КАК СОЗДАТЬ РАСШИРЕНИЕ VISUAL STUDIO CODE

11 сентября, 2021 12:00 пп 2 540 views | Комментариев нет

Development | Amber | Комментировать запись

Visual Studio Code – это редактор кода от Microsoft, доступный для систем Windows, Linux и macOS. Для внедрения дополнительных функций он предлагает готовые расширения, которые вы можете установить через Visual Studio Code Marketplace. Но если вы не можете найти расширение, которое делает именно то, что вам нужно, вы можете создать необходимое расширение самостоятельно.

В этом руководстве вы узнаете, как создать свое первое расширение Visual Studio Code.

Требования

Для выполнения этого урока нужно:

- Загрузить и установить последнюю версию Visual Studio Code.
- Установить Node.js. Инструкции по установке зависят от дистрибутива: Mac OS, Ubuntu, CentOS, Debian.

Это руководство было проверено на версиях Node v14.4.0, npm v6.14.5, yo v3.1.1 и generator-code v1.2.16.

1: Установка инструментов

Команда Visual Studio Code разработала специальный генератор для создания расширений. Он генерирует все необходимые стартовые файлы, чтобы вы легко могли начать создание вашего расширения.

Рубрики

Arch Linux
Centos
Cloud Server
Debian
Development
FreeBSD
FTP
General
Java
LAMP Stack
LEMP Stack
Linux
MariaDB
mySQL
PHP
Python
Quickstart
RHEL
Ruby
SSH
Turnkey
Ubuntu
VPS
Без категорий

Поиск по меткам

Ansible Apache Ce
CentOS 6 CentOS 7 C
Server CSS Debian
Debian 10 Debian 8 Debian 9
Django DNS Docker Git Go
HTML Java Javascript
LAMP stack Let's Encrypt

- yo – интерфейс командной строки для Yeoman.
- generator-code – генератор Yeoman для написания расширений Visual Studio Code.

Используйте встроенный терминал Visual Studio Code, чтобы при помощи npx запустить локальные копии yo и generator-code, а затем введите команду yo code для инициализации вашего нового проекта:

```
npx -p yo -p generator-code yo code
```

После этого Yeoman запустит генератор кода.

2: Создание расширения

Теперь вы готовы начать разработку вашего первого расширения. На вашем экране вы увидите сообщение:

```
Welcome to the Visual Studio Code Extension Generator!
```

Сейчас вам нужно будет ответить на несколько вопросов о проекте: указать, какое расширение вы создаете, а также выбрать между TypeScript и JavaScript. В этом уроке мы выберем JavaScript.

Затем вам будет предложено еще несколько вопросов. В этом мануале мы выбрали следующие ответы:

```
? What type of extension do you want to create? New Extension (JavaScript)
? What's the name of your extension? testytest
? What's the identifier of your extension? testytest
? What's the description of your extension? This is a test extension
? Enable JavaScript type checking in 'jsconfig.json'? Yes
? Initialize a git repository? Yes
? Which package manager to use? npm
```

После завершения этого процесса вы получите все файлы, необходимые для начала работы. Два самых важных файла:

Откройте package.json и взгляните на его содержимое. Вы увидите название, описание проекта и т.п. В нем есть два очень важных раздела.

- activationEvents – это список событий, которые активируют ваше расширение. Расширения загружаются по ленивой загрузке, поэтому они не активируются, пока не произойдет одно из перечисленных событий.
- commands – список команд, которые пользователи смогут запускать через ваше расширение.

Мы вернемся к ним в ближайшее время.

```
{
  // ...
  "activationEvents": [
    "onCommand:testytest.helloWorld"
  ],
  "main": "./extension.js",
  "contributes": {
    "commands": [
      {
        "command": "testytest.helloWorld",
        "title": "Hello World"
      }
    ]
  },
  // ...
}
```

Вы также можете просмотреть файл extension.js. В нем мы напишем код для нашего расширения. Здесь уже есть шаблонный код, давайте разберемся с ним.

В выделенной ниже строке мы регистрируем в VS Code нашу команду. Обратите внимание, что имя helloWorld совпадает с именем команды в package.json. Это не случайно. Пакет package.json определяет, какие команды доступны пользователю, но файл extension.js регистрирует код для этой команды.

```
 */
function activate(context) {
  console.log('Congratulations, your extension "testytest" is now active!');

  let disposable = vscode.commands.registerCommand('testytest.helloWorld', function (
    vscode.window.showInformationMessage('Hello World from testytest!');
  });

  context.subscriptions.push(disposable);
}

// ...
```

В этом примере наша команда будет только отображать на экране пользователя сообщение «Hello World».

3: Отладка расширения

Теперь, когда все необходимые файлы установлены, мы можем запустить наше расширение.

Папка .vscode – это место, где VS Code хранит конфигурационные файлы проекта. В нашем случае он включает файл launch.json, содержащий конфигурации отладки.

```
// ...
{
  // ...
  "configurations": [
    {
      "name": "Run Extension",
      "type": "extensionHost",
      "request": "launch",
      "runtimeExecutable": "${execPath}",
      "args": [
        "--extensionDevelopmentPath=${workspaceFolder}"
      ]
    },
```

В этом файле проводится отладка расширения. Откройте вкладку debug в левой части экрана, а затем кликните на плей.

Это откроет новый (отладочный) экземпляр VS Code.

Открыв его, вы можете развернуть палитру команд (с помощью Command + Shift + P на Mac или Ctrl + Shift + P в Windows) и запустить Hello World.

Вы увидите всплывающее сообщение «Hello World» в правом нижнем углу.

4: Редактирование расширения

Прежде чем приступить к работе над кодом, давайте еще раз взглянем на раздел activateEvents в файле package.json. Как вы уже знаете, этот раздел содержит список событий, которые активируют наше расширение всякий раз, когда происходят. По умолчанию расширение активируется при запуске нашей команды.

Теоретически это событие может быть любым (что определяется символом *). Если установить для события активации значение *, то ваше расширение будет загружено при запуске VS Code.

```
{
  // ...
  "activationEvents": [
    "*"
  ],
  // ...
}
```

> **Примечание:** Этого делать не нужно, это просто комментарий.

Итак, у нас есть необходимые файлы и мы знаем, как их отлаживать. Приступим же к созданию нашего расширения. Предположим, мы хотим,

Сначала давайте обновим название нашей команды. Откройте extension.js и обновите имя команды с extension.helloworld на extension.createBoilerplate.

```
// ...

/**
 * @param {vscode.ExtensionContext} context
 */
function activate(context) {
  console.log('Congratulations, your extension "testytest" is now active!');

  let disposable = vscode.commands.registerCommand('testytest.createBoilerplate', fur
    vscode.window.showInformationMessage('Hello World from testytest!');
  });

  context.subscriptions.push(disposable);
}

// ...
```

Соответствующим образом обновите package.json:

```
{
  // ...
  "activationEvents": [
    "onCommand:testytest.createBoilerplate"
  ],
  "main": "./extension.js",
  "contributes": {
    "commands": [
      {
        "command": "testytest.createBoilerplate",
        "title": "Create Boilerplate"
      }
    ]
  },
  // ...
}
```

```
const fs = require('fs');
const path = require('path');
```

Также нам нужно получить путь к текущей папке. Внутри раздела command поместите следующий фрагмент:

```
if (!vscode.workspace) {
  return vscode.window.showErrorMessage('Please open a project folder first');
}

const folderPath = vscode.workspace.workspaceFolders[0].uri
  .toString()
  .split(':')[1];
```

Нам также нужно присвоить шаблонный HTML-код переменной в файле extension.js, чтобы он автоматически записывался в файл. Вот этот шаблонный HTML:

```
const htmlContent = `<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>Document</title>
  <link rel="stylesheet" href="app.css" />
</head>
<body>
  <script src="app.js"></script>
</body>
</html>`;
```

Теперь нужно вызвать функцию writeFile модуля файловой системы и передать ее в пути к папке и HTML-коде.

Обратите внимание, мы используем модуль path, чтобы объединить путь к папке с именем файла, который мы хотим создать. Если внутри обратного вызова есть ошибка, мы отображаем ее пользователю. В

```
fs.writeFile(path.join(folderPath, 'index.html'), htmlContent, (err) => {
  if (err) {
    return vscode.window.showErrorMessage('Failed to create boilerplate file!');
  }
  vscode.window.showInformationMessage('Created boilerplate files');
});
```

Вот как выглядит полный код extension.js:

```
//...
  let disposable = vscode.commands.registerCommand(
    'testytest.createBoilerplate', async function () {
    // The code you place here will be executed every time your command is executed

    if (!vscode.workspace) {
      return vscode.window.showErrorMessage('Please open a project folder first');
    }

    const folderPath = vscode.workspace.workspaceFolders[0].uri
      .toString()
      .split(':')[1];

    const htmlContent = `<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>Document</title>
  <link rel="stylesheet" href="app.css" />
</head>
<body>
  <script src="app.js"></script>
</body>
</html>`;

    fs.writeFile(path.join(folderPath, 'index.html'), htmlContent, (err) => {
      if (err) {
        return vscode.window.showErrorMessage(
          'Failed to create boilerplate file!'
```

```
    // ...
  }
// ...
```

Попробуйте выполнить отладку вашего нового расширения. Откройте палитру команд и запустите Create Boilerplate (помните, мы изменили имя).

После выполнения команды вы увидите только что созданный файл index.html и сообщение, которое уведомляет вас об этом:

```
Created boilerplate files.
```

Заключение

Чтобы узнать больше о том, какие API можно использовать и как именно их использовать, прочтите документацию по API от Visual Studio.

Tags: Node.js, Visual Studio Code

Добавить комментарий

Для отправки комментария вам необходимо авторизоваться.

Тарифы

Доступность

Помощь

Техподдержка

Информаторий

Способы оплаты

Отзывы

Информация

О компании

Контакты