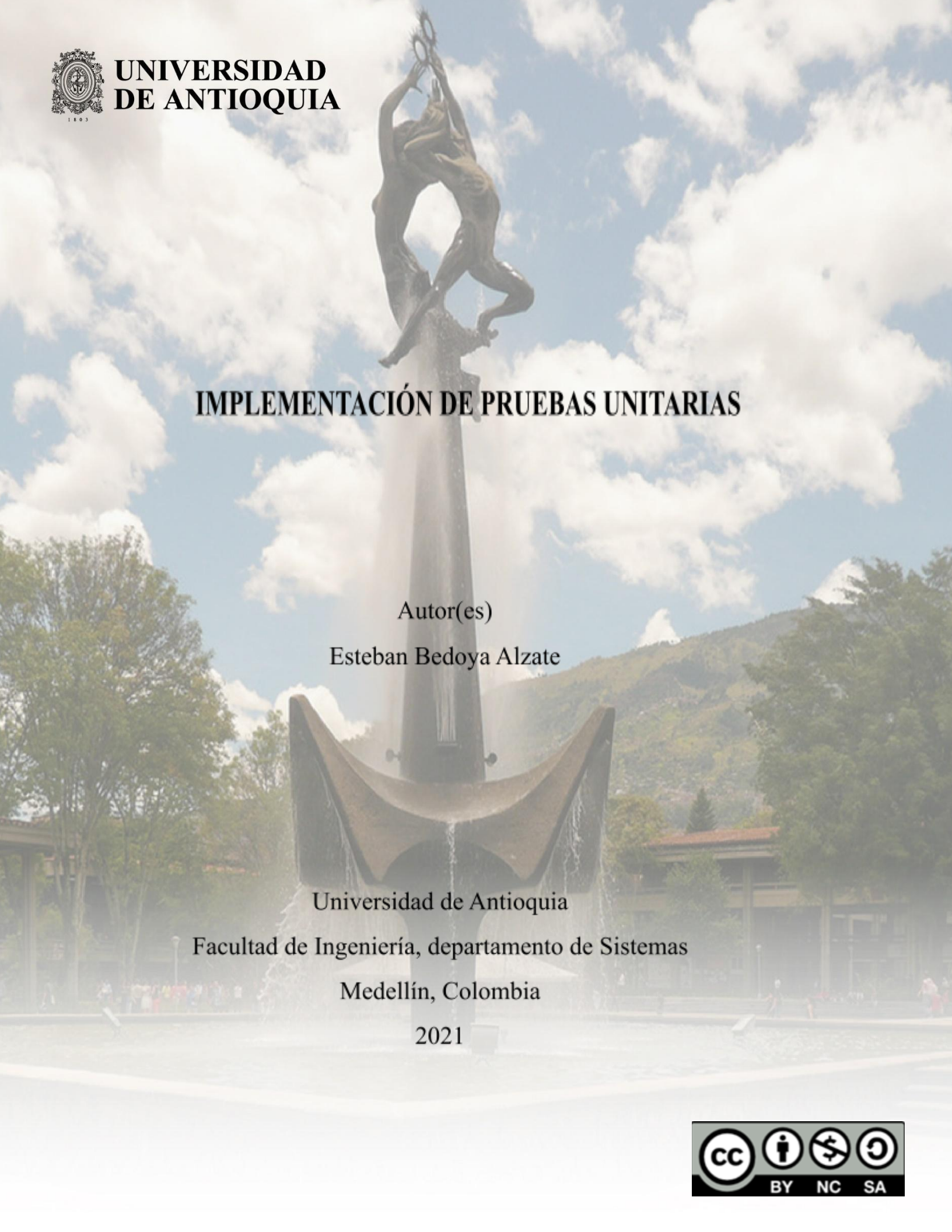


UNIVERSIDAD DE ANTIOQUIA

IMPLEMENTACIÓN DE PRUEBAS UNITARIAS

Autor(es)

Esteban Bedoya Alzate

Universidad de Antioquia

Facultad de Ingeniería, departamento de Sistemas

Medellín, Colombia

2021

IMPLEMENTACIÓN DE PRUEBAS UNITARIAS

Esteban Bedoya Alzate

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:
Ingeniería de Sistemas

Asesores (a):

Carlos Mauricio Duque Restrepo
Pablo Alejandro Giorgi

Universidad de Antioquia
Facultad de Ingeniería, departamento de Sistemas
Medellín, Colombia
2021

IMPLEMENTACIÓN DE PRUEBAS UNITARIAS

Resumen

En el informe se presenta el proyecto realizado durante los últimos 6 meses para la empresa Wolox[1], el proyecto estuvo enfocado en el desarrollo de software, más específicamente en la implementación de pruebas unitarias para un cliente externo de la empresa. Se decidió enfocar el proyecto en base a la pruebas unitarias debido a que generalmente se suelen omitir en algunos proyectos o realizar de una forma poco apropiada por culpa de las negociaciones, tiempos de entrega, falta de conocimiento o malas prácticas. Las personas no técnicas en los proyectos suelen ser poco conscientes del impacto positivo de estas además de que representa un tiempo adicional en los desarrollos, por esto prefieren dar prioridad en algunos casos a más funcionalidades o tareas de soporte de la plataforma en construcción que a las mismas pruebas unitarios.

Debido a esto y sus múltiples beneficios se realizó esta propuesta en la cual se logró llevar a cabo el proyecto impulsando está buena práctica tanto en desarrollos de frontend como backend, logrando alcanzar muy buenas coberturas de pruebas unitarias sobre los desarrollos propios, realizando también una limpieza de archivos de pruebas mal creadas y con errores de ejecución, así cómo fomentar el desarrollo de estas pruebas a medida que se realicen las funcionalidades para mantener una mejor práctica y descubrir errores en etapas tempranas de desarrollo.

Introducción

Las pruebas unitarias son la forma de comprobar el correcto funcionamiento de una unidad de código y hoy en día se convirtieron en una parte fundamental del desarrollo de software, ayudando a elevar la calidad del producto con beneficios cómo el de prevenir errores en etapas tempranas del desarrollo, legibilidad de código, confiabilidad en el producto, entre otros ahorrando tiempo y dinero en los proyectos. La mayoría de empresas de desarrollo de Software lo tienen implementado cómo parte de sus soluciones a clientes externos. Es por esto que WOLOX que es una compañía de desarrollo de software que provee soluciones tecnológicas para empresas en procesos de innovación digital cuenta con la necesidad de implementar las pruebas unitarias en sus distintos proyectos. En este contexto mi práctica está enfocada en el desarrollo de pruebas unitarias en medio de un proyecto externo de Wolox en el que se crearán nuevas funcionalidades a demanda del cliente. Con la ayuda de distintas herramientas cómo Jest para los servicios de Backend y Karma para el cliente frontend, además de la realización de prácticas recomendadas tales cómo el patrón AAA(Preparar, Actuar y Afirmar), aislar de factores externos a la porción de código que queremos probar, legibilidad en las pruebas y resultados consistentes, se implementaran las pruebas unitarias haciéndolo parte del desarrollo integral de la solución. Todo esto bajo la metodología ágil denominado SCRUM con la que logramos tener una mejor cohesión en el equipo y elevar la agilidad en los desarrollos conjuntos.

Objetivos

General:
Aumentar la calidad de las soluciones propuestas por la empresa, a través de la implementación de las pruebas unitarias.

Específicos:

- Adquirir conocimiento de frameworks enfocados a pruebas unitarias cómo lo son Jest con NodeJs para backend y Karma con Angular para frontend.
- Desarrollar pruebas unitarias de las funcionalidades que se implementen tanto en backend cómo en frontend con un nivel de cobertura por encima del 70%, con tecnologías cómo Jest, Jazmin y karma.
- Realizar pruebas con casos bordes y con posibles errores, sin omitir los casos normales del flujo en los distintos componentes.

Marco Teórico

Para el desarrollo de esta práctica qué se basa en la implementación de *pruebas unitarias* las cuales consisten en aislar una parte del código y verificar que éste funcione bien, estos se suelen realizar en la etapa de desarrollo por el mismo desarrollador que crea la funcionalidad, y se puede usar distintas metodologías para esto, en *TDD*(Desarrollo guiado por pruebas) se escribe primero la prueba unitaria y luego el código que hace que la prueba sea exitosa y así escribimos pruebas hasta lograr generalizar la solución para el desarrollo. A su vez existe la alternativa de crear las pruebas unitarias a medida que se desarrolla la funcionalidad asegurando que todas las pruebas sean exitosas al final del ciclo de implementación de la funcionalidad.

Las pruebas unitarias se suelen realizar por medio del *patrón AAA* [2] recomendada por la comunidad, consiste en una forma de escribir cada prueba en 3 partes:

1. ***Arrange:*** Inicializar, consiste en establecer los valores de los datos que vamos a utilizar en las pruebas.
2. ***Act:*** Actuar, realiza la llamada al método a probar con los parámetros preparados para tal fin.
3. ***Assert:*** Comprobar, revisar que el método ejecutado se comporte tal y cómo lo tenemos previsto.

Por medio de estos podremos describir distintas pruebas que no ayudarán a comprobar el correcto funcionamiento del código, para esto también

debemos tener en cuenta las distintas pruebas que podemos realizar entre esto se encuentran las *pruebas positivas* que se enfocan en verificar el uso correcto y sin errores del código o las *pruebas negativas* las cuales se basan en verificar los casos bordes o con errores que pueden aparecer en el uso de la aplicación.
En la práctica se estará enfocando en frontend con el framework para typescript de Angular[3] y en backend con nodeJS[4]. Para estas tecnologías existen distintos frameworks para realizar pruebas unitarias, por estándar de WOLOX se tiene definido las siguientes herramientas:

- ***Jest:*** Es un marco de prueba de JavaScript diseñado para garantizar la exactitud de cualquier base de código JavaScript. Permite escribir pruebas con una API accesible, familiar y rica en funciones que brindan resultados rápidamente.[5]
- ***Jasmine:*** Es una suite de testing que sigue la metodología Behavior Driven Development, está por defecto en los proyectos de angular.
- ***Karma:*** Es el test-runner, es decir, el módulo que permite automatizar algunas de las tareas de las suites de testing en el navegador, y se usa para ejecutar las pruebas en un entorno realista[6]

Metodología

En Wolox siempre se trabaja bajo la metodología SCRUM[7] el cual es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas(por ejemplo entregas continuas, retroalimentación de parte del equipo, adaptabilidad al cambio), se usa para trabajar colectivamente y obtener los mejores resultados posibles. Con una facilidad al cambio y una planeación constante de lo que se va a realizar durante un periodo de tiempo llamado sprint de 2 semanas en el que se tiene en cuenta distintas reuniones o también llamadas ceremonias cada una con un objetivo distinto que se muestra a continuación.

- ***Daily:*** Reunión realizada cada día en la que todos los integrantes del equipo muestran sus avances, impedimentos y lo que realizará el día actual para dar contexto al equipo y levantar alertas si son necesarias.

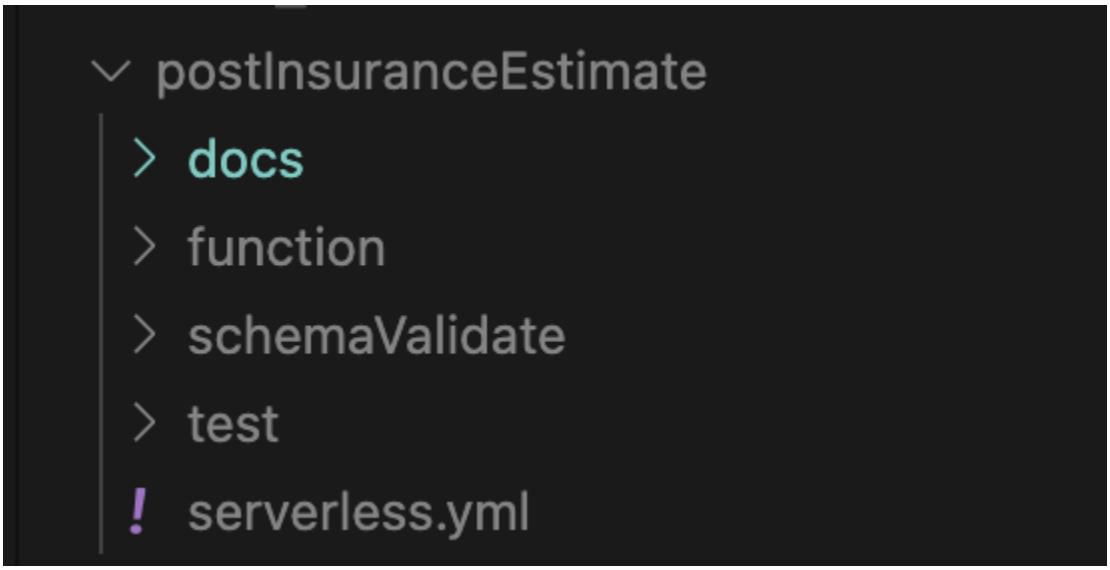
- ***Planning:*** Realizada al inicio del sprint en el que se planea todo el trabajo durante las 2 semanas, se estima todas las tareas y se deja a cargo de alguien cada tarea para que posteriormente éste la trabaje.
- ***Refinamiento:*** Está reunión realizada en medio del sprint anterior al que hará efecto se muestran las distintas tareas por parte de la dueña del producto y se revisan tanto a nivel técnico cómo de negocio hasta que estén listas para ser trabajadas.
- ***Retrospectiva:*** Se realiza siempre al final de cada sprint y tiene cómo objetivo revisar todo lo ocurrido en esas 2 semanas, verificar puntos positivos y negativos, y generar planes de acción a las oportunidades de mejora que se presenten en el sprint.
- ***Demos(Interna/externa):*** Esta ceremonia tiene cómo objetivo mostrar el trabajo desarrollado durante el sprint, puede ser interna para el equipo y externa para el cliente

Resultados y análisis

Durante los meses que he estado con la práctica académica he tenido la oportunidad de sumarme a un proyecto con un cliente externo de Wolox, éste proyecto se desarrolla con el framework Angular para el frontend y en backend se usa nodeJS, acorde a lo propuesto para la práctica se logró realizar desarrollos a la par con implementación de pruebas unitarias, en éste tiempo enfocadas en backend con Jest y en frontend con karma y jasmine.

BACKEND

El proyecto a nivel de backend tiene una arquitectura de serverless en el que por cada endpoint que se realice se requiere crear un función lambda con una única función y que tendrá una estructura cómo la que se muestra en la imagen:



En la carpeta de test se realizan las pruebas unitarias correspondientes a esa lambda, con un único archivo que tiene el nombre de la función y la extensión test.js para que pueda funcionar en los distintos pipelines por temas de configuración y del framework.

Cómo ejemplo de los test se muestran las pruebas unitarias realizadas a este servicio por mi parte.

```
Run | Debug
describe('postInsuranceEstimate', () => {      You, 6 months ago • ✨ Create post insurance es
  let token = {};
  let response = {};

  beforeAll(async () => {…
  });

  Run | Debug
  describe('error response if the body  are not present', () => {…
  });

  Run | Debug
  describe('success response with correct body sent', () => {…
  });

  Run | Debug
  describe('Should respond 409 because estimate has policy', () => {…
  });

  Run | Debug
  describe('Should respond 400 because estimate and stockId is not the same ', () => {…
  });
});
```

En éste servicio específico se tienen 4 "*describe*" que representan 4 casos de prueba entre casos exitosos y casos bordes con errores, en los cuales cada uno examina distintas partes de el caso por medio de los "*it*" cómo lo es el código de respuesta, el body de la petición, la respuesta general o el esquema del servicio.

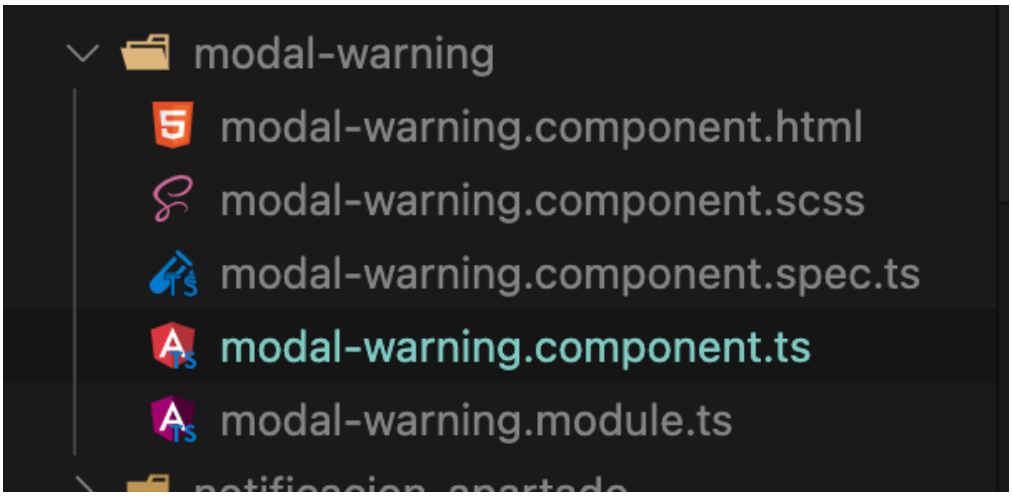
Además de la cobertura que nos muestra la herramienta con un porcentaje siempre mayor al 70%, en este ejemplo solo un archivo se baja el 100% por 5 líneas no cubiertas.

postInsuranceEstimate/function	100	83.33	100	100	
postInsuranceEstimate.js	100	83.33	100	100	5
postInsuranceEstimate/schemaValidate	100	100	100	100	
request.js	100	100	100	100	
response.js	100	100	100	100	

FRONTEND

En frontend se tiene cómo arquitectura la estándar definida por el mismo Angular a la hora de crear los test ya que que al generar por medio del CLI de angular un componente, servicio, módulo entre otros éste crea en una

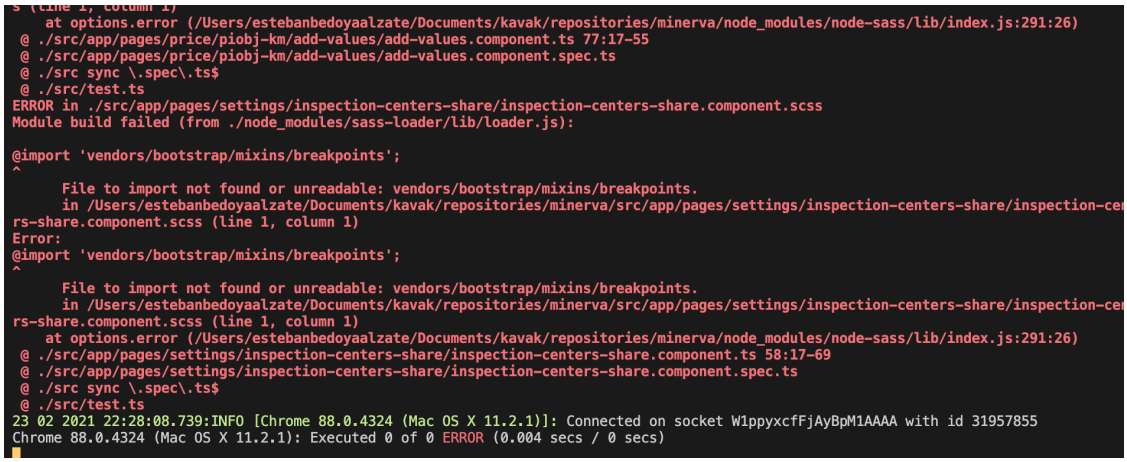
carpeta propia todos los archivos necesarios incluyendo un archivo .spec el cual contiene la configuración inicial del test y un ejemplo básico de cómo realizarlo con jasmine.



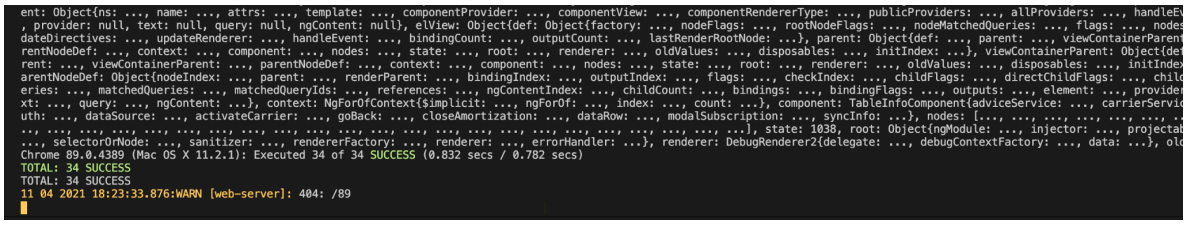
(Archivos generados por el CLI de Angular)

Ya que angular genera automáticamente los archivos de test unitarios nos genera un beneficio alto ya que crea una configuración inicial pero a la vez trae problemas ya que si no se trabajan las pruebas unitarias a medida que se desarrolla las funcionalidad estas quedan obsoletas y generan basura lo que ocasiona errores al correr los test.

Está problemática estuvo en el proyecto ya que se encontraron alrededor de 60 archivos de test que no funcionaban, por lo que se decidió eliminar todos estos y empezar desde cero con cada funcionalidad que se realizaba cómo se muestra en las siguientes imágenes.

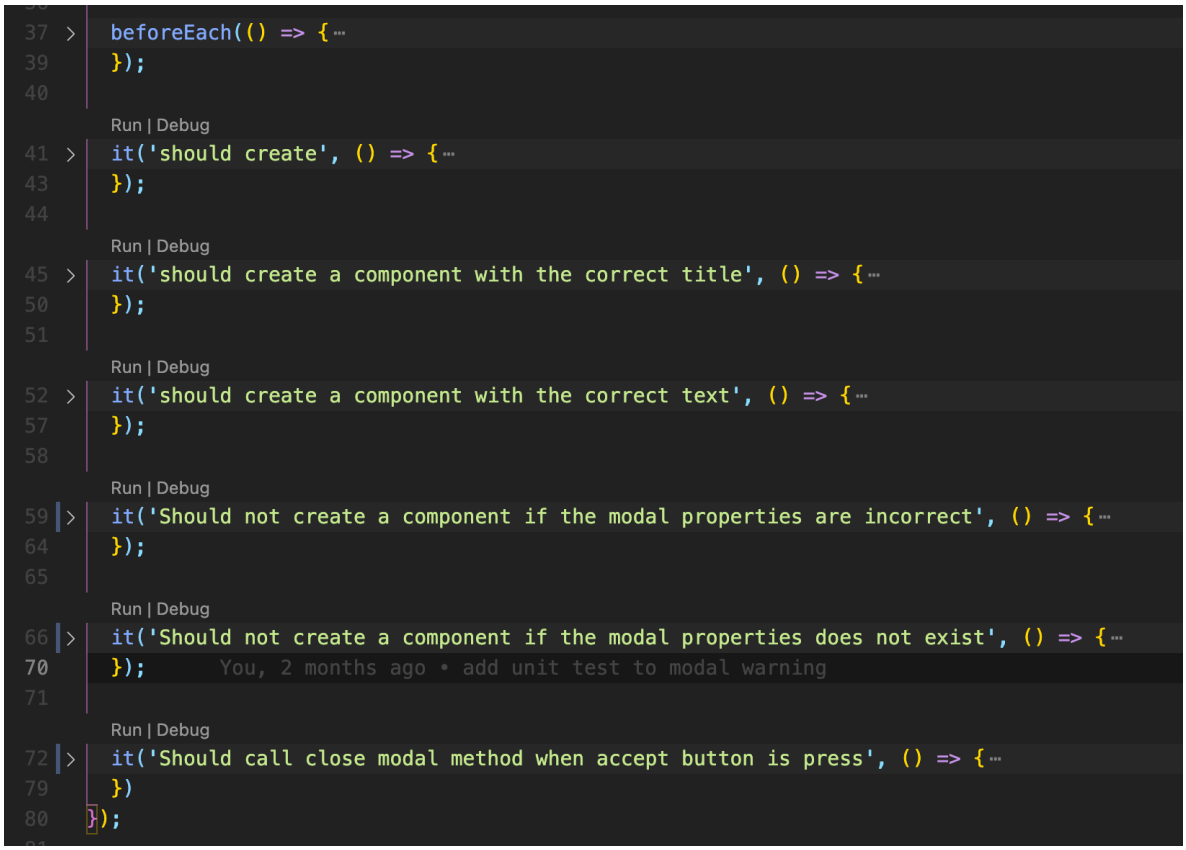


(Terminal luego de correr los test anteriores a los desarrollos realizados en la práctica)



(Terminal luego de correr los test posteriores a la realización de la práctica)

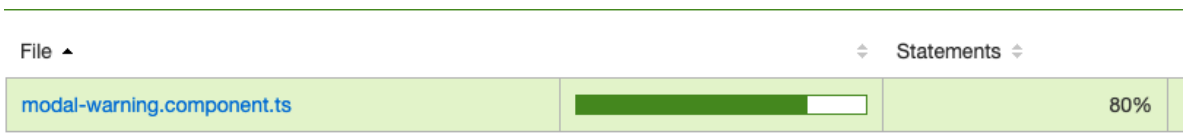
En frontend se realizaron test orientados tanto a la funcionalidad de los componentes y servicios cómo a la parte visual, verificando si el componente creaba de la manera esperada las diferentes partes que lo componían, podría ser un texto que mostrará en pantalla o simplemente una conexión con otro servicio.



(Ejemplo de test unitarios en frontend)

A lo largo del proyecto se logró realizar 34 test unitarios entre los que se probaron tanto casos bordes cómo casos exitosos, estos se hicieron para 6 funcionalidades distintas entre las que están servicios y componentes.

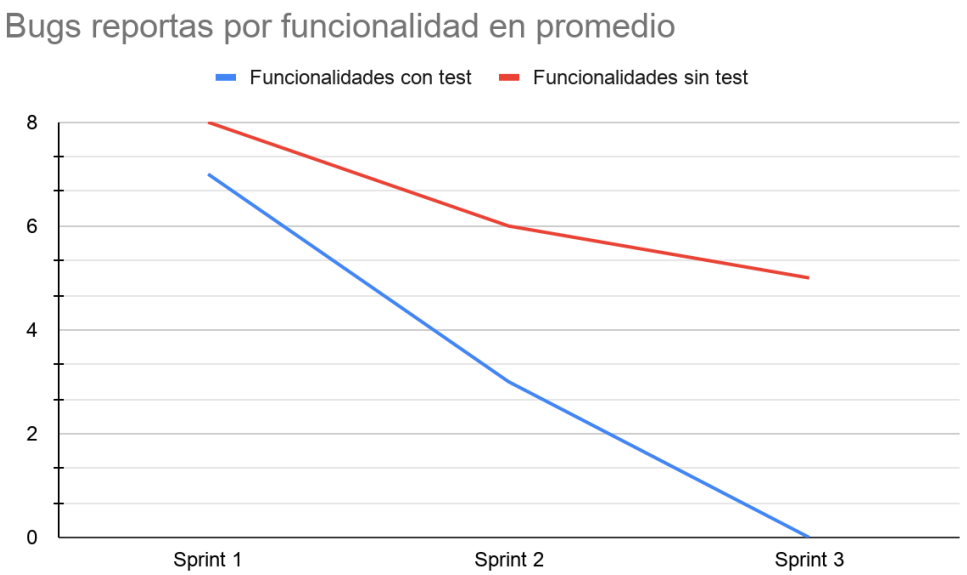
A la vez se tiene cómo ejemplo de la cobertura lograda para una funcionalidad en específico del componente "*modal-warning.component.ts*" el cual logró obtener un 80% de cobertura cómo se muestra a continuación.



(Cobertura de componente "*modal-warning.component.ts*")

Impacto de los test unitarios en el proyecto:

En el proyecto general se logró realizar pruebas unitarios tanto de frontend como backend pero no a todos los proyectos que conforman el ecosistema del cliente externo, es por esto que existen funcionalidades que están en repositorios en los que no se realizaron pruebas unitarias, debido a esto se puede comparar de forma sencilla los bugs reportados en promedio a medida que pasaban 3 sprints para las funcionalidades con test y las que no lo tenían, cómo se puede ver a continuación:



Conclusiones

- La metodología ágil es fundamental para el trabajo en equipo ya que uno cómo desarrollador puede estar involucrado en todos los procesos del equipo, dar retroalimentación de estos y tener

conocimiento del trabajo de los demás, lo que hace que exista una buena coordinación entre todos los integrantes del equipo.

- Las pruebas unitarias terminan siendo una parte muy importante en el desarrollo de software, por más que inicialmente aumente el tiempo de entrega de las funcionalidades permite a la larga ahorrarse tiempos en correción de bugs futuros, gracias a que a la hora de realizar modificaciones sobre estas funcionalidades las pruebas serán una guía para revisar si lo que cambiamos de una funcionalidad previamente construida no afecta otros procesos.
- En un mundo de desarrollo de software que apunta cada vez más a crear menos documentación escrita, las pruebas unitarias resultan ser una opción muy buena para documentar qué es lo que se quiere en cada componente de una aplicación, debido a que cada prueba tiene una descripción muy clara de cómo debería funcionar cada parte del proyecto.
- Las pruebas unitarias en frontend fueron un poco más costosas en tiempo a comparación con backend, esto debido a que particularmente para angular en los test se debe aislar cada componente de forma que solo dependa de éste simulando su funcionamiento normal, lo cual requiere una serie de configuraciones que en especial cuando se tiene poca experiencia se puede tardar cierto tiempo en realizarse a diferencia del backend con la arquitectura de serverless cada lambda está aislada de la otra y solo se necesita realizar mocks de la base de datos para lograr realizar las pruebas.
- Se debe tener cuidado a la hora de trabajar con angular ya que si no se realizan los test a la vez que construimos los componentes y servicios estos con el tiempo terminan siendo una deuda técnica muy costosa con todas las configuraciones iniciales que requieren las pruebas unitarias en frontend.
- En el proyecto se logró evidenciar la mejora en los bugs reportados a medida que pasaban los sprints, ya que por más que inicialmente se reportaba una cantidad similar de errores, cuando se solucionaban las funcionalidades con pruebas unitarias no se afectaba tanto cómo las funcionalidades sin test con los nuevos cambios que se agregaron.

Referencias Bibliográficas

[1] Wolox | Página principal de la empresa, Recuperado 20 Septiembre, 2020, https://www.wolox.com.ar/

[2] CODELAPPS. (2020). Anatomía de una prueba unitaria. [online] Available at: http://codelapps.com/code/anatomia-de-una-prueba-unitaria/.

[2] Angular Docs. (s.f.). Recuperado 20 Septiembre, 2020, de https://angular.io/docs

[4] NodeJS | Desarrollo web NodeJs, documentación. (s.f.). Recuperado 20 Septiembre, 2020, de https://desarrolloweb.com/home/nodejs.

[5] Jest. | Recuperado 21 Septiembre, 2020, de https://jestjs.io/es-ES/

[6] Cómo usar testing angular | Recuperado 21 Septiembre, 2020, de https://www.digital55.com/desarrollo-tecnologia/como-usar-testing-angular-jasmine-karma/

[7] Proyectos Ágiles. (2019). Qué es SCRUM. [online] Available at: https://proyectosagiles.org/que-es-scrum/.