PARTH PATEL (1233553622)

SRINIVAS OGURI (1232467017)

PART - 2

*(F)*

***In What Ways Are the Results the Same?***

All of PMD, COPILOT AND Chatgpt identified unnecessary semicolons as a readability issue.

Apart from that, they also showed some unconventional structure in “for” loop.

All of them pointed to the thread-unsafe nature of the static cache in PowerCache either directly like ChatGPT and Copilot did or indirectly like PMD did by mentioning it in PowerCached.\_\_cache.

Lastly, All tools recommended to improve comments and overall readability.

***In What Ways Are the Results the Different?***

Both Chatgpt and Copilot handled edge cases efficiently by flagging absence of handling of negative power, however PMD failed to so which may to result of the ruleset provided to it.

PMD focused more on issues like annotations, control statements and unnecessary semicolons.

Copilot gave a more balanced answer by discussing efficiency, cache, and potential bugs and thread safety.

However, ChatGPT also took architectural considerations like SOLID principle in detail apart from discussing efficiency,cache,bugs and thread safety.

#### **Do They Perform Static Analysis Using the Same Methods?**

While PMD relies on predefined rulesets for analyzing syntax, structure, and code smells.

Gen AI tools like ChatGPT and Copilot use AI to infer patterns, best practices and logic flaws.

While PMD does not evaluate all kind of logical designs and flaws and architecture and thread safety using the default ruleset. Custom Rules can be added to check for it.

While Gen AI does not need to provided with custom rules as their analysis depends on context provided in the code and questions asked based on their training data.

**Which Tool Would I Choose and Why?**

**PARTH :**

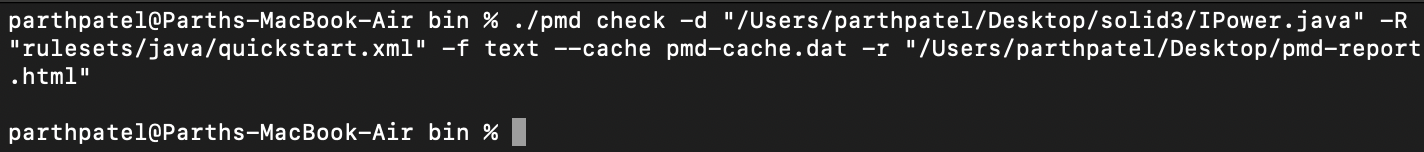
In the current scenario, I would chose static analysis is very important in large scale projects and GenAI tools cannot be scaled to that scale but PMD can be.

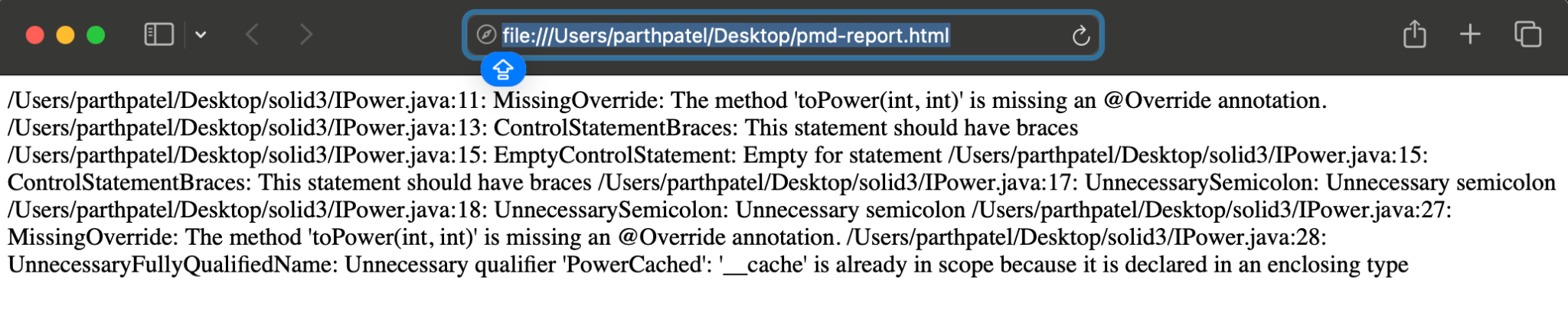
Apart from that, it is not very difficult to write custom rules for PMD for thread safety, caching and architecture.

Apart from that result from PMD are consistent as opposed to GenAI results which can vary.

In future GenAI might be a better choice but right PMD is far better than GenAI tools like ChatGpt and Copilot.

*(A)*





**In text format :**

/Users/parthpatel/Desktop/solid3/IPower.java:11: MissingOverride: The method 'toPower(int, int)' is missing an @Override annotation. /Users/parthpatel/Desktop/solid3/IPower.java:13: ControlStatementBraces: This statement should have braces /Users/parthpatel/Desktop/solid3/IPower.java:15: EmptyControlStatement: Empty for statement /Users/parthpatel/Desktop/solid3/IPower.java:15: ControlStatementBraces: This statement should have braces /Users/parthpatel/Desktop/solid3/IPower.java:17: UnnecessarySemicolon: Unnecessary semicolon /Users/parthpatel/Desktop/solid3/IPower.java:18: UnnecessarySemicolon: Unnecessary semicolon /Users/parthpatel/Desktop/solid3/IPower.java:27: MissingOverride: The method 'toPower(int, int)' is missing an @Override annotation. /Users/parthpatel/Desktop/solid3/IPower.java:28: UnnecessaryFullyQualifiedName: Unnecessary qualifier 'PowerCached': '\_\_cache' is already in scope because it is declared in an enclosing type

*(B)*

**CHAT-GPT VERSION :**

**Conversation link :** [**https://chatgpt.com/share/6747e11c-95c8-8013-b968-35a2f73dcc35**](https://chatgpt.com/share/6747e11c-95c8-8013-b968-35a2f73dcc35)

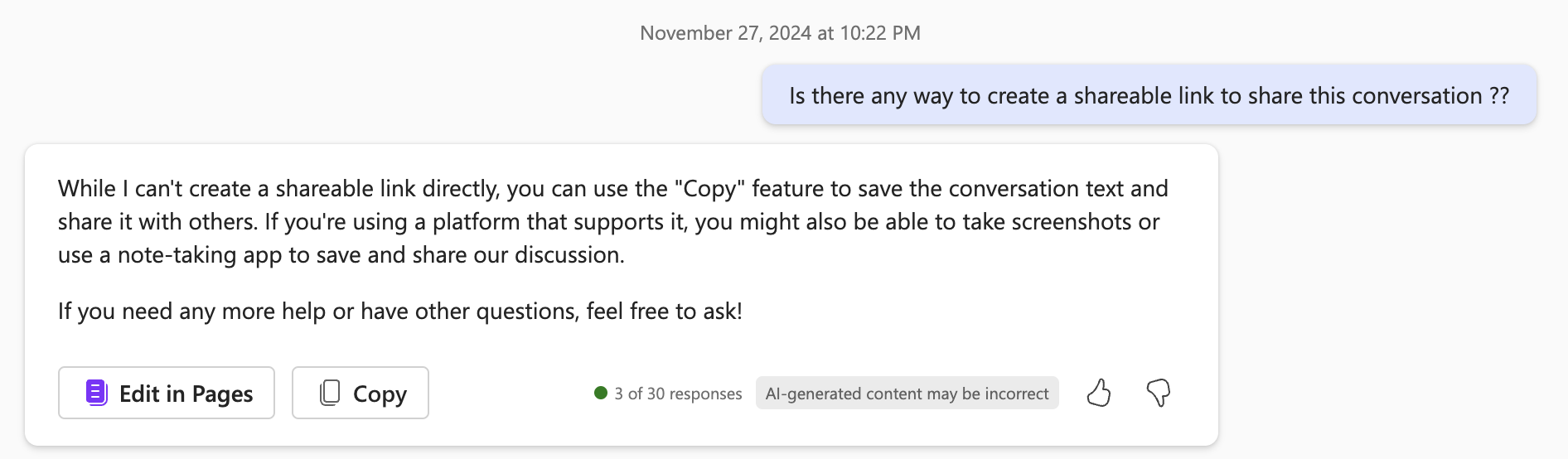
### **Recommendations**

**Implement the following changes for improved performance, reliability, and readability:**

1. **Synchronize cache access or use ConcurrentHashMap for thread safety.**
2. **Add missing cache initialization logic in PowerCached.toPower.**
3. **Replace unconventional loops in PowerSimple with standard structures.**
4. **Consider introducing a bounded cache to prevent memory issues.**

(C)

**CO-PILOT VERSION :**

**Conversation link : **

**Sure, let's run a static analysis on your code. Here are some observations and potential improvements:**

### **1. Code Structure and Readability**

* **Extra Semicolons: In PowerSimple, the for loop contains unnecessary semicolons which can be removed for better readability.**

**for (int i = 0; i < pow; res \*= n, i++);**

### **2. Efficiency and Performance**

* **Caching Mechanism: The PowerCached class uses a nested Map for caching results. This can be optimized by using a single Map with a composite key or a more efficient caching strategy.**

**private static Map<String, Integer> cache = new HashMap<>();**

**public int toPower(int n, int pow) {**

**String key = n + "," + pow;**

**return cache.computeIfAbsent(key, k -> computePower(n, pow));**

**}**

**private int computePower(int n, int pow) {**

**if (pow == 0) return 1;**

**return n \* computePower(n, pow - 1);**

**}**

### **3. Thread Safety**

* **Static Cache: The static cache in PowerCached is not thread-safe. If this class is used in a multi-threaded environment, consider using ConcurrentHashMap or synchronizing access to the cache.**

**private static Map<Integer, Map<Integer, Integer>> \_\_cache = new ConcurrentHashMap<>();**

### **4. Code Comments and Documentation**

* **Comments: Ensure comments are clear and concise. The comment // resolves the inefficiency in AlmostBest is not very informative. Provide more context or remove it if unnecessary.**

### **5. Potential Bugs**

* **Null Check: In PowerCached, the toPower method does not update the cache if entry is null. This can lead to redundant calculations.**

**if (entry == null) {**

**entry = new HashMap<>();**

**PowerCached.\_\_cache.put(n, entry);**

**}**

### **6. Design Principles**

* **Single Responsibility Principle: Both PowerSimple and PowerCached adhere to the Single Responsibility Principle by focusing on the power calculation. However, consider separating the caching logic into a different class if it grows more complex.**

### **7. Edge Cases**

* **Negative Powers: The current implementation does not handle negative powers. Consider adding logic to handle such cases if required.**

**By addressing these points, you can improve the readability, performance, and robustness of your code. If you have any specific concerns or need further assistance, feel free to ask!**

(D)

(E)

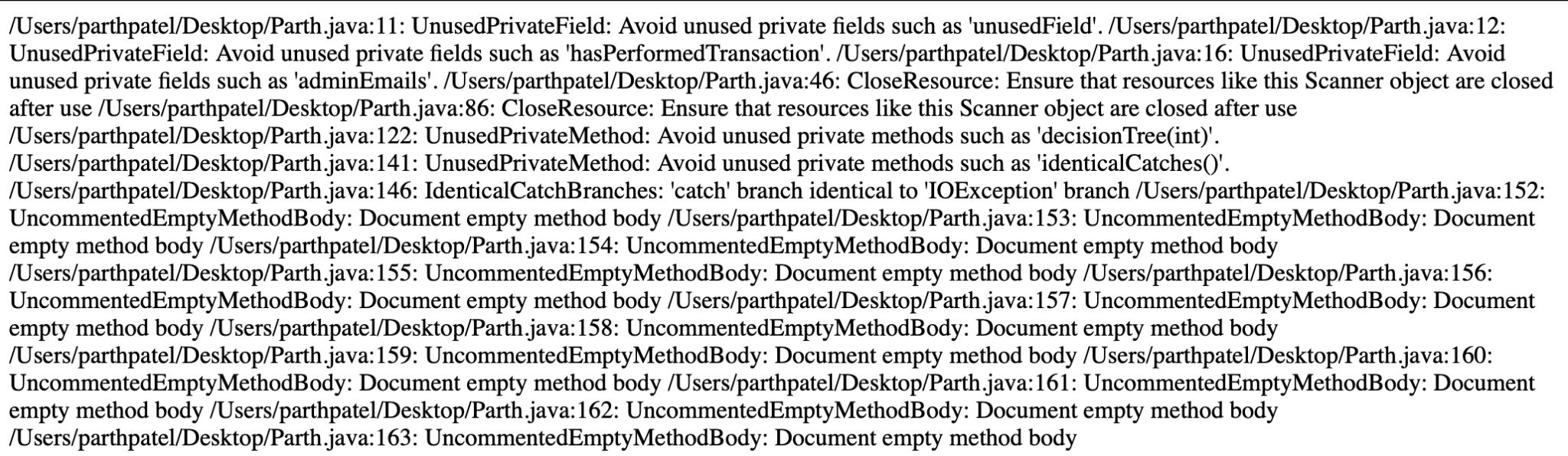
New File : Parth.java (Added in git)

PMD :

****

***REPORT :***

**/Users/parthpatel/Desktop/Parth.java:11: UnusedPrivateField: Avoid unused private fields such as 'unusedField'. /Users/parthpatel/Desktop/Parth.java:12: UnusedPrivateField: Avoid unused private fields such as 'hasPerformedTransaction'. /Users/parthpatel/Desktop/Parth.java:16: UnusedPrivateField: Avoid unused private fields such as 'adminEmails'. /Users/parthpatel/Desktop/Parth.java:46: CloseResource: Ensure that resources like this Scanner object are closed after use /Users/parthpatel/Desktop/Parth.java:86: CloseResource: Ensure that resources like this Scanner object are closed after use /Users/parthpatel/Desktop/Parth.java:122: UnusedPrivateMethod: Avoid unused private methods such as 'decisionTree(int)'. /Users/parthpatel/Desktop/Parth.java:141: UnusedPrivateMethod: Avoid unused private methods such as 'identicalCatches()'. /Users/parthpatel/Desktop/Parth.java:146: IdenticalCatchBranches: 'catch' branch identical to 'IOException' branch /Users/parthpatel/Desktop/Parth.java:152: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:153: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:154: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:155: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:156: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:157: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:158: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:159: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:160: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:161: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:162: UncommentedEmptyMethodBody: Document empty method body /Users/parthpatel/Desktop/Parth.java:163: UncommentedEmptyMethodBody: Document empty method body**

****

**CHAT-GPT VERSION :**

**Conversation link :** [**https://chatgpt.com/share/6747e8a4-2820-8013-8b91-004cfbfa18dc**](https://chatgpt.com/share/6747e8a4-2820-8013-8b91-004cfbfa18dc)

**The static analysis of your provided code reveals several issues and opportunities for improvement across different areas of software quality:**

### **1. Code Smells**

* **Unused Fields:**
  + **unusedField: This field is never referenced in the code.**
  + **hasPerformedTransaction: This boolean is declared but unused.**
  + **Resolution: Remove unused fields to reduce clutter.**
* **God Class:**
  + **The UserAccountSystem class performs multiple responsibilities, such as:**
    - **User management.**
    - **Logging errors.**
    - **Running the application.**
    - **Handling transactions.**
  + **Resolution: Refactor into smaller, single-responsibility classes. For example:**
    - **UserManager for user-related operations.**
    - **TransactionManager for transactions.**
    - **Logger for logging functionality.**
* **Identical Catch Blocks:**
  + **The identicalCatches method handles both IOException and generic Exception with the same logic.**
  + **Resolution: Combine them into a single block or differentiate their handling.**
* **TooManyMethods:**
  + **The UserAccountSystem class has a large number of methods, many of which (e.g., extraMethod1 through extraMethod12) serve no purpose.**
  + **Resolution: Remove or modularize the code to make the class concise.**

### **2. Metrics**

* **NPath Complexity:**
  + **The decisionTree method has a high NPath complexity due to nested if-else blocks.**
  + **Resolution: Refactor to simplify the logic or use a switch-case structure if applicable.**
* **Weighted Methods per Class (WMC):**
  + **UserAccountSystem has an excessive number of methods, making it hard to maintain and test.**
  + **Resolution: Split functionality into different classes or interfaces.**
* **Data Class:**
  + **The User class exhibits behavior of a Data Class with only getters and setters.**
  + **Resolution: Introduce meaningful methods within the User class to encapsulate operations related to users.**

### **3. Maintainability Issues**

* **Violation of DRY (Don't Repeat Yourself):**
  + **The error logging and user input/output operations are repetitive.**
  + **Resolution: Create utility/helper methods or classes for repetitive tasks.**
* **File Handling Without Robustness:**
  + **The loadUsersData method doesn't validate user data from the file.**
  + **Resolution: Add validation for data format and error handling for malformed entries.**

### **4. Security Concerns**

* **Sensitive Data Exposure:**
  + **Storing plaintext passwords (password in the User class) poses a security risk.**
  + **Resolution: Use hashing (e.g., bcrypt) for password storage.**

### **5. Usability Concerns**

* **Scanner Resource Management:**
  + **The Scanner object in runApp and performTransaction is not closed.**
  + **Resolution: Use a try-with-resources statement or close the Scanner explicitly.**
* **Hard-Coded File Path:**
  + **The FILE\_PATH constant is hard-coded.**
  + **Resolution: Use a configuration file or environment variables to manage paths.**

### **6. Suggestions for Refactoring**

* **Extract Logging:**
  + **Move the logs functionality to a separate Logger class.**
* **Extract User Management:**
  + **Create a UserManager class to handle operations like loading user data and managing login attempts.**
* **Simplify Transactions:**
  + **Separate deposit and withdrawal operations into a TransactionManager class.**

### **7. Potential Bugs**

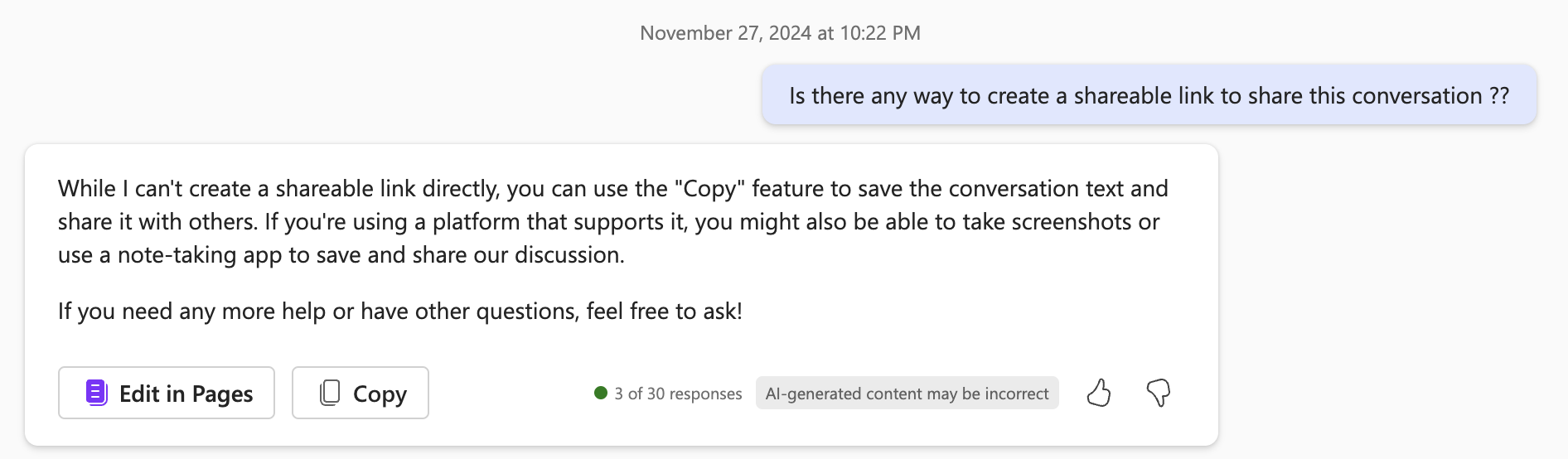
* **Concurrent Modification Issues:**
  + **The users map is accessed without synchronization, which could lead to issues in a multithreaded context.**
  + **Resolution: Use a thread-safe collection like ConcurrentHashMap or synchronize access.**
* **Infinite Loop in runApp:**
  + **The while (true) loop in runApp doesn't provide a way to exit gracefully.**
  + **Resolution: Include an exit option for the user.**

### **Key Recommendations**

* **Code Refactoring:**
  + **Apply the Single Responsibility Principle (SRP) to break down the UserAccountSystem class.**
  + **Address unused fields and methods.**
* **Security Enhancements:**
  + **Use encrypted passwords.**
  + **Validate inputs to prevent injection attacks.**
* **Improved Maintainability:**
  + **Modularize code for better readability and reusability.**
  + **Document the purpose of methods and classes.**

(C)

**CO-PILOT VERSION :**

**Conversation link : **

**Sure, I can help with that! Here are some issues identified in your code through static analysis:**

### **1. Unused Fields**

* **unusedField and hasPerformedTransaction in UserAccountSystem are declared but never used.**

### **2. God Class**

* **UserAccountSystem has too many responsibilities, including user management, logging, and transaction handling. This violates the Single Responsibility Principle.**

### **3. High NPath Complexity**

* **The decisionTree method has a high NPath complexity due to nested if-else statements. This makes the code harder to maintain and test.**

### **4. Identical Catch Blocks**

* **The identicalCatches method has identical catch blocks for IOException and Exception. This redundancy can be reduced by catching Exception only once.**

### **5. Too Many Methods**

* **UserAccountSystem has many empty methods (extraMethod1 to extraMethod12). These should be removed if not needed.**

### **6. Suspected Data Class**

* **The User class is a data class with only getters and setters. Consider adding behavior to this class to make it more cohesive.**

### **7. Potential Security Issue**

* **Storing passwords as plain text in the User class is a security risk. Consider using hashed passwords instead.**

### **8. Resource Management**

* **The Scanner object in runApp and performTransaction methods is not closed, which can lead to resource leaks.**

### **9. Logging**

* **The logError method adds error messages to a list but does not provide a way to persist or view these logs. Consider integrating a proper logging framework.**

### **10. Magic Numbers**

* **The number 3 in the login method for failed login attempts should be replaced with a named constant for better readability.**

### **Recommendations:**

* **Refactor UserAccountSystem to separate concerns (e.g., user management, logging, transaction handling).**
* **Simplify the decisionTree method to reduce complexity.**
* **Consolidate the catch blocks in identicalCatches.**
* **Remove unused fields and methods.**
* **Enhance the User class with relevant behavior.**
* **Implement password hashing for security.**
* **Ensure proper resource management by closing the Scanner object.**