

VS Code

Visual Studio Code

Course Introduction

Fast, cross platform, intellisense and auto complete, debugging, rich refactoring, sits perfectly between editor and IDE.

Editor vs. IDE



Main Parts

UI Framework - Electron

Editor - Monaco

Brains – TypeScript & OmniSharp/Roslyn

Command palette shortcut - Shift + Command + P

Get up and Running with VS Code

On command prompt use command 'code' to open this editor with options -r and -n.

We can use auto save in on or off mode. Using preview mode for markdown code.

Use '>' and '?' after clicking Ctrl + P.

Using side by side editing and viewing into split mode using command with 1, 2, 3 keys or '\`' key.

Getting Ready for Node or ASP.NET 5

Getting Node.js

```
choco install nodejs
choco install nodejs.install
```

Getting ASP.NET 5

```
$ brew tap aspnet/dnx
$ brew update
$ echo source dnmv.sh >> ${HOME}/.bash_profile
$ brew install dnmv
$ dnmv upgrade
```

NPM – package manager for Node like NuGet to install package.

`npm install -g package-name`

```
$ npm install -g yo gulp bower typescript tsd
$ npm install -g generator-hottowel generator-aspnet
```

Refactoring

Bracket matching and emmet

Use 'go to bracket' command from command prompt to go to matching bracket or use command + shift + [.

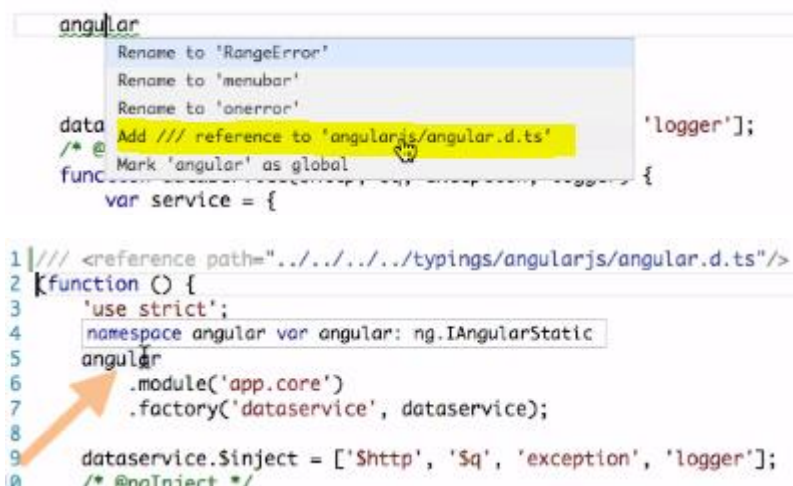
Using Emmet feature – `div > ul > li * 5` or `div.container` then hit tab.

Download the cheat sheet from - <https://docs.emmet.io/>

Use multi cursor feature to have find and replace like feature.

Shortcut to have multi cursor selection editing – alt + mouse click on multiple variables

We can use below highlighted options for the 'any' type variables in JS file which VS code doesn't recognized, It will give its intellisense and code complete for it as well:



We can also use tsd file approach which comes from NPM so that we do not need to give reference of the file on top of every file like above approach:



To use inbuilt snippet use that name and press tab.

We can use our custom snippet by choosing 'user snippet' from command prompt.

```

javascript.json
1 {
2   "Angular Controller": {
3     "prefix": "ngcontroller",
4     "body": [
5       "(function() {",
6       "'use strict';",
7       "",
8       "\tangular",
9       "\t\t.module('${Module}').",
10      "\t\t.controller('${Controller}Controller', ${Controller}
      Controller):".

```

Angular style guide and various snippet references:

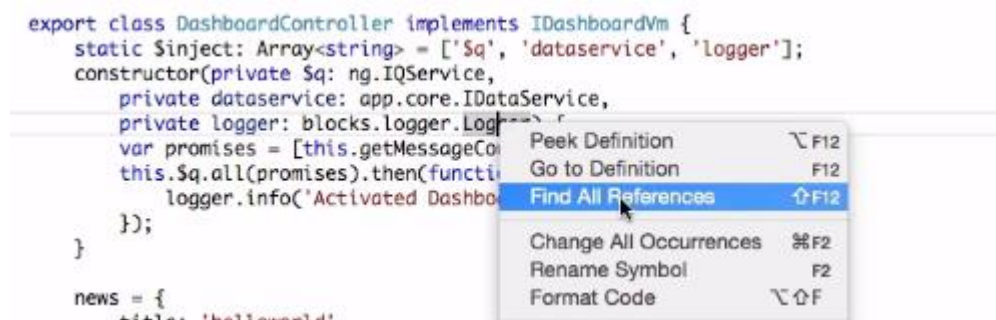
<https://github.com/johnpapa/angular-styleguide>

Practice – Create custom code snippet

Use F12 for go to definition of variable and command + F12 for a peek. Use shift + F12 to see all the references/uses of that variable, even we can interact in peek window by writing something in it.

To go back to previous file use command + P key twice.

Even we can use right click of the mouse:



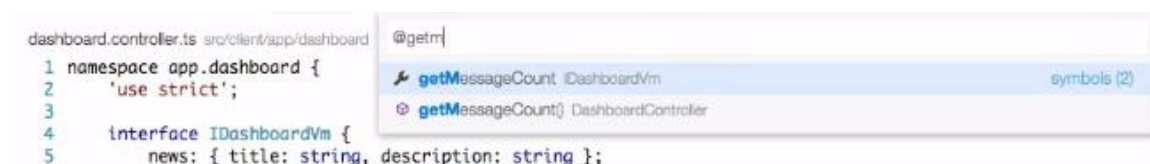
```

export class DashboardController implements IDashboardVm {
  static $inject: Array<string> = ['$q', 'dataservice', 'logger'];
  constructor(private $q: ng.IQService,
    private dataservice: app.core.IDataService,
    private logger: blocks.logger.Logger) {
    var promises = [this.getMessageCount()];
    this.$q.all(promises).then(function() {
      logger.info('Activated DashboardController');
    });
  }
}

news = {
  title: 'Dashboard',
  description: 'Dashboard description'
};

```

Use command + shift + o, to find the symbols in the project file, we can use ‘.’ after o to categorize them:



```

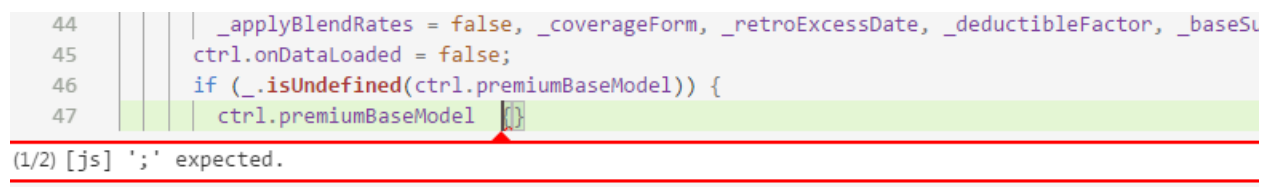
dashboard.controller.ts src/client/app/dashboard
1 namespace app.dashboard {
2   'use strict';
3
4   interface IDashboardVm {
5     news: { title: string, description: string };
6   }
7 }

```

To rename a variable use F2, it will replace in all the reference.

Use ‘Format code’ command to format the code of the file or shift + command + F.

Press F8 to see errors in more details or command + shift + m:



```

44 | _applyBlendRates = false, _coverageForm, _retroExcessDate, _deductibleFactor, _baseSu
45 | ctrl.onDataLoaded = false;
46 | if (._isUndefined(ctrl.premiumBaseModel)) {
47 |   ctrl.premiumBaseModel

```

(1/2) [js] ';' expected.

Use alt + arrow keys to move the line up and down and alt + shift + arrow to copy the lines up and down.

Themes, Preferences, and Keyboard Shortcuts

To change the user preferences use keyboard shortcut `ctrl + ,`



Practice: implement setting as per video

We can have Default setting, User setting and Workspace setting. Workspace setting can be checked-in into SVN and shared among the team. It will override the user preferences.

Key bindings

<code>⇧⌘F</code>	Format Code
<code>⇧F12</code>	Peek
<code>⇧F12</code>	Show all References
<code>F12</code>	Go to Definition
<code>⌘P</code>	Go to File
<code>⌘1</code>	Focus 1 st Pane (or 2 nd or 3 rd)
<code>⇧⌘F</code>	Full Screen

Practice: create custom key bindings

```
{ "key": "ctrl+shift+l",  
  "command": "workbench.action.changeToLightTheme"}
```

Language Features

While finding something into CSS use symbols on command prompt, use Emmet feature as well. For cross browser prefix use `!-` then name of style like transition then tab key, it will apply it for all the browser vendors.

In less file we can find the references for variables.

To get a preview of ReadMe.md file, which is a markdown file, use `ctrl + shift + v`. we can preview it with live typing.

We can set various setting rules for JavaScript in setting.json file with Lint:

```

settings.json Users\john\Library\Application Support\Code\User
31 "files.exclude": {
32   "**/.git": true,
33   "**/.DS_Store": true
34   /**/*.js": { "when": "$(basename).ts" } // hides js files when ts exist
35 },
36
37 "javascript.validate.lint.comparisonOperatorsNotStrict": "warning",
38 "javascript.validate.lint.curlyBracketsMustNotBeOmitted": "error",
39 "javascript.validate.lint.functionsInsideLoops": "warning",
40 "javascript.validate.lint.missingSemicolon": "warning",
41 "javascript.validate.lint.redeclaredVariables": "warning",
42 "javascript.validate.lint.undeclaredVariables": "warning",
43 "javascript.validate.lint.unusedFunctions": "warning",
44 "javascript.validate.lint.unusedVariables": "warning",
45

```

Practice – use angular.d.ts typing files for getting intellisense

To get these typing files use tsd.json file and run npm command as below

npm install tsd -g

tsd query -r -o -a install angularjs --save

In command prompt type Tasks: Run Task to see all available task commands. Like grunt serve-local

Re-watch - <https://app.pluralsight.com/player?course=visual-studio-code&author=john-papa&name=visual-studio-code-m6&clip=9&mode=live>

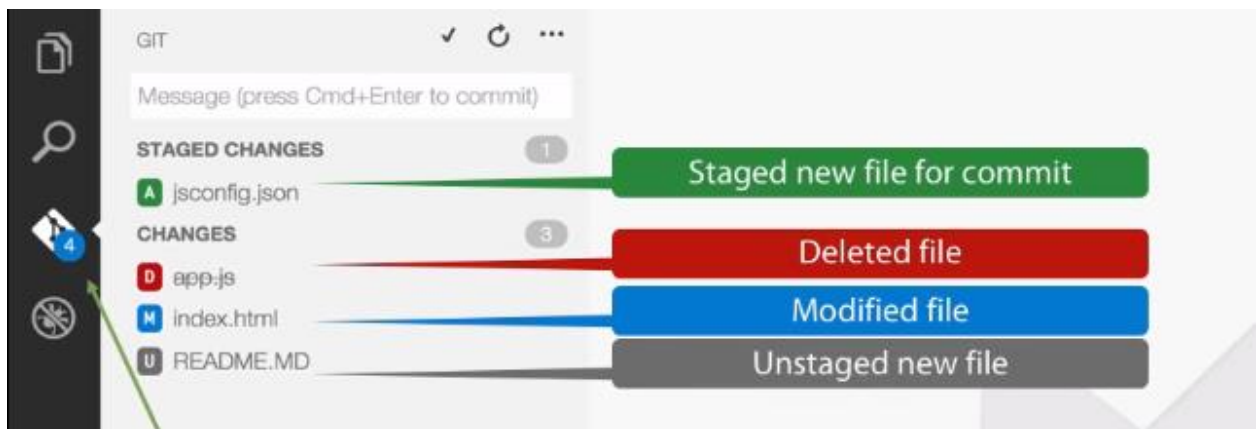
Version Control and Git

Install Git using <http://git-scm.com/download>

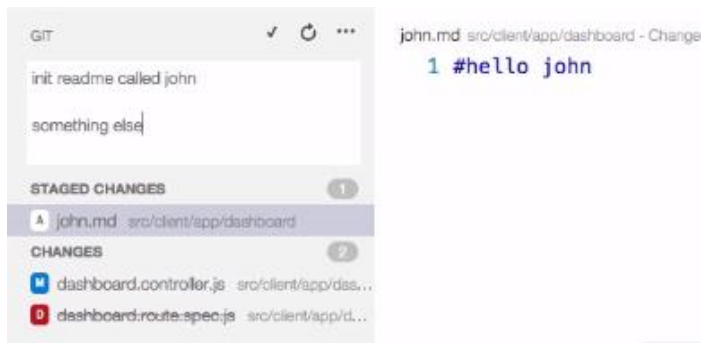
By this we can connect to local repo or remote repo like Github. Remote online provider that we can use the Git is like Github, VS Online, Bitbucket

Visual Studio Code Git's feature

Git viewlet



Use Command + Enter to commit the changes:



We can see red, blue, green lines as gutter indicators for code changes.

Practice: committing files from Git

Initialize Git repository and creating .gitignore file.

Installing http-server using NPM:

`npm install http-server -g`

Note: if we write `npm list -g --depth=0`, it will give us a list for all of the components that NPM's installed globally.

```
$ npm list -g --depth=0
/Users/john/.npm-packages/lib
├─ babel@5.4.7
├─ bower@1.4.1
├─ generator-aspnet@0.0.50
├─ generator-hottowel@0.3.6 -> /Users/john/_git/generator-hottowel
├─ grunt-cli@0.1.13
├─ gulp@3.9.0
├─ gulp-devtools@0.0.3
├─ http-server@0.8.0
└─ jsdoc@3.3.0
```

On Git you should generally have a readme file as well. When you do init from Git viewlet it added to your Git only, for public Github repository, to do this we need to use below command using our account to connect with the repository remotely and making a push:

...or push an existing repository from the command line

```
git remote add origin git@github.com:johnpapa/newidea.git
git push -u origin master
```

After connecting we can use below commands:

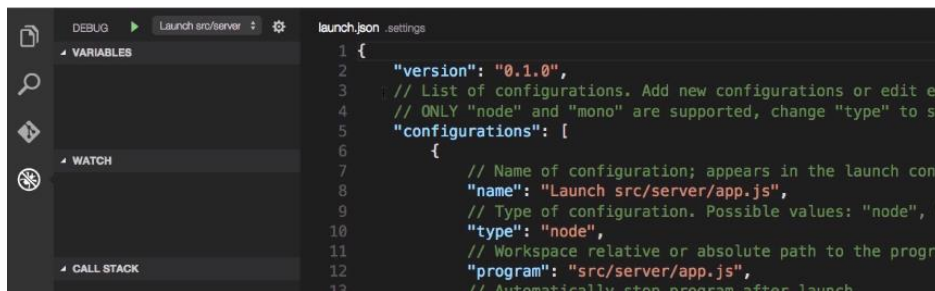


Git Credentials

<http://jpapa.me/gitcredhelper>

Debugging

Using debugger viewlet, we need to use launch.json to configure the debugging setups like below:



When we are running a node process and we want to see the first couple of lines that are running use 'stopOnEntry' setting configuration.

Debugging key bindings

F5	Run / Pause / Run to Cursor
⇧ F5	Stop
F10	Step Over
F11	Step Into
⇧ F11	Step Out
F9	Toggle Breakpoint
↺	Restart

Attaching the debugger to a process

```
{
  "name": "Attach",
  "type": "node",
  // TCP/IP address. Default is "localhost".
  "address": "localhost",
  // Port to attach to.
  "port": 5858,
  "sourceMaps": false
}
```

Practice – attaching debugger to a process

TypeScript debugging and source maps

Re-watch - <https://app.pluralsight.com/player?course=visual-studio-code&author=john-papa&name=visual-studio-code-m8&clip=8&mode=live>

To debug the TypeScript code we need to set sourceMaps as true.

Use VS code to debug server side code, and browser to debug client side code.

Integrating Code with External Tools via Tasks

Creating a Babel transpiling task – from ES6 to JavaScript by creating tasks.json file

tasks.json :settings

```
1 {
2   "version": "0.1.0",
3   "command": "${workspaceRoot}/node_modules/.bin/babel",
4   "isShellCommand": true,
5   "tasks": [
6     {
7       "args": ["js", "--out-dir", "dist", "--source_maps"],
8       "taskName": "babel",
9       "isBuildCommand": true
```

Watching with Babel

```
{
  "args": ["js", "--out-dir", "dist", "-w", "--source_maps"],
  "taskName": "babel-watch",
  "isBuildCommand": false
}
```

To kill a task we can use 'terminate running task' command from command prompt.

Configure Transpiling TypeScript task

tasks.json :settings

```
1 {
2   "version": "0.1.0",
3
4   // The command is tsc. Assumes that tsc has been installed using npm install -g typescript
5   "command": "tsc",
6
7   // The command is a shell script
8   "isShellCommand": true,
9
10  // Show the output window only if unrecognized errors occur.
11  "showOutput": "silent",
12
13  // args is the HelloWorld program to compile.
14  "args": ["-p", "src/client", "-w"],
15
16  // use the standard tsc problem matcher to find compile problems
17  // in the output.
18  "problemMatcher": "$tsc"
19 }
```

Problem matchers – it grabs any problems that compiler notifies using regular expressions and pull those errors and warnings in vs code.

```
// use the standard tsc problem matcher to find compile problems
// in the output.
"problemMatcher": "$tsc" }
```


Processing Task Output with Problem Matchers

VS Code processes the output from a task with a problem matcher and we ship with a number of them 'in the box', we'll talk about how to make your own ones soon:

- TypeScript: `$tsc` assumes that file names in the output are relative to the opened folder.
- JSHint: `$jshint` assumes that file names are reported as an absolute path.
- JSHint Stylish: `$jshint-stylish` assumes that file names are reported as an absolute path.
- ESLint Compact: `$eslint-compact` assumes that file names in the output are relative to the opened folder.
- ESLint Stylish: `$eslint-stylish` assumes that file names in the output are relative to the opened folder.
- CSharp and VB Compiler: `$mscompile` assumes that file names are reported as an absolute path.
- Less: `$lessCompile` assumes that file names are reported as absolute path.

Automatic Gulp Detection – it will detect tasks from the Gulp file by scanning without task.json.

Custom Tasks with Gulp using problem matchers – we can create new task using an existing task

```
"tasks": [
  {
    "taskName": "vet",
    "args": [],
    "problemMatcher": [
      "$jshint",
      "$jshint-stylish"
    ]
  },
],
```

```
"tasks": [
  {
    "taskName": "vet2",
    "args": ["vet"],
    "suppressTaskName": true,
    "problemMatcher": [
      "$jshint",
      "$jshint-stylish"
    ]
  },
],
```

Now it will pull those errors in VS code like below:

```
15     };
16     vm.messageCount = 0
(1/2) Missing semicolon.
```

Automatic Grunt Detection

```
gruntfile.js
1 module.exports = function (grunt) {
2
3   grunt.initConfig({
4     jshint: {
5       files: ['gruntfile.js', '**/*.js']
6     }
7   });
8
9   grunt.loadNpmTasks('grunt-contrib-jshint');
10
11   grunt.registerTask('grunt-hint', ['jshint']);
12
13 };
```





Debugging Gulp and a Gulp Task

```

"configurations": [
  {
    "name": "node gulp debugger",
    "type": "node",
    "program": "./node_modules/gulp/bin/gulp.js",
    "stopOnEntry": true,
    "args": ["vet"],
    "runtimeArgs": ["--no-lazy"]
  },

```

Tasks commands

-  > Run Task
-  > Terminate Task
-  > Configure Task Runner
-  Toggle Output

Practice – create tasks for babel transpiling with watch

Practice – create tasks for TypeScript transpiling

Practice – create JSHint task with Grunt

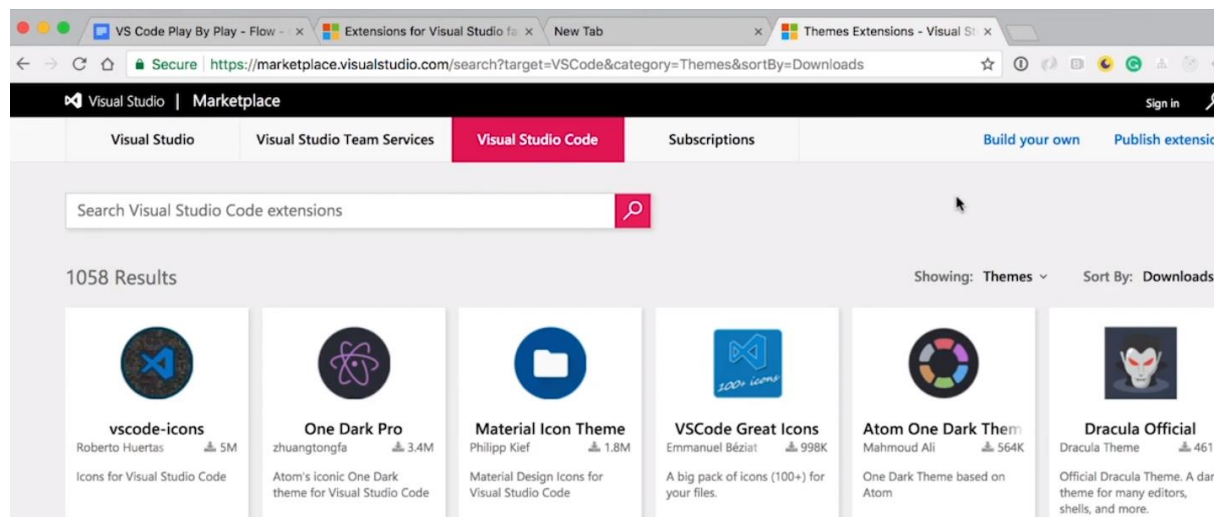
Play by Play: Visual Studio Code Can Do That

Customizing VS Code

Apply theme Hop Light and Winter Is Coming Dark, Atom One Dark, Atom One Light, Cobalt2.

Apply icon themes – Material Icons

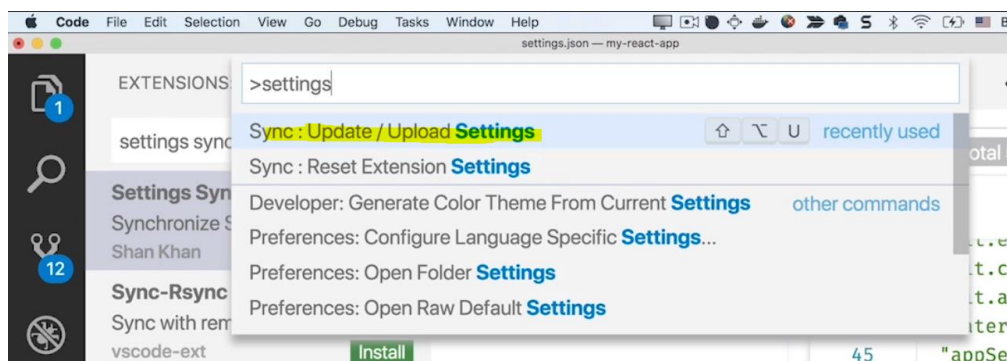
Search themes on marketplace:



Install and use Fira Code and Menlo font family and fontLigatures as true:

	USER SETTINGS	WORKSPACE SETTINGS
41	<code>git.enableSmartCommit": true,</code>	
42	<code>"git.confirmSync": false,</code>	
43	<code>"git.autofetch": true,</code>	
44	<code>"material-icon-theme.showUpdateMessage": true,</code>	
45	<code>"appService.showRemoteFiles": true,</code>	
46	<code>"editor.fontFamily": "Fira Code",</code>	
47	<code>"editor.fontLigatures": true,</code>	
48	<code>}</code>	

Synchronizing custom settings using extension Setting Sync:



OUTPUT

Code Settings Sync

CODE SETTINGS SYNC UPLOAD SUMMARY

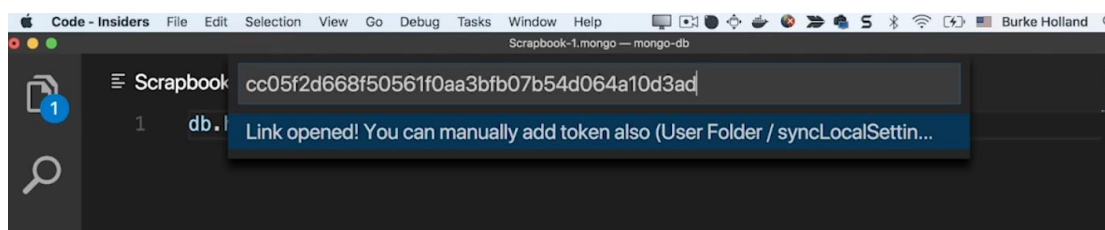
Version: 2.8.7

GitHub Token: `cc05f2d668f50561f0aa3bfb07b54d064a10d3ad`

GitHub Gist: `44a1821429048f13a7df69b39d837d77`

GitHub Gist Type: Secret

Restarting Visual Studio Code may be required to apply color and file icon theme.

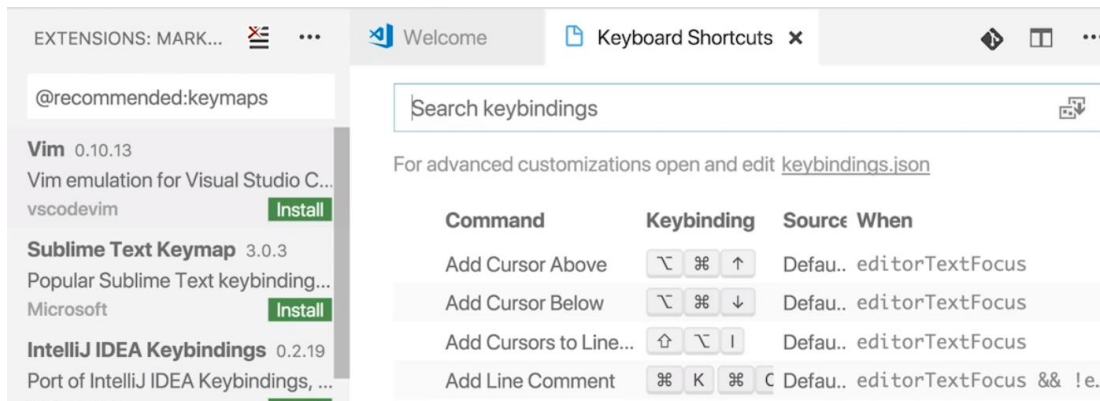
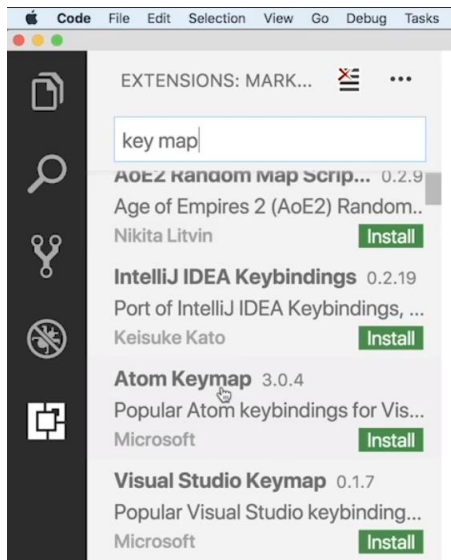


Maximizing Productivity

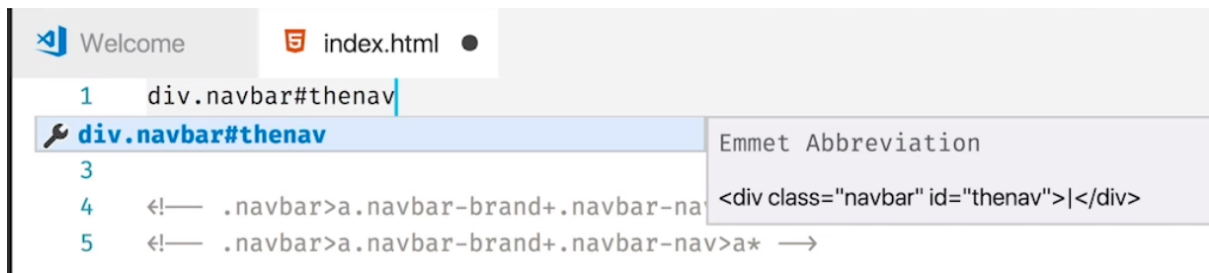
Enable autosave after a slight delay:

```
29 "files.autoSave": "afterDelay",
30 "files.autoSaveDelay": 1000,
```

Customizing keyboard shortcuts and keymaps



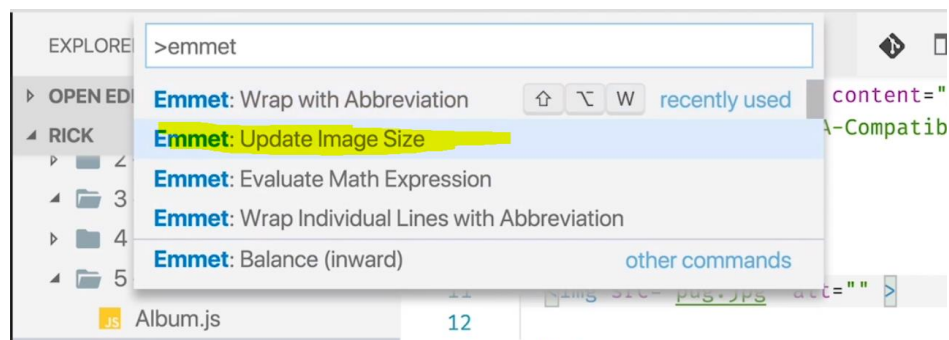
Emmet for HTML files – help in generating mark-ups and css without typing the angular brackets:



Enabling emmet for other types of web files

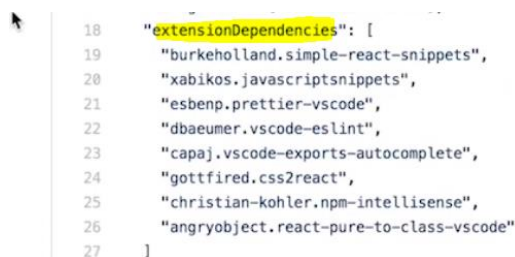


Modifying images with Emmet



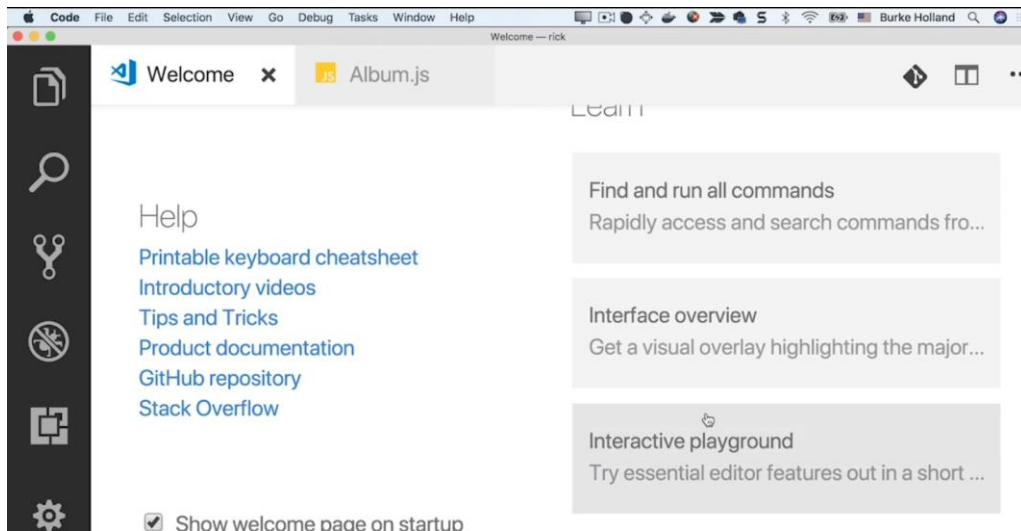
Extensions for lighting up angular – angular v5 snippets, Angular Essentials

Publish an extension pack – using package.json



Code Checking and Debugging

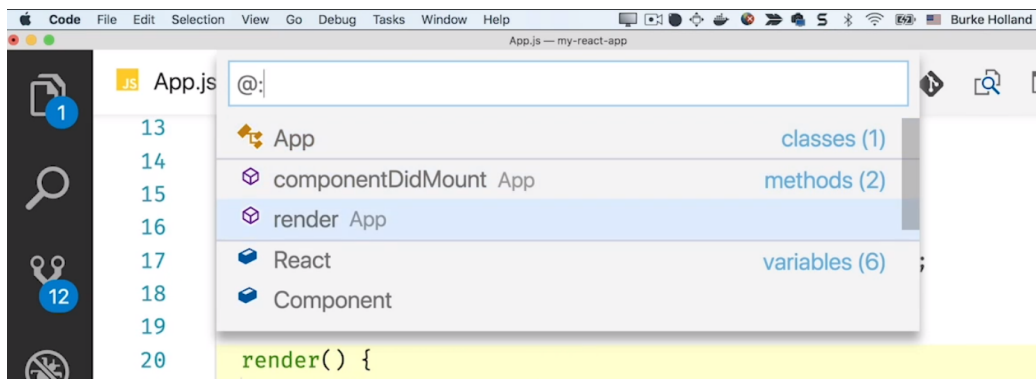
Trying out the interactive playground



Common keyboard shortcuts

Ctrl + p + p to go back previous file.

Finding references and symbols – shift + F12 after highlighting the keyword and use ctrl + shift + o for symbol:



Efficiencies with keyboard shortcuts – ctrl + j to close bottom panel.

Try full screen and zen modes.

Use option + shift + down/up arrow to copy and paste a line, also use option + down / up arrow to move that line up and down.

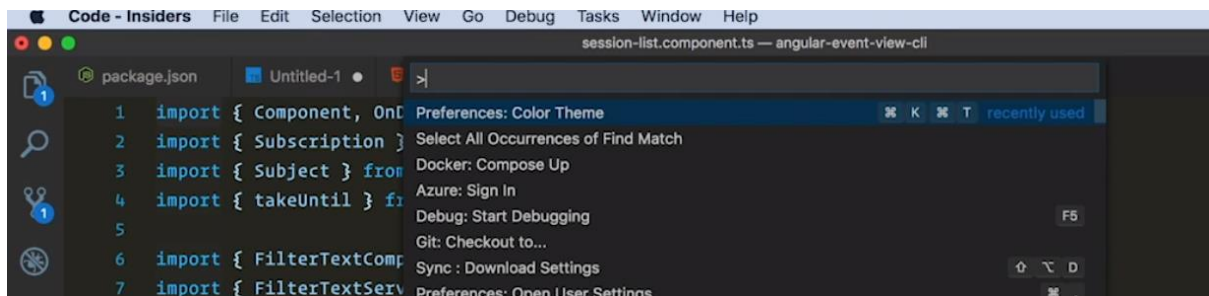
```

14   componentDidMount() {
15     fetch('/api/message')
16       .then(response => response.json())
17       .then(json => this.setState({ message: json }));
18       .then(json => this.setState({ message: json }));

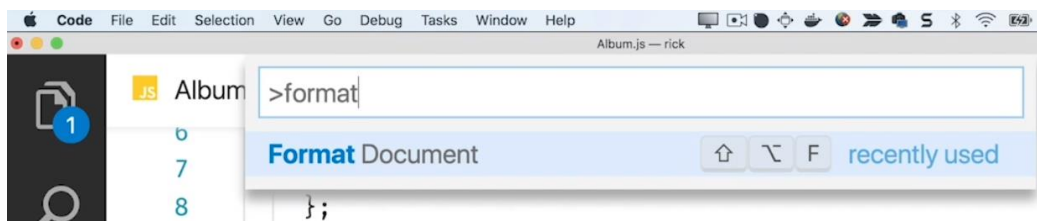
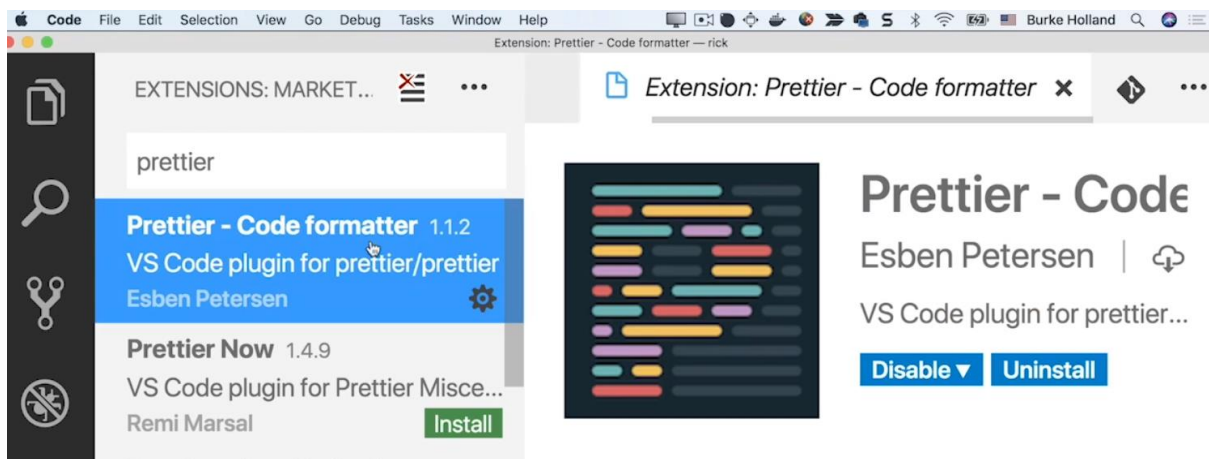
```

Multi cursor – command + d or use option 'select all occurrences of find match'

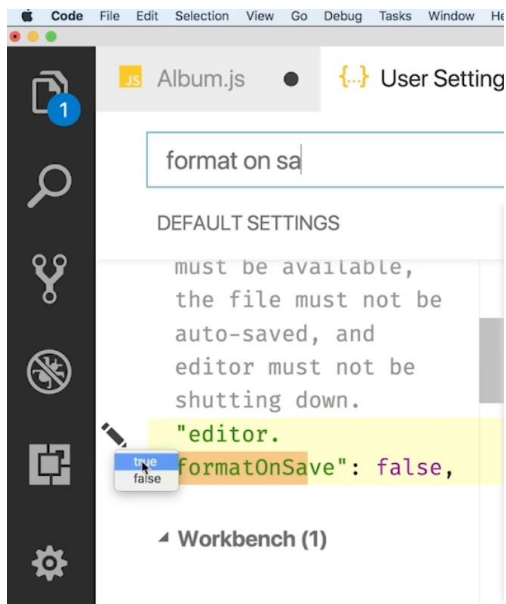

```
package.json  Untitled-1 •
1  "@angular-devkit/core": "^0.2.0",
2  "@angular/animations": "^5.0.0-rc.6",
3  "@angular/common": "^5.0.0-rc.6",
4  "@angular/compiler": "^5.0.0-rc.6",
5  "@angular/core": "^5.0.0-rc.6",
6  "@angular/forms": "^5.0.0-rc.6",
7  "@angular/http": "^5.0.0-rc.6",
8  "@angular/platform-browser": "^5.0.0-rc.6",
9  "@angular/platform-browser-dynamic": "^5.0.0-rc.6",
10 "@angular/platform-server": "^5.0.0-rc.6",
11 "@angular/router": "^5.0.0-rc.6",
12
```



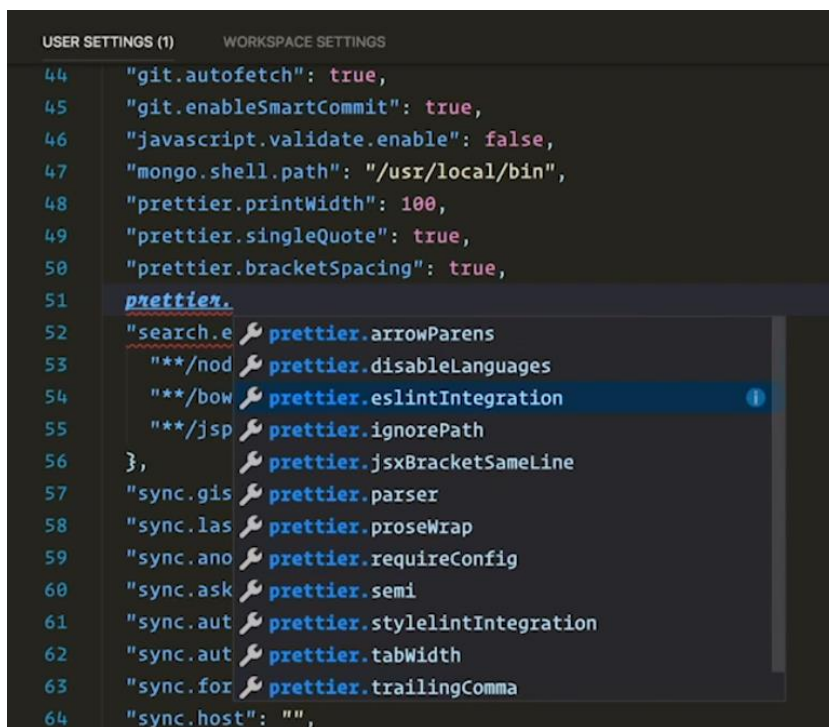
Formatting code with the prettier extension



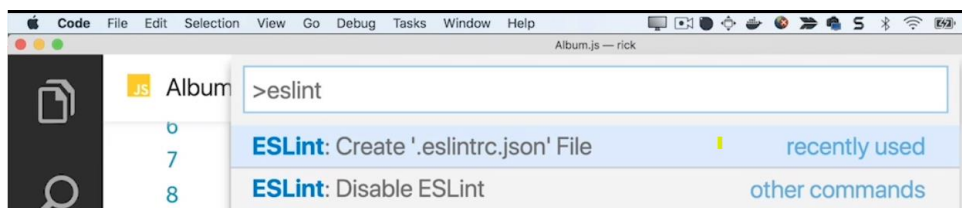
Turn formatOnSave as true:



Use its setting tabSize, singleQuote, bracketSpacing and line breaking number, use prettier.rc file to store them:



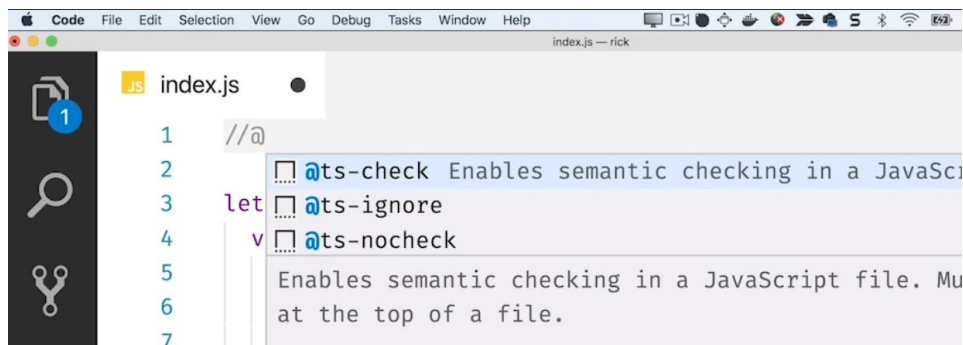
Verify code correctness with ESLint



Use prettier setting eslintIntegration as true to avoid collision of rules with ESLint file.

Containers and Deployment

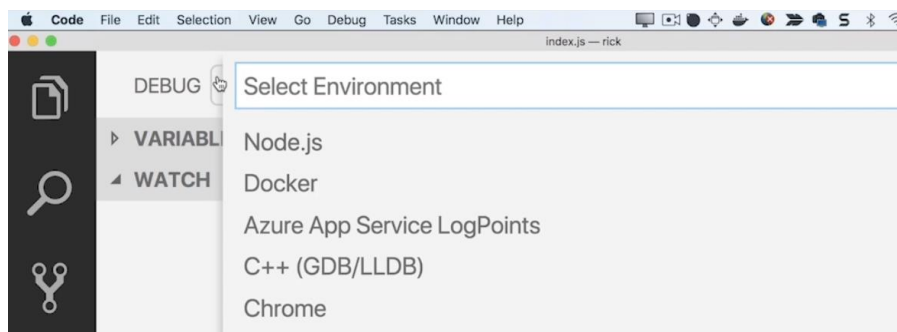
Checking JavaScript with typescript compiler



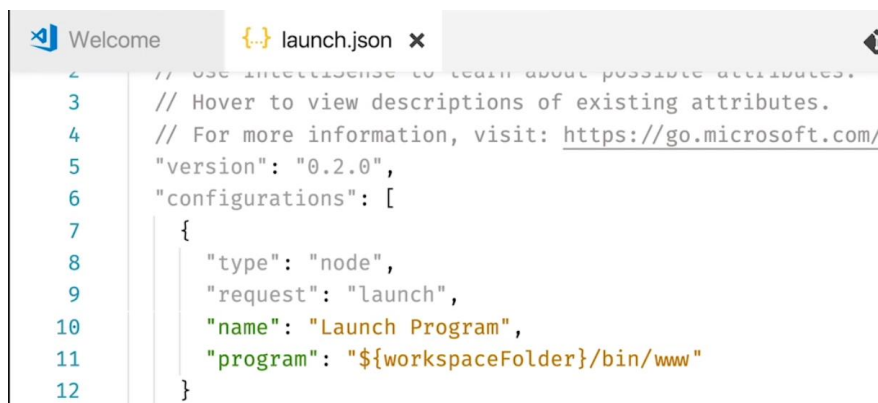
Use F8 to toggle through the errors in the file.

Use 'ctrl + .' to fix suggestion suggested by yellow light bulb.

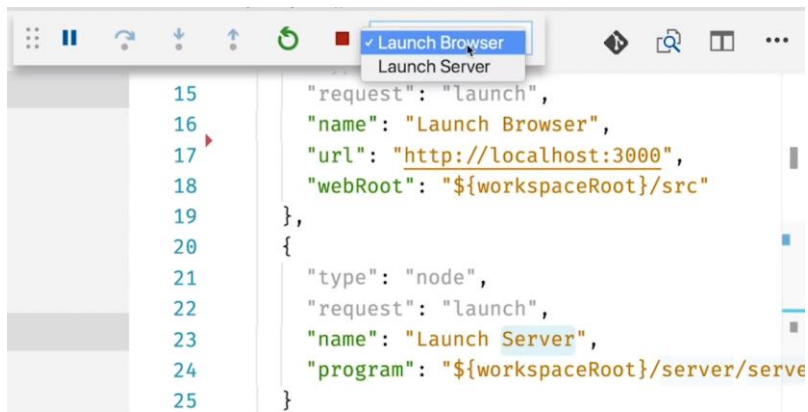
Debugging and launch configuration: just click on the play button



For debug configuration use launch.json file:

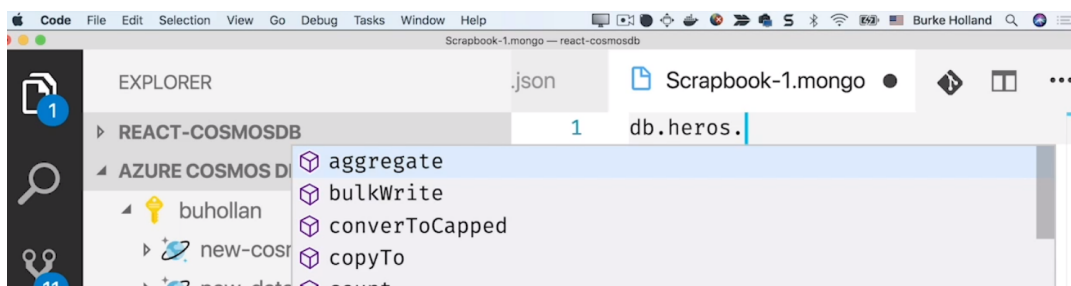
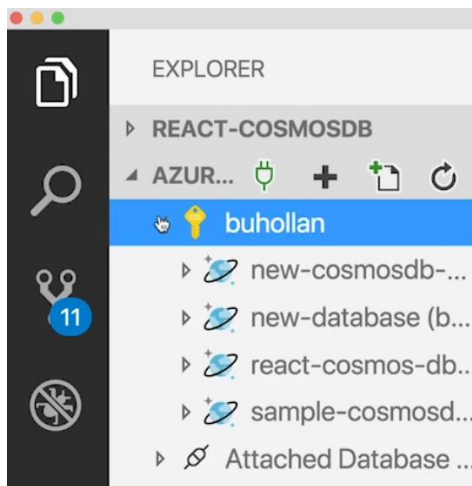


We can also debug browser and server portion at the same time by using combined configuration in launch.json file:



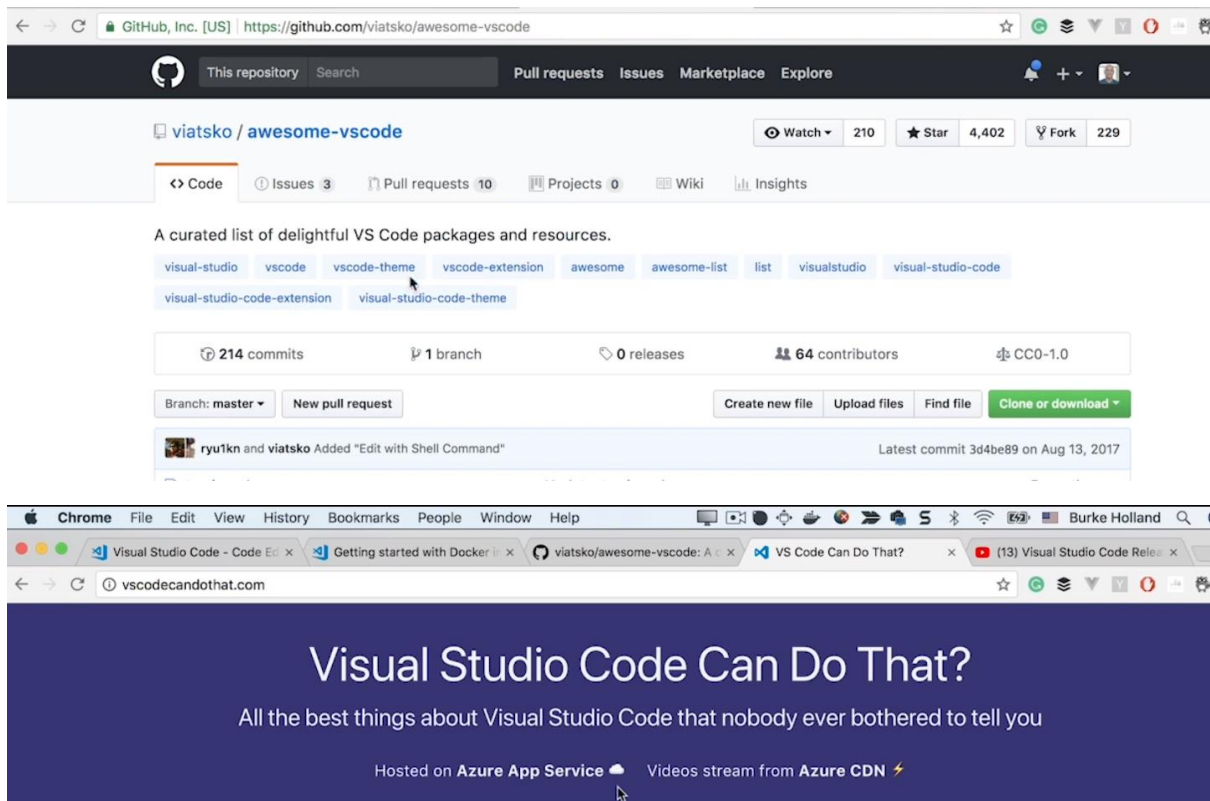
```
15  "request": "launch",
16  "name": "Launch Browser",
17  "url": "http://localhost:3000",
18  "webRoot": "${workspaceRoot}/src"
19  },
20  {
21  "type": "node",
22  "request": "launch",
23  "name": "Launch Server",
24  "program": "${workspaceRoot}/server/serve
25  }
```

Database extension – MongoDB and Azure CosmosDB



Next Steps

References



Hacking VS Code: Write Your First Extension for Visual Studio Code

Introduction

Electron is a native shell for windows, mac, and Linux that hosts JavaScript-based app like Monaco. Visual studio code is a combination of Monaco and Electron.

Starting Simple

While using VS code in presentation mode, use window.zoomLevel setting. We can set two type of setting User settings or Workspace settings.

Practice – Apply best VS code settings on IDE

Creating User Snippets - it consist as a JSON object with 3 properties – prefix, body, description.

```
"Print to console if variable exists": {
  "prefix": "logIf",
  "body": [
    "if (${1:Variable Name}) {",
    "  console.log(\"Logging response for: $1\");",
    "} else {",
    "  console.log(\"Unable to log because $1 is null\");",
    "}",
    "$2"
  ],
  "description": "Log output to console if variable exists"
}
```

Installing snippet files from Market Place



Practice – creating custom user snippet

Practice – installing best snippets from market place

Syncing settings – install visual studio code settings sync and generate a Gist token. By this we can share our settings, key bindings, snippets and extensions across all the places.

Practice – sync visual studio settings

Creating a Hello World Extension

For this we need node.js and NPM, Yeoman, Yo Code tools. After firing command “yo code” choose “New Extension (JavaScript)” option from console and other details. It will generate the skeleton of files which are required to create extension. It needs vscode module.

Note: if a folder name start with a ‘.’ It means the folder is hidden.

We use activate and deactivate methods in extension.js file to write code for the extension.

```
var editor = vscode.window.activeTextEditor;

if(!editor) {
    vscode.window.showErrorMessage("No file is open, can't say Hello");
    return;
}
```

```
editor.edit(function(editBuilder){
    editBuilder.delete(editor.selection);
}).then(function(editBuilder){
    editor.edit(function(editBuilder){
        editBuilder.insert(editor.selection.start, "Hello World!");
    });
});
```

Practice – creating HelloWorld extension

Building a Real World Extension

Setting key bindings for keyboard shortcuts:

```
"keybindings": [{
    "command": "extension.insertLink",
    "key": "shift+ctrl+l",
    "mac": "shift+cmd+l",
    "when": "editorTextFocus"
}],{
```



```
let updateTemplateWithDate = (template) => {
  let today = new Date();
  let year = today.getFullYear();
  let month = ('0' + (today.getMonth() + 1)).slice(-2);

  template = template.replace("${year}", year);
  template = template.replace("${month}", month);

  return template;
};
```

Writing Unit Tests for extensions

```
// Defines a Mocha test suite to group tests of similar kind together
suite("Extension Tests", function() {

  // Defines a Mocha unit test
  test("Something 1", function() {
    assert.equal(-1, [1, 2, 3].indexOf(5));
    assert.equal(-1, [1, 2, 3].indexOf(0));
  });
});
```

```
test("Update Template with Date", () => {
  let template = "/images/${year}/${month}";

  let updatedTemplate = myExtension.updateTemplateWithDate(template);

  assert.notEqual(updatedTemplate, null, "Updated Template was Null");
});
```

Installing our extension locally – copy your extension to the VS code extension directory.

Your Extensions Folder

VS Code looks for extensions under your extensions folder `.vscode/extensions`. Depending on your platform it is located:

- Windows `%USERPROFILE%\vscode\extensions`
- Mac `$HOME/.vscode/extensions`
- Linux `$HOME/.vscode/extensions`

Distributing Our Extension

To distribute it on VS code market place first we need to download publishing tool, then update README.md file, add links for resources, then set gallery banner color, Add icon and then publish.

Create an account at visualstudio.com, generate a personal access token, use NPM to download the VSCE publishing tool, and then create a publisher account with VSCE.

```
"publisher": "jeffaTech",
"repository": {
  "type": "git",
  "url": "https://github.com/jeffa00/static-site-hero.git"
},
"bugs": {
  "url": "https://github.com/jeffa00/static-site-hero/issues"
},
"homepage": "https://github.com/jeffa00/static-site-hero/blob/master/README.md",
"icon": "images/StaticSiteHeroLogo.png",
```

Practice – publish an extension on VS code market place