Diseño y cobertura de pruebas de software para el backend.

Juan David Rivera Florez

Informe de práctica para optar al título de Ingeniero de Sistemas otorgado por UdeA

Asesor
Gabriel Darío Uribe Guerra, Matemático
Juan Alberto Zapata Zapata, Ingeniero de Sistemas

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas
Ingeniería de Sistemas
Medellín, Colombia
2023

Tabla de contenido

LISTA DE TABLAS

LISTADE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

Daily's	Reuniones diarias con el equipo de Ti, de duración de 30 minutos
HU.	Historias de usuario
Planning	Reuniones con todo los equipos del proyecto, donde se proponen las Hu al realizar.
Review	Reunión donde se revisa todas las hu planteadas al principio del spring
Ti	Equipo técnico
Sprints	Espacio de tiempo, destinado para las Hu's

RESUMEN

Las pruebas de software consisten en un proceso de evaluación y verificación hechas para apliciones con la intención de verificar que cada parte de sus funcionalidades cumplan con el objetivo deseado, además la implementación de estas pruebas aportan muchos beneficios a las aplicaciones algunos de estos beneficios son : prevenciones de errores, disminución de costos del desarrollo y mejora el rendimiento[1]. Sin embargo en el proyecto de Mis Aliados se encontró un baja cobertura de pruebas en el código del backend, lo que genera errores para el usuario final, pérdidas en las solicitudes finalizadas y disminución de la confiabilidad en la plataforma, por este motivo el siguiente trabajo busca diseñar y dar una cobertura de pruebas de software para el backend del proyecto Mis aliados; con la intención de prevenir errores, mejorar el flujo de trabajo de los desarrolladores y dar más confiabilidad a la plataforma. Para lograr esto se definió un conjuntos de reglas para el desarrollo de nuevas funcionalidades, se crearon las estructuras para las pruebas unitarias dando así una facilidad a las nuevas pruebas en el futuro, con ayuda del equipo se refactoriza el código pensado en el testing y se aumentó la cobertura haciendo pruebas unitarias a las funcionalidades actuales del backend. Como resultado se obtiene un sistema pruebas unitarias sólido y reglas de buenas prácticas que permiten la estabilidad de parte del Backend para el proyecto.

Palabras clave **— Pruebas de software, Backend, Java, Spring Boot, Testing, Arquitetura.**

ABSTRACT

Software testing is a process of evaluation and verification made for applications with the intention of verifying that each part of its functionalities comply with the desired objective, which gives many benefits to the applications that implement these tests such as error prevention, reduction of development costs and performance improvement[1]. However, in the Mis Aliados project a low test coverage was found in the backend code, which leads to errors for the end user, losses in the finished applications and loss of reliability in the platform, for this reason the following work seeks to design and provide a software test coverage for the backend of the Mis Aliados project; with the intention of preventing errors, improving the workflow of developers and giving more reliability to the platform. To achieve this, a set of rules was defined for the development of new functionalities, structures for unit tests were created, thus facilitating new tests in the future, with the help of the team the code was refactored with testing in mind and the coverage was increased by performing unit tests to the current functionalities of the backend. The result is a solid unit testing system and best practice rules that allow the stability of part of the backend for the project.

Keywords **— Software Testing, Backend, Java, Spring Boot, Testing, Architecture.**

I. INTRODUCCIÓN

Experimentality es una empresa prestadora de servicios de desarrollo de software cuyo objetivo es brindar soluciones eficientes y a la medida de las diferentes necesidades de sus clientes, que en su mayoría son empresas tales como Sura y Bancolombia. Mis aliados es un proyecto a cargo de Experimentality que nace en 2015 con la intención de crear una plataforma que genere conexión entre "aliados" quienes son personas expertas en diferentes áreas y ofrecen sus habilidades a clientes que las requieran, siendo de beneficio tanto para quien contrata como para el experto, garantizando que sea un servicio confiable y con calidad.

[2]Fig. 1. Evolución de Mis Aliados

Mis aliados es un proyecto complejo al igual que su Backend el cual se divide en tres: Service, Person y Adm. Estos son extensos y comparten una arquitectura hexagonal. Gracias a que el proyecto ha crecido con los años en paralelo, también lo hace el Backend y con esto se ha detectado una baja cobertura en su sistema de pruebas de software, lo que ha generado cierta inestabilidad e inconsistencia en algunos datos. El objetivo es aumentar esa cobertura, hallando los principales problemas tales como herramientas desactualizadas, códigos que no fueron pensados para pruebas y errores en la lógica que aún no han sido identificados; para así mejorar la estructura del proyecto y darle una alta cobertura en pruebas de Software que permitirán detectar problemas a tiempo y enriquecer futuros desarrollos.

Utilizando una metodología ágil [1]. con sprints de 11 días, acompañado de reuniones diarias de 30 minutos donde se informan avances y se propusieron actividades para los sprints con la meta de cumplir los objetivos planteados y estableciendo un ritmo de trabajo que logró obtener una estructura clara y definida para las pruebas facilitando así nuevas en el futuro del proyecto además de métricas y reglas para nuevos desarrollos logrando una mejor estabilidad y mantenibilidad dentro el proyecto Mis Aliados.

[1] Las metodologías ágiles son estrategias integrales que impulsan a las organizaciones a gestionar sus proyectos con rapidez y flexibilidad. Estas metodologías ayudan al desarrollo de los proyectos que las implementen.

II. OBJETIVOS

A. Objetivo general

Analizar, diseñar e implementar mejoras en el Backend de Mis aliados y aumentar su cobertura de pruebas lo que permitirá mejorar la calidad y eficiencia de la plataforma permitiendo así prestar un mejor servicio a los usuarios, además de proporcionarle al negocio una mayor estabilidad de la plataforma.

B. Objetivos específicos

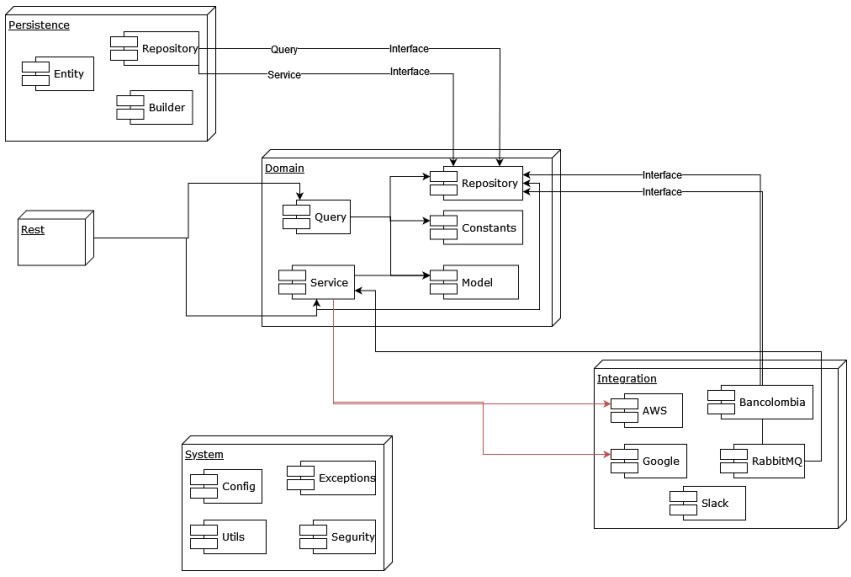
- Verificar el estado del backend del proyecto, leyendo su documentación y conociendo la tecnología en que se desarrolla.
- Analizar diferentes herramientas de testing que se acople mejor al código.
- Identificar las clases y modelos a los cuales será necesario hacerles un refactor para crear pruebas adecuadas.
- Definir reglas y buenas prácticas para los nuevos desarrollos.
- Crear constructores/builders basados en los modelos del Backend para la creación de pruebas.
- Crear pruebas Unitarias y Funcionales adecuadas para el proyecto, que se ejecuten cada que se haga un despliegue.

III. MARCO TEÓRICO

El proyecto inició como idea desde el 2015 y en el 2016 se lanzó la página “Su aliado” como inicio del servicio, el proyecto fue creciendo y se fue desarrollando. A mediados del 2017 cambió su nombre por Mis aliados y fue en ese momento que tomó más fuerza y logró seguir creciendo hasta la actualidad.

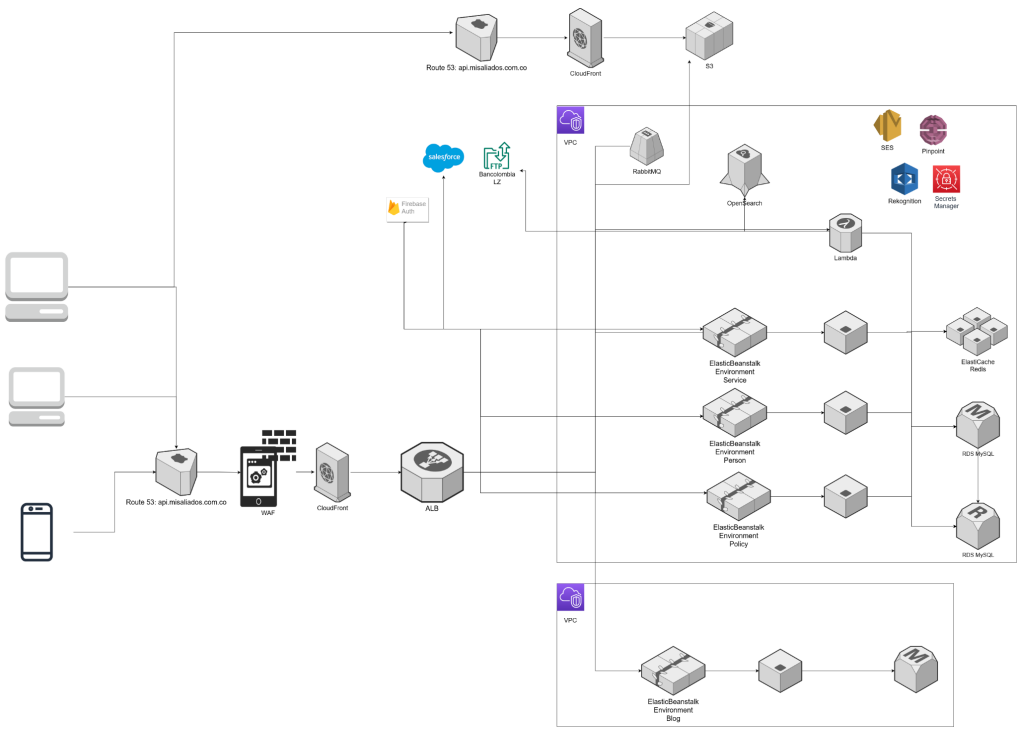
Todos los proyectos cuentan con una arquitectura de software que es un conjunto de patrones que se establece al inicio del proyecto con el fin de que los desarrolladores de software tengan una misma línea de trabajo, se cumplan los objetivos requeridos y se respeten las limitaciones dadas por los dueños del negocio[3]. Mis aliados cuenta con una arquitectura hexagonal que separa en varias capas el código para que cada capa tenga su propia responsabilidad, así cada capa estará desacoplada permitiendo que evolucionen de manera individual y aislada, dando la opción de que cualquier parte de la aplicación sea reutilizable.

El desacoplamiento de las capas nos da una mayor facilidad al momento del testing ya que se pueden hacer pruebas a cada capa de forma individual [4]-[5].



[6]Fig. 2. Arquitectura backend de Mis aliados

Mis aliados y su arquitectura están desplegadas usando la tecnología de AWS, con la intención de prestar a los usuarios una respuesta rápida y sin interrupciones. gracias a la arquitectura en la nube dada por AWS



[7]Fig. 3. Arquitectura Aws de Mis Aliados

TABLA I
TECNOLOGÍAS UTILIZADAS PARA LA IMPLEMENTACIÓN DEL PROYECTO

Tecnología/Lenguaje	Definiciones
Java	Java es una plataforma de lenguaje de programación de alto nivel creada por Sun Microsystems en 1995. [8], utilizada para la creación de miles de aplicaciones y servicios.
Spring Boot	Framework que nace de la dificultad que existía al momento de crear proyectos Java enterprise edition, ya que estos proyectos eran muy complejos y al momento de desplegarlos eran muy engorrosos. Spring Boot contiene una infraestructura más ligera que permite una fácil configuración [9].
MySql	MySql es el sistema de gestión de base de datos relacional [10].
Swagger	Swagger, herramienta que se encarga de la documentación de las APIs, actualizando dicha documentación junto a cada cambio desplegado[11].
Mockito	Mockito es un framework que permite realizar pruebas unitarias de manera sencilla y fáciles de leer.[12]
JUnit	JUnit es un framework que permite realizar testing en el lenguaje de programación Java de forma controlada. [13]

IV. METODOLOGÍA

Con el fin de completar con los objetivos planteados se realizó una capacitación en tecnología y arquitectura necesarias, así como la familiarización con la lógica del negocio conversando con las partes implicadas (Usuarios, aliados, equipo Mis aliados). Una vez comprendido el funcionamiento de la plataforma y la lógica de negocio, se utilizó una metodología ágil tipo scrum con sprints de 11 días y reuniones diarias (Daily's) de 30 minutos con equipo de Ti, donde se contextualizo más el proyecto de una manera más técnica y asignando tareas básicas para comprobar el conocimiento adquirido, así entrando al flujo de trabajo dentro de sprints.

Dichos sprints inician con el Planning donde todos los equipos en Mis Aliados se reúnen y hacen la asignación de Historias de Usuario (Hu) que deben ser terminadas durante el tiempo del sprint. Tres días antes del fin del sprint se harán Review con el equipo de Ti, los Po y las demás parte del negocio mostrando los resultados de las Hu asignadas en el Planning.

Ya hecho el análisis, se hace la propuesta donde se muestra el diseño y el plan requerido para cumplir con los objetivos al equipo de Ti con el cual se dialoga, se hacen propuestas de mejora y se aprueba dicha propuesta.

Teniendo en cuenta lo anterior en cada Planning se asignan Hu asociadas a la propuesta además de Hu requeridas por el negocio, tales como soporte o nuevas funcionalidades para la plataforma y en las Daily's se muestran los avances de dichas Hu's así como las dificultades encontradas y se da una retroalimentación a cargo del equipo, las implementaciones se realizan dentro el flujo normal del sprint. Este flujo de trabajo permitió un avance constante donde se identificaron errores a tiempo y se otorgaron resultados esperados por el proyecto.

A. Análisis

- Se realizó el entendimiento del modelo y lógica del negocio, para identificar los requerimientos o necesidades y así lograr su correcto diseño .
- Se analizó el backend por completo y así familiarizarse y facilitar el trabajo con el.
- Se efectuó un análisis del código existente.
- Se presentaron las primeras pruebas unitarias y la primera versión del diseño de pruebas para verificar con el equipo y dar el resultado esperado.
- Se llevó a cabo un desglose de actividades en sprints.

B. *Diseño de solución*

- Investigación de las herramientas disponibles para pruebas unitarias
- Investigación de herramientas o patrones para optimizacion del codigo
- Investigación de buenas prácticas de programación.

- Propuesta para la realización de refactorizaciones en el código existente, para obtener un código pensado para las pruebas.

C. Capacitación

- Fundamentación y refinamiento de conocimientos en Java, Spring Boot y en aplicaciones Rest para el manejo del backend e implementación de pruebas de software.
- Capacitación en herramientas como Mockito, JUnit, Swagger, MySql, servicios en la nube de AWS (Certificación en AWS), Spring Data JPA, Lambda, Python entre otros Frameworks y tecnologías.

D. Implementación

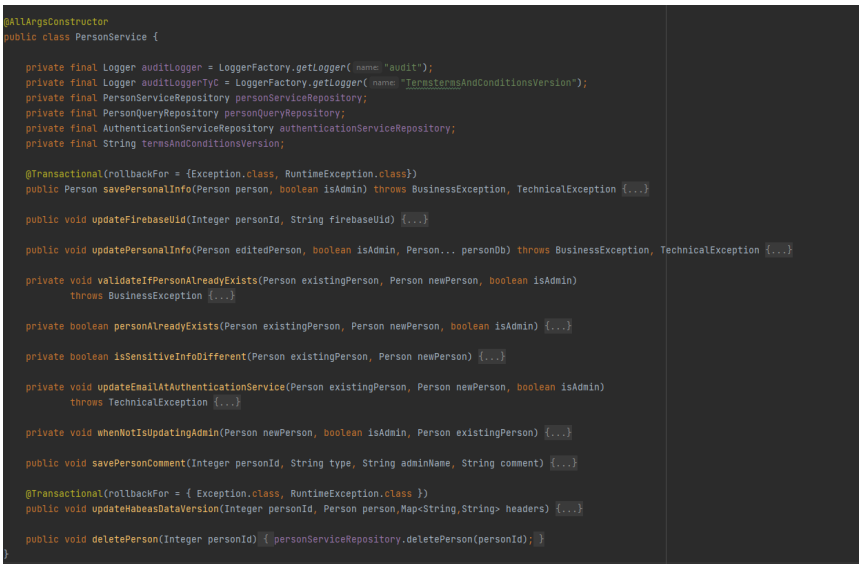
- Creación de documentación con reglas de buenas prácticas, pensada para futuros desarrollos.
- Creación de constructores (Data Builder) para futuras pruebas Unitarias
- Refactor de clases y modelos, para mejoramiento del backend
- Creación de Pruebas unitarias, con énfasis en los flujos más críticos

V. RESULTADOS

A continuación se mostraran los resultados que se obtuvieron gracias a las Hu realizadas y las fases ya antes mencionadas.

Se consultaron múltiples bibliografías y documentación con el fin de encontrar la mejor forma de implementar las pruebas unitarias, y así obtener los fundamentos primordiales para la construcción del diseño de prueba para un backend carente de estas, de este modo poder entregar resultados que beneficien al proyecto. La bibliografía va desde libros virtuales, videos, páginas web, repositorios universitarios y asesores, permitiendo que la curva de aprendizaje fuera la óptima para cumplir con los tiempos del sprint, y además coordinar e incorporar con el equipo de trabajo.

Durante las fases de desarrollo uno de los primeros avances/ resultados fue la refactorización del modelo de Person, que está conformado por modelo, service, query, rest y clases el cual es implementado en esta clase.

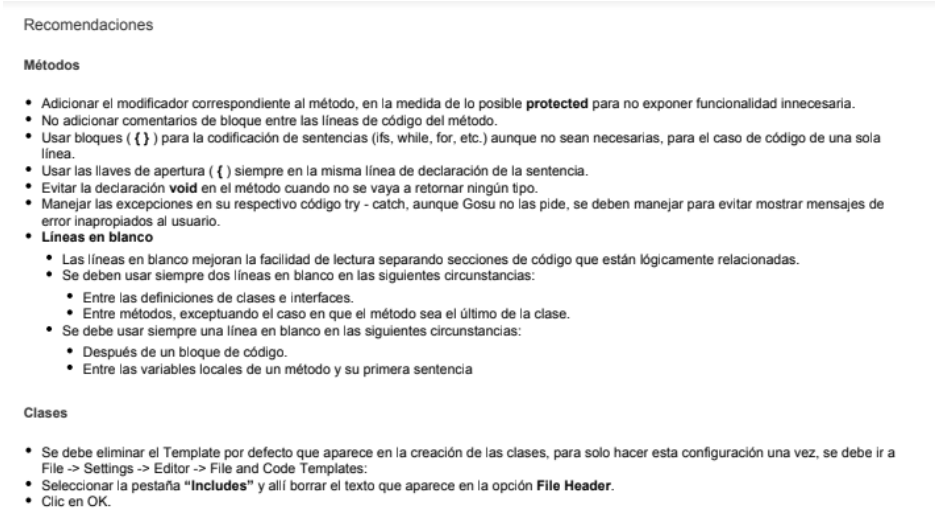


```
@AllArgsConstructor
public class PersonService {

    private final Logger auditLogger = LoggerFactory.getLogger( name: "audit");
    private final Logger auditLoggerTyC = LoggerFactory.getLogger( name: "TermstermsAndConditionsVersion");
    private final PersonServiceRepository personServiceRepository;
    private final PersonQueryRepository personQueryRepository;
    private final AuthenticationServiceRepository authenticationServiceRepository;
    private final String termsAndConditionsVersion;

    @Transactional(rollbackFor = {Exception.class, RuntimeException.class})
    public Person savePersonalInfo(Person person, boolean isAdmin) throws BusinessException, TechnicalException {...}

    public void updateFirebaseUid(Integer personId, String firebaseUid) {...}

    public void updatePersonalInfo(Person editedPerson, boolean isAdmin, Person... personDb) throws BusinessException, TechnicalException {...}

    private void validateIfPersonAlreadyExists(Person existingPerson, Person newPerson, boolean isAdmin)
            throws BusinessException {...}

    private boolean personAlreadyExists(Person existingPerson, Person newPerson, boolean isAdmin) {...}

    private boolean isSensitiveInfoDifferent(Person existingPerson, Person newPerson) {...}

    private void updateEmailAtAuthenticationService(Person existingPerson, Person newPerson, boolean isAdmin)
            throws TechnicalException {...}

    private void whenNotIsUpdatingAdmin(Person newPerson, boolean isAdmin, Person existingPerson) {...}

    public void savePersonComment(Integer personId, String type, String adminName, String comment) {...}

    @Transactional(rollbackFor = { Exception.class, RuntimeException.class })
    public void updateHabeasDataVersion(Integer personId, Person person,Map<String,String> headers) {...}

    public void deletePerson(Integer personId) { personServiceRepository.deletePerson(personId); }
}
```

Fig. 4. Refactor Person

Lo que se consiguió con esta refactorización fue dar una única responsabilidad, la cual fue que esta clase person sólo se encargará de hacer las operaciones que tengan que ver con ma_persons (Nombre en Bd) y así las clases hijas (Contractor, Independent, Aspirant) utilizaran los métodos de esta de manera más efectiva y dando más posibilidades al momento de crear pruebas unitarias.

El siguiente paso gracias al fruto de investigación y a la documentación de aliados como Sura fue proponer un estándar de nombramiento y buenas prácticas al programar , ya que el equipo ya contaba con estas prácticas se propuso dicho estándar con la intención de que el código permaneciera estable y entendible para cualquier parte del equipo de Ti además es algo de mucha utilidad para nuevos integrantes.



Recomendaciones

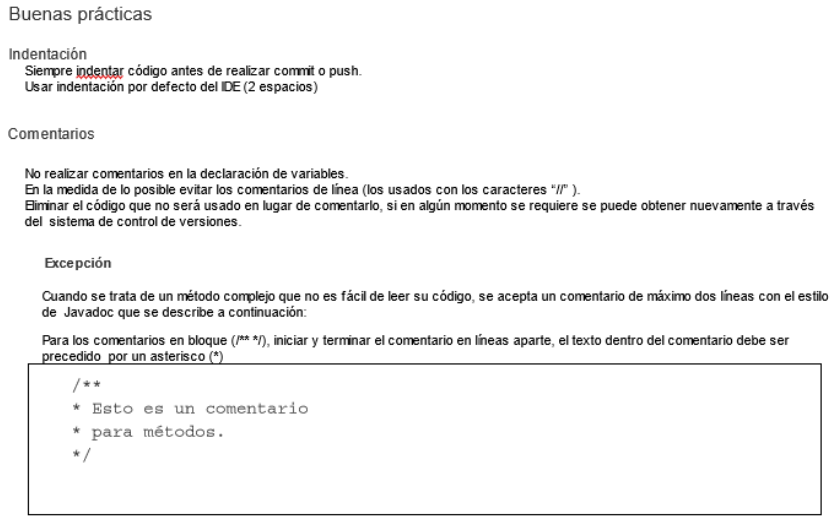
Métodos

- Adicionar el modificador correspondiente al método, en la medida de lo posible **protected** para no exponer funcionalidad innecesaria.
- No adicionar comentarios de bloque entre las líneas de código del método.
- Usar bloques (**{ }**) para la codificación de sentencias (ifs, while, for, etc.) aunque no sean necesarias, para el caso de código de una sola línea.
- Usar las llaves de apertura (**{**) siempre en la misma línea de declaración de la sentencia.
- Evitar la declaración **void** en el método cuando no se vaya a retornar ningún tipo.
- Manejar las excepciones en su respectivo código try - catch, aunque Gosu no las pide, se deben manejar para evitar mostrar mensajes de error inapropiados al usuario.
- **Líneas en blanco**
  - Las líneas en blanco mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas.
  - Se deben usar siempre dos líneas en blanco en las siguientes circunstancias:
    - Entre las definiciones de clases e interfaces.
    - Entre métodos, exceptuando el caso en que el método sea el último de la clase.
  - Se debe usar siempre una línea en blanco en las siguientes circunstancias:
    - Después de un bloque de código.
    - Entre las variables locales de un método y su primera sentencia

Clases

- Se debe eliminar el Template por defecto que aparece en la creación de las clases, para solo hacer esta configuración una vez, se debe ir a File -> Settings -> Editor -> File and Code Templates:
- Seleccionar la pestaña **"Includes"** y allí borrar el texto que aparece en la opción **File Header**.
- Clic en OK.

[14]Fig. 5. Estándar de nombramientos



Buenas prácticas

Indentación

Siempre indentar código antes de realizar commit o push.
Usar indentación por defecto del IDE (2 espacios)

Comentarios

No realizar comentarios en la declaración de variables.
En la medida de lo posible evitar los comentarios de línea (los usados con los caracteres "//").
Eliminar el código que no será usado en lugar de comentarlo, si en algún momento se requiere se puede obtener nuevamente a través del sistema de control de versiones.

Excepción

Cuando se trata de un método complejo que no es fácil de leer su código, se acepta un comentario de máximo dos líneas con el estilo de Javadoc que se describe a continuación:

Para los comentarios en bloque (/** */), iniciar y terminar el comentario en líneas aparte, el texto dentro del comentario debe ser precedido por un asterisco (*)

```
/**
* Esto es un comentario
* para métodos.
*/
```

[15]Fig. 6. Buenas prácticas

Este fue el punto inicial de los resultados donde además de implementar cambios en las nuevas funcionalidades, se hizo una socialización con el equipo, donde se concluyeron nuevas refactorizaciones del código y se planteó un plan para estos cambios, el cual consistió en que varias partes del equipo participaran en este momento. De manera individual se hizo el refactor de todo el backend encargado de la expedicion de polizas, que consistió en eliminar código obsoleto, estructura de carpetas dentro del repositorio, eliminación de clases de configuración utilizando herramientas de spring framework (`@Component`), creación de nuevas secrets keys para las variables de entorno y el cambio de nombre tanto en el repositorio como en la instancias de AWS.

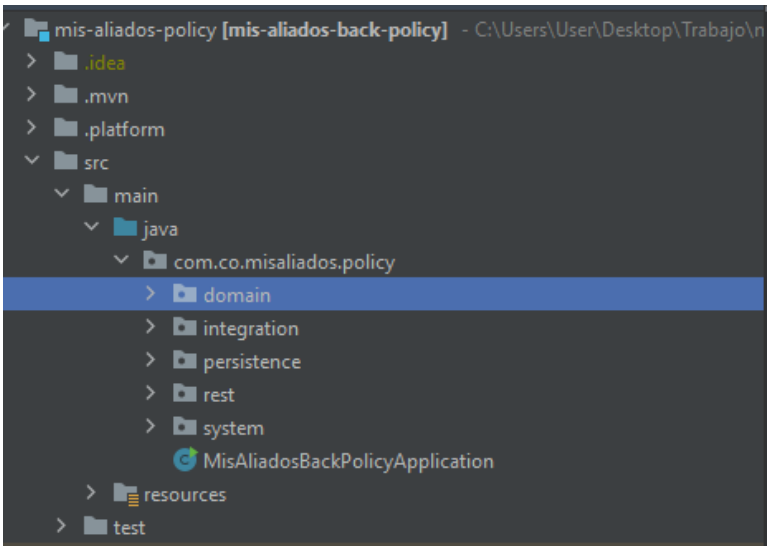


Fig. 7. Refactor de Policy

Como paso final, con el conocimiento del negocio se detectaron los flujos más importantes para el negocio, estos son las creación de solicitudes, la cotización de los independientes, creación de independientes y gestionamiento de información por parte de estos, aprobación automática de los aspirantes.

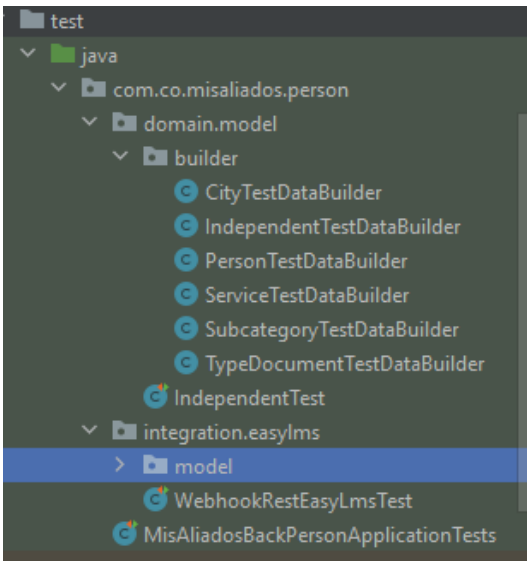


Fig. 8. Back de Person

Fig. 9. Independent Test

```
class WebhookRestEasyLmsTest {

    @Autowired
    private WebApplicationContext webAppContext;
    @Autowired
    private CertificationQuery personCertificationQuery;
    private MockMvc mockMvc;

    @BeforeEach
     void iniciarServicioParqueoTest() {...}

    @Test
     void WebhookRequestCourseApprovedTest() throws Exception {...}

    @Test
     void WebhookRequestCourseReprovedTest() throws Exception {...}

    @Test
     void WebhookRequestEvaluation() throws Exception {...}

    @Test
     void WebhookRequestExamReproved() throws Exception {...}

    @Test
     void WebhookRequestExamApproved() throws Exception {...}
}
```

Fig. 10. Aprobación de Aspirantes

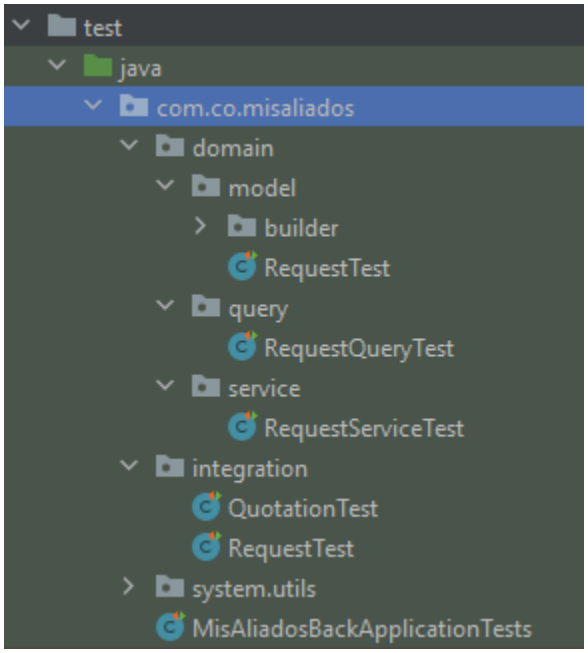


Fig. 11. Back de Service

```
class RequestTest {

    private Request requestTest;

    @BeforeEach
    public void init() { requestTest = RequestTestDataBuilder.requestWithoutIndependent(); }

    @Test
    void validateMinimumFieldsToRequestToDescriptionShort(){...}

    @Test
    void validateMinimumFieldsToRequestToDescriptionLong(){...}

    @Test
    void validateMinimumFieldsToRequestWhenItIsOnSiteAndTheCityIsNull(){...}

    @Test
    void validateMinimumFieldsToRequestServiceNull(){...}

    @Test
    void validateMinimumFieldsToRequestServiceModality(){...}

    @Test
    void validateMinimumFieldsToRequestCoupon(){...}

    @Test
    void validateMinimumFieldsToRequest() throws BusinessException {...}
```

Fig. 12. Request Test

```
class RequestQueryTest {

    @Mock
    private RequestQueryRepository requestQueryRepository;

    @InjectMocks
    private RequestQuery requestQuery;

    private Request request;

    @BeforeEach
    void setUp() throws BusinessException {...}

    @Test
    void getRequestByIdIsAdmin() throws BusinessException {...}

    @Test
    void getRequestByIdIsNoAdmin() throws BusinessException {...}
}
```

Fig. 13. Request Query Test

```
class RequestServiceTest {

    @Mock
    private  ServiceQuery serviceQuery;

    @InjectMocks
    private RequestService requestService;

    private Request request;

    @BeforeEach
    void setUp(){...}

    @Test
    void createRequest() throws BusinessException {...}

    @Test void createBuybackRequest() throws BusinessException {...}

    @Test
    void createPostServiceRequest() throws BusinessException {...}

    @Test
    void massiveRequestCreationAsync() throws BusinessException {...}

}
```

Fig. 14. Request Service Test

```
@SpringBootTest
@ActiveProfiles("test")
class QuotationTest {

    public static final int PERSON_ID = 2274;
    public static final String INDEPENDENT_TEST = "INDEPENDENT_TEST";
    Request request;

    Integer quotationId;

    @Autowired
    private RequestService requestService;

    @Autowired
    private QuotationService quotationService;

    @BeforeEach
    void init() throws TechnicalException, BusinessException {...}

    @Test
    void quotationQuotedByIndependent() throws BusinessException, TechnicalException {...}
}
```

Fig. 15. Quotation Test Integration

```
@Test
@DisplayName("BellezaSolicitudTest")
void createRequestWithCategoryBelleza() throws Exception {...}

@Test
@DisplayName("LocativosSolicitudTest")
void createRequestWithCategoryLocativos() throws Exception {...}

@Test
@DisplayName("TecnologiaSolicitudTest")
void createRequestWithCategoryTecnologia() throws Exception {...}

@Test
@DisplayName("FacebookSolicitudTest")
void createRequestWithCategoryTypeContactFacebook() throws Exception {...}

@Test
@DisplayName("WhatsAppSolicitudTest")
void createRequestWithCategoryTypeContactWhatsApp() throws Exception {...}

@Test
@DisplayName("VirtualSolicitudTest")
void createRequestWithCategoryVirtual() throws Exception {...}

@Test
@DisplayName("ContractorJustCreatedTest")
void createRequestWithContractorJustCreated() throws Exception {...}
```

Fig. 16. Request Test Integration

Estas son la pruebas unitarias y de integración que se hicieron en el momento de practicas, ademas de esto se completaron funcionalidades como creación de cupones de referidos para los contratantes, nueva gestión de cupones para cuando se califica un servicio, se limito el numero de cupones por la cédula del contratante, mantenimiento en el sistema de carga masiva, creación de reportes para el equipo de procesos, mejoramiento en sistema de errores de negocio, atención de soporte del aplicativo en despliegue lo que me dio una basta experiencia y buenos conocimientos en las herramientas como desarrollador backend y listo para otros retos como desarrollador.

VI. CONCLUSIONES

La evidencia que se presentó anteriormente demuestra que las pruebas en el código son una tarea muy importante y vital en el momento del desarrollo de software, que debe ser tomada muy seriamente y hecha junto a cada nuevo desarrollador de funcionalidades del sistema. Dejar esta tarea tan importante en el último lugar implica muchas complicaciones como código no preparado para pruebas y errores que se pudieron prevenir desde el desarrollo de la funcionalidad entre otras. Lo que significa para las empresas pérdida de dinero y problemas para el usuario final. Esto es parte de unas buenas prácticas de desarrollo y aunque el proyecto tenga una de las mejores arquitecturas posibles dejar que las funcionalidades no sean comprobadas por las pruebas de software deja pie a errores humanos, otro dato importante que se puede concluir con el desarrollo de este proyecto.

Después de aprender sobre las herramientas de Mockito y JUnit, se puede concluir que el inicio de nuevos desarrollos debe ser con una prueba unitaria que falla, para que al final del desarrollo esta sea aprobada, con esto sabremos que la funcionalidad cumple con su cometido.

Y para finalizar es importante recalcar que el estudio y preparación de todas las herramientas aun la más básica o lenguaje permite encontrar las mejores soluciones y la optimización del código haciéndolo más escalable y mantenible para cualquier persona que en un futuro tenga la necesidad de trabajar sobre ese backend.

REFERENCIAS

[1] "¿Qué son las pruebas de software y cómo funcionan? | IBM". IBM - Deutschland | IBM. https://www.ibm.com/co-es/topics/software-testing .
[2] Documentación interna Mis Aliados. (2020).

[3] H. Meyer. "Manufacturing Execution Systems: Optimal Design, Planning, and Deployment, 1st Edition".https://www-accessengineeringlibrary-com.udea.lookproxy.com/content/book/9780071623834/chapter/chapter7#/p2001776c9970121002.

[4]E.Salguero."Arquitectura Hexagonal". Medium. https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f.

[5] A. Cockburn. "Agile Software Development". Google Books. https://books.google.com.co/books?id=i39yimbrzh4C&printsec=frontcover&dq=Alistair+Cockburn&hl=es-419&sa=X&redir_esc=y#v=onepage&q=Alistair%20Cockburn&f=false.
[6]-[7] P. Gallego. Tecnología Mis aliados. (2022).
[8]"¿Qué es la tecnología Java y por qué la necesito?" https://www.java.com/es/download/help/whatis_java.html.

[9] M. V. Gonzalez. "¿Qué es Spring Boot?" Codmind. https://blog.codmind.com/que-es-spring-boot/.

[10] "Qué es MySQL: Características y ventajas". OpenWebinars.net. https://openwebinars.net/blog/que-es-mysql/.

[11] "API Documentation Made Easy - Get Started | Swagger". API Documentation & Design Tools for Teams | Swagger. https://swagger.io/solutions/api-documentation/.

[12] "Mockito framework site". Mockito framework site. https://site.mockito.org/.

[13] "JUnit 5 User Guide". JUnit. https://junit.org/junit5/docs/current/user-guide/.

[14]-[15] Sura. Estandares de Nombramiento y Buenas prácticas. (2022).