., та допомагають сервісному рівню отримувати, зберігати та обробляти дані з БД.

Сервісний рівень визначається під пакетом сервісів у ua.testerossa.service та ніколи не приймає модель в якості вхідних даних і ніколи не повертає її. Це ще одна з найкращих практик, якої потрібно дотримуватись при розробці багаторівневої архітектури Spring-додатку з автоматизації зміни старну ресурсів. Рівень контролера взаємодіє з сервісним рівнем, коли він отримує запит від

представлення (view), при цьому він не повинен мати доступу до об'єктів моделі та завжди повинен спілкуватися в термінах нейтральних DTO.

```
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>
  List<T> findAll();

  List<T> findAll(Sort sort);

  List<T> findAllById(Iterable<ID> ids);

  <S extends T> List<S> saveAll(Iterable<S> entities);

  void flush();

  <S extends T> S saveAndFlush(S entity);

  <S extends T> List<S> saveAllAndFlush(Iterable<S> entities);
```

Рис. 2.4. Методи взаємодії з базою даних

Рівень контролера пов'язує все разом з моменту перехоплення запиту до моменту підготовки та відправлення відповіді. Цей рівень представлений в пакеті ua.testerossa.controller.

Контролери додатку по автоматизаціїї змін стану ресурсів будуть працювати на концепції Spring WebMVC. Вони повинні відповідати на вхідні веб-запити об'єктами Spring ModelAndView, що містять дані для відображення на відповідних представленнях/формах Thymeleaf.

Основний метод для запуску програми Spring Boot додатку буде знаходитись у класі, що містить анотацію @SpringBootApplication, яка включає автоконфігурацію, сканування компонентів і конфігурацію завантаження Spring.

Чудовою особливістю Spring Boot є вбудований сервер. Отже усе, що знадобиться для запуску програми - це створити виконуваний jar-файл за допомогою Maven і запустити метод, зображений на рисунку 2.5.

```java
@SpringBootApplication
public class SpringSecurityApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringSecurityApplication.class, args);
  }
```

Рис. 2.5. Точка запуску Spring Boot додатку

2.5.2 Захист додатку з Spring Security

У 2020 році через локдаун у світовому ринку стався масштабний зсув у бік оцифрування. У результаті багатьом організаціям знадобились сайти та додатки, щоб пропонувати своїм клієнтам товари або послуги. Однак відсутність заходів безпеки стала значною проблемою при такому швидкому впровадженні цифрової трансформації. Особливо важлива безпека корпоративних додатків, оскільки сучасні застосунки часто доступні в різних мережах і під'єднані до хмари, що підвищує їхню вразливість до загроз і порушень безпеки.

Зростає потреба забезпечити безпеку не тільки на рівні мережі, а й на рівні самих застосунків, адже атаки хакерів все частіше націлюються саме на них. Такий підхід дає все більше переваг і саме його було застосовано у створенні корпоративного додатку з автоматизації зміни стану ресурсів тестування.

Модуль Security у Spring представляє собою середовище авторизації, аутентифікації та контролю доступу з широким спектром доступних налаштувань. Загалом, це стандартний фреймворк, що застосовується з метою захисту програмних додатків, які працюють на основі Spring.

У додатку по автоматизації зміни стану ресурсів тестування Spring Security буде працювати захисною оболонкою для всього функціоналу програми, як зображено на рис. 2.6.

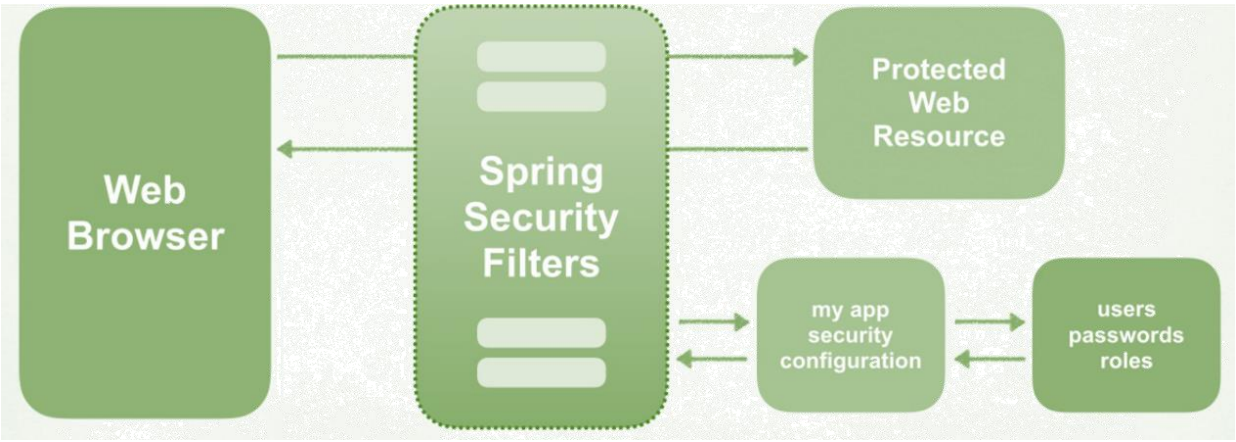

Рис. 2.5. Місце Spring Security у структурі програми

Дана оболонка Spring Security Filters виконує робочий процес за діаграмою на рисунку 2.5 та показує, як така інформація про користувача, як ім'я, пароль, повноваження, зберігається та керується в процесі роботи програми.

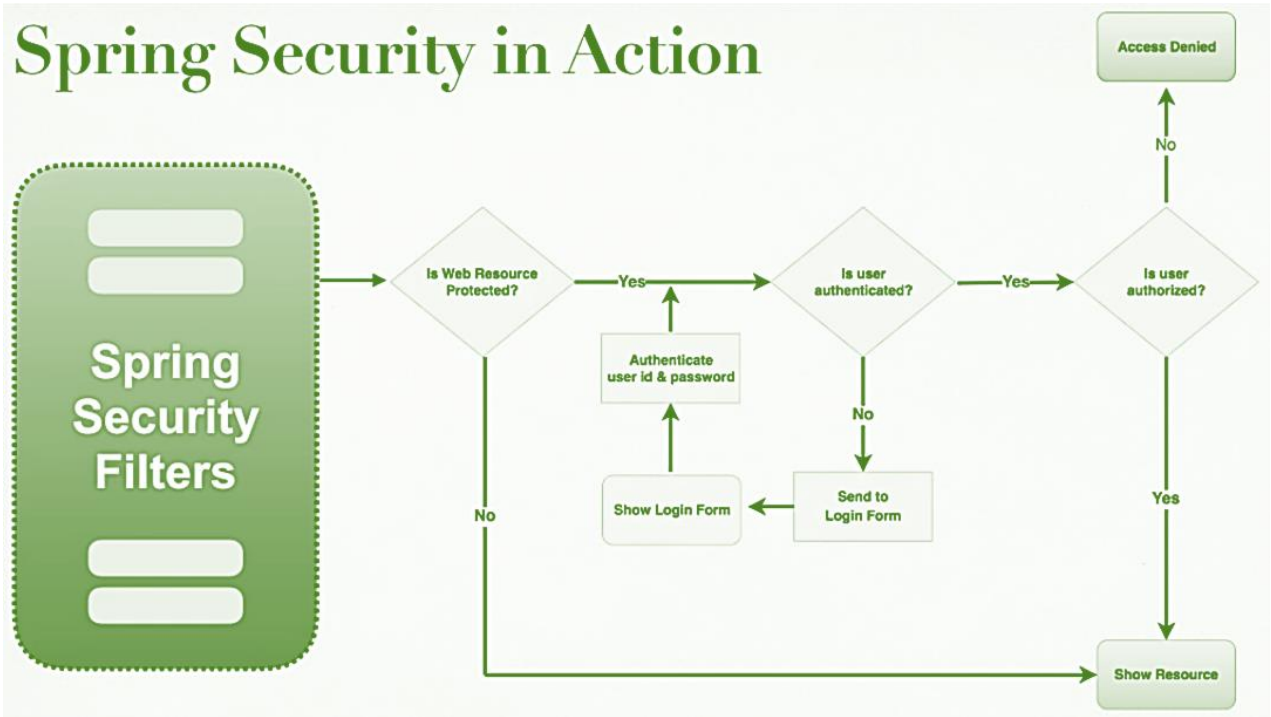

Рис. 2.6. Діаграма роботи фільтру Spring Security

Для інтеграції Spring Security у додаток автоматизації зміни стану ресурсів тестування необхідно реалізувати ряд кроків.

По-перше, у проект додається стартер-пакет spring-boot-starter-security. Зазвичай включення будь-якого стартера в POM-файл нічого не дає: щоб відбулися зміни в роботі програми, потрібно написання додаткового коду. Проте у випадку Spring Security все інакше, адже відразу за замовчуванням створюються функціонал користувача. Ім'я його user, а пароль генерується автоматично під час запуску програми.

Таким чином, під час одного тільки додавання spring-boot-starter-security в POM-файл відбувається:

- Spring Security створює користувача з ім'ям user та автоматично згенерованим паролем, який можна бачити в консолі працюючого додатку;
- створюється сторінка з формою для введення імені та пароля – функціонал Form-based аутентифікації;
- ім'я та пароль для входу в програму автоматично перевіряються;
- всі url виявляються недоступними, поки без завершення успіхом процесу входу під зареєстрованим користувачем;

Наступним кроком є робота з InMemoryUserDetailsManager. З точки зору отримання параметрів користувача із запиту, за допомогою описаної вище Form-Based автентифікації, ім'я та пароль відправляються через форму та приймаються на сервер із запиту як POST-параметри.

Щоб створити свого користувача в In-Memory автентифікації необхідно написати клас-конфігурацію SecurityConfig, який розширює клас WebSecurityConfigurerAdapter і зробити його біном за допомогою @EnableWebSecurity. Аутентифікацію виконує AuthenticationManager, але він не потребує явної взаємодії зі сторони розробки, замість цього необхідно перевизначити метод configure (AuthenticationManagerBuilder auth) класу WebSecurityConfigurerAdapter – так отримується доступ до AuthenticationManagerBuilder. У ньому потрібно задати тип аутентифікації – місце зберігання користувача. У випадку додатку по автоматизації зміни стану ресурсів тестування тип аутентифікації буде не In-Memory, а Jdbc. Далі йдуть

специфічні налаштування обраного AuthenticationManager, в яких уточнюється, як AuthenticationManager отримує збереженого користувача, щоб потім порівняти його з введеними даними. Насправді AuthenticationManager формує не тільки ім'я та пароль, а ще дозволи користувача.

Для формування повноцінної схеми захисту у додатку буде створено отримання дозволу до окремих частин функціоналу через систему ролей. Важливим є створення PasswordEncoder, завдяки якому задається, як шифрувати пароль. При стандартній реалізації Encoder, він не робить нічого, тобто залишає пароль у первісному вигляді. Звичайно, в створюваному застосунку такий NoOpPasswordEncoder не придатний – пароль буде додатково шифруватися за допомогою BCryptPasswordEncoder з посиленим рівнем шарів захисту.

Одним із методів авторизації у Spring Security є використання анотації @PreAuthorize, у якій за допомогою виразів можна наочно описати правила, дотримуючись яких модуль авторизації вирішує, чи дозволити чи заборонити проведення операції. У додатку з автоматизації стану ресурсів тестування доступ до кожної частини функціоналу регулюється наведеною анотацією з додаванням необхідного дозволу. Наприклад, для простого перегляду наявних ресурсів тестування потрібний дозвіл користувача описується таким чином: @PreAuthorize("hasAuthority('read')").

І останнім кроком буде налаштування автентифікації та авторизації під час роботи з БД через DaoAuthenticationProvider. Цей постачальник автентифікації, ймовірно, є найбільш часто використовуваним у фреймворку. Як і більшість інших провайдерів автентифікації, DaoAuthenticationProvider використовує UserDetailsService для пошуку імені користувача, пароля та наданих повноважень, але на відміну від більшості інших провайдерів автентифікації, які використовують UserDetailsService, цей провайдер автентифікації фактично вимагає надання паролю та оцінює його дійсність.

2.6 Аналіз безпечності та швидкості алгоритмів шифрування

Важливим етапом захисту обігу даних у програмі є криптографічний захист інформації, адже криптографія є основою безпеки сучасних інформаційних систем. Вона оточує нас повсюди: зберігання облікових даних, зв'язок електронною поштою, мобільні платежі та цифрова готівка – це лише частина найпоширеніших випадків.

Криптографія є однією з найскладніших тем інформаційної безпеки, не можна піддаватися спокусі запровадити власні бібліотеки, оскільки неможливо обмеженими силами перевірити новий алгоритм на всі можливі загрози та постійно слідкувати за оновленням ризиків безпеки даних. Більшість сучасних мов мають реалізовані крипто бібліотеки та модулі, тому для прийняття оптимального рішення щодо кращого способу захисту інформації необхідно виконати детальний аналіз та порівняння.

Шифрування – це процес кодування або шифрування даних, щоб їх міг прочитати лише той, хто має засоби для повернення їх до початкового стану. Технології шифрування є одним із важливих елементів будь-якого безпечного обчислювального середовища.

Безпека шифрування полягає в здатності алгоритму генерувати зашифрований текст, який нелегко повернути до початкового відкритого стану. Для налаштування будь-якої базової схеми шифрування, дуже важливо, щоб наступні питання були вирішені правильно:

- наскільки безпечним алгоритм вважається на даний момент у криптографічній літературі;
- характеристики продуктивності алгоритму (наприклад чи підтримує паралельне шифрування та його «чиста швидкість»);
- наскільки політично безпечним є рішення використовувати певний алгоритм;
- використання правильно підібраних ключів і їх розмірів.

Важливо пильно налаштувати всі параметри, адже незначний прорахунок в безпеці конфігурації зробить всю криптосистему відкритою для атак.

В рамках дослідження за допомогою бібліотеки javax.crypto було реалізовано шифрування найпопулярнішими у JDK Sun алгоритмами: AES128, AES256, SHA256, SHA512, DES, Triple DES і Blowfish. Для кожної реалізації було виміряно швидкість шифрування у мілісекундах (рис. 2.7) та описано основні характеристики, важливі для безпеки даних (табл. 1).

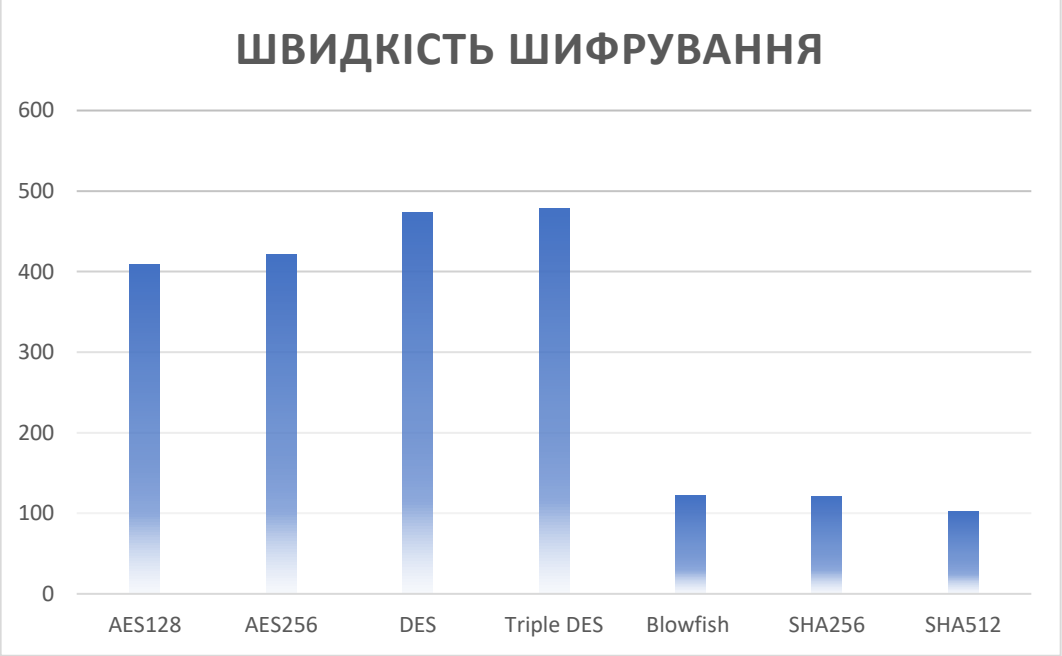

Рис. 2.7. Різниця швидкості роботи алгоритмів

Потрібно відзначити, що AES розшифровується як Advanced Encryption Standard і є алгоритмом, який більшість людей використає зрештою, якщо у них немає вагомих причин використовувати щось інше. Це рішення є політично безпечним, адже стандарт шифрування Національного інституту стандартів і технологій США (NIST) схвалює AES із 192 або 256-бітними ключами для шифрування секретних даних.

Таблиця 1

Порівняльна характеристика алгоритмів

Алгоритм	Розміри ключа	Швидкість	Залежність швидкості від розміру ключа	Коментар
AES	128-256	Середня	Так	Безпечний, його перевага полягає в тому, що він дозволяє використовувати 256-бітний розмір ключа, який має захистити від певних майбутніх атак (collision attacks і потенційних алгоритмів квантового обчислення)
DES	56	Повільна	Ні	Дуже небезпечно: навіть одна машина Copacobana вартістю 10000 USD може знайти ключ DES приблизно за тиждень.
Triple DES	112-168	Дуже повільна	Ні	Доволі безпечний, особливо для невеликих розмірів даних. 168-бітний варіант, за оцінками NIST, забезпечує безпеку даних до 20304 року, проте AES пропонує вищий рівень безпеки за меншої вартості CPU.
Blowfish	128-448	Швидка	Ні	Вважається безпечним, але пройшов менше крипто аналізу, ніж інші алгоритми. Blowfish було замінено Twofish, але останній не підтримується як стандарт у Java.
SHA	256-512	Дуже швидка	Так	Проблема полягає в тому, що лише SHA256 та SHA512 (тип SHA2, який генерує 256/512-бітове хеш-значення) залишається безпечним — SHA-1 (перша версія SHA, яка генерує 160-бітове хеш-значення) має відомі вразливості.

Окрім вибору алгоритму потрібно визначити розмір ключа, що забезпечує «міцність» шифрування, припускаючи, що фактичний алгоритм в інших аспектах безпечний. Розмір ключа зазвичай корелює з кількістю припущень, які зловмиснику потрібно буде зробити, щоб знайти ключ за допомогою «brute force», припускаючи, що всі можливі ключі однаково ймовірні.

Використання ключів додає новий рівень безпеки до методів захисту інформації. Існує дві загальні категорії алгоритмів на основі ключів:

- симетричні алгоритми шифрування: симетричні алгоритми використовують той самий ключ для шифрування та дешифрування. Ці алгоритми можуть працювати в потоковому режимі (з бітами або байтами даних) або в блочному режимі (з блоками даних фіксованого розміру). Вони зазвичай використовуються для таких програм, як шифрування даних, файлів і особливо даних у мережах зв’язку (наприклад електронні листи, TLS, миттєві повідомлення тощо);
- асиметричні алгоритми шифрування: на відміну від симетричних алгоритмів, які використовують той самий ключ для обох операцій шифрування та дешифрування, асиметричні алгоритми використовують два окремих ключі для цих двох операцій. Ці алгоритми використовуються для обчислення цифрових підписів і протоколів встановлення ключів.

Як обрати розмір ключа? Загалом, мінімальний необхідний розмір ключа – це деяка комбінація максимального розміру ключа, який зараз можна використати для заданого алгоритму, екстрапольованого до кількості років, які потрібно зберігати конфіденційними зашифровані дані. Наприклад, NIST рекомендує мінімум «128 біт міцності», щоб зберегти конфіденційність даних «після 2030 року». А за посібником із найкращих методів шифрування даних компанії Visa, що випускає кредитні картки, з метою передачі номерів кредитних карток «ключі повинні мати потужність принаймні 112 еквівалентних бітів». Вони фактично вважають 128 біт (як у мінімальному розмірі ключа AES)

«міцнішими, ніж потрібно», хоч можливо на це впливає відносно короткий термін служби кредитних карток.

2.7 Обґрунтування вибору механізму асинхронного програмування

Для реалізації автоматичного внесення змін до ресурсів тестування через відкладенний таймер або задану дату та час необхідно розробити паралельні потоки роботи у створюваному додатку.

На відміну від процедурного програмування, асинхронне програмування полягає у створенні неблокуючого коду шляхом запуску всіх завдань в окремих потоках, замість одного основного потоку програми,а також постійному повідомленні основного потоку про хід виконання, статус завершення або про збій у виконанні завдання. Таким чином, основний потік програми не буде блокуватися в очікуванні завершення завдання, оскільки воно може виконуватися паралельно. Такий паралелізм значно підвищить продуктивність програми та зменшить час відповіді сервера.

Completable Future – це новітній функціонал для асинхронного програмування на мові Java, який буде застосовуватись у застосунку.

CompletableFuture є розширенням Future API, яке було представлено ще в Java 5. Future використовується як посилання на результати асинхронних обчислень з CompletableFutureJavata та містить методи isDone() та get() для підтвердження виконання обчислення та отримання результату обчислення після його завершення. Future API не є оптимальним варіантом для додатку з автоматизації зміни стану ресурсів тестування через певні обмеження:

- Future API не дозволяє ручне управління завершенням роботи – програма призупиняється та перестає відповідати користувачам, якщо віддалений сервер з Future API вийде з ладу;
- неможливо паралельно виконати декілька завдань, а потім об'єднати результати разом;

- для Future API немає конструкцій по обробці винятків та механізму для створення декількох етапів обробки, які можуть бути з'єднані між собою, це потрібно робити тільки вручну;
- Future не має механізму повідомлення про завершення роботи його API.

На щастя, з виходом Java 8, CompletableFuture став вирішенням усіх перерахованиї вище проблемами і забезпечує набагато кращий підхід до асинхронного програмування на Java. Він реалізує інтерфейси Future та CompletionStage та забезпечує наступні переваги:

- у ситуації коли віддалений сервіс з CompletableFuture API не працює під час його використання, є можливість вручну завершити виконання потоку, щоб отримати дані;
- CompletableFuture дозволяє об'єднувати кілька API по принципу ланцюжка, чим дозволяє створювати асинхронний робочий процес;
- має механізм об'єднання декількох CompletableFuture в єдиний;
- надає механізм обробки виключень;
- дозволяє реалізувати функцію зворотного виклику CompletableFuture API, що викликається коли відповідь стає доступною.

У CompletableFuture є 50 методів з різноманітним функціоналом, тому вибір даного API є гарним вкладенням у подальший розвиток ПЗ через легшу підтримку та можливість масштабування на основі вбудованих технологій.

2.8 Висновки

В процесі проведених досліджень та аналізу було встановлено, що Spring Framework є великою та зручною платформою для створення веб-проектів на Java, що складається з безлічі незалежних модулів для різних завдань: від простих веб-додатків до Big Data.

Основна перевага фреймворку Spring полягає у можливості розробки додатку як набору слабко зв'язаних компонентів за допомогою Dependency Injection. Що менше компоненти програми знатимуть один про одного, то

простіше розробляти новий і підтримувати наявний функціонал програми. Проте у даному підході до створення програми важливо утримати баланс та групувати код у класах за його бізнес-логікою, таким чином, дотримуючись у створенні додатку принципу «Low Coupling, High Cohesion».

Для розробки застосунку по автоматизації зміни стану ресурсів тестування буде використано наступний перелік модулей фреймворку:

- Spring Boot;
- Spring Data JPA;
- Spring MVC Thymeleaf;
- Spring Security.

Кожен з них додасть свій вклад у розробку та функціонування ПЗ. Spring Boot автоматично сконфігурує проект на основі стартових пакетів, швидко та легко управлятиме залежностями та полегшить розгортання додатку на Spring за допомогою підтримки вбудованого серверу для запуску додатків. Специфікація Java для керування реляційними даними Spring Data JPA дозволить отримувати доступ і зберігати дані між об’єктом/класом Java і реляційною базою даних. Spring MVC Thymeleaf надасть набір Spring-інтеграцій, які дозволять використовувати його як повнофункціональну заміну JSP в Spring-додатку для створення інтерфейсу користувача. Фреймворк Spring Security забезпечить аутентифікацію, авторизацію та захист від поширених атак.

Доклавши достатніх зусиль, практично будь-яку криптографічну систему можна успішно атакувати. Головне питання полягає в тому, скільки часу та ресурсів необхідно щоб її зламати. Як видно з проведеного аналізу, є багато деталей, на які слід звернути увагу, щоб зробити все правильно під час розробки та реалізації схеми шифрування у додатку.

Найбільш перевіреним та збалансованим алгоритмом шифрування зараз можна вважати AES. Для того щоб зберегти дані в безпеці принаймні 20 або 30 років краще використовувати 256-бітний ключ; 128-бітний ключ може підійти, якщо ретельно продумано захист від brute force, а також якщо протягом життя

даних малоймовірна загроза квантових обчислень. Наприклад, для даних кредитних карток можна бути впевненим, що термін дії всіх існуючих закінчиться до того часу, коли квантові комп’ютери стануть нашою реальністю.

Отже, для шифрування особливо цінних данних у додатку було обрано алгоритм шифрування AES з 256-бітний ключем. Функціонал відкладених автоматичних змін у ресурсах тестування буде реалізовано за допомогою нового класу для асинхронної роботи CompletableFuture.

РОЗДІЛ 3
РОЗРОБКА ЗАСТОСУНКУ ДЛЯ АВТОМАТИЗАЦІЇ ЗМІНИ СТАНУ РЕСУРСІВ ТЕСТУВАННЯ

3.1. Структура інформаційної системи автоматизації зміни стану ресурсів тестування

Внутрішня структура створюваної системи являє собою веб-сайт, який складається з двох частин: клієнтська презентаційна (frontend) та серверна (backend). Користувач системи взаємодіє безпосередньо з презентаційною частиною, через функціонал якої має змогу:

- зареєструватись;
- увійти в систему;
- переглянути обраний вид ресурсів;
- додати/видалити/змінити ресурс з часовим відкладенням, при умові наявності необдних дозволів у його ролі;
- оперувати дозволами та ролями користувачів, доступно для ролі admin.

Для автентифікації, авторизації, отримання дозволів та оперування даними з БД frontend-частина взаємодіє з backend-частиною системи. Система комунцкації презентаційної і серверної частини додатку зображена на рис. 3.1.

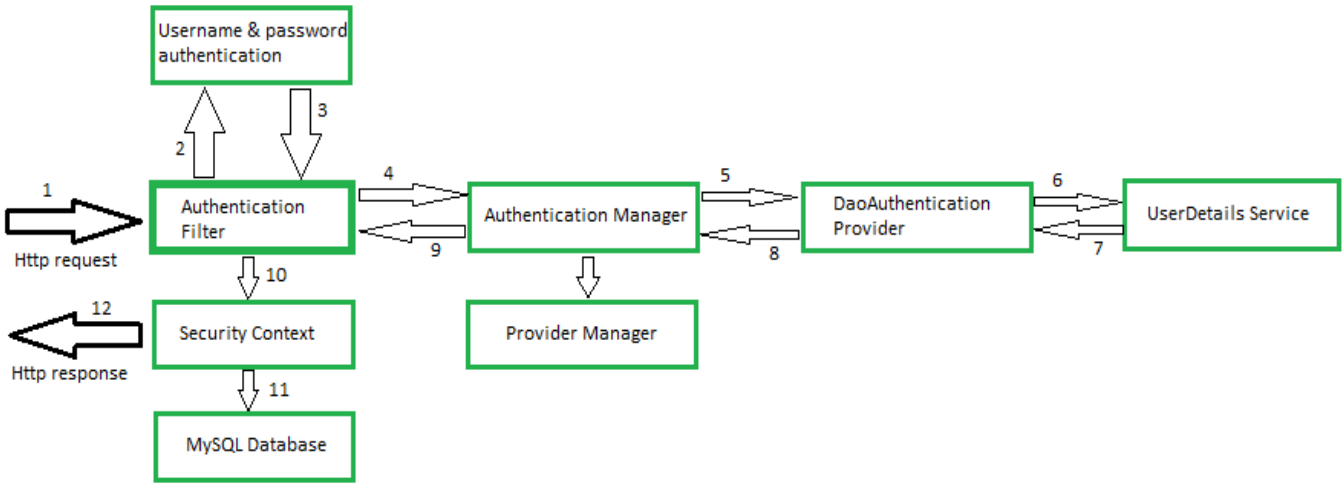

Рис. 3.1. Архітектура комунікації частин програми

Алгоритм взаємодії має наступну структуру:

1. користувач за допомгою веб-клієнта (браузера) надсилає запит певної веб-сторінки застосунку;
2. незалежно від почтакової сторінки додаток для автоматизації зміни стану об’єктів тестування (надалі testerossa) перенаправляє клієнта на сторінку авторизації;
3. перед тим, як запит потрапляє до обробки бізнес-логіки та відправки результату, він спочатку перехоплюється ланцюжком фільтрів безпеки для аутентифікації та авторизації;
4. коли запит перехоплюється відповідним фільтром аутентифікації (AuthenticationFilter), він отримує ім'я користувача та пароль із запиту і створює об'єкт аутентифікації;
5. використовуючи створений об'єкт аутентифікації, фільтр викликає метод аутентифікації менеджера аутентифікації (Authentication Manager), який є лише інтерфейсом, а фактична реалізація методу автентифікації забезпечується менеджером провайдерів (ProviderManager);
6. зі свого методу автентифікації ProviderManager викликає метод автентифікації відповідного AuthenticateProvider, у відповідь отримуючи Основний об'єкт автентифікації (Principal Authentication Object), якщо автентифікація є успішною;
7. у testerossa для реалізації інтерфейсу з єдиним методом автентифікації AuthenticationProvider було обрано провайдер автентифікації DaoAuthenticationProvider, який отримує дані користувача з UserDetailsService, на цьому кроці відбувається вся фактична автентифікація;
8. використовуючи сервіс UserDetails, постачальник автентифікації (AuthenticationProvider) отримує об'єкт користувача, що відповідає обліковим даним з бази даних, ці дані об'єкта користувача порівнюються з вхідними обліковими даними об'єкта автентифікації, якщо автентифікація успішна, то

у відповідь повертається Основний об'єкт автентифікації (Principal Authentication Object);

9. після проходженння зазначених вище перевірок, якщо клієнт вводив URL, відмінний від авторизаційного, він перенаправляється на обраний функціонал або на початкову сторінку з усіма ресурсами тестування, за умови наявності необхідних дозволів, якщо дозволів немає, відображається строрінка помилки;
10. якщо обрана бізнес-логіка потребує зв'язку з базою даних, то по налаштованому при запуску додатку підключенню до БД, відбувається взаємодія з сервером MySQL, що приймає запит до бази даних, обробляє його, а потім відправляє результати у відповідь;
11. контролер завершує виконнання обробки запиту, формуючи результат, та надає презентаційній частині програми відповідь для відображення.

3.2 Стурктура та опис бази даних

Незважаючи на те, що більшість систем управління БД мають стандартні функції, які дають змогу клієнтам створювати й одержувати доступ до інформації в базі даних, проте способи виконання завдань можуть відрізнятися.

Додаткові можливості, функції та підтримка є особливими у кожній СУБД, для проекту testerossa було обрано систему MySQL, зважаючи на перелічені переваги:

- MySQL сумісна з усіма сучасними платформами, такими як Windows, Unix, Linux;
- простота завантаження та управління;
- MySQL забезпечує повну безпеку даних, адже тільки авторизовані користувачі мають доступ до бази даних;
- масштабованість та безкоштовне використання.

У застосунку testerossa доступ до бази даних доступ MySQL налаштовано за допомогою Spring Data JPA, що є зручним механізм для взаємодії з сутностями

бази даних, організації їх у репозиторії, зміні та вилученню даних, в частині випадків у додатку для описаних маніпуляцій буде достатньо тільки оголосити інтерфейс і метод у ньому, без імплементації.

Основним елементом взаємодії бази даних та додатку testerossa є репозиторій, через який сервіс отримує необхідні дані для бізнес-логіки, як зображено на рис. 3.2.

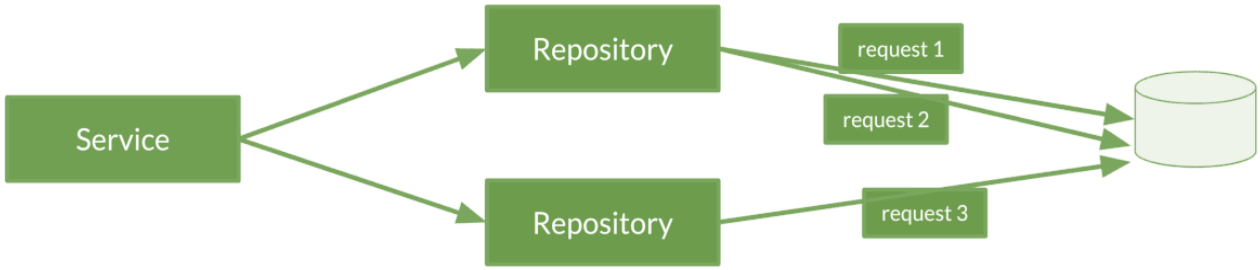

Рис. 3.2. Архітектура взаємодії з базою даних

Обраний інтерфейс JpaRepository забезпечує основні операції з пошуку, збереження, видалення даних (CRUD операції). Додаткові запити до даних можна будувати прямо з імені методу, що було зроблено для пошуку користувача за поштою: Optional<User> findByEmail(String email);

Структура бази даних є змінюваною в залежності від створюваних користувачами потрібних тестових ресурсів, у базовому наборі є стандартні ресурси, з якими структура БД має наступний вигляд:

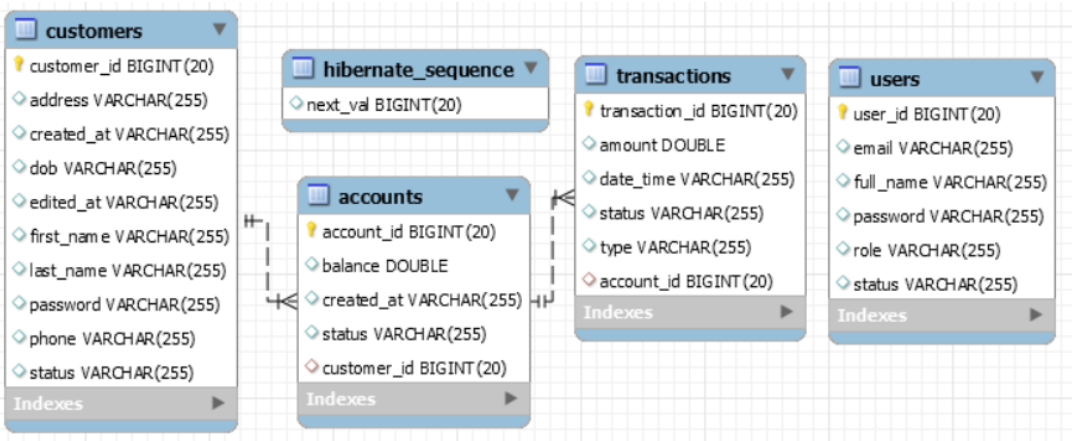

Рис. 3.3. Структура бази даних

3.3Опис ІС testerossa та алгоритмів її функціонування

Основна обробника даних в системі відбувається в серверній частині застосунку. В якості вхідних даних в програмному комплексі використовуються текстові дані з форм, які користувач заповнює для зміни стану ресурсів тестування. Вихідними даними веб-додатку можна вважати множину збережених у системі тестових ресурсів, готових до використання та відображених на сторінках перегляду ресурсів, сгрупованих за їх типом.

Функціональне призначення системи полягає в наданні клієнту інтуїтивно-зрозумілого і зручного інструмента для управління станом ресурсів тестування з можливістю створення сценарію їх змін у майбутньому.

Для розробки застосунку було застосовано наступні технології:

- Java 11 – створення серверної логіки;
- HTML – створення розмітки UI;
- Thymeleaf – серверний механізм Java-шаблонів;
- CSS, JavaScript, Bootstrap – стилізація та функціонал компонентів UI;
- Spring Security – для аутентифікації та авторизації;
- Spring Data JPA – API для роботи з базою даних;
- Spring Boot – створення бізнес логіки додатку;
- MySQL – керування реляційною БД;
- Project Lombok – скорочення шаблонного коду;
- JUnit – тестування компонентів програми.

Розроблений застосунок для автоматизації зміни стану ресурсів тестування має відповідати наступним вимогам:

- інтуїтивно-зрозумілий інтерфейс для роботи з системою;
- наявна система автентифікації та авторизації;
- реєстрація користувачів;
- зручна навігація по веб-сайту;

- можливість роботи в браузерах на ПК, ноутбуках та мобільних пристроях з різними характеристиками та конфігураціями;
- шифування особливо цінних даних користувачів;
- наявність модулів створення, оновлення та видалення ресурсів;
- функціонал відкладення обраних змін;
- гнучкість для модифікацій та масштабування.

Інтерфейс користувача веб-сайту складається з основних модулів:

1. сторінка авторизації;
2. сторінка реєстрації;
3. сторінка помилки;
4. сторінка доступних ресурсів;
5. сторінка управління користувачами;
6. сторінка зміни стану ресурса;

Після переходу за інтернет-адресою програмного комплексу testerossa, користувач побачить сторінку авторизації, що зображена на рис. 3.3.

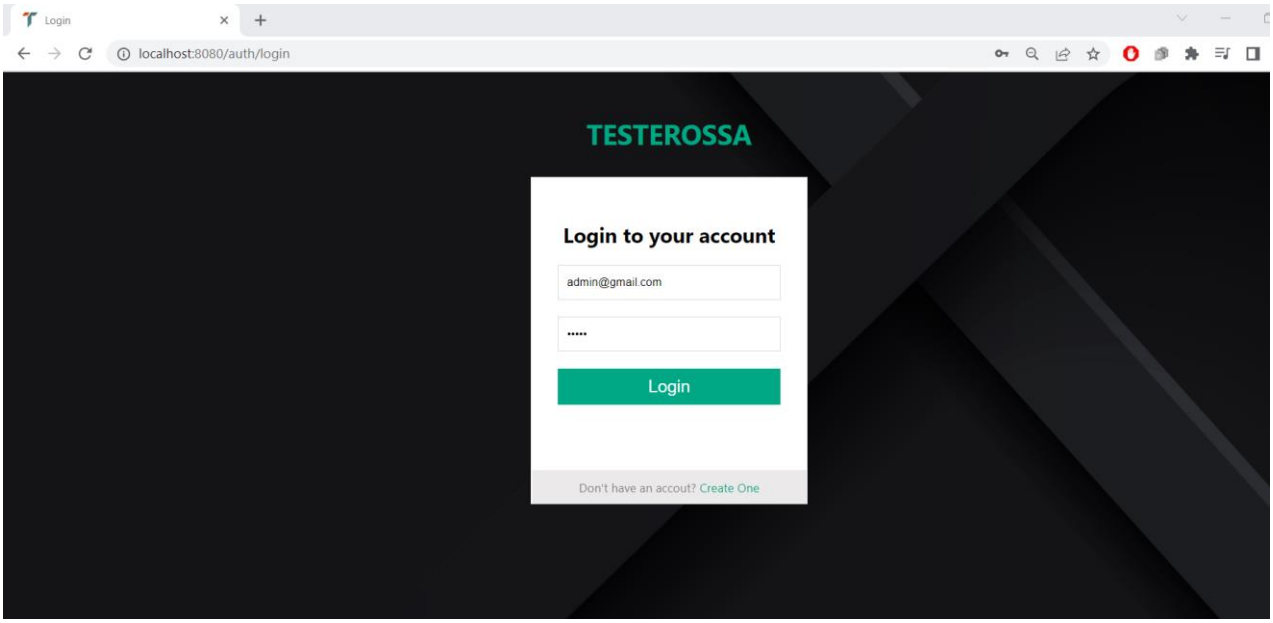

Рис. 3.4. Сторінка авторизації

Якщо користувач авторизований в системі, то він може увійти за посиланням «Login», зазначивши у формі свій логін-пошту та пароль. Для реєстрації в системі потрібно натиснути на посилання «Don't have an accout? Create One» та ввести облікові дані у формі на рис. 3.5.

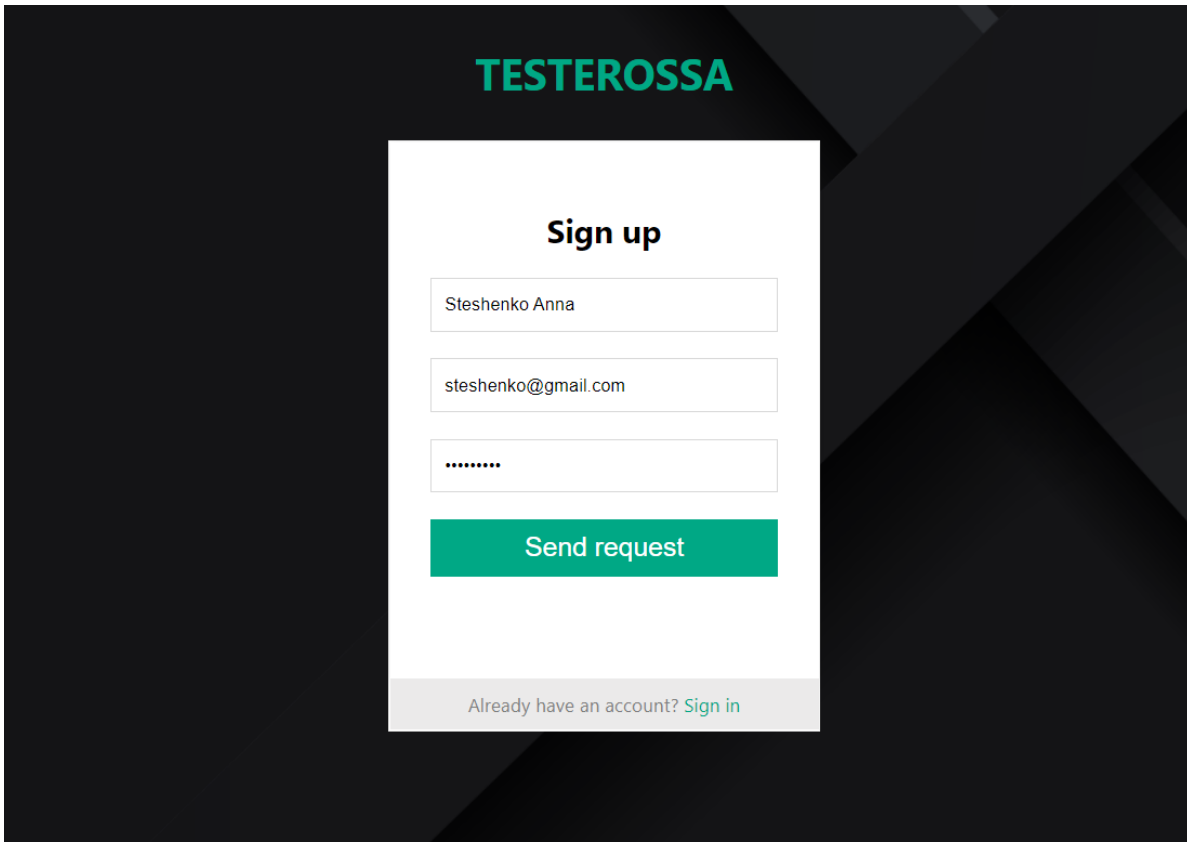

Рис. 3.5. Сторінка реєстрації в системі

Після успішної аворизації користувач потрапляє на сторінку доступних йому тестових ресурсів, зображену на рис. 3.6.

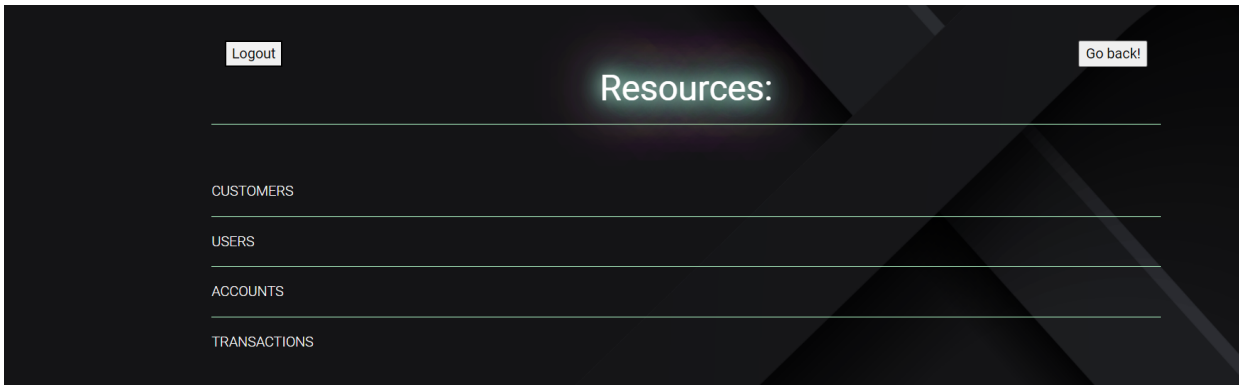

Рис. 3.6. Сторінка доступних ресурсів

Після вибору тестового ресурсу, користувач може бачити доступні об’єкти, які виділяються при взаємодії з ними на інтерфейсі користувача (рис. 3.7 – 3.9).

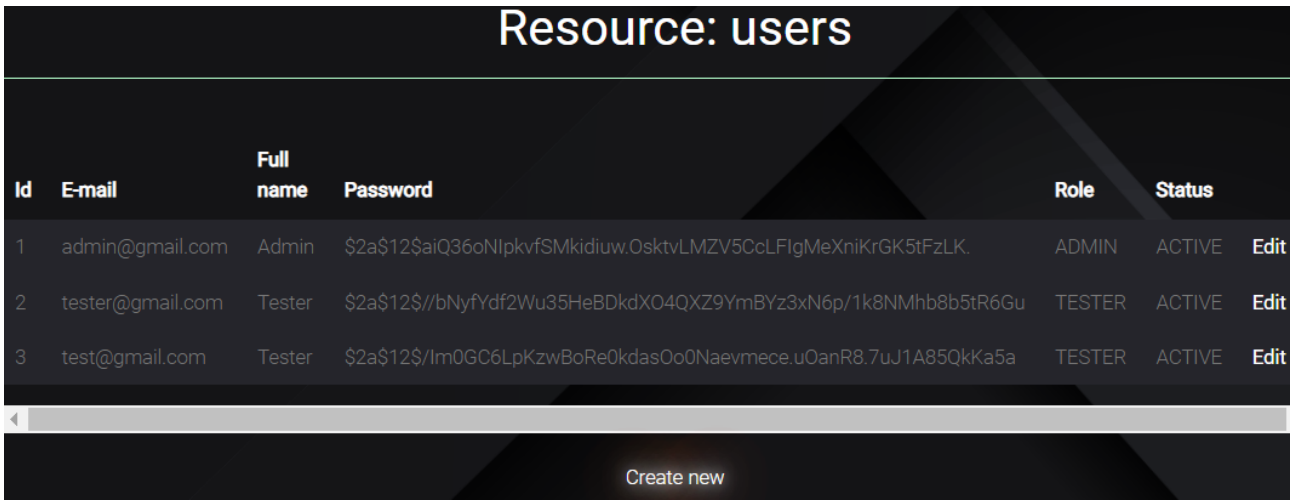

Рис. 3.7. Доступні об’єкти типу ресурса users

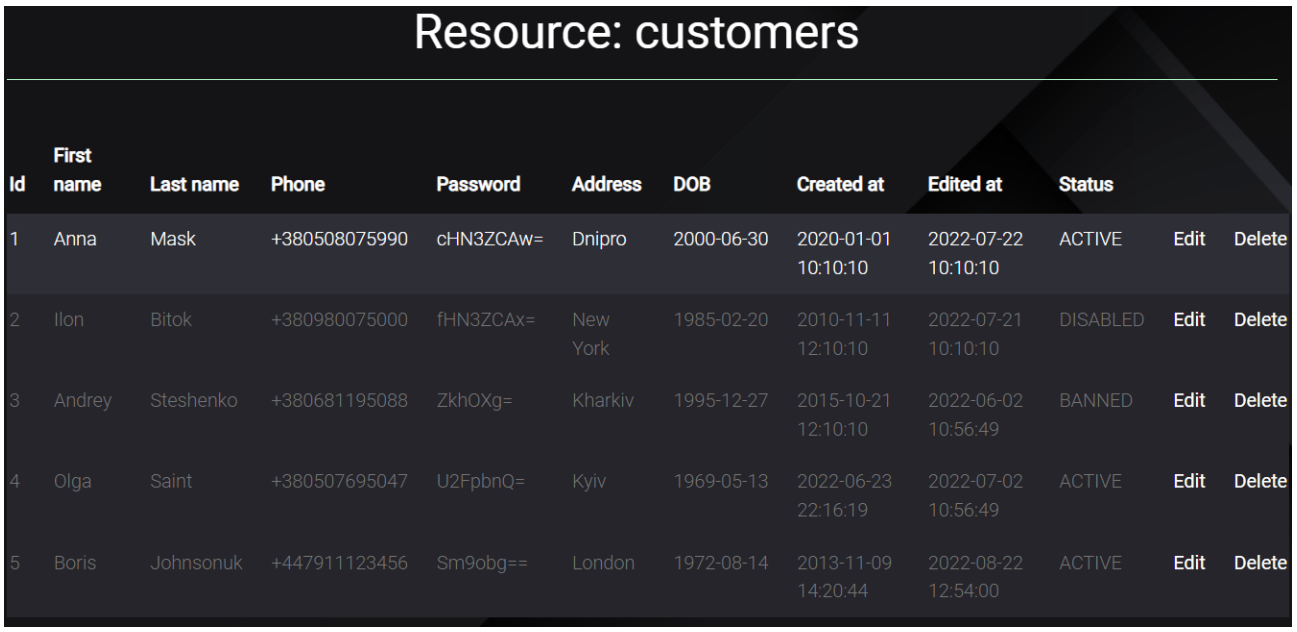

Рис. 3.8. Доступні об’єкти типу ресурса customers

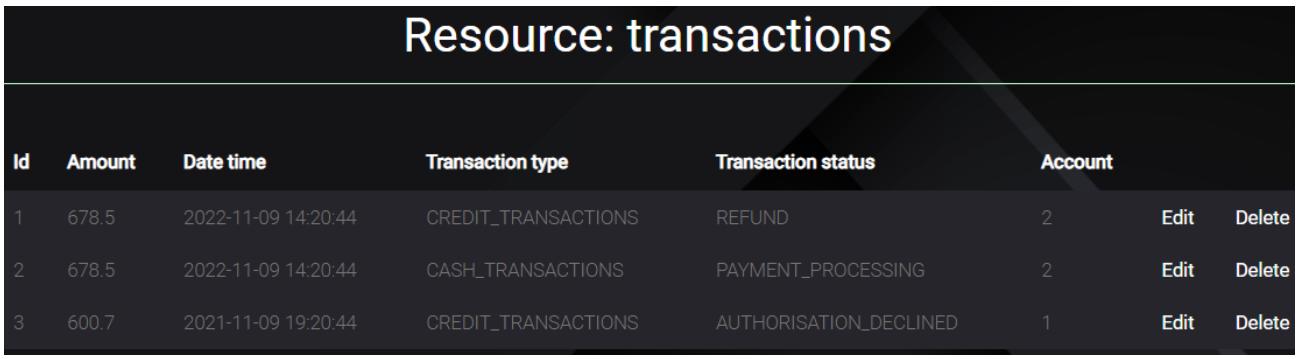

Рис. 3.9. Доступні об’єкти типу ресурса transactions

Для відкладення змін ресурсу у часі доступно дві опції (рис. 3.10):

– таймер з завданням конкретного часу виконання операції;
– затримака у будь-якій часовій одиниці.

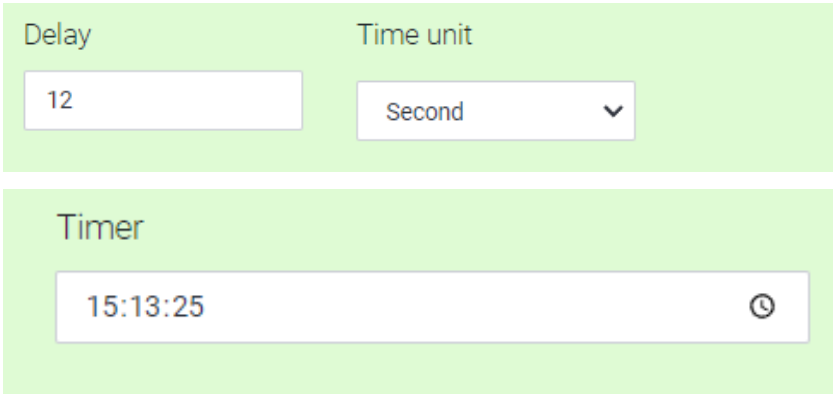

Рис. 3.10. Функціонал затримки змін

При необхідності внесення змін у ресурс, є можливість створення, редагування та видалення об’єктів з відкладенням у часі або без нього. Для створення нового ресурсу з можливим планування, сторінка буде мати такий вигляд, як на рисунках 3.11 – 3.12.

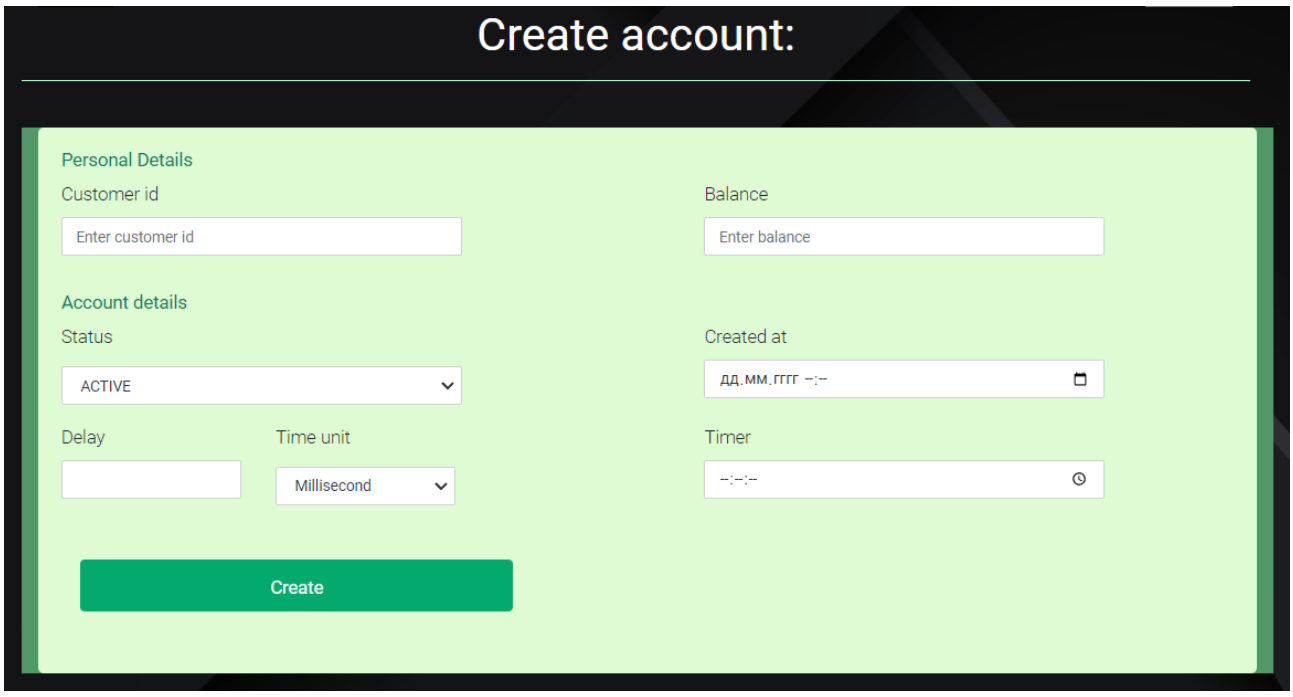

Рис. 3.11. Створення ресурсу account

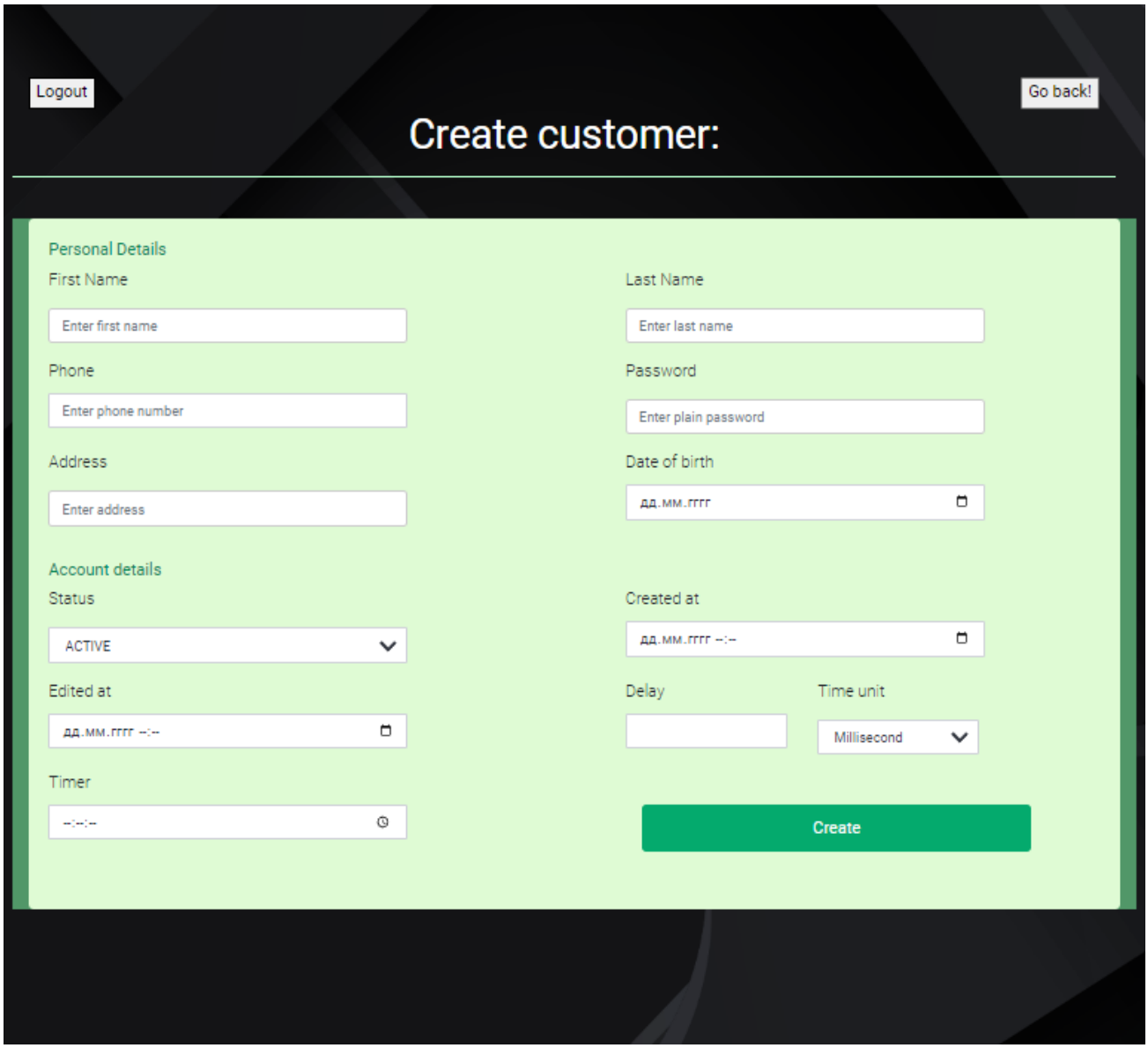

Рис. 3.12. Створення ресурсу customer

При редагуванні обраного ресурсу, id є незмінним, а всі інші параметри для зручності проставляються автоматично з вже наявної інформації (рис. 3.13 – 3.14).

Також слід зазначити, що при створенні та редагуванні ресурсу тестування з паролем користувач вводить його без шифрування заздалегідь, всі процеси проходять автоматично на серверній частині додатку, зберігаючи час клієнту.

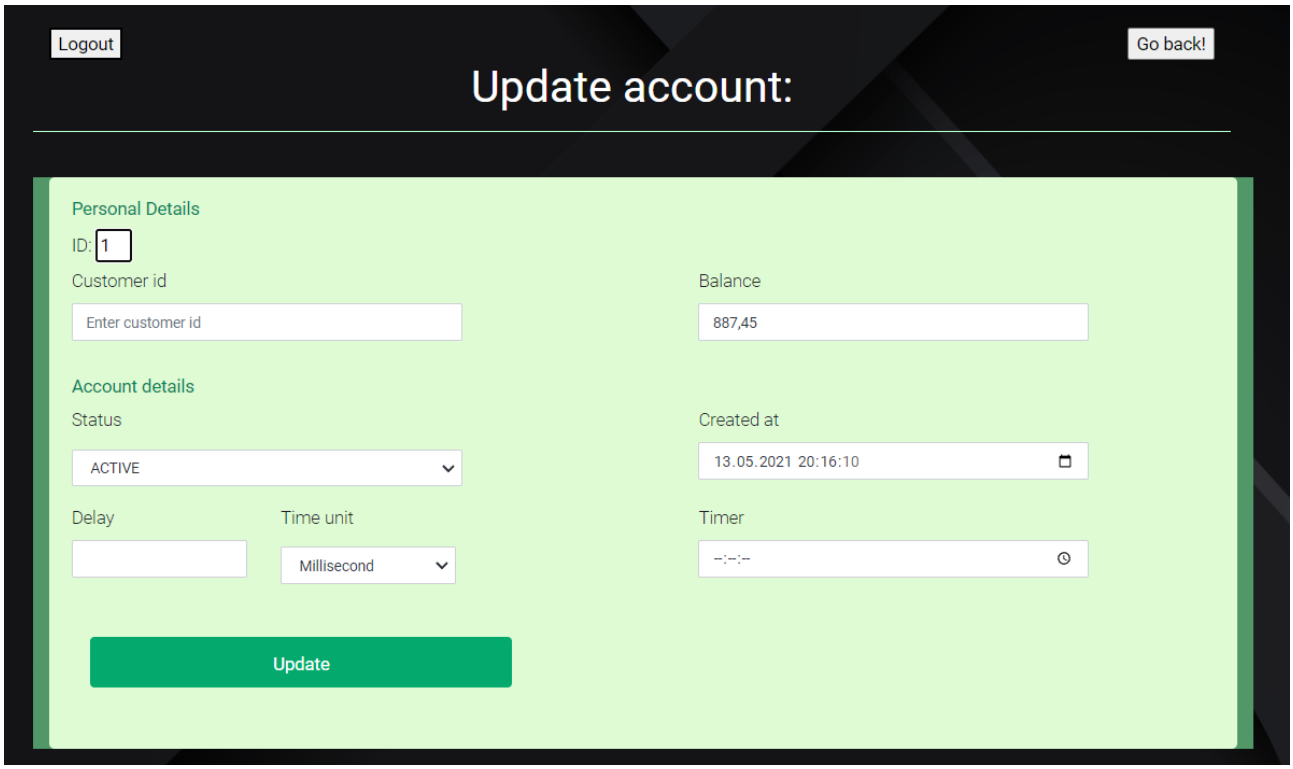

Рис. 3.13. Оновлення ресурсу account

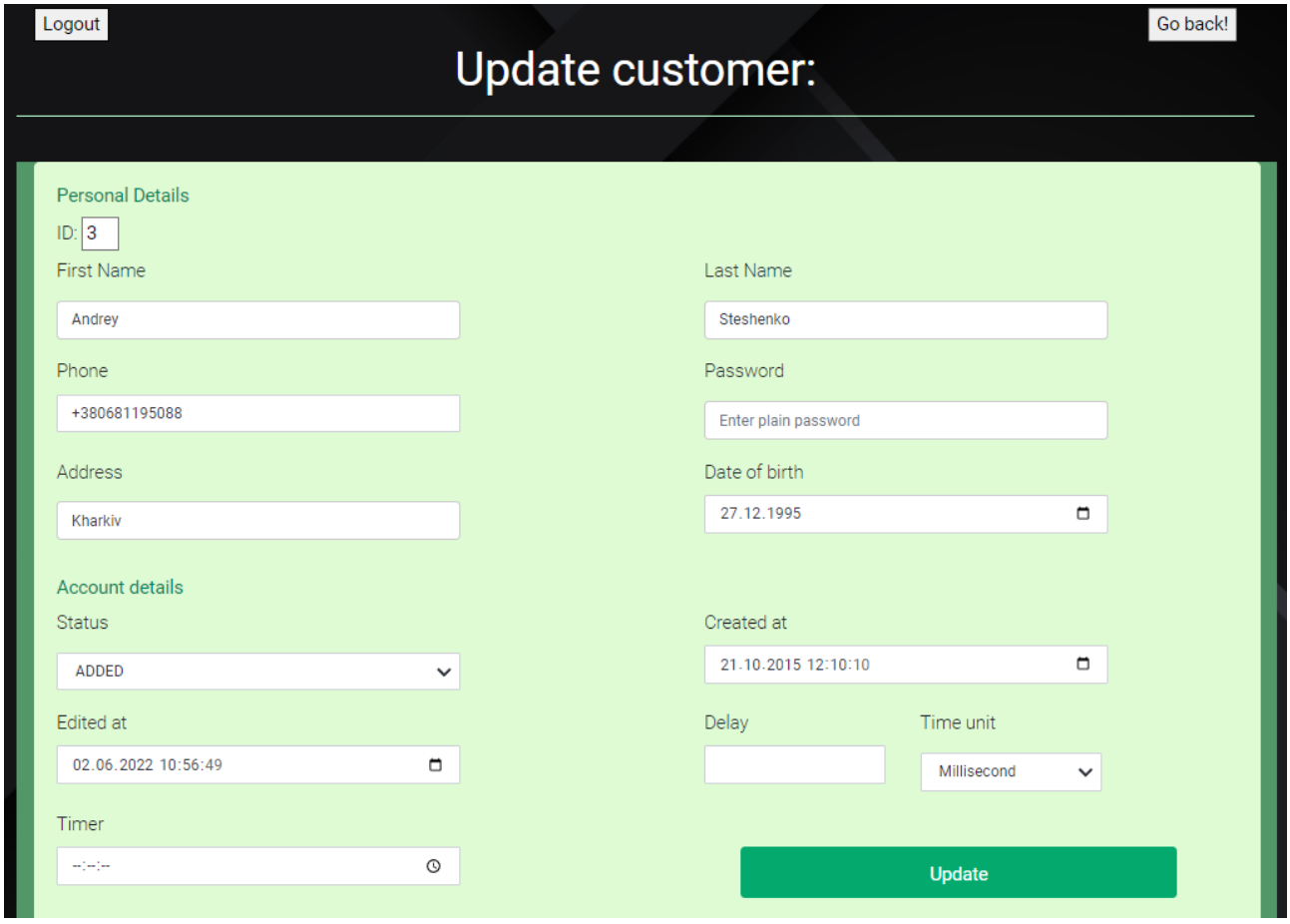

Рис. 3.14. Оновлення ресурсу customer

Якщо користувач спробую зробити дію, на яку не має належних прав або ролі, метою безпечності вихідних даних відобразиться узагальнена сторінка помилки з можливістю повернення назад.

3.4Висновки

У даному розділі було проведено проектування та розробку системи з автоматизації сміни стану ресурсів тестування testerossa, в якій було використано багато технологій, які зараз є популярними на ринку java-застосунків.

В процесі розробки було підключено JPA та використано методи інтерфейсу JpaRepository для роботи з БД. Основний акцент було побудовано на забезпеченні гнучкої системи ролей та доступів до кожної частини функціоналу. Для зміни стану ресурсів користуачу є доступним зручний інтуїтивно-зрозумілий інтерфейс, який без знання мови SQL допоможе створити, змінити або видалити потрібні ресурси. За допомогою асинхронного програмування додано можливість відкладених автоматичних змін об’єктів.

Проведено ретельне тестування додатку, в ході якого не було знайдено очевидних можливостей зламати логіку розробленої системи. Вся система зберігає можливість масштабування і додавання нової бізнес-логіки в найкоротші строки та з мінімальними витратами.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є розроблена інформаційна система для автоматизації зміни стану ресурсів тестування «testerossa», що містить клієнтську та адміністративну частини.

У цій роботі створено надійну архітектуру та її сучасну реалізацію на основі популярного та ефективного фреймворку Spring. Клієнтська частина застосунку дозволяє без роботи з кодом виконувати всі CRUD-операції з та доступним за дозволами ролі користувача ресурсом тестування. За допомогою асинхронного програмування створено швидкий та безпечний функціонал по відкладанню внесення змін за таймером або обраним часовим проміжком. Адміністративна частина є інтуїтивно-зрозумілою, спеціальної підготовки для управління ролями та статусом користувачів у системі не потребується.

Для функціонування додатку була створена і початково наповнена база даних, що надає високу оперативність за рахунок підтримки мови запитів SQL.

Розроблений додаток допоможе підвищити ефективність, швидкість та складність сценаріїв для повного тестування інформаційної системи клієнта. Розроблена основа додатку є багатофункціональною, а отже отримані результати можуть застосовуватися як у прямій практиці розробки програмного забезпечення зі зміною бізнес-логіки, так і поглиблення рівня самостійного опанування галузі. Для розробки застосунку використовувались актуальні технології програмування: Java11, Spring Boot, Spring Data JPA, MySQL, Spring Security, Lombok, JUnit, Thymeleaf, HTML5, CSS3, JavaScript та Bootstrap.

Створений проект testerossa спрямований на підвищення зручності управління тестовими ресурсами та зменшення дій для внесення змін в об'єкти тестування. Отже, він буде корисним для підтримки тестування в більшості корпоративних систем. Таким чином, в процесі виконання кваліфікаційної роботи було досягнуто всіх цілей та виконано усі вимоги, які висувалися при проектуванні інформаційної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Блох Дж. Effective Java (3-е видання). – СПб: Діалектика, 2014. – 466 с.
2. Encryption and Decryption in Java Cryptography / URL: https://www.veracode.com/blog/research/encryption-and-decryption-java-cryptography (дата звернення: 14.09.2022).
3. Дейт К. Дж. Введення в системи баз даних (8-е видання). – М.: Вільямс, 2005. – 1328с.
4. Java AES Encryption and Decryption / URL: https://www.baeldung.com/java-aes-encryption-decryption (дата звернення: 26.09.2022).
5. Walls C. Spring in Action (5th Edition) –Manning, 2018. –520 с.
6. Spilca L. Spring Security in Action –Manning, 2020. –560 с.
7. Удовик І.М., Коротенко Л.М., Шевцова О.С. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Програмне забезпечення”. – М: НТУ «Дніпровська політехніка», 2013. 16 с.
8. Кей Хорстманн. Бібліотека професіонала/ Java 11 (сьоме видання). – СПб: «Діалектика», 2020. – 936с.
9. Програмуваня по-українськи URL: programming.in.ua (дата звернення: 03.05.2021).
10. Чіртік А.В. Популярний самовчитель HTML. – СПб.: Пітер, 2012. –56 с.
11. MySQL Documentation / URL: https://dev.mysql.com/doc/ (дата звернення: 22.10.2022).
12. Thymeleaf + Spring / URL: https://www.thymeleaf.org/documentation.html (дата звернення: 28.10.2022).
13. How to secure REST with Spring Security / URL: https://www.infoworld.com/article/3630107/how-to-secure-rest-with-spring-security.html (дата звернення: 15.11.2022).
14. Heckler M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications – O'Reilly Media, 2021. – 328 с.

15. Security with Spring / URL: https://www.baeldung.com/security-spring (дата звернення: 07.11.2022).
16. Walls C. Spring Boot in Action –Manning, 2016. –264 с.
17. Scheduled Tasks Documentation / URL: https://www.aquaclusters.com/app/home/project/public/aquadatastudio/wikibook/Documentation21.0/page/Scheduled-Tasks/Scheduled-Tasks / (дата звернення: 02.11.2022).
18. Transaction Status / URL: https://epayments-support.ingenico.com/en/get-started/transaction-status-full/ (дата звернення: 15.11.2022).
19. Building RESTful Web Services / URL: https://www.tutorialspoint.com/spring_boot/spring_boot_building_restful_web_seservic.htm (дата звернення: 08.10.2022).
20. Ташков П.А. Веб-мастеринг. HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка. –М.: Пітер, 2010. –512 с.
21. Кузнєцов М.В., Сімдянов І.В. MySQL 5. Найповніший посібник. –СПб.: БХВ-Петербург, 2010. –1024 с.
22. Веллінг Л., Томсон Л. MySQL. Навчальний посібник. –СПб.: Вільямс, 2009. – 294 с.
23. Deinum M. Spring 5 Recipes – A problem-solution approach – Apress, 2017. – 870 с.
24. Cosmina I., Harrop R., Schaefer C., Ho C. Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools (5th edition) –Apress, 2017. –878с.
25. Spring scheduler – виконання коду за розкладом / URL: https://java-master.com/spring-scheduler (дата звернення: 29.11.2022).
26. Spring Security @PreAuthorize Annotation / URL: https://www.appsdeveloperblog.com/spring-security-preauthorize-annotation-example/ (дата звернення: 12.10.2022).
27. K. Siva Prasad Reddy. Beginning Spring Boot 2: Applications and Microservices with the Spring Framework – Apress, 2017. – 324 с.

28. What is Spring Framework? An Unorthodox Guide / URL: https://www.marcobehler.com/guides/spring-framework (дата звернення: 11.09.2022).
29. Benjamin Evans, Martijn Verburg, Jason Clark. The Well-Grounded Java Developer, Second Edition – Manning, 2022. – 74 с.
30. Spring vs. the World: Comparing Spring Boot Alternatives / URL: https://www.jrebel.com/blog/spring-boot-alternatives (дата звернення: 21.09.2022).
31. Moises Macero. Learn Microservices with Spring Boot: A Practical Approach to RESTful Services using RabbitMQ, Eureka, Ribbon, Zuul and Cucumber – Apress, 2017. – 344 с.
32. Neal Ford, Mark Richards, Pramod Sadalage, Zhamak Dehghani. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures – O'Reilly Media, 2021. – 459 с.
33. Java EE at a Glance/ URL: https://www.oracle.com/java/technologies/java-ee-glance.html (дата звернення: 29.09.2022).
34. Spring Boot / URL: https://spring.io/projects/spring-boot (дата звернення: 17.10.2022).
35. Spring MVC Execution Flow / URL: https://javagyansite.com//2020/03/01/spring-mvc-execution-flow/ (дата звернення: 20.10.2022).
36. Inversion of Control(IoC) in Spring / URL: https://javagyansite.com//2018/07/22/inversion-of-controlioc/ (дата звернення: 18.10.2022).
37. Spring Boot : Auto Configuration / URL: https://javagyansite.com//2020/02/17/spring-boot-auto-configuration/ (дата звернення: 11.10.2022).
38. Excellent Ways to Secure Your Spring Boot Application / URL: https://developer.okta.com/blog/2018/07/30/10-ways-to-secure-spring-boot (дата звернення: 22.11.2022).

39. Spring Framework Annotations / URL: https://springframework.guru/spring-framework-annotations/ (дата звернення: 26.02.2021).
40. Java Spring vulnerabilities / URL: https://cybersecurity.att.com/blogs/labs-research/java-spring-vulnerabilities (дата звернення: 13.09.2022).
41. Greg L. Learning Spring Boot 2.0 - Second Edition: Simplify the development of lightning fast applications based on microservices and reactive programming – Apress, 2017. – 370 с.
42. Antonov A. Spring Boot 2.0 Cookbook - Second Edition: Configure, test, extend, deploy, and monitor Spring Boot application – Packt Publishing, 2018. – 286 с.
43. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Grady Booch. Design Patterns: Elements of Reusable Object-Oriented Software – Addison-Wesley Professional, 1994. – 416 с.
44. Jim Blandy, Jason Orendorff, Leonora Tindall. Programming Rust: Fast, Safe Systems Development – O'Reilly Media, 2021. – 735 с.
45. Martin Fowler. Refactoring: Improving the Design of Existing Code (2nd Edition) – Addison-Wesley Professional, 2018. – 448 с.
46. Mark Richards. Fundamentals of Software Architecture: An Engineering Approach – O'Reilly Media, 2020. – 419 с.
47. Eric Freeman. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software 2nd Edition – O'Reilly Media, 2021. – 669 с.
48. Sam Newman. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith – O'Reilly Media, 2019. – 272 с.
49. Michael Feathers. Working Effectively with Legacy Code – Pearson, 2004. – 464 с.
50. Titus Winters, Tom Manshreck, Hyrum Wright. Software Engineering at Google: Lessons Learned from Programming Over Time – O'Reilly Media, 2020. – 599 с.
51. Merih Taze. Engineers Survival Guide: Advice, tactics, and tricks After a decade of working at Facebook, Snapchat, and Microsoft –Taze, 2021. – 245 с.
52. Sam Newman. Building Microservices: Designing Fine-Grained Systems – O'Reilly Media, 2021. – 612 с.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

SecurityConfig.java

```
package ua.testerossa.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  private final UserDetailsService userDetailsService;
  public static final String LOGIN_ENDPOINT = "/auth/login";
  public static final String LOGOUT_ENDPOINT = "/auth/logout";
  public static final String SUCCESS_ENDPOINT = "/auth/success";
  public static final String FAILURE_ENDPOINT = "/auth/error";
  @Autowired
  public SecurityConfig(@Qualifier("userDetailsServiceImpl") UserDetailsService userDetailsService) {
    this.userDetailsService = userDetailsService;
  }
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/auth/*").permitAll()
```

```
        .anyRequest()
        .authenticated()
        .and()
        .formLogin()
        .loginPage(LOGIN_ENDPOINT).permitAll()
        .failureUrl(FAILURE_ENDPOINT).permitAll()
        .defaultSuccessUrl(SUCCESS_ENDPOINT)
        .and()
        .logout()
        .logoutRequestMatcher(new AntPathRequestMatcher(LOGOUT_ENDPOINT, "POST"))
        .invalidateHttpSession(true)
        .clearAuthentication(true)
        .deleteCookies("JSESSIONID")
        .logoutSuccessUrl(LOGIN_ENDPOINT);
  }
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(daoAuthenticationProvider());
  }
  @Bean
  protected PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(12);
  }
  @Bean
  protected DaoAuthenticationProvider daoAuthenticationProvider() {
    DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
    daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
    daoAuthenticationProvider.setUserDetailsService(userDetailsService);
    return daoAuthenticationProvider;
  }
  @Override
  public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers("/resources/**", "/static/**");
  }
}
```

UserDetails.java

```
package ua.testerossa.security;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;
```

```
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import ua.testerossa.model.Status;
import ua.testerossa.model.dto.User;
import java.util.Collection;
import java.util.List;

@Data
@Slf4j
public class SecurityUser implements UserDetails {
    private final String username;
    private final String password;
    private final List<SimpleGrantedAuthority> authorities;
    private final boolean isActive;

    public SecurityUser(String username, String password, List<SimpleGrantedAuthority> authorities,
boolean isActive) {
        this.username = username;
        this.password = password;
        this.authorities = authorities;
        this.isActive = isActive;
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }
    @Override
    public String getPassword() {
        return password;
    }
    @Override
    public String getUsername() {
        return username;
    }
    @Override
    public boolean isAccountNonExpired() {
```

```
        return isActive;
    }
    @Override
    public boolean isAccountNonLocked() {
        return isActive;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return isActive;
    }
    @Override
    public boolean isEnabled() {
        return isActive;
    }
    public static UserDetails fromUser(User user) {
        log.info("fromUser user:{}, authorities:{}", user.toString(), user.getRole().getAuthorities());
        return new org.springframework.security.core.userdetails.User(
                user.getEmail(), user.getPassword(),
                user.getStatus().equals(Status.ACTIVE),
                user.getStatus().equals(Status.ACTIVE),
                user.getStatus().equals(Status.ACTIVE),
                user.getStatus().equals(Status.ACTIVE),
                user.getRole().getAuthorities()
        );
    }
}
```

SecurityUtils.java

```
package ua.testerossa.utils;
import com.sun.mail.util.BASE64EncoderStream;
import lombok.extern.slf4j.Slf4j;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
```

```
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.spec.KeySpec;
import java.util.Arrays;
import java.util.Base64;

@Slf4j
public class SecurityUtils {
  public static final String SECRET_KEY_64 = "KaPdSgVk";
  public static final String SECRET_KEY_128 = "cRfUjXnZr4u7x!A%";
  public static final String SECRET_KEY_256 = "q4t7w!z%C*F-JaNcRfUjXn2r5u8x/A?D";
  public static final String SECRET_KEY_512 = "WmZq4t7w!z%C*F-
JaNcRfUjXn2r5u8x/A?D(G+KbPeSgVkYp3s6v9y$B&E)H@McQ";
  private static final String SALT = "ssshhhhhhhhhhh!!!!";

  public static String des(String strToEncrypt) {
    try {
      SecretKey key = KeyGenerator.getInstance("DES").generateKey();
      Cipher ecipher = Cipher.getInstance("DES");
      ecipher.init(Cipher.ENCRYPT_MODE, key);
      byte[] utf8 = strToEncrypt.getBytes(StandardCharsets.UTF_8);
      byte[] enc = ecipher.doFinal(utf8);
      enc = BASE64EncoderStream.encode(enc);
      return new String(enc);
    } catch (Exception e) {
      System.out.println("Error occured during des encryption: " + e.toString());
    }
    return null;
  }
  public static String tripleDes(String strToEncrypt) {
    try {
      final MessageDigest md = MessageDigest.getInstance("md5");
      final byte[] digestOfPassword = md.digest(SECRET_KEY_128.getBytes(StandardCharsets.UTF_8));
      final byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);
      for (int j = 0, k = 16; j < 8; ) {
        keyBytes[k++] = keyBytes[j++];
```

```
    }
    final SecretKey key = new SecretKeySpec(keyBytes, "DESede");
    final Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] plainTextBytes = strToEncrypt.getBytes(StandardCharsets.UTF_8);
    return Base64.getEncoder()
        .encodeToString(cipher.doFinal(plainTextBytes));
  } catch (Exception e) {
    System.out.println("Error occured during des encryption: " + e.toString());
  }
  return null;
}
public static String blowfish(String strToEncrypt) {
  try {
    byte[] KeyData = SECRET_KEY_128.getBytes(StandardCharsets.UTF_8);
    SecretKeySpec KS = new SecretKeySpec(KeyData, "Blowfish");
    Cipher cipher = Cipher.getInstance("Blowfish");
    cipher.init(Cipher.ENCRYPT_MODE, KS);
    return Base64.getEncoder().
        encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_8)));
  } catch (Exception e) {
    System.out.println("Error occured during blowfish encryption: " + e.toString());
  }
  return null;
}
public static String md5(String strToEncrypt) {
  try {
    StringBuilder code = new StringBuilder();
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");
    byte[] bytes = strToEncrypt.getBytes(StandardCharsets.UTF_8);
    byte[] digest = messageDigest.digest(bytes);
    for (byte aDigest : digest) {
      code.append(Integer.toHexString(0x0100 + (aDigest & 0x00FF)).substring(1));
    }
    return code.toString();
  } catch (Exception e) {
    System.out.println("Error occured during md5 encryption: " + e.toString());
```

```
  }
  return null;
}
public static String sha256(String strToEncrypt) {
  try {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(strToEncrypt.getBytes(StandardCharsets.UTF_8));
    return Base64.getEncoder().
        encodeToString(hash);
  } catch (Exception e) {
    System.out.println("Error occured during sha256 encryption: " + e.toString());
  }
  return null;
}
public static String sha512(String passwordToHash) {
  try {
    MessageDigest md = MessageDigest.getInstance("SHA-512");
    byte[] bytes = md.digest(passwordToHash.getBytes(StandardCharsets.UTF_8));
    StringBuilder sb = new StringBuilder();
    for (byte aByte : bytes) {
      sb.append(Integer.toString((aByte & 0xff) + 0x100, 16).substring(1));
    }
    return sb.toString();
  } catch (Exception e) {
    System.out.println("Error occured during sha512 encryption: " + e.toString());
  }
  return null;
}
public static String aes(String strToEncrypt, String key) {
  log.info("aes: strToEncrypt={}", strToEncrypt);
  try {
    byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    IvParameterSpec ivspec = new IvParameterSpec(iv);
    SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
    KeySpec spec = new PBEKeySpec(key.toCharArray(), SALT.getBytes(), 65536, 256);
    SecretKey tmp = factory.generateSecret(spec);
    SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
```

```
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
    return Base64.getEncoder()
        .encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_8)));
   } catch (Exception e) {
    System.out.println("Error while encrypting: " + e.toString());
   }
   return null;
  }
  public static String decrypt(String strToDecrypt, String key) {
   log.info("decrypt: strToEncrypt={}", strToDecrypt);
   try {
    byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    IvParameterSpec ivspec = new IvParameterSpec(iv);
    SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
    KeySpec spec = new PBEKeySpec(key.toCharArray(), SALT.getBytes(), 65536, 256);
    SecretKey tmp = factory.generateSecret(spec);
    SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
    cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);
    return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
   } catch (Exception e) {
    System.out.println("Error while decrypting: " + e.toString());
   }
   return null;
  }
}
```

style.java

```
package ua.testerossa.controller;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```
import org.springframework.web.bind.annotation.PostMapping;
import ua.testerossa.model.Timer;
import ua.testerossa.model.dto.Customer;
import ua.testerossa.service.CustomerService;
import ua.testerossa.utils.SecurityUtils;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;

@Slf4j
@Controller
public class CustomerRestController {
 private final CustomerService customerService;
 @Autowired
 public CustomerRestController(CustomerService customerService) {
  this.customerService = customerService;
 }
 @GetMapping("/customers")
 @PreAuthorize("hasAuthority('read')")
 public String findAll(Model model) {
  List<Customer> customers = customerService.findAll();
  model.addAttribute("customers", customers);
  return "customer-list";
 }
 @GetMapping("/customer-create")
 @PreAuthorize("hasAnyAuthority('read', 'create')")
 public String createCustomerForm(Model model) {
  model.addAttribute("customer", new Customer());
  model.addAttribute("timer", new Timer());
  return "customer-create";
 }

 @PostMapping("/customer-create")
 @PreAuthorize("hasAnyAuthority('read', 'create')")
 public String createCustomer(Customer customer, Timer timer) {
  log.info("createCustomer:customer={}, timer={}", customer.toString(), timer.toString());
  customer.setPassword(SecurityUtils.aes(customer.getPassword(), SecurityUtils.SECRET_KEY_256));
  //delay
  if (timer.getDelayTime() != 0) {
   CompletableFuture.runAsync(() -> {
    try {
```

```
        log.info("createCustomer delay: {}", timer.getDelayTime());
        TimeUnit.MILLISECONDS.sleep(timer.getDelayTime());
        customerService.saveCustomer(customer);
       } catch (InterruptedException ex) {
        log.error("Exception on createCustomer: {}" + ex.getMessage());
       }
     });
   } else {
     customerService.saveCustomer(customer);
   }
   return "redirect:/customers";
}
@GetMapping("/customer-update/{id}")
@PreAuthorize("hasAnyAuthority('read', 'update')")
public String updateCustomerForm(@PathVariable("id") Long id, Model model) {
   Customer customer = customerService.findById(id);
   customer.setPassword(SecurityUtils.decrypt(customer.getPassword(), SecurityUtils.SECRET_KEY_256));
   model.addAttribute("customer", customer);
   model.addAttribute("timer", new Timer());
   return "customer-update";
}
@PostMapping("/customer-update")
@PreAuthorize("hasAnyAuthority('read', 'update')")
public String updateCustomer(Customer customer, Timer timer) {
   log.info("updateCustomer:customer={}, timer={}", customer.toString(), timer.toString());
   //encode
   customer.setPassword(SecurityUtils.aes(customer.getPassword(), SecurityUtils.SECRET_KEY_256));
   //delay update
   if (timer.getDelayTime() != 0) {
     CompletableFuture.runAsync(() -> {
       try {
         log.info("updateCustomer delay: {}", timer.getDelayTime());
         TimeUnit.MILLISECONDS.sleep(timer.getDelayTime());
         customerService.saveCustomer(customer);
       } catch (InterruptedException ex) {
         log.error("Exception on updateCustomer: {}" + ex.getMessage());
       }
     });
   } else {
     customerService.saveCustomer(customer);
   }
```

```
    return "redirect:/customers";
  }
  @GetMapping("customer-delete/{id}")
  @PreAuthorize("hasAnyAuthority('read', 'delete')")
  public String deleteCustomer(@PathVariable("id") Long id) {
    customerService.deleteById(id);
    return "redirect:/customers";
  }
}
```

transaction-update.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link href="https://fonts.googleapis.com/css?family=Roboto:300,400&display=swap" rel="stylesheet">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" type="text/css" th:href="@{/resources/css/bootstrap.min.css}">
    <!-- Style -->
    <link rel="stylesheet" type="text/css" th:href="@{/resources/css/style.css}">
    <link rel="shortcut icon" th:href="@{/resources/images/logo.png}" type="image/png">
    <title>Update transaction</title>
</head>
<body>
<div class="content">
    <div class="container">
        <div class="container">
            <div class="container">
                <div class="row">
                    <div class="col-sm">
                        <form action="/auth/logout" method="POST">
                            <button type="submit">Logout</button>
                        </form>
                    </div>
                    <div class="col-sm">
                        <form>
                            <input type="button" value="Go back!" onclick="history.back()" class="right">
```

```
                </form>
            </div>
        </div>
    </div>
    <h2>Update transaction:</h2>
    <hr>
    <br>
    <div class="custom-form col-xl-9 col-lg-9 col-md-12 col-sm-12 col-12">
        <div class="card h-100">
            <div class="card-body">
                <form action="#" th:action="@{/transaction-update}" th:object="${transaction}"
method="post"
                      class="table custom-table">
                    <div class="row gutters">
                        <div class="col-xl-12 col-lg-12 col-md-12 col-sm-12 col-12">
                            <h6 class="form-title mb-2 text-primary">Personal Details</h6>
                            <label for="id"> ID:</label>
                            <input readonly type="number" th:field="*{transactionId}" id="id" size="5">
                        </div>
                        <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                            <div class="form-group">
                                <label for="account">Account id</label>
                                <input th:field="*{account}" type="number" class="form-control" id="account"
                                       placeholder="Enter account id">
                            </div>
                        </div>
                    </div>
                    <div class="row gutters">
                        <div class="col-xl-12 col-lg-12 col-md-12 col-sm-12 col-12">
                            <h6 class="mt-3 mb-2 text-primary">Transaction details</h6>
                        </div>
                        <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                            <div class="form-group">
                                <label for="amount">Amount</label>
                                <input th:field="*{amount}" type="number" class="form-control" id="amount"
                                       placeholder="Enter amount">
                            </div>
```

```
                </div>
                <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                    <div class="form-group">
                        <label for="type">Status</label>
                        <select th:field="*{type}" class="form-control" id="type">
                            <option value="CASH_TRANSACTIONS">CASH TRANSACTIONS</option>
                            <option value="NON_CASH_TRANSACTIONS">NON CASH
TRANSACTIONS</option>
                            <option value="CREDIT_TRANSACTIONS">CREDIT
TRANSACTIONS</option>
                        </select>
                    </div>
                </div>
                <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                    <div class="form-group">
                        <label for="dateTime">Date time</label>
                        <input th:field="*{dateTime}" type="datetime-local" class="form-control"
                            id="dateTime">
                    </div>
                </div>
                <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                    <div class="form-group">
                        <label for="status">Status</label>
                        <select th:field="*{status}" class="form-control" id="status">
                            <option value="INVALID_OR_INCOMPLETE">INVALID OR
INCOMPLETE</option>
                            <option value="CANCELLED_BY_CUSTOMER">CANCELLED BY
CUSTOMER</option>
                            <option value="AUTHORISATION_DECLINED">AUTHORISATION
DECLINED</option>
                            <option value="WAITING_FOR_CLIENT_PAYMENT">WAITING FOR
CLIENT PAYMENT</option>
                            <option
value="AUTHORISED_WAITING_EXTERNAL_RESULT">AUTHORISED WAITING EXTERNAL
RESULT</option>
                            <option value="AUTHORISATION_WAITING">AUTHORISATION
WAITING</option>
```

```
                        <option value="AUTHORISATION_NOT_KNOWN">AUTHORISATION
NOT KNOWN</option>
                        <option value="PAYMENT_DELETED">PAYMENT DELETED</option>
                        <option value="REFUND">REFUND</option>
                        <option value="BEING_PROCESSED">BEING PROCESSED</option>
                        <option value="PAYMENT_REFUSED">PAYMENT REFUSED</option>
                        <option value="PAYMENT_PROCESSING">PAYMENT
PROCESSING</option>
                        <option
value="PAYMENT_REQUESTED">PAYMENT_REQUESTED</option>
                        <option value="REFUND_DECLINED_BY_ACQUIRER">REFUND
DECLINED BY ACQUIRER</option>
                        <option value="PAYMENT_PROCESSED_BY_MERCHANT">PAYMENT
PROCESSED BY MERCHANT</option>
                    </select>
                </div>
            </div>
            <div class="col-xl-2 col-lg-2 col-md-2 col-sm-2 col-4">
                <div class="form-group">
                    <div th:object="${timer}">
                        <label for="delay">Delay</label>
                        <input th:field="*{delay}" type="number" class="form-control"
                            id="delay">
                    </div>
                </div>
            </div>
            <div class="col-xl-2 col-lg-2 col-md-2 col-sm-2 col-4">
                <div class="form-group">
                    <div th:object="${timer}">
                        <label for="timeUnit">Time unit</label>
                        <select th:field="*{timeUnit}" class="form-control" id="timeUnit">
                            <option value="MILLISECOND">Millisecond</option>
                            <option value="SECOND">Second</option>
                            <option value="MINUTE">Minute</option>
                            <option value="HOUR">Hour</option>
                            <option value="DAY">Day</option>
                            <option value="WEEK">WEEK</option>
```

```
                        <option value="MONTH">Month</option>
                        <option value="YEAR">Year</option>
                    </select>
                </div>
            </div>
            <div class="col-xl-2 col-lg-2 col-md-2 col-sm-2 col-4">
            </div>
            <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                <div class="form-group">
                    <div th:object="${timer}">
                        <label for="timer">Timer</label>
                        <input th:field="*{time}" type="time" class="form-control" step="1"
                            id="timer">
                    </div>
                </div>
            </div>
            <div class="col-xl-6 col-lg-6 col-md-6 col-sm-6 col-12">
                <div class="row gutters">
                    <br><br><br><br>
                    <div class="text-lg-center">
                        <input type="submit" value="Update">
                    </div>
                </div>
            </div>
        </div>
    </form>
</div>
</div>
</div>
</div>
<script th:href="@{/resources/js/jquery-3.3.1.min.js}"></script>
<script th:href="@{/resources/js/bootstrap.min.js}"></script>
</body>
</html>
```

style.css

@importurl('https://fonts.googleapis.com/css?family=Poppins:400,500,600,700,800,900');body{font-family:"Roboto",-apple-system,BlinkMacSystemFont,"SegoeUI",Roboto,"HelveticaNeue",Arial,"NotoSans",sans-serif,"AppleColorEmoji","SegoeUIEmoji","SegoeUISymbol","NotoColorEmoji";background-image:url("../../resources/images/2-back.png");background-color:#19191d;font-weight:300;}p{color:#b3b3b3;font-weight:300;text-align:center;}h1,h2,h3,h4,h5,h6,.h1,.h2,.h3,.h4,.h5,.h6{font-family:"Roboto",-apple-system,BlinkMacSystemFont,"SegoeUI",Roboto,"HelveticaNeue",Arial,"NotoSans",sans-serif,"AppleColorEmoji","SegoeUIEmoji","SegoeUISymbol","NotoColorEmoji";}a{-webkit-transition:.3sallease;-o-transition:.3sallease;transition:.3sallease;}a,a:hover,a:active{text-decoration:none!important;color:white;}.content{padding:7rem0;}h2{font-size:40px;color:#fff;text-align:center;}.custom-table{min-width:1100px;}.custom-tabletheadtr,.custom-tabletheadth{border-top:none;border-bottom:none!important;color:#fff;}.custom-tabletbodyth,.custom-tabletbodytd{color:#777;padding-bottom:10px;padding-top:10px;font-weight:300;}.custom-tabletbodythsmall,.custom-tabletbodytdsmall{color:#b3b3b3;font-weight:300;}.custom-tabletbodytr:not(.spacer){border-radius:7px;overflow:hidden;-webkit-transition:.3sallease;-o-transition:.3sallease;transition:.3sallease;}.custom-tabletbodytr:not(.spacer):hover{-webkit-box-shadow:02px10px-5pxrgba(0,0,0,0.1);box-shadow:02px10px-5pxrgba(0,0,0,0.1);}.custom-tabletbodytrth,.custom-tabletbodytrtd{background:#25252b;border:none;-webkit-transition:.3sallease;-o-transition:.3sallease;transition:.3sallease;}.custom-tabletbodytrtha,.custom-tabletbodytrtda{color:#ffffff;font-weight:bolder;}.custom-tabletbodytrth:first-child,.custom-tabletbodytrtd:first-child{border-top-left-radius:0px;border-bottom-left-radius:0px;}.custom-tabletbodytrth:last-child,.custom-tabletbodytrtd:last-child{border-top-right-radius:0px;border-bottom-right-radius:0px;}.custom-tabletbodytr.spacertd{padding:0!important;height:1px;border-radius:0!important;background:transparent!important;}.custom-tabletbodytr.activeth,.custom-tabletbodytr.activetd,.custom-tabletbodytr:hoverth,.custom-tabletbodytr:hovertd{color:#fff;background:#2e2e36;}.custom-tabletbodytr.activetha,.custom-tabletbodytr.activetda,.custom-tabletbodytr:hovertha,.custom-tabletbodytr:hovertda{color:#fff;}hr{border-top:1pxsolid#b5ffcc;}.glow{color:#fff;text-align:center;animation:glow6sease-in-outinfinitealternate;}a.glow{text-align:center;font-size:80px;font-weight:400;animation:glow4sease-in-outinfinitealternate;}@-webkit-keyframesglow{from{text-shadow:0010px#fff,0020px#fff,0030px#03412a,0040px#04aa6d,0050px#e60073,0060px#e60073,0070px#e60073;}to{text-shadow:0020px#fff,0030px#04aa6d,0040px#04aa6d,0050px#04aa6d,0060px#ff4da6,0070px#ff4da6,0080px#ff4da6;}}/*createform*/*{box-sizing:border-box;}input[type=text],select,textarea{width:100%;padding:12px;border:1pxsolid#ccc;border-radius:4px;box-sizing:border-box;margin-top:6px;margin-bottom:16px;resize:vertical;}input[type=submit]{background-color:#04AA6D;color:white;padding:12px20px;border:none;border-radius:4px;cursor:pointer;margin-left:2rem;margin-top:2rem;width:380px;height:45px;}input[type=number]:read-only{width:3%!important;}input[type=submit]:hover{background-color:#257344;}.container-create{border-radius:5px;background-color:#f2f2f2;padding:20px;}/*createform*/.custom-form{min-width:1100px;background-color:#519768;}.form-control{border:1pxsolid#cfd1d8;-webkit-border-radius:2px;-moz-border-radius:2px;border-radius:2px;font-size:.825rem;background:#ffffff;color:#2e323c;}.card{background:#ffffff;-webkit-border-radius:5px;-moz-border-radius:5px;border-radius:5px;border:0;margin-bottom:1rem;}.text-primary{color:#048555!important;font-

weight:500!important;}div.card.h-100{background-color:#dffbd4;}.form-groupinput,.form-groupselect{max-width:22rem;}input.right{float:right;text-align:right;}

ДОДАТОК Б

ВІДГУК КЕРІВНИКА

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Наукового керівника Мещерякова Леоніда Івановича, д.т.н., проф. каф. ПЗКС
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

на магістерську роботу

студента Стешенко Анни Андріївни
(прізвище, ім'я, по батькові)

курсу І групи 122м-21-1
спеціальності 122 Комп'ютерні науки
освітньої програми «122 Комп'ютерні науки»
на тему Розробка програмного забезпечення для автоматизації зміни стану ресурсів тестування

Актуальність теми Розглянута магістерська кваліфікаційна робота присвячена знаходженню та реалізації нового розширеного методу проведення мануального тестування. Даний підхід є перспективним у тестуванні в корпоративних системах завдяки запровадженню належного рівня безпеки інформації. На сьогоднішній момент сфера розробки, а разом із нею тестування, розвивається швидкими темпами, таким чином магістерська робота відзначається актуальністю.

Мета досліджень Полягає в підвищенні автоматизованості, ефективності, швидкості та доступної для користувача варіативності сценаріїв для повного тестування інформаційних систем, за допомогою використання мови Java та можливостей фреймворку Spring.

Коротка характеристика розділів роботи Перший розділ роботи складається

з аналітичного розбору існуючих методик тестування та створення програмного забезпечення за темою магістерської роботи. Другий розділ містить огляд шляхів виконання наймасштабніших завдань забезпечення реалізації створюваного застосунку. Третій розділ присвячено тонкощам програмної реалізації застосунку для управління ресурсами тестування.

Практичне значення роботи Отримані результати роботи є актуальними у проектуванні додатків з управління ресурсами тестування в корпоративних системах. Також надані результати дослідження є підставою для більш поглибленого вивчення проблем, пов'язаних з використанням ресурсів у мануальному тестуванні.

Зауваження та недоліки В роботі недостатньо повно оглянуто реалізацію програмного інтерфейсу користувача, а також процес реєстрації в системі може бути поліпшено відправкою листа.

Висновки та оцінка Магістром було проведено порівняльний аналіз існуючих засобів для зміни стану ресурсів тестування, виявлено недоліки та реалізовано програмний апарат для вирішення досліджених проблем. Під час виконання магістерської кваліфікаційної роботи студентка Стешенко А.А. постала кваліфікованим та впевненим спеціалістом, який знаходить оптимальні рішення у складних технічних питаннях. Вважаю, що магістерська кваліфікаційна робота заслуговує оцінку «відмінно», а Стешенко А. А. – присвоєння кваліфікації «магістра» по спеціальності комп'ютерних наук.

Науковий керівник Мещеряков Л.І., док. техн. наук, проф., проф. каф. ПЗКС

(прізвище, ім'я, по батькові, посада, місце роботи)

«___»_____20__ р. _____

(підпис)

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Стешенко_А.А.doc	Пояснювальна записка до проекту. Документ Word.
Диплом_ Стешенко_А.А.pdf	Пояснювальна записка до проекту в форматі PDF.
Програма	
Program.rar	Архів, що містить коди програми для запуску проекту.
Презентація	
Презентація_Стешенко.ppt	Презентація для проекту.