AI-Powered Code Review Assistant - Challenger Project

Overview

Build a standalone AI assistant that provides intelligent, contextual code review feedback. Engineers often need quick feedback on specific sections of code, but traditional code review can be slow and generic AI chat interfaces lack precision. Your challenge is to create a focused, block-level code review experience.

The Challenge

Create a web application where users can:

1. Paste or write code in an editor
2. Select specific lines or blocks of code
3. Request AI feedback on that exact selection
4. Receive contextual suggestions, improvements, or explanations
5. Iterate on the conversation within the context of that code block

Think of it as "inline comments meet AI assistant" - the AI should understand the surrounding code context, not just the selected portion.

Core Requirements

Your solution must include:

1. **Code editor interface**: Users can input and view code with syntax highlighting.
2. **Selection-based interaction**: Users can highlight specific code sections to prompt the AI.
3. **Contextual AI responses**: The AI receives enough context (surrounding code, language, file type) to give meaningful feedback.
4. **Inline conversation threads**: Comments and AI responses should be visually tied to specific code sections.
5. **Multiple conversation threads**: Support multiple independent discussions about different parts of the same code file.

Technical Constraints

- **Standalone application**: No external services or credentials required from us.
- **Language agnostic**: Should work with multiple programming languages.
- **Use any AI API**: OpenAI, Anthropic, local models, or mock responses for demo purposes.
- **Your choice of stack**: React, Vue, vanilla JavaScript, Python Flask, whatever you're comfortable with.

What We're Evaluating

Technical execution

- Code quality and organization.
- How you handle state management for multiple comment threads.
- UI/UX decisions for a developer-focused tool.
- API integration approach.

Product thinking

- How do you make the interaction feel natural?
- What context do you provide to the AI for better responses?
- How do you handle edge cases (very long files, multiple languages, nested selections)?

Scaling considerations

- How would this work with real codebases?
- What would need to change for team collaboration?
- Security and privacy considerations.

Expected Deliverables

1. Working Application

A functional prototype that demonstrates the core workflow. Shortcuts are fine, but it should actually work.

2. Brief Documentation (README)

- How to run it.
- Key architectural decisions.
- What you'd do differently with more time.
- How you used AI tools (if applicable).
- Trade-offs you made.

3. Code

- Clear structure and readable code.
- Comments where decisions aren't obvious.
- Git commits showing your progression.

Suggested Approach

Phase 1 (1-1.5 hours): Core UI

- Basic code editor (Monaco, CodeMirror, or simple textarea).
- Text selection functionality.
- Comment thread UI structure.

Phase 2 (1.5-2 hours): AI Integration

- Choose and integrate an AI API.
- Build prompt that includes code context.
- Display responses inline.

Phase 3 (1-1.5 hours): Multiple Threads

- Support multiple comment threads.
- Visual indicators for which code has comments.
- Thread management.

Phase 4 (0.5-1 hour): Polish & Documentation

- Handle edge cases.
- Write your README.
- Test the key workflows.

Bonus Ideas (If You Have Time)

- **Suggested actions**: AI can propose code changes, not just feedback.
- **Diff view**: Show before/after when AI suggests changes.
- **Language detection**: Automatically detect the programming language.
- **Conversation history**: See past threads even after editing code.
- **Export feedback**: Generate a summary report of all AI suggestions.

Using AI Tools

We encourage you to use AI tools (ChatGPT, Claude, Copilot, etc.) to help build this. Please document:

- What you used AI for.
- How you verified or adapted its suggestions.
- What worked well and what didn't.

Example User Flow

```
1. User pastes a Python function into the editor

2. User highlights lines 5-8 (a complex list comprehension)

3. User clicks "Ask AI" or similar action

4. User types: "Is there a clearer way to write this?"

5. AI responds with context about the full function

6. AI suggests an alternative approach with explanation

7. User can follow up with more questions

8. User highlights a different section and starts a new thread
```

Technical Notes

- **For AI API**: If you don't want to use a paid API, you can mock responses or use a free tier. Just make the integration clean so swapping providers would be easy.
- **For the editor**: Don't overthink this. Even a well-styled textarea with line numbers works for a prototype.
- **For state management**: Multiple threads tied to code ranges is the interesting challenge here.

Evaluation Criteria

Must have:

- Working code selection and comment interface.
- AI integration that provides contextual responses.
- Multiple independent comment threads.
- Clean, understandable code.

Nice to have:

- Polished UI/UX.

- Smart context management for AI prompts.
- Thoughtful edge case handling.
- Clear documentation of trade-offs.

Context: Why This Matters

Block comments are coming to WordPress Core 6.9 this year. This represents a significant shift in how editorial teams collaborate on content. By integrating AI directly into this workflow, we can create a more natural collaborative experience than traditional separate chat interfaces. Your prototype will help us understand what's possible and what editorial teams actually need.

Remember: We're evaluating your technical skills, product thinking, and how you approach building something new in an existing ecosystem. Show us how you work through problems, make tradeoffs, and build with users in mind.