1.-Entender el negocio y el Funcionamiento del API

Analizo **qué problema resuelve el API** y **qué contrato expone**.

- Flujo de negocio impactado
- Endpoints, métodos, auth
- Códigos de respuesta esperados

Ejemplo:
API de login → impacta seguridad y acceso.
Espero 200 con token o 401 con error.

2.-Definir escenarios y alcance

Diseño **casos positivos, negativos y borde** priorizando riesgo.

- Request válida
- Datos inválidos
- Autenticación incorrecta

Ejemplo:

- Login válido → 200
- Password incorrecto → 401
- Body incompleto → 400

3.-Separar datos y ambiente

Preparo **datos controlados y reutilizables**.

- Usuarios válidos / inválidos
- Tokens dinámicos
- Variables por ambiente

Ejemplo:
Usuario standard_user para happy path
Usuario inexistente para negativos

4.-Ejecutar las pruebas

Primero valido que el ambiente tenga la (API) servicio deployado y procedo con las pruebas manuales Validando:

- Status code
- Body
- Headers
- Reglas de negocio

Ejemplos

1.-Test de Login válido (Cypress)

Dado un usuario válido
Cuando hace login
Entonces el API devuelve 200

```
it('Login con exitoso devuelve token', () => {

cy.request('POST', '/api/login', {

username: 'standard_user',

password: 'secret_sauce'

}).then((response) => {

expect(response.status).to.eq(200);

expect(response.body).to.have.property('token');

  });

});
```

2.-Test de Login válido (Postman)

Dado un usuario válido
Cuando hace login
Entonces el API devuelve 200

Body

```
{
"username": "standard_user",
"password": "secret_sauce'"
}
```

Scripts

```
pm.test("Status code is 200", () => {
pm.response.to.have.status(200);
});

pm.test("Response contains token", () => {
const json = pm.response.json();
pm.expect(json).to.have.property("token");
});
```

Test Result

200 ok