@push.rocks/smartrule

A library for creating and managing decision trees with smart rules.

- readme.md for @push.rocks/smartrule

readme.md for @push.rocks/smartrule

@push.rocks/smartrule

a smart rule library for handling decision trees.

Install

To install `@push.rocks/smartrule`, use the following command with npm:

```
npm install @push.rocks/smartrule --save
```

Or if you prefer using Yarn:

```
yarn add @push.rocks/smartrule
```

Ensure you have TypeScript and necessary typings installed in your project. If you haven't, you can add TypeScript and the types for Node.js by running:

```
npm install typescript @types/node --save-dev
```

Usage

`@push.rocks/smartrule` is designed to simplify handling decision trees in your application, allowing you to manage complex business logic based on a set of predefined rules. Below we will walk through using this library to create and manage a simple set of decision rules programmatically.

Getting Started

First, ensure that you import `SmartRule` & define interfaces or types that you plan to use with it if necessary:

```
import { SmartRule } from '@push.rocks/smartrule';

interface IMessage {
  id: string;
  content: string;
  userType: 'admin' | 'user';
}
```

Creating a Decision Rule Instance

To start using smart rules, instantiate `SmartRule` with your data type:

```
const messageRule = new SmartRule<IMessage>();
```

Adding Rules

Rules are added by specifying a priority, a check function, and an action function. The check function decides whether the action function should be executed based on the input object. Let's define rules for our `IMessage` objects:

```
// Rule 1: Greet Admins
messageRule.createRule(
  1, // priority
  async (message: IMessage) => {
    if (message.userType === 'admin') {
      return 'apply-continue'; // Apply action and continue evaluating
    }
    return 'continue'; // Continue without applying
  },
  async (message: IMessage) => {
    console.log(`Welcome, admin with ID: ${message.id}`);
  }
);

// Rule 2: Delete messages containing forbidden words
messageRule.createRule(
```

```
  2, // priority is higher, so this rule gets evaluated first
  async (message: IMessage) => {
    const forbiddenWords = ['forbidden', 'unallowed'];
    const containsForbiddenWord = forbiddenWords.some((word) =>
      message.content.includes(word)
    );
    return containsForbiddenWord ? 'apply-stop' : 'continue'; // Stops evaluating if forbidden word found
  },
  async (message: IMessage) => {
    console.log(`Message with ID: ${message.id} contained a forbidden word and was deleted.`);
    // Logic to delete the message would go here
  }
);
```

Making Decisions

To evaluate an object against the defined rules, use the `makeDecision` method:

```
const testMessage: IMessage = {
  id: '001',
  content: 'This is a welcome message for an admin containing a forbidden word',
  userType: 'admin'
};

messageRule.makeDecision(testMessage).then(() => {
  console.log('Decision making process completed.');
  // Implement further logic as needed
});
```

Advanced Usage

- **Rule Prioritization**: You can manage the order in which rules are evaluated based on their priority. Rules with lower numbers are evaluated first.
- **Decision Flow Control**: Through your check function return value (`apply-continue`, `apply-stop`, `continue`, or `stop`), you can precisely control how the decision-making process progresses after evaluating each rule.
- **Asynchronous Support**: Both check and action functions support asynchronous operations, making it easy to integrate with APIs, databases, or other asynchronous data

sources in your rules.

Handling Complex Scenarios

`@push.rocks/smartrule` is especially useful when you have complex decision-making processes in your application. By segmenting the logic into individual rules, you make your code more organized, maintainable, and scalable. You can easily add new rules or modify existing ones without affecting other parts of your decision tree.

By following this guide, you should now have a basic understanding of how to use `@push.rocks/smartrule` for managing decision trees in TypeScript using ESM syntax. This library provides a powerful yet simple abstraction for creating and managing complex decision-making logic, allowing you to focus on implementing the business rules specific to your application.

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

Capital GmbH of any derivative works.