**Практическая работа 1**

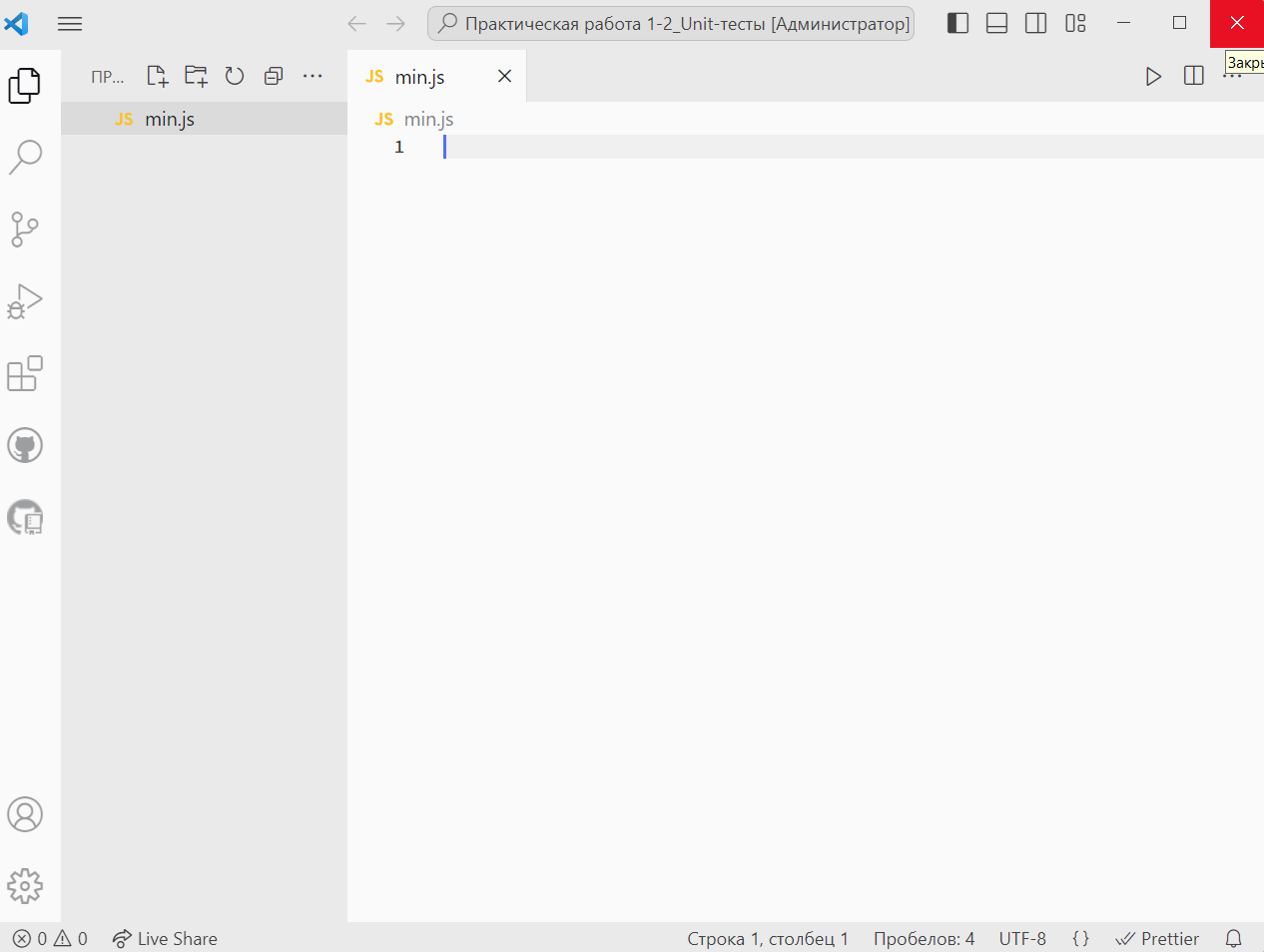
**Задание:** Для приложения javaScript разработать Unit-тест, который тестирует работу функции min().

Для функции min() установить модуль тестирования testMin. Проверить работу функции.

**Ход работы:**

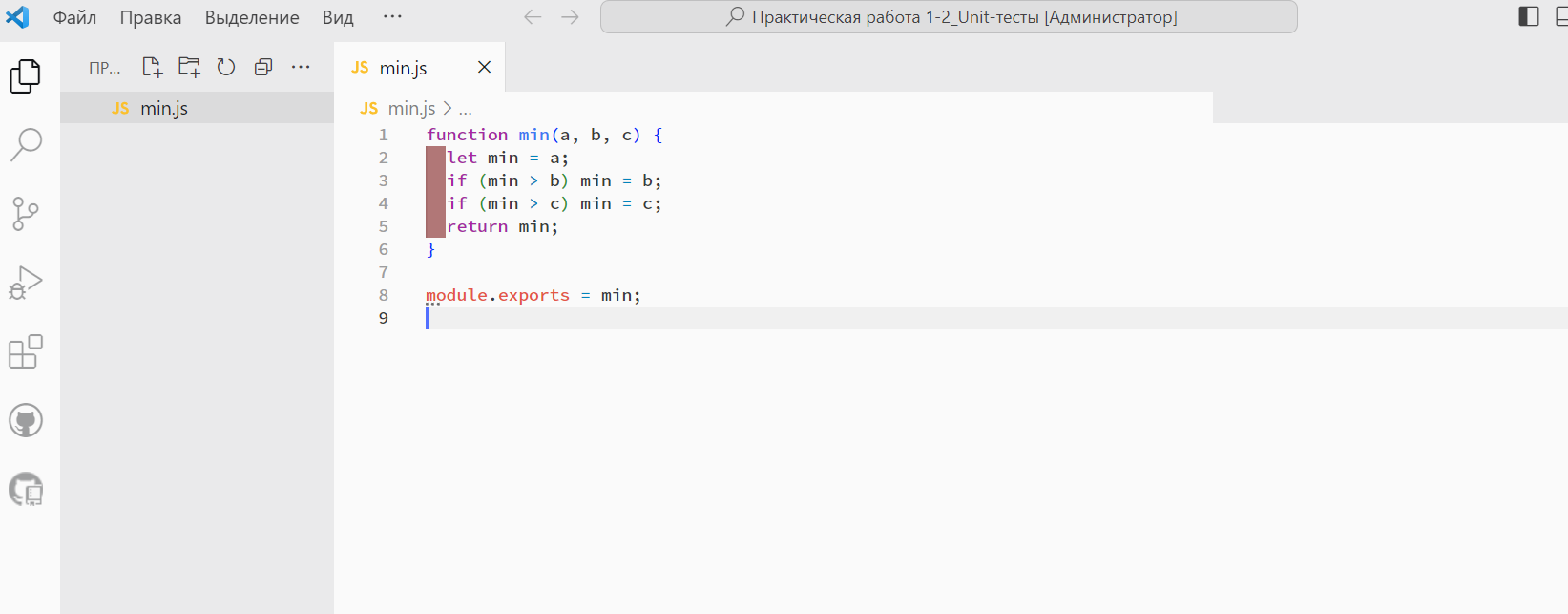
**1. Создать приложение**

Запустить Visual Studio Code. Чтобы создать приложение javaScript создаем файл **JS.**



**2. Подготовка приложения min.js**

**2.1. Текст программы, которую нужно протестировать**

****

Программа должна быть модулем. Тогда другой файл может импортировать его функции и использовать. Чтобы другой файл мог импортировать, необходимо сделать экспорт функции min из модуля:

module.exports = min;

экспорт располагается в конце кода модуля.

Поскольку, эта программа будет тестироваться из другого *модуля тестирования*, то  больше ничего вводить не нужно. В соответствии с условием задачи, нужно протестировать работу функции Min(), а это уже будет осуществляться из модуля тестирования. На данный момент наша программа готова к тестированию.

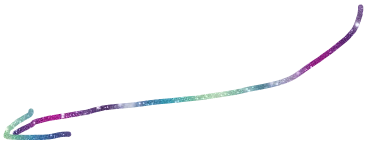
**3. Подготовка среды тестирования.**

**3.1. Установка фреймоворка тестирования Jest.**

**npm i –save-dev jest**

появились файлы package.json и package-lock.json,

добавляем в проект в файл package.json скрипт, по которому будет происходить вызов программы тестирования:

**4. Создание теста**

Тест создается отдельным файлом. Программа, которая будет тестироваться не знает об этом. Программа-тест, которая будет тестировать вызывает функции программы, которая тестируется. В нашем случае программа-тест будет вызывать функцию

min(a, b, c)

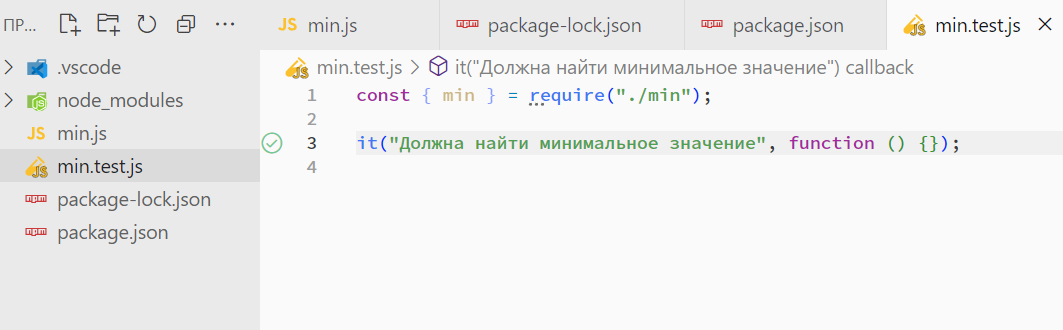
Создаем файл с названием по шаблону xxx.test.js, в нашем случае min.test.js

Создаем строку импорта функции min из файла-модуля min.js

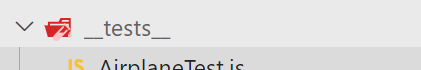
Вводим вызов функции it или test (на ваше усмотрение), vs code добавит импорт этой функции самостоятельно в первую строку программы.

В функции it добавим аргументы. Первый аргумент строка для описания, что должна сделать тестируемая функция, в нашем случае 'Должна найти минимальное значение'.

Вторым аргументом будет тело функции – теста. Сейчас наш код выглядит так:



В среде тестировщиков принято размещать тесты в каталоге :

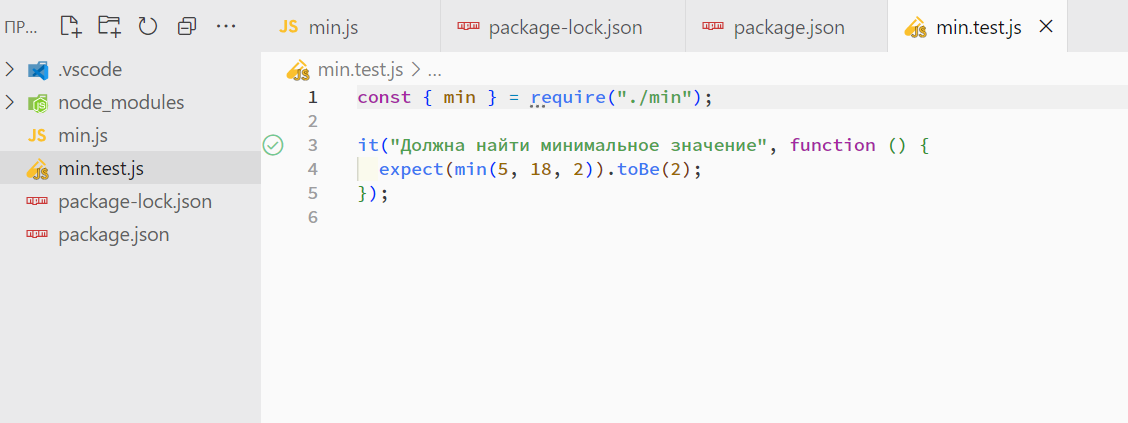


Настройки конфигурации jest учитываю данное правило. И могут «попросить» вас запускать тесты только из данного каталога.

**4.2. Структура служебного метода it**

Программный код, который тестирует функцию min(), будет представлять из себя метод expect() , который запускает функцию min с тестовыми аргументами, например, min(5, 18, 2). Функция min возвращает значение, которое мы хотим проверить. Далее цепочкой применяем метод .toBe , который проверяет, что тестируемая функция возвращает результат, который записан как аргумент метода .toBe.

Метод *expect()* возвращает объект «обертку», у которого есть ряд методов для сопоставления полученного значения с ожидаемым. Один из таких методов мы и использовали — *toBe*.



Давайте разберем основные из этих методов:

* **toBe()** — подходит, если нам надо сравнивать примитивные значения или является ли переданное значение ссылкой на тот же объект, что указан как ожидаемое значение. Сравниваются значения при помощи *Object.is()*. В отличие от === это дает возможность отличать 0 от -0, проверить равенство NaN c NaN.
* **toEqual()** — подойдёт, если нам необходимо сравнить структуру более сложных типов. Он сравнит все поля переданного объекта с ожидаемым. Проверит каждый элемент массива. И сделает это рекурсивно по всей вложенности.
* **toContain()** — проверят содержит массив или итерируемый объект значение. Для сравнения используется оператор ===.
* **toContainEqual()** — проверяет или содержит массив элемент с ожидаемой структурой.
* **toHaveLength()** — проверяет или свойство length у объекта соответствует ожидаемому.
* **toBeNull()** — проверяет на равенство с null.
* **toBeUndefined()** — проверяет на равенство с undefined.
* **toBeDefined()** — противоположность toBeUndefined. Проверяет или значение !== undefined.
* **toBeTruthy()** — проверяет или в булевом контексте значение соответствует true. Тоесть любые значения кроме false, null, undefined, 0, NaN и пустых строк.
* **toBeFalsy()** — противоположность toBeTruthy(). Проверяет или в булевом контексте значение соответствует false.
* **toBeGreaterThan()** и **toBeGreaterThanOrEqual()** — первый метод проверяет или переданное числовое значение больше, чем ожидаемое >, второй проверяет больше или равно ожидаемому >=.
* **toBeLessThan()** и **toBeLessThanOrEqual()** — противоположность toBeGreaterThan() и toBeGreaterThanOrEqual()
* **toBeCloseTo()** — удобно использовать для чисел с плавающей запятой, когда вам не важна точность и вы не хотите, чтобы тест зависел от незначительной разницы в дроби. Вторым аргументом можно передать до какого знака после запятой необходима точность при сравнении.
* **toMatch()** — проверяет соответствие строки регулярному выражению.
* **toThrow()** — используется в случаях, когда надо проверить исключение. Можно проверить как сам факт ошибки, так и проверить на выброс исключения определенного класса, либо по сообщению ошибки, либо по соответствию сообщения регулярному выражению.
* **not** — это свойство позволяет сделать проверки на НЕравенство. Оно предоставляет объект, который имеет все методы перечисленные выше, но работать они будут наоборот.

Полный перечень сопоставлений доступен по ссылке:

https://jestjs.io/ru/docs/expect

**5. Запуск теста на выполнение и проверка результата тестирования**

Чтобы запустить тест на выполнение, нужно в терминале запустить команду:

npm test

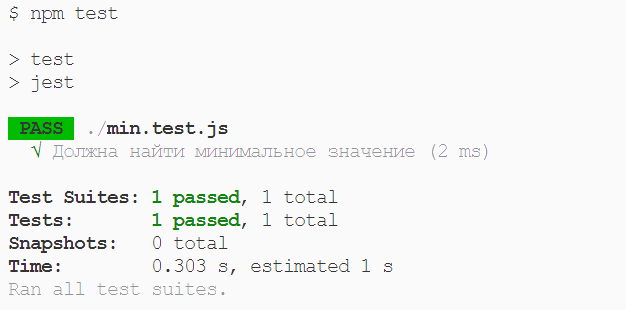
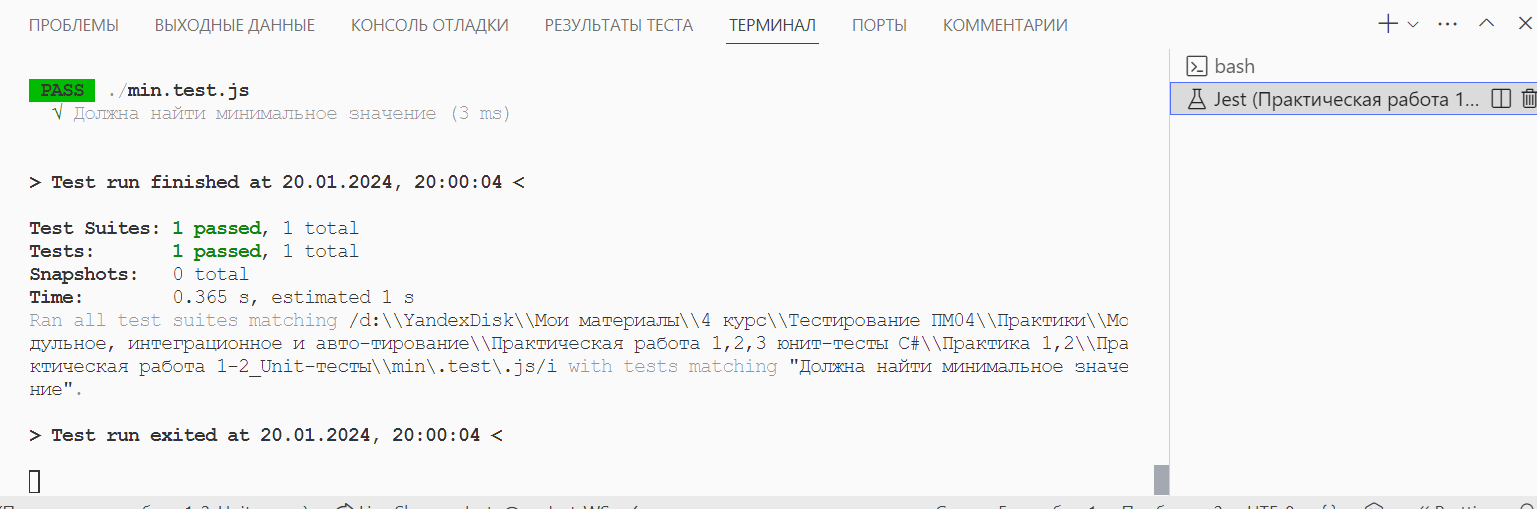
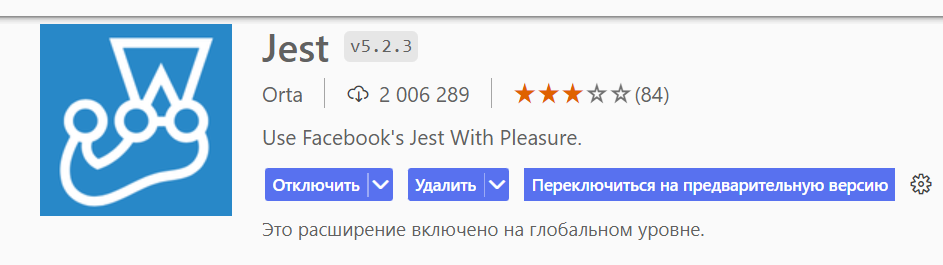


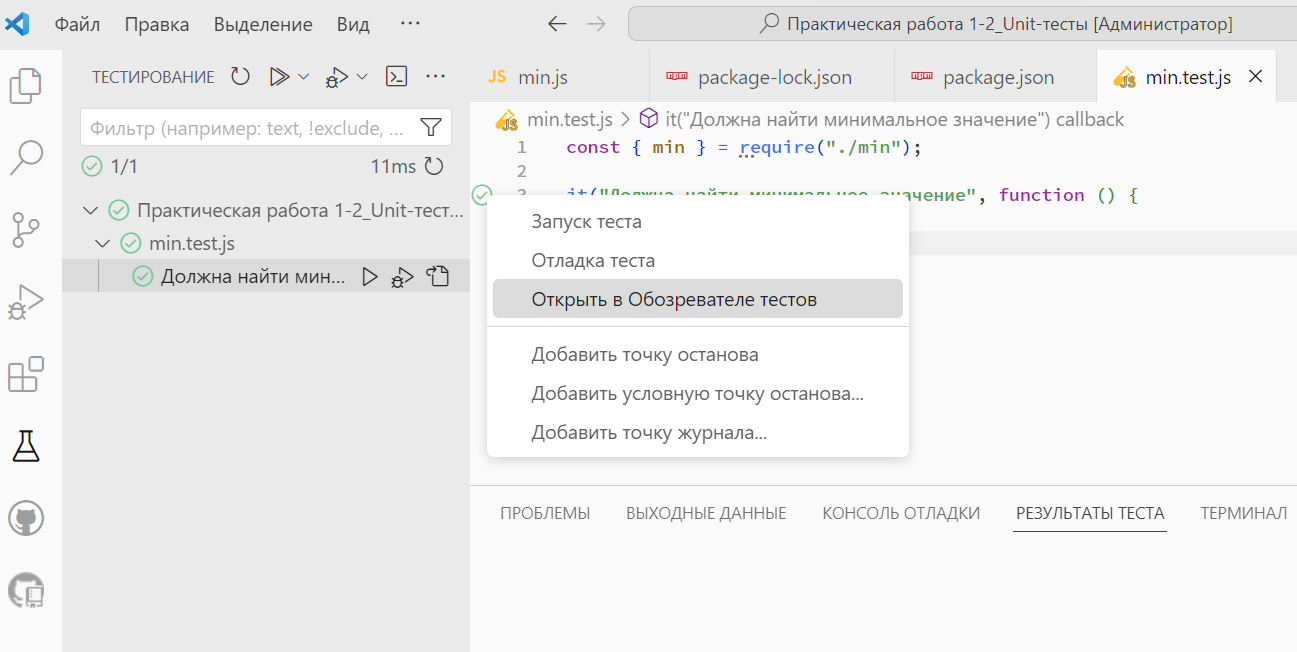
Рис. Вызов команды запуска тестирования и просмотр результата

После запуска теста, результат можно просмотреть в терминале фреймворка Jest:



А также просмотреть тесты в специальном окне Test Explorer, которое открывается если его вызвать, щелкнув по соответствующему значку в окне кода программы. Такая настройка появляется в случае установки плагина ‘Jest’ в VSCode.







Соответственно после запуска теста в окне Test Explorer будет отображен положительный результат теста. После этого можно сделать вывод о том, что функция min() для данного случая работает правильно.

**6. Итог. Взаимодействие между модулями**

В данной работе в сформированы два модуля. Один содержит функцию min(), которую нужно протестировать. Второй модуль содержит методы, которые тестируют.

В VS code каждый из проектов запускается с помощью различных команд меню. Так, проект min запускается стандартным способом node min.js. А тестирующий модуль min.test.js запускается настройкой скрипта npm test.

**7. Практическая часть. Задание для работы**

Разработать программу для подсчета объема и площади боковой поверхности цилиндра и создать модульный тест. Скриншоты программы, модуля тестирования и результатов тестирования оформить в формате файла Word. Файл назвать “Практика тестирование 1 Фамилия-Группа.docx”. Разместить в папке задания.

Формулы для расчета объема цилиндра и площади боковой поверхности найти самостоятельно.