

Declaration

I, [Yujie Hu], declare that this assignment, titled [A3: research report], is my own original work and has not been copied from any other source except where explicitly acknowledged. I have not engaged in plagiarism, collusion, or any other form of academic misconduct in the preparation and submission of this assignment. All sources of information and data used in this assignment have been properly cited and referenced in accordance with the prescribed guidelines. I have not used unauthorized assistance in the preparation of this assignment and have not allowed any other student to copy my work. I am aware that any breach of academic integrity may result in disciplinary action as per the [policies of Monash University](#), which may include failing this assignment or the course, and further academic penalties.

Signature: _____ Yujie Hu _____

Date: _____ 15/10/2024 _____

Self-Evaluation

Rate your performance for each criteria. Put a ☒ (tick) in the box where you think your work belongs.

Criteria	Exceeds Expectations	Meets Expectations	Needs Improvement	Fail to meet expectations
Introduction	<input checked="" type="checkbox"/>			
Main Body	<input checked="" type="checkbox"/>			
Reflection	<input checked="" type="checkbox"/>			
AI Acknowledgement	<input checked="" type="checkbox"/>			

Section 1: Introduction

JavaScript plays a vital role in modern web development, driving interactive and dynamic user experiences across a variety of platforms. As JavaScript projects grow in complexity, the debugging process becomes increasingly time-consuming and error-prone. To address this, generative AI (GenAI) has emerged as a promising tool to simplify and enhance the debugging process. Leveraging machine learning algorithms trained on extensive datasets of code, GenAI tools can analyse JavaScript code, recognise patterns, and suggest potential fixes, which can significantly reduce the manual effort required.

This report explores five strategies for using GenAI to enhance JavaScript debugging, focusing on their potential benefits, drawbacks, and practical use cases. The relevance of this topic lies in its potential to

revolutionize modern web development, making debugging faster, smarter, and more intuitive. As development cycles shorten and the demand for error-free, responsive applications grows, GenAI-driven tools could become indispensable for JavaScript developers, facilitating smoother workflows and improving code quality.

Section 2: Main Body

2.1 Strategy 1: Bug Detection through Pattern Recognition

GenAI helps developers detect bugs by analyzing code patterns and identifying potential issues. AI tools like GitHub Copilot, powered by GPT, go beyond basic error detection offered by IDEs like Visual Studio Code. While IDEs can catch syntax errors and simple logic mistakes, Copilot goes further by identifying logical errors and other complex issues, such as the misuse of asynchronous functions or incorrect API calls. By leveraging machine learning models trained on large datasets, Copilot can understand the broader context of the code, allowing it to detect deeper, more nuanced errors [1].

Benefits:

- Early bug detection helps developers prevent syntax and runtime errors.
- Reduces time spent manually reviewing code.

Drawbacks:

- AI models may fail to recognise bugs in non-standard code patterns that lack sufficient training data.

The following snapshot shows how Copilot helps in identifying codes with logic error.



The screenshot displays the GitHub Copilot interface within a code editor. On the left, a chat window titled "why the app always redirect me to the login page" shows a conversation with Copilot. The user's query is: "The issue is that the redirection to the login page happens before the registration process completes. This is because the `router.push('/FireLogin')` line is executed immediately, without waiting for the `createUserWithEmailAndPassword` promise to resolve. To fix this, you should move the redirection inside the `.then()` block, so it only happens after the registration is successful." On the right, the code editor shows a JavaScript function `const register = () => {`. The function calls `router.push('/FireLogin')` on line 21, followed by `createUserWithEmailAndPassword(auth, email.value, password.value)` on line 23. The `createUserWithEmailAndPassword` function has a `.then((data) => {` block on line 24, which contains `console.log('Firebase Register Successful!')` on line 25 and `.catch((error) => {` on line 27, which contains `console.log(error.code)` on line 28. The `register` function ends on line 30 with `}`. The code is enclosed in `</script>` on line 31. Line numbers 19 through 32 are visible on the left margin of the code editor.

2.2 Strategy 2: Real-Time Error Correction and Feedback

Once bugs are detected, GenAI can assist by suggesting fixes based on the context of the code. Instead of simply highlighting issues, AI-powered tools like GitHub Copilot provide real-time code completion and bug fix suggestions. The tool can generate contextually appropriate code based on the developer's input, minimizing the time spent on debugging by automatically writing out potential fixes [2].

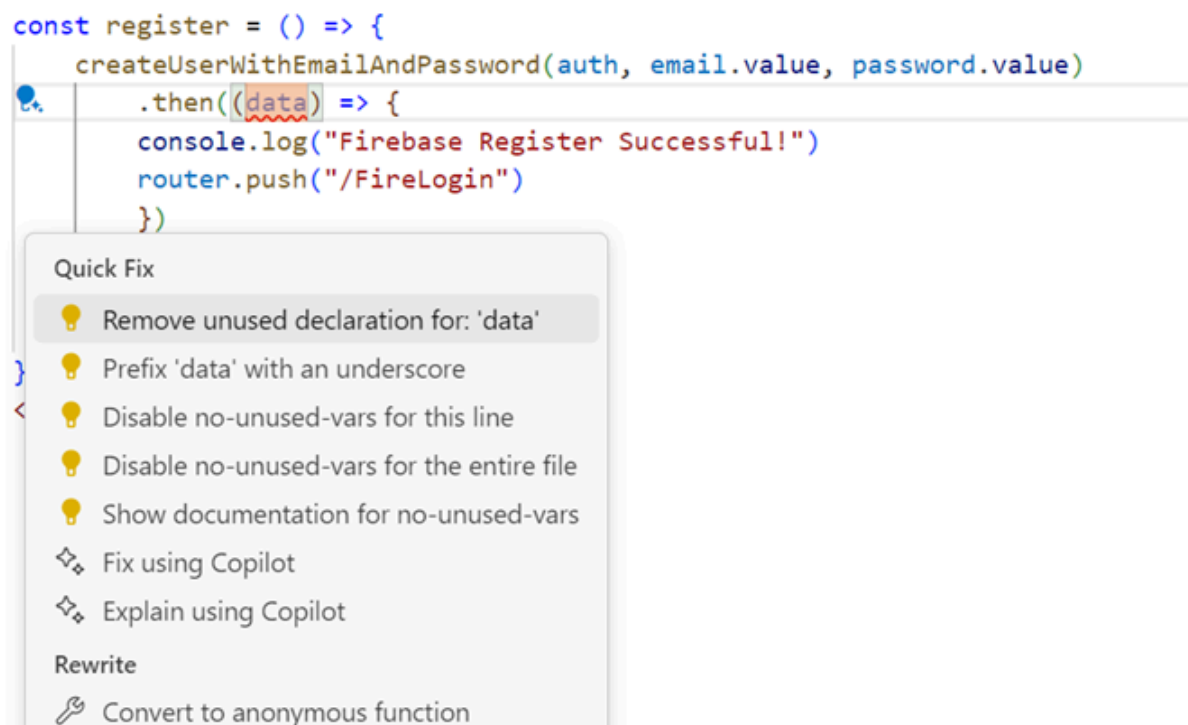
Benefits:

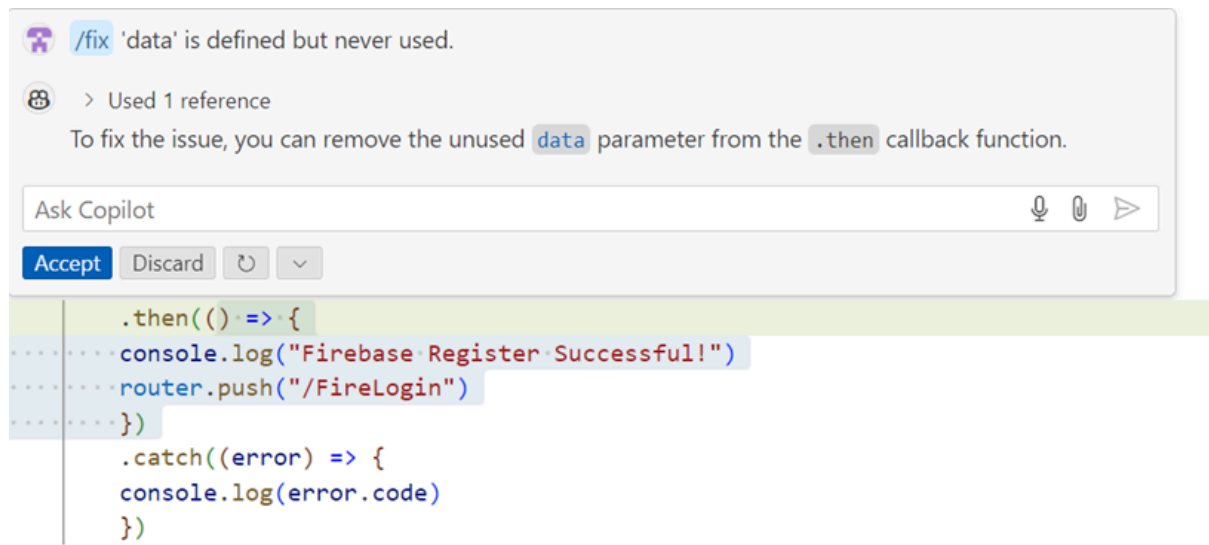
- Accelerates development by offering solutions as errors occur.
- Enhances the learning experience for novice developers by providing detailed feedback on mistakes.

Drawbacks:

- Developers may become overly reliant on AI, potentially limiting their ability to solve problems independently.

The snapshots demonstrate how Copilot helps fix an error related to the unused "data" parameter. The first snapshot shows the JavaScript code before debugging, while the second snapshot illustrates the auto-generated code provided by Copilot.





2.3 Strategy 3: Auto-Generation of Debugging Tests

Unit tests are essential for ensuring that individual pieces of code work correctly. However, writing comprehensive tests can be a tedious and time-consuming process. GenAI tools can help by automatically generating unit tests based on code structure, identifying edge cases and logical conditions, and simulating a variety of inputs.

Github Copilot Chat can auto-generate tests to validate JavaScript functionality under different inputs reducing the likelihood of bugs slipping through the cracks [2]. This can include common cases, boundary values, and invalid data scenarios.,

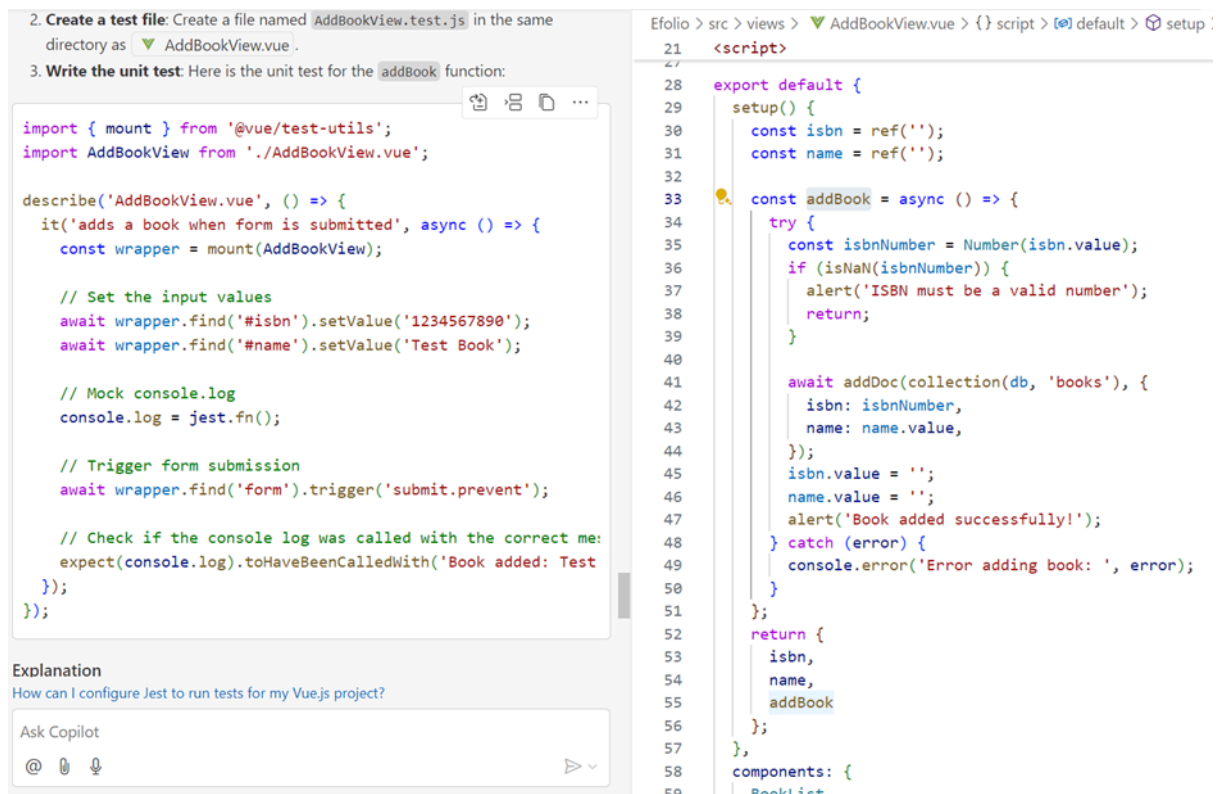
Benefits:

- Automates the creation of test cases, reducing manual workload.
- Ensures thorough test coverage by identifying edge cases developers might miss.

Drawbacks:

- AI-generated tests may not always reflect real-world use cases or developer intent, leading to unnecessary or irrelevant tests.

The following snapshot illustrates how Copilot is used to generate unit test (left side) for the addBook function on the right side.



2.4 Strategy 4: Interactive Debugging with AI Chatbots

AI-driven chatbots enhance the debugging process by allowing developers to describe issues in natural language, making it easier to communicate complex problems. These chatbots analyze the current state of the code and provide tailored suggestions, such as step-by-step debugging instructions or relevant code snippets. This approach is particularly helpful for developers who may not be familiar with specific frameworks or libraries, as the chatbot offers clear guidance. GitHub Copilot Chat, for example, acts as an AI-powered assistant that interprets developer input and delivers targeted solutions to fix or debug JavaScript code effectively [2].

Benefits:


- Improves the accessibility of debugging tools for less experienced developers.
- Speeds up the identification of potential solutions through personalized suggestions.

Drawbacks:

- Natural language input may lead to ambiguous or incomplete interpretations, limiting the effectiveness of the suggestions.

The snapshot below shows my conversation with the Copilot chatbot in plain language.

why is my submitRating function not working appropriately?

 **GitHub Copilot**

> Used 1 reference

To diagnose why your `submitRating` function is not working appropriately, let's first review the function and then write a unit test to identify potential issues.

Step-by-Step Plan

1. **Review the `submitRating` function:** Understand what the function does.
2. **Set up Jest and Vue Test Utils:** Ensure the testing environment is ready.
3. **Write the unit test:** Create a test file and write a test for the `submitRating` function.

Step 1: Review the `submitRating` Function

Let's assume your `submitRating` function looks something like this:

```
<template>
  <form @submit.prevent="submitRating(providerId)">
    <div class="row text-center">
      <div class="col-sm-6">
        <label for="rating">Rate the provider (1-5):</label>
        <input type="number" id="rating" v-model.number="rating">
      </div>
    </div>
  </form>
</template>
```

How can I troubleshoot issues with the updateRating action in my Vue.js component?

Ask Copilot

@ 📎 🗣️ ➤

2.5 Strategy 5: AI-Powered Code Refactoring

GenAI can also assist in code refactoring, a process that improves code readability and structure without changing its external behaviour. This is especially important for long-term maintainability and reducing the likelihood of bugs.

GitHub Copilot can suggest code refactoring options to improve the structure and maintainability of JavaScript code. By analyzing code patterns, Copilot identifies sections that can be optimized or simplified, ensuring that the code remains clean and easier to debug over time [2].

Benefits:

- Refactoring helps reduce technical debt, leading to fewer bugs in the future.
- Increases code clarity, which aids in debugging and maintenance.

Drawbacks:

- Refactoring suggestions may sometimes conflict with existing design patterns or stylistic preferences, requiring manual review by developers.

The following snapshot shows how Copilot helps in refactoring complex function (original codes on right side).

Here's the refactored `handleLogin` function, breaking it down into smaller, more manageable functions:

```
<script setup>
import { ref } from 'vue'
import { useRouter } from 'vue-router'
import store from '@store/store'
import { userAccounts } from '@store/userAccount'

const router = useRouter()

// Define reactive state for form inputs and error message
const email = ref('')
const password = ref('')
const errorMessage = ref('')

const adminUsername = 'admin@monash'

const handleLogin = () => {
  const user = findUserByEmail(email.value);

  if (user) {
    if (isAdmin(user)) {
      handleAdminLogin(user);
    } else {
      errorMessage.value = 'User not found';
    }
  }
};

const findUserByEmail = (email) => {
  return userAccounts.find((account) => account.username === email);
};

const isAdmin = (user) => {
  return user.username === adminUsername;
};

const handleAdminLogin = (user) => {
  if (user.password === password.value) {
    console.log('Login successful, admin');
    clearErrorMessage();
    authenticateUser(email.value, 'admin');
    router.replace('/home');
  } else {
    errorMessage.value = 'Incorrect password';
  }
};

const handleUserLogin = (user) => {
  if (user.password === password.value) {
    console.log('Login successful, normal user');
    clearErrorMessage();
  }
};
```

```
60 <script setup>
73
74 const adminUsername = 'admin@monash'
75
76 const handleLogin = () => {
77   // find the user account
78   const user = userAccounts.find(
79     (account) => account.username === email.value
80   );
81
82   if (user) {
83     if (user.username === adminUsername && user.password === password.value) {
84       console.log('Login successful, admin')
85       errorMessage.value = '' // Clear the error message
86       store.state.isAuthenticated = true
87       store.state.user = email
88       store.state.usertype = 'admin'
89       router.replace('/home') // Navigate to homepage if login succeeded
90     } else if (user && user.password === password.value) {
91       console.log('Login successful, normal user')
92       errorMessage.value = '' // Clear the error message
93       store.state.isAuthenticated = true
94       store.state.user = email
95       store.state.usertype = 'aged user'
96       router.replace('/home') // Navigate to homepage if login succeeded
97     }
98   } else {
99     console.log('Login failed')
100   }
101 }
102
103 </script>
104
105 <style scoped>
106
107 </style>
108
109
110
111
112
```

```
    errorMessage.value = 'Incorrect password';  
  }  
};  
  
const clearErrorMessage = () => {  
  errorMessage.value = '';  
};  
  
const authenticateUser = (email, userType) => {  
  store.state.isAuthenticated = true;  
  store.state.user = email;  
  store.state.usertype = userType;  
};  
</script>
```

1 vulnerability >

Explanation

1. **findUserByEmail Function:** Finds the user by email.
2. **isAdmin Function:** Checks if the user is an admin.
3. **handleAdminLogin Function:** Handles the login logic for admin users.
4. **handleUserLogin Function:** Handles the login logic for regular users.
5. **clearErrorMessage Function:** Clears the error message.
6. **authenticateUser Function:** Authenticates the user and updates the store.

This refactoring breaks down the `handleLogin` function into smaller, more manageable functions, making the code easier to read and maintain.

Section 3: Reflection

The integration of GenAI tools like GitHub Copilot into JavaScript debugging workflows marks a significant advancement in software development. These tools not only automate repetitive tasks such as bug detection and unit test generation but also provide intelligent suggestions for fixing and refactoring code. This reduces the burden on developers, allowing them to focus on more complex and creative aspects of development.

However, the reliance on AI presents ethical challenges, including over-dependence on automated systems and the potential for reinforcing biases embedded in training data. Developers must ensure that AI-generated suggestions align with project goals and ethical standards. As AI technology continues to evolve, maintaining transparency, accountability, and human oversight will be essential. While these tools will likely become more advanced, collaboration between AI and human developers will remain crucial to avoid misuse and uphold ethical standards in software development.

Section 4: Conclusion

Generative AI has the potential to revolutionize JavaScript debugging by automating time-consuming tasks such as bug detection, code refactoring, and unit test generation. Tools like GitHub Copilot provide developers with powerful solutions to streamline the debugging process and maintain clean, efficient codebases. While these tools offer numerous benefits, developers must remain mindful of their limitations and ensure they maintain control over the code they produce.

Section 5: Acknowledgement of AI use

I have used Open AI GPT to brainstorm the ideas how GenAI can help in debugging process, as well as use it to rephrase my paragraphs to more explaining text after I research, analyse and discuss the related strategies.

References

[1] Swimm.io, "AI Tools for Developers: AI Code Review – How It Works and 3 Tools You Should Know," 2023. [Online]. Available: <https://swimm.io/learn/ai-tools-for-developers/ai-code-review-how-it-works-and-3-tools-you-should-know>

[2] GitHub, "GitHub Copilot: Your AI Pair Programmer," *GitHub Docs*, 2022. [Online]. Available: <https://github.com/features/copilot/>.