Введение в тестирование в Vite с использованием Vitest

Vitest — это тестовый фреймворк нового поколения, созданный специально для работы с Vite. Он предлагает ряд преимуществ для тестирования в среде Vite, включая:

1. **Интеграцию с Vite**:
   - Vitest разработан с учетом Vite с самого начала, используя его улучшения, такие как мгновенная перезагрузка модулей (Hot Module Reload, HMR).
2. **Унифицированную конфигурацию с Vite**:
   - Одним из основных преимуществ Vitest является его унифицированная конфигурация с Vite. Если присутствует, Vitest будет читать ваш корневой файл конфигурации `vite.config.ts`, чтобы соответствовать плагинам и настройкам вашего приложения Vite. Например, конфигурация `Vite resolve.alias` и плагины будут работать "из коробки".

3. **Переиспользование конфигурации и плагинов Vite**:
    - Vitest позволяет переиспользовать конфигурацию и плагины Vite, обеспечивая согласованность между вашим приложением и тестами.
4. **Совместимость с Jest**:
    - Vitest предоставляет совместимый API с Jest, что позволяет использовать его как замену в большинстве проектов.
5. **Улучшенный режим наблюдения**:
    - Vitest предлагает умный и мгновенный режим наблюдения, который перезапускает только связанные изменения, аналогично HMR для тестов.
6. **Соответствие тестовой и сборочной среды**:
    - Vitest использует ту же конфигурацию, что и Vite, что обеспечивает соответствие тестовой среды и среды сборки, увеличивая надежность тестов.

Демонстрация примеров тестирования

```
# Создание нового проекта Vite с React
npx create-vite my-vite-project --template react

# Переход в каталог проекта
cd my-vite-project

# Установка Vitest и Jest (для совместимости API и снимков)
npm install vitest jest @vitest/react @types/jest --save-dev
```

1. Юнит-тестирование:

Создайте файлы `component.js` и `component.test.js` в каталоге `src`.

component.js:

```
// src/component.js
// Это простая функция, которая принимает два числа и возвращает их сумму.
export function add(a, b) {
    return a + b;
}
```

component.test.js:

```
// src/component.test.js
// Здесь мы импортируем функцию add из файла component.js, чтобы можно было ее
протестировать.
import { add } from "./component";
// Импортируем функцию test из библиотеки vitest, чтобы создать тест.
import { test } from "vitest";

// Создаем новый тест с описанием 'adds 1 + 2 to equal 3'.
test("adds 1 + 2 to equal 3", () => {
    // Используем функцию expect, чтобы проверить результат функции add.
    // Метод toBe проверяет, что результат функции add(1, 2) равен 3.
    expect(add(1, 2)).toBe(3);
});
```

Запустите тесты, выполнив следующую команду в терминале:

```
npx vitest
```

2. Интеграционное тестирование:

component.js:

```
// src/component.js
// Это компонент React, который рендерит текстовое поле и вызывает функцию onChange
при изменении текста.
import React from "react";

export function InputComponent({ onChange }) {
    return <input type="text" onChange={onChange} />;
}
```

component.test.js:

```
// src/component.test.js
// Импортируем необходимые библиотеки и компонент.
import React from "react";
import { render, fireEvent } from "@vitest/react";
import { InputComponent } from "./component";

// Создаем тест, который проверяет, что функция onChange вызывается с правильным
текстом.
test("calls onChange with the text", () => {
    // Создаем фиктивную (mock) функцию handleChange с помощью jest.fn().
    const handleChange = jest.fn();
    // Рендерим компонент с помощью функции render и передаем фиктивную функцию в
качестве пропса onChange.
    const { getByRole } = render(<InputComponent onChange={handleChange} />);
    // Используем функцию fireEvent для имитации ввода текста в текстовое поле.
    fireEvent.change(getByRole("textbox"), { target: { value: "Hello" } });
    // Проверяем, что фиктивная функция была вызвана с текстом 'Hello'.
    expect(handleChange).toHaveBeenCalledWith("Hello");
});
```

Запустите тесты аналогичным образом, используя команду `npx vitest` в терминале.

3. Тестирование снимками:

component.js:

```
// src/component.js
// Это простой компонент React, который рендерит "Hello World" в div элемент.
import React from "react";

export function MyComponent() {
    return <div>Hello World</div>;
}
```

component.test.js:

```
// src/component.test.js
// Импортируем необходимые библиотеки и компонент.
import React from "react";
import { render } from "@vitest/react";
import { MyComponent } from "./component";

// Создаем тест, который проверяет, что рендер компонента соответствует снимку.
test("matches the snapshot", () => {
```

```
    // Рендерим компонент с помощью функции render.
    const { asFragment } = render(<MyComponent />);
    // Используем функцию expect и метод toMatchSnapshot, чтобы проверить, что
рендер компонента соответствует снимку.
    expect(asFragment()).toMatchSnapshot();
});
```

Запуск тестов:

- Когда вы запускаете тесты с помощью команды `npx vitest`, Vitest ищет файлы с тестами в вашем проекте, запускает тесты и сообщает вам, прошли ли тесты успешно или нет.
- Если тест прошел успешно, вы увидите зеленую галочку рядом с описанием теста в терминале.
- Если тест провалился, вы увидите красный крестик и информацию об ошибке, что поможет вам понять, что пошло не так.

Настройка React с Vitest

Для настройки React с Vitest, сначала нужно создать новый проект React с использованием Vite. Затем установить Vitest и необходимые библиотеки для тестирования. Вот шаги для выполнения этого:

1. **Создание нового проекта React с использованием Vite:**

```
npx create-vite my-vite-react-project --template react
cd my-vite-react-project
```

2. **Установка Vitest и необходимых библиотек для тестирования:**

```
npm install vitest jest @vitest/react @types/jest --save-dev
```

3. **Создание компонента React для тестирования:** Создайте файл `ButtonComponent.js` в каталоге `src` вашего проекта:

```
// src/ButtonComponent.js
import React, { useState } from "react";
```

```
export function ButtonComponent() {
    const [clicked, setClicked] = useState(false);

    return (
        <button onClick={() => setClicked(true)}>
            {clicked ? "Clicked" : "Click me"}
        </button>
    );
}
```

4. **Создание модульного теста для компонента React:** Создайте файл `ButtonComponent.test.js` в каталоге `src` вашего проекта:

```
// src/ButtonComponent.test.js
import React from "react";
import { render, fireEvent } from "@vitest/react";
import { ButtonComponent } from "./ButtonComponent";

test("renders the button and handles click", () => {
    const { getByText } = render(<ButtonComponent />);
    const button = getByText("Click me");
    expect(button).toBeInTheDocument();
    fireEvent.click(button);
    expect(button).toHaveTextContent("Clicked");
});
```

5. **Запуск теста:** Запустите тесты, выполнив следующую команду в терминале:

```
npx vitest
```

Объяснение:

- В файле `ButtonComponent.js`, мы создали простой компонент кнопки, который изменяет свой текст при клике.
- В файле `ButtonComponent.test.js`, мы создали тест, который проверяет, что кнопка отображается правильно и реагирует на клик.
    - Метод `render` из `@vitest/react` используется для рендеринга компонента в виртуальном DOM.
    - Метод `getByText` используется для поиска элемента по тексту.
    - Метод `expect` с `toBeInTheDocument` и `toHaveTextContent` используется для проверки, что элемент присутствует в документе и имеет правильный текст.

- Функция `fireEvent.click` из `@vitest/react` используется для имитации клика по кнопке.

Тестирования компонента React с использованием Vitest

Для тестирования компонента React с использованием Vitest, который реагирует на изменение состояния, создадим компонент кнопки инкремента. Этот компонент будет увеличивать значение счетчика на 1 каждый раз, когда пользователь нажимает на кнопку.

1. **Создание компонента кнопки инкремента**:

```
// src/IncrementButton.js
import React, { useState } from "react";

function IncrementButton() {
    const [count, setCount] = useState(0);

    return (
        <div>
            <button onClick={() => setCount(count + 1)}>Increment</button>
            <span>{count}</span>
        </div>
    );
}

export default IncrementButton;
```

2. **Создание теста для компонента кнопки инкремента**:

```
// src/IncrementButton.test.js
import React from "react";
import { render, fireEvent } from "@vitest/react";
import IncrementButton from "./IncrementButton";

test("increments the count on button click", () => {
    const { getByText } = render(<IncrementButton />);
    const button = getByText("Increment");
    const countSpan = document.querySelector("span");
    expect(countSpan.textContent).toBe("0"); // проверка начального состояния
    fireEvent.click(button); // имитация клика
    expect(countSpan.textContent).toBe("1"); // проверка состояния после клика
});
```

3. **Запуск теста**:

```
npx vitest
```

Объяснение:

- В файле `IncrementButton.js`, мы создаем компонент `IncrementButton` с начальным состоянием `count`, установленным в `0`. При каждом клике на кнопку значение `count` увеличивается на `1`.
- В файле `IncrementButton.test.js`, мы создаем тест, который:
  - Использует функцию `render` из `@vitest/react` для рендеринга компонента в виртуальном DOM.
  - Использует функцию `getByText` для получения кнопки по тексту 'Increment'.
  - Использует `document.querySelector` для получения элемента `span`, который отображает текущее значение `count`.
  - Использует функцию `expect` с методом `toBe` для проверки начального состояния `count`.
  - Использует функцию `fireEvent.click` из `@vitest/react` для имитации клика по кнопке.
  - Использует функцию `expect` с методом `toBe` для проверки, что значение `count` увеличилось на `1` после клика.
- Команда `npx vitest` в терминале запускает тесты и выводит результаты тестирования в консоли. Если тест пройден успешно, вы увидите зеленую галочку рядом с описанием теста. Если тест провалился, вы увидите красный крестик и информацию об ошибке.