

Demo: Performing Code review and refactoring with Github Copilot

You can perform code review using Github Copilot. It helps you to verify the code is following best practices, following naming conventions, doing vulnerability checks and proper error handling.

Prerequisites

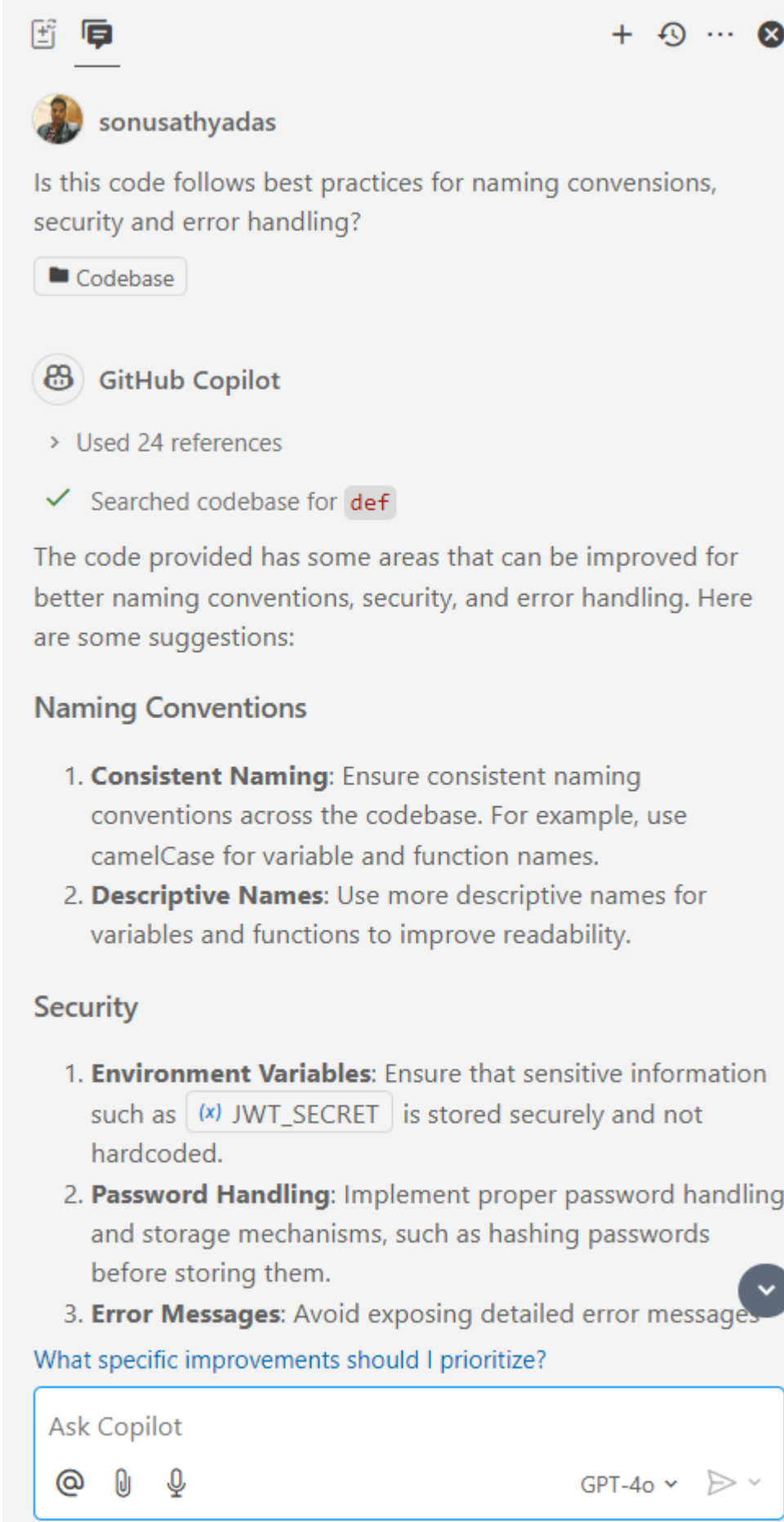
- VS Code
- GitHub Copilot extension for VS Code
- GitHub Copilot subscription
- NodeJS 18 or later
- Book API project

Code review and refactoring

1. Open the `book-api` project in VS Code.
2. Open the Copilot Chat window and refer the `codebase` by clicking the `attachment icon`.
3. Run the following prompt in the chat.

```
Is this code follows best practices for naming convensions, security and error handling?
```

4. This will analyze the entire project and displays the results of analysis based on naming convensions, security best practices and error handling.



The screenshot shows a chat interface with a user profile for 'sonusathyadas' and a question about code best practices. A 'Codebase' button is visible. The GitHub Copilot response includes a summary of references, a search status for 'def', and detailed suggestions under 'Naming Conventions' and 'Security'. A follow-up question is asked at the bottom, and the interface includes a text input field, icons for attachments, and a dropdown for the model 'GPT-4o'.

sonusathyadas

Is this code follows best practices for naming convensions, security and error handling?

Codebase

GitHub Copilot

> Used 24 references

✓ Searched codebase for `def`

The code provided has some areas that can be improved for better naming conventions, security, and error handling. Here are some suggestions:

Naming Conventions

1. **Consistent Naming:** Ensure consistent naming conventions across the codebase. For example, use camelCase for variable and function names.
2. **Descriptive Names:** Use more descriptive names for variables and functions to improve readability.

Security

1. **Environment Variables:** Ensure that sensitive information such as `(x) JWT_SECRET` is stored securely and not hardcoded.
2. **Password Handling:** Implement proper password handling and storage mechanisms, such as hashing passwords before storing them.
3. **Error Messages:** Avoid exposing detailed error messages.

What specific improvements should I prioritize?

Ask Copilot

@ 📎 🎤 GPT-4o ▾ ➤ ▾

5. This will also show the suggestions for improvements on this area. If any improvements are required, Copilot will show the suggested code in the Chat window.

1. **Consistent Error Handling:** Ensure consistent error handling across the codebase. Use try-catch blocks where necessary and handle errors gracefully.
2. **Logging:** Implement proper logging for errors to help with debugging and monitoring.

Example Improvements

JS authController.js



```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt'); // Use bcrypt

class AuthController {
  static async login(req, res) {
    const { username, password } = req.body;

    // This is a mock user validation.
    if (username === 'admin' && await bcrypt.compare(password, 'admin')) {
      const token = jwt.sign({ username }, process.env.JWT_SECRET, { expiresIn: '1h' });
      res.json({ token });
    } else {
      res.status(401).json({ message: 'Invalid credentials' });
    }
  }
}

module.exports = AuthController;
```

What specific improvements should I prioritize?

Ask Copilot

 Demo_19-Code_Review_and_refactoring.md Current file 



GPT-4o ▾



6. Click on the **Apply in Editor** icon to refactor the existing code and update the code files.