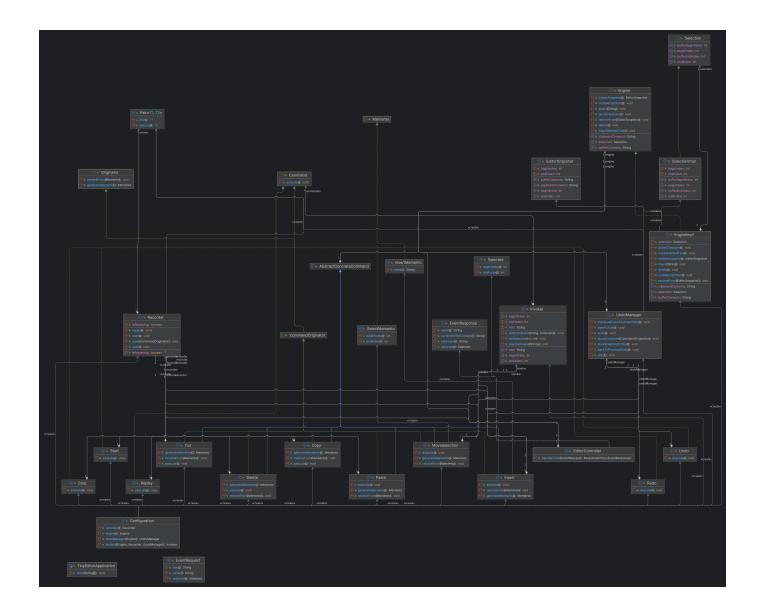# Design

The tiny text editor is a web-based frontend that interacts using REST API with a backend written in Java. The backend is responsible for managing the text editing operations and providing the frontend with the necessary functionality to perform these operations. The backend is built using the Spring Boot framework, which provides a robust and scalable platform for developing web applications and REST APIs. The frontend is built using Electron, which allows the use of web technologies to create a desktop application.

To ensure the robustness and correctness of the backend, over 40 unit tests were written using JUnit and Mockito, achieving over 90% test coverage. These tests verify the functionality of various backend components, ensuring the editor performs efficiently and reliably.

This section provides an overview of the design decisions made during the development of the application and the rationale behind each choice.

UML Diagram

Backend Design choices

Command Pattern

The command pattern is used to encapsulate the operations performed by the editor. Each command is represented as an object that contains the necessary information to execute the operation. This design pattern allows for easy extensibility and maintainability of the editor, as new commands can be added without modifying the existing codebase. The command pattern also provides a way to undo and redo operations, as each command object stores the state of the editor before and after the operation.

Memento Pattern

The memento pattern is used for recording the state of the editor and then saving it. After that those commands can be replayed. It is also used to implement the undo/redo functionality. In order to implement the memento pattern for undo and redo functionality we choose 4 lists, which are past states and future states. These 2 lists consist of snapshots of the engine at certain stages. For example a snapshot is capture and

stored in the past state list at the beginning. Then a number k is chosen, default 5 which decides after how many commands another state is captured. In order to undo we go to the last state and apply n number of commands so that the state before the current state is reached. and that current state is then added to the future states list. If redo is called then the last state is taken from the future states list and applied to the engine.