DOI: https://doi.org/10.33216/1998-7927-2024-285-5-5-9

УДК 004.41

ПОРІВНЯННЯ ДВОХ ПІДХОДІВ ДО UNIT-ТЕСТУВАННЯ ANGULAR-ДОДАТКІВ

Полупан Ю.В., Родіонов П.Ю., Дьомін М.К.

COMPARISON OF TWO APPROACHES TO UNIT TESTING ANGULAR APPS

Polupan Yu.V., Rodionov P.Yu., Domin M.K.

В сучасному світі реактивні системи набирають все більшої популярності, при цьому в основі реактивності є робота з асинхронними потоками даних. Тестування реактивних систем стоїть окремою задачею, в якій є низка відкритих питань, наприклад, вимірювання ефективності тестів, використовуючі той чи інший підхід до тестування.

Для тестування Angular-додатків в роботі було обрано два підходи: 1) з використанням зв'язки Jasmine + Karma, що встановлюється за замовчуванням при створенні додатку; 2) з використанням Jest.

В статті тестується Angular-додаток, який має 7 тестів, 3 з яких – тести синхронних блоків коду і 4 – для асинхронних блоків. При тестуванні використовувались інструменти, що пропонує Angular із коробки та фреймворк Jest.

Для вимирювання ефективності Unit-тестів, як правило, використовуються чотири критерії [2, 4]: захист від помилок (protection against bugs); стійкість до рефакторингу (resilience to refactoring); швидкість зворотнього зв'язку (fast feedback); простота підтримки (maintain ability). Порівнюючи різні підходи до тестування можна визначити як той чи інший підхід впливає на конкретний критерій ефективності Unit-тестів, підвищуючі його, або навпаки зменшуючі. Як відомо, на сьогодні не існує автоматичного способу вимірювання ефективності з отриманням значень для кожного критерію і тому кожен тест потрібно оцінювати окремо. Для отримання значення по критерію «швидкість зворотнього зв'язку» будемо використовувати час виконання тестів, в тому числі з визначенням відсотка покриття коду тестами при допомозі команди coverage. Цей критерій показує, що чим швидше виконується тест на етапі розробки, тим меньше часу витрачається на усунення знайденних помилок, тим самим, значно підвищуючи ефективність Unit-тестів.

В результаті порівняння роботи двох фреймворків для тестування, можно зробити висновок, що при використанні Jest тести виконуються в середньому на 60% швидше за рахунок того, що система відстежує зміни файлів та не виконує непотрібних дій. При тестуванні ассинхронного коду така ж тенденція (збільшення швидкості тестування на 60%) зберігається, тому Jest відмінно підходить для виконання тестування Angular-проектів, які містять багато асинхронного коду.

Так як ефективність Unit-тестів залежить від швидкості зворотнього зв'язку (fast feedback), то, використовуючі Jest, ми, тим самим, значно підвищуємо ефективність Unit-тестування та всього проекту вцілому.

Ключові слова: *Unit-тестування, Angular-додаток, Jasmine, Karma, Jest, модульні тести, --test coverage, Angular CLI, ChromeHeadless, реактивність, синхронний та асинхронний код.*

Вступ

В останні десятиліття тестування програмного забезпечення розвивається дуже швидко. Раніше була доступна лише невелика кількість інструментів для цього, але тепер є величезний вибір, в якому розробники не обмежені і від їхніх поглядів на ПЗ, на ризики, тестові підходи та стратегії залежить на скільки якісний продукт буде випущений у результаті. Існуючі технології безперечно впливають на підхід до тестування, і поки їхній розвиток буде на належному рівні, цей вплив продовжуватиметься.

Види тестування QA включають безліч методів, які допомагають переконатися, що зміни в

коді працюють належним чином або, навпаки, є якісь помилки. Сам процес може здійснюватися на будь-якому етапі розробки. Зараз можна виділити 6 основних видів тестування, які допоможуть виявити помилки, недоробки та невідповідності вимогам, а також оцінити рівень якості програмного забезпечення перед його випуском [1, 2]:

1. Модульні тести, блочне тестування або юніт-тестування (англ. unit testing) - тести дуже низького рівня, які близькі до вихідного коду програми. Полягають у перевірці окремих методів та функцій, компонентів чи модулів. Їх можна легко і дешево автоматизувати, при цьому можуть швидко виконуватися за допомогою сервера безперервної інтеграції.

2. Інтеграційні тести – перевіряють, як різні модулі спільно працюють. Наприклад, взаємодіють із базою даних.

3. Функціональне тестування програмного забезпечення – фокусується на бізнес-вимогах додатків. Необхідно, щоб перевірити результат дії та не перевіряти проміжні стани системи під час його отримання.

4. Наскрізне тестування - відтворює поведінку користувача з програмним забезпеченням. Перевіряє, наскільки різні запити користувача реалізуються правильно і просто.

5. Приймальне тестування – формальна частина. Перевіряє, чи задовольняє система бізнес-вимогам. Для їх проведення потрібно, щоб програмне забезпечення повністю працювало.

6. Тестування продуктивності - проводиться для оцінки, як система працює при певному робочому навантаженні. Дозволяє виміряти надійність, швидкість та можливість для масштабування програми.

Всі методи та етапи тестування потрібні для загальної працездатності програми, перевірки програми на можливу появу багів при отриманні невірних даних чи при проведенні несподіваних дій.

У сучасному цифровому світі критично важливою є можливість обробляти дані в режимі реального часу, коли миттєва реакція програми на дії користувача призводить до автоматичного оновлення системи. Для створення подібних додатків добре підходить такий фреймворк, як Angular, в основі якого лежить бібліотека RxJS з її можливістю швидко та просто створювати реактивний код.

Unit-тести для будь якого програмного забезпечення потрібні в першу чергу по двом причинам [2, 3, 4]:

1. Знизити вірогідність поломки робочого, вже перевіреного коду;

2. Краще розуміти власний код та відловити помилки заздалегідь.

Термін “юніт” стосовно тестування позначає мінімально можливу частину коду, яку можна протестувати ізольовано. Це може бути функція, метод або навіть окремий рядок коду. Кожен “юніт” має бути протестований окремо від решти коду, щоб переконатися в його коректності. Такий підхід дає змогу виявити помилки та проблеми в коді на ранніх стадіях розробки.

Хороші unit-тести повинні відповідати наступним критеріям [2]:

1. Прості і зрозумілі. Якщо розробник через деякийсь час дивиться на свій тест або тест свого колеги і в нього з’являється бажання видалити або переписати цей тест – то це вже неправильно. Інший випадок, якщо розробник дивиться на код додатку, а потім на тест і розуміє, що тест складніший за код – то це теж неправильно;

2. Тести повинні забезпечувати Quality gata, тобто, якщо код проходить перевірку тестами, то його пропускають далі в розробку, якщо ні – відправляють на доробку;

3. Найменування тестів. Тести повинні мати таке ім’я, щоб у випадку, коли тест не буде пройдено, розробник точно міг визначити, який тест впав;

4. Unit-тести повинні бути однорідними, тобто, щоб не було можливості в команді визначити хто конкретно написав той чи інший тест. Для цього потрібно залучати Code review та статичний аналіз коду. Тести потрібно перевіряти на якість коду так само, як і основний код;

5. Unit-тести повинні бути без side ефектів. Вони повинні бути незалежними і запускатися окремо;

6. Чисті тести. Код тесту повинен читатися легко, навіть якщо тестує складний код;

7. Тести повинні бути швидкими. Вони не повинні конектитись до бази даних, щось отримувати з мережі чи читати з файлової системи. Навіть на дуже великому проекті всі тести повинні проходитись не довше декількох хвилин;

8. Тести повинні перевіряти ЩО робить код, а не те ЯК він це робить. Це робиться для того, щоб була можливість легко змінити основний код або перефакторити.

9. Unit-тести повинні бути іменно Unit-тестами, а не інтеграційними. Наприклад, різні допоміжні утиліти дозволяють писати інтеграційні тести, які можуть бути, в результаті, дуже не стабільними;

10.При написанні тестів треба дотримуватись піраміди тестування. Unit-тестів повинно бути багато, а тестів вищого рівня менше. Коли на проекті багато інтеграційних тестів і мало Unit-тестів – це антипаттерн, тобто перевернута піраміда.

Для вимирювання ефективності Unit-тестів, як правило, використовуються чотири критерії [2]: захист від помилок (protection against bugs); стійкість до рефакторингу (resilience to refactoring); швидкість зворотнього зв'язку (fast feedback); простота підтримки (maintain ability). Порівнюючи різні підходи до тестування можна визначити як той чи інший підхід впливає на конкретний критерій ефективності Unit-тестів, підвищуючі його, або навпаки зменшуючі. Як відомо, на сьогодні не існує автоматичного способу вимірювання ефективності з отриманням значень для кожного критерію і тому кожен тест потрібно оцінювати окремо. Для отримання значення по критерію «захист від помилок» краще за все використовувати команду coverage, для визначення того, яка кількість коду покрита тестами. Чим більший відсоток коду покритий тестами, тим більша вірогідність того, що основний код додатку захищений від помилок. Зазвичай хорошим показником є 80% покриття. Для отримання значення по критерію «швидкість зворотнього зв'язку» будемо використовувати час виконання тестів, в тому числі, і з визначенням відсотку покриття.

Виклад основного матеріалу

Почати писати Unit-тести в Angular додатках можна, використовуючи Jasmine в якості середи тестування та Karma для запуску тестів. Також Angular надає такі утиліти, як TestBed і Async, щоб спростити тестування асинхронного коду, компонентів, директив або сервісів.

Jasmine є середовищем модульного тестування для JavaScript. Він може запускати тести як у Node.js, так і у браузері. Він використовується в середовищі Angular та особливо популярний у проектах на його основі. Це гарний вибір для проектів Vanilla JS, а також проектів, що базуються на інших платформах.

Karma – це інструмент автоматизації тестування, створений командою Angular JS в Google і який дозволяє розробникам тестів налаштовувати те на яких пристроях чи браузерах будуть запускатися тести (за замовчуванням це Chrome) і які тестові фреймворки та плагіни братимуть у цьому участь.

Так як нас цікавить швидкість виконання тестів, то краще для тестування використовувати headless браузер. У headless браузері немає графічного інтерфейсу, і таким чином можна зберігати результати тесту всередині свого терміналу. Він в основному використовується інженерами з тестування програмного забезпечення, оскільки браузери без GUI працюють швидше, так як їм не потрібно малювати візуальний контент. Для використання ChromeHeadless необхідні додаткові налаштування Karma [2, 4].

Так от, якщо мова йде про тестування Angular-додатків, то за замовчуванням мається на увазі використання зв'язки Jasmine + Karma, які встановлюються автоматично при створенні додатку. Jasmine за замовчуванням інтегрована з Karma, а Angular CLI вміє працювати з Karma з коробки. Але писати тести для JavaScript можна також використовуючи тестовий фреймворк Jest. В роботі, як раз, ставиться задача порівняти швидкість тестування як синхронного так і асинхронного коду, використовуючі інструменти для тестування Jest та Karma.

Jest – це фреймворк для тестування, розроблений Facebook, має відкритий вихідний код і заснований на Jasmine. На сьогоднішній день компанія Facebook переробила більшу частину його функціоналу і створила на його основі безліч нових можливостей. Відмінно підходить для тестування проектів, що використовують Node, React, Angular, Vue, Babel, TypeScript і не тільки. Також у Jest є потужні та швидкі вбудовані засоби для аналізу покриття коду тестами.

Створений за допомогою Angular CLI новий Angular-додаток за замовчуванням встановлює всі інструменти, необхідні для unit-тестування [5, 6], і вже містить тест для компонента AppComponent. Файли тестів мають назву у форматі *.spec.ts. При створенні елементів через Angular CLI ці файли тестів створюються за замовчуванням автоматично.

Angular CLI дозволяє згенерувати звіт про покриття додатку тестами в окремій директорії, що має назву coverage. Щоб згенерувати звіт потрібно запустити тести з прапором --code-coverage для Karma, а для Jest – просто --coverage.

Для порівняння швидкості тестування при допомозі зв'язки Jasmine + Karma та Jest було створено Angular- додаток з одним компонентом та одним сервісом. Компонент містить два блоки асинхронного коду та роботу з сервісом. Загалом додаток містить сім тестів (рис. 1.). Тести під номерами 2, 3, 4, 6 містять роботу з асинхронними даними з використанням об'єкту Observable [5, 7, 8], тести з номерами 1, 5, 7 – не містять роботу з асинхронними данимим [7, 9]. Звіт про тестування в терміналі та браузері можно побачити на рис. 1-2.

```
> ng test

⠸ Generating browser application bundles (phase: building)...3
0 11 2024 19:18:00.833:WARN [karma]: No captured browser, open
 http://localhost:9876/
30 11 2024 19:18:01.244:INFO [karma-server]: Karma v6.3.20 ser
ver started at http://localhost:9876/
30 11 2024 19:18:01.245:INFO [launcher]: Launching browsers Ch
rome with concurrency unlimited
⠸ Generating browser application bundles (phase: building)...3
0 11 2024 19:18:01.254:INFO [launcher]: Starting browser Chrom
e
✓ Browser application bundle generation complete.
30 11 2024 19:18:33.722:WARN [karma]: No captured browser, ope
n http://localhost:9876/
30 11 2024 19:18:39.233:INFO [Chrome 131.0.0.0 (Windows 10)]:
Connected on socket nzYZ5T1Fz4ufdybBAAAB with id 99980763
Chrome 131.0.0.0 (Windows 10): Executed 7 of 7 SUCCESS (1.564
secs / 1.277 secs)
TOTAL: 7 SUCCESS
```

Рис. 1. Звіт про тестування при використанні зв'язки Karma+Jasmine

Рис. 2. Тест раннер Karma з успішно виконаними тестами.

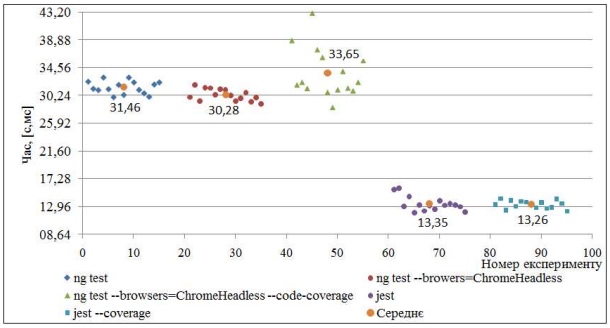

Рис. 3. Тестування Angular-додатку. Тестування синхронного та асинхронного коду разом

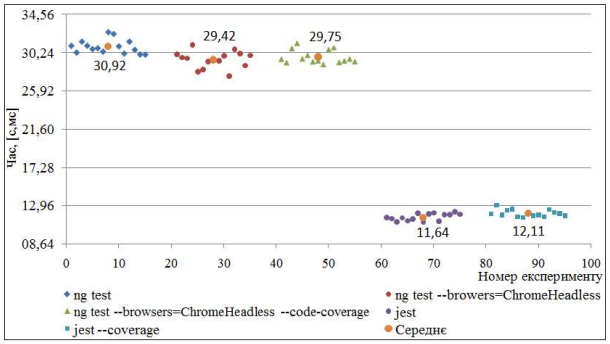

Рис. 4. Тестування Angular-додатку. Тестування асинхронного коду

Для порівняння швидкості тестування виконувалось по 15 експериментів для кожної команди для кожного фреймворка тестування. Умови для тестування в усіх експериментах були однакові (ОС, поточне навантаження на систему та інш.). Результати наведені на рис. 3-4.

Висновки

В результаті порівняння роботи двох фреймворків для тестування, можно зробити висновок, що при використанні Jest тести виконуються в середньому на 60% швидше за рахунок того, що система відстежує зміни файлів та не виконує непотрібних дій, хоча творці Angular все ще радять використовувати саме Jasmine, а не Jest для Unit-тестування. При роботі з Jest результат тестування можна побачити одразу в терміналі, не витрачаючі час на завантаження браузера, на відміну від Jasmine з налаштуваннями за замовчанням. При тестуванні ассинхронного коду така ж тенденція (збільшення швидкості тестування на 60%) зберігається, тому Jest відмінно підходить для виконання тестування Angular-проектів, які містять багато асинхронного коду.

Так як ефективність Unit-тестів залежить від швидкості зворотнього зв'язку (fast feedback), як було сказано вище, тобто чим швидше виконується тест на етапі розробки, тим меньше часу витрачається на усунення помилок, то, тим самим, ми значно підвищуємо ефективність Unit-тестів та всього проекту вцілому.

Література

1. Офіційна документація. https://v17.angular.io/guide/testing
2. Vladimir Khorikov "Unit Testing Principles, Practices, and Patterns" / Manning, 2020, 304 p. ISBN: 978-1617296277.
3. Mauricio Aniche "Effective Software Testing: A developer's guide" / Manning. 2022, 328 p. ISBN: 978-1633439931.
4. Vladimir Khorikov "The Art of Unit Testing, Third Edition: with examples in JavaScript 3rd ed. Edition" / Manning. 2024. 288p. ISBN: 978-1617297489
5. Lamis Chebbi "Reactive Patterns with RxJS for Angular", Published by Packt Publishing Ltd, 2022.
6. Sergi Mansilla "Reactive Programming with RxJS". The Pragmatic Programmers, 2015.
7. Adam Freeman, «Pro Angular. Build Powerful and Dynamic Web Apps». Fifth Edition. Publisher: Apress Berkeley, CA. 2022, 880 pages. DOI: https://doi.org/10.1007/978-1-4842-8176-5.
8. Majid Hajian «Progressive Web Apps with Angular» // Publisher: Apress Berkeley, CA. 2019, 380 pages. DOI: https://doi.org/10.1007/978-1-4842-4448-7.

9. Victor Hugo Garcia «Getting Started with Angular. Create and Deploy Angular Applications» // Publisher: Apress Berkeley, CA. 2023, 373 pages. DOI: https://doi.org/10.1007/978-1-4842-9206-8.

References

1. Oficiyna documentaciya. https://v17.angular.io/guide/testing
2. Vladimir Khorikov “Unit Testing Principles, Practices, and Patterns” / Manning, 2020, 304 p. ISBN: 978-1617296277.
3. Mauricio Aniche ”Effective Software Testing: A developer's guide” / Manning. 2022, 328 p. ISBN: 978-1633439931.
4. Vladimir Khorikov “The Art of Unit Testing, Third Edition: with examples in JavaScript 3rd ed. Edition” / Manning. 2024. 288p. ISBN: 978-1617297489
5. Lamis Chebbi "Reactive Patterns with RxJS for Angular", Published by Packt Publishing Ltd, 2022.
6. Sergi Mansilla "Reactive Programming with RxJS". The Pragmatic Programmers, 2015.
7. Adam Freeman, «Pro Angular. Build Powerful and Dynamic Web Apps». Fifth Edition. Publisher: Apress Berkeley, CA. 2022, 880 pages. DOI: https://doi.org/10.1007/978-1-4842-8176-5.
8. Majid Hajian «Progressive Web Apps with Angular» // Publisher: Apress Berkeley, CA. 2019, 380 pages. DOI: https://doi.org/10.1007/978-1-4842-4448-7.
9. Victor Hugo Garcia «Getting Started with Angular. Create and Deploy Angular Applications» // Publisher: Apress Berkeley, CA. 2023, 373 pages. DOI: https://doi.org/10.1007/978-1-4842-9206-8.

Polupan Yu.V., Rodionov P.Yu., Domin M.K. Comparison of two approaches to Unit testing Angular apps

In the modern world, reactive systems are gaining more and more popularity, and reactivity is based on working with asynchronous data streams. Testing reactive systems is a separate task, in which there are a number of open questions, for example, the extinction of the effectiveness of tests using one or another approach to testing.

For testing Angular applications, two approaches were chosen in the work: 1) using the Jasmine + Karma connection, which is installed by default when creating the application; 2) using Jest.

The article tests an Angular application that has 7 tests, 3 of which are tests of synchronous code blocks and 4 for asynchronous blocks. The tools offered by Angular out of the box and the Jest framework were used during testing.

To extinction the effectiveness of Unit tests, as a rule, four criteria are used: protection against bugs; resilience to refactoring; fast feedback; maintainability. By comparing different approaches to testing, you can determine how a particular approach affects a specific criterion of Unit-test efficiency, increasing it or, conversely, decreasing it. As is known, today there is no automatic method of measuring efficiency with obtaining values for each criterion, and therefore each test must be evaluated separately. To obtain a value for the criterion "feedback speed" we will use the test execution time, including determining the percentage of code coverage by tests using the coverage command. This criterion shows that the faster the test is executed at the development stage, the less time is spent on eliminating the errors found, thereby significantly increasing the efficiency of Unit-tests.

As a result of comparing the work of two testing frameworks, we can conclude that when using Jest, tests are executed on average 60% faster due to the fact that the system tracks file changes and does not perform unnecessary actions. When testing asynchronous code, the same trend (60% increase in testing speed) persists, so Jest is great for testing Angular projects that contain a lot of asynchronous code.

Since the effectiveness of unit tests depends on fast feedback, by using Jest, we significantly increase the effectiveness of unit testing and the entire project as a whole.

Keywords: *Unit testing, Angular application, Jasmine, Karma, Jest, unit tests, --test coverage, Angular CLI, ChromeHeadless, reactivity, synchronous and asynchronous code.*

Полупан Юлія Вікторівна – к.т.н., доц., доц., КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра інформатики та програмної інженерії, juliy_polupan@i.ua

Родіонов Павло Юрійович – к.е.н., доц., КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра інформатики та програмної інженерії.

Дьомін Максим Костянтинович – к. т. н., доц., доц. СНУ ім. В. Даля, факультет інформаційних технологій та електроніки, кафедра інформаційних техноолгій та програмування.

Стаття подана 15.10.2024.