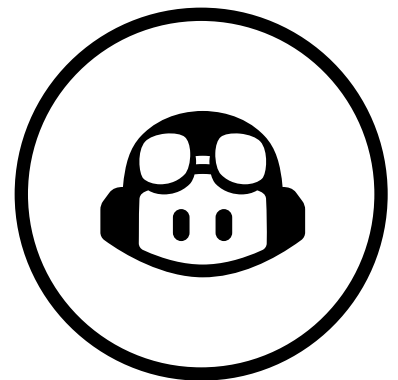


# THE TRANSFORMATIVE POTENTIAL OF **GitHub Copilot** IN **QA AUTOMATION**

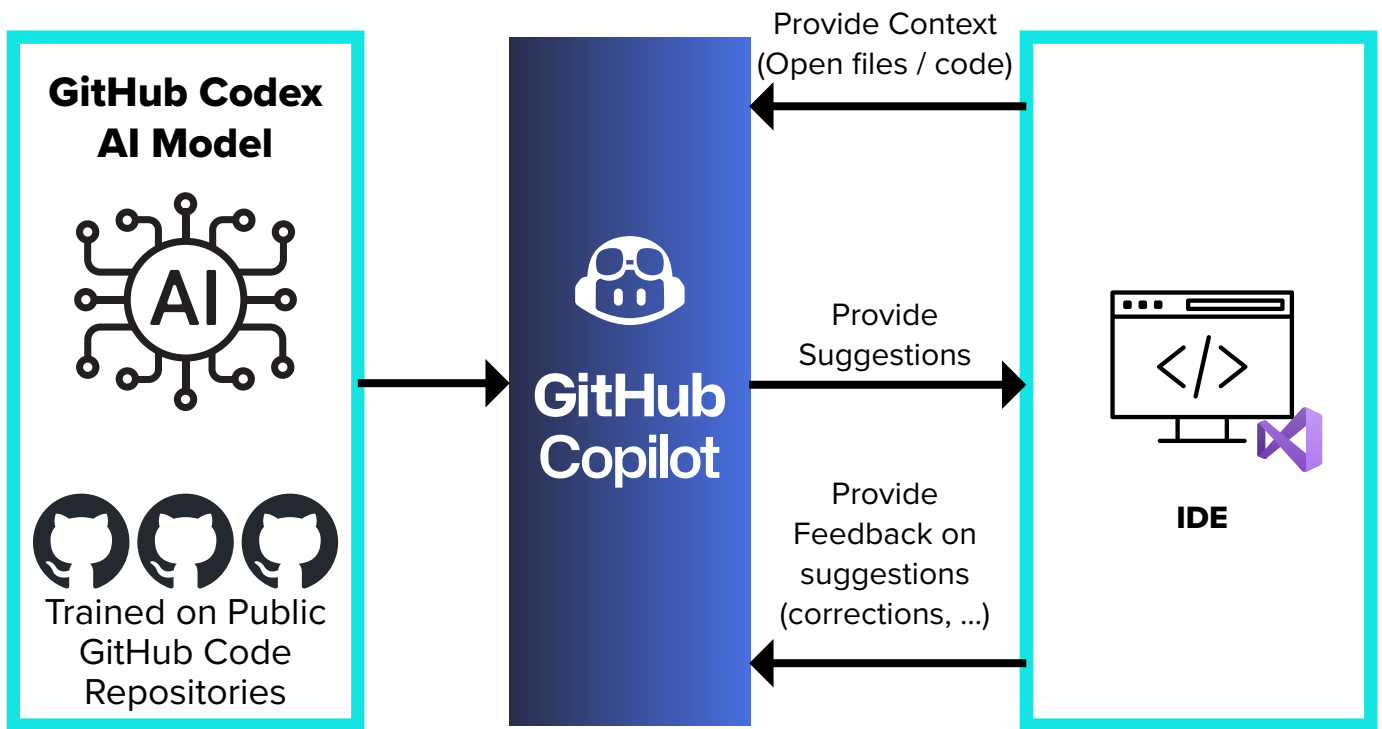
## Table of Content

1. How GitHub Copilot helps us in writing Automation Tests.....	3
2. Problem Statement: Testing Dynamic Single-Page Applications with Playwright....	6
3. Problem Statement: Chained API Requests with Complex Error Handling and Data Dependency.....	8
4. Practical Uses in Software Development....	10
5. Future Directions.....	13
6. Conclusion.....	14



**SAY 🖐️ TO YOUR AI  
PAIR PROGRAMMER**

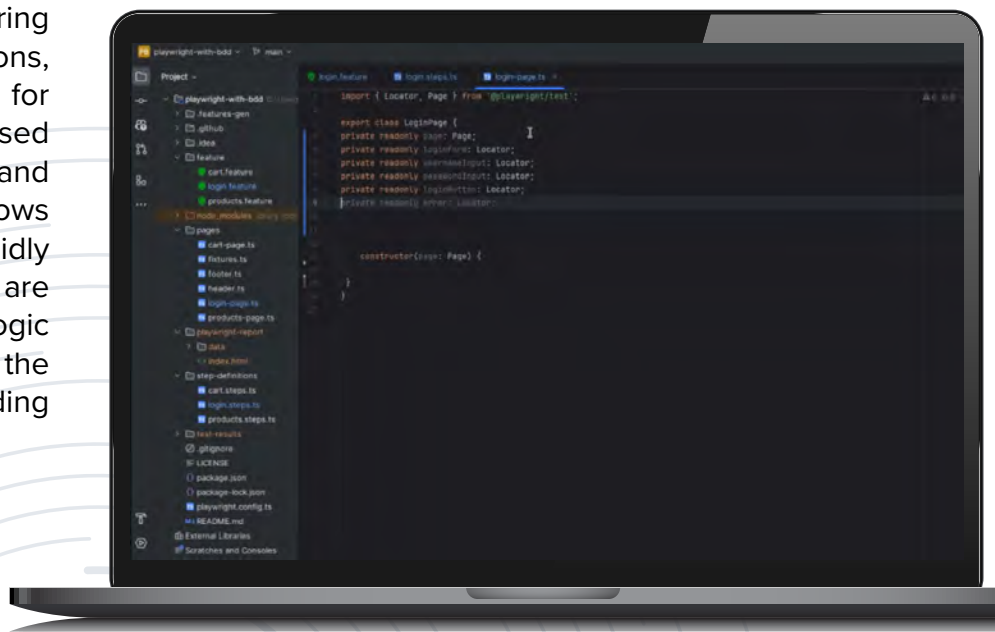
# 1 How GitHub Copilot helps us in writing Automation Tests



GitHub Copilot represents a significant advancement in the toolkit of automation testers by providing an AI-driven coding assistant directly within their development environment, specifically Visual Studio Code. Here are some key features of GitHub Copilot that are particularly beneficial for automation testers.

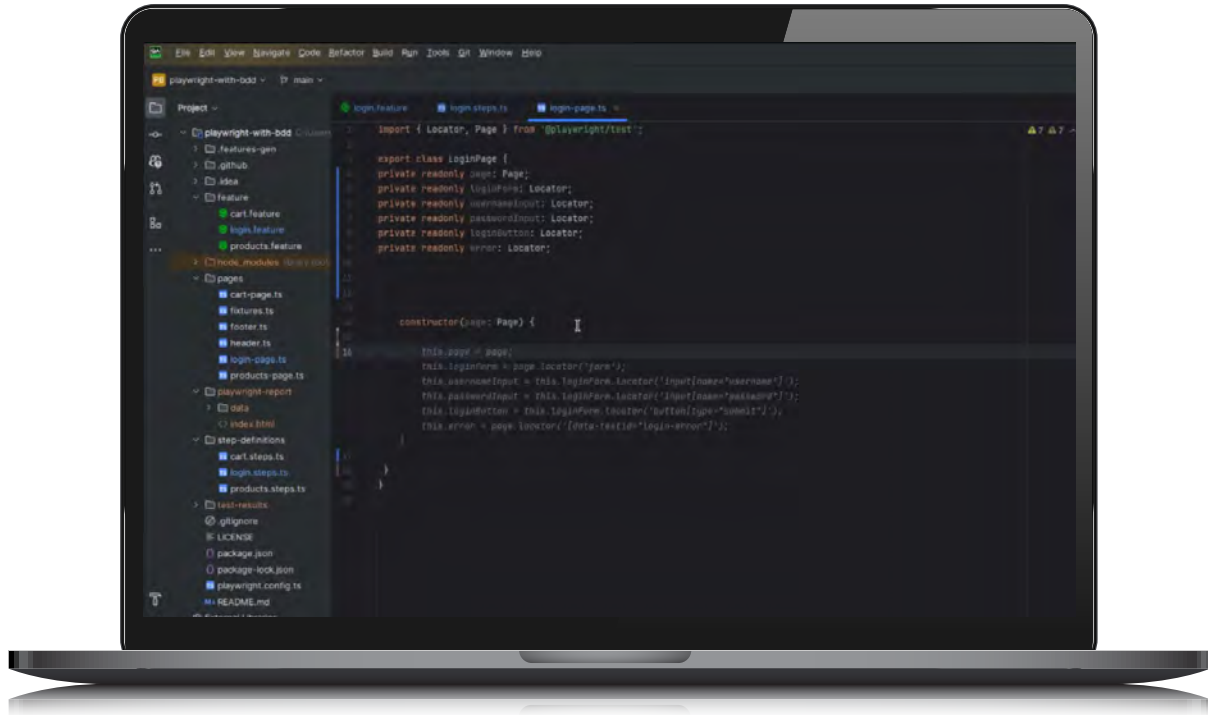
## 1.1 Context-Aware Code Suggestions

GitHub Copilot excels in offering context-aware code completions, not just for simple lines but for entire functions or blocks based on the current code and comments. This feature allows automation testers to rapidly generate test scripts that are aligned with the application's logic and requirements, minimizing the time spent on routine coding tasks.



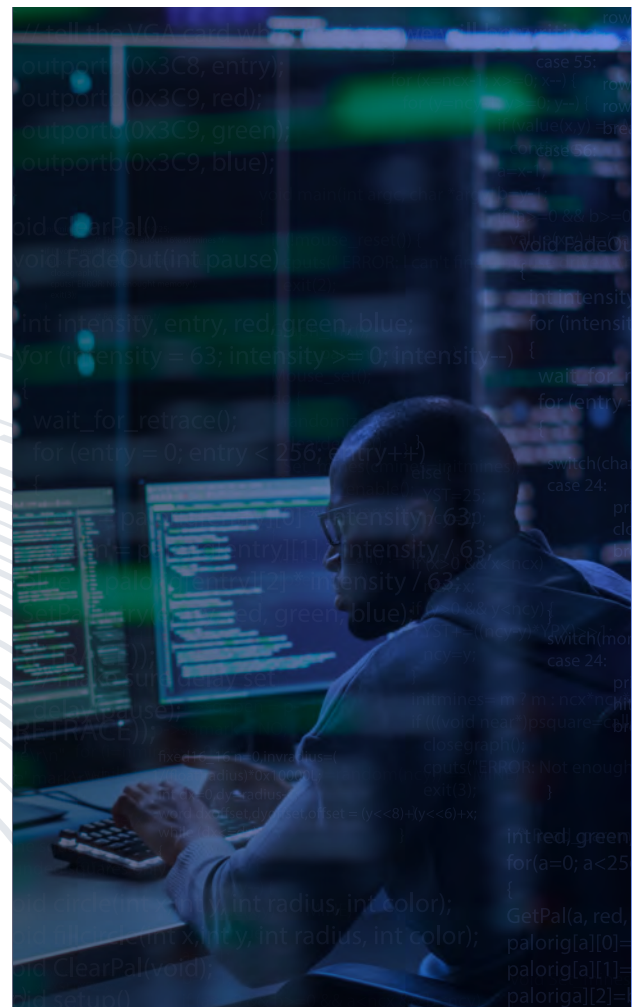
## 1.2 Real-Time Code Generation

Unlike traditional auto complete tools, Copilot can generate real-time, relevant code suggestions by predicting not just the next line of code but potentially the next several lines or entire functions. For testers, this means quicker script development and the ability to focus on higher-level test planning and execution.



## 1.3 Support for Multiple Programming Languages

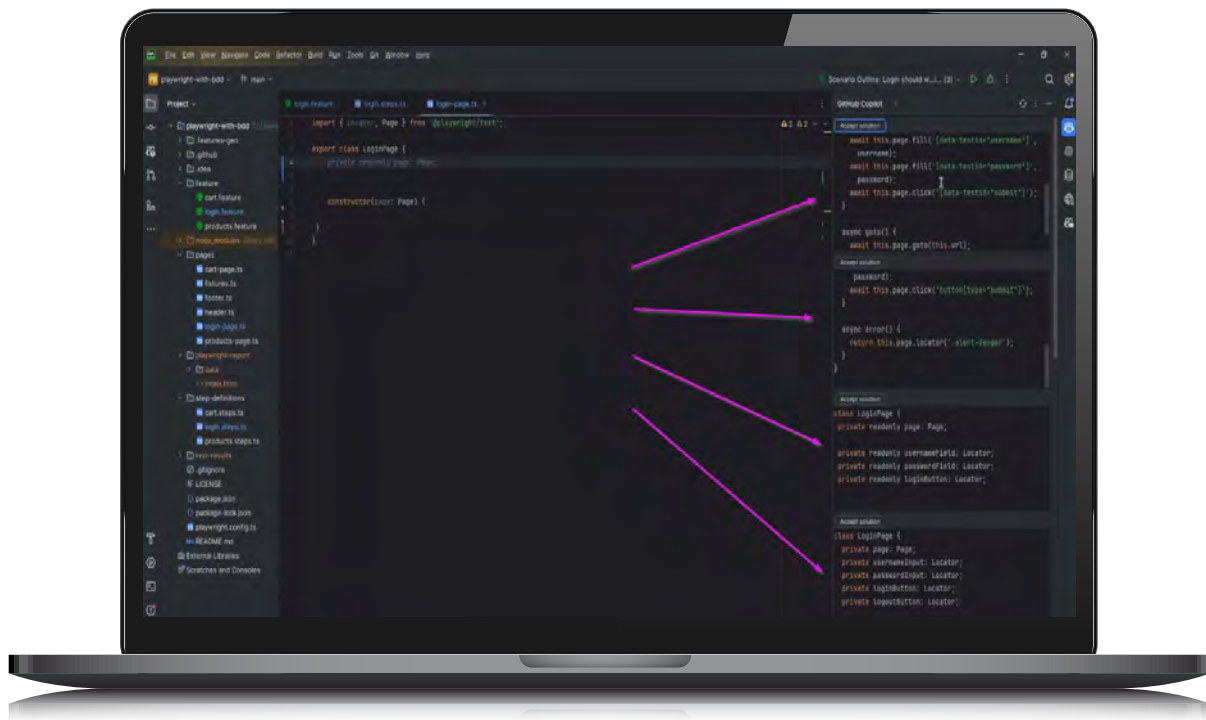
Copilot supports a wide array of programming languages and frameworks commonly used in automation testing, such as JavaScript, Python, Java, and more. This broad support ensures that automation testers can leverage Copilot's capabilities regardless of the tech stack used for their testing frameworks like Selenium, Cypress, or Playwright.





## 1.4 Code suggestions directly within Visual Studio Code

GitHub Copilot provides an intuitive interface for interacting with its code suggestions directly within Visual Studio Code, allowing users to quickly accept or reject suggestions as they code. GitHub Copilot aims to enhance productivity and facilitate a more efficient coding experience by providing intelligent suggestions that developers can easily accept, reject, or modify as they work.



For automation testers, GitHub Copilot is more than just a coding assistant; it acts as a catalyst for increased productivity, enhanced learning, and deeper focus on creating more effective and comprehensive test suites. By reducing the mechanical aspects of coding, Copilot allows testers to devote more energy to strategic tasks like test design and optimization, thereby enhancing the overall quality of software testing efforts.



# 2

## Problem Statement Testing Dynamic Single-Page Applications with Playwright

### Scenario

Imagine testing an SPA that includes a sequence of user interactions leading up to a final submission form. Each step may reveal new, dynamically loaded elements based on the data entered or choices made in the previous steps. These elements are not only loaded asynchronously but may also depend on data fetched from APIs in response to user inputs.

### Typical Issues in Complex Playwright Automation



#### Dynamic Content Loading

Ensuring elements are interactable as they load dynamically based on previous user actions or API responses.



#### State Dependency

Each step may alter the state of the application, which influences subsequent steps.



#### Robust Validation

Validating the state of the application at various stages to ensure correct behaviour throughout the test.

### How GitHub Copilot Resolves This Issue

#### 1 Writing Dynamic Selectors and Waits

GitHub Copilot can suggest appropriate dynamic selectors and wait commands that ensure elements are fully loaded and ready for interaction before proceeding. This is crucial in SPAs where elements may appear conditionally.

#### Example Code with Copilot:

As you type commands to interact with dynamically loaded elements, Copilot may suggest:

```
// Assume you have filled in the first part of a form and submitted it
await page.fill('#input-field', 'example data');
await page.click('#submit-button');

// Copilot suggests waiting for the next set of dynamic elements
await page.waitForSelector('#dynamic-loaded-element', { state: 'visible' })

// Interact with new elements
await page.check('#checkbox-for-options');
await page.click('#next-section-button');

// Copilot can suggest assertions to check if the SPA state is as expected
await expect(page.locator('#status-display')).toHaveText('Processing complete');
```

## 2 Handling Asynchronous Operations and API Interactions

Handling API calls that affect the UI can be tricky. Copilot can provide snippets to handle these effectively, ensuring that the UI updates are captured in the tests.

### Example Code with Copilot:

```
// Example of handling an API call response before proceeding
await page.click('#fetch-data-button');
await page.waitForResponse(response => response.url() === 'https://api.example.com/data');

// Validate the response has affected the page as expected
await expect(page.locator('#response-output')).toHaveText('Data loaded successfully');
```

## 3 Complex Scenario Management

For scenarios involving multiple dependent interactions, Copilot can help orchestrate these steps, ensuring that each action is performed in the correct order and validated properly.

### Example Code with Copilot:

```
// Navigate through a complex multi-step process
await page.fill('#start-date', '2021-01-01');
await page.fill('#end-date', '2021-12-31');
await page.click('#load-data-for-period');
await page.waitForSelector('#data-summary', { state: 'visible' });

// Copilot suggests checking that all parts of the summary are correct
await expect(page.locator('#summary-total')).toHaveText('Total: 1000');
await expect(page.locator('#summary-average')).toHaveText('Average: 50');
```

## Benefits of Using GitHub Copilot in Complex Playwright Automation



### Accelerates Development

Copilot can drastically speed up the process of writing complex automation scripts.

GitHub Copilot acts as a virtual assistant, suggesting contextually relevant code snippets and handling patterns that can navigate the complexities of modern web applications, particularly SPAs managed with Playwright. This tool aids in building robust, maintainable, and effective automation tests.



### Improves Code Quality

By suggesting optimized and proven coding patterns, Copilot helps maintain high quality and reliability in test scripts.



### Enhances Learning

For new developers or testers, Copilot serves as an on-the-fly educational tool, providing guidance on best practices in automation.

# 3

## Problem Statement Chained API Requests with Complex Error Handling and Data Dependency

### Scenario

In advanced API testing scenarios, multiple interconnected API calls might be required. For instance, fetching a user's authentication token, using that token to access a user-specific resource, and then performing an action based on that resource's data. Managing the dependencies between these calls and ensuring robust error handling can become quite complex.

### Typical Issues in Complex Cypress API Testing



#### Managing Dependencies

Handling the output of one API call and using it as an input for subsequent calls.



#### Dynamic Data Handling

Extracting and using dynamic data from responses.



#### Comprehensive Error Handling

Appropriately handling possible errors at each step of the chain.

### How GitHub Copilot Resolves This Issue

#### 1 Code Suggestions for Chained Requests

GitHub Copilot can suggest patterns for chaining requests in Cypress, where the output of one request is smoothly passed to the next, including handling errors that may occur in the chain.

#### Example Code with Copilot:

While writing the initial API request to fetch an authentication token, start typing, and Copilot might suggest.

```
// Fetch the authentication token
cy.request({
  method: 'POST',
  url: '/api/login',
  body: {
    username: 'exampleUser',
    password: 'examplePass'
  }
}).then((response => {
  expect(response.body).to.have.property('token');
  const authToken = response.body.token;

  // Use the token to fetch user-specific data
  cy.request({
    method: 'GET',
    url: '/api/user/data',
    headers: { 'Authorization': 'Bearer ${authToken}' }
  }).then((userDataResponse => {
    expect(userDataResponse.status).to.eq(200);
    // Extract and use data from the response
    const userData = userDataResponse.body;

    // Further action based on userData
    // Copilot might suggest handling userData to perform further actions
  }));
});
```



## 2 Advanced Error Handling

Copilot can help draft comprehensive error handling for each step of the API chain, ensuring the test suite can reliably handle and report errors.

### Example Code with Copilot:

```
cy.request({
  method: 'GET',
  url: '/api/resource',
  failOnStatusCode: false
}).then((response) => {
  if (response.status === 404) {
    cy.log('Resource not found, handling error..');
  } else {
    expect(response.body).to.include('expected data');
  }
});
```

## 3 Dynamic Data Validation

Copilot can assist in writing assertions for dynamically extracted data, ensuring that the validations are robust and cover various edge cases.

### Example Code with Copilot:

```
cy.request('GET', '/api/resource').then((response) => {
  const details = response.body.details;
  expect(details).to.have.property('name').and.to.not.be.null;
  // Copilot might suggest more specific checks depending on previous code patterns
});
```

## Benefits of Using GitHub Copilot in Complex Cypress API Automation



### Streamlined Code Generation

Quickly generates code snippets that fit into the broader context of the test scenario.



### Enhanced Reliability

Offers suggestions that ensure tests are both comprehensive and robust, particularly in handling edge cases and errors.



### Educational Tool

Acts as a learning aid by introducing advanced Cypress features and best practices through real-time coding assistance.

GitHub Copilot can significantly enhance productivity and test reliability by guiding through complex Cypress API test scenarios, ensuring that tests are well-structured, maintainable, and effective in catching issues early in the development cycle.

# 4

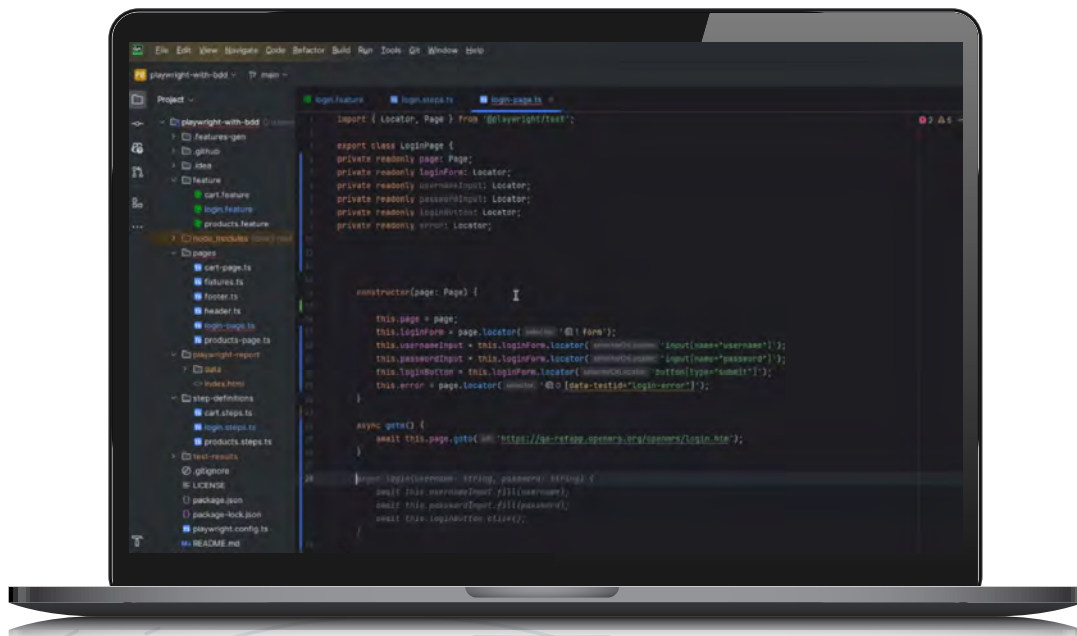
# Practical Uses in Software Development

GitHub Copilot, as a cutting-edge AI-driven code assistant developed by GitHub in collaboration with OpenAI, brings a revolutionary approach to software development. Its practical uses span across various stages of the software development lifecycle, enhancing productivity, improving code quality, and accelerating learning. Here's an overview of some key practical applications of GitHub Copilot in software development:

## 4.1 Generating Boilerplate Code

Automation testing often involves a lot of repetitive, boilerplate code, especially when setting up test environments or writing initial setups for tests. GitHub Copilot can quickly generate this boilerplate code based on minimal input from the tester, saving time and allowing testers to focus on more complex tasks.

**Example:** Automatically generating the setup and teardown methods for test classes in frameworks like pytest for Python or JUnit for Java.



## 4.2 Writing Comprehensive Test Cases

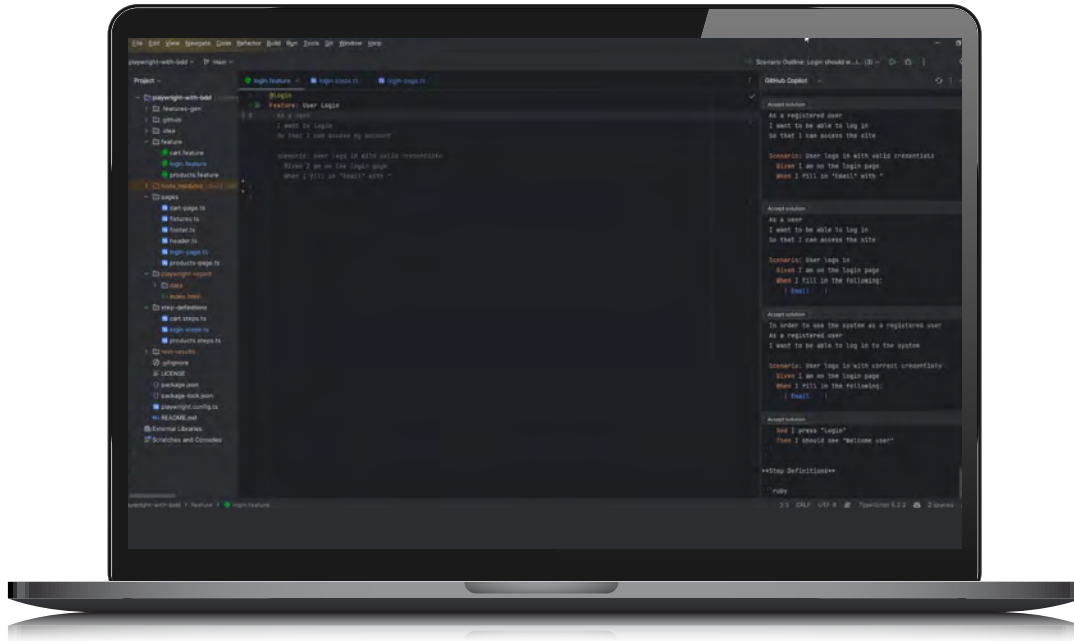
GitHub Copilot can suggest entire test cases based on the description provided in comments or based on the function signatures it analyses. This can be particularly useful in ensuring that all potential scenarios are covered, including edge cases that might not be immediately obvious.

**Example:** Generating test cases for boundary value analysis and error handling paths by simply describing the test intention in a comment.

## 4.3 Generating API Test Scripts

For API testing, writing tests can sometimes be monotonous and repetitive. GitHub Copilot can generate test scripts for different HTTP methods, handling various response codes and integrating parameterized testing seamlessly.

**Example:** Writing comprehensive REST API tests for all CRUD operations with appropriate validation checks.



## 4.4 Enhancing Test Coverage

With its vast knowledge base, GitHub Copilot can suggest tests that cover less obvious paths or errors in the code, significantly improving the coverage of the test suite without manual intervention.

**Example:** Suggesting tests for race conditions or concurrency issues in a piece of code handling multi-threaded operations.

## 4.5 Creating Mocks and Stubs

GitHub Copilot can assist in creating complex mocks and stubs required for unit testing, especially when dealing with external dependencies like databases, APIs, or third-party services.

**Example:** Generating code to mock network responses or database interactions, ensuring tests remain fast and reliable without external dependencies.

## 4.6 Data-Driven Testing

GitHub Copilot can help generate data-driven tests by providing templates that iterate over various sets of input data. This helps in validating the robustness of the application under test against various input combinations.

- Example: Creating parameterized test functions that run with multiple sets of data to validate form inputs on a web application.

## 4.7 Automated Security Testing

Security testing often involves checking for common vulnerabilities such as SQL injections, XSS, and CSRF. GitHub Copilot can provide code snippets or entire test scripts that help automate these security testing processes.

- Example: Generating scripts that attempt to exploit these vulnerabilities in a controlled testing environment.

## 4.8 Performance Testing Scripts

GitHub Copilot can assist in writing scripts for performance testing by generating code to simulate multiple users or high-load scenarios using tools like JMeter or Locust.

- Example: Creating user scenarios for load testing web applications, including user behaviour like logging in, navigating pages, and performing actions concurrently.

GitHub Copilot serves as a powerful tool for automation testers by reducing the time spent on routine tasks, enhancing learning opportunities, and encouraging the adoption of best practices. Its ability to provide context-aware code suggestions revolutionizes the way testers write and maintain their test suites, potentially leading to higher quality software and more efficient testing cycles.





# 5 Future Directions

The integration of GitHub Copilot into the world of automation testing opens up several exciting future directions and possibilities. As AI and machine learning technologies continue to evolve, the role of tools like Copilot is set to

become increasingly significant in shaping how automation testers work and contribute to software development. Here are some potential future directions for GitHub Copilot in the realm of automation testing:

## 5.1 Integration with Additional Tools and Platforms

Expanding the integration capabilities of GitHub Copilot to seamlessly work with a broader range of testing frameworks and tools, such as Selenium, Cypress, Appium, and beyond, could significantly enhance its utility. Integration with CI/CD pipelines and version control systems could also streamline testing and deployment processes, making Copilot an even more indispensable part of the development lifecycle.

## 5.3 Learning and Adaptation Enhancements

Future versions of GitHub Copilot could feature improved learning algorithms that adapt more dynamically to the coding habits of users. This could involve more sophisticated pattern recognition capabilities to better predict and suggest relevant code, reducing the cognitive load on testers and speeding up the development process.

## 5.2 Advanced Code Analysis and Test Generation

Leveraging more advanced AI techniques, Copilot could evolve to not only suggest code but also analyze existing code bases for potential inefficiencies or errors. It could automatically generate test cases that cover missing or under-tested code paths, greatly improving test coverage and application reliability.

## 5.4 Ethical AI Use and Transparency

As AI's role in development expands, ensuring ethical use and transparency in how AI models like Copilot are trained and operate will be crucial. Future developments could include more transparent AI processes and the ability for users to understand and control what data is used to train these models.

The future of GitHub Copilot in automation testing looks promising, with potential enhancements that could transform it from a helpful tool to an essential part of every tester's toolkit. As Copilot evolves, it could play a pivotal role in driving innovation, efficiency, and effectiveness in the testing domain, ultimately leading to higher quality software products and faster development cycles.

# 6 Conclusion

In conclusion, GitHub Copilot stands as a transformative tool for automation testers, significantly enhancing their ability to write, review, and maintain test scripts efficiently. By leveraging advanced AI to generate code suggestions directly in Visual Studio Code, Copilot not only boosts productivity but also improves the quality of test scripts by embedding best practices and modern programming paradigms. However, while Copilot offers numerous advantages, it also introduces challenges that require careful consideration,

such as ensuring code integrity and managing the over-reliance on automated suggestions. For automation testers, GitHub Copilot is not just a tool but a potential catalyst for innovation, driving higher standards of testing and software development when used judiciously alongside human expertise. As the field of software testing continues to evolve, embracing tools like GitHub Copilot can be pivotal, provided testers remain vigilant and proactive in integrating AI capabilities into their workflows.

## References

GitHub Copilot Documentation  
OpenAI Research Papers  
Studies on AI impact in test automation

## Appendix

A: Installation Guide for GitHub Copilot in VSCode  
B: FAQs on GitHub Copilot Usage and Capabilities  
C: Case Studies of GitHub Copilot in Large-Scale Enterprises

This white paper serves as a comprehensive guide to understanding and leveraging GitHub Copilot within Visual Studio Code to improve coding practices, accelerate learning curves, and enhance overall productivity.

## About the Author



**Akhilesh Shukla**

Senior Test Architect, PDES

Akhilesh Shukla is a seasoned Senior Test Architect with extensive experience in software testing and quality assurance, currently leading QA initiatives at Happiest Minds Technologies. Renowned for his expertise in advanced test automation, Akhilesh excels in leveraging tools like Cypress and Playwright to enhance software quality. He is continuously contributing to the Testing Practice across many automation solutions and has a strong background in designing and implementing comprehensive testing strategies.

**For more information, write to us at [business@happiestminds.com](mailto:business@happiestminds.com)**

### About Happiest Minds

Happiest Minds Technologies Limited (NSE: HAPSTMNDS), a Mindful IT Company, enables digital transformation for enterprises and technology providers by delivering seamless customer experiences, business efficiency and actionable insights. We do this by leveraging a spectrum of disruptive technologies such as: artificial intelligence, blockchain, cloud, digital process automation, internet of things, robotics/drones, security, virtual/ augmented reality, etc. Positioned as 'Born Digital . Born Agile', our capabilities span Product & Digital Engineering Services (PDES), Generative AI Business Services (GBS) and Infrastructure Management & Security Services (IMSS). We deliver these services across industry groups: Banking, Financial Services & Insurance (BFSI), EdTech, Healthcare & Life Sciences, Hi-Tech and Media & Entertainment, Industrial, Manufacturing, Energy & Utilities, and Retail, CPG & Logistics. The company has been recognized for its excellence in Corporate Governance practices by Golden Peacock and ICSI. A Great Place to Work Certified™ company, Happiest Minds is headquartered in Bengaluru, India with operations in the U.S., UK, Canada, Australia, and the Middle East.