GUÍA 2.3.3

Configuración de pruebas unitarias en componentes front-end

Sigla	Asignatura	Experiencia de Aprendizaje
DSY1104	Desarrollo Full Stack II	EA Configuración de Jasmine y Karma
Tiempo	**Modalidad de Trabajo**	**Indicadores de logro**
2 h	Individual	IL 2.2 – IL 2.3

Antecedentes generales

Esta guía tiene como objetivo enumerar las acciones necesarias para dar solución a los problemas planteados.

Requerimientos para esta actividad

Para el desarrollo de esta actividad deberás disponer de:

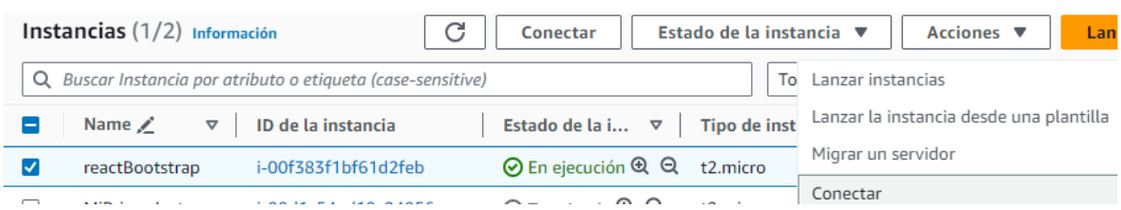
- Computador
- AWS – EC2

Actividad

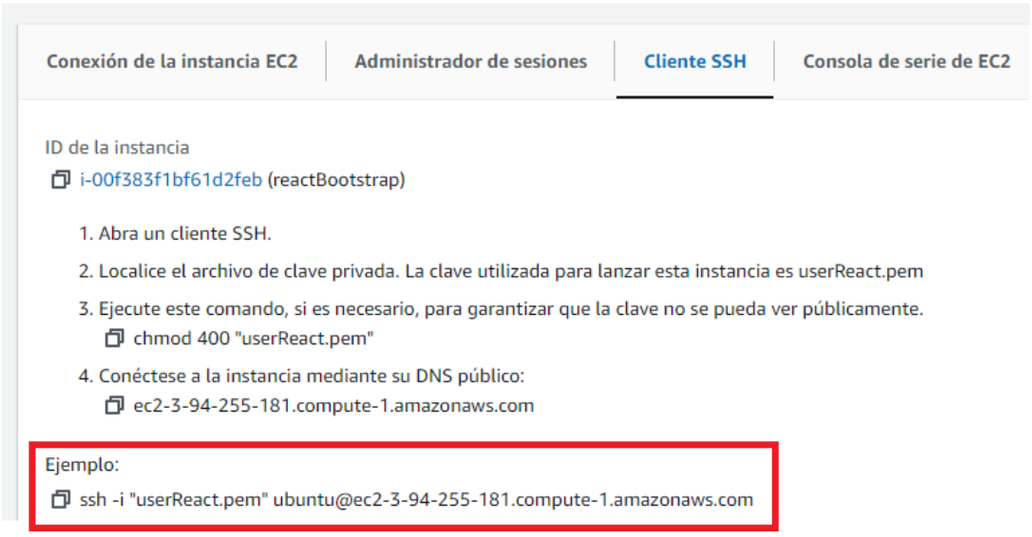
Esta actividad consiste en enumerar las acciones necesarias para dar solución a los casos que se verán a continuación, para ello los estudiantes deberán realizar la actividad de forma individual.

Sigue las Instrucciones

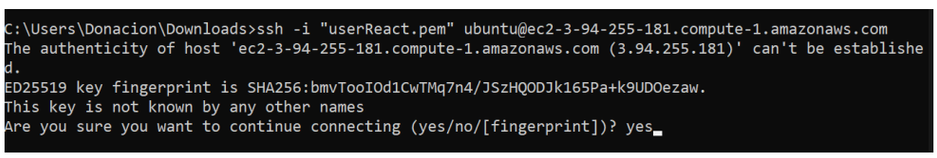
1. Ingresar a EC2 > Instancias y presionar Conectar



2. Se abrirá una nueva pestaña de conexión y copia el código que esta enmarcado en rojo:



3. Abre un CMD y ejecútalo como administrador. Debes dirigirte a la dirección donde esta userReact.pem y copiar el ejemplo anterior y colocar yes



4. Instalación de herramientas de testeo

```bash
npm install --save-dev jest @testing-library/react @testing-library/jest-dom
```

5. Ejemplo de componente optimizado

```jsx
import React, { lazy, Suspense } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'));

function OptimizedApp() {
  return (
    <Suspense fallback={<div>Cargando...</div>}>
      <LazyComponent />
    </Suspense>
  );
}
```

6. Estructura básica de pruebas unitarias

```javascript
import { render, screen } from '@testing-library/react';
import MiComponente from './MiComponente';

test('renderiza el componente correctamente', () => {
  render(<MiComponente />);
  expect(screen.getByText('Texto esperado')).toBeInTheDocument();
});
```

7. Mocking de dependencias

```javascript
jest.mock('./api');
import { fetchData } from './api';

test('maneja la carga de datos', async () => {
  fetchData.mockResolvedValue({ id: 1, name: 'Test' });
  // ... resto del test
});
```

8. Ejemplo de prueba de integración

```javascript
import { render, screen, waitFor } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import App from './App';

test('flujo de login completo', async () => {
  render(<App />);
  userEvent.type(screen.getByLabelText('Usuario'), 'testuser');
  userEvent.type(screen.getByLabelText('Contraseña'), 'password');
  userEvent.click(screen.getByText('Iniciar sesión'));
  await waitFor(() =>
expect(screen.getByText('Bienvenido')).toBeInTheDocument());
});
```

9. Herramienta de medición de rendimiento

```javascript
import { performance } from 'perf_hooks';

function medirRendimiento(componente) {
  const inicio = performance.now();
  render(componente);
  const fin = performance.now();
  console.log(`Tiempo de renderizado: ${fin - inicio} ms`);
}
```

10. Pruebas de inyección de código

```javascript
test('previene inyección XSS', () => {
  render(<Input />);
  userEvent.type(screen.getByRole('textbox'), '<script>alert("XSS")</script>');
  expect(screen.getByRole('textbox')).toHaveValue('<script>alert("XSS")</script>');
  expect(document.body.innerHTML).not.toContain('<script>alert("XSS")</script>');
});
```

11. Implementación de pruebas de accesibilidad

```javascript
import { axe } from 'jest-axe';

test('no tiene violaciones de accesibilidad', async () => {
```

```
  const { container } = render(<App />);
  const results = await axe(container);
  expect(results).toHaveNoViolations();
});
```

12. Configuración de Jest para t2.micro (AWS)

```javascript
// jest.config.js
module.exports = {
  maxWorkers: 2, // Limitar el número de workers
  maxConcurrency: 1, // Ejecutar tests en serie
};
```

13. Ejemplo de buildspec.yml

```yaml
version: 0.2

phases:

  install:

    runtime-versions:

      nodejs: 14

  pre_build:

    commands:

      - npm install

  build:

    commands:

      - npm test

      - npm run build

artifacts:

  files:

    - '**/*'

  base-directory: 'build'
```

14. Logging efectivo

```javascript
const winston = require('winston');

const logger = winston.createLogger({

  level: 'info',
```

```
  format: winston.format.json(),
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'error.log', level: 'error' })
  ]
});
```

15. Configuración de seguridad en EC2

    - Asegúrate de que el puerto 3000 esté abierto en el grupo de seguridad de tu instancia EC2.

    ```bash
    curl http://tu-ip-publica:3000
    ```

16. Ejecutar la aplicación
    - Inicia la aplicación:
    ```bash
    npm start
    ```
    - Accede a tu aplicación desde un navegador usando la IP pública de tu instancia EC2: `http://tu-ip-publica:3000`

    Nota: Este método es para desarrollo. Para producción, se recomienda construir la aplicación y servirla con un servidor web como Nginx.

17. Debería levantar la App

Recursos de apoyo

- Documentación Create React App: https://cra.link/deployment