Exercise 2 – AI-assisted coding (30 points)

Introducion lecture: 6th Feb. Deadline: 2nd Mars.

Task 2.1 – Prompting strategy for code generation (5 points)

Select one self-contained group of requirements (i.e. requirements that can be implemented independently from other groups) consisting of at least six (6) functional requirements and two (2) non-functional requirements. This group should represent a meaningful feature set of the system (for example, AI-usage logging or user authentication) and be suitable for isolated design and implementation.

Example scope of the requirements:

- Functional requirement (user story):
  As a student, I want to log how I used an AI tool for an assignment, so that my AI usage is documented and transparent.
- Non-functional requirement (user story – usability):
  As a student, I want the AI-usage logging interface to be simple and quick to use, so that logging does not interrupt my study workflow.

This task focuses on designing a structured prompting strategy for building components of the *AIGuidebook* website using AI-assisted code development tools (e.g. ChatGPT, GitHub Copilot, or similar). NTNU provides GitHub Copilot licences for students, and the InnSpill platform offers free access; additional tools may require personal licences.

Using the requirements and dependency map from Task 1.3, you will design a systematic prompting approach that supports requirement-driven code generation, component-level architectural planning, iterative refinement and debugging, and effective human–AI co-development. The goal is not to write code, but to define a repeatable prompting strategy that enables developers to reliably generate and validate React components aligned with real system requirements.

Your strategy must demonstrate understanding of a full-stack web architecture, for example React for frontend, and Node.js + Express for backend.

Deliveriables:

1. Prompt Strategy Document (maximum 500 words) describing a prompting approaches and patterns used.
2. Example Prompt including:

a. Role and system context

b. Project context

c. Technical requirements

d. Output format and constraints

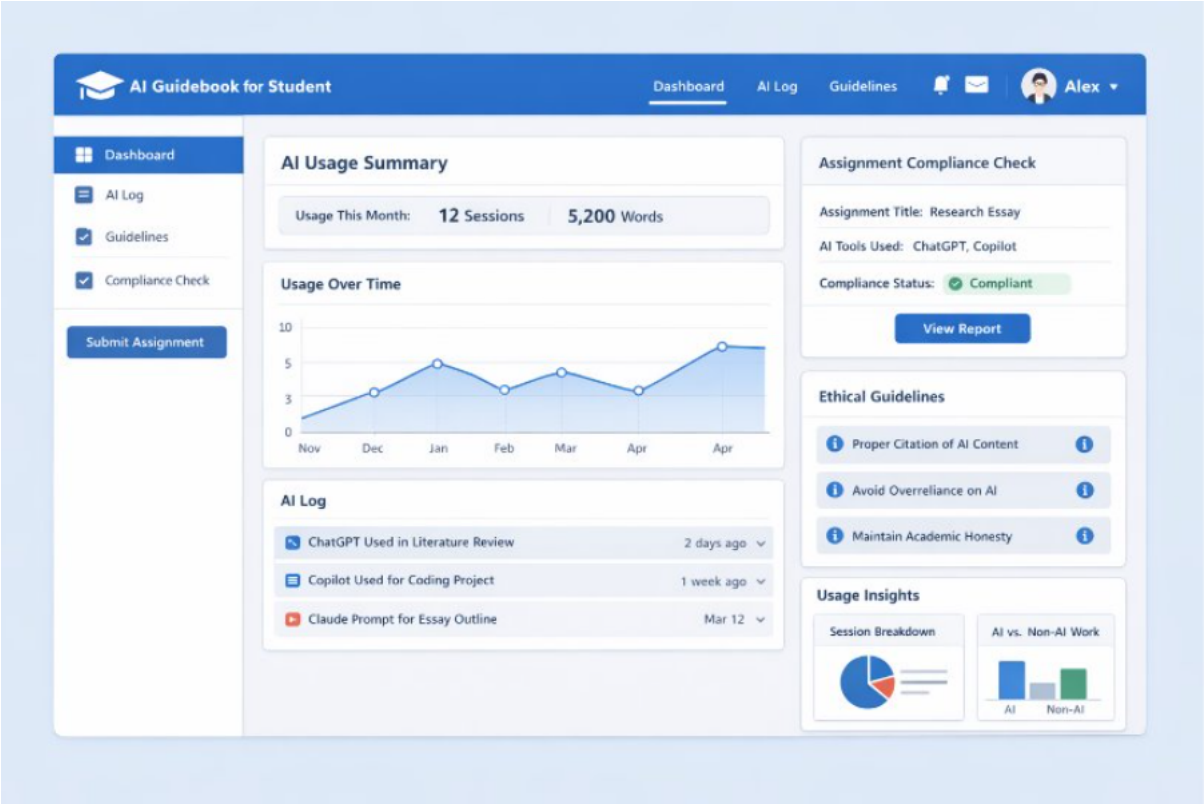
e. Expected structured response (folder structure, source code, explanation)

Evaluation criteria:

- Quality of Prompt Strategy: The strategy is coherent, repeatable, and covers known prompting patterns (at least 2)
- Technical Understanding of full-stack web development: Clear awareness of a full-stack web project structure, component organisation, and architectural concerns.

Task 2.2 AI-Based Code Generation (15 points)

Use ChatGPT, Claude, GitHub Copilot, or similar AI tools to generate a full-stack web application that implements the requirements selected from Exercise 2.1. Following your defined prompting strategy and requirement set, you will guide an AI code assistant to generate the initial version of the system. Look at a mock-up below as an inspiration:



Your objective is to deliver:

- Correct installation and configuration of all required libraries and dependencies

- Revision of AI-generated code if needed
- A working full stack web application that runs locally in your computer
- A clear and complete explanation of the codebase structure and design decisions

Deliverables:

- A document including:
- Link to a public GitHub repository containing the full project
- Screenshots demonstrating successful execution
- Explanation of the role of each major component
- What code was generated by AI and what was written or modified by you

Evaluation Criteria:

- Successful code execution: The application runs correctly in your local environment and meets the agreed functional scope
- Quality of document: Clear explanation of architecture, component responsibilities, and responsible use of AI in development

Task 2.3 – Code Review (10 points)

In this task, you will analyse, review, and improve the quality of the codebase produced in Task 2.2. You will apply code review practices to identify and reason about code smells—structural weaknesses that can reduce maintainability, readability, reliability, or performance. You must use at least two (2) of the following review approaches:

- Manual code review
- Peer review with classmates
- AI-assisted code analysis tools (e.g. ChatGPT, GitHub Copilot, SonarLint)

Examples of code smells include (but are not limited to): long or overly complex components, duplicated logic, primitive obsession, large classes, long parameter lists, temporary fields, shotgun surgery, dead or unreachable code, excessive comments, deeply nested JSX, mixing UI and business logic, and missing error handling.

If peer review is used, participating groups must agree on the collaboration in advance, and the codebase must be made available for review one week before the submission deadline.

If AI tools are used, you must:

- Include sample prompts and responses
- Clearly state which suggestions were accepted or rejected

- Explain the reasoning behind these decisions

Deliverables:

- Code Review Summary Document that:
- Describes the review method used in sufficient detail to be repeatable
- Lists identified code smells, including:
- Name of the smell
- Short description
- Identify method
- Relevant original code snippets

Evaluation Criteria:

- Coverage: demonstrates broad and systematic coverage of the codebase, including components, pages, utilities, and data flow.
- Quality of code smell identification: Focus on meaningful structural issues rather than superficial style problems
- Documented review methodology: The review process is clearly described and reproducible