动手实验：使用 Jasmine 测试 React

预计所需时间: 45 分钟

在这个实验中，您将学习如何使用 Jasmine 测试 React 中的逻辑。

实验开始于一个 创建 React 应用程序，然后添加逻辑和相关测试。

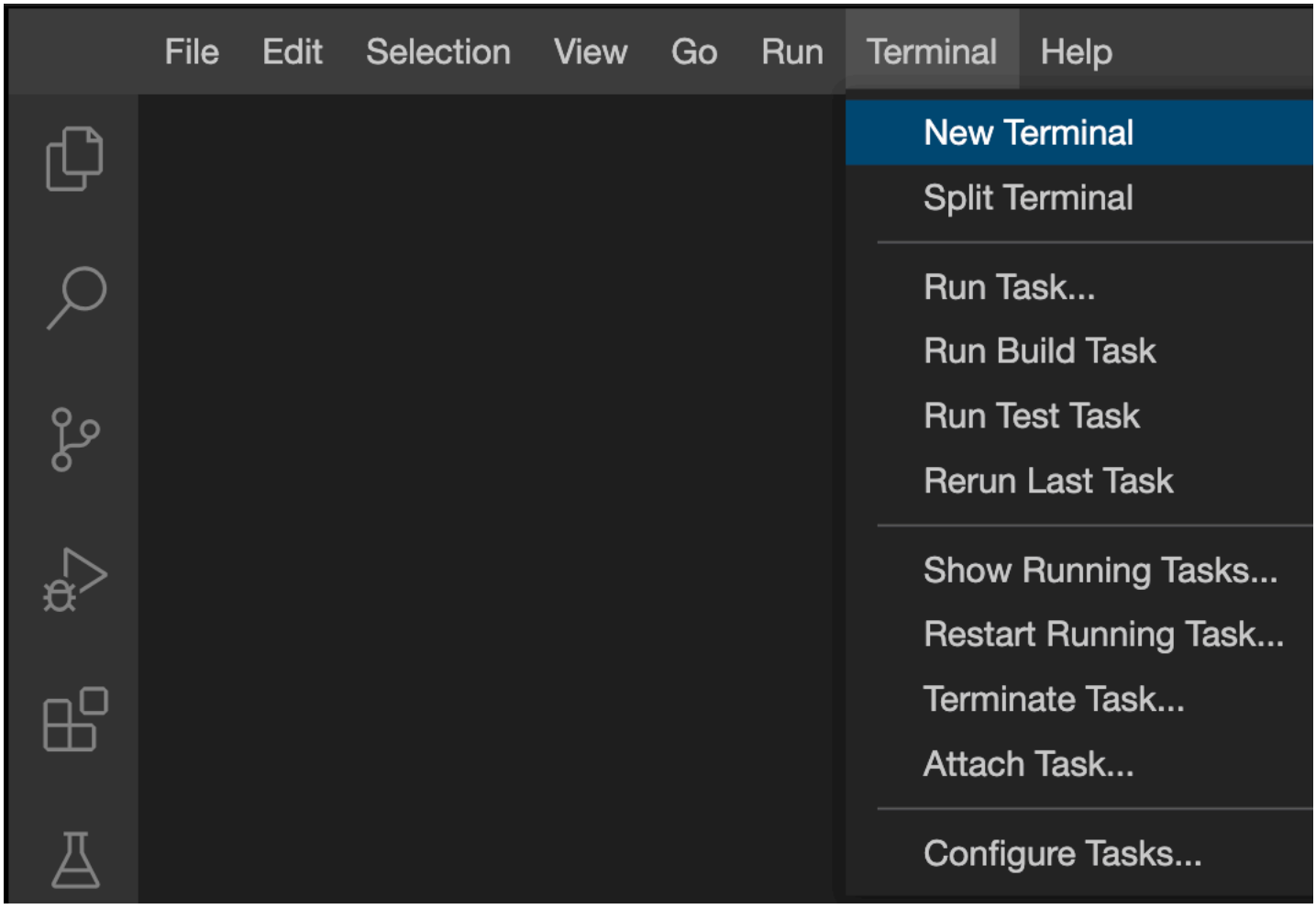
目标：

完成此实验后，您将能够：

- 使用 Jasmine 编写单元测试
- 使用命令行运行测试套件

设置：获取源代码

1. 通过编辑器中的菜单打开终端窗口：**终端 > 新建终端**。



2. 如果您当前不在项目文件夹中，请复制并粘贴以下代码以切换到您的项目文件夹。

   ```
   cd /home/project
   ```

3. 如果 Git 仓库尚不存在，请运行以下命令以克隆包含本实验所需的起始代码的 Git 仓库。

   ```
   [ ! -d 'test-react-with-jasmine' ] && git clone https://github.com/ibm-developer-skills-network/test-react-with-jasmine.git
   ```

4. 切换到目录 **test-react-with-jasmine** 开始实验。

   ```
   cd test-react-with-jasmine
   ```

5. 安装依赖项。

   ```
   npm install
   ```

6. 运行 React 应用程序。

   ```
   npm start
   ```

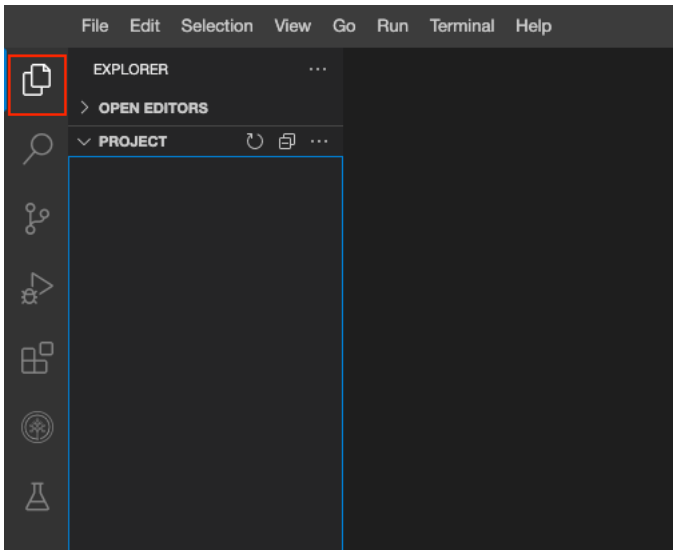
7. 要在浏览器中测试您的应用程序，请先运行该应用程序。

Launch Application

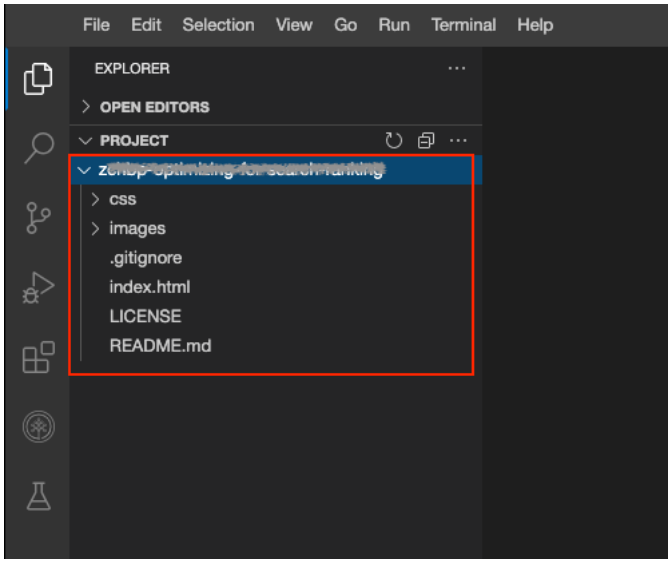
在云IDE中处理文件

如果您是云IDE的新手，本节将向您展示如何在云IDE中创建和编辑项目中的文件。

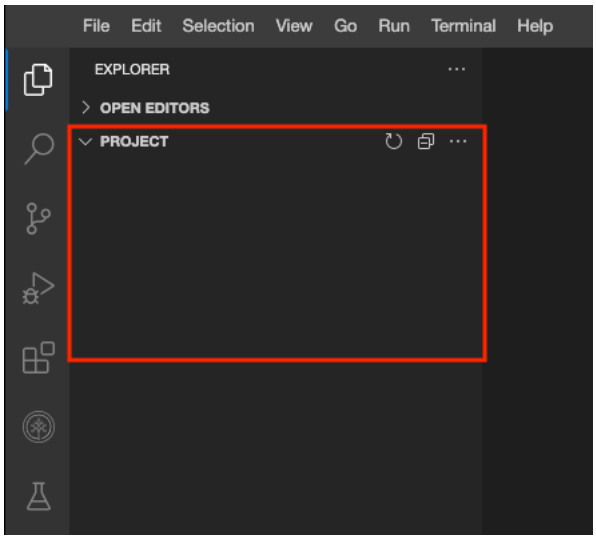
要查看云IDE中的文件和目录，请单击此文件图标以显示它。



如果您已经克隆了（使用 `git clone` 命令）基础代码/起始代码，那么它看起来如下：

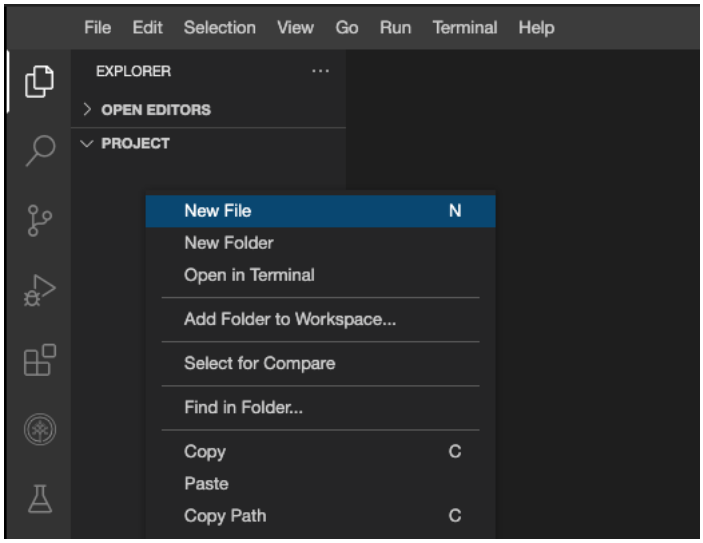


否则，一个空白项目看起来是这样的：



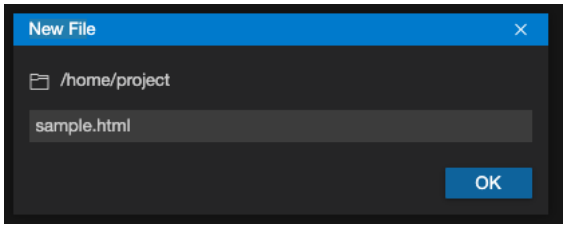
创建新文件

您可以右键单击并选择“新建文件”选项来创建项目中的文件。

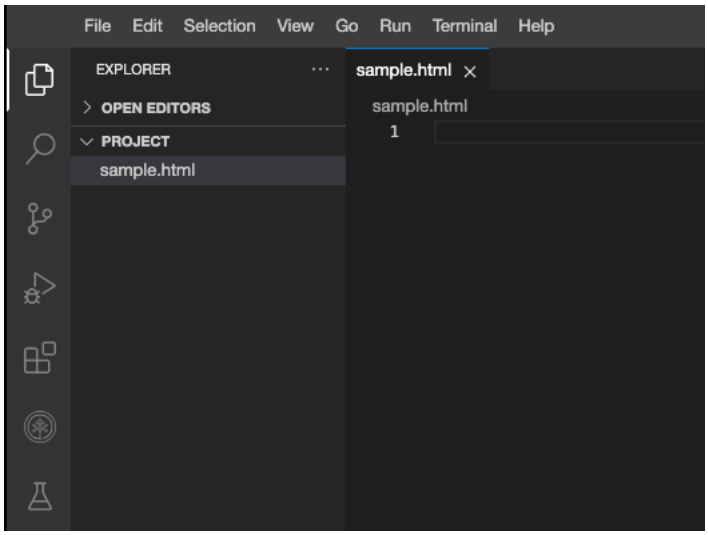


您也可以选择 文件 -> 新建文件 来实现相同的操作。

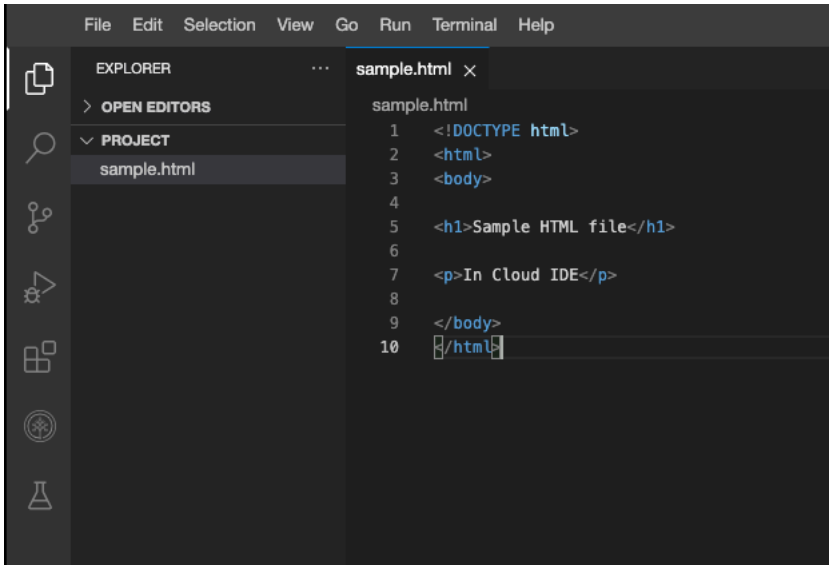
然后系统会提示您输入新文件的名称。在下面的示例中，我们创建了 `sample.html`。



单击目录结构中的文件名 `sample.html` 将在右侧窗格中打开该文件。您可以创建各种不同类型的文件；例如，JavaScript 文件可以命名为 `FILE_NAME.js`。

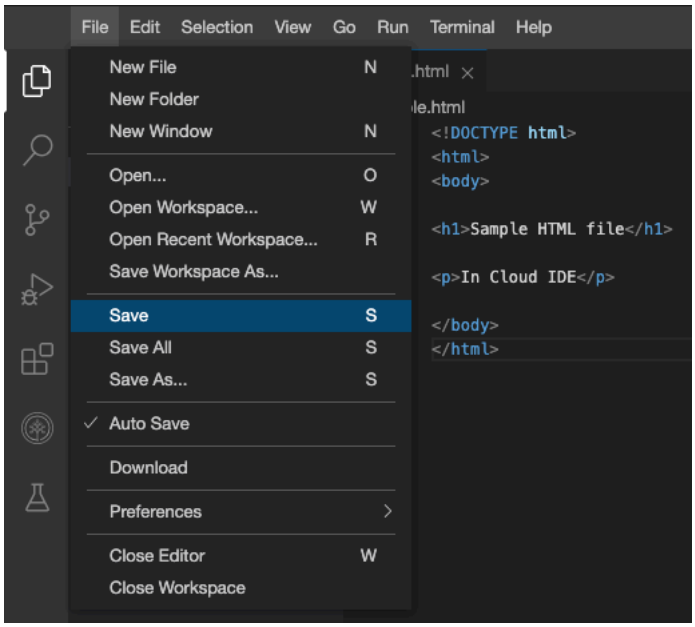


在示例中，我们粘贴了一些基本的 html 代码，然后保存了文件。



并通过以下方式保存：

- 进入菜单。
- 在 Mac 上按 ⌘ Cmd + S 或在 Windows 上按 CTRL + S。
- 或者它也可以为您自动保存。



练习 1：使用 npm 安装 Jasmine

第一步是安装 `jasmine` npm 包。这个模块允许你为你的 Node.js 代码运行 Jasmine 测试。输出将默认显示在你的终端中。

在你继续之前，让我们停止我们的 React 应用程序，并通过按 `CTRL + C` 将光标返回到终端。

你可以使用 npm 在你的项目中本地安装 Jasmine：

```
npm install --save-dev jasmine
```

通过上述本地安装，您可以使用 `npx jasmine ...` 命令调用 CLI 工具。

练习 2：初始化 Jasmine

运行以下命令以初始化 Jasmine 项目：

```
npx jasmine init
```

通过初始化 Jasmine，它会创建 `spec/support/jasmine.json` 文件，该文件作为 Jasmine 的配置文件。

```
{
  // Spec directory path relative to the current working dir when Jasmine is executed.
  "spec_dir": "spec",
  // Array of filepaths (and globs) relative to spec_dir to include and exclude
  "spec_files": [
    "**/*[sS]pec.?(m)js"
  ],
  // Array of filepaths (and globs) relative to spec_dir to include before Jasmine specs
  "helpers": [
    "helpers/**/*.?(m)js"
  ],
  // Configuration of the Jasmine environment
  // "env" is optional, as are all of its properties.
  "env": {
    // Stop execution of a spec after the first expectation failure in it
    "stopSpecOnExpectationFailure": false,
    // Run specs in semi-random order
    "random": false
  }
}
```

您还可以选择生成示例规范和源文件：

```
npx jasmine examples
```

您现在应该能够编写您的第一个测试套件。

但首先，我们将对 `package.json` 进行更改，以将 `jasmine` 设置为测试脚本。

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "jasmine",
    "eject": "react-scripts eject"
},
```

练习 3：编写第一个测试套件

首先，让我们创建一个可以测试的模块。

创建一个名为 `src/helloWorld.js` 的新文件，内容如下：

```
function helloWorld() {
    return "hello world";
}
module.exports = helloWorld;
```

这是一个简单的模块，当调用时返回 `hello world`。

为了测试这个模块，我们将创建一个名为 `spec/helloWorldSpec.js` 的 Jasmine 规范文件，内容如下：

```
const helloWorld = require('../src/helloWorld.js');
describe("helloWorld", () => {
    it("returns hello world", () => {
      var actual = helloWorld();
      expect(actual).toBe("hello world");
    });
  })
```

Jasmine 函数

以下函数定义提供了使用细节。

describe

`describe` 函数用于将相关的规范分组，通常位于每个测试文件的顶部。字符串参数用于命名规范的集合，并将与规范连接以形成规范的完整名称。这使得在大型测试套件中更容易找到规范。如果你正确命名，它们的描述将以传统的行为驱动开发（BDD）风格呈现为完整的句子。

it

通过调用全局的 Jasmine 函数 `it` 定义规范，`it` 像 `describe` 一样，接受一个字符串和一个函数。字符串是规范的标题，函数是规范或测试。一个规范包含一个或多个期望，用于测试代码的状态。在 Jasmine 中，期望是一个要么为真要么为假的断言。一个通过的规范包含所有为真的期望。一个失败的规范包含一个或多个为假的期望。

expect

期望是通过函数 `expect` 构建的，该函数接受一个称为实际值的值。它与一个匹配器函数相关联，该函数接受预期值。

Jasmine 包含丰富的匹配器集合，你可以在 API 文档 中找到完整列表。

练习 4：让我们进行测试

要对 hello world 运行单元测试，我们可以选择运行

```
npx jasmine
```

或

```
npm test
```

您将注意到以下结果：

```
1 spec, 0 failures
```

这意味着测试通过了。

如果您注意到，在 `Started` 下，一个绿色的点 `.` 代表通过的测试。失败的测试将显示红色的 F。

练习 5：新模块和测试

让我们添加两个新模块。

`src/factorial.js`

```
const factorial = (inputNumber) => {
  if (inputNumber < 0)
    throw new Error("We don't allow factorial of a negative number.");
  if (inputNumber === 1 || inputNumber === 0) return 1;
  return inputNumber * exports.factorial(inputNumber - 1);
};
exports.factorial = factorial;
```

和 `spec/factorialSpec.js`

```
const factorial = require("../src/factorial.js");
describe("factorial", function () {
  var calc;
  //This will be called before running each spec
  beforeEach(function () {
    calc = factorial;
  });
  describe("when calc is used to peform factorial operations", function () {
    it("should be able to calculate factorial of 0", function () {
      expect(calc.factorial(1)).toEqual(1);
    });
    it("should be able to calculate factorial of 1", function () {
      expect(calc.factorial(1)).toEqual(1);
    });
    it("should be able to calculate factorial of 5", function () {
      expect(calc.factorial(5)).toEqual(120);
    });
    it("should be able to throw error in factorial operation when the number is negative", function () {
      expect(function () {
        calc.factorial(-2);
      }).toThrowError(Error);
    });
  });
});
```

让我们还添加一个测试，如果传入的参数不是数字，则抛出一个错误。

```
    it("should be able to throw error in factorial operation when the input is not a number", function () {
      expect(function () {
        calc.factorial(true);
      }).toThrowError(Error);
    });
```

现在，通过执行来运行测试

```
npx jasmine
```

因此，将生成以下消息：

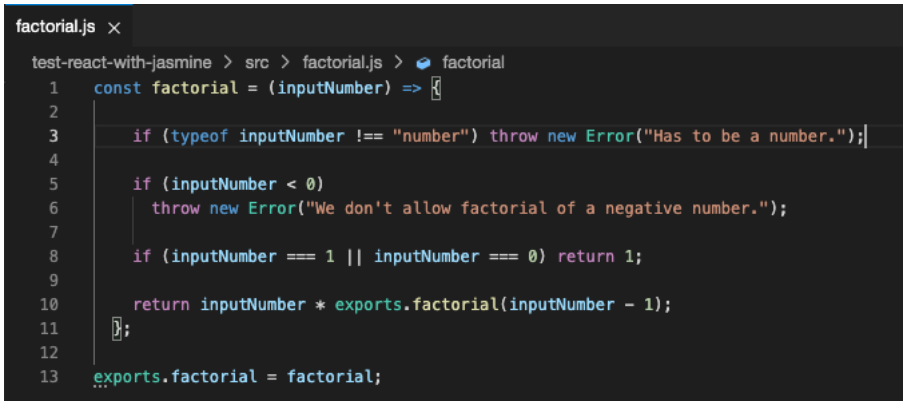
```
Started
.....F
Failures:
```

```
  1) factorial when calc is used to peform factorial operations should be able to throw error in factorial operation when the input is
    Message:
      Expected function to throw an Error.
...
6 specs, 1 failure
```

要通过这个测试，我们必须在 `src/factorial.js` 中的阶乘函数开头插入以下代码行。

```
    if (typeof inputNumber !== "number") throw new Error("Has to be a number.");
```

它将如下所示：



练习 6：重新运行测试

通过执行以下命令重新运行你的测试：

```
npx jasmine
```

将生成以下消息，其中 6 个点 `......` 代表 6 个通过的测试。

```
Started
......
6 specs, 0 failures
```

恭喜！您已完成使用 Jasmine 测试 React 的实验。

摘要：

在本实验中，您使用 Jasmine 测试了两个用 React 编写的模块。

Author(s)

Muhammad Yahya

(C) IBM Corporation. 保留所有权利。