Software Systems Development

Semester Project

Topic: Customized Rule Editor Framework

REPORT FILE

Srikant Konduri 2022201017 CSE
Abhiram G P 2022201024 CSE
Prateek Pandey 2022201035 CSE
Anshita Shrivastava 2022202014 CSIS

Intoduction

We have created an application where an user can write custom rules and upload it into our application. Our application server takes this file as input and processes it to retrieve data as to how the described code should be validated and it returns a json file containing all the basic conditional constructs described in the input file along with their basic syntax.

At the front-end this json file is received and it is used to furnish a pallet of buttons that has conditional constructs . The front-end also provides a code editor where the code will appear for each conditional construct that is selected from the pallet at the position of the cursor. Now any code we write here in the editor can be validated according to the rules we have given by just clicking the validate button. If the code is valid we will get a pop up saying its valid. If the code is having wrong syntax a pop up will arise providing the line number at which the code went wrong.

<u>Implementation</u>

First we are uploading a rule.json file which contains the rules that describe the conditional constructs of our language. It contains 3 attributes types array, constructs array and specialSymbols array.

1. Types

This attribute contains conditional construct types like if, else etc. The following is an example of a content of type.

```
{
  "type": "if",
  "order": [["condition"],["scope"]]
}
```

2. Constructs

This attribute contains the rules for our conditional construct types like if, else etc. The following is an example of a content of Constructs.

```
{
     "name": "EIf",
     "type": "else-if",
     "conditionStart": "#",
     "conditionEnd": "#",
     "scopeStart": ":",
     "scopeEnd": "!",
     "pre": ["If", "EIf"]
 }
```

3. specialSymbols

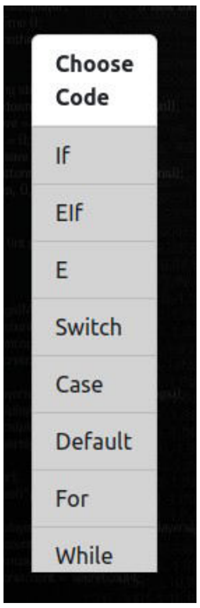
This contains all the special symbols used in our language.

```
"specialSymbols": ["#", ":", "!", "/"]
```

This file which we uploaded into our application is then sent to the server where it is processed and it returns another json file as shown below. For this we call localhost:5001/api/upload (considering our server runs on localhost 5001)

```
{
 statements: [
   { title: 'If', code: 'If#<cond>#:<code here>!' },
   { title: 'Elf', code: 'Elf#<cond>#:<code here>!' },
   { title: 'E', code: 'E:<code here>!' },
   { title: 'Switch', code: 'Switch#<cond>#:<code here>!' },
   { title: 'Case', code: 'Case#<cond>#:<code here>!' },
   { title: 'Default', code: 'Default:<code here>!' },
   { title: 'For', code: 'For#<cond>/<cond>/<cond>#:<code here>!' },
   { title: 'While', code: 'While#<cond>#:<code here>!' },
   { title: 'Do', code: 'Do:<code here>!' },
   { title: 'Do-While', code: 'Do-While#<cond>#<code here>' }
 ]
}
```

This returned json data is used in the front-end to populate the button pallete.



Now if you press any of the buttons in this pallete the corresponding code will appear on our code editor at the position of our cursor.

Validation

Validation of our code is done on server side. Once we have entered the code in the editor and when we click validate button we call localhost:5001/api/process (considering our server runs on localhost 5001) and the code in the editor is sent for validation to our backend where its validated.

If the code is valid we get a response like shown below.

```
{
        "valid": true,
        "lineNumber": -1
    }
```

If the code is invalid we get a response like shown below.

```
{
        "valid": false,
        "lineNumber": 24
    }
```

This response is used to show if the code is valid or not at the front end.

Integration

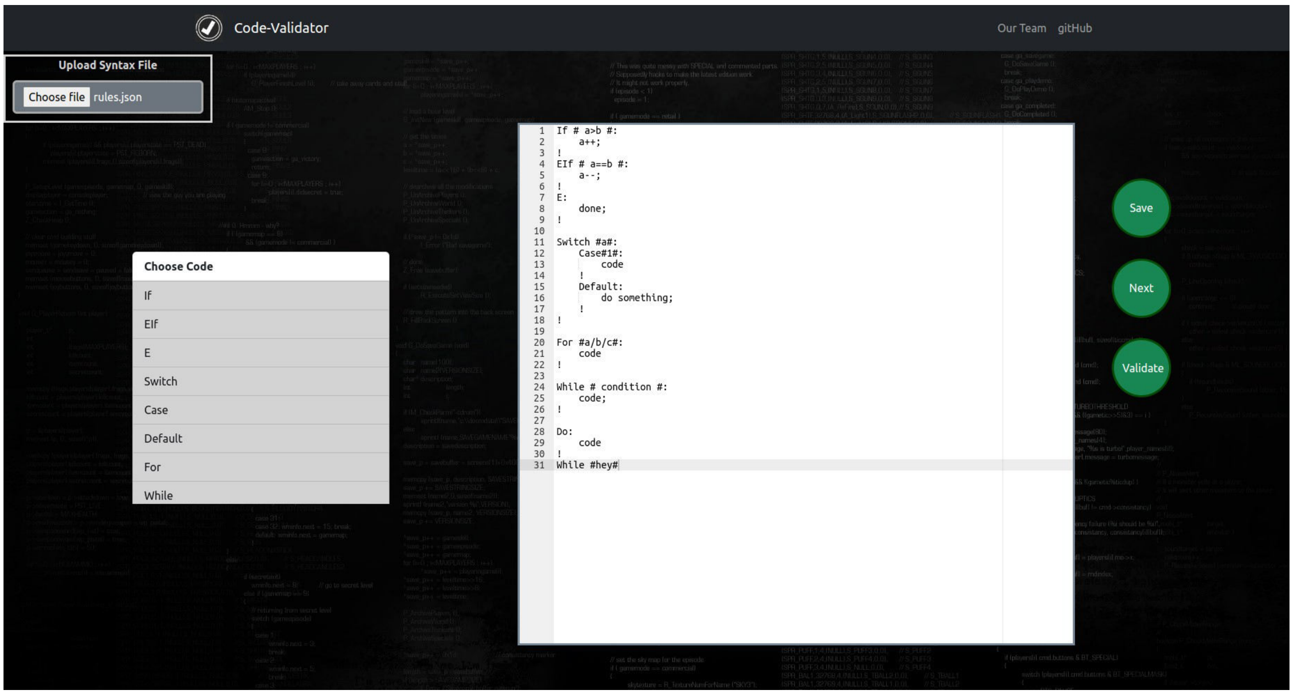
All the code that is validated is convereted to base64 format and is pushed into an array. When the “Save” button is pressed a proj_id is sent to the backend which will search in the “Integrate” collection for that particular document if the document exists it will update it with latest base_64_encoded array in the DB.

Front End:

1. The user uploads the json file having the conditional constructs using the “Upload File” button.

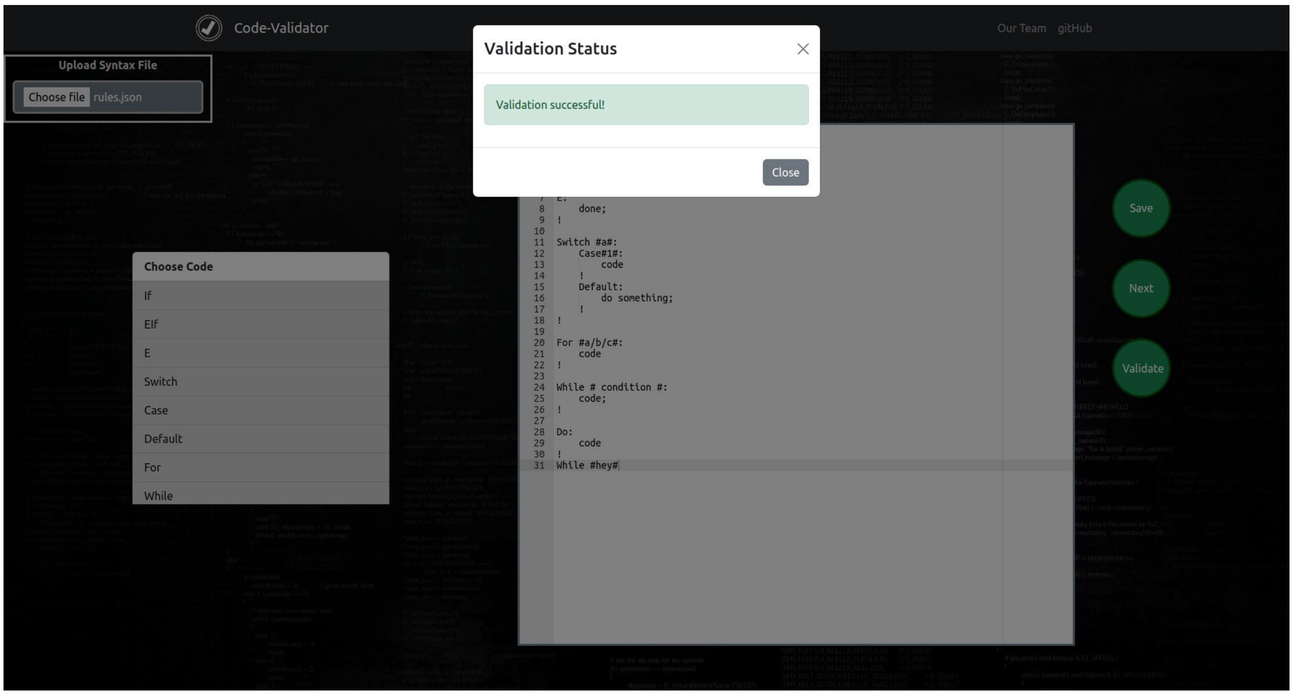
2. It displays a list of all the conditional constructs in the “Choose Code” area.

3. The user can click on any of the code from “Choose Code” which goes to the code editor or write the code in the code editor.



```
If # a>b #:
    a++;
!
EIf # a==b #:
    a--;
!
E:
    done;
!

Switch #a#:
    Case#1#:
        code
    !
    Default:
        do something;
    !
!

For #a/b/c#:
    code
!

While # condition #:
    code;
!

Do:
    code
!
While #hey#
```

4. The user clicks on the validate button which displays either “Validation Successful” on successful validation or “Error..” with line number in case of any error.



Validation successful!

```
E:
    done;
!

Switch #a#:
    Case#1#:
        code
    !
    Default:
        do something;
    !
!

For #a/b/c#:
    code
!

While # condition #:
    code;
!

Do:
    code
!
While #hey#
```

5. The string gets converted to base64 and on “Save” the data gets saved in MongoDB.
On click of “Next” the user moves to next page.