Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Кафедра ІПЗ

Лабораторна робота №7

Тема: «Побудова базового автоматизаційного рішення»

Виконала

ст. гр. ІП-21-3

Лаба Ю.С.

Перевірив

Храбатин Р.І.

м. Івано-Франківськ

2025

**Мета роботи:** навчитися створювати тести для перевірки користувацьких сценаріїв, розібратися в існуючих інструментах для створення тестів для перевірки користувацьких сценаріїв та навчитися їх обирати.

**Короткі теоретичні відомості**

Автоматизація тестування є важливою частиною забезпечення якості ПЗ, яка дозволяє підвищити ефективність і надійність перевірки функціональності системи. Проте успішність автоматизації значною мірою залежить не лише від кількості тестів, а й від якісної архітектури рішення.

Ключовими атрибутами якісної автоматизації є:

* Підтримуваність — простота внесення змін у тести без значних витрат.
* Продуктивність — швидке та ефективне виконання тестів.
* Легкість навчання — зрозуміла структура та документація.
* Модульність — незалежність та багаторазове використання компонентів.
* Перевикористання — уніфікація сценаріїв, змінних та компонентів.
* Надійність — стабільне проходження тестів без випадкових збоїв.
* Масштабованість — можливість розширення та адаптації архітектури.
* Інтеграція — здатність взаємодіяти з іншими системами та сервісами.

При побудові автоматизованого рішення варто дотримуватись принципів:

* SOLID — для побудови розширюваних і гнучких компонентів.
* DRY (Don’t Repeat Yourself) — уникнення дублювання коду.
* KISS (Keep It Simple, Stupid) — простота реалізації.
* POM (Page Object Model) — ізоляція логіки роботи зі сторінками.

Архітектура рішення може мати багаторівневу структуру:

* Ядро: скрипти, логери, конфігурації.
* Бізнес-рівень: логіка, компоненти, PageObjects, API контракти.
* Тестовий рівень: тести, мок-дані.
* Сервісний рівень: залежності, налаштування CI/CD, лінтери, README тощо.

**Хід роботи**

1. **API – tests**
2. **Get all books** — перевіряє, що API повертає список книг зі статусом 200 і що поле books є масивом.

test("Get all books", **async** () **=>** {

**const** response = await axios.get(`${BASE\_URL}/Books`);

    expect(response.status).toBe(200);

    expect(Array.isArray(response.data.books)).toBe(true);

  });

1. **Get all books and check if a specific ISBN exists** — додатково перевіряє, що в списку книг є книга з конкретним ISBN.

test("Get all books and check if a specific ISBN exists", **async** () **=>** {

**const** expectedIsbn = "9781449331818";

**const** response = await axios.get(`${BASE\_URL}/Books`);

    expect(response.status).toBe(200);

    expect(Array.isArray(response.data.books)).toBe(true);

    expect(response.data.books.length).toBeGreaterThan(0);

**const** bookExists = response.data.books.some(

      (book) **=>** book.isbn === expectedIsbn

    );

    expect(bookExists).toBe(true);

  });

1. **User login and get token** — тестує успішний вхід користувача і отримання токена.

test("User login and get token", **async** () **=>** {

**const** credentials = {

      userName: "testuser123",

      password: "Test@1234",

    };

**const** response = await axios.post(

      "https://demoqa.com/Account/v1/Login",

      credentials

    );

    expect(response.status).toBe(200);

    expect(response.data.token).toBeDefined();

  });

1. **Get user info without token returns 401** — перевіряє, що запит інформації про користувача без токена повертає помилку 401 (неавторизований).

test("Get user info without token returns 401", **async** () **=>** {

**const** userName = "testuser123";

    try {

      await axios.get(`https://demoqa.com/Account/v1/User/${userName}`);

    } catch (error) {

      expect(error.response.status).toBe(401);

    }

  });

1. **User login and get token (GenerateToken endpoint)** — перевіряє успішну генерацію токена з додатковою перевіркою поля expires.

test("User login and get token", **async** () **=>** {

**const** loginData = {

      userName: "testuser123",

      password: "Test@1234",

    };

**const** response = await axios.post(

      "https://demoqa.com/Account/v1/GenerateToken",

      loginData

    );

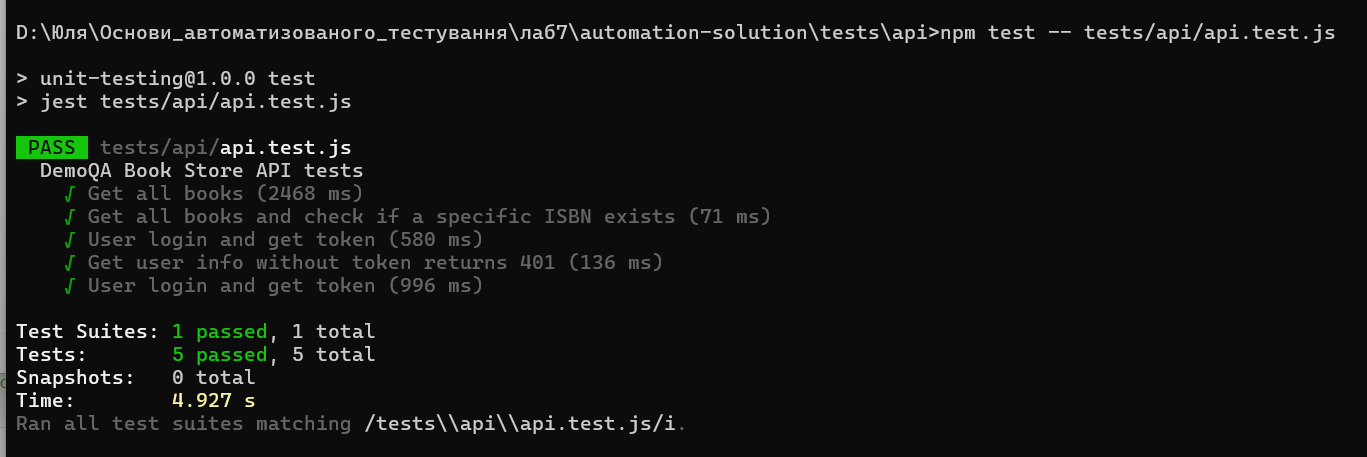
    expect(response.status).toBe(200);

    expect(response.data.token).toBeDefined();

    expect(response.data.expires).toBeDefined();

  });

Результат:

****

1. **Data-tests**
2. **Get all books**  
   Перевіряє, що API повертає список усіх книг зі статусом 200, і що поле books є масивом.

  test("Get all books", **async** () **=>** {

**const** response = await axios.get(`${BASE\_URL}/Books`);

    expect(response.status).toBe(200);

    expect(Array.isArray(response.data.books)).toBe(true);

    expect(response.data.books.length).toBeGreaterThan(0);

  });

1. **Get all books and check if a specific ISBN exists**  
   Перевіряє, що в отриманому списку книг присутня книга з конкретним ISBN.

  test("Check if book with specific ISBN exists", **async** () **=>** {

**const** isbn = "9781449331818";

**const** response = await axios.get(`${BASE\_URL}/Books`);

**const** bookExists = response.data.books.some((book) **=>** book.isbn === isbn);

    expect(bookExists).toBe(true);

  });

1. **Get user info without token returns 401**  
   Перевіряє, що спроба отримати інформацію про користувача без авторизаційного токена повертає помилку 401 (Unauthorized).

  test("Create a new user successfully", **async** () **=>** {

**const** uniqueUser = `testuser${Date.now()}`;

**const** userData = { userName: uniqueUser, password: "Test@1234" };

**const** response = await axios.post(

      "https://demoqa.com/Account/v1/User",

      userData

    );

    expect(response.status).toBe(201);

    expect(response.data.username).toBe(uniqueUser);

    expect(response.data.userID).toBeDefined();

  });

1. **Generate authentication token for user**  
   Перевіряє, що можна отримати валідний токен автентифікації для користувача через API GenerateToken.

  test("Fail to create user with existing username", **async** () **=>** {

**const** userData = { userName: "testuser123", password: "Test@1234" };

    try {

      await axios.post("https://demoqa.com/Account/v1/User", userData);

    } catch (error) {

      expect([400, 406]).toContain(error.response.status);

    }

  });

1. **User login and get token (POST /Account/v1/Login)**  
   Перевіряє, що при логіні користувача через API Login повертається статус 200 та валідний токен.

  test("Generate authentication token for user", **async** () **=>** {

**const** loginData = {

      userName: "testuser123",

      password: "Test@1234",

    };

**const** response = await axios.post(

      "https://demoqa.com/Account/v1/GenerateToken",

      loginData

    );

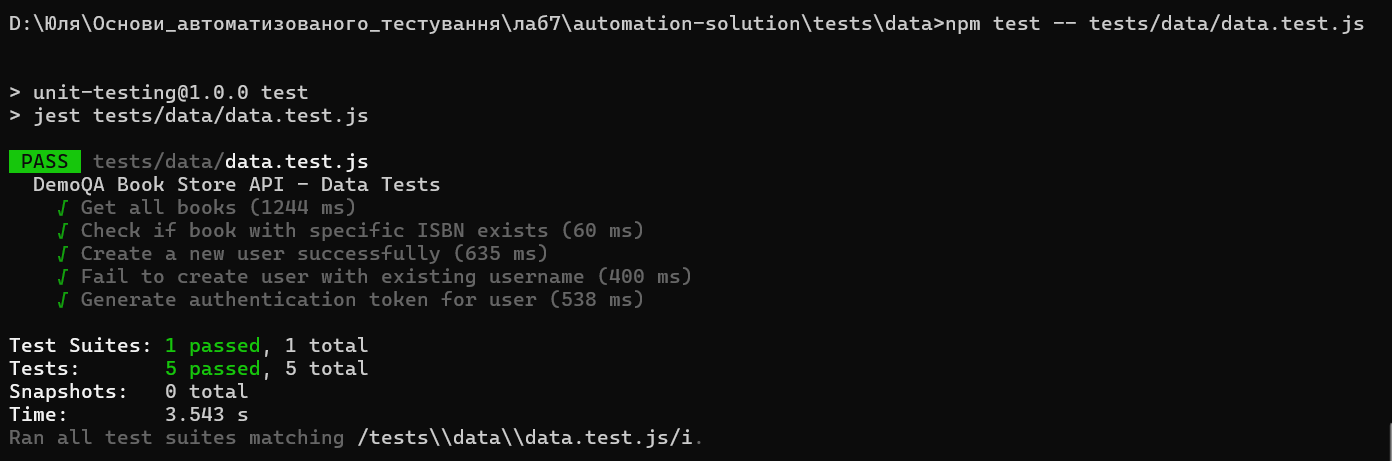
    expect(response.status).toBe(200);

    expect(response.data.token).toBeDefined();

    expect(typeof response.data.token).toBe("string");

    expect(response.data.expires).toBeDefined();

  });

****

1. **E2E-tests**
2. Radio Buttons

* Відкриваємо сторінку з радіокнопками: <https://demoqa.com/radio-button>.
* Очікуємо на появу елемента "Yes" та клікаємо на нього.
* Чекаємо, поки з’явиться елемент із класом .text-success.
* Перевіряємо, що його текст дорівнює "Yes" — радіокнопка була обрана успішно.

test('Radio Buttons - should select "Yes" and check response', **async** () **=>** {

    await page.goto('https://demoqa.com/radio-button', { waitUntil: 'networkidle2' });

    await page.waitForSelector('label[for="yesRadio"]', { visible: true });

    await page.click('label[for="yesRadio"]');

    await page.waitForSelector('.text-success', { visible: true });

**const** result = await page.$eval('.text-success', el **=>** el.textContent.trim());

    expect(result).toBe('Yes');

  });

1. Web Tables

* Переходимо на сторінку з таблицями: <https://demoqa.com/webtables>.
* Чекаємо на кнопку Add, натискаємо її.
* Очікуємо на появу модальної форми (поле #firstName).
* Перевіряємо, що модальне вікно дійсно видиме — форма успішно з’явилась.

test('Web Tables - should click Add button and see modal form', **async** () **=>** {

    await page.goto('https://demoqa.com/webtables', { waitUntil: 'networkidle2' });

    await page.waitForSelector('#addNewRecordButton', { visible: true });

    await page.click('#addNewRecordButton');

    await page.waitForSelector('#firstName', { visible: true });

**const** formVisible = await page.$eval('#registration-form-modal', el **=>** getComputedStyle(el).display !== 'none');

    expect(formVisible).toBe(true);

  });

1. Drag and Drop

* Відкриваємо сторінку з перетягуванням: <https://demoqa.com/droppable>.
* Знаходимо обидва елементи: той, який треба перетягнути (#draggable) і куди перетягнути (#droppable).
* Емулюємо мишку: натискаємо, перетягуємо в ціль і відпускаємо.
* Очікуємо на зміну тексту у приймаючому блоці на "Dropped!" — перетягування спрацювало.

test('Drag and Drop - should move element from left to right',

**async** () **=>** {

      await page.goto('https://demoqa.com/droppable', { waitUntil: 'networkidle2' });

**const** draggable = await page.$('#draggable');

**const** droppable = await page.$('#droppable');

**const** draggableBox = await draggable.boundingBox();

**const** droppableBox = await droppable.boundingBox();

      await page.mouse.move(

        draggableBox.x + draggableBox.width / 2,

        draggableBox.y + draggableBox.height / 2

      );

      await page.mouse.down();

      await page.mouse.move(

        droppableBox.x + droppableBox.width / 2,

        droppableBox.y + droppableBox.height / 2,

        { steps: 20 }

      );

      await page.mouse.up();

      await page.waitForFunction(

        () **=>** document.querySelector('#droppable p').textContent.includes('Dropped!'),

        { timeout: 5000 }

      );

    },

    15000

  );

1. Dynamic Properties

* Переходимо на сторінку з динамічними властивостями: <https://demoqa.com/dynamic-properties>.
* Очікуємо, поки кнопка #enableAfter стане доступною для кліку (enabled).
* Перевіряємо, що вона справді активна — динамічна активація працює.

test('Dynamic Properties - should wait for enabled button',

**async** () **=>** {

      await page.goto('https://demoqa.com/dynamic-properties', { waitUntil: 'domcontentloaded' });

      await page.waitForSelector('#enableAfter:enabled', { timeout: 10000 });

**const** isEnabled = await page.$eval('#enableAfter', el **=>** !el.disabled);

      expect(isEnabled).toBe(true);

    },

    15000

  );

1. Modal Dialog

* Відкриваємо сторінку з модальними вікнами: <https://demoqa.com/modal-dialogs>.
* Натискаємо кнопку Show Small Modal.
* Очікуємо появу заголовку Small Modal.
* Перевіряємо, що вікно відкрилось правильно.
* Натискаємо кнопку Close, щоб закрити його.
* Перевіряємо, що вікно успішно зникло.

test('Modal Dialog - should open and close small modal',

**async** () **=>** {

      await page.goto('https://demoqa.com/modal-dialogs', { waitUntil: 'domcontentloaded' });

      await page.click('#showSmallModal');

      await page.waitForSelector('#example-modal-sizes-title-sm', { visible: true });

**const** modalTitle = await page.$eval('#example-modal-sizes-title-sm', el **=>** el.textContent);

      expect(modalTitle).toBe('Small Modal');

      await page.click('#closeSmallModal');

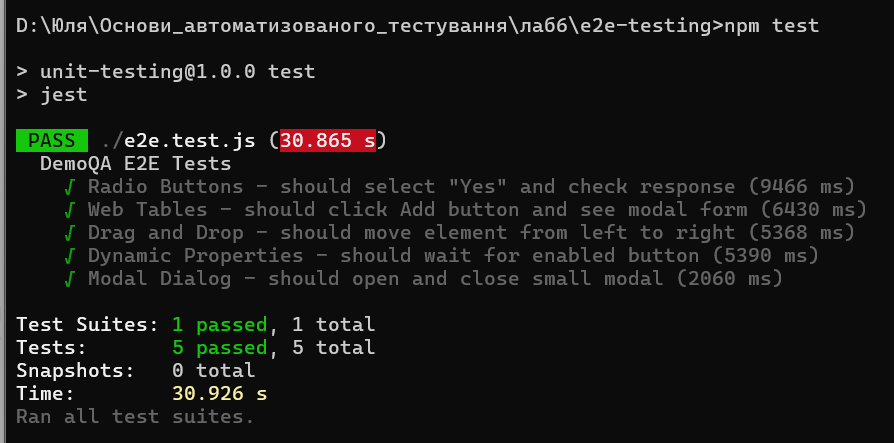
      await page.waitForSelector('#example-modal-sizes-title-sm', { hidden: true });

    },

    15000

  );

Результат:

****

1. **UI-tests**
2. Display search box on Book Store page  
   Перевіряє, що на сторінці Book Store присутнє поле пошуку з відповідним селектором #searchBox.

  test("Should display search box", **async** () **=>** {

    await page.goto("https://demoqa.com/books", { waitUntil: "networkidle2" });

**const** searchBox = await page.$("#searchBox");

    expect(searchBox).toBeTruthy();

  }, 10000);

1. Display books table  
   Перевіряє, що на сторінці завантажується таблиця з книгами, представлена елементом із класом .rt-table.

  test("Should find book by search", **async** () **=>** {

    await page.goto("https://demoqa.com/books");

    await page.type("#searchBox", "Git");

    await page.waitForSelector(".rt-tbody .rt-tr-group");

**const** bookText = await page.$eval(

      ".rt-tbody .rt-tr-group",

      (el) **=>** el.textContent

    );

    expect(bookText).toMatch(/Git/i);

  });

1. Find book by title using search  
   Перевіряє, що функціонал пошуку працює коректно — при введенні ключового слова відображаються відповідні книги (наприклад, "Git").

  test("Should display books table", **async** () **=>** {

    await page.goto("https://demoqa.com/books", { waitUntil: "networkidle2" });

**const** table = await page.$(".rt-table");

    expect(table).toBeTruthy();

  }, 10000);

1. Check correct table column headers  
   Перевіряє, що таблиця має правильні заголовки стовпців — “Title”, “Author” та “Publisher”.

  test("Should have correct table column headers", **async** () **=>** {

    await page.goto("https://demoqa.com/books");

**const** headers = await page.$$eval(".rt-thead.-header .rt-th", (ths) **=>**

      ths.map((th) **=>** th.textContent.trim())

    );

    expect(headers).toEqual(

      expect.arrayContaining(["Image", "Title", "Author", "Publisher"])

    );

  });

1. Scroll to the bottom of the Book Store page  
   Перевіряє, що сторінка підтримує прокручування донизу, що може бути необхідним для завантаження всіх книг або елементів.

  test("Should scroll to the bottom of the book list page", **async** () **=>** {

    await page.goto("https://demoqa.com/books");

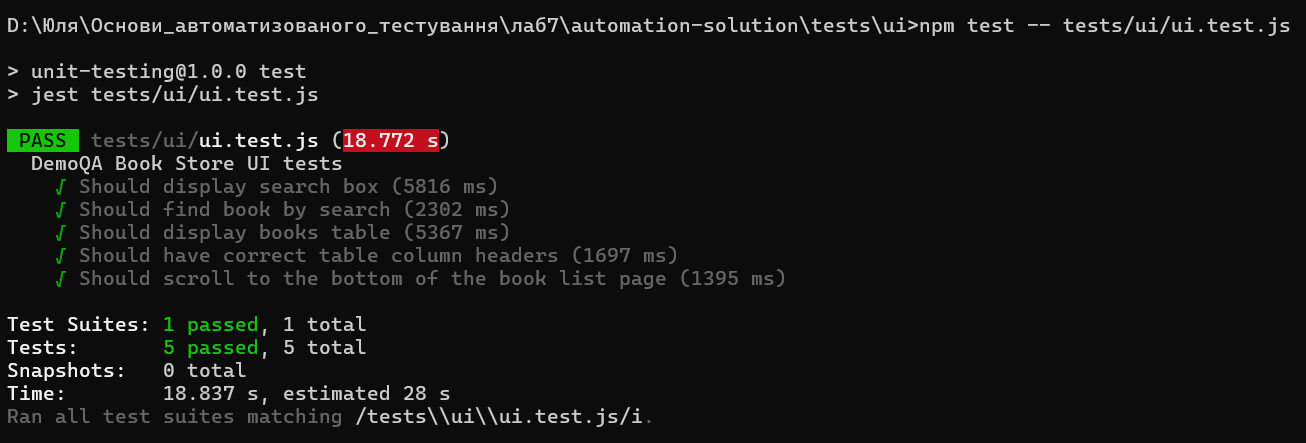
    await page.evaluate(() **=>** window.scrollTo(0, document.body.scrollHeight));

**const** scrollPosition = await page.evaluate(() **=>** window.scrollY);

    expect(scrollPosition).toBeGreaterThan(0);

  });

Результат:

****

1. **Unit-tests**

1) isValidISBN  
Перевіряє, чи є переданий рядок валідним ISBN-13 — тобто містить рівно 13 цифр.

export **const** isValidISBN = (isbn) **=>** {

  return typeof isbn === 'string' && /^\d{13}$/.test(isbn);

};

Тест: Validate correct ISBN format  
Перевіряє, що функція коректно ідентифікує правильний ISBN (13 цифр) та відкидає неправильні значення.

describe('isValidISBN', () **=>** {

  test('validates correct ISBN format', () **=>** {

    expect(isValidISBN('9781234567890')).toBe(true);

    expect(isValidISBN('1234567890')).toBe(false);

    expect(isValidISBN(9781234567890)).toBe(false);

  });

});

2) filterBooksByAuthor  
Фільтрує список книг за заданим ім’ям автора.

export **const** filterBooksByAuthor = (books, author) **=>** {

  return books.filter(book **=>** book.author === author);

};

Тест: Filter books by author  
Перевіряє, що функція повертає тільки книги, які належать певному автору.

describe('filterBooksByAuthor', () **=>** {

  test('filters books by author correctly', () **=>** {

**const** result = filterBooksByAuthor(mockBooks, 'Author 1');

    expect(result.length).toBe(2);

    expect(result.every(book **=>** book.author === 'Author 1')).toBe(true);

  });

});

3) hasBookByISBN  
Перевіряє, чи існує в списку книга з певним ISBN.

export **const** hasBookByISBN = (books, isbn) **=>** {

  return books.some(book **=>** book.isbn === isbn);

};

Тест: Check book presence by ISBN  
Перевіряє, що функція правильно визначає наявність або відсутність книги за ISBN.

describe('hasBookByISBN', () **=>** {

  test('checks presence of book by ISBN', () **=>** {

    expect(hasBookByISBN(mockBooks, '9780987654321')).toBe(true);

    expect(hasBookByISBN(mockBooks, '0000000000000')).toBe(false);

  });

});

4) sortBooksByTitle  
Сортує список книг за назвою в алфавітному порядку.

export **const** sortBooksByTitle = (books) **=>** {

  return [...books].sort((a, b) **=>** a.title.localeCompare(b.title));

};

Тест: Sort books by title alphabetically  
Перевіряє, що функція сортує книги за назвою від А до Я.

describe('sortBooksByTitle', () **=>** {

  test('sorts books by title alphabetically', () **=>** {

**const** sorted = sortBooksByTitle(mockBooks);

    expect(sorted[0].title).toBe('Book A');

    expect(sorted[1].title).toBe('Book B');

    expect(sorted[2].title).toBe('Book C');

  });

});

5) getUniquePublishers  
Повертає список унікальних видавництв зі списку книг.

export **const** getUniquePublishers = (books) **=>** {

**const** publishers = books.map(book **=>** book.publisher);

  return [...new Set(publishers)];

};

Тест: Get unique publishers from book list  
Перевіряє, що функція повертає всі унікальні видавництва без дублікатів.

describe('getUniquePublishers', () **=>** {

  test('returns unique publishers from book list', () **=>** {

**const** publishers = getUniquePublishers(mockBooks);

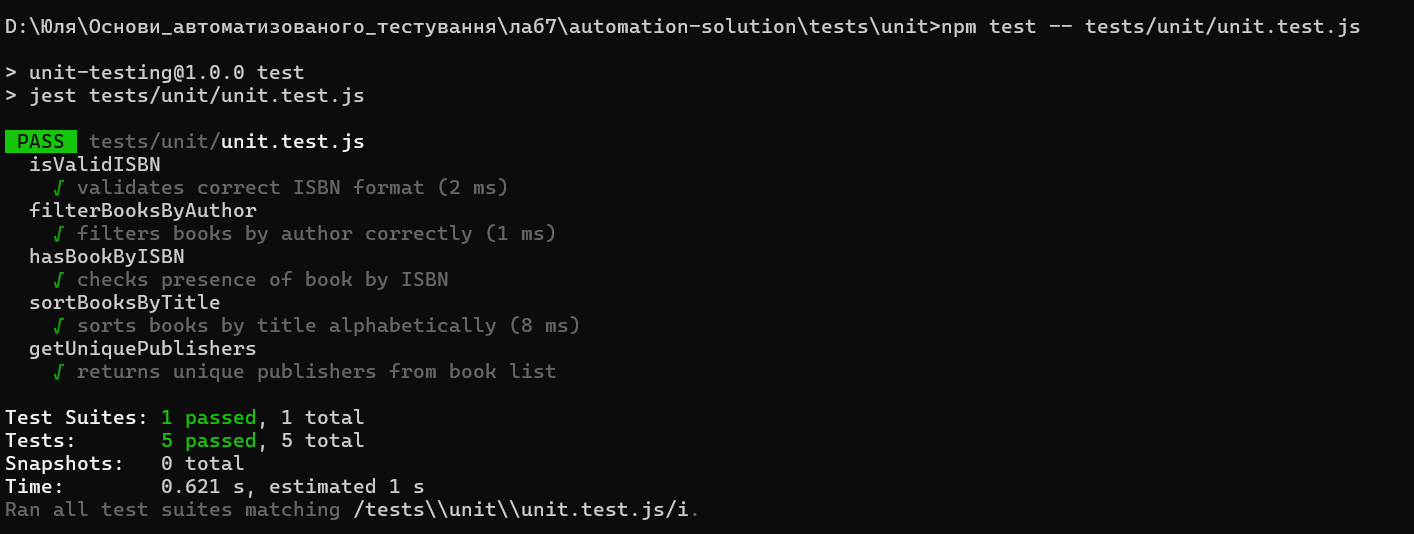
    expect(publishers).toContain('Pub 1');

    expect(publishers).toContain('Pub 2');

    expect(publishers.length).toBe(2);

  });

Результат:

****

**Висновок:** у ході лабораторної роботи було створено цілісне автоматизаційне рішення, що охоплює різні типи тестування: unit, UI, API та end-to-end. Рішення базується на модульній архітектурі з чіткою структурою, що відповідає принципам KISS та DRY. Це дозволяє забезпечити гнучкість, масштабованість та підтримуваність тестів.

**Посилання на репозиторій:** <https://github.com/yuliia-laba/lab7>