RULE FRAMEWORK EDITOR

SOFTWARE SYSTEMS DEVELOPMENT

Monsoon - 2022

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

INSTRUCTOR : SAI ANIRUDH KARI

SUBMITTED BY :

PRUDHVINADH REDDY K (2022201007)

PRIYANSHU BANSAL (2022201079)

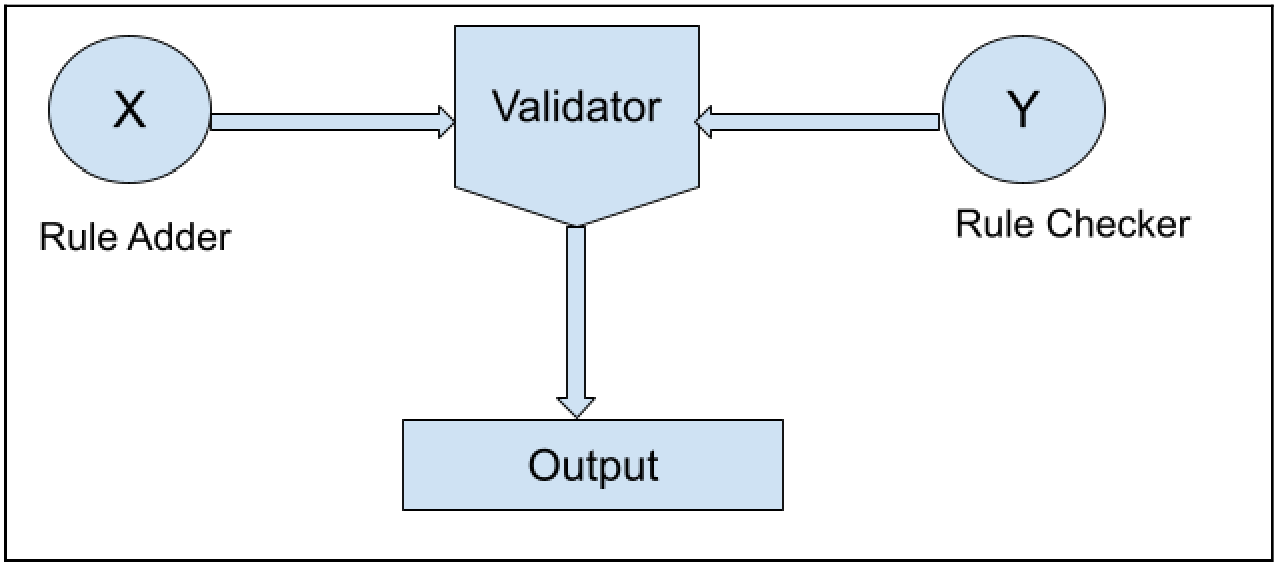
NEERAJ ASDEV (2022201056)

SHIVANI SINHA (2022202020)

INTRODUCTION :

- Every programming language (C, Java, etc.) has some predefined set of rules that apply to their syntax. These rules validate every single word/line of the code when the code is being parsed.
- The parser usually accepts the input and generates the output either as a parse tree or some positive statement when passed. Otherwise, it would generate a syntax error. Or the most simplest output forms can be either valid or invalid.
- This project aims to develop a system where we would implement the function of a parser to parse some simple statements called CONDITIONAL CONSTRUCTS (which we will see later on..) and it will validate them in multiple syntax definitions (language/environment).

BASIC STRUCTURE :



TECHNOLOGIES USED :

- FRONT END : React
- BACKEND : Mongodb
- Node js is base for all above
- Express for Routing
- MERN STACK

IMPLEMENTATION :

- When you use the rule editor to build a rule, you add the appropriate terms and phrases to the editing area. Appropriate authentication is required to add rules to the rule editor. Rule Adder can be done by two ways : Either by writing manually or by uploading a file consisting of rules.
- The rule editor provides an editing area into which you add terms and phrases to build rules. You can add terms and phrases in the following ways:
    - Type directly in the editing area
    - Copy text from another editor or application, and paste it into the editing area
    - Select predefined terms and phrases from a completion menu
- There are three basic elements in the Rule Set Editor:
    1. Conditions
    2. Actions
    3. Rules
- In the Rule Set Editor, you combine conditions with actions to create rules that you can use to implement validator

WORKING :

- Validator accepts adder’s JSON file as reference/rulebook and checker’s code as an input then it will parse it accordingly. Basically, a validator can also be called as a parser.
- Tokenization : It will tokenize the given code by checker/tester and verify it against adder’s conditional construct syntax.
- Generates tokens and their types and passes them onto the next stage i.e. parsing.
- Parsing : Parses the tokens and identifies whether the code is syntactically correct or not.
- Finally, provides an output to the user or checker whether the code entered is valid or not.

EXTENDED FEATURES :

- **SOFT CONSTRUCT VALIDATION** : Extended support for these soft constructs is implemented where conditional rules are stored in JSON format. Soft constructs will be checked only after the hard constructs are verified.
- **ENCODING MODULES** : To enhance security, the validated code can be converted into a secure encrypted hash using base-64 encryption method. Decoding options will be provided to the user in the form of a checkbox on the top where the encrypted hash will be decoded back to the original code.