Task 2.1

Selected self-contained requirement group

Functional requirements

1. R1 Log AI use.
2. R2 View NTNU guidelines for AI use.
3. R4 Show compliance status for each logged prompt.
4. R6 Explain why a prompt is not compliant.
5. R7 Add reflection or description to a logged prompt.
6. R8 Auto generate an AI declaration form from the logged usage.

Non-functional requirements

1. R26 Security so user data is not leaked publicly.
2. R28 Reflection prompts must be visually distinct in the user interface.

Dependency order

- R1 and R2 enable R4.
- R4 enables R6.
- R7 depends on R1.
- R8 depends on R1, R6, and R7.

Prompt strategy document

The prompt is treated as a collaborative artifact. At the start of each session the requirement slice, the dependency order, and the rules for the assistant are restated. Keeping these rules stable helps the assistant produce consistent outputs that are easy to review and easy to validate.

Each interaction begins with a role based prompt that sets expectations for secure defaults, clear separation of concerns, and step by step collaboration. Strict output constraints are included as well. The assistant must explain briefly before writing code, produce only one file at a time, and always ask before moving on. This reduces the risk of the assistant quietly introducing extra features.

Before writing code, the assistant is asked to clarify the requirements. It should point out unclear wording and propose concrete assumptions that can be checked later. For this requirement group, the biggest ambiguity is usually compliance. To keep it explainable, compliance is implemented as a rules based checker for the first version. It should return a clear status and a list of reasons, not just a label. The same clarification is done for R1, so the logging format includes what later requirements need, such as timestamp, tool name, the prompt, an optional reflection field, and the computed compliance result.

Security is handled early so it does not become an afterthought. R26 is made concrete through input validation on every API route, safe CORS settings for local development, and avoiding sensitive content in server logs. The backend stays bound to localhost by default. If persistence is used, a local file database or a JSON file with careful validation is preferred, since the goal is a local prototype that still behaves responsibly.

Next, the assistant proposes architecture before implementation. This includes a minimal client and server structure, a small data model, and a clear API contract describing routes, request and response JSON, validation rules, and error cases. A short verification checklist is included that maps each requirement to a manual test. This creates a stable backbone that frontend components can rely on and it reduces later rewrites.

Implementation follows the dependency order from Task 1.3. First logging and persistence, then guideline viewing, then compliance status and explanations, then reflection with a visually distinct UI for R28, and finally declaration generation for R8. After each increment, the assistant must state what changed, how to test it, and which requirement it satisfies. If something is missing, it is handled in the next smallest patch instead of expanding scope.

When debugging, the error and the smallest relevant code are shared. The assistant responds with root cause, a minimal patch, and re-test steps that prove the fix. This keeps the iteration tight and focused on requirements.

Prompt to test

Role

You are a senior full-stack engineer and reviewer. Build a local prototype with React on the frontend and Node.js with Express on the backend. Do not call external AI services. The compliance check must be rules-based and explainable. Treat this prompt as a collaborative artifact and keep the rules stable across iterations.

Project context

The system helps students log AI use, read NTNU guidelines, see compliance status with explanations, add reflection, and generate an AI declaration for an assignment.

Requirements

Functional requirements R1 Log AI use. R2 View NTNU guidelines for AI use. R4 Show compliance status for each logged prompt. R6 Explain why a prompt is not compliant. R7 Add reflection or description to a logged prompt. R8 Auto generate an AI declaration form from the logged usage.

Non-functional requirements R26 Security so user data is not leaked publicly. R28 Reflection prompts must be visually distinct in the user interface.

Dependency order Implement logging and guidelines first. Then implement compliance status and explanations. Then implement reflection. Then implement declaration generation.

Technical constraints Use a simple persistence solution so logs survive restart. Validate inputs and return clear error responses. Avoid sensitive data in server logs. Bind the backend to localhost. Use safe CORS settings for local development.

Working method rules Explain your reasoning briefly before writing code. Produce only one file at a time. State the filename on the first line of each code response. If information is missing, ask for clarification before coding. Do not skip steps in the development process. Do not add features that are not required.

Output format for the first response 1. Requirement checklist and how each will be verified. 2. Ambiguities and proposed assumptions. 3. Proposed folder structure for client and server. 4. API contract table with routes, request and response JSON, validation rules, and error cases. 5. A short implementation plan in the dependency order. 6. Ask us which file to generate first.

Output format for code responses Generate exactly one file per response. Start with the filename on the first line. Then provide the full file content. Then explain how it connects to the requirements. Then ask which file to generate next.

Debugging rule If we paste an error, respond with root cause, a minimal patch, and re-test steps.