Prompt Structure:

1. Personna
2. Goal
3. Context
4. Format
5. Examples
6. Guidelines
7. Restrictions

Disclaimers:

1. Prompting is not a one -shot thing, it involves practice and making a lot of mistakes
2. This does not replace developers or product owners
3. Experience level is still very important
4. A.I do not help you with EaC (unless you find a way to fit all the EaC documentation into a context window)
5. You might not get what you want on the first try. Think iteratively
6. A.I hallucinates!
7. You are limited by your imagination

Techniques and Frameworks

1. Add “Be concise but include important information” to reduce verbose out – short answers
2. Another variation of the techniques above is “respond in 2 sentences or 100 words”
3. Respond with “I don’t know” If the information is not complete or uncertain – Reduce hallucination
4. “Ask me questions” framework
5. Target persona
6. Examples framework
7. Feedback framework
8. Extract writing style and tone
9. Best practices / priming the chat
10. Reverse engineering framework
11. Give me 3 prompt variations to do [task] that I can give to chatgpt

Advanced Techniques

1. Chain of Thought
2. Tree of thought
3. Chain of Draft

Best Practices

1. Start a new chat when chat gets too long
2. When working on detailed problems, start new chat when chat gets too long to avoid hallucination
3. Break down problems into multiple prompts
4. Say thank you for encouraging positive reinforcement 😊
5. Rename your chats and delete old chats
6. Keep a document of successful prompts

For UI guys you can add MDS documentation to the context box and have AI generate based on it   
  
  
  
**Structured Prompt Templates for Engineers & Product Owners**

**🔄 General Prompts for All Engineers**

- -Always prefer simple solutions and efficient

- Ensure the new version follows best practices.

- Explain why the changes improve performance or maintainability.

- Recommend ways to improve code readability and maintainability.

- Identify redundant code and suggest refactoring.

- Analyze the code snippet and identify potential problems.

- You are careful to only make changes that are requested or you are confident are well understood and related to the change being requested

- When fixing an issue or bug, do not introduce a new pattern or technology without first exhausting all options for the existing implementation. And if you finally do this, make sure to remove the old implementation afterwards, so we don't have duplicate logic.

- Avoid writing scripts in files if possible, especially if the script is likely only to be run once

- Identify inefficient code patterns.

- Suggest ways to improve execution time and memory usage.

- Provide trade-offs for different optimization approaches.

- Provide a solution with a clear explanation.

- Suggest best practices to prevent similar issues in the future.

- Identify potential security risks, including authentication flaws and injection attacks.

- Suggest solutions and best practices for securing the application.

- Ensure compliance with OWASP security guidelines.

- Ensure log levels (INFO, DEBUG, ERROR) follow best practices.

- Generate parameterized tests for different edge cases

- Ensure RESTful principles are followed.

- Provide recommendations on endpoint structure, versioning, and pagination.

- Suggest best practices for error handling and rate limiting.

**🖥️ Frontend Engineers (React)**

**System Prompts:**

- Generate clean, modular React components following best practices.

- Use functional components with hooks where applicable.

- Ensure reusability by splitting UI logic from business logic.

- Follow our team's coding standards for styling (CSS Modules).

- Recommend the best approach (useState, useReducer, or context) based on the use case.

- Optimize re-renders and avoid unnecessary state updates.

- Suggest best practices for using Redux or MobX.

- Use fetch or Axios following best error-handling practices.

- Implement retry logic for network failures.

- Handle loading states properly and provide meaningful UI feedback.

- Use fetch or Axios following best error-handling practices.

- Implement retry logic for network failures.

- Handle loading states properly and provide meaningful UI feedback.

- Ensure ARIA attributes and semantic HTML are used correctly.

- Provide best practices for keyboard navigation and screen reader support.

- Recommend tools for automated accessibility testing (e.g., Lighthouse, axe-core).

- Use pagination and lazy loading where applicable.

- Use Jest and React Testing Library for testing components.

- Ensure unit tests cover edge cases and state changes.

- Avoid implementation details in tests to ensure maintainability.

**⚙️ Backend Engineers (Spring Boot)**

- Generate clear, efficient, maintainable Java code following best practices.

- Include relevant annotations (@RestController, @RequestMapping, etc.).

- Implement proper error handling using @ControllerAdvice.

- Follow our [internal coding standards] and avoid deprecated methods.

- Optimize database queries and indexing strategies.

- Ensure transactions are correctly managed to avoid data inconsistencies.

- Apply security best practices, in the [file]

- Ensure sensitive data is encrypted and not logged.

- Use SLF4J with Logback for structured logging.

- Avoid logging sensitive data such as passwords or API keys.

- Implement distributed tracing for microservices.

- Use Junit 5 and Mockito for unit testing.

- Mock external dependencies properly.

- Generate parameterized tests to cover edge cases.

- Apply SOLID principles where possible.

- Ensure adherence to SOLID and DRY principles.

- Detect potential performance bottlenecks and suggest improvements.

- Generate API documentation in OpenAPI 3.0 format.

- Include clear descriptions, request/response examples, and error codes.