AI-Based Assignment with Tech Stack

AI-Native Developer Assignment: Guidelines

This assignment evaluates your development skills, understanding of core software design principles, and your ability to leverage AI-powered code assistants effectively within the context of an AI-native SaaS platform for ITSM and ITOM.

1. Assignment Goal & Context

The primary goal is to assess your proficiency in:

- Building a functional application demonstrating integration.
- Implementing basic AI simulation logic on the backend.
- Applying fundamental software design patterns.
- Crucially: Effectively using an AI code assistant (e.g., GitHub Copilot, Cursor AI, Windsurf AI) to accelerate development, improve code quality, and demonstrate prompt engineering skills. We also want to see how you might integrate AI assistants into broader project workflows using "rules" or "templates" if your chosen assistant supports it.

The assignments are framed around core features of an AI-native ITSM and ITOM platform, designed to provide intelligent assistance to IT operations teams.

2. General Guidelines for Candidates

- Time Commitment: Each assignment is designed to take approximately 2-3 hours of focused work. Please complete it at your own pace.
- Technology Stack: While our desired stack is Golang/Java for the core backend, Python for the AI/ML ecosystem, and React/Next.js for the frontend, you do not need to be proficient in all three. You should demonstrate expertise in at least two core components relevant to the assignment. Feel free to choose the specific technologies you're most comfortable with within these broader categories. Justify your chosen tech stack for the modules you implement in your README.md.
- AI Code Assistant Usage is Mandatory: You must use an AI code assistant throughout the development process.
    - Logging: You are required to maintain a detailed log of your AI code assistant interactions. This includes:
        - Prompts: The exact prompts you used.

        - AI Responses/Suggestions: The relevant code or text snippets suggested by the AI.
        - Your Action: What you did with the AI's suggestion (accepted, modified, rejected, asked for refinement, etc.) and why.
        - Context: A brief description of the task you were performing when you used the AI.
    - Prompt Engineering & Rules/Templates: In your README.md, you should also discuss:
        - Your prompt engineering strategy (e.g., how you made prompts specific, provided context, iterated).
        - If your AI assistant (like Cursor AI) supports "rules" or "templates" for project-wide standards, show how you would use or *could* use them for this assignment (even if for conceptual benefit). Provide examples.
- Completeness over Perfection: Focus on delivering a working solution that meets the core requirements of your chosen modules. Clean, readable code and thoughtful design are more important than an exhaustive feature set.
- Open-Book: Feel free to use any online resources, documentation, or tutorials.

3. Deliverables

Please submit the following:

1. Working Codebase: A complete, runnable application demonstrating your chosen modules. Submit it as a zipped folder or a link to a Git repository.
2. Comprehensive README.md File: This is a critical component and must include:
    - Setup Instructions: Clear, step-by-step instructions on how to set up and run your implemented components. Include any necessary dependencies or environment variables.
    - Core Functionality Overview: A brief explanation of what the application does and how to use it, including the user journey and logic you defined for your chosen modules.
    - Software Design Choices & Justification: A detailed explanation of key architectural decisions made for the modules you implemented. This should cover:
        - Chosen technologies and why.

        - Database schema design (if applicable) and relationships.
        - API design (endpoints, request/response formats) for implemented components.
        - Frontend component structure and state management approach (if implemented).
        - Module Selection Rationale: Clearly state which modules you chose to implement and why, as well as how you envision the *unimplemented* modules integrating into the overall system.
    - AI Code Assistant Usage Log (Detailed): As described in Section 2, provide a clear, structured log of your AI interactions, including prompts, AI responses, your actions, and context. Aim for at least 5-7 significant interactions.
    - Prompt Engineering & AI Assistant Strategy Discussion:
        - Describe your approach to crafting effective prompts. What techniques did you use (e.g., being specific, providing examples, iterating)?
        - If your AI assistant supports "rules" (like Cursor AI's Rules) or "templates" for enforcing coding standards or generating boilerplate for larger projects, discuss how you might use or *could have used* them for this assignment. Provide a concrete example of a rule/template you would define for a larger project.
    - Assumptions Made: Any assumptions or simplifications made during development.
    - Potential Improvements & Future Enhancements: Ideas for how the application could be extended or improved with more time, especially regarding integrating the unimplemented modules.

AI Assignments for 3-5 Years Experience

Choose ONE of the following assignments. For your chosen assignment, you MUST implement the Mandatory Core Modules and at least ONE of the Flexible Modules. You are responsible for defining the full user journey, detailed logic, and specific data models to achieve the expected output.

Mandatory Core Modules for ALL Candidates:

1. AI Code Assistant Usage & Prompt Engineering: As per Section 2.

2. Core AI Logic (Simulated): Implement the logic that takes raw operational event data and generates AI-driven insights (e.g., "Category", "Suggested Action").
    - Choose ONE language for this: Python (preferred for AI ecosystem for its libraries) OR Golang (for its performance and concurrency) OR Java.
    - *Expected Output (from this module):* A function or module that takes an event object as input and returns the event object augmented with derived fields like ai_category and ai_suggested_action.
3. Data Persistence: Store the ingested events and their AI-generated insights.
    - Choose ONE language for this: Coupled with your AI Logic choice.
    - *Expected Output:* A defined database schema (e.g., for SQLite, PostgreSQL, or MongoDB) and basic CRUD operations (at least create and read) for your event data.

Flexible Modules (Choose to implement at least ONE of the following):

Option A: Backend Event Ingestion API (Focus on Go Backend / Python Backend / Java Backend)

- Description: Build a RESTful API endpoint to ingest raw operational events (e.g., POST /events). This API will call your "Core AI Logic" and then persist the event with its generated insights.
- Expected Output: A working API endpoint that accepts JSON event data, processes it through your AI logic, stores it, and returns a success response. Basic error handling for API requests.
- Tech Stack: Implement this API in Golang (e.g., net/http, Gin, or Echo) OR Python (e.g., Flask, FastAPI) OR Java (e.g. REST API).

Option B: Frontend Insight Dashboard (Focus on React/Next.js Frontend)

- Description: Build a simple web UI to display the operational events and their AI-generated insights from your backend/database. Allow for basic viewing and perhaps one simple filter.
- Expected Output: A web page that lists the ingested events with their ai_category and ai_suggested_action. A detail view for a single event. The UI should be responsive and user-friendly.
- Tech Stack: React or Next.js. You'll need to define conceptual API endpoints your frontend *would* consume from the backend (even if you only mock these if not implementing the backend API).

Assignment 1: AI-Powered Incident Triage Assistant

Scenario: Your platform aims to help IT support teams quickly prioritize and categorize incoming service incidents. You'll build the "Event-to-Insight" system where the "events" are incident reports. The system should ingest an incident, apply AI logic to suggest its severity and category, and then either provide an API for its retrieval or display it in a web dashboard.

Expected Output (depending on chosen flexible module):

- If choosing Backend Event Ingestion API: A working API endpoint (POST /incidents) that accepts incident data (title, description, affected_service), processes it through your AI logic to assign ai_severity and ai_category, stores it, and returns the created incident.
- If choosing Frontend Insight Dashboard: A web interface allowing input of incident data. Upon submission, the UI displays the AI-suggested severity and category alongside the incident details. A list view displays all incidents with their AI-derived insights.

Assignment 2: AI-Driven Alert Correlation & Suggestion

Scenario: Your platform processes raw alerts from various monitoring systems. You'll build the "Event-to-Insight" system where the "events" are incoming alerts. The system should ingest an alert, use AI logic to suggest a potential root cause and action, and potentially correlate it with other recent alerts from the same source.

Expected Output (depending on chosen flexible module):

- If choosing Backend Event Ingestion API: A working API endpoint (POST /alerts) that accepts alert data (source_system, metric_type, value_exceeded), applies AI logic to determine ai_suggested_root_cause, ai_suggested_action, and a ai_correlation_id (if applicable), stores it, and returns the processed alert.
- If choosing Frontend Insight Dashboard: A web interface allowing input of alert data. Upon submission, the UI displays the AI-suggested root_cause, action, and correlation_id for the alert. A list view shows alerts, potentially grouped by correlation_id.

Assignment 3: AI-Assisted Knowledge Base Search

Scenario: Your platform needs a self-service component where users can quickly find answers to IT questions. You'll build the "Event-to-Insight" system where the "events" are user search queries. The system should accept a user's problem query, use AI logic

to find relevant knowledge base articles (simulated as hardcoded data), and provide a summarized answer.

Expected Output (depending on chosen flexible module):

- If choosing Backend Event Ingestion API: A working API endpoint (POST /search-query) that accepts a user's query, processes it through AI logic to find relevant simulated knowledge base articles and generate an ai_summary_answer, and returns these results.
- If choosing Frontend Insight Dashboard: A web interface with a search bar. When a user types a query, the UI displays an AI-generated summary_answer and a list of relevant_articles (titles/IDs), allowing users to click to view full simulated article content.

---

4. Evaluation Criteria

Each assignment will be evaluated based on the following:

1. Functionality & Robustness (30%):
    - Does the candidate's chosen set of implemented modules work as expected?
    - Is the implemented solution stable and robust?
    - For Go/Java (if chosen): Demonstrated understanding of concurrency patterns and error handling.
    - For Python (if chosen): Clean and effective implementation of the AI logic.
    - For React/Next.js (if chosen): Responsiveness, interactivity, and effective state management.
2. Software Design & Architecture (25%):
    - Overall System Design: How well do the implemented modules fit into a cohesive system? Does the candidate articulate how *unimplemented* modules would integrate and exchange data?
    - Module-Specific Design:
        - Go (if chosen): Idiomatic Go design, effective use of Go features.
        - Java (if chosen): Idiomatic Java design, effective use of Java features.

        - Python (if chosen): Modularity of AI logic, API design (if implemented).
        - React/Next.js (if chosen): Component architecture, state management, and API interaction strategy.
    - Database Schema: Is the data model appropriate for the problem (even if simple)?
3. Code Quality & Best Practices (15%):
    - High readability, maintainability, and consistency within chosen language(s).
    - Adherence to language-specific best practices (e.g., PEP8 for Python, Go idioms, React Hooks rules, Java Google Style Guide).
    - Comprehensive error handling and logging within implemented modules.
4. AI Code Assistant Utilization & Prompt Engineering (25%):
    - Usage Log Quality: Is the AI code assistant usage log detailed, clear, and comprehensive (prompts, AI responses, actions, context)?
    - Effectiveness: Does the log demonstrate how the AI assistant genuinely aided productivity (e.g., boilerplate, function stubs, refactoring, debugging) in the implemented modules?
    - Prompt Engineering Skill: How well did the candidate craft their prompts? Did they iterate on prompts? Did they use specific instructions, context, or examples?
    - Rules/Templates Discussion: Does the candidate demonstrate an understanding of how AI assistant "rules" or "templates" could be used for larger projects, and provide a concrete example relevant to their chosen technology stack?
5. Documentation & Communication (5%):
    - Clarity and completeness of the README.md (setup instructions, design choices for implemented modules, AI usage log, assumptions, future improvements).
    - Effectiveness in explaining how unimplemented parts would integrate into the overall system.