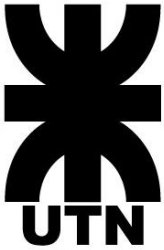

UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL PARANÁ

INFORME PRACTICA PROFESIONAL SUPERVISADA

Que para obtener el título de:

Técnico Superior en Programación

Presenta:

Federico Valentin Buchet

Profesor a Cargo:

Lic. Ernesto Zapata Icart

Paraná, Entre Rios Mayo,2024

I.ASPECTOS GENERALES:

1.1 Antecedentes:

La realización de la pasantía en la Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER), surge ante la necesidad de poner en practica lo aprendido durante el cursado de la Tecnicatura Superior en Programación, utilizando tanto los conocimientos teóricos para fundamentar nuestras decisiones, como los conocimientos prácticos para implementar dichas decisiones.

Debido al aumento de la demanda de profesionales altamente capacitados y con una vasta experiencia en el sector, debido a un mundo tecnológico que avanza a pasos agigantados, existe una gran motivación a la búsqueda de oportunidades que permitan la reinserción en el campo, por tal motivo la realización de la pasantía en la Caja de Jubilaciones de Entre Ríos(CAJAJPER), surge como respuesta a dicha demanda, dando el puntapié inicial para el crecimiento técnico , personal y profesional de cualquier interesado en el desarrollo de software.

1.2 Objetivos:

1.2.1 Generales:

El objetivo general de la realización de la pasantía en la Caja de Jubilaciones y Pensiones de Entre Ríos es poder brindar a cualquier estudiante a punto de egresar de la carrera de Tecnicatura Superior en Programación, una experiencia práctica que nos permita integrar los conocimientos teóricos y prácticos adquiridos durante el cursado de la carrera en un entorno real. Fomentando el desarrollo y crecimiento de dichos conocimientos, dar un mejor entendimiento de la forma en que se desarrolla software y adquirir las habilidades necesarias para poder insertarse de forma correcta en el mercado laboral.

1.2.2 Específicos

1.Dominio de tecnologías actuales: Poner en práctica y adquirir un conocimiento profundo de las tecnologías y herramientas empleadas en el desarrollo de software, ya sean herramientas de versionado (git, github, gitlab), como así también los lenguajes y sus frameworks (Java-Spring, JavaScript-React).

2.Participacion activa en proyectos de la empresa: Colaborar de manera activa y plena el diseño, desarrollo e implementación de los distintos proyectos que la empresa necesite para su correcto funcionamiento.

3.Mejorar el trabajo en equipo: Mejorar las habilidades de trabajo en equipo al dividir las tareas en partes mas pequeñas para poder resolver mejor y mas eficientemente los distintos

requerimientos, compartiendo ideas y opiniones sobre distintas formas de resolver un problema.

4. Puesta en práctica de los conocimientos del ciclo de vida del software:

Iniciando desde la obtención de los requerimientos, el análisis de los mismos, diseño del software que brinde una solución a dichos requerimientos, creación e implementación del mismo.

5.Gestion eficiente del tiempo y los recursos con los que se cuenta: Mejorar las habilidades de gestión del tiempo, y recursos, priorizando la obtención de la solución con el menor gasto de recursos posible en el mejor tiempo posible, adaptándose a los cambios de requisitos que pudieran surgir y cumpliendo con los plazos acordados.

Estos objetivos serán los encargados de garantizar que los estudiantes adquieran las habilidades técnicas relevantes para el mundo moderno, como así también al mejoramiento de las habilidades blandas, las cuales son cada vez mas importantes en un mundo cada vez más interconectado.

1.3 Delimitación de la Práctica Profesional

La delimitación de la practica profesional se encuentra dada por los siguientes puntos, los cuales buscan dar claridad sobre el marco de actuación de la misma:

A) Enfoque tecnológico:

La pasantía se centra en el área de desarrollo de software y programación, las actividades y los proyectos realizados están íntegramente relacionados con estas áreas, excluyendo cualquier aspecto no relacionado a las mismas.

B) Responsabilidades:

Las responsabilidades del pasante se centran en las actividades de software realizadas en la Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER). Cualquier otra actividad o iniciativa externa o en proyectos no autorizados por la Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER) no se considerarán parte de la pasantía.

C)Colaboración con otras áreas de la empresa:

La pasantía no implica la colaboración con otras áreas de la empresa

que no estén relacionadas con el desarrollo de software, a menos que sea autorizado por el profesor que supervisa la misma o el supervisor de la pasantía en la Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER).

1.4 Limitaciones:

A lo largo de la realización de la pasantía en la Caja de Jubilaciones y Pensiones de Entre Ríos se dan ciertos factores que pueden afectar a la realización de la misma, dichas limitaciones son:

1)Disponibilidad de Recursos:

La limitación de los recursos técnicos como el hardware o software con el que se cuenta, podrían llegar a afectar en el tiempo de cumplimiento de ciertas actividades, como así también en la elección de ciertas tecnologías en detrimento de otras para el diseño de ciertos programas de software.

2)Cambio en el Organigrama de la Empresa:

Cualquier cambio en la estructura organizativa de la empresa o en el equipo de desarrollo podría afectar a los tiempos de entrega de software, requiriendo volver a determinar nuevos plazos de entrega, como así también definir nuevas formas de trabajo en equipo.

3)Restricciones de Confidencialidad:

Ciertos proyectos realizados, como es de esperar, pueden contener información confidencial y de vital importancia para la empresa, lo cual afectara a cierta información incluida en el informe, con el fin de respetar dicha restricción.

4)Cambios en los Requerimientos de los Proyectos:

Los requerimientos de los proyectos pueden ir variando a lo largo de el desarrollo de software, lo que afectaría de manera directa a los plazos previamente previstos de entrega del mismo, como así también al alcance del mismo, provocando una redefinición de los mismos con el fin de poder realizarlos de manera correcta y ordenada.

5)Limitaciones de Tiempo:

Factores como plazos estrictos para ciertos requerimiento o eventos no previstos con antelación, podrían afectar directamente a la ejecución de ciertas actividades y la participación del pasante en algunos aspectos de la pasantía.

6) Interferencias externas:

Eventos imprevistos e interrupciones externas que afecten al normal desenvolvimiento del pasante en la realización de la pasantía.

Estas limitaciones son las que podrían surgir durante el desarrollo de la pasantía, las cuales deberán ser tenidas en cuenta para minimizar su efecto en la realización de la misma.

II.EVALUACION INSTITUCIONAL

2.1 Descripción General de la Institución (Empresa):

La Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER), es la institución encargada de gestionar y administrar las jubilaciones y pensiones de los trabajadores y empleados de la provincia de Entre Ríos y toda aquella información relevante para la correcta administración de las mismas.

2.2 Objetivos de la Institución:

- Administrar de manera eficiente los recursos financieros para asegurar la sostenibilidad del sistema jubilatorio
- Llevar a cabo recopilación y análisis de datos estadísticos para realizar previsiones a futuro y estudiar procesos.
- Otorgar las prestaciones(beneficios) necesarios para los trabajadores que habiendo llegado a determinada edad y cumpliendo con el tiempo de servicio laboral requerido, les correspondiere acceder a los mismos.

2.3 Visión:

Ser conocida por ser la institución encargada de administrar las jubilaciones y pensiones, brindando acceso a todos los beneficiarios de la información pertinente, garantizando una correcta transparencia con la información.

2.4 Misión:

La Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER) tiene como misión administrar de manera transparente y eficiente los recursos destinados a todos los beneficiarios de la Caja, como así también la información de los mismos.

2.5 Actividad Institucional:

La Caja de Jubilaciones y Pensiones de Entre Ríos (CAJAJPER) tiene como tarea principal administrar los fondos de todos los beneficiarios de la misma, como así también la realización de todos los tramites, informes, recibos que requiriesen los distintos beneficiarios.

2.6 Estructura Organizacional:

La Caja de Jubilaciones y Pensiones de Entre Ríos esta dividida en distintas áreas según su tarea a desempeñar, las cuales son: Presidencia, la cual se encuentra en lo mas alto de la jerarquía, luego encontramos el área de liquidación, tesorería, recursos humanos, informática y el área de mesa de entrada, esta última encargada de realizar los trámites y brindar la información pertinente para la obtención de los distintos beneficios.

2.7 Descripción del grupo humano:

El grupo humano con el cual desempeño mis tareas esta conformado por profesionales muy capacitados en el área de la Informática y Sistemas, entre los cuales se encuentran Técnicos Superiores en Programación, Analistas en Sistemas, Licenciados en Sistemas, Ingenieros Informáticos y Contadores. Los cuales demuestran cada día una gran preparación para el trabajo que realizan, como así también una gran calidad de habilidades blandas, las cuales son cada vez mas importantes en el mundo actual.

2.8 Descripción del Escenario de Trabajo:

En el área en el cual realizo mis actividades (Informática), realizo el mantenimiento e implementación de aplicaciones de software, garantizando la utilización de las mejores prácticas, como así también la correcta seguridad de los datos.

Para esto mismo realizo el relevamiento de determinados requerimientos tanto de mis superiores, como así también de los distintos organismos estatales que trabajen en estrecha cooperación con la Caja.

III. Desarrollo de la Práctica Profesional:

3.1 Desarrollo de las Actividades Desarrolladas en la Práctica Profesional:

Con el objetivo de poder adaptarnos de la mejor manera posible al entorno y contexto en el que trabajamos, como así también comprender el vocabulario adecuado y reglas de negocio necesarias para poder desempeñarnos de la mejor manera, empezamos con una introducción a los distintos sistemas utilizados en la Caja. Descubriendo y analizando las distintas tecnologías y herramientas utilizadas para llevar a cabo los distintos objetivos.

El primer programa informático con el que me tope fue un a aplicación realizada en el Lenguaje de Programación Cobol, el cual siguiendo los estándares actuales, es de bajo nivel, esto produjo que comprender que hacían sus instrucciones no pareciera tan complicado si se agarrara dichas instrucciones de manera aislada , pero pensando las distintas partes de la aplicación como un todo trabajando en conjunto dificulta su entendimiento , ya que diversas sentencias , hoy en día denominadas mala practica por los eruditos de la programación, como lo puede ser el famoso “GO TO” el cual es utilizado para mover la ejecución del programa de arriba abajo , llamando a funciones o código escrito en otras partes , complicaba el entendimiento del código , ya que la ejecución no parece seguir un orden lógico a primera vista para el entendimiento de un programador que se encuentre actualmente de alto nivel.

Además hay otras cuestiones a tener en cuenta a la hora de trabajar con Cobol, como son el hecho de que hay que especificar prácticamente todos los tamaños de las variables, largo en el caso de las cadenas y números, y cantidad de ceros después de la coma en el caso de los decimales, como así también el hecho de que el cambio en la longitud de los valores como los números puede causar que todo el programa explote o que simplemente no funcione, pone una gran dificultad a la hora de su mantenibilidad y escalamiento. Como asi también la poca información relevante que se puede llegar a encontrar en internet, puede hacer de este lenguaje un verdadero dolor de cabeza.

Y como si esto no pareciera bastante grave, dicha aplicación posee alrededor de 16000 líneas, lo cual es relativamente poco conociendo lo que puede llegar a crecer un simple controlador en el lenguaje de programación Java su framework Spring, pero siendo un programa Cobol, si era muy grave, además de que no era la única en su tipo, sino que existen actualmente varias más de su tipo.

Sin embargo contamos con la ayuda de la persona encargada de la creación de dicho programa , un Senior en Cobol , con mas de 30 años de experiencia en el mismo , el cual maneja de manera muy profesional las distintas aplicaciones escritas en dicho lenguaje, pero dicho programador ya consciente de los problemas que tienen dichas aplicaciones y el alto costo que podría llegar a provocar dejar de mantenerlo y la alta dependencia del mismo que poseen dichos sistemas(otro problema relacionado a las aplicaciones antiguas), decidió de que era hora de realizar una actualización y migración de los sistemas .

Para tal motivo empezó a recopilar la distinta información que los programas reciben y generan, para posteriormente ser migrados/actualizados.

Sin embargo, mi encuentro con dicha aplicación solo fue de manera pasajera al inicio de la pasantía, y como un camino a llegar en el futuro a medida que me iba interiorizando en las reglas de negocio y el funcionamiento de las formas de trabajo del área Informática.

Luego de esto me fue encomendado, junto con otros 2 compañeros con los que realizo la pasantía (también estudiantes de la Universidad Tecnológica Nacional), realizar el mantenimiento de una aplicación PHP, algo que dio mas tranquilidad a la hora de trabajar con los sistemas, al ser PHP un lenguaje de alto nivel y más actualizado.

Dicha aplicación PHP había sido creada entre el año 2005 y 2006 y siendo actualizada y mantenida hasta los tiempos actuales, la misma estaba escrita en la versión 5.4 de PHP, dato importante a tener en cuenta más adelante, la misma estaba desplegada en un servidor interno de la Caja, nuestra tarea fue agregar la funcionalidad de realizar unos informes en PDF, en la que debíamos agregar la funcionalidad de generar unos reportes en la misma.

La tarea consistía en realizar una consulta SQL, obtener dichos datos con PHP y generar el PDF.

Esto me permitió acercarme mas a la forma en la que estaban estructurados los datos en la Base de Datos, dicha base de datos era MySQL, en su versión 5, esto mismo me permitió acercarme más al lenguaje utilizado propiamente por gente que comprende las reglas de negocio de las tareas realizadas dentro de la Caja de Jubilaciones y Pensiones de Entre Ríos, ya que las tablas y nombres de los campos describían de manera clara que guardaban y para que era utilizadas , ciertas palabras como “Haber”, “Retroactivo”, “Escalafón”, me acercaban cada vez mas al vocabulario utilizado dentro del contexto de la Caja.

Me encantaría dar más información para especificar un poco más sobre el contexto de dichos nombres, pero comprendo que por temas de seguridad no puedo especificar demasiado sobre los distintos nombres de las tablas y campos que existen en los distintos esquemas de la Base de Datos, sin embargo, tratare de explicarme de la forma mas clara y brindando los ejemplos pertinentes para garantizar su entendimiento.

Volviendo a la generación del reporte PDF, una vez que había tenido el primer contacto con MySQL y las distintas tablas de la misma, solo quedaba comprender que tabla o tablas, como así también que campos necesitaba para el mismo, para esto debía seguir comprendiendo la forma en la que estaba estructurada la información y como se relacionaban las tablas.

Esto tuvo su dificultad, pero no me llevo tanto tiempo , ya que generalmente las Bases de Datos Relacionales como lo son MySQL o PostgreeSQL, en utilizacion de sus distintos entornos de trabajo como puede ser PhpMyAdmin , PgMyAdmin o DBeaber , le permiten al usuario encargado de su administración poder visualizar de manera graficar el DER(Diagrama Entidad Relacion) de la Base de Datos , en este caso PhpMyAdmin era el utilizado para la Base de Datos MySQL de esta aplicación , una vez visualizado el DER , solo quedaba anotar las tablas y sus relaciones para las consultas , como en la Tecnicatura Superior en Programación habíamos hecho uso de distintos DER en la materia Metodología de Sistemas, además de que en la Diseño y Administración de Bases de Datos nos enfocábamos en llevar a la práctica dicho DER, no me resulto demasiado complicado comprender las relaciones.

Una vez encontrada las tablas , entendiendo las relaciones , comprendiendo los resultados que debía obtener , surgió un problema para el que no había documentación, ejemplo o mención alguna en ningún libro , video o sitio de internet referido al ámbito de bases de datos .

Dicho problema era el hecho de que habían creado unas tablas cuyos nombres eran fechas, esto me dificulto mucho hacer consultas que comprendieran varios años , ya que para saber en el rango en el que se encontraran determinados resultados debería hacer un recorrido por todas las tablas , lo que crearía una consulta gigante que uniera todos los resultados de cada tabla , algo verdaderamente ineficiente , pero que según mi tutor , lo había hecho un antiguo programador de la empresa , justificándose de que de esa forma los datos ocuparían menos espacio y seria mas eficiente almacenarlos y consumirlos , que si estuvieran en tablas separadas, algo que no posee absolutamente ningún sentido para cualquier persona que haya dedicado un tiempo a la comprensión del funcionamiento de las tablas relacionales y su forma de indexación.

Para solucionar el problema de la manera más rápida posible, aunque quizá la menos eficiente, decidí hacer uso de los Procedimientos Almacenados de MySQL, algo odiado por muchos, querido por otros, gracias a su difícil mantenibilidad en caso de que el encargado de codificarlos no sea prolijo y su poco conocimiento por muchos programadores backend , algo que ciertamente no es muy positivo ya que muchas veces, como en este caso , el procedimiento no solo que ahorra código , sino que es más eficiente a la hora de ejecutarse, ya que en el caso de MySQL, cuando escribes una consulta SQL dentro de un procedimiento almacenado , el mismo guarda en cache el mejor camino de ejecución para devolver el resultado indicado, de forma que no debe volver a encontrar dicho camino , algo que aumenta su velocidad.

El procedimiento que codifique, se encargaba de unir a través de un bucle while, todas las consultas a las tablas indicadas, es decir, realizaba una consulta a una tabla con nombre fecha, aplicando el filtro indicado, y luego unía el resultado con un UNION ALL, porque los resultados iguales eran también importantes, a la siguiente tabla con nombre fecha, así hasta la última.

Algo que no había mencionado hasta este entonces pero que es de vital importancia, es el hecho de que el usuario que necesitaba el informe PDF, iba a enviarme además de los filtros de búsqueda de un formulario, el rango de fechas que necesitaba, es decir me pedía unos datos filtrados desde tal punto a tal punto, algo que gracias al uso del procedimiento almacenado de MySQL pude resolver fácilmente a través de los parámetros del procedimiento.

Por ultimo pero no menos importante , el programador encargado en su momento de crear las tablas que poseían el problema de los mapeos , también había agregado a cada elemento de las tablas , un atributo fecha , es decir ,si la tabla se llamaba ‘2023-12’ , había un atributo que tenía ‘2023-12-01’ , un campo que causaba una repetición innecesaria y que poco aportaba a la tabla , ya que la misma era bastante explicita con su contenido y cualquier programador que viera la tabla , sabría que la misma contiene algún valor de esa

fecha. Sin embargo, dicho campo aporto algo bueno y es el hecho de evitar tener que “hardcodear” el campo fecha por cada unión y que pueda ser visualizado en el informe de manera clara.

Una vez solucionado el problema de las consultas y habiendo obtenido un resultado que había sido validado por mi superior, tenía que empezar a realizar el llamado del mismo con PHP.

En dicho sistema PHP, ya existían consultas similares a las del procedimiento, pero con código de programación, pero yo decidí no hacerlas con el mismo y delegar la responsabilidad de las mismas y su resultado al procedimiento, ya que al estar PHP siendo utilizado sin ningún framework como Laravel, el cual separa responsabilidades de Frontend y Backend , el código PHP se había vuelto el denominado código Espagueti , el cual posee tanto código de Frontend , como código de Backend , dificultando el entendimiento del mismo y su mantenibilidad al estar todo junto.

Esto justifico la decisión de mover toda la lógica de la búsqueda de datos al procedimiento, y PHP solo debería obtener los datos del formulario y enviarlos. Esto nos permitió disminuir mucho código, y facilitar el entendimiento y mantenibilidad del código PHP, al quitarle responsabilidades.

Por último, una vez que PHP hubiera recibido los datos del formulario, los hubiera mandado en la llamada al procedimiento almacenado de MySQL, luego PHP hubiera guardado los datos que devuelve la llamada en una variable, simplemente quedo generar el PDF con dichos datos y subir los cambios.

Luego de terminada dicho requerimiento, fueron surgiendo otros sobre la marcha, los cuales también requerían consultas sobre las tablas con nombre fecha, lo cual, como es de imaginar, fue resuelto también con los procedimientos almacenados y el uso de funciones.

He de mencionar el hecho de que mi conocimiento de los procedimientos almacenados viene de la parte de la materia Diseño y Administración de Bases de Datos de la Tecnicatura Universitaria en Programación , algo que me permitió estar mas familiarizado con dichas estructuras , y no solo eso , poder tener en mente su uso en las situaciones que me fueron surgiendo , algo que no habría ocurrido de ser de otro modo , y que habría dificultado mi performance a la hora de resolver ciertos requerimientos , ya que no habría podido entre otras cosas separar responsabilidades en la arquitectura de la aplicación , ni optimizar la forma de obtener resultados a la Base de Datos.

Luego de pasado un tiempo mi superior me mostro que, en determinadas fechas, la aplicación PHP, poseía una serie de pasos que se ejecutaban de forma manual y asincrónica en determinadas fechas.

Dichas series de pasos y tareas, eran encargadas de utilizar ciertos datos que generaba el programa mencionado al inicio de este informe, hecho en el lenguaje de programación COBOL, y volcarlo a la Base de Datos MySQL.

Toda la ejecución de las tareas, las cuales era más de 30, tardaba alrededor de 3 horas, pero de las cuales, 2 horas y media, se iban en la ejecución de una sola tarea, algo que llamo mucho mi atención, ya que seguro había una forma de optimizarla.

Dicha tarea se encargaba de la inserción de alrededor de 1000000 de datos en una tabla con nombre fecha que era creada cada mes, por tal motivo se me ocurrió proponer migrar todas las tareas de migración a un lenguaje diferente y mas escalable y mantenible, el cual no solo seria utilizado de forma solitaria, sino en combinación con todo su ecosistema de herramientas, dicho lenguaje de programación era JAVA y su framework Spring.

De realizarse dicha migración, desde PHP a JAVA, no solo disminuiríamos los tiempos de realizar la migración de datos desde la aplicación Cobol hacia la base de datos MySQL, que he de mencionar que los datos de Cobol se guardan en cientos de archivos .txt, ya que el mismo no puede o al menos no era utilizado para conectarse a una Base de Datos como MySQL, de ahí el intermediario de PHP y su utilización para el llenado de la Base de Datos.

Volviendo a JAVA y Spring, la justificación de su uso, venia dada principalmente para separar responsabilidades, ya que ya no habría más archivos de código que tuvieran responsabilidades de Frontend y Backend, ya que con Spring realizaríamos todas las tareas relacionadas al campo del Backend, y la librería de React en el lado del Frontend.

Logrando de esta forma, separar totalmente las responsabilidades de cada campo, de manera que cualquier futuro programador que viniese a realizar tareas de mantenimiento a la aplicación, sabría que encontrar en qué lugar y como encontrarlo. Aplicación entendida como una parte Backend en JAVA con Spring y una parte Frontend en JavaScript con React

Aquí he de hacer hincapié en el hecho de que cuando realice la propuesta de actualización, mi superior me brindo apoyo total e incondicional, ya que el confiaba tanto como yo que dicha actualización del sistema seria buena para toda la empresa, además de que debido al buen trabajo que había realizado con los requerimientos que habían ido surgiendo hasta ese entonces había demostrado ser capaz de realizar propuestas como esta y llevarlas a cabo.

A su vez, uno de los mantenedores de los servidores Caja de Jubilaciones y Pensiones de Entre Ríos, al enterarse de que llevaríamos a cabo una actualización de los sistemas, me sugirió además actualizar la Base de Datos utilizada por PHP, la cual era MySQL en su versión 5.0.

Aunque la actualización me parecía bien, decidí recomendar una base de datos PostgreSQL, debido no solo a que era mejor y más fácil de manejar en cuanto a la creación y manejo de Procedimientos Almacenados , llamados Funciones en PostgreSQL , no existiendo distinción entre Procedimientos y Funciones como en el caso de MySQL , sino

también por su gran cantidad tipos de datos , como también a su gran velocidad de inserciones, algo muy importante para el nuevo sistema, además de que era una base de datos con la que me manejaba muy bien debido a que era con la que había trabajado en la materia Diseño y Administración de Bases de Datos de la carrera.

Dicha recomendación fue bien tomada y una Base de Datos PostgreSQL en su versión 16.0, fue subida a uno de los servidores para su posterior utilización.

Ahora solo quedaba empezar a definir de qué forma se migraría la parte Backend del sistema PHP a JAVA con Spring.

Para esto, primero tuvimos que definir el esquema de MySQL, en PostgreSQL, pero con algunas diferencias ya que había nombres de columnas que podrían llegar a perder a un administrador que no conociera las relaciones.

Un ejemplo podría ser cuando tienes una tabla que tiene un id_per , y esta definida una tabla persona y otra permutaciones , la abreviación per podría hacer alusión a cualquiera de las 2 si no se hiciera uso del contexto de la tabla que posee el id , por tal motivo ,para ciertas columnas se decidio poner el nombre completo , en este caso quedaría id_persona o id_permutacion , dependiendo de a que nos quisiéramos referir.

Ademas de esto también descubri que habría mas de un esquema con tablas con nombre fecha , por tal motivo decidí crear una sola tabla por cada esquema , que agrupara todos los registros de cada una , de esta forma el manejo de las mismas sería más fácil y la resolución de los requerimientos venideros que implicaran el uso de dichas tablas serian resueltos más rápidamente.

Como dichas tablas eran verdaderamente grandes, mas de 190 millones de datos y subiendo, además de que su importancia era alta para las reglas de negocio, al ser consumidas muy seguido, había que tener en cuenta técnicas de optimización para que no se tardara mucho al consultar los registros de las mismas.

Para esto decidí utilizar los índices, algo que vimos en la Tecnicatura Universitaria en Programación, no tan en profundidad, debido principalmente a que su utilidad se observa de mejor manera en ciertos casos especiales, además de que los índices utilizados de mala manera pueden provocar una baja del rendimiento general las distintas consultas a las tablas de toda la Base de Datos.

Sin embargo, si se utilizan de buena manera no deberían ocurrir dichos problemas.

Dichos índices fueron utilizados para las columnas que tenían que ver con los INNER JOIN , es decir , las relaciones entre tablas , ya que siempre serian consultados a través de ahí, los índices debían ser ubicados de esa manera , la utilización de los mismos , aunque parezca poco , provoco una bajada drástica del tiempo de búsqueda de los registros en las tablas grandes , pasando de 31 segundos en la tabla mas grande , a 0.012 segundos luego de la utilización de los índices, algo verdaderamente increíble si se tiene en cuenta que el tiempo aceptable que nos habían dado como máximo para una consulta sobre las tablas , al menos por ahora , era de 5 segundos.

Ahora bien, una vez definida toda la estructura de tablas en PostgreSQL, había que realizar una migración de datos entre la base de datos MySQL, y la de PostgreSQL, para tal motivo, decidí utilizar 2 aplicaciones hechas con JAVA en su versión 17 y Spring Framework.

La primera de ellas se encargaría de manejar todos los datos de las tablas de MySQL,para esto tenia que definir todas las clases en la aplicación JAVA para poder manejar las tablas, pero como iban a ser solo datos que serian movidos entre aplicaciones y bases de datos decidí utilizar un record .Un record es similar al patron DTO utilizado por los distintos endpoints del Backend para devolver las entidades ,pero sin exponer todos su atributos , o devolver uniones entre tablas en una sola clase , es decir , podría devolver datos de las personas y sus distintas mascotas como un DTO , en la que tendría el nombre de la persona y una lista de nombres de sus mascotas , y no devolver una lista de personas con todos sus atributos(documento,id,fechanacimiento,etc), como asi tampoco todos los datos de sus mascotas(nombre,edad,id,raza,etc),como los DTO son utilizados para esto , su estado debería ser inmutable , pero como yo solo quería hacer uso de la inmutabilidad y ya había un tipo de estructura específica para esto en las nuevas versiones de JAVA , simplemente defini todas las tablas de la base de datos como records y no como entidades, ya que lo importante aquí era solo mover los datos de un lugar a otro sin modificaciones.

Para que se pueda entender la diferencia, una clase entidad se vería así:

```
public class Persona implements Serializable {

    private Integer id;
    private String nombre;
    private Integer edad;
    private Integer documento;
    private String emailUsuario;

    public Integer getId(){
        return this.id;
    }

    public String getNombre(){
        return this.nombre;
    }
```

```
public Integer getEdad(){
        return this.edad;
}

public Integer getDocumento(){
        return this.documento;
}

public Integer getEmailUsuario(){
        return this.emailUsuario;
}

public void setId(Integer id){
        this.id=id;
}

public void setNombre(String nombre){
        this.nombre=nombre;
}

public void setEdad(Integer  edad){
        this.edad=edad;
}

public void setDocumento(Integer documento){
        this.documento=documento;
}

public void setEmailUsuario(String emailUsuario){
        this.emailUsuario=emailUsuario;
```

```
}

@Override
public String toString(){
        return “Nombre:”+this.nombre+”,Documento:”+this.documento+
                “,Edad:”+this.edad+”,Email:”+this.emailUsuario;
}
```

Y un record se veria como lo siguiente :

```
public record PersonaRecord(Integer id,String nombre,Integer edad,Integer
        document,String emailUsuario){
}
```

Esto es algo que disminuye en gran medida el código y cumple con los requisitos que yo necesitaba , además de que internamente agrega todos los getters a los atributos, pero sin llamarlos getters , es decir , que si yo llamo a el nombre de un PersonaRecord , simplemente debería haver personaRecordInstance.nombre(), y eso me lo devolvería , algo que me ahorra todo el código de los getters , además de que al ser inmutables siempre , no define setters por debajo , y por ultimo también agrega una implementación por defecto del método toString en caso de ser necesario , aunque la misma puede ser sobreescrita si se quisiere.

Volviendo a la definición de records en la primer aplicación , una vez definidos todos los records para el manejo de todas las tablas de la Base de Datos MySQL, decidí agregar un repositorio por cada tabla hecho con JDBC, los cuales me devolvían los datos de las tablas , volcados en records , luego cree un Controlador con un solo endpoint que traía todos los datos por cada record, para que los datos puedan ser obtenidos desde la 2da aplicación.

El endpoint que exponía los datos de cada record se veía de una manera muy similar a esta:

```
@RestController
@RequestMapping(value=”/personas”)
public class PersonaController{

        private PersonaRepository personaRepository;
```

```
    @Autowired
    public PersonaController(PersonaRepository personaRepository){
        this.personaRepositoy=personaRepository;
    }

    @GetMapping()
    public ResponseEntity<Map<String,Object> getAllPersona(){
        Map<String,Object> map=new HashMap<>();
        List<PersonaRecord> personas=personaRepository.findAll();
        map.put("personas",personas);

        return new ResponseEntity(map,HttpStatus.OK);
    }
}
```

En este controlador simple, se exponía una ruta general a la que la otra aplicación accedería a través de un método HTTP GET, en la ruta localhost:8081/personas, recuperando el JSON que contenía a todas las personas.

He de recalar algo importante, y es que el no uso de la especificación JPA es algo que debería llamar la atención si ya no lo había hecho, y es que, si yo hubiera utilizado JPA, algo que es prácticamente un estándar en el ecosistema JAVA y también en el de Spring con Spring Data JPA, yo habría podido a través del mapeo de la clase con las diversas anotaciones que me permiten pensar en vez de una tabla de una base de datos , en un objeto y pensar en consultar a los objetos y no las tablas , el código habría sido quizás mas conciso y simple , además de más entendible .Pero hay que tener en cuenta la existencia de las tablas con nombre fecha , las cuales no me permitirían mapear todas las tablas debido a que si quisiera mapearlas , debería mapearlas 1 por 1 , y cada mes se agregaría una nueva con otro nombre fecha distinto, algo que me requeriría escribir una nueva clase por cada mes o modificar la ya existente , algo que no considere optimo ni algo bueno , por ese motivo se decidió utilizar JDBC , el cual realiza consultas SQL nativas y me permite consultar las tablas con nombre fecha de manera más fácil.

Por último, el uso de los Controladores y Endpoints para poder ser consumidos por otros Controladores , quizás no sea lo mejor para la comunicación entre aplicaciones de este

estilo , ya que eran tareas que se podrían ejecutar de manera asincrónica , y para esto sería mejor la utilización de herramientas de mensajeria como RabbitMQ o Kafka , sin embargo mi conocimiento de dichas herramientas no era el mejor para el momento de la escritura de dichas aplicaciones , además del tiempo que tenía disponible para la creación de ambas aplicaciones , no me lo permitirían ,por tal motivo se decidió que serían comunicadas a través de OpenFeign , y en un futuro si se pudiera , migradas a RabbitMQ o Kafka.

Luego de haber terminado con la creación, y prueba de la primera aplicación para verificar su correcto funcionamiento, había que empezar la codificación de la que sería quien consumiría los datos y los insertaría en la Base de Datos PostgreSQL.

Para la 2da aplicación , defini el dato que de devolvía cada endpoint de la primera aplicación como un record, luego tuve que definir su entidad en la base de datos postgree , para esto si se utilizo JPA con su implementación Hibernate , la cual me permitió mapear cada entidad a la tabla correspondiente.

Una entidad se vería como lo siguiente :

```
@Entity
@Table(name=”personas”)
public class Persona implements Serializable{

    @Id
    @GeneratedValue(strategy=GeneratedValue.IDENTITY)
    private Integer id;
    private String nombre;
    private Integer edad;
    private Integer documento;
    @Column(name=”email_usuario”)
    private String emailUsuario;

    //GETTERS
    //SETTERS
    //TOSTRING
```

}

Nótese el uso de la anotación @Table para especificar la tabla a la que hace referencia la clase y la anotación @Column para especificar el atributo al que hacer referencia en la tabla, ya que los atributos en las Bases de Datos siguen una nomenclatura de snake_case, y el lenguaje JAVA utiliza lowerCamelCase para la definición de los atributos.

A continuación de dicho mapeo por cada tabla a entidad , había que crear su correspondiente Repository,Service y ServiceImplementation, todo esto para poder dividir la aplicación en distintas capas las cuales mejorarían la separación de dependencias y podríamos hacer uso de ciertos patrones como la inyección de dependencias en el caso de los repositorios y servicios.

Cada repositorio de las distintas entidades se vería como la siguiente interfaz:

```
public interface PersonaRepository extends JpaRepository<Persona, Integer>{
}
```

En la cual extendemos de JpaRepository , una interfaz genérica de Spring Data JPA , que brinda los métodos necesarios para el manejo de cualquier operación necesaria de cada entidad , entre los cuales se encuentran el método findAll() para obtener todos los registros , el método findById(Integer id) para obtener una entidad por id , el método void/Entity save(Entity entity) que guarda una entidad y la devuelve en caso de que nosotros lo indiquemos y el método deleteById(Integer id) que borra una entidad por id , entre otros.

El service de cada entidad se vería como lo siguiente:

```
public interface PersonaService {
	List<Persona> findAll();
	Optional<Persona> findById(Integer id);
	Persona save(Persona persona);
	void deleteById(Integer id);
}
```

Luego necesitamos de su implementación para poder hacer uso de la inyección de dependencias :

```
@Service
public class PersonaServiceImpl{

	private PersonaRepository personaRepository;

	@Autowired
	public PersonaServiceImpl(PersonaRepository personaRepository){
		this.personaRepository=personaRepository;
	}

	@Override
	public List<Persona> findAll(){
		return this.personaRepository.findAll();
	}

	public Optional<Persona> findById(Integer id){
		return this.personaRepository.findById(id);
	}

	public Persona save(Persona persona){
		return this.personaRepository.save(persona);
	}

	public void deleteById(Integer id){
		this.personaRepository.deleteById(id);
	}
```

}

Aquí definimos la implantación de cada método del service, además encapsulamos la lógica de los métodos de obtención de datos de la Base de Datos, de forma de que desde el exterior solo será visible el service y no el repositorio.

También hacemos uso de la inyección de dependencias por constructor con el @Autowired, además de anotar nuestra implementación del servicio como @Service para que sea cargado como un componente de Spring y que podamos inyectar nuestro service en un futuro.

Una vez definido esto y habiendo empezado a probar la migración de algunas tablas y sus inserciones, en búsqueda de errores y optimizaciones , observe que había algo que había pasado por alto y es el hecho de que las tablas de mas de 190 millones de datos tardarían muchísimo tiempo en migrarse a través de JPA , ya que si obtuviera por meses los datos de las tablas por nombre fecha y los fuera insertando en una sola utilizando un bucle for uno por uno tardaría mucho y probando el método saveAll(List<T> list) también definido por la interfaz JpaRepository obtenia datos no muy buenos a la hora de medir las inserciones por tiempo , en el que 5 millones de inserciones me tardaban alrededor de 20 minutos , había algo que podía mejorarse , investigando con uno de los administradores de los servidores de Base de Datos llegamos a la conclusión de que había 2 problemas uno era el hecho de que el disco utilizado por PostgreSQL no era de estado solido(algo que disminuiría la velocidad de escritura) , y otro era la tasa de transferencia , el primer problema no era mejorable según los requisitos que me habían mencionado , por tal motivo había que mejorar el problema de la tasa de transferencia, algo que me permitió descubrir que el método saveAll() a través de los logs de la traza del método que te permite visualizar Hibernate , realiza el envió e inserción de un registro 1 por 1 algo que provocaba un alto tiempo de respuesta, una vez reconocido el problema , se me ocurrió volver al uso de una de las herramientas más útiles y queridas a lo largo de mi pasantía , el uso de Procedimientos Almacenados.

Como es de conocimiento general entre los programadores especializados, la gran velocidad de los Procedimientos Almacenados tanto para las actualizaciones como inserciones masivas, lo convierten en la herramienta ideal para el trabajo con grandes volúmenes de datos dentro de la misma base, por ello supe desde un primer momento que si el problema estaba en la tasa de transferencia desde el lenguaje a la Base de Datos , debía mandar todos los datos juntos y que el Procedimiento Almacenado dentro de la Base de Datos los insertara de manera masiva,de esta forma ya no habría problemas con la tasa de transferencia y el procedimiento almacenado se encargaría de realizar el trabajo.

Una vez determinada la herramienta para resolver el problema , empecé a codificarlo,lo cual fue una experiencia bastante enriquecedora , ya que nunca había realizado un procedimiento almacenado que insertara datos masivamente.

Luego de esto estaba el desafio de llamar al procedimiento desde la aplicación que realizaría las inserciones en PostgreSQL, el problema aquí fue principalmente el hecho de que llamar al procedimiento desde la aplicación Java con Hibernate estaba siendo imposible.

Habia probado diversas formas de llamar a distintos procedimientos almacenados y de distintos tipos, procedimientos que devolvían datos , otros que no devolvían, procedimientos que recibían parámetros , otros que no los recibían , y aun asi ninguna de las diferentes formas parecía funcionar con el procedimiento.

El procedimiento almacenado , recibía una lista de la tabla a insertar, insertaba los datos de la lista , y devolvía la cantidad insertada , aquí el problema era que Hibernate por algún motivo no permitia enviar una lista por parámetros del procedimiento , debido a esto , decidi buscar una herramienta para poder realizarlo.

JOOQ

La herramienta que logre encontrar para tal motivo fue la biblioteca JOOQ, la cual , según su propia documentación , fue creada para trabajar a la par de los distintos ORM, ya sean Hibernate, TopLink, EclipseLink y etc.; y no para competir con ellos o reemplazarlos .

La misma consistia en una nueva forma de realizar consultas a Base de Datos , bastante similar a realizar una consulta SQL normal , pero de forma mas programática con Java, permitiendo validar en tiempo de compilación los parámetros enviados a la base de datos , algo que JDBC no es capaz de ofrecer actualmente.

Dicha herramienta me fue de bastante utilidad, ya que una vez que agregabas su dependencia al proyecto Spring y compilabas el mismo , la misma mapeaba todas las tablas de la base de datos, en una especie de tablas virtuales dentro de la aplicación, donde estaría definida cada tabla en forma de clase, procedimiento que también era realizado con los distintos procedimientos almacenados.

Esto me permitió definir el tipo que recibía el procedimiento como una clase en Java, ahora solo tenía que crear un arreglo del mismo y enviarlo.

Cosa que hice y que permitió aumentar los tiempos de inserciones masivas, pasando de 5 millones de registros en 20 min., a 5 millones de registros en 10 segundos. Una diferencia verdaderamente enorme considerando que debía realizar la inserción de alrededor de 190 millones de registros.

Hecho esto, ya estaban todos los pasos para poder realizar la migración de datos desde de MySQL, hacia PostgreSQL, ahora solo quedaba seguir mapeando las tablas restantes, y definiendo algún que otro procedimiento para las inserciones masivas en otras tablas que lo requiriesen .Ya que no realizaría un procedimiento almacenado para insertar tan solo 60000 datos en una o varias tablas.

Algo que pude percibir mientras realizaba las inserciones masivas, era como los segundos aumentaban cuando la tabla ya estaba mucho mas llena, segundos que serian mucho mayores si la tabla poseía muchas restricciones en sus campos, algo previsible, pero interesante de observar de cara a la comprensión y manejo responsable de una Base de Datos, no agregando índices ni restricciones porque si, sin justificación aparente.

FIN DE LA MIGRACION DE TABLAS

La migración de las tablas de un tipo de base de datos a otro, me permitió y le permitirá a cualquiera que tenga la oportunidad de realizarlo, comprender que no solo es mover datos de un lado a otro, es comprender las distintas relaciones entre tablas, las distintas restricciones de los campos de dichas tablas, el porqué de los datos de las tablas.

Como así también el poder ir aprendiendo distintas herramientas en el camino y profundizando en otras, en este caso lo fue el caso de los procedimientos almacenados, herramienta que ya conocía pero que pude no solo poner en practica nuevamente, sino que profundizar aún más en su funcionamiento, mejorando en su utilización y diferentes funcionalidades.

TERCERA Y ULTIMA APLICACIÓN

Esta tercera y última aplicación en la que tuve la oportunidad de participar, consistia en la migración de la funcionalidad de la visualización de datos desde la aplicación legacy PHP , hacia JAVA en su versión 17.

La aplicación PHP de este caso ,había sido realizada alrededor de los años 2007 y 2007, la misma llevaba en funcionamiento ya mas de 16 años , y había sufrido diversos parches y actualizaciones alrededor de los años, sin embargo , debido a que el equipo que en su momento la realizo no pudo llevar a cabo la implementación de diferentes arquitecturas y patrones de diseño para mantener el código estructurado de tal manera que fuera mas mantenible , la aplicación fue poco a poco decayendo.

La aplicación no solo contenia código spagueti , sino que sus archivos tampoco estaban estructurados de una manera en la que fuera mas fácil aislar la funcionalidad por modulos y mejorar su posterior mantenibilidad y extensión.

Por este motivo y el referente a que la versión utilizada por PHP era una bastante antigua , sumado al hecho de que PHP no es un lenguaje que mantenga retrocompatibilidad con versiones pasadas, algo que fue muy importante a la hora de definir el lenguaje para la nueva aplicación, fue suficiente para cambiar el esquema.

Para la nueva aplicación se elegiría JAVA en su versión 17 con Spring para el Backend , y JavaScript con su framework React para el Frontend.

Si bien para ese entonces ya existita una nueva versión LTS de JAVA (versión de soporte extendido), la cual era la 21, aun habían pasado pocos meses desde su lanzamiento, por tal motivo se decidio ir por la 17 y esperar a un futuro para su posterior actualización.

Para este proyecto trabaje con otro compañero que también realizada la pasantia junto conmigo, él se encargaría de realizar el frontend y yo del backend.

Dicho backend trabajaba de forma directa con la Base de Datos de PostgreSql que había sido migrada desde MySQL , por tal motivo pude reutilizar las entidades definidas en dicha aplicación , ya que serian las mismas.

Luego de creadas las entidades procedi a crear los distintos repositorios.Como los repositorios de Spring Data Jpa son siempre iguales en su definición, omitiré mas comentarios sobre los mismos.

Una vez realizado esto, había que migrar la funcionalidad de la aplicación PHP , donde no existía algo denominado Backend ni Frontend, ya que era un monolito , y además no utilizaba ningún framework web como lo es el caso de Laravel o Symphony.

Como yo ya comprendía la lógica detrás de los datos mostrados en la aplicación PHP, simplemente tuve que generar los distintos métodos en los servicios , para que estos devolvieran los datos necesarios para ser enviados al frontend.

El frontend o cliente , simplemente consumiría el Api Rest hecha con Java y Spring a través de sus distintos enpoints, obtendría los datos y los mostraria.

A modo de ejemplo , un enpoint simple , que solo realiza un listado de una determinada entidad , se ve asi :

```java
@RestController
@RequestMapping(value = "/anexos")
public class AnexoController {

    1 usage
    private AnexoService anexoService;

    no usages
    @GetMapping()
    public ResponseEntity<Map<String, Object>> getAll() {
        Map<String, Object> map = new HashMap<>();

        List<Anexo> anexos = anexoService.getAll();

        map.put("anexos", anexos);
        map.put("message", "listado de anexos");

        return new ResponseEntity<>(map, HttpStatus.OK);
    }
}
```

Aquí puede verse como la primer anotación define que será una clase que contendrá distintos enpoints, con la anotación @RestController, la misma es seguida por la anotación

@RequestMapping y una ruta con “/anexo”, dicha anotación es utilizada para definir una ruta general , la cual nos permite tener rutas hijas llamadas iguales , ya que si yo tuviera un endpoint con la ruta “/primero” por ejemplo, yo no podría tener varias rutas “/primero” en el resto de la aplicación ,sin embargo al utilizar la ruta general , esto si es posible , ya que podría tener una que sea /anexos/primero y otra /escalafones/primero, pudiendo reutilizar los nombres , y según el contexto darle un significado u otro.

Luego de la definición de la clase , puede verse como en el constructor se encuentra la anotación @Autowired, la cual inyectara el servicio para su posterior utilización , haciendo uso de la inyección de dependencias.

Donde dicho patron básicamente consiste,en el caso de Spring, en delegarle la responsabilidad de instanciar un determinado objeto, en este caso el del Service, donde el

contenedor de Spring creara diversas instancias para cada objeto anotado con @Autowired donde sea requerido.

Así mismo también puede observarse la anotación @GetMapping() y el método asignado a ella , en este caso dicha anotación no tiene asignada una ruta, pero cuando el cliente realice una consulta a la ruta "/anexos" del tipo Get , el método se ejecutara.

Dicho método devuelve un ResponseEntity , un objeto que nos permite definir la respuesta a devolver, en este caso yo utilizo un Map para poder retornar los objetos y no directamente el objeto List<Anexo> , ya que si en un futuro yo quisiera agregar mas datos a la respuesta por algún motivo , no me vea obligado a modificar el código. Esto nos permitiría no solo no modificar el código en el Backend, sino también en el Frontend,pudiendo expandir el método, pero no modificarlo, algo que es muy buena practica.

Muchos enpoints como el anterior con su respectiva clase fueron creados para este Backend, no muestro un ejemplo de código de la aplicación real, por motivos de seguridad y privacidad, pero a grandes rasgos se ven como el anterior.

Un punto importante a mencionar una vez habiendo migrado y realizado todas las consultas, como asi también habiendolas probado para verificar su optimización , fue el hecho de que PostgreSQL por algún motivo que actualmente desconozco , es mas lento en el uso de las subconsultas que MySQL, algo que fue importante haber encontrado a tiempo , ya que tuve que volver a realizar ciertas consultas que se habían vuelto mucho mas lentas debido a la forma de manejo del SGBD.

Sin embargo , considero que puedo estar haciendo suposiciones aceleradas al haber estado haciendo uso de dichas subconsultas en consultas relacionadas a las tablas mas grandes , algo que seguramente , aun poseyendo índices y utilizando los mismos para todos los JOIN , aumentaría la carga de tiempo de las consultas.

Algo que no sucedia en cuanto a la utilización de MySQL , ya que en dicha Base de Datos , las tablas no superaban los 1500000 millones de datos al estar separadas.

Cuando se termino de realizar casi toda la migración de la antigua aplicación PHP , también uno de mis superiores , mas epecificamente el que programaba en el lenguaje COBOL , me indico que debia realizar la migración de antiguos informes que se seguían realizando en dicho lenguaje , para que los realizara y acoplara a la nueva aplicación.

Esto fue con el motivo de seguir utilizando cada vez en menor medida las aplicaciones escritas en COBOL , como asi también la gran cantidad de archivos .txt que estas mismas utilizan como una especie de base de datos, ya que el mismo no puede conectarse a una , como si lo puede hacer un lenguaje como JAVA o C# entre otros.

Una vez terminada dicha aplicación , la misma ya era lo suficientemente grande como para que empiece a ser un poco tedioso encontrar algunos archivos , si bien se siguió una estructura de carpetas bien unificada , la cantidad de archivos por cada una había crecido en gran medida.

Por tal motivo , y viendo que a futuro dicha dificultad podría empeorar , se empezó a analizar la división de la aplicación en una arquitectura de microservicios, lo que permitiría tener quizás la misma división de carpetas de forma estructurada, pero con una menor cantidad de archivos, lo que aumentaría su mantenibilidad.

Ademas de esto , la arquitectura de microservicios , nos permitirá tener aplicaciones mas especificas para ciertos problemas , un caso podría ser una aplicación que solo genere determinados informes , ya sean en .pdf,.csv o .txt , mientras que otra solo se encargaría de la visualización de los datos.A su vez la aplicación de generación de informes también podría ser dividida en 3 , una especifica para cada tipo de informe , logrando una mayor mantenibilidad y especificidad en un problema.

Sin embargo , hay que tener en cuenta las otras cuestiones al pensar en este tipo de cambios arquitectónicos , ya que no solo implican un cambio en la arquitectura de las aplicaciones , sino también a nivel de la infraestructura de las mismas.

Es decir , cada aplicación al estar desplegada consume una determinada cantidad de recursos, al desplegar varias aplicaciones, los recursos consumidos por las mismas aumentarían , por lo cual implicarían también un mayor gasto para las mismas.

También hay que tener en cuenta la base de datos para la aplicación , cada aplicación deberá tener su propia base de datos o utilizarían todas la misma?.

Esta preguntas , sin duda , por mas simple que parezcan son de vital importancia a la hora de pensar el nuevo sistema , ya que un buen planeamiento y especificación de ciertas cuestiones importantes , nos ayudaran y ahorraran bastante tiempo en un futuro , además de permitirnos evaluar el mejor camino en base a los recursos y tiempo disponible , siendo este ultimo punto , uno de los mas importantes , ya que el cambio de la arquitectura también requerirá que se dedique una gran cantidad de tiempo a su creación , como así también al mantenimiento de los antiguos sistemas en caso de alguna falla o expansión mientras el nuevo sistema no está disponible.

Estas y otras cuestiones relacionadas a las ya especificadas están siendo evaluadas para analizar los problemas y beneficios del cambio de la arquitectura.

Mientras se evalua dicho cambio , y realizo mis aportes teóricos y técnicos , continuo expandiendo el Backend de la aplicación según van surgiendo los nuevos requerimientos de la misma.

Por ultimo , algo que quizás el lector se ha preguntado , es donde esta Git?,Como usan Git?,Porque no se menciona Git?.

Estas preguntas son muy importantes y para empezar , no es que Git se haya pasado por alto y no se haya tenido en cuenta , sino que ocurre la siguiente cuestión.

En la Caja de Jubilaciones y Pensiones de Entre Rios, el área encargada de validar la utilización de Git , es muy reacia a la utilización de herramientas como Git , ya que poseen un miedo desenfrenado a la filtración de datos importantes , en este caso el código que ellos consideran importante.

Si bien GitHub y GitLab son 2 herramientas ampliamente utilizadas como empresas como Netflix, Amazon,Microsoft,IBM,Oracle, etc, e implementa cada vez mas medidas de seguridad como puede ser la authenticacion con doble factor, esto parece no ser suficiente como para convencer de su uso e implementación en el ámbito del sector publico , debido al miedo de que por error se filtren datos importantes que afecten a la organización y su personal.

Esto ha provocado serios problemas a la hora de manejar código , ya que si quisiéramos trabajar en equipo , en el mismo código , tendríamos que estar compartiendo toda la aplicación , por la red local de la Caja de Jubilaciones y Pensiones de Entre Rios, para que otros miembros del equipo pueda acceder y realizar cambios en el mismo.

Para bien o para mal , yo solo me encargo del Backend y mi otro compañero de equipo se encarga del Frontend, no interfiriendo en el mismo código , pudiendo suplir el uso de Git , al menos por ahora.

Ademas de que hemos controlado las versiones de las aplicaciones a la antigua , haciendo una copia de los archivos y trabajando con nuevos , además de crear copias y trabajar con las copias a mano,para no afectar el código de la aplicación principal , teniendo un profundo cuidado de no dañar código que afecte al funcionamiento correcto de la aplicación.

Para esto último los tests habrían sido de mucha ayuda , pero lamentablemente debido al poco tiempo con el que se contaba para la realización de las aplicaciones y el complimiento de los requerimientos , los mismos serán parte , al menos temporalmente , de la deuda técnica , tema muy hablado actualmente en diversos foros de internet, por parte de muchos desarrolladores de software que también se enfrentan a dicho problema.

Sin embargo a mi consideración la utilización de una herramienta como Git , y su contraparte como puede ser GitHub o GitLab , son verdaderamente cruciales a la hora de asegurar el código y poder trabajar en equipo con otros desarrolladores en la misma aplicación , ya que si bien en mi caso fueron 2 distintas , si en un futuro se quisiera aumentar la cantidad de desarrolladores en el Backend o Frotend para agilizar los tiempos de desarrollo en un futuro y mejorar su mantenibilidad , seria verdaderamente engorroso pasarse los archivos entre computadoras y cuidar que un desarrollador no rompa el código del otro miembro del equipo.

Por tal motivo se están llevando a cabo esfuerzos para que al menos se pueda hacer uso de Git de manera local en la red local.

Aunque aun teniendo Git de manera local, estaríamos muy lejos de tener los beneficios que brinda GitHub o GitLab a nivel de control de versiones y trabajo colaborativo.

Esto también implica que lamentablemente durante el transcurso de esta pasantia, no he podido mejorar mis aptitudes en el uso de Git y GitHub/GitLab, algo en lo que muchos profesores de la carrera hicieron énfasis, tanto en su uso, como en su enseñanza.

Sin embargo, es de las pocas cosas negativas que se destacan de la misma.

Para ir terminando , una de mis ultimas tareas fue la implementación de Docker para poder desplegar las aplicaciones y que no intervegan las versiones de JAVA entre una aplicación y otra , ya que si quisiéramos utilizar JAVA 21 para una determinada aplicación , era prácticamente obligatorio migrar todas las demás aplicaciones que se esten ejecutando en el mismo servidor a JAVA 21, ya sea que esten en JAVA 8 , JAVA 11 , JAVA 17,etc.

Y asi nos iría ocurriendo conforme vayan surgiendo nuevas aplicaciones , independientemente del lenguaje utilizado , por tal motivo , se implemento Docker , yo no participe tanto en la definición de la estructura del mismo en el servidor , pero si en la creación de las imágenes en las distintas aplicaciones JAVA para que puedan ser utilizadas cuando se creen los contenedores para cada una.

Esto nos permitirá poder realizar en un futuro , nuevas aplicaciones en cualquier versión de JAVA según los requerimientos, y además no solo eso , sino que también se podría definir todas las disitinas versiones de las aplicaciones a utilizar , no interfiriendo con los demás contenedores y por consiguiente aplicaciones.

Esto da por concluida mis tareas a lo largo de la pasantia.

CONCLUSION

A lo largo de la pasantia , han ido sucediendo diversas situaciones que me han permitido ir convirtiéndome en un mejor desarrollador de software , como asi también en comprender mejor el funcionamiento y forma de trabajo de una organización empresarial real .

Esto ha sido un camino que empezó con la muestra y explicación por parte de mis superiores de como era el y los sistemas actuales que ellos poseían y como interactuaría con los mismos, a fin de mejorarlos y mantenerlos, algo que escalo de tal manera que no solo pude participar en su mejora y expansion , sino en su reemplazo , en la realización de nuevos sistemas .

Sistemas en los que pude seguir poniendo en practica todo lo que había aprendido , como asi también trabajar en equipo , obteniendo y mejorando tanto las habilidades practicas como sociales , algo de vital importancia tanto para los desarrolladores , como para las empresas que los emplean.

Para esto tuve en cuenta las buenas decisiones que se habían tomado en un pasado en la realización de ciertas aplicaciones, pero evaluando y haciendo uso de las nuevas herramientas provistas por la comunidad actualmente , evitando antiguos errores y previniendo futuros inconvenientes relacionados a los mismos.

Un caso que se puede mencionar es el uso de los procedimientos almacenados , ya que si bien a lo largo del transcurso de este informe me he encargado de mencionar sus beneficios y la gran utilidad que pueden dar a la hora de trabajar con la base de datos y aislar cierta lógica en la misma , quizás no he hecho tanto énfasis en los problemas que pueden traer los mismos al programador menos experimentado.

Ya que si bien los procedimientos almacenados poseen grandes beneficios, el entendimiento de su códigos y mantenibilidad por parte de nuevos programadores , e incluso algunos también bastante experimientados puede ser tedioso, ya que su forma de crearlos y definirlos a nivel de sintaxis y lógica puede ser complicado si no se posee una buena base tanto de el lenguaje de Base de Datos SQL , como el propio de los distintos procedimientos en cada base de datos , como puede ser PLPG/SQL , PL/SQL, o TRANSACT/SQL .

Esto permite comprender y tener en cuenta que cada herramienta ,por mas utilizada que sea , y por mas brillos que tenga su trayectoria de uso en el tiempo , no esta exenta de desventajas , por tal motivo un buen desarrollador de software debe estar siempre analizando no solo el uso de la herramienta y como soluciona esta el problema que se posee, sino en como afecta esta misma el resto del ambiente en el que se emplea y el no será inerte a ciertos cambios fuertes.

Para esto el desarrollador de software siempre deberá estar en contante aprendizaje , para , como se dice actualmente “no quedarse en el tiempo” , en esta profesión , a diferencia de quizás muchas otras , como puede ser el campo de la contabilidad o economía, en las que la teoría y practica , como asi también las herramientas utilizadas por los profesionales , no realizan tantos cambios bruscos a lo largo de los años, nuestro campo si los realiza , algo para lo que hay que estar preparado, capacitándose , estudiando y mejorando , para poder brindar siempre no solo las capacidades lógicas necesarias para la comprensión de los sistemas y su definición , sino también las técnicas para lograr llevarlos a cabo de la mejor manera y siguiendo los mejores estándares de la industria , haciendo de uno un gran profesional.

Para ir terminando puedo concluir que la experiencia obtenida en la pasantía en la Caja de Jubilaciones y Pensiones de Entre Ríos , fue verdaderamente valiosa y gratificante , no solo en el campo técnico , sino también en el campo humano , conviviendo de una gran manera con mis compañeros de equipo, convirtiéndose en la primer gran pieza de mi vida profesional.

RECOMENDACIONES:

Para los futuros profesionales en el campo del desarrollo del software , les recomiendo encarecidamente no estar desactualizados , siempre tratar de estar al pendiente de las ultimas tecnologías en el mercado , y las distintas actualizaciones de las que ya utilizan.

Sin embargo, es muy probable , que es su primer experiencia profesional se les encomiende trabajar en el mantenimiento de un software legacy, algo que sucedió en mi caso , por tal motivo , tampoco hay que venirse abajo si no se conoce al 100% las nuevas tecnologías , ya que las mismas también tardan un tiempo en llegar a implementarse y utilizarse en el ámbito laboral.

A su vez , también es de vital importancia , pulir sus habilidades sociales , algo que les mejorara en gran medida su estadia en las empresas en las que se desempeñen , no solo mejorando ellos mismos como personas , sino también contribuyendo a sus compañeros de equipo.