FORMATO PRUEBA DE SOFTWARE

- **Denominación del Programa de Formación:** Tecnología en Análisis y Desarrollo de Sistemas de información
- **Código del Programa de Formación:** 228106/ Versión 102
- **Nombre del Proyecto:** Software para el sector empresarial en el departamento del huila.
- **Fase del Proyecto:** Ejecución
- **Actividad de Proyecto:** Construir la capa de datos, lógica de negocios y presentación del sistema de información aplicando estándares de calidad y buenas prácticas de ergonomía.
- **Competencia:** Construir el sistema que cumpla con los requisitos de la solución informática.
- **Resultados de aprendizaje a alcanzar:** Ejecutar y documentar las pruebas del software, aplicando técnicas de ensayo-error, de acuerdo con el plan diseñado y los procedimientos establecidos por la empresa

INTRODUCCIÓN

El propósito del siguiente documento es realizar seguimiento a cada uno de los proyectos formativos que se están desarrollando a lo largo del tecnólogo ADSO por medio de la implementación de pruebas unitarias.

Las pruebas unitarias nos permiten asegurarnos de que los métodos individuales del código de un proyecto funcionen correctamente, así como la integración de todo el proyecto.

Se busca en los requerimientos funcionales: encontrar errores relacionados con la funcionalidad del sistema y en los no funcionales, fallas de rendimiento, seguridad, usabilidad, fiabilidad.

Existen 4 tipos de pruebas:

- **Pruebas unitarias.** Diseñadas para probar una parte pequeña y específica de funcionalidad. Ej. Un método.

- **Pruebas de integración.** Diseñadas para probar la interacción entre los distintos componentes de un sistema.

- **Pruebas de sistema**. Diseñadas para probar el sistema en su totalidad como si de una caja negra se tratase.

- **Pruebas de aceptación.** Diseñadas para verificar que el sistema cumple con los requisitos exigidos por el usuario.

- **Test capa Controller**

DOCUMENTOS DE REFERENCIA	
Versión:	**Título:**
1.0	AuthService – Login sin 2FA

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Realizar login y cargar el usuario cuando no se requiere 2FA.**		
Objetivo	**Validar que el endpoint /login responda correctamente y que se realice la consulta a /me.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Credenciales de login (email, password).**		
Procedimiento de Prueba	**Enviar POST al endpoint /auth/login, recibir respuesta sin 2FA, luego solicitar /auth/me y actualizar el UserStore.**		
Datos de salida - Resultado Esperado	• **Respuesta del login sin 2FA.** • **Llamada al endpoint /me.** • **Carga del usuario en UserStore.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('Login should POST credentials and load user when 2FA not required', (done) => {
  const credentials: AuthLoginCredentials = { email: 'test@example.com', password: 'pass123' };
  const loginResponse: AuthLoginResponse = { requiresTwoFactor: false, message: 'Login OK' } as AuthLoginResponse;
  const mockUser: Partial<User> = { id: 1, email: 'test@example.com', firstName: 'Test' } as unknown as User;

  service.Login(credentials).subscribe(res => {
    expect(res).toEqual(loginResponse);
    expect(userStore.set).toHaveBeenCalledWith(mockUser as User);
    done();
  });

  const loginReq = httpMock.expectOne(`${baseUrl}/login`);
  expect(loginReq.request.method).toBe('POST');
  expect(loginReq.request.body).toEqual(credentials);
  expect(loginReq.request.withCredentials).toBeTrue();
  loginReq.flush(loginResponse);

  const meReq = httpMock.expectOne(`${baseUrl}/me`);
  expect(meReq.request.method).toBe('GET');
  meReq.flush(mockUser);
});
```

DOCUMENTOS DE REFERENCIA	
Versión:	**Título:**
1.0	AuthService – Login con 2FA requerido

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Realizar login cuando se requiere 2FA sin cargar al usuario.**		
Objetivo	**Validar que al solicitar /login con requiresTwoFactor = true no se invoque /me.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Credenciales de login (email, password).**		
Procedimiento de Prueba	**Enviar POST al endpoint /auth/login y verificar que no se realiza ninguna otra solicitud.**		
Datos de salida - Resultado Esperado	• **Respuesta indicando que se requiere 2FA.** • **No cargar usuario en UserStore.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('Login should return response without loading user when 2FA required', (done) => {
  const credentials: AuthLoginCredentials = { email: 'test@example.com', password: 'pass123' };
  const loginResponse: AuthLoginResponse = { requiresTwoFactor: true, message: '2FA required' } as AuthLoginResponse;

  service.Login(credentials).subscribe(res => {
    expect(res).toEqual(loginResponse);
    expect(userStore.set).not.toHaveBeenCalled();
    done();
  });

  const loginReq = httpMock.expectOne(`${baseUrl}/login`);
  loginReq.flush(loginResponse);
});
```

DOCUMENTOS DE REFERENCIA	
Versión:	**Título:**
1.0	AuthService – Obtener usuario actual (GetMe)

Detalle de la prueba

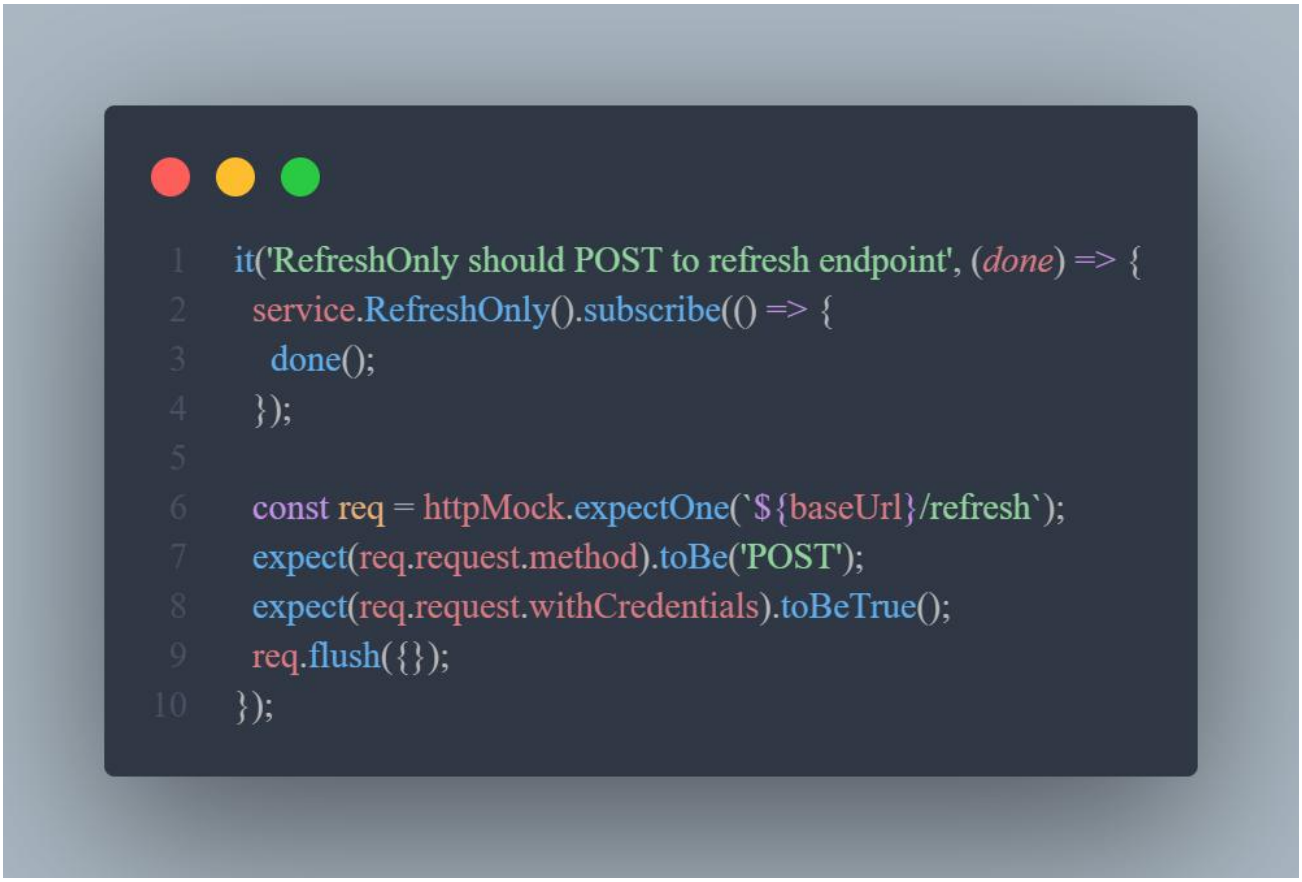
Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener datos del usuario autenticado.**		
Objetivo	**Validar que el endpoint /me devuelva los datos del usuario y se almacenen en UserStore.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Ninguno**		
Procedimiento de Prueba	**Realizar GET al endpoint /auth/me y registrar el usuario obtenido.**		
Datos de salida - Resultado Esperado	• **Respuesta con los datos del usuario.** • **Ejecución de UserStore.set.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('GetMe should fetch user and update store', (done) => {
  const mockUser: Partial<User> = { id: 1, email: 'test@example.com', firstName: 'Test' } as unknown as User;

  service.GetMe().subscribe(user => {
    expect(user).toEqual(mockUser as User);
    expect(userStore.set).toHaveBeenCalledWith(mockUser as User);
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/me`);
  expect(req.request.method).toBe('GET');
  expect(req.request.withCredentials).toBeTrue();
  req.flush(mockUser);
});
```

1.0	AuthService – RefreshOnly

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Renovar la sesión del usuario.**		
Objetivo	**Validar que el endpoint /refresh se invoque correctamente.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Ninguno**		
Procedimiento de Prueba	**Enviar POST al endpoint /auth/refresh.**		
Datos de salida - Resultado Esperado	• **Recibir respuesta vacía o de confirmación.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**



```
it('RefreshOnly should POST to refresh endpoint', (done) => {
  service.RefreshOnly().subscribe(() => {
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/refresh`);
  expect(req.request.method).toBe('POST');
  expect(req.request.withCredentials).toBeTrue();
  req.flush({});
});
```

1.0	AuthService – Logout

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Cerrar la sesión y limpiar el estado del usuario.**		
Objetivo	**Validar que /logout cierre sesión, limpie UserStore y navegue a inicio.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Ninguno**		
Procedimiento de Prueba	**Enviar POST a /auth/logout, limpiar datos y navegar a /.**		
Datos de salida - Resultado Esperado	• **Respuesta exitosa.** • **UserStore.clear ejecutado.** • **Navegación a /.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('logout should POST to logout, clear store, and navigate', (done) => {
  service.logout().subscribe(() => {
    expect(userStore.clear).toHaveBeenCalled();
    expect(router.navigate).toHaveBeenCalledWith(['/']);
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/logout`);
  expect(req.request.method).toBe('POST');
  expect(req.request.withCredentials).toBeTrue();
  req.flush({});
});
```

1.0	AuthService – Registro de usuario

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Registrar nuevos usuarios en el sistema.**		
Objetivo	**Validar el envio de datos de registro al endpoint correspondiente.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Datos del usuario.**		
Procedimiento de Prueba	**Enviar POST a /auth/register con los datos proporcionados.**		
Datos de salida - Resultado Esperado	• **Recibir confirmación del registro exitoso.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('Register should POST registration data', (done) => {
  const registerDto = { email: 'new@example.com', password: 'newpass123' } as any;

  service.Register(registerDto).subscribe(() => {
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/register`);
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(registerDto);
  expect(req.request.withCredentials).toBeTrue();
  req.flush({ success: true });
});
```

1.0	AuthService – Confirmación 2FA

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Confirmar código 2FA y cargar datos del usuario.**		
Objetivo	**Validar que /confirmar-2fa procese el código y posteriormente se solicite /me.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**email y código de verificación.**		
Procedimiento de Prueba	**Enviar POST a /auth/confirmar-2fa, luego GET a /auth/me.**		
Datos de salida - Resultado Esperado	• **Respuesta de confirmación.** • **Carga del usuario en UserStore.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('ConfirmTwoFactor should POST verification code and load user', (done) => {
  const verifyDto = { email: 'test@example.com', code: '123456' } as any;
  const confirmResponse = { success: true } as any;
  const mockUser: Partial<User> = { id: 1, email: 'test@example.com' };

  service.ConfirmTwoFactor(verifyDto).subscribe(res => {
    expect(res).toEqual(confirmResponse);
    expect(userStore.set).toHaveBeenCalledWith(mockUser as User);
    done();
  });

  const confirmReq = httpMock.expectOne(`${baseUrl}/confirmar-2fa`);
  expect(confirmReq.request.method).toBe('POST');
  expect(confirmReq.request.body).toEqual(verifyDto);
  confirmReq.flush(confirmResponse);

  const meReq = httpMock.expectOne(`${baseUrl}/me`);
  meReq.flush(mockUser);
});
```

1.0	AuthService – Solicitud de restablecimiento de contraseña

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Generar un código de recuperación de contraseña.**		
Objetivo	**Validar que se envíe el email correctamente al endpoint de recuperación.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**email del usuario.**		
Procedimiento de Prueba	**Enviar POST a /auth/recuperar/enviar-codigo con el email.**		
Datos de salida - Resultado Esperado	• **Recibir confirmación del envío del código.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('RequestPasswordReset should POST email', (done) => {
  const email = 'reset@example.com';

  service.RequestPasswordReset(email).subscribe(() => {
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/recuperar/enviar-codigo`);
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual({ email });
  req.flush({ success: true });
});
```

1.0	AuthService – Confirmación de restablecimiento de contraseña

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Confirmar el nuevo password mediante código.**		
Objetivo	**Validar que los parámetros (email, código, nueva contraseña) se envíen correctamente.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**email, código, nueva contraseña.**		
Procedimiento de Prueba	**Enviar POST a /auth/recuperar/confirmar con los parámetros correspondientes.**		
Datos de salida - Resultado Esperado	• **Recibir confirmación de que el password fue restablecido.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('ConfirmPasswordReset should POST reset confirmation', (done) => {
  const params = { email: 'reset@example.com', code: '123456', newPassword: 'newpass' };

  service.ConfirmPasswordReset(params).subscribe(() => {
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/recuperar/confirmar`);
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(params);
  req.flush({ success: true });
});
```

SERVICIO GENERICO

1.0	GenericService – getAll

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener todos los elementos del recurso configurado.**		
Objetivo	**Validar que el método getAll() construya correctamente la URL del recurso y realice una solicitud GET.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Ninguno**		
Procedimiento de Prueba	**Invocar getAll(), capturar la solicitud generada y validar que use la ruta ${apiURL}/dummy.**		
Datos de salida - Resultado Esperado	• **Solicitud GET enviada a la URL del recurso y respuesta recibida exitosamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should build correct URL for getAll', () => {
  service.getAll().subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy`);
  expect(req.request.method).toBe('GET');
  req.flush([]);
});
```

1.0	GenericService – getById

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener un elemento específico por su identificador.**		
Objetivo	**Validar que la URL generada sea ${apiURL}/dummy/{id} y realice correctamente la solicitud GET.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**id del elemento.**		
Procedimiento de Prueba	**Invocar getById(id) y validar que la URL generada incluya el id correspondiente.**		
Datos de salida - Resultado Esperado	• **Solicitud GET enviada a la URL del elemento con respuesta válida.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should build correct URL for getById', () => {
  const id = 123;
  service.getById(id).subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy/${id}`);
  expect(req.request.method).toBe('GET');
  req.flush({ id, name: 'test' });
});
```

1.0	GenericService – create

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Crear una nueva entidad en el recurso.**		
Objetivo	**Validar que el método create() realice un POST a la URL base del recurso con el body correcto.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Objeto DTO con los campos de la nueva entidad.**		
Procedimiento de Prueba	**Invocar create(dto) y validar método, body y URL generada.**		
Datos de salida - Resultado Esperado	• **Se recibe la entidad creada con su identificador.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should create a new entity', () => {
  const dto = { name: 'new' } as any;
  service.create(dto).subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy`);
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(dto);
  req.flush({ id: 1, name: 'new' });
});
```

1.0	GenericService – update

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Actualizar una entidad existente mediante su identificador.**		
Objetivo	**Validar que se construya la URL ${apiURL}/dummy/{id} y se envíe un PUT con el body correspondiente.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**id de la entidad, objeto DTO actualizado.**		
Procedimiento de Prueba	**Invocar update(id, dto) y verificar método PUT, URL y contenido enviado.**		
Datos de salida - Resultado Esperado	• **Entidad actualizada correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should update an existing entity', () => {
  const id = 5;
  const dto = { name: 'updated' } as any;
  service.update(id, dto).subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy/${id}`);
  expect(req.request.method).toBe('PUT');
  expect(req.request.body).toEqual(dto);
  req.flush({ id, name: 'updated' });
});
```

1.0	GenericService – delete

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Eliminar una entidad por su identificador.**		
Objetivo	**Validar que la URL generada ${apiURL}/dummy/{id} sea correcta y que se envíe una solicitud DELETE.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**id de la entidad.**		
Procedimiento de Prueba	**Invocar delete(id) y validar el método DELETE y la URL.**		
Datos de salida - Resultado Esperado	• **Respuesta exitosa indicando eliminación.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should delete an entity', () => {
  const id = 7;
  service.delete(id).subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy/${id}`);
  expect(req.request.method).toBe('DELETE');
  req.flush(null);
});
```

1.0	GenericService – deleteLogic (soft-delete)

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Realizar una eliminación lógica agregando sufijo /soft-delete.**		
Objetivo	**Validar que el método deleteLogic() construya la URL correcta y envíe PATCH sin body.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**id de la entidad.**		
Procedimiento de Prueba	**Invocar deleteLogic(id) y capturar la solicitud para validar URL y método.**		
Datos de salida - Resultado Esperado	• **Respuesta exitosa confirmando la eliminación lógica.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should perform logical delete via PATCH', () => {
  const id = 9;
  service.deleteLogic(id).subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy/${id}/soft-delete`);
  expect(req.request.method).toBe('PATCH');
  expect(req.request.body).toBeNull();
  req.flush(null);
});
```

1.0	GenericService – changeActiveStatus

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Cambiar el estado activo/inactivo de la entidad.**		
Objetivo	**Validar que el método genere la URL ${apiURL}/dummy/{id}/estado y envíe PATCH con { active }.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**id de la entidad, valor booleano del estado.**		
Procedimiento de Prueba	**Invocar changeActiveStatus(id, active) y validar URL, método y body enviado.**		
Datos de salida - Resultado Esperado	• **Entidad con el estado actualizado.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('should change active status', () => {
  const id = 11;
  const active = false;
  service.changeActiveStatus(id, active).subscribe();
  const req = httpMock.expectOne(`${environment.apiURL}/dummy/${id}/estado`);
  expect(req.request.method).toBe('PATCH');
  expect(req.request.body).toEqual({ active });
  req.flush({ id, name: 'test', active });
});
```

PLAZA Services

1.0	SquareService – getAll (normalización de imágenes)

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener todas las plazas y normalizar la propiedad images.**		
Objetivo	**Validar que getAll() invoque getCards() y que imágenes nulas o indefinidas sean reemplazadas por [].**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Mock con imágenes null y undefined.**		
Procedimiento de Prueba	• **Invocar getAll().** • **Interceptar GET a /plaza/cards.** • **Validar que imágenes nulas o inexistentes se conviertan en [].**		
Datos de salida - Resultado Esperado	• **GET a /plaza/cards.** • **Normalización de images.** • **Datos compatibles con SquareSelectModel.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAll should call getCards and return formatted items', (done) => {
  service.getAll().subscribe(res => {
    expect(res).toEqual(expected_getAll);
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/cards`);
  expect(req.request.method).toBe('GET');
  req.flush(mockData_getAll);
});
```

1.0	SquareService – getCards (preservación de imágenes)

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Mantener intactas las imágenes que el backend entrega.**		
Objetivo	**Validar que getCards() no altere las imágenes existentes del modelo ImageModel.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Mock con imágenes completas del backend.**		
Procedimiento de Prueba	• **Invocar getCards().** • **Interceptar GET a /plaza/cards.** • **Verificar que images no sea modificado.** • **Validar que primaryImagePath se mantenga igual.**		
Datos de salida - Resultado Esperado	• **images se preserva exactamente.** • **primaryImagePath no cambia.** • **Se devuelve un SquareSelectModel válido.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getCards should preserve images when they exist', (done) => {
  service.getCards().subscribe(res => {
    expect(res[0].images).toEqual(mockImages);
    expect(res[0].primaryImagePath).toBe('/uploads/test.jpg');
    done();
  });

  const req = httpMock.expectOne(`${baseUrl}/cards`);
  expect(req.request.method).toBe('GET');
  req.flush(mockData_getCards);
});
```

Establishment services

1.0	EstablishmentService – getAll con parámetros por defecto

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener todos los establecimientos aplicando el límite por defecto.**		
Objetivo	**Validar que getAll() construya HttpParams con el limit predeterminado.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**Ninguno**		
Procedimiento de Prueba	• **Llamar getAll(), interceptar la solicitud y verificar que el parámetro limit coincida con el valor configurado.**		
Datos de salida - Resultado Esperado	• **Solicitud GET enviada con limit = establishmentsDefaultLimit.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAll should call base URL with default params', () => {
  service.getAll().subscribe();

  const req = httpMock.expectOne(
    r => r.url === baseUrl && r.params.get('limit') === defaultLimit.toString()
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList);
});
```

1.0	EstablishmentService – getAll con activeOnly

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Filtrar establecimientos activos.**		
Objetivo	**Verificar que se envíe activeOnly=true en los parámetros.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**{ activeOnly: true }**		
Procedimiento de Prueba	• **Ejecutar getAll({ activeOnly: true }) y validar el parámetro.**		
Datos de salida - Resultado Esperado	• **Parámetro activeOnly=true.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAll should apply activeOnly=true', () => {
  service.getAll({ activeOnly: true }).subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === baseUrl &&
      r.params.get('activeOnly') === 'true' &&
      r.params.get('limit') === defaultLimit.toString()
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList);
});
```

1.0	EstablishmentService – getAll (sobrescritura de límite)

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Modificar dinámicamente el límite de resultados.**		
Objetivo	**Validar que limit en GetAllOptions reemplace el límite por defecto.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**{ limit: 5 }**		
Procedimiento de Prueba	• **Ejecutar getAll({ limit: 5 }) y validar el parámetro.**		
Datos de salida - Resultado Esperado	• **limit=5**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAll should override limit param', () => {
  service.getAll({ limit: 5 }).subscribe();

  const req = httpMock.expectOne(
    r => r.url === baseUrl && r.params.get('limit') === '5'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList);
});
```

1.0	EstablishmentService – getByPlaza

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener establecimientos asociados a una plaza.**		
Objetivo	**Validar URL /plaza/{id} y parámetros.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**plazaId=10, { activeOnly:true, limit:20 }**		
Procedimiento de Prueba	• **Ejecutar y verificar URL + params.**		
Datos de salida - Resultado Esperado	• **URL correcta** • **activeOnly=true** • **limit=20**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getByPlaza should send correct URL and params', () => {
  service.getByPlaza(10, { activeOnly: true, limit: 20 }).subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === `${baseUrl}/plaza/10` &&
      r.params.get('activeOnly') === 'true' &&
      r.params.get('limit') === '20'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList);
});
```

1.0	EstablishmentService – getById sin parámetros

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Consultar establecimiento por ID.**		
Objetivo	**Validar URL sin parámetros.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	**id=7**		
Procedimiento de Prueba	• **Ejecutar service.getById(7).** • **Interceptar solicitud HTTP.** • **Verificar ausencia de parámetros.**		
Datos de salida - Resultado Esperado	• **GET /Establishments/7** • **Sin parámetros en la request**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getById should work without params', () => {
  service.getById(7).subscribe();

  const req = httpMock.expectOne(`${baseUrl}/7`);
  expect(req.request.method).toBe('GET');
  expect(req.request.params.keys().length).toBe(0);
  req.flush(mockSelectList[0]);
});
```

1.0	EstablishmentService – getById con activeOnly

Detalle de la prueba

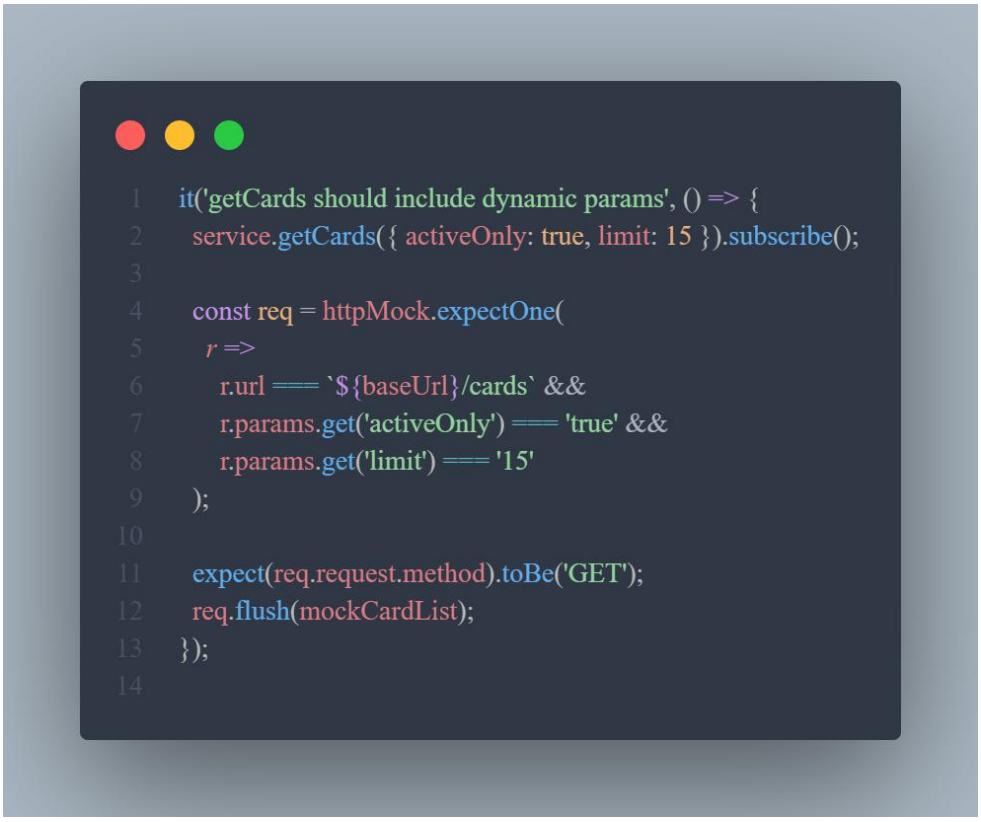
Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Filtrar establecimiento solo si está activo.**		
Objetivo	**Validar que el parámetro activeOnly=true sea enviado.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **id = 7** • **activeOnly = true**		
Procedimiento de Prueba	• **Ejecutar getById(7, true) y verificar URL + parámetros.**		
Datos de salida - Resultado Esperado	• **GET /Establishments/7** • **Param: activeOnly = true**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getById should set activeOnly=true', () => {
  service.getById(7, true).subscribe();

  const req = httpMock.expectOne(
    r => r.url === `${baseUrl}/7` && r.params.get('activeOnly') === 'true'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList[0]);
});
```

1.0	EstablishmentService – getCards

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener tarjetas de establecimientos con filtros dinámicos.**		
Objetivo	**Validar parámetros construidos mediante buildParams.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **{ activeOnly: true, limit: 15 }**		
Procedimiento de Prueba	• **Ejecutar getCards({ activeOnly:true, limit:15 })** • **Verificar ruta /Establishments/cards** • **Verificar parámetros generados.**		
Datos de salida - Resultado Esperado	• **activeOnly=true** • **limit=15**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

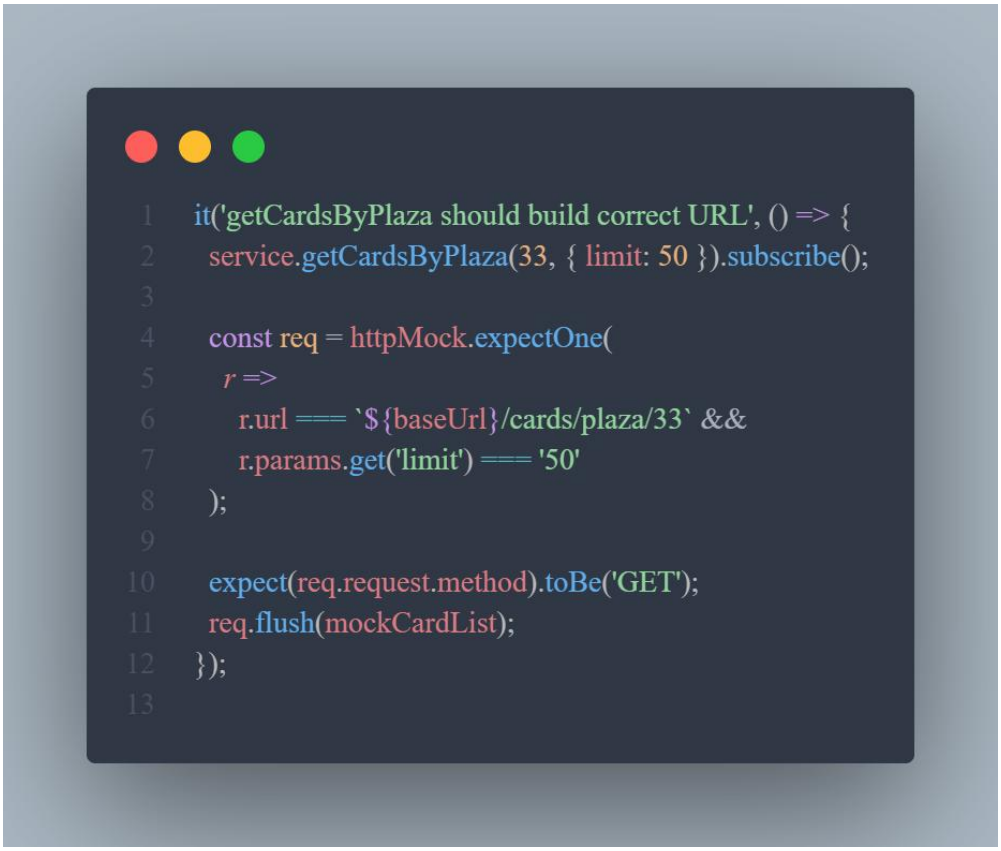


```
it('getCards should include dynamic params', () => {
  service.getCards({ activeOnly: true, limit: 15 }).subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === `${baseUrl}/cards` &&
      r.params.get('activeOnly') === 'true' &&
      r.params.get('limit') === '15'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockCardList);
});
```

1.0	EstablishmentService – getCardsByPlaza

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Consultar tarjetas por plaza.**		
Objetivo	**Validar URL /cards/plaza/{id} y sus parámetros.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **plazaId = 33** • **{limit: 50}**		
Procedimiento de Prueba	• **URL** • **Parámetros**		
Datos de salida - Resultado Esperado	• **GET /Establishments/cards/plaza/33** • **limit=50**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**



```
it('getCardsByPlaza should build correct URL', () => {
  service.getCardsByPlaza(33, { limit: 50 }).subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === `${baseUrl}/cards/plaza/33` &&
      r.params.get('limit') === '50'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockCardList);
});
```

1.0	EstablishmentService – getAllAny

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Permitir obtener la lista de establecimientos enviando únicamente un límite directo sin otras opciones.**		
Objetivo	**Validar que getAllAny(limit) delegue correctamente en getAll({ limit }) y construya los parámetros esperados.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **limit = 99**		
Procedimiento de Prueba	• **Ejecutar getAllAny(99).** • **Interceptar la solicitud HTTP.** • **Validar que la URL sea .../Establishments.** • **Validar que el parámetro limit sea "99".**		
Datos de salida - Resultado Esperado	• **Solicitud GET correcta** • **limit=99**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAllAny should call getAll with limit', () => {
  service.getAllAny(99).subscribe();

  const req = httpMock.expectOne(
    r => r.url === baseUrl && r.params.get('limit') === '99'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList);
});
```

1.0	EstablishmentService – getAllActive

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Listar únicamente establecimientos activos.**		
Objetivo	**Validar que getAllActive(limit) llame internamente a getAll({ activeOnly:true, limit }).**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **limit = 12**		

Procedimiento de Prueba	• **Ejecutar getAllActive(12).**
	• **Interceptar la solicitud.**
	• **Validar parámetros:**
	  ○ **activeOnly = true**
	  ○ **limit = "12"**		

Datos de salida - Resultado Esperado	• **Parámetros enviados correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAllActive should call getAll with activeOnly=true', () => {
  service.getAllActive(12).subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === baseUrl &&
      r.params.get('activeOnly') === 'true' &&
      r.params.get('limit') === '12'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockSelectList);
});
```

1.0	EstablishmentService – getCardsAny

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Listar tarjetas de establecimientos sin filtros.**		
Objetivo	**Validar que getCardsAny() utilice los valores por defecto del servicio.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **Ninguno**		
Procedimiento de Prueba	• **Ejecutar getCardsAny().** • **Validar que la ruta sea /Establishments/cards.** • **Validar que el parámetro limit coincida con el defaultLimit del entorno.**		
Datos de salida - Resultado Esperado	• **limit = defaultLimit**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getCardsAny should call getCards without options', () => {
  service.getCardsAny().subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === `${baseUrl}/cards` &&
      r.params.get('limit') === defaultLimit.toString()
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockCardList);
});
```

1.0	EstablishmentService – getCardsActive

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener tarjetas solo de establecimientos activos.**		
Objetivo	**Validar que se envíe el parámetro activeOnly=true.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **Ninguno**		
Procedimiento de Prueba	• **Ejecutar getCardsActive().** • **Interceptar la solicitud.** • **Validar que activeOnly=true.**		
Datos de salida - Resultado Esperado	• **El filtro se envía correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getCardsActive should call getCards with activeOnly=true', () => {
  service.getCardsActive().subscribe();

  const req = httpMock.expectOne(
    r =>
      r.url === `${baseUrl}/cards` &&
      r.params.get('activeOnly') === 'true'
  );

  expect(req.request.method).toBe('GET');
  req.flush(mockCardList);
});
```

1.0	EstablishmentService – create

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Crear un establecimiento nuevo excluyendo campos no válidos para el API.**		
Objetivo	**Verificar que el DTO enviado NO contenga:** **files** **images** **imagesToDelete**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **DTO completo de creación.**		

Procedimiento de Prueba	• **Ejecutar create(dtoCreate).**
	• **Interceptar POST a /Establishments.**
	• **Verificar que el body solo incluya:**
	  ○ **name**
	  ○ **description**
	  ○ **areaM2**
	  ○ **uvtQtv**
	  ○ **plazaId**
	  ○ **address**		

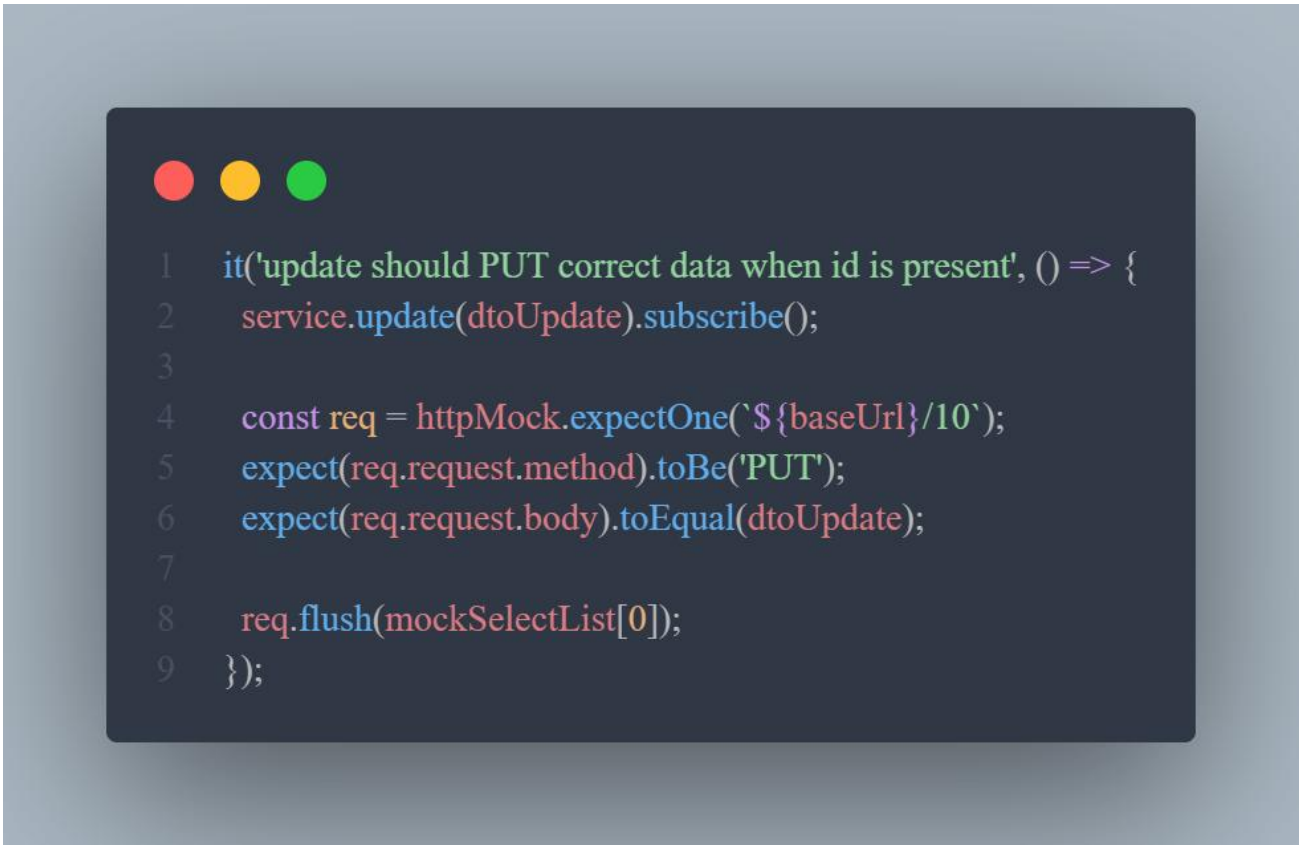
Datos de salida - Resultado Esperado	• **DTO limpio enviado correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**


```
it('create should POST cleaned DTO without file fields', () => {
  service.create(dtoCreate).subscribe();

  const req = httpMock.expectOne(baseUrl);
  expect(req.request.method).toBe('POST');

  // files, images y imagesToDelete deben ser removidos
  expect(req.request.body).toEqual({
    name: 'Test',
    description: 'Desc',
    address: 'Addr'
  });

  req.flush(mockSelectList[0]);
});
```

1.0	EstablishmentService – update correcto

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Actualizar establecimiento enviando el DTO completo.**		
Objetivo	**Verificar que la solicitud PUT se envíe correctamente a /Establishments/{id}.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **DTO válido con id = 10**		
Procedimiento de Prueba	• **Ejecutar update(dtoUpdate).** • **Interceptar PUT.** • **Validar la URL y el body.**		
Datos de salida - Resultado Esperado	• **PUT correcto al recurso y con datos válidos.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

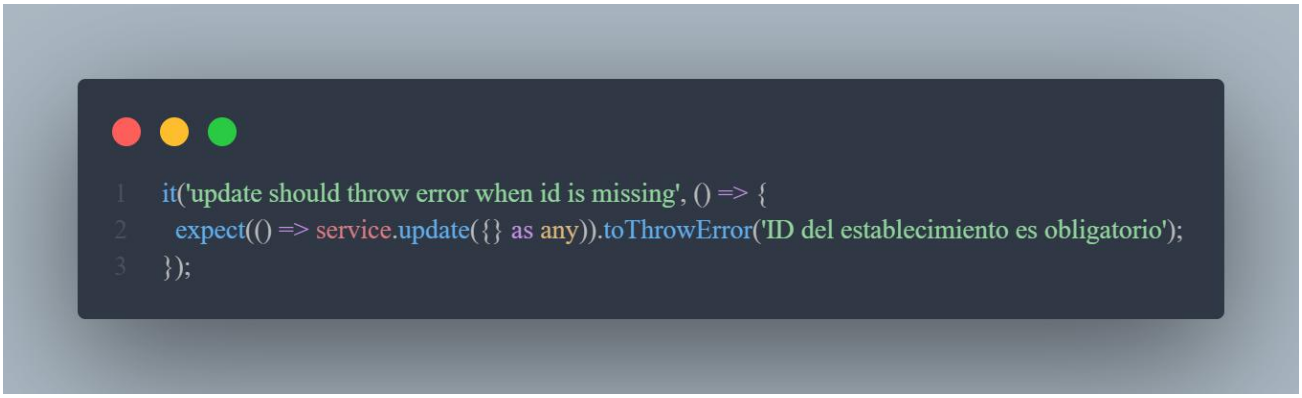


```
it('update should PUT correct data when id is present', () => {
  service.update(dtoUpdate).subscribe();

  const req = httpMock.expectOne(`${baseUrl}/10`);
  expect(req.request.method).toBe('PUT');
  expect(req.request.body).toEqual(dtoUpdate);

  req.flush(mockSelectList[0]);
});
```

1.0	EstablishmentService – update sin ID

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Evitar actualización sin ID obligatorio.**		
Objetivo	**Verificar que update() arroje un error cuando id no está definido.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **DTO inválido sin ID ({})**		
Procedimiento de Prueba	• **Ejecutar update({} as any).** • **Validar que se arroje el error esperado.**		
Datos de salida - Resultado Esperado	• **Error: "ID del establecimiento es obligatorio"**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**



```
it('update should throw error when id is missing', () => {
  expect(() => service.update({} as any)).toThrowError('ID del establecimiento es obligatorio');
});
```

1.0	EstablishmentService – delete

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Eliminar establecimiento completamente.**		
Objetivo	**Verificar que DELETE se envía a /Establishments/{id}.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **id = 77**		
Procedimiento de Prueba	• **Ejecutar service.delete(77)** • **Interceptar DELETE** • **Verificar URL correcta**		
Datos de salida - Resultado Esperado	• **DELETE exitoso.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('delete should DELETE by id', () => {
  service.delete(77).subscribe();

  const req = httpMock.expectOne(`${baseUrl}/77`);
  expect(req.request.method).toBe('DELETE');
  req.flush({});
});
```

1.0	EstablishmentService – deleteLogic

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Verificar DELETE en /Establishments/{id}/logic.**		
Objetivo	**Verificar que DELETE se envía a /Establishments/{id}.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **id = 55**		
Procedimiento de Prueba	• **Interceptar la solicitud DELETE.**		
Datos de salida - Resultado Esperado	• **URL correcta y método DELETE.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('deleteLogic should DELETE logical endpoint', () => {
  service.deleteLogic(55).subscribe();

  const req = httpMock.expectOne(`${baseUrl}/55/logic`);
  expect(req.request.method).toBe('DELETE');
  req.flush({});
});
```

1.0	EstablishmentService – changeActiveStatus

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Cambiar estado activo/inactivo del establecimiento.**		
Objetivo	**Verificar PATCH en /Establishments/{id}/estado enviando { active }.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **Ejecutar changeActiveStatus(99, true).** • **Interceptar PATCH.** • **Verificar body { active:true }.**		
Procedimiento de Prueba	• **PATCH correcto.**		
Datos de salida - Resultado Esperado	• **Prueba Exitosa**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('changeActiveStatus should PATCH active state', () => {
  service.changeActiveStatus(99, true).subscribe();

  const req = httpMock.expectOne(`${baseUrl}/99/estado`);
  expect(req.request.method).toBe('PATCH');
  expect(req.request.body).toEqual({ active: true });

  req.flush(mockSelectList[0]);
});
```

1.0	ContractService – getAll (endpoint /mine)

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener todos los contratos del usuario autenticado.**		
Objetivo	**Verificar que el método getAll() realice un GET al endpoint /contract/mine incluyendo un timestamp _ts.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **Ninguno**		

Procedimiento de Prueba	• **Ejecutar service.getAll().**
	• **Interceptar la solicitud.**
	• **Verificar:**
	  ○ **Método: GET**
	  ○ **URL contiene /contract/mine**
	  ○ **Parámetro _ts presente**
	• **Validar que el response coincida con los contratos simulados.**		

Datos de salida - Resultado Esperado	• **El método obtiene correctamente la lista desde /mine.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getAll should fetch contracts from /mine endpoint', () => {
  const mockContracts: ContractSelectModel[] = [
    { id: 1, name: 'Contract 1' } as unknown as ContractSelectModel,
    { id: 2, name: 'Contract 2' } as unknown as ContractSelectModel
  ];

  service.getAll().subscribe(contracts => {
    expect(contracts).toEqual(mockContracts);
    expect(contracts.length).toBe(2);
  });

  const req = httpMock.expectOne(r => r.url.includes(`${baseUrl}/mine`));
  expect(req.request.method).toBe('GET');
  expect(req.request.params.has('_ts')).toBeTrue();
  req.flush(mockContracts);
});
```

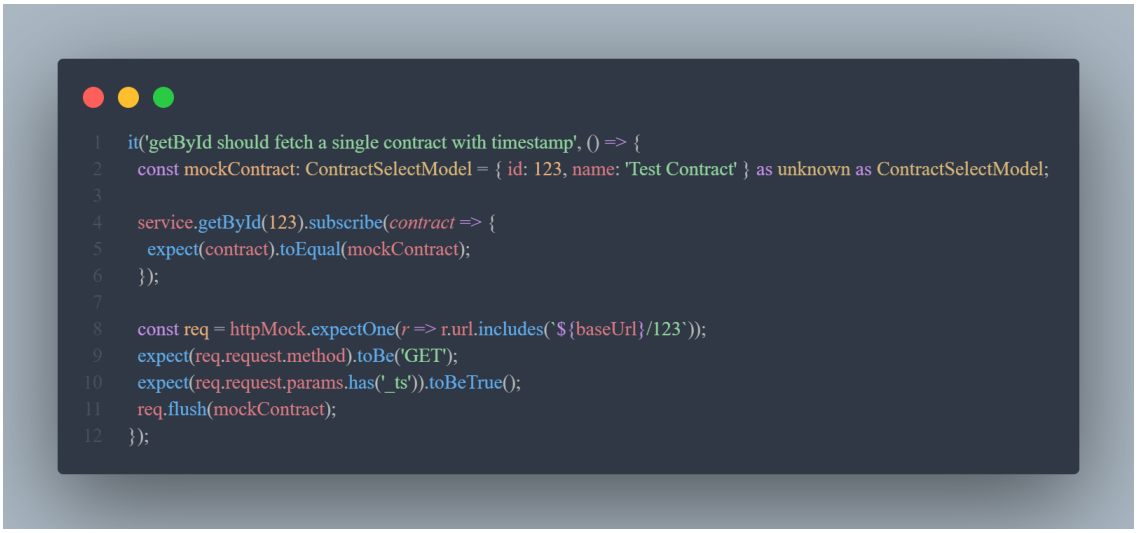
1.0	ContractService – getById

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Obtener un contrato específico por ID.**		
Objetivo	**Validar que la solicitud incluya _ts y consulte correctamente /contract/{id}.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **id = 123**		

Procedimiento de Prueba	• **Ejecutar getById(123).**
	• **Interceptar solicitud GET.**
	• **Verificar:**
	  ○ **URL /contract/123**
	  ○ **Param _ts presente**		

Datos de salida - Resultado Esperado	• **Se retorna el contrato correcto.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**



```
it('getById should fetch a single contract with timestamp', () => {
  const mockContract: ContractSelectModel = { id: 123, name: 'Test Contract' } as unknown as ContractSelectModel;

  service.getById(123).subscribe(contract => {
    expect(contract).toEqual(mockContract);
  });

  const req = httpMock.expectOne(r => r.url.includes(`${baseUrl}/123`));
  expect(req.request.method).toBe('GET');
  expect(req.request.params.has('_ts')).toBeTrue();
  req.flush(mockContract);
});
```

1.0	ContractService – create

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Crear un contrato nuevo.**		
Objetivo	**Verificar que el método haga un POST al endpoint base y envíe el DTO completo.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **DTO válido de creación.**		

Procedimiento de Prueba	• **Ejecutar create(dto).**
	• **Interceptar POST.**
	• **Verificar:**
	  ○ **URL igual al base**
	  ○ **Body igual al DTO**		

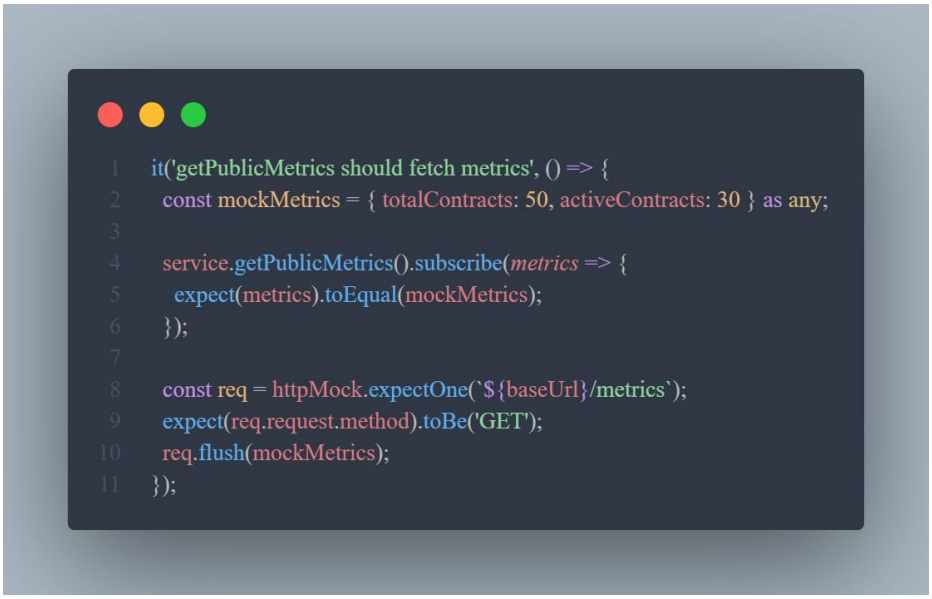
Datos de salida - Resultado Esperado	• **Contrato creado correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('create should POST a new contract', () => {
  const newContract: ContractCreateModel = { name: 'New Contract' } as unknown as ContractCreateModel;
  const createdContract: ContractSelectModel = { id: 999, name: 'New Contract' } as unknown as ContractSelectModel;

  service.create(newContract).subscribe(contract => {
    expect(contract).toEqual(createdContract);
  });

  const req = httpMock.expectOne(baseUrl);
  expect(req.request.method).toBe('POST');
  expect(req.request.body).toEqual(newContract);
  req.flush(createdContract);
});
```

1.0	ContractService – getPublicMetrics

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Consultar métricas públicas de contratos.**		
Objetivo	**Verificar que el método haga GET al endpoint /contract/metrics.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **Ninguno**		
Procedimiento de Prueba	• **Ejecutar getPublicMetrics().** • **Verificar ruta /contract/metrics.** • **Validar respuesta con mock.**		
Datos de salida - Resultado Esperado	• **Métricas retornadas correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**



```
it('getPublicMetrics should fetch metrics', () => {
  const mockMetrics = { totalContracts: 50, activeContracts: 30 } as any;

  service.getPublicMetrics().subscribe(metrics => {
    expect(metrics).toEqual(mockMetrics);
  });

  const req = httpMock.expectOne(`${baseUrl}/metrics`);
  expect(req.request.method).toBe('GET');
  req.flush(mockMetrics);
});
```

1.0	ContractService – downloadContractPdf

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Descargar PDF de un contrato.**		
Objetivo	• **Validar que el método:** • **Haga GET a /contract/{id}/pdf** • **Solicite responseType: 'blob'** • **Incluya _ts**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **id = 123**		

Procedimiento de Prueba	• **Ejecutar downloadContractPdf(123).**
	• **Interceptar GET.**
	• **Verificar:**
	  ○ **URL contiene /contract/123/pdf**
	  ○ **responseType = 'blob'**
	  ○ **Param _ts presente**		

Datos de salida - Resultado Esperado	• **PDF recibido correctamente.**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('downloadContractPdf should request PDF blob', () => {
  const mockBlob = new Blob(['PDF content'], { type: 'application/pdf' });

  service.downloadContractPdf(123).subscribe(blob => {
    expect(blob).toEqual(mockBlob);
  });

  const req = httpMock.expectOne(r => r.url.includes(`${baseUrl}/123/pdf`));
  expect(req.request.method).toBe('GET');
  expect(req.request.responseType).toBe('blob');
  expect(req.request.params.has('_ts')).toBeTrue();
  req.flush(mockBlob);
});
```

1.0	ContractService – getMonthlyObligations

Detalle de la prueba

Fecha de realización: 24/11/2025	**Duración de la prueba: 3 ms**		
Requerimientos Funcional de la prueba	**Consultar obligaciones mensuales asociadas a un contrato.**		
Objetivo	**Verificar que el método haga GET a /contract/{id}/obligations con _ts.**		
Tipo de Prueba	**Unitaria**		
Datos de entrada de la prueba	• **id = 123**		

Procedimiento de Prueba	• **Ejecutar getMonthlyObligations(123).**
	• **Interceptar GET.**
	• **Verificar:**
	  ○ **URL contiene /contract/123/obligations**
	  ○ **Param _ts existe**		

Datos de salida - Resultado Esperado	• **Obligaciones obtenidas correctamente**		
Resultado Obtenido Prueba Exitosa	**¿Prueba Exitosa?**	**Si (x)**	**No ( )**

```
it('getMonthlyObligations should fetch obligations for a contract', () => {
  const mockObligations = [
    { id: 1, amount: 100, dueDate: '2024-01-01' },
    { id: 2, amount: 200, dueDate: '2024-02-01' }
  ] as any;

  service.getMonthlyObligations(123).subscribe(obligations => {
    expect(obligations).toEqual(mockObligations);
    expect(obligations.length).toBe(2);
  });

  const req = httpMock.expectOne(r => r.url.includes(`${baseUrl}/123/obligations`));
  expect(req.request.method).toBe('GET');
  expect(req.request.params.has('_ts')).toBeTrue();
  req.flush(mockObligations);
});
```