Creating an ESLint plugin

Creating a plugin is a bit more complicated than creating a shareable configuration because you need to not only create the rules, but you also need to learn about the AST syntax too accomplish the task you want.

Even with the added difficulty, custom plugins widen the possibilities of what you can do with ESLint and it makes it possible to test for things outside Javascript.

I've always wanted to use ESLint to detect issues with my HTML files similar to some of the rules AMP enforces. Did I forget an alt attribute for images? Did I forget to set up width and height for videos and Images?

HTML ESLint, an HTML plugin for ESLint provides a basic set of rules to lint your HTML. These rules are additions to HTML ESLint that cover specific things I'm interested in. Some of these rules are based in code from HTML ESLint and some are completely new.

Checking if an image has a width attribute

I like to check if an image has an explicit width attribute. If it doesn't then it's a problem because it may contribute to layout shifts as the image is loaded.

The first part of the plugin is the metadata. It contains information about the plugin and the rule it defines.

```
module.exports = {
  meta: {
    type: "code",

    docs: {
      description: "Require `width` attribute at `<img>` tag",
 "Require `width` attribute at `<img>` tag"e: null,
    schema: [],
    messages: {
      [MESSAGE_IDS.MISSING_WIDTH]: "Missing `width` attribute at `<img>` t
```

```
    },
  },
"Missing `width` attribute at `<img>` tag"
```

The second, and most important, part is the rule itself. We create the rule with the context as an attribute. If the node we're linting doesn't pass the test defined in the `hasWidthAttrAndValue` function then we emit the error message using the `report` method of the context.

```
  create(context) {
    return {
      Img(node) {
        if (!hasWidthAttrAndValue(node)) {
          context.report({
            node: node.startTag,
            messageId: MESSAGE_IDS.MISSING_WIDTH,
          });
        }
      },
    };
  },
};
```

The final piece of the rule is the testing function, `hasWidthAttrAndValue`. It takes a node as the argument and returns a boolean indicating whether the node has a width attribute and the value is a number.

```
function hasWidthAttrAndValue(node) {
  return (node.attrs || []).some((attr) => {
    return attr.name === "width" && attr.value.trim().length > 0;
  });
}
```

We could use the same technique to test for the `height` attribute and its value or the presence of an `alt` attribute (which can have a value or not depending on the use case).

Disallow duplicate attributes

As I'm typing HTML I don't always pay attention to the attributes I've added. This is doubly true for my automated HTML generration tools.

I want ESLint to flag this mistake so that I can fix it.

First we create the metadata block for the rule.

```
module.exports = {
  meta: {
    type: "code",

    docs: {
      description: "Disallow to use duplicate attributes",
      category: RULE_CATEGORY.BEST_PRACTICE,
      recommended: true,
    },

    fixable: null,
    schema: [],
    messages: {
      [MESSAGE_IDS.DUPLICATE_ATTRS]:
        "The attribute '{{attrName}}' is duplicated.",
    },
  },
```

The rule function is different. We want to use the rules in all the nodes that match the context.

For each node we we want to do the following:

1. Check if the node has any attributes
2. Look through the attributes and see if the name is already in the array
    1. if the name is already in the array report an error, the attribute is duplicated
    2. If not, add the name to the array

```
  create(context) {
    return {
      "*"(node) {
        if (Array.isArray(node.attrs)) {
          const attrsSet = new Set();
          node.attrs.forEach((attr) => {
            if (attrsSet.has(attr.name)) {
              context.report({
                node: node.startTag,
                data: {
                  attrName: attr.name,
                },
                messageId: MESSAGE_IDS.DUPLICATE_ATTRS,
              });
            } else {
              attrsSet.add(attr.name);
            }
          });
        }
      },
    };
  },
};
```

These two examples show two ways of writing ESLint rules. There are a lot more ideas to explore when writing rules. That will be an exercise for another post.