УДК 004.415
DOI: https://doi.org/ 10.17721/3041-2323.2024.89-97

Вікторія ГОЛОТЮК, студ.
ORCID ID: 0009-0008-3947-4613
e-mail: viktoriia_holotiuk@knu.ua
Київський національний університет
імені Тараса Шевченка, Київ, Україна

Олена ВАЩІЛІНА, канд. фіз.-мат. наук, доц.
ORCID ID: 0000-0001-6867-6216
e-mail: olenavashchilina@knu.ua
Київський національний університет
імені Тараса Шевченка, Київ, Україна

ЗАСТОСУВАННЯ МОДУЛЬНОГО ТЕСТУВАННЯ У ПРОЦЕСІ РОЗРОБЛЕННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ

Здійснено аналіз важливості модульного тестування в контексті сучасних мобільних застосунків, висвітлено головні методи та підходи до тестування, а також наведено конкретні приклади реалізації, що демонструють підвищення якості та надійності продукту. Особливу увагу приділено огляду популярних інструментів і платформ для модульного тестування мобільних застосунків, таких як Mockito, XCTest, JUnit та Espresso, з детальним описом їхньої функціональності та переваг.

Ключові слова: ***модульне тестування, мобільні застосунки, якість, надійність, розроблення.***

Вступ

У сучасному світі мобільні застосунки стали невід'ємною частиною життя мільйонів користувачів, а мобільні технології розвиваються надзвичайно швидкими темпами. З огляду на це потреба в забезпеченні надійності, стабільності та високої якості таких застосунків зростає. Модульне тестування є одним із найефективніших інструментів для досягнення цих цілей. Воно дозволяє виявляти та виправляти помилки на ранніх етапах розроблення, що допомагає уникнути проблем на рівні інтеграції та покращує загальну якість програмного забезпечення (GeeksForGeeks, n. d., QALight, n. d.).

Результати

Модульне тестування – це процес перевірки найменших одиниць коду, таких як функції, методи або класи, ізольовано від решти системи. Його основна мета – переконатися, що кожен окремий компонент працює відповідно до визначених вимог і не містить помилок.

До основних принципів модульного тестування належать (Khorikov, 2020):

• ізоляція (кожен тест перевіряє окремий модуль або компонент, без впливу інших частин системи);

• незалежність (кожен тест виконується окремо, і його результат не залежить від того, чи працюють інші тести);

• повторюваність (тести повинні давати однакові результати у кожному запуску);

• точність (тест має точно виявляти причину помилки, прив'язуючи її до конкретного модуля або функції);

• чітка структура (кожен тест має бути простим і зрозумілим, із чіткими вхідними даними, очікуваним результатом і перевіркою).

Використання модульного тестування має численні переваги. Однією з головних переваг є раннє виявлення помилок, що дає змогу швидко їх виправляти. Це також забезпечує стабільність і надійність застосунку у разі внесення нових змін, дозволяючи уникати помилок під час виконання вже працюючих функцій. Швидкий зворотний зв'язок для розробників дає можливість швидко тестувати нові зміни, що прискорює процес розроблення та мінімізує ризики появи помилок у кінцевій версії продукту. Також тести допомагають у масштабуванні, забезпечуючи впевненість у тому, що внесені зміни не зашкодять існуючому функціоналу. Крім того, тести можуть слугувати як документація, адже, описуючи поведінку окремих модулів, вони допомагають новим розробникам швидше розібратися у структурі програми, полегшуючи її підтримку та розвиток.

Проте модульне тестування має свої недоліки. Написання та підтримка тестів може бути трудомістким процесом, що збільшує витрати часу. Ще одним недоліком є те, що тести перевіряють лише окремі компоненти, не враховуючи їхню взаємодію з іншими частинами системи. Розв'язком цієї проблеми є поєднання модульних тестів з інтеграційним і кінцевим тестуванням. Також

може виникати складність тестування специфічних функцій мобільних пристроїв, таких як камера або GPS, проте для цього можна використовувати mock-об'єкти або емулятори, які дозволяють імітувати їхню поведінку без необхідності взаємодії з реальними пристроями. Отже, незважаючи на певні виклики, модульне тестування є важливим елементом забезпечення якості мобільних застосунків, а правильна реалізація цього підходу в процес розроблення дозволяє максимізувати його переваги (ZapTest, n. d.).

Успішне модульне тестування мобільних застосунків потребує використання спеціалізованих платформ та інструментів, які дозволяють автоматизувати процеси перевірки та підвищити ефективність тестування. Розглянемо найпопулярніші з них: Mockito, XCTest, JUnit, Espresso (Medium, n. d.).

Mockito є популярною бібліотекою для Java, яка дозволяє створювати mock-об'єкти для імітування поведінки реальних залежностей під час тестування. Вона надає можливість ізолювати модулі, що тестуються, від зовнішніх компонентів, таких як бази даних або API. Завдяки цьому розробники можуть зосередитися на перевірці конкретної бізнес-логіки, не відволікаючись на деталі реалізації зовнішніх залежностей. Це робить тести більш швидкими, надійними і легшими для розуміння, що значно підвищує ефективність тестування.

XCTest − це фреймворк для тестування, який входить до складу Xcode й активно використовується для розроблення застосунків на iOS. Цей інструмент підтримує як модульне, так і інтеграційне тестування та дозволяє розробникам перевіряти не тільки окремі модулі, а і їхню взаємодію між собою. XCTest пропонує різноманітні функції, що включає вимірювання продуктивності, а також можливість створення UI-тестів. Саме це робить його універсальним інструментом, здатним забезпечити високий рівень якості застосунків на платформах Apple.

JUnit є стандартним фреймворком для модульного тестування Java і широко використовується в Android-розробці. Він забезпечує простий і зрозумілий синтаксис для створення та запуску тестів, наприклад, анотації, що полегшують організацію тестів. JUnit дозволяє перевіряти бізнес-логіку та функціональність модулів, що робить його невід'ємною частиною процесу розроблення

Android-застосунків. Використання JUnit допомагає підтримувати високий рівень організації коду і забезпечує надійність кінцевого продукту.

Espresso також є потужним інструментом для автоматизованого тестування інтерфейсу користувача в Android-застосунках. Фреймворк дозволяє розробникам писати тести для перевірки взаємодії з UI-компонентами, забезпечуючи простий та інтуїтивно зрозумілий спосіб перевірки функціональності застосунків. Це дозволяє інфраструктурі виконувати дії, подібні до дій користувача, наприклад натискання кнопок, введення тексту в текстові поля та вибір параметрів у меню. Завдяки цьому Espresso забезпечує виявлення проблем інтерфейсу застосунку на ранніх етапах розроблення та підвищує якість користувацького досвіду, що робить його важливим елементом комплексного процесу тестування. Порівняльну характеристику інструментів модульного тестування Mockito, XCTest, JUnit, Espresso наведено у табл. 1, що допоможе наочно продемонструвати, який інструмент краще підходить для різних типів тестування та мов програмування.

Вибір правильних платформ та інструментів для модульного тестування є критично важливим для забезпечення якості мобільних застосунків, а впровадження модульного тестування охоплює кілька ключових етапів, починаючи з планування та розроблення, і закінчуючи тестуванням, інтеграцією та релізом. Розглянемо ключові аспекти кожного із цих етапів (GeeksForGeeks, n. d.**;** ZapTest, n. d.**)**.

Першим етапом є планування. Тут розробники визначають вимоги до застосунку, а також планують, які модулі потрібно протестувати. Важливо вже на цьому етапі розглянути, як модульне тестування вплине на архітектуру застосунку, щоб забезпечити легкість у написанні тестів.

Наступний етап – це розроблення тесту. Під час написання коду розробники створюють модульні тести паралельно з реалізацією нових функцій. Цей підхід, відомий як розроблення через тестування (TDD), сприяє створенню тестів перед написанням коду, що дозволяє зменшити кількість помилок і покращити структуру коду.

Таблиця 1

Порівняльна характеристика інструментів модульного тестування

Параметр	Mockito	XCTest	JUnit	Espresso
Платформа	Java/ Android	iOS/macOS	Java/ Android	Android
Тип тестування	Модульне (мокування)	Модульне, інтеграційне UI	Модульне	UI/ інтеграційне
Основне призначення	Створення mock-об'єктів	Комплексне тестування iOS- застосунків	Модульне тестування Java коду	Тестування користувацького інтерфейсу
Автоматизація	Вручну через створення mock-об'єктів	Підтримка автоматизованих тестів	Автоматизація через JUnit правила	Підтримка автоматизованих сценаріїв UI
Інтеграція з IDE	Android Studio	Xcode	Eclipse, IntelliJ IDEA	Android Studio
Складність використання	Середня	Низька	Низька	Висока
Підтримка асинхронного тестування	Обмежена	Наявна	Через додаткові бібліотеки	Наявна
Можливості звітування	Обмежені	Розширені	Розширені	Базові

Коли код написано, розпочинають етап тестування. Усі модульні тести запускають, щоб виявити будь-які недоліки в логіці та функціональності окремих модулів. Автоматизація тестування забезпечує швидкий зворотний зв'язок, що дозволяє розробникам швидко усувати помилки.

Після виявлення та виправлення помилок, наступає етап інтеграції. Тут модульні тести допомагають перевірити, чи коректно взаємодіють нові модулі з уже існуючими компонентами системи.

На етапі впровадження та підтримки системи модульні тести також відіграють важливу роль. Перед випуском нової версії застосунку всі тести повинні бути успішно пройдені, що гарантує відсутність критичних помилок у кінцевій версії. Модульні тести також мають оновлюватися разом із внесенням нових функцій або

виправленням помилок. Регулярне проведення тестів під час оновлень забезпечує стабільність і надійність застосунку протягом його життєвого циклу.

Для прикладу розглянемо тестування мобільного застосунку, який допомагає користувачам організовувати свій час. Він надає можливість створювати, редагувати та видаляти завдання, а також сортувати їх за категоріями. Для забезпечення стабільності та надійності функцій впроваджено модульне тестування, що дозволило створити якісний кінцевий продукт.

Перед початком роботи з мобільним застосунком критично важливо забезпечити коректність і надійність бази даних, яка є основою для зберігання всієї інформації, включаючи завдання та категорії. Тестування ініціалізації бази даних є першим кроком у верифікації її функціональності. Тест, зображений на рис. 1, перевіряє, чи правильно створюється база даних та чи існують усі необхідні таблиці. Цей процес забезпечує правильність подальших операцій із базою даних, зокрема і вставлення, оновлення та видалення даних.

```
test('Database is initialized correctly', () async {
  final db = await databaseHelper.database;
  expect(db.isOpen, true);
});

test('Tables are created correctly', () async {
  final db = await databaseHelper.database;
  // Перевірка існування таблиці tasks
  final taskTable = await db.rawQuery("SELECT name FROM sqlite_master WHERE type='table' AND name='${DatabaseHelper.taskTable}'");
  expect(taskTable.isNotEmpty, true);
  // Перевірка існування таблиці categories
  final categoryTable = await db.rawQuery("SELECT name FROM sqlite_master WHERE type='table' AND name='${DatabaseHelper.categoryTable}'");
  expect(categoryTable.isNotEmpty, true);
  // Перевірка існування таблиці reminders
  final reminderTable = await db.rawQuery("SELECT name FROM sqlite_master WHERE type='table' AND name='${DatabaseHelper.reminderTable}'");
  expect(reminderTable.isNotEmpty, true);
});
```

Рис. 1. *Модульне тестування ініціалізації бази даних*

Важливим аспектом була також перевірка функціональності додавання нових завдань. Тест, зображений на рис. 2, перевіряє, чи система правильно реагує на спробу додати завдання. Він перевіряє, чи при успішному додаванні повертається коректний ідентифікатор нового завдання. Процес починається з того, що створюється об'єкт Task, який містить усі необхідні дані: заголовок,

опис, початковий і кінцевий терміни виконання, ідентифікатор категорії та статус виконання.

```
test('Insert task: should insert a task into the database', () async {
  final task = Task(
    title: 'Test Task',
    description: 'Test Description',
    deadlineStart: DateTime.now(),
    deadlineEnd: DateTime.now().add(Duration(hours: 1)),
    categoryId: 1,
    isCompleted: false,
  );
  final id = await taskService.insertTask(task);
  expect(id, greaterThanOrEqualTo(0));
  await taskService.deleteTask(id);

});
```

Рис. 2. *Модульне тестування додавання категорії*

Ще одним важливим моментом була перевірка коректності роботи функції отримання категорій. Тест, що зображений на рис. 3, перевіряє, чи система може правильно знайти категорію за її унікальним ідентифікатором. Спочатку категорію додають до бази даних, а потім за її ID здійснюється запит.

```
test('Get category by ID: should retrieve a category from the database by its ID', () async {
  final category = Category(id: 0, name: 'Test Category');
  final id = await categoryService.insertCategory(category);
  final fetchedCategory = await categoryService.getCategoryById(id);
  expect(fetchedCategory, isNotNull);
  expect(fetchedCategory!.id, id);
  expect(fetchedCategory.name, 'Test Category');
  await categoryService.deleteCategory(id);
});
```

Рис. 3. *Модульне тестування отримання категорії*

На прикладі цієї програмної системи можна переконатися у важливості модульного тестування для мобільних застосунків. Завдяки вказаному підходу до тестування досягнуто впевненості в коректності роботи бази даних, а також у правильності оброблення дій користувачів. Це сприяло створенню надійного застосунку, що відповідає вимогам користувачів.

Дискусія і висновки

У підсумку можна стверджувати, що модульне тестування є потужним інструментом, який дозволяє розробникам створювати більш надійні, стабільні та якісні мобільні застосунки. Цей засіб не лише допомагає виявляти й усувати помилки, але і сприяє покращенню архітектури програми, полегшує процес рефакторингу та забезпечує впевненість у стабільності продукту протягом усього життєвого циклу розроблення. У контексті стрімкого розвитку мобільних технологій та зростаючих вимог користувачів до якості програмного забезпечення, модульне тестування стає не просто корисним доповненням, а необхідним елементом процесу розроблення програм, що забезпечує конкурентоспроможність та успіх мобільних застосунків на ринку.

Список використаних джерел

GeeksForGeeks. (n. d.). *Unit Testing – Software Testing.* https://www.geeksforgeeks.org/unit-testing-software-testing/ (Дата звернення: 20.09.2024).

Khorikov, V. (2020). *Unit Testing. Principles, Practices, and Patterns.* New York: Manning Publications.

Medium. (n. d.). *Unit Testing in Android: A Guide of Using JUnit, Mockito, MockK, and Espresso.* https://dogukanincee.medium.com/unit-testing-in-android-a-guide-of-using-junit-mockito-mockk-and-espresso-d7a47819ada5 (Дата звернення: 21.09.2024).

QALight. (б. д.). *Модульне тестування.* https://qalight.ua/baza-znaniy/modulne-testuvannya/ (Дата звернення: 21.09.2024).

ZapTest. (б. д.). *Що таке модульне тестування?* https://rb.gy/rgg4b2 (Дата звернення: 21.09.2024).

References

GeeksForGeeks. (n. d.). *Unit Testing – Software Testing.* https://www.geeksforgeeks.org/unit-testing-software-testing/ (Accessed: 20.09.2024).

Khorikov, V. (2020). *Unit Testing. Principles, Practices, and Patterns.* New York: Manning Publications.

Medium. (n. d.). *Unit Testing in Android: A Guide of Using JUnit, Mockito, MockK, and Espresso.* https://dogukanincee.medium.com/unit-testing-in-android-a-guide-of-using-junit-mockito-mockk-and-espresso-d7a47819ada5 (Accessed:21.09.2024).

QALight. (n. d.). *Modular Testing.* https://qalight.ua/baza-znaniy/modulne-testuvannya/ (Accessed: 21.09.2024).

ZapTest. (n. d.). *What is unit testing?* https://rb.gy/rgg4b2 (Accessed: 21.09.2024).

Отримано редакцією журналу / Received: 25.09.24
Прорецензовано / Revised: 27.09.24
Схвалено до друку / Accepted: 01.10.24

Viktoriia HOLOTIUK, Student
ORCID ID: 0009-0008-3947-4613
e-mail: viktoriia_holotiuk@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

Olena Vashchilina, PhD (Phys. & Math.), Assoc. Prof.
ORCID ID: 0000-0001-6867-6216
e-mail: olenavashchilina@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

APPLICATION OF UNIT TESTING IN THE MOBILE APPLICATION DEVELOPMENT PROCESS

The report analyzes the significance of modular testing in the context of modern mobile applications, highlighting key methods and approaches to testing, as well as providing specific examples of implementations that demonstrate improvements in product quality and reliability. Particular attention is given to the review of popular tools and platforms for unit testing mobile applications, such as Mockito, XCTest, JUnit, and Espresso, with a detailed description of their functionality and advantages.

Keywords: ***unit testing, mobile applications, quality, reliability, development.***

Автори заявляють про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.

The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.