Approach

I have created a **Python/Django** service to host **CreditPolicies,** and corresponding Javascript client to interact with it.

Each credit policy will have it's own **ConditionTree**, which is a tree of conditions which are checked against given **customer-data/credit-application.** I feel this will be a generic enough solution, which can be used to create many complicated credit policies, because in the end, a **CreditPolicy** is going to be a combination of various user-defined rules and regulations.

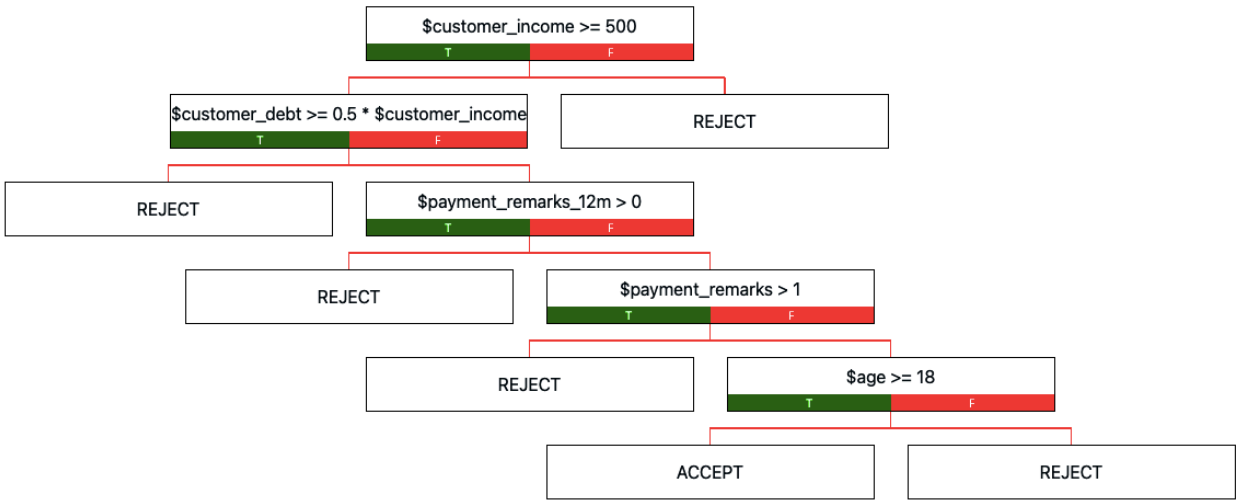


Figure 1.0

The above figure depicts an example of the condition tree, which represents the rules and regulations of the sample credit policy that is described in the problem statement, given to me.

Overview of the process.

- User creates a credit policy. Following information is required for creating a credit policy
  - **Name**: Name of the Policy
  - **Attributes:** The attributes of the policy. Attributes are the things that a condition/expression is made of. For example in the expression **$customer_income >= 500, customer_income** is a policy attribute of type **number.** When the user creates a conditional expression he can only use the attributes, defined in the policy. For example **$customer_pan == 100,** will not be a valid conditional expression. Following data is needed to create a attribute
    - **Name**: Attribute name
    - **Type:** type of the attribute
    - **ExampleValue:** The value will be used substituted in place of the attribute, to test the correctness of the conditional expression.
  - **RootCondition:** The root conditional expression, for the policy. Following info is required to for root_condition
    - **Name:** Condition Name
    - **Expression:** Conditional expression. Ex: **$customer_income >= 500**
- The policy that got created is still incomplete, as the condition tree is not complete. Now the user will add further values/children. For each possible value of the condition, i.e True or False, user has three options, also shown in the image
  - ACCEPT
  - REJECT (Also needs to give a rejection error)
  - NEW CONDITION

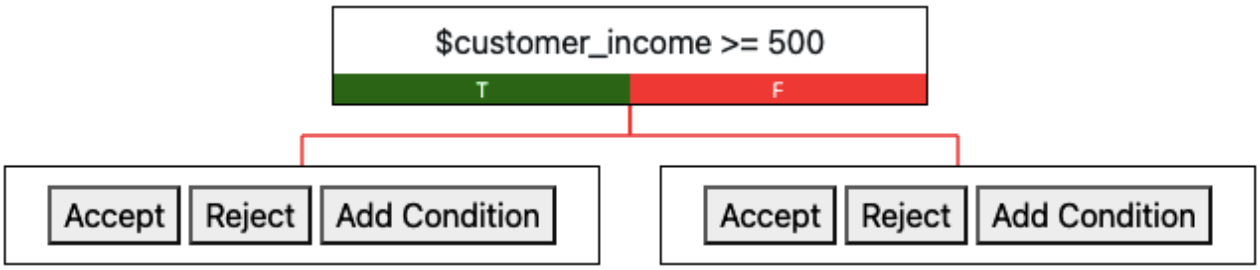
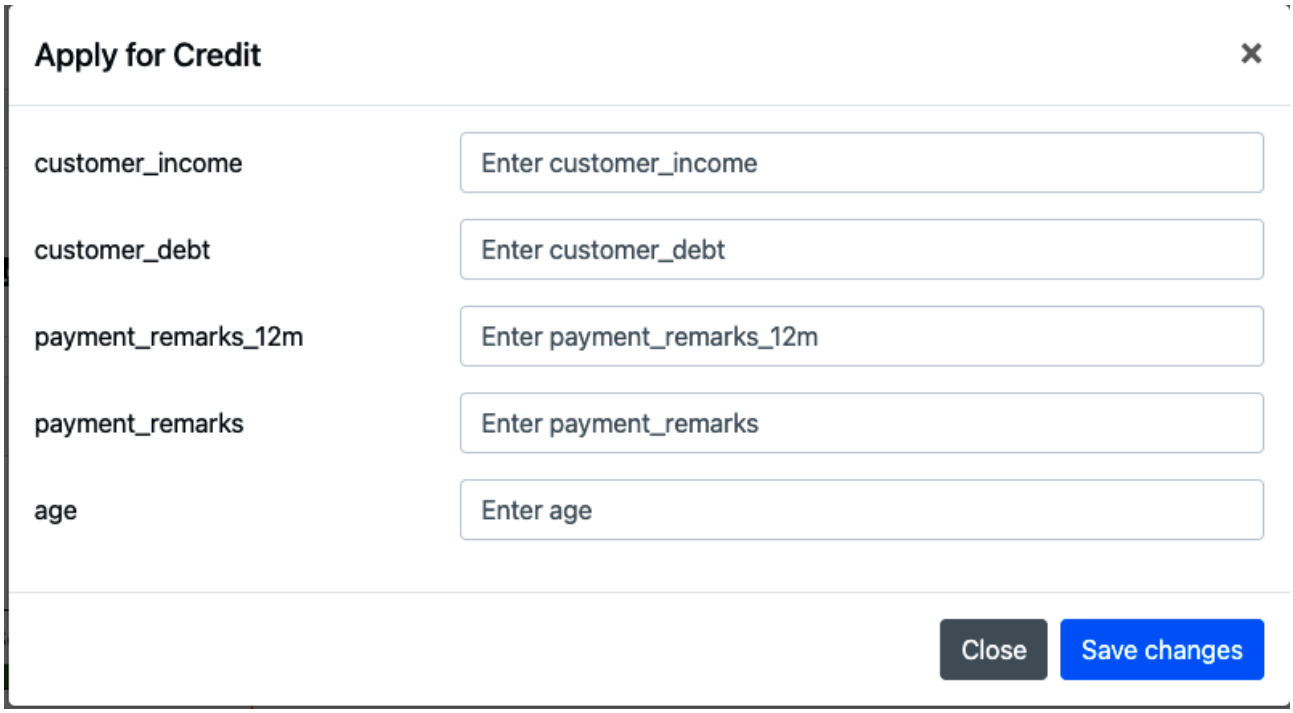


Figure 1.1

- User will continue building the condition tree in a similar fashion, until it's complete.



- Once a credit policy has been created, user can apply for credit, providing data for the policy attributes. The system checks evaluates each condition in the condition tree, to get a result and sends it to client. If the application is rejected, a rejection error is shown. (The above image shows the data screen to apply for credit.)

Build Process:

To build the project download the project, and just run command **./build.sh.**

This command does the following things,
- Starts an nginx server.
- Starts the database server.
- Builds and starts up the python/django service.

Urls:
- **ApiService :**
  - **http://localhost:9010/api/**
- **ApiDoc**: Has a description of apis, and the corresponding request and response data
  - **http://localhost:9010/api/swagger/**
- **Client:**
  - **http://localhost:9010/client**/