UNIVERSIDAD DE ANTIOQUIA

Integración módulo de Reportes a la calculadora de 4ners points.

Autor(es)

Santiago Cadavid Bustamante

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Sistemas

Medellín, Colombia

2019

Integración módulo de Reportes a la calculadora de 4ners points

Santiago Cadavid Bustamante

Informe de práctica o monografía o investigación o tesis o trabajo de grado
como requisito para optar al título de:
Ingeniero de Sistemas

Asesores (a) o Director(a) o Co- Directores(a).

Raúl Ramos Pollán. PhD Ingeniería de Informática.
María Fernanda Galeano Cubillos. Psicología.

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Sistemas
Medellín, Colombia
2019

Resumen
La aplicación llamada calculadora de 4ners points fue creada a mediados del año 2018 en la casa de software S4N, bajo su propio uso, para facilitar todos los procesos de entrega de bonos de fidelidad a sus empleados y de esta forma incentivar el desarrollo de software dentro de la empresa. Sin embargo, aún existían algunos reprocesos como la generación de reportes a partir de dichos datos, por lo que con el tiempo se volvió una necesidad de negocio el automatizar dicho proceso. Por lo tanto, partiendo de esta necesidad, se presentó la oportunidad de crear el módulo de reportes que -a día de hoy- automatiza dicho proceso y que a partir de datos almacenados por los procesos de la calculadora, se generan reportes en formato excel (*.xls*) que dan un resumen de todos los bonos reclamados. Adicional a ello, dicha funcionalidad genera más reportes -en formato excel- como el de proveedores, categorías y usuarios activos en S4N que puedan acceder a la calculadora, dando como resultado un beneficio grande a los que usan la calculadora como administradores al evitar un reproceso grande que tenían de tener que generar reportes, entrando a cada usuario y a cada diligencia hecha por ellos.

Finalmente, a parte de agregar el módulo de reportes, se vió una necesidad prioritaria de organizar toda la infraestructura del proyecto, debido al gran acoplamiento que se tenía con AWS [1], generando un antipatrón de diseño conocido como vendor lock-in [2] en servicios como la base de datos y los ejecutables finales de la aplicación. Por lo que fue necesario dockerizar los componentes de back, front y base de datos para montarlos en una máquina de EC2 independiente, separando cada responsabilidad y recurso. Además a esto, se agrega despliegue continuo e integración continua a la aplicación por medio de la herramienta de gitlab CD/CI [3]. Aunque con este cambio se logró un impacto positivo a nivel técnico y no se vea reflejado de lado de negocio, en desarrollos futuros se verá cómo es aún más fácil implementar cambios en la plataforma y desplegarlos rápidamente.

Introducción

La calculadora de 4ners points es una aplicación creada en la empresa S4N con el fin de premiar la fidelidad de sus trabajadores e incentivarlos a continuar sus labores en la compañía. Antes de que existiera la aplicación de la calculadora, todo proceso referente a esta repartición de bonos se hacía de manera manual, donde se tenían que llenar formularios y presentar formatos en los que cada empleado debía diligenciar qué bonificaciones quería, algo insólito para una empresa bajo el contexto de casa de software, por lo que a mediados del año 2018 se empieza a implementar esta iniciativa y así evitar todos los procesos y reprocesos que conllevaba. A finales de este año la aplicación salió a producción como beta para realizar un análisis de experiencia de usuario. En dicho experimento los resultados arrojados fueron positivos, donde la funcionalidad core de calcular los puntos resultó favorable al reducir la maniobrabilidad manual de este proceso, sin embargo, se vió también la necesidad de agregar más módulos a la plataforma que ayuden a complementar dicha funcionalidad y a eliminar aún más el boilerplate casero que presentan procesos como la generación de reportes a partir de dichos datos. Agregando a ello oportunidades de mejora tanto en interfaz gráfica, que no se tuvieron en cuenta en su momento, como en refactor de código backend.

Todas las mejoras mencionadas anteriormente son meramente viendo la aplicación del lado de desarrollo, es decir como código, pero también se presentó oportunidades de mejora que se abordaron en esta práctica empresarial del lado de infraestructura, en el cual, el problema que se presentó fue el aprovisionamiento de los componentes de la aplicación. En este caso los componentes de backend y frontend eran desplegados con el ejecutable (.jar) en artifactory y la base de datos estaba ligada al servicio RDS de AWS [1], por lo que cada vez que se tenía que hacer un cambio y desplegar el componente, se debía compilar el código para generar el respectivo ejecutable y hacer el cambio en el repositorio de artifactory manualmente, y con la base de datos se debía entrar a RDS a modificar las respectivas tablas o la información correspondiente.

Por lo tanto, al observar todas estas oportunidades de mejora de la aplicación se propuso como alcance la implementación de un módulo de reportes para facilitar la generación de información mejor visible para los administradores y también, el aprovisionamiento de los componentes de diferente manera, los cuales se prepararían en componentes independientes con ayuda de docker y así hacer un despliegue e integración continua a cada cambio que sufra la plataforma.

En ese orden de ideas y con el alcance claro, se puede evidenciar dos claros impactos positivos que se lograron cumplir, y que a su vez afectaron dos fines completamente distintos pero que se unen; un impacto de lado de negocio, facilitando compactar la información en reportes específicos que requieren y un

impacto de lado técnico, ayudando a mejorar la infraestructura de la aplicación para que sea más adaptable a cambios e independiente, deshaciendo antipatrones que se usaron con anterioridad.

Finalmente la metodología empleada para el desarrollo de las nuevas funcionalidades fue SCRUM [4], dado que facilitó la entrega de las necesidades, haciéndolas de manera frecuente y que ayuda a responder a los cambios continuos que se tuvieron en el camino. Sin embargo, se sufrieron varias limitaciones como el hecho de no tener los suficientes permisos para poder acceder a la infraestructura y hacer cambios propios, por lo que se realizaron de la mano de David Montaño, el experto en infraestructura de S4N, además de limitaciones de tiempo al tener que estudiar por cuenta propia temas relacionados a infraestructura como docker, docker-compose, ansible y algunos servicios de AWS [1]. Más adelante se explicará de manera detallada todos estos puntos respectivos a la metodología.

Objetivo general

Desarrollar un módulo integrado a la aplicación 4ners points que mediante una interfaz gráfica amigable y usable, permita realizar la generación de reportes en formato excel dados ciertos filtros y ciertas reglas de negocio, tomando como base información almacenada para finalmente eliminar el proceso manual y entregar informes que den valor para la toma de futuras decisiones en cuanto a la estrategia de premiación por fidelidad.

Objetivos específicos

- Conocer cómo está conformado el equipo de trabajo y las labores que desempeñan dentro del proyecto Calculadora 4ners points.
- Entender la arquitectura de referencia con la que está construida la aplicación Calculadora 4ners points.
- Diseñar un módulo de reportes que permita generar reportes en formato excel de la información que se tiene en base de datos de acuerdo a las necesidades de negocio.
- Implementar un módulo de reportes que exponga servicios para generar reportes en formato excel teniendo en cuenta las necesidades de negocio.
- Mejorar el aprovisionamiento en infraestructura de los componentes de la aplicación usando herramientas como docker, docker-compose y ansible.

Marco teórico

Como anteriormente se mencionaba, la calculadora de 4ners points fue implementada hace alrededor de un año, bajo lenguajes de programación que permiten la fácil escalabilidad y mantenibilidad del código. El proyecto contiene diferentes módulos, los cuales permiten sistematizar mediante la implementación de una plataforma desarrollada con tecnología apropiada para soportar concurrencia, usabilidad y buen desempeño el proceso de cálculo de cantidad de puntos por empleado, por lo cual, dado que es extensible, también permite realizar la generación de reportes por medio de dicho stack tecnológico. Además de proveer una gran cobertura de funcionalidades y librerías que pueden ser anexadas para cumplir cualquier requerimiento.

Scala [5] se ha caracterizado por lo ya dicho, ya que es un lenguaje de programación robusto que corre sobre la Java Virtual Machine (JVM) y permite programar tanto funcionalmente como orientado a objetos, sin embargo, en la arquitectura de referencia que maneja el componente de backend, se evidencia el uso de ADT (Algebraic Data Types) [6] lo que no es conveniente parafrasear que el backend es una mezcla de ambos paradigmas. En este sentido, la programación funcional prima y se apoya bajo un framework de servicios llamado Play, el cual facilita de gran manera la exposición de lógica a través de un API que finalmente será consumida por un cliente de frontend. Los tipos estáticos de Scala ayudan a evitar errores en aplicaciones complejas, y hallar los errores en tiempo de compilación (y no de ejecución), por lo que hace que el desarrollo sea más consistente y seguro. La evaluación de sus expresiones es rigurosa, lo que permite evaluar expresiones al instante y demostrar rápidamente su valor. En este sentido, el backend de la aplicación es potente y la experiencia que se ha tenido con la aplicación demuestra lo potente que llega a ser Scala.

Por parte del cliente, o frontend -si así se desea llamar-, se tiene el framework Angular 5 [7], el cual provee una funcionalidad homóloga a Scala en cuanto a robustez, ya que es un marco SPA [7] creado para grandes aplicaciones en los que se debe tener en cuenta todos sus módulos. Su gran acceso a múltiples librerías permite desarrollar fácilmente módulos para exportar contenido en formatos .xls y además de generar una experiencia de usuario reactiva, dando paso a una gran usabilidad. Angular además de todas estas características permite el consumo de servicios web de manera asíncrona y fácil, por lo que las integraciones que se dan con el componente de backend en Scala no representan mayor dificultad.

La persistencia de datos, implementada en una base de datos relacional como lo es Postgres [8], representa un gran reto, ya que continúa con la definición de los demás microservicios, dando robustez y consistencia a la información que se almacena. Al

ser una base de datos relacional se puede jugar con los índices y con las llaves primarias y foráneas para hacer búsquedas rápidas a través de los mismos queries.
Respecto a herramientas de contenedores, también se presenta un gran desafío para aprenderlas y organizar la infraestructura de la aplicación. Para este caso, el uso de docker es ajustable, ya que representa el crear imágenes por cada componente que se comportan de manera independiente. Docker [9] como tecnología de creación de contenedores nos ofrece la capacidad de modularizar los componentes, restaurarlos cada vez que sufran cambios e implementarlos fácilmente, además de dar un plus a la hora de realizar el despliegue a producción con ayuda de CI/CD [3].

Metodología

El proceso de desarrollo de las nuevas funcionalidades de la aplicación de la calculadora de puntos se ha llevado a cabo bajo el marco de trabajo SCRUM [4], el cual se enfoca en trabajos pequeños, entregas rápidas y constantes, y buena comunicación con el cliente.

Al inicio del proyecto se plantearon unos objetivos con el product owner (PO) y el equipo de desarrollo, también se realizó la definición y priorización de las historias de usuario, se acordaron sprints cada dos semanas y la retroalimentación de los usuarios de esta plataforma durante el proceso. El proyecto se dividió en dos grandes releases o entregables que componen toda la funcionalidad trazada como alcance durante el desarrollo completo de la aplicación.

El release 1 se compone de la funcionalidad relacionada con infraestructura, en ella se realizó un estudio de las herramientas docker, docker-compose y ansible, para preparar en contenedores cada servicio de la aplicación (back, front y base de datos) y ser desplegados en una máquina de EC2. El release 2 comprende el módulo funcional de generación de reportes; en el cual comprende reportes como el resumen de calculadora, en el que la sale información resumen de lo que cada empleado pidió como bonificación, el reporte de proveedores, en el que sale la información de los proveedores con los que S4N hace convenios para regalar los bonos, el reporte de categorías, que consiste en mostrar información de todos las categorías que tienen los bonos y por último, el reporte de usuarios, en el cual se muestra información básica de cada empleado que puede acceder a la calculadora de puntos.

Funcionalidad de infraestructura

En primer lugar se empieza entonces por dejar la infraestructura de la aplicación lista con las mejoras planteadas como objetivo, para tal desarrollo, se hizo un estudio e investigación breve de docker como herramienta útil para crear los

contenedores por aplicación. Se estimó dejar cada microservicio como contenedor en una semana, por lo que en menos de un mes ya se debían tener todos los componentes independientes como imágenes de docker. Y de esta manera sucedió.

El primer paso fue empezar por la persistencia, la cual está implementada en postgresql, generando un dump, con el cual se creó el contenedor de la base de datos; todo esto a partir de una imagen de postgres:9.6, a la cual se le pueden compartir archivos ejecutables y dumps desde la máquina local para ser ejecutados y recrear la base de datos guardada como backup, a parte de crear variables de entorno como el usuario y la contraseña para que el backend posteriormente pueda acceder. Dicho contenedor parte de una imagen que se descarga haciendo pull de dockerhub y a través de parámetros como el puerto **-p 5432:5432**, los volúmenes **-v ./postgres_scripts:/docker-entrypoint-initdb.d ./db:/db** y variables de entorno se convierte en un proceso que corre únicamente la configuración de base de datos. Todo esto fue finalmente verificado conectando la base de datos con DataGrip y ejecutando queries de consulta, como se puede observar en la Figura 1.

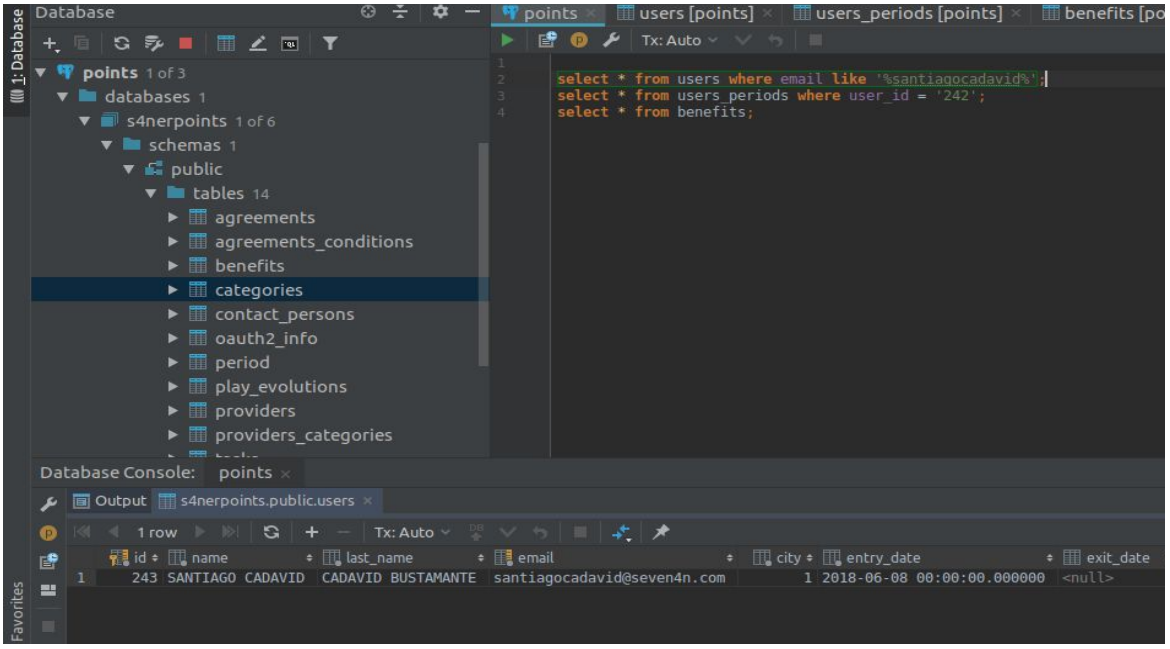


Figura 1. Prueba funcional de conexión a base de datos dockerizada con DataGrip.

En una segunda iteración, se tomó como referencia, crear la imagen de docker del componente de back-end, *s4nerpoints-back*. Para crear la imagen fue sumamente fácil, debido a que sbt, el compilador de Scala, contiene un comando llamado docker, mediante el cual según la configuración del proyecto (build.sbt) se puede crear la imagen de docker con solo ejecutar la instrucción **sbt docker:publishLocal** en el compilador. De esta forma, se hace un pull de una imagen del openjdk:8 y dadas unas variables de entorno que se encuentran en el application.conf del proyecto, se crea la imagen, agregando también parámetros como el puerto (que

correrá por el 9000). Finalmente, la imagen de docker que se crea se llama s4nerspoints:1.0 por defecto, por lo que posteriormente es necesario tagearla con el nombre de **105725941292.dkr.ecr.us-east-1.amazonaws.com/points/back**, con el fin de usar docker-compose y hacer push al repositorio de aws. En la Figura 2 se puede observar la demostración de la creación de la imagen docker del backend.

Figura 2. Creación de la imagen del componente de backend etiquetada con el repositorio de aws.

Finalmente, el procedimiento para generar la imagen del front también fue fácil, ya que corresponde a entrar en el proyecto, especificamente en el directorio donde está el dockerfile y correr el comando **docker build -t 105725941292.dkr.ecr.us-east-1.amazonaws.com/points/front .**, de esta manera se crea la imagen con dicho tag, parecido al del back. Adicional se crea en la sección de scripts en el package.json, una abreviación para hacer clean del proyecto, build y finalmente construir la imagen, todo esto solo con correr **npm run dev** y **npm run all** si se desea crear la imagen en ambiente de desarrollo y producción respectivamente.

De esta forma, los tres componentes que conforman la aplicación se prepararon en contenedores para con ayuda de docker-compose correr toda la aplicación. Por lo tanto, se creó otro repositorio llamada ***deployment***, en el que se encuentra toda la configuración de despliegue del componente en ambiente local y cómo se observa en ambiente de producción. En la Figura 3 se observa dicha estructura y el yaml del docker-compose, cabe resaltar que tenemos 3 servicios a subir dockerizados y que de cada uno se debe construir su imagen correspondiente. En este punto ya tenemos la imagen de la base de datos que parte de una imagen de postgres, la imagen del backend que se construye a partir del código con ayuda del sbt y finalmente la imagen del frontend que también se construye a partir del código, sin embargo, en este caso, nos fue de ayuda que el front está montado sobre un servidor node, y que el dockerfile se basa en una imagen de node como se explica anteriormente.

Sin embargo, a pesar de que se obtuvo cada componente dockerizado, el uso de docker-compose juega un papel importante a la hora de levantar toda la aplicación, ya que con solo ejecutar el comando **docker-compose up -d** estás subiendo cada componente, obviamente todo dependiendo de lo configurado en el archivo de docker-compose. En la Figura 3 se puede observar la configuración de las imágenes.

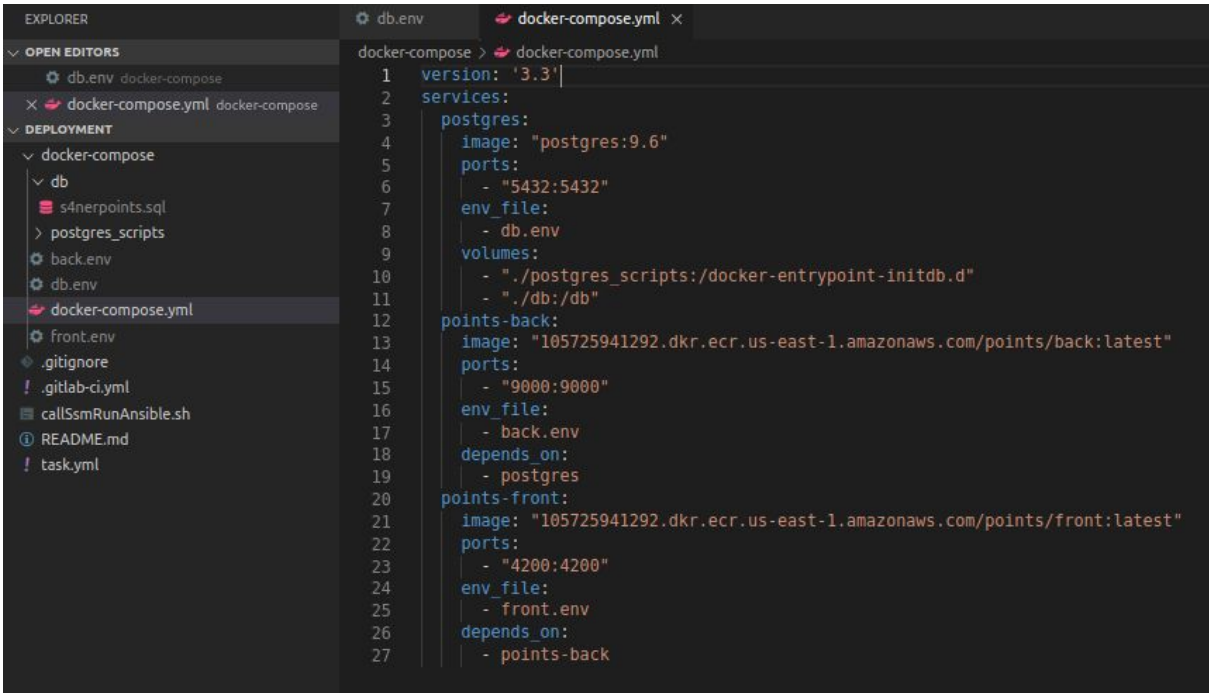


Figura 3. Estructura de paquetes del proyecto *deployment*, correspondiente a despliegue con docker.

Se puede observar también los archivos .env que funcionan como las variables de entorno para cada componente, y además, el backup de la base de datos (el archivo .sql) con el cual se alimenta y se sube la imagen de postgres. Cabe aclarar que los archivos en entorno se tienen en local, pero no se versionan debido a que es información sensible. En su momento cuando se realizó el despliegue en la nube de la aplicación se guardaron en un bucket de s3 y luego se recuperaron a la hora del despliegue.
Para bajar todos los componentes usando docker-compose simplemente se usa el comando **docker-compose kill**. Si se desea bajar alguno de los 3, se usan comandos de docker normalmente.

En el transcurso de organizar el despliegue se encontraron varios conflictos como lo fue la autenticación con Google, ya que el servicio de GCP (Google Cloud Platform) [10] provee tokens para poder realizar bien la autenticación, donde exige el host origen del cual es enviada la petición, por lo que se crearon 2 tokens a generar: uno desde el host de producción, *points.s4n.co* y el otro desde local, *localhost:4200*.
Otro conflicto que se sorteó, corresponde a los CORS, ya que de lado del servidor no se estaban aceptando las peticiones que el front ejecutaba, se configuró entonces del lado del backend que aceptara peticiones del host *localhost:4200*.

Sumado a ello, se tuvo que realizar otro dump de la base de datos, recreando las sentencias de creación del superusuario y de la base de datos, ya que las tablas de la db estaban resultando duplicadas y no dejaba controlar la base de datos con el usuario de aplicación. Para todo ello, se reestructuró las sentencias de creación iniciales para la imagen de docker de postgres.

Teniendo todos estos insumos y sabiendo que la aplicación corría perfectamente en ambiente local se procedió a realizar el despliegue en ambiente de producción. Para hacerlo posible también se siguieron unos pasos; donde primeramente se realizó la creación de la instancia EC2 en el servicio de AWS para la aplicación points, todo esto mediante un template de Terraform, tomando como guía el template de una aplicación que se había creado antes llamada psychological. Después de creada la máquina de EC2, se dispuso a correr los componentes de la aplicación allí mismo, contando con la ayuda de gitlab, ya que se configura los pipelines de CI para los proyectos de backend y frontend, que consisten básicamente en hacer pull del proyecto, crear la imagen del servicio, loguearse a la cuenta de AWS, y hacer push a un repositorio de ECR de la imagen creada (para el backend se agrega el paso de añadirle el tag a la imagen). En su momento ocurrió un error con la máquina de EC2 que corre el gitlab ya que no había suficiente espacio para crear la imagen del frontend, sin embargo, se depuró de todos los procesos que se estaban almacenando en dicha máquina.

Finalmente, al automatizar la parte de la creación y push de las imágenes quedaba faltando automatizar la parte de correr la aplicación (incluyendo la base de datos), por lo que se usó ansible para configurar la máquina de EC2 creada anteriormente y controlarla remotamente, simplemente dándole comandos de instalar algunas dependencias, hacer pull de las imágenes de docker, hacer pull de los archivos .env que reposan en S3 junto al backup de la base de datos; y todo esto mezclándolo como si fuera en local, para finalmente correr el comando **docker-compose up -d** en la máquina de EC2 y subir la aplicación.

Funcionalidad de negocio

Para lograr visualizar cómo se comporta la aplicación y la relación que tiene cada columna fue necesario acceder a base de datos para ver las tablas y de dónde se podría sacar dicha información. Se construyeron entonces dos queries que retorna la información que se pide en el reporte por parte del negocio. El reporte se compone de dos pestañas, la primera contienen las columnas Nombre, Apellido, Correo, Bono, Puntos, Cantidad, Convenio, Proveedor y Categoría, y la segunda contienen las columnas Proveedor, Nit, Contacto, Teléfono, Correo y Categoría.

Dado que la implementación de la aplicación está basada en una arquitectura hexagonal, es importante separar las responsabilidades de cada lógica. Por lo que para automatizar el proceso, se procedió desde el componente de back a crear el paquete reports dentro de la estructura de paquetes, con su respectivos dao, models y services.

A modo de resumen, el dao está compuesto por el patrón repository que tiene la responsabilidad de conectar con base de datos, construir el query y traer el reporte. Los modelos componen las filas de los reportes (de forma tipada), el servicio de reportes que tiene su trait (clase de abstracta) e implementación se encargan de definir la firma del método y de implementar la creación del excel para que finalmente el controller sea el que haga el response de la petición del frontend.

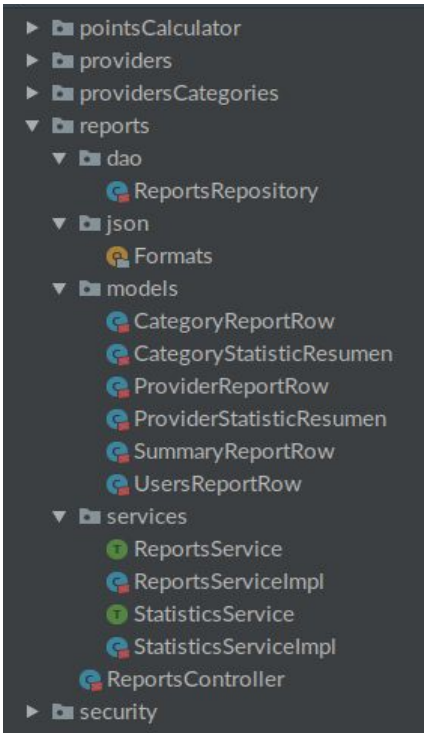


Figura 4. Paquetes módulo reportes

A continuación se describe cada uno de los paquetes que corresponden a la estructura de paquetes del nuevo módulo de reportes del lado del backend.

dao

Dado su patrón repositorio, se crea el query usando slick para traer el reporte dependiendo del periodo actual en el que esté siendo usado la calculadora. Eso de lado del reporte principal, pero también se hacen queries para extraer información de base de datos para generar los demás reportes.

ReportsRepository.scala

```scala
package co.com.s4nerpoints.reports.dao

import ...

class ReportsRepository @Inject()(protected val dbConfigProvider: DatabaseConfigProvider,
                                  val providersTable: ProvidersTable,
                                  val usersTable: UsersTable)
                                 (implicit executionContext: ExecutionContext)
  extends UsersBenefitsTable with UsersPeriodsTable with BenefitsTable with AgreementsTable
    with ProvidersCategoriesTable with CategoriesTable with ContactPersonsTable {

  import profile.api._

  override val providers = providersTable.providers

  def getSummaryReport(periodId: Option[Long]): Future[Seq[SummaryReportRow]] = {...}

  def getProvidersReport: Future[Seq[ProviderReportRow]] = {...}

  def getCategoriesReport: Future[Seq[CategoryReportRow]] = {...}

  def getUsersReport(periodId: Long): Future[Seq[UsersReportRow]] = {
    val query = for {
      up  <-  usersPeriods          if up.periodId === periodId
      u   <-  users                 if up.userId === u.id
    } yield (u.id, u.name, u.email, u.entryDate, up.totalPoints, up.currentPoints)

    db.run(query.result)
      .map(_.map(t => UsersReportRow(t._1, t._2, t._3, t._4, t._5, t._6)).sortBy(_.userId))
  }

}
```

Figura 5. Clase ReportsRepository.scala correspondiente al repositorio del backend de reportes.

services

Dado que la firma de los métodos generateSummaryReport, generateProvidersReport, generateCategoriesReport, generateUsersReport retornan un Future[Array[Byte]], se realiza la lógica de construcción del excel como stream, todo esto teniendo en cuenta lo traído de base de datos con el patrón repositorio.

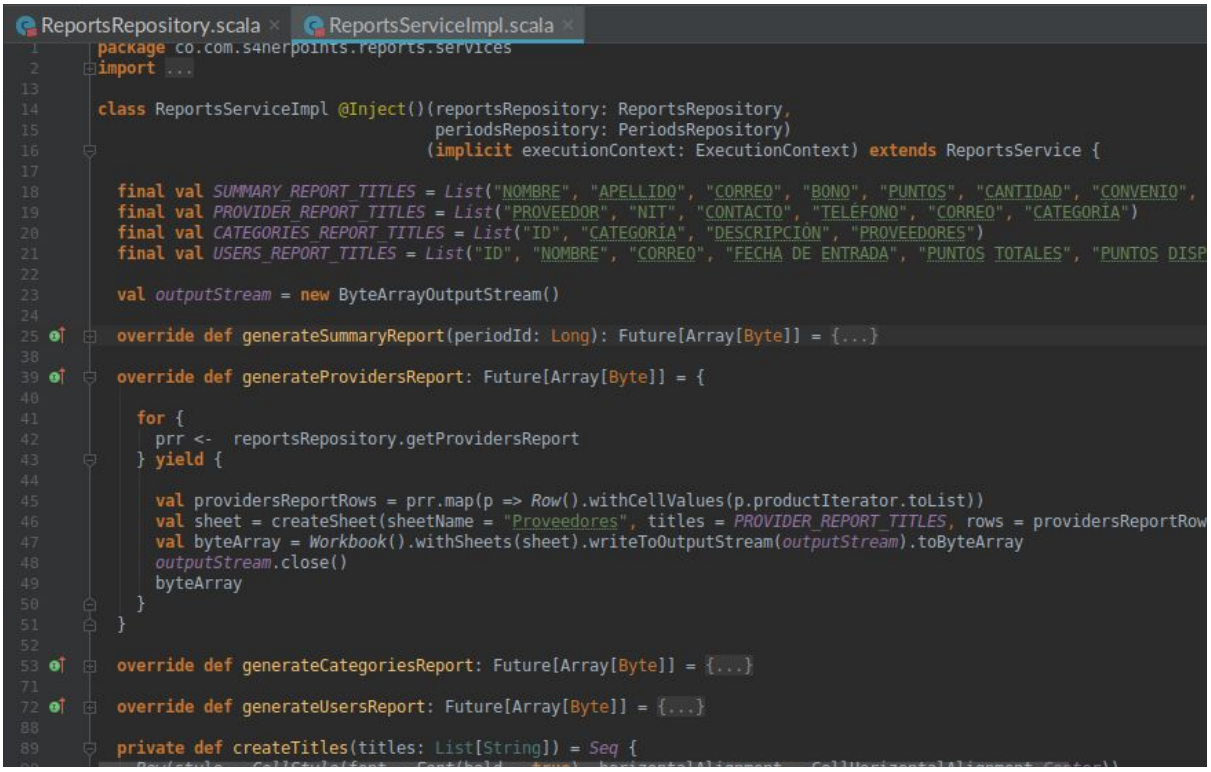


Figura 6. Clase ReportsServiceImpl.scala correspondiente a la implementación del servicio de reportes.

controller

Es el controlador el que expone el API con Play framework y con uso del google auth retorna el excel como un binario.

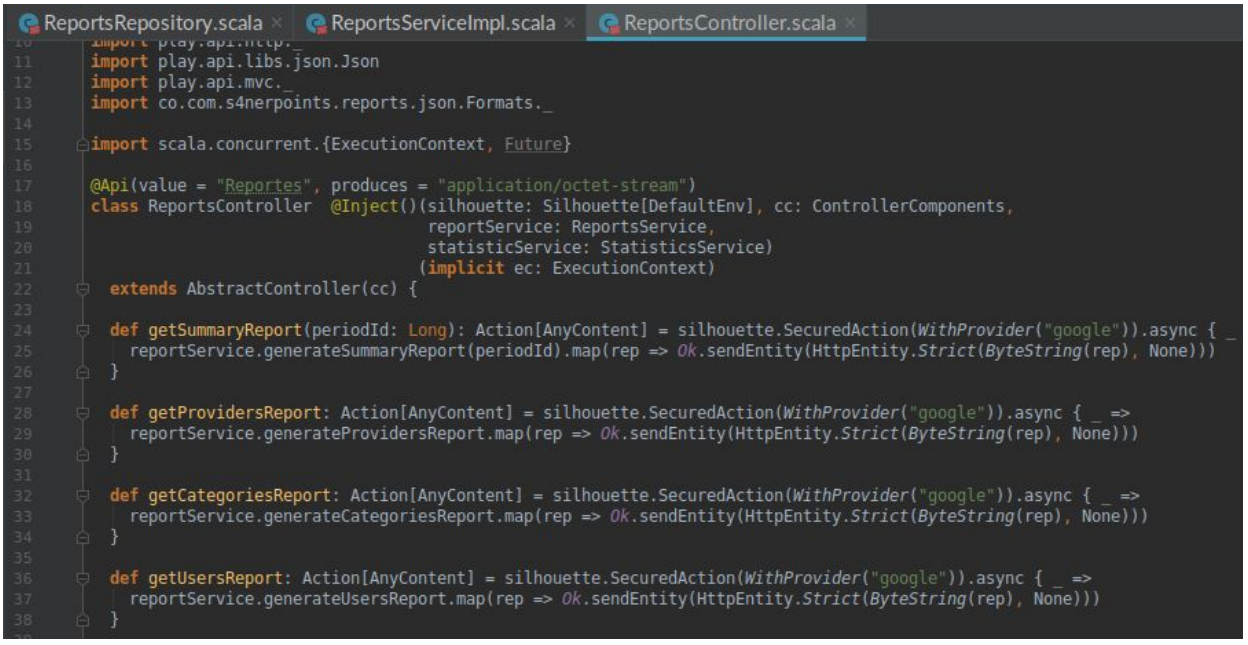


Figura 7. Clase ReportsController.scala correspondiente a la implementación del controlador de reportes.

Ya teniendo los servicios de backend expuestos y enviando el excel como un array binario, se procedió a crear el botón de generar reporte desde el frontend, para

conectar la visual y finalmente exportar el array en formato .xls como se muestra en la Figura 8.

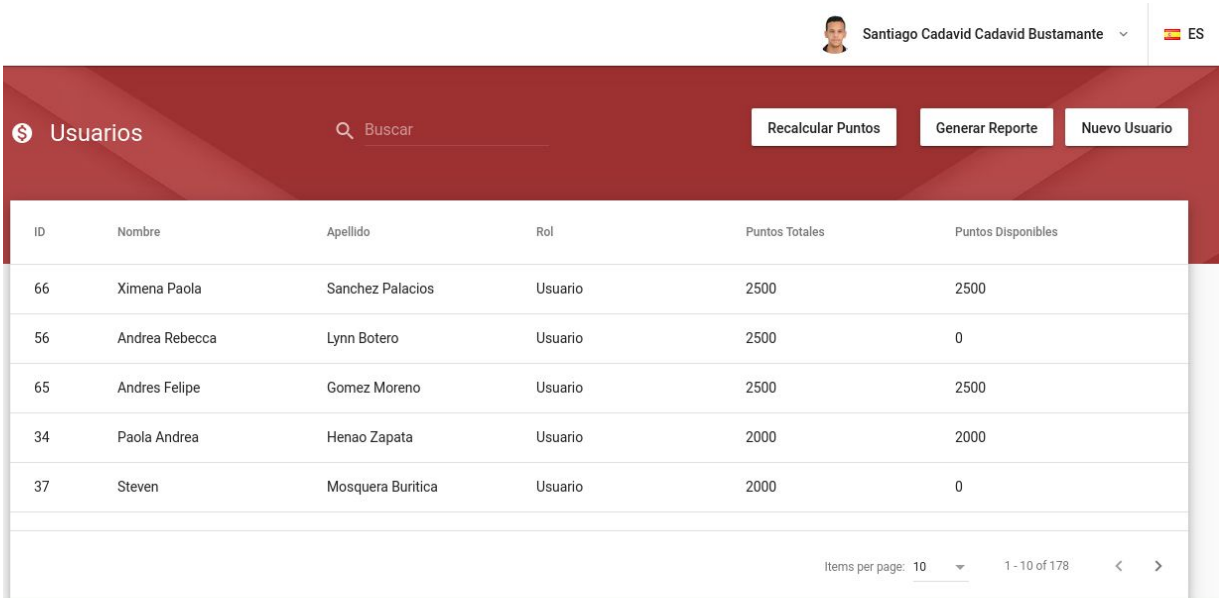


Figura 8. Interfaz de usuario en la sección de usuarios, botón generar reporte.

Los resultados de dicha integración se detallan más adelante.

Resultados y análisis

Funcionalidad de infraestructura

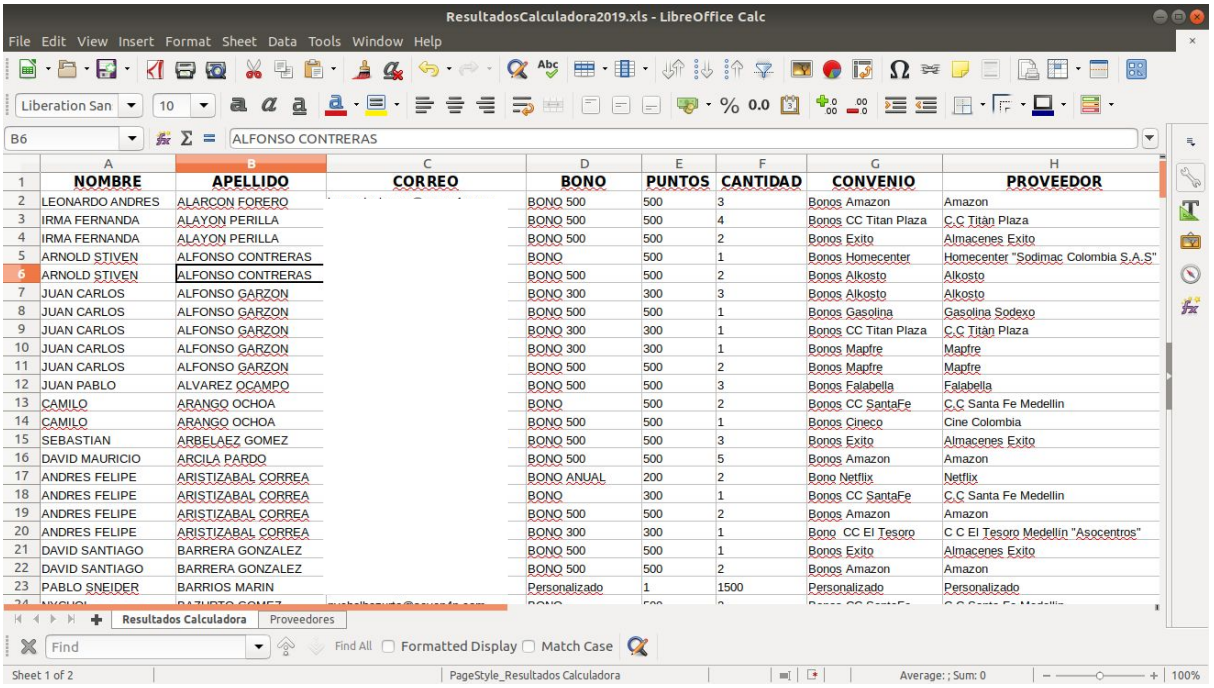
Como se mencionaba con anterioridad, la funcionalidad de infraestructura nació en el proyecto con el ánimo de mejorar cómo se hacía el despliegue de la aplicación tanto en ambiente local como en producción. Y no era para menos, ya que para correr la aplicación en ambiente local, se tenía que acceder a cada código y correr manualmente lo que se tenía, agregando la dificultad de aprovisionar la base de datos; al fin de cuentas eran problemas de configuración e instalación de programas como postgres, node, etc. Es por esto que el beneficio de haber realizado esta mejora aunque no es visible, trajo un valor inmenso para la plataforma, ya que permite realizar despliegues en producción continuos, trabajar de manera rápida y abstraer toda la capa de configuración necesaria para correr la aplicación en ambiente local para futuros desarrollos. El resultado final de este entregable fue el despliegue de la aplicación en producción el 19 de agosto, dos días antes de que la calculadora empezara a funcionar en un nuevo periodo donde los empleados se iban a ver recompensados.

Funcionalidad de negocio

El resultado final de crear el módulo de reportes en la aplicación de calculadora de puntos fue exitoso, ya que se pudo quitar el proceso manual que se tenía que hacer

para generar un reporte, y de esta manera, con solo ejecutar el evento desde la visual, se puede descargar un excel en formato .xls que trae toda la información diligenciada por los empleados. Cada vez que se realiza la descarga del excel, es de la imagen y del último estado en el que esté la calculadora, ya que con el tiempo los empleados pueden ir cambiando los bonos que van escogiendo, por lo que el reporte generado va de acuerdo a lo que han escogido en ese preciso momento. Esta opción gustó mucho por parte de negocio, ya que pueden sacar informes a partir de distintas imágenes del reporte mientras la calculadora está encendida y observar qué cambios hacen los empleados frecuentemente frente a los bonos que escogen.

Como resultado final se realizó el despliegue a producción con esta nueva funcionalidad, y negocio quedó satisfecho con el trabajo realizado. El despliegue se hizo el 9 de septiembre cuando la calculadora fue cerrada, y la generación del primer excel se realizó el mismo día, generando el reporte que se visualiza en la Figura 9.



	A	B	C	D	E	F	G	H
1	NOMBRE	APELLIDO	CORREO	BONO	PUNTOS	CANTIDAD	CONVENIO	PROVEEDOR
2	LEONARDO ANDRES	ALARCON FORERO		BONO 500	500	3	Bonos Amazon	Amazon
3	IRMA FERNANDA	ALAYON PERILLA		BONO 500	500	4	Bonos CC Titan Plaza	C.C Titan Plaza
4	IRMA FERNANDA	ALAYON PERILLA		BONO 500	500	2	Bonos Exito	Almacenes Exito
5	ARNOLD STIVEN	ALFONSO CONTRERAS		BONO	500	1	Bonos Homecenter	Homecenter "Sodimac Colombia S.A.S"
6	ARNOLD STIVEN	ALFONSO CONTRERAS		BONO 500	500	2	Bonos Alkosto	Alkosto
7	JUAN CARLOS	ALFONSO GARZON		BONO 300	300	3	Bonos Alkosto	Alkosto
8	JUAN CARLOS	ALFONSO GARZON		BONO 500	500	1	Bonos Gasolina	Gasolina Sodexo
9	JUAN CARLOS	ALFONSO GARZON		BONO 300	300	1	Bonos CC Titan Plaza	C.C Titan Plaza
10	JUAN CARLOS	ALFONSO GARZON		BONO 300	300	1	Bonos Mapfre	Mapfre
11	JUAN CARLOS	ALFONSO GARZON		BONO 500	500	2	Bonos Mapfre	Mapfre
12	JUAN PABLO	ALVAREZ OCAMPO		BONO 500	500	3	Bonos Falabella	Falabella
13	CAMILO	ARANGO OCHOA		BONO	500	2	Bonos CC SantaFe	C.C Santa Fe Medellin
14	CAMILO	ARANGO OCHOA		BONO 500	500	1	Bonos Cineco	Cine Colombia
15	SEBASTIAN	ARBELAEZ GOMEZ		BONO 500	500	3	Bonos Exito	Almacenes Exito
16	DAVID MAURICIO	ARCILA PARDO		BONO 500	500	5	Bonos Amazon	Amazon
17	ANDRES FELIPE	ARISTIZABAL CORREA		BONO ANUAL	200	2	Bono Netflix	Netflix
18	ANDRES FELIPE	ARISTIZABAL CORREA		BONO	300	1	Bonos CC SantaFe	C.C Santa Fe Medellin
19	ANDRES FELIPE	ARISTIZABAL CORREA		BONO 500	500	2	Bonos Amazon	Amazon
20	ANDRES FELIPE	ARISTIZABAL CORREA		BONO 300	300	1	Bono CC El Tesoro	C C El Tesoro Medellin "Asocentros"
21	DAVID SANTIAGO	BARRERA GONZALEZ		BONO 500	500	1	Bonos Exito	Almacenes Exito
22	DAVID SANTIAGO	BARRERA GONZALEZ		BONO 500	500	2	Bonos Amazon	Amazon
23	PABLO SNEIDER	BARRIOS MARIN		Personalizado	1	1500	Personalizado	Personalizado

Figura 9. Reporte resumen de calculadora periodo 2019 exportado desde la aplicación.

De esta forma, no solo la calculadora brinda la capacidad de calcular puntos para los empleados, sino que también genera informes a los administradores de la aplicación.

Conclusiones

Gracias al marco de trabajo elegido al inicio del proyecto se pudo trabajar de manera organizada precisando, entre el equipo de desarrollo y la product owner, objetivos de corto alcance bien definidos, logrando así entregas continuas y con una tasa más alta de aceptación. De esta forma, se concluye que los objetivos propuestos en un principio se cumplieron a cabalidad y que impactaron positivamente la aplicación.

Además de ello, se puede concluir:

- La modalidad de práctica empresarial realizada da un gran acercamiento a la vida laboral y al campo de desarrollo de software, y más en una empresa como S4N donde siempre se vela por hacer los mejores productos de software.
- La colaboración de personas que conozcan el negocio fue fundamental en la creación de las necesidades como historias de usuario. En este caso, es bueno resaltar la labor de la product owner al ser clara con los requerimientos que pedía.
- El aprendizaje del lenguaje de programación Scala fue en un principio complicado, sin embargo, teniendo la arquitectura de referencia de los demás servicios se hizo más entendible el proyecto de backend.
- El aprendizaje de herramientas como docker y docker-compose fue fundamental para lograr que la aplicación fuera desplegada fácilmente en ambiente local y de producción, dando versatilidad y adaptabilidad a los cambios.
- La creación del módulo de reportes fue una gran experiencia para aprender lenguaje de programación funcional y reconocer cómo se trabaja bajo un marco como Scrum.
- El acercamiento a la vida laboral y la aplicación de los conceptos vistos en toda la carrera en este tiempo de desarrollo fue el aprendizaje más importante recibido.

Referencias bibliográficas

[1] Amazon, E. C. (2014). Amazon.

[2] Opara-Martins, J., Sahandi, R., & Tian, F. (2014, November). Critical review of vendor lock-in and its impact on adoption of cloud computing. In International Conference on Information Society (i-Society 2014) (pp. 92-97). IEEE.

[3] Deshpande, A., & Riehle, D. (2008, September). Continuous integration in open source software development. In IFIP International Conference on Open Source Systems (pp. 273-280). Springer, Boston, MA.

[4] Schwaber, K., & Beedle, M. (2002). Agile software development with Scrum (Vol. 1). Upper Saddle River: Prentice Hall.

[5] Odersky, M., Spoon, L., & Venners, B. (2008). Programming in scala. Artima Inc.

[6] Jones, S. P., Washburn, G., & Weirich, S. (2004). Wobbly types: type inference for generalised algebraic data types. Technical Report MS-CIS-05-26, Univ. of Pennsylvania.
[7] Schmiedehausen, K. (2018). Single page application architecture with angular.
[8] Stonebraker, M., & Kemnitz, G. (1991). The POSTGRES next generation database management system. Communications of the ACM, 34(10), 78-92.
[9] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.
[10] Google Cloud Platform. https://cloud.google.com/.