



Name:

Usama Shafiq (181400180)

Rana Ali Raza (181400054)

Usama Illyas (181400036)

Assignment:

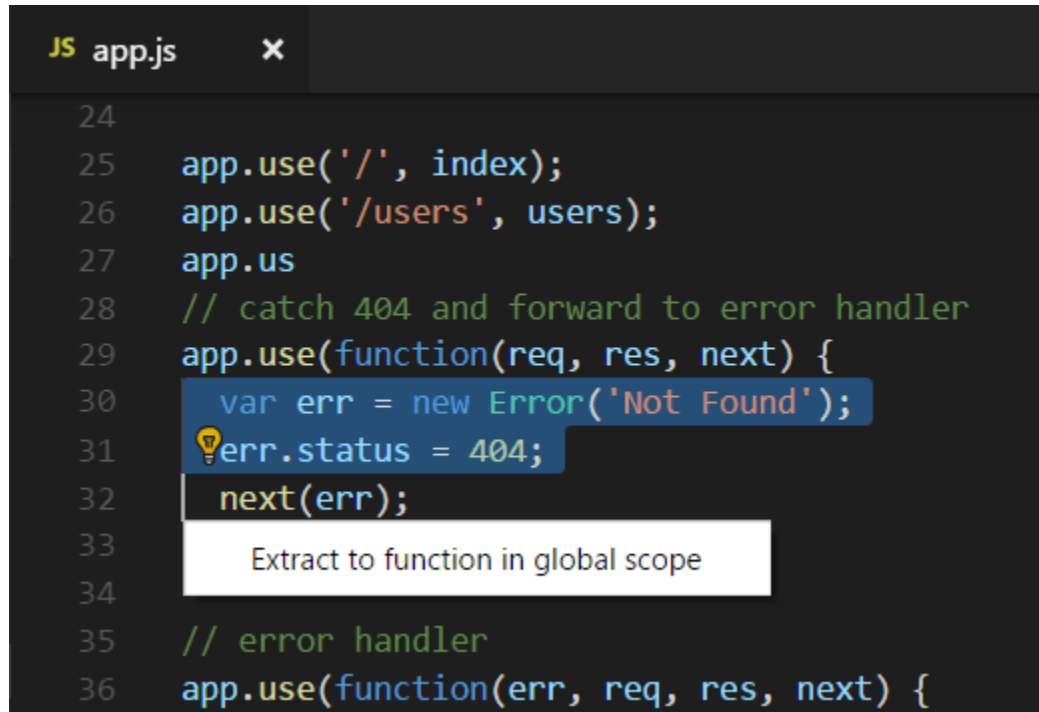
2

Course:


CS-368A-Software Re-Engineering

Refactoring:

Source code refactoring can improve the quality and maintainability of your project by restructuring your code while not modifying the runtime behavior. Visual Studio Code supports refactoring operations (refactoring's) such as Extract Method and Extract Variable to improve your code base from within your editor.



The screenshot shows a Visual Studio Code editor window with a file named 'app.js'. The code is as follows:

```
24
25 app.use('/', index);
26 app.use('/users', users);
27 app.us
28 // catch 404 and forward to error handler
29 app.use(function(req, res, next) {
30     var err = new Error('Not Found');
31      err.status = 404;
32     next(err);
33
34
35 // error handler
36 app.use(function(err, req, res, next) {
```

A blue selection box highlights the code block from line 30 to 32. A white tooltip box with a lightbulb icon is positioned over the selection, displaying the text 'Extract to function in global scope'.

For example:

A common refactoring used to avoid duplicating code (a maintenance headache) is the Extract Method refactoring, where you select source code that you'd like to reuse elsewhere and pull it out into its own shared method.

Refactoring's are provided by a language service and VS Code has built-in support for Typescript and JavaScript refactoring through the Typescript language service. Refactoring support for other programming languages is provided through VS Code extensions that contribute language services. The UI and commands for refactoring are the same across languages, and in this topic we'll demonstrate refactoring support with the Typescript language service.

Code Actions = Quick Fixes and refactoring's:

In VS Code, Code Actions can provide both refactoring's and quick fixes for detected issues (highlighted with green squiggles). An available Code Action is announced by a lightbulb near the source code when the cursor is on a squiggle or selected text region. Clicking on the Code Action lightbulb or using the Quick Fix command Ctrl+. Will display Quick Fixes and refactoring's.

If you'd just like to see refactoring's without Quick Fixes, you can use the Refactor command (Ctrl+Shift+R)

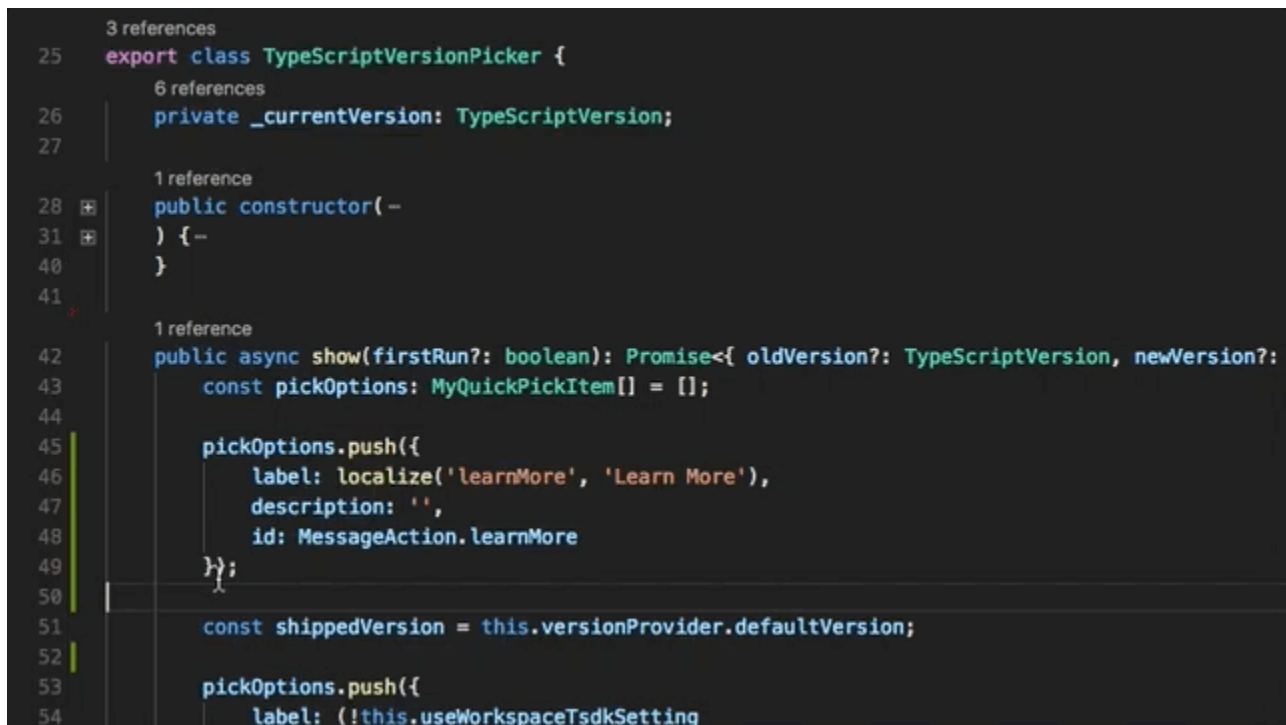
Refactoring actions:

Extract Method:

Select the source code you'd like to extract and then click on the lightbulb in the gutter or press(Ctrl+.) to see available refactoring's. Source code fragments can be extracted into a new method, or into a new function at various different scopes. During the extract refactoring, you will be prompted to provide a meaningful name.

Extract Variable:

Typescript language service provides Extract to const refactoring to create a new local variable for the currently selected expression:



When working with classes, you can also extract a value to a new property.

Rename symbol:

Renaming is a common operation related to refactoring source code and VS Code has a separate Rename Symbol command (F2). Some languages support rename symbol across files. Press F2 and then type the new desired name and press Enter. All usages of the symbol will be renamed, across files.



```
JS app.js x
7
8 var index = require('./routes/index');
9 var users = require('./routes/users');
10
11 var app = express();
12     expressApp
13 // view engine setup
14 app.set('views', path.join(__dirname, 'views'));
15 app.set('view engine', 'jade');
16
```

Key bindings for Code Actions:

The `editor.action.codeAction` command lets you configure key bindings for specific Code Actions. This key binding, for example, triggers the Extract function refactoring Code Actions:

```
{
  "Key": "ctrl+shift+r ctrl+e",
  "Command": "editor.action.codeAction",
  "Args": {
    "Kind": "refactor.extract.function"
  }
}
```

Code Action kinds are specified by extensions using the enhanced `CodeActionProvided` API. Kinds are hierarchical, so `"kind": "refactor"` will show all refactoring Code Actions, whereas `"kind": "refactor.extract.function"` will only show Extract function refactoring's.

Using the above key binding, if only a single `"refactor.extract.function"` Code Action is available, it will be automatically applied. If multiple Extract function Code Actions are available, we bring up a context menu to select them:

```
private static getApplyFromUser(arg: any) {
  switch (typeof arg.apply === 'string' ? arg.apply.toLowerCase() : '') {
    case 'first':
      return CodeActionAutoApply.First;

    case 'never':
      return CodeActionAutoApply.Never;

    case 'ifsingle':
    default:
      return CodeActionAutoApply.IfSingle;
  }
}
```

Extract to method in class

Extract to function in module

You can also control how/when Code Actions are automatically applied using the apply argument:

```
{
  "Key": "ctrl+shift+r ctrl+e",
  "Command": "editor.action.codeAction",
  "Args": {
    "Kind": "refactor.extract.function",
    "Apply": "first"
  }
}
```

Valid values for “apply”:

- “First”- Always automatically apply the first available Code Action.
- “If single”- Default. Automatically apply the Code Action if only one is available. Otherwise, show the context menu.
- “Never”- Always show the Code Action context menu, even if only a single Code Action is available.

When a Code Action key binding is configured with “preferred”: true, only preferred Quick Fixes and refactoring’s are shown. A preferred Quick Fix addresses the underlying error, while a preferred refactoring is the most common refactoring choice. For example, while multiple refactor.extract.constant refactoring’s may exist, each extracting to a different scope in the file, the preferred refactor.extract.constant refactoring is the one that extracts to a local variable.

This key binding uses “preferred”: true to create a refactoring that always tries to extract the selected source code to a constant in the local scope:

```
{
  "Key": "shift+ctrl+e",
  "Command": "editor.action.codeAction",
  "Args": {
    "Kind": "refactor.extract.constant",
    "Preferred": true,
    "Apply": "if single"
  }
}
```

Extensions with refactoring's:

You can find extensions that support refactoring by looking in the VS Code Marketplace. You can go to the Extensions view (Ctrl+Shift+X) and type 'refactor' in the search box. You can then sort by install count or ratings to see which extensions are popular.



Python

MS-python

IntelliSense (Pylance), Linting, Debugging (multi-threaded, remote), Jupyter Notebooks, code formatting, refactoring, unit tests, and more.



Language Support for Java(TM) by Red Hat

Red hat

Java Linting, intelligence, formatting, refactoring, Maven/Gradle support and more...



PHP IntelliSense

Felixbecker

Advanced Auto completion and Refactoring support for PHP



PHP IntelliSense

Zobo

Advanced Auto completion and Refactoring support for PHP