# Rule Checker Application Code Documentation

## Table of Contents

1. Overview  
2. Workflow  
3. Architectural Decisions  
4. Assumptions  
5. Module-Specific Documentation  
6. Future Enhancements  
7. Test Cases  
8. Conclusion

## Overview

The Rule Checker Application is designed to analyze and correct sentences based on a set of grammatical and procedural rules.   
It uses SpaCy for sentence analysis and the Groq language model for intelligent corrections, ensuring that text adheres to   
best practices for clarity and precision.  
  
This application is modular, making it scalable, maintainable, and adaptable for future improvements.

## Workflow

**Step 1: User Input**  
- The application prompts the user to input sentences for analysis.  
- Sentences can be entered one by one or as a block of text. For example:  
 - Single sentence: Turn shaft assembly.  
 - Multiple sentences: Turn shaft assembly. The safety procedures are supplied by the manufacturer.   
 Set the TEST switch to the middle position and release the SHORT-CIRCUIT TEST switch.  
  
**Step 2: Text Analysis (RuleChecker)**  
- The `RuleChecker` module uses SpaCy's NLP capabilities to:  
 - Identify grammatical issues.  
 - Detect sentences that do not follow procedural rules.  
  
**Step 3: Correction (GroqChain)**  
- Flagged sentences are sent to the Groq cloud’s language model via the `GroqChain` module.  
- The language model corrects issues based on predefined rules provided through a prompt template.  
  
**Step 4: Logging**  
- Centralized logging captures key events and outputs across all modules.  
- Logs are stored in `logs/application.log` for traceability.  
  
**Step 5: Output**  
- Corrected sentences are displayed alongside their original versions for user review.

## Architectural Decisions

**1. Modular Structure**  
- Rationale: A modular architecture ensures separation of concerns and makes the application easy to maintain and extend.  
- Components:  
 - main.py: Orchestrates the workflow.  
 - rule\_checker.py: Handles text analysis.  
 - groq\_chain.py: Connects to the Groq cloud’s language model.  
 - logger\_config.py: Manages centralized logging.  
 - config.py: Stores sensitive configurations.  
  
**2. Dependency Selection**  
- SpaCy: Selected for its efficient NLP capabilities, enabling robust sentence parsing and analysis.  
- Groq cloud Language Model: Chosen for its context-aware correction abilities.  
  
**3. Logging**  
- All modules write logs to a centralized file for consistency and easier debugging.  
- Logs include timestamps, filenames, and log levels.

**4. Prompt-Driven Corrections**  
- A structured prompt guides the Groq cloud’s language model to provide consistent corrections while adhering to predefined rules.

## Assumptions

1. Input:  
 - Input text is in English.  
 - Sentences are complete and grammatically valid.  
  
2. Model Dependency:  
 - Accuracy depends on SpaCy’s `en\_core\_web\_sm` and the Groq language model.  
  
3. System Requirements:  
 - Python 3.8 or later.  
 - Libraries listed in `requirements.txt` are installed.  
  
4. Performance:  
 - The application is optimized for small to medium-sized text inputs.

## Module-Specific Documentation

**#main.py**  
- Purpose: Orchestrates the overall workflow of the application.  
- Key Responsibilities:  
 - Handles user input.  
 - Integrates `RuleChecker` and `GroqChain` modules.  
 - Outputs corrected sentences.  
  
**#rule\_checker.py**  
- Purpose: Analyzes text for grammatical and procedural issues.  
- Key Responsibilities:  
 - Parses text using SpaCy.  
 - Detects issues like missing articles, passive voice, and multiple instructions.  
  
**# groq\_chain.py**  
- Purpose: Connects to the Groq language model for sentence corrections.  
- Key Responsibilities:  
 - Corrects sentences flagged by the `RuleChecker`.  
 - Constructs prompts to guide the Groq language model.  
  
**# logger\_config.py**  
- Purpose: Provides a centralized logging configuration.  
- Responsibilities:  
 - Ensures all modules write logs to the same file.  
 - Includes timestamps and filenames in log messages.  
  
**# config.py**  
- Purpose: Stores sensitive data, such as API keys.  
- Best Practices:  
 - Exclude config.py from version control using `.gitignore`.  
 - Use environment variables for sensitive configurations.

## Future Enhancements

1. Multi-language Support:  
 - Extend the application to support additional languages.  
  
2. Enhanced Rule Set:  
 - Add advanced grammatical and procedural rules.  
  
3. Large Language Models:  
 - Replace the current model with a larger language model (e.g., GPT-4) if a paid API key is available.  
  
4. Web Interface:  
 - Develop a web-based UI for improved usability.

## Test Cases

Test Case 1  
Original Sentence: Turn shaft assembly.  
Issues: Missing article or demonstrative adjective.  
Corrected Sentence: Turn the shaft assembly.  
  
 Test Case 2  
Original Sentence: The main gear leg is held by the side stay.  
Issues: Not in active voice.  
Corrected Sentence: The side stay holds the main gear leg.  
  
 Test Case 3  
Original Sentence:Set the TEST switch to the middle position and release the SHORT-CIRCUIT TEST switch.  
Issues:Contains multiple instructions.  
Corrected Sentence:  
A. Set the TEST switch to the middle position.  
B. Release the SHORT-CIRCUIT TEST switch.  
Test Case 4  
Original Sentence: The test can be continued.  
Issues: Not in active voice., Not in imperative form.  
Corrected Sentence: Continue the test.  
  
 Test Case 5  
Original Sentence: Turn shaft assembly. The safety procedures are supplied by the manufacturer.   
Set the TEST switch to the middle position and release the SHORT-CIRCUIT TEST switch.  
Issues: Combination of issues, including missing article, passive voice, and multiple instructions.  
Corrected Sentences:   
1. Turn the shaft assembly.  
2. The manufacturer supplies the safety procedures.  
3. A. Set the TEST switch to the middle position.  
 B. Release the SHORT-CIRCUIT TEST switch.

## Conclusion

This documentation outlines the application’s structure, workflow, and key decisions made during development.   
The modular architecture, combined with centralized logging and prompt-driven corrections, ensures scalability and maintainability.   
Future enhancements will further expand the application’s capabilities, making it a comprehensive tool for sentence analysis and correction.