Universidad ORT Uruguay

Facultad de Ingeniería

Arquitectura de Software
Obligatorio

Serafin Revetria (209143)
Santiago Topolansky (228360)
Mauro Minetti (235931)

Repositorio:
https://github.com/ORTArqSoft/RevetriaMinettiTopolansky.git

Entregado como requisito de la materia Arquitectura de Software

24 de junio de 2021

Declaraciones de autoría

Nosotros, Santiago Topolansky, Serafin Revetria y Mauro Minetti declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Arquitectura de Software;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

1. Introducción **3**

2. Descripción del Sistema **4**
2.1. Requerimientos Funcionales 4
2.2. Requerimientos No Funcionales 8
2.2.1. Atributos de Calidad 8
2.2.2. Restricciones 11

3. Descripción de Arquitectura **13**
3.1. Vista de Módulos 13
3.1.1. Vista de Descomposición 13
3.2. Vista de Componentes y Conectores 16
3.2.1. Componentes y Conectores - Reservation System 16
3.2.2. Componentes y Conectores - Asignación de Cupos 19
3.2.3. Componentes y Conectores - Filtros sobre Reservas 21
3.3. Vista de Asignación 22
3.3.1. Vista de Despliegue 22

4. Conclusiones **25**
4.1. Reflexión Final 25
Bibliografía 27

1. Introducción

VacPlaner es un sistema de coordinación logística digital para países que deseen implementar un plan de vacunación durante una emergencia sanitaria como la que transitamos actualmente. Esta herramienta pretende resolver todas las necesidades que una autoridad sanitaria podría tener a la hora de organizar y llevar adelante un plan de vacunación de gran escala, como la administración de vacunatorios en distintos lugares del país, suministro de dosis, sistema de reservas de alto rendimiento disponible 24/7 para la ciudadanía, configuración de roles y permisos para cada funcionalidad y exposición y disponibilidad de los datos para obtener estadísticas de interés sobre el avance del plan, entre otras prestaciones importantes.

El producto actual fue concebido con la intención de realizar una prueba de concepto en un país determinado, Uruguay. Aquí se podría corroborar el correcto funcionamiento de la aplicación e identificar los cambios necesarios, antes de poder implementarlo en los demás países de la región. Por esta razón, desde un primer momento se diseñó la arquitectura base de este sistema tal que cumplieran con la extensibilidad y escalabilidad suficiente para lograr este objetivo sin mayores esfuerzos de adaptación a los distintos clientes (países).

Este documento describe de forma detallada el análisis llevado adelante por el equipo para encontrar las soluciones más acertadas a cada uno de los problemas planteados. Mediante la descripción de distintas vistas, con sus respectivos diagramas, realizaremos una exposición profunda de las estructuras y patrones arquitectónicos seleccionados para arribar a una solución que satisface en la mayor medida posible los requerimientos solicitados.

Nota: Es importante destacar que como cualquier proyecto real, durante el desarrollo del obligatorio, el equipo tuvo que tomar decisiones importantes respecto a la administración de sus recursos, en algunos casos optando por reducir alcances o implementar versiones minimalistas que permitieran verificar al menos la viabilidad de las ideas. Estos casos se encontrarán claramente específicados en la sección correspondiente

2. Descripción del Sistema

2.1. Requerimientos Funcionales

El sistema desarrollado cumple con la siguiente especificación funcional:

1. **Configuración de Plan de Vacunación**: El sistema por defecto no incluirá ninguna información relativa a un país, ni a un plan de vacunación. Para comenzar a configurar un nuevo plan de vacunación, un miembro del equipo técnico deberá dar de alta haciendo uso de **ConfigAPI**:

   - Un nuevo plan de vacunación
   - Estados del país (Departamentos o Estados)
   - Zonas del departamento (o Barrios)
   - Criterios de Asignación deseados en el país
   - Validaciones deseadas sobre las reservas
   - La/s APIs externa de mensajería deseada/s
   - Usuarios y roles deseados (**Mirar Punto 3**)

   Una vez en su lugar esta información básica de un plan de vacunación, el usuario de rol **Autoridad Sanitaria** podrá dar de alta mediante la misma ConfigAPI:

   - Centros de Vacunación (Seleccionando estado y zona)
   - Tipos de Vacuna (Proveyendo nombre y recomendaciones especiales)
   - Periodos de vacunación (Seleccionando un centro de vacunación determinado, un criterio de asignación, cantidad de dosis para distribuir entre sus turnos, tipo de vacuna, así como fechas de inicio y fin del mismo)

   Nota: Para la entrega, sí contaremos con datos de prueba ya configurados simulando que este proceso ya se llevó a cabo en Uruguay, para permitir una prueba más real, ágil y robusta del sistema.

2. **Suministro de Vacunas**: Una vez cargadas las entidades en el punto anterior, la autoridad sanitaria tendrá la posibilidad de administrar nuevos cupos a un periodo de vacunación particular.

   Esta operación también se llevará a cabo mediante ConfigAPI.

3. **Autenticación y Gestión de Permisos**: El sistema permite una estricta protección del acceso a las funcionalidades. Los usuarios serán creados con usuario, contraseña y rol. Luego, podrán autenticarse contra **AuthenticationAPI**, una API específicamente diseñada para el acceso al sistema. Al momento de la autenticación, obtendrán un token que les permitirá operar con el sistema durante una cantidad de tiempo establecida en el archivo de configuración (Sencillamente configurable).

   Por defecto estos son los roles del sistema, y los permisos asociados a los mismos:

   - **SuperAdmin**
     - Crear Usuarios
     - ABM de Centros de Vacunación
     - ABM de tipos de Vacuna
     - ABM de Periodos de Vacunación
     - ABM de Estados
     - ABM de Zonas
     - Alta-Baja de Criterios de Asignación
     - Alta-Baja de Validaciones de Reserva
     - ABM de APIs externas
     - Registrar Acto Vacunatorio
     - Consultar Estadísticas
   - **Autoridad Sanitaria**
     - ABM de Centros de Vacunación
     - ABM de tipos de Vacuna
     - ABM de Periodos de Vacunación
   - **Rol técnico**
     - Crear Usuarios
     - ABM de Estados
     - ABM de Zonas
     - Alta-Baja de Criterios de Asignación
     - Alta-Baja de Validaciones de Reserva
     - ABM de APIs externas
   - **Vacunador**
     - Registrar Acto Vacunatorio
   - **Estadística**
     - Consultar Estadísticas

   Ya se desplegará el sistema con esta serie de permisos preestablecidos, aunque los roles y los permisos asociados a los mismos son sencillamente configurables mediante un archivo de configuración.

4. **Agenda para Reservas**: La funcionalidad principal del sistema permitirá que la ciudadanía pueda reservar un turno de vacunación proveyendo sus datos personales, y horario, fecha y lugar de preferencia. El sistema validará la información de acuerdo a las validaciones propias del plan de vacunación actual y corroborará la validez de los datos provistos contra una API externa disponibilizada por cada país.

   Se realizará el mejor esfuerzo por asignar a la persona en un vacunatorio con cupos disponibles en el turno/día seleccionado, y de encontrarse un cupo, se le devolverá a la persona los datos de su reserva. En el caso de no poder encontrar un cupo disponible, se persistirá la información del ciudadano y se mantendrá registrado como pendiente de asignación, para que se lo pueda considerar cuando se asignen nuevos cupos.

   En ambos casos el sistema responderá con un código de reserva, que el ciudadano podrá utilizar luego para consultar o cancelar su reserva (Ver punto 5).

5. **Consulta y Cancelación de Reservas**: El sistema permitirá a los ciudadanos que se hayan agendado, o hayan quedado pendientes de asignar, que haciendo uso de una api dedicada **APICheckReservationAPI**, puedan verificar el estado de su reserva, o puedan cancelarla en caso de así desearlo.

6. **Integración con APIs de Mensajeria y DNI**: El sistema permitirá utilizar cuantas APIs se deseen de mensajería en simultáneo, siempre que cuenten con la misma interfaz previamente establecida.

   También será posible modificar la url de la API de Registro Civil, para adecuarse a las particularidades de cada país.

   Es interesante destacar que ambas modificaciones serán realizadas en tiempo de ejecución, mientras el sistema se encuentra desplegado. Esto permitirá no frenar la ejecución del mismo para mantenimiento.

   En caso de que se deseen implementar otros cambios, como la forma de comunicarse con la API, o nuevas reglas de negocio, se deberá modificar la aplicación para hacerlo.

7. **Generación de Estadísticas**: Dado que el sistema es una herramienta de interés nacional por la importante que será el correcto desarrollo del plan de vacunación, se expondrá una herramienta independiente del sistema de gestión del plan, llamada **VacQueryTool**, que permitirá a la población o entidades interesadas realizar consultas complejas sobre el desarrollo del plan de vacunación. Estas consultas serán veloces y actualizadas en tiempo real.

Por otro lado, otras ciertas consultas de información estratégica relacionada a la distribución de las reservas en el territorio nacional, serán de acceso exclusivo para el rol estadístico, como se mencionó en el punto 3.

Estas son las consultas que permitirá realizar el sistema en su etapa inicial:

a) Listar por estado y zona la cantidad de vacunas suministradas entre 2 fechas. (Acceso público)

b) Listar por estado la cantidad de reservas sin asignar (Acceso público)

c) Listar por estado y turno la cantidad de vacunas suministradas (Acceso público)

d) Listar por estado y zona la cantidad actual de reservas pendientes de asignación (Acceso restringido).

En el caso de que se quisieran implementar nuevas consultas, se debería modificar el código de la aplicación **VacQueryTool**, deteniendo el servicio durante cierto periodo de tiempo. Sin embargo, no se vería reflejado en el resto del sistema (Gestión del Plan y Reservas), que funciona de forma independiente.

Nota. En este requerimiento, nos faltó implementar una consulta de las solicitadas, de Acceso Restringido. Esta pedía "Dado un vacunatorio y una fecha el sistema debe desplegar los datos sobre la cantidad de vacunas dadas hasta el momento y la cantidad de vacunas remanentes". No se implementó por falta de tiempo, aunque hubiera sido viable hacerlo siguiendo el modelo conceptual de las anteriores.

8. **Detalle de Actividad y Gestión de Errores** : El sistema está construido considerando que se le va a dar un uso intensivo al mismo, y que deberá soportar una alta demanda. En estos escenarios es común el surgimiento de errores inesperados, por lo que nuestro sistema cuenta con un registro estricto de Logs, que mantienen una traza de ejecución que podría ser inspeccionada en caso de fallos, permitiendo un análisis rápido y eficaz de las causas.

Bugs Conocidos:

Al momento de la entrega casi todas las funcionalidades implementadas se comportan de acuerdo a lo especificado arriba. Sin embargo, tenemos conocimiento de algunos casos de borde donde el sistema podría no comportarse como se desea, pero que optamos por no corregir debido al pequeño margen de tiempo restante. Estos son:

- Falta validación de existencia de código de zona y estado. En el caso de que una persona se intente agendar para una zona o estado inexistente, si los demás datos son correctos, el módulo de reservas intentará hacer la reserva pero no encontrará ningún cupo disponible. Así es que la reserva será almacenada como "no-asignada", hasta que se abran nuevos cupos. Sin embargo, esto nunca

ocurrirá ya que no existe la combinación de estado y zona elegida por el usuario.

De cualquier forma, siempre existe para el usuario la posibilidad de cancelar la reserva actual, por lo que aún si no se corrigiera este defecto, sería posible agendar al 100 % de las personas que así lo deseen. Además, consideramos que posiblemente esos sean datos validados en front end también, brindando una capa extra de validación.

- Procesamiento repetido de mensajes en **SyncService**. En algunas ocasiones, el servicio de sincronización que carga los datos a la base de datos para consultas pesadas, procesa más de una vez el mismo mensaje proveniente de una cola de mensaje que reporta cada vez que se suministra un nuevo acto vacunatorio. No hemos determinado a qué se debe este error ni en que condiciones particulares se configura, ya que no es un error común, por lo que no le hemos otorgado mayor relevancia.
- Error de configuración de clave primaria. Al momento de agregar una nueva zona, un error en la selección de las claves primarias permite al usuario agregar la zona aunque ya exista otra con el mismo código de zona y código de estado.
- Fecha no incluida. Al momento de ejecutar la consulta de VacQueryTool c), notamos que la fecha "hasta"que se pasa no es incluida en los datos a seleccionar. Implicaría un arreglo sencillo de condición lógica en la consulta.

2.2. Requerimientos No Funcionales

2.2.1. Atributos de Calidad

Una de las primeras actividades que llevamos a cabo como equipo antes de comenzar a pensar en soluciones para resolver la arquitectura, fue el análisis de los atributos de calidad (ACs). Los ACs son aquellos aspectos donde la aplicación debe cumplir condiciones particulares de comportamiento, de forma claramente medible. Más allá de la funcionalidad (RF), los ACs son los que terminan determinando las tácticas y patrones a seleccionar para el mejor diseño de la arquitectura.

A continuación presentamos el análisis de ACs realizado para la elección de nuestra arquitectura, cruzándo dichos requerimientos con los requerimientos funcionales listados en la sección 2.1.

Modificabilidad
Refiere a la capacidad de un sistema de cambiar, centrándose en el costo y riesgo asociados. Intenta considerar qué elementos son más propensos a cambiar, la probabilidad de cambio de los mismos, cuándo y quién realiza los cambios, y cuál es su costo.[3]

El sistema debe permitir la mayor configurabilidad posible para permitir una adopción ágil y accesible a las particularidades de los distintos países.

1. **RF1.** Configuración del Plan de Vacunación mediante APIs. Esta característica permite que el sistema no deba ser modificado y recompilado antes de ser puesto en producción en un país u otro. Es posible gestionar todas las entidades descriptas en el RF1 de la sección 2.1 mediante APIs en tiempo de ejecución.

2. **RF3.** Aunque la letra original solicitaba la existencia de algunos roles particulares, se tuvo en cuenta la posibilidad de que entre un país y otro existan diferentes categorías de autoridades y demás interesados. Por esta razón, quisimos que se pudieran crear fácilmente nuevos roles (y asignarles a cada uno permisos de un conjunto preestablecido), modificando un sencillo archivo de configuración.

3. **RF4.** Los mecanismos de validación sobre las reservas son modificables a través de una api, buscando minimizar así el costo de cambio. Esta probablemente sería una característica muy cambiable de un país a otro, debido a las particularidades propias de cada cultura y plan de vacunación.

4. **RF6.** Para una adecuación más rápida a las particularidades de cada país con el menor costo posible, se tuvo en cuenta que los proveedores de mensajería e identidad civil serían variables y modificables en tiempo de ejecución. Por esta razón se implementó la posibilidad de agregar multiples apis de mensajería, así como eliminarlas. Respecto al proveedor de identidad nacional, se permite agregar y modificar el mismo con el mismo mecanismo.

   Nota: Aunque nuestra solución reduce el costo de cambio de forma significativa, solo permite la modificación de la url utilizada como api. El resto de los parámetros, métodos, verbos, y manejo de errores se asumieron invariables de api en api. Entendemos que esto es una debilidad de nuestra solución.

5. **RF8.** En los requerimientos originales se especificaba la necesidad de poder cambiar las herramientas o librerías concretas que se utilicen para producir la información de logs con el menor costo posible. Tuvimos en cuenta esta necesidad a la hora de diseñar nuestro mecanismo de generación de logs.

Escalabilidad

La escalabilidad es la abilida de un sistema de manejar aumentos de carga sin comprometer la performance. Existen dos estrategias para mejorar la escalabilidad: [2]

- Vertical: Consiste en la estrategia de agregar más recursos, es decir, más capacidad de cómputo (memoria, discos, CPU). Generalmente implica una mejora en el hardware.

- Horizontal: Consiste en aumentar la cantidad de instancias de ejecución y dividir entre ellas la carga. No cualquier sistema puede ser replicado por eso es importante que estos sistemas sean concebidos desde el principio con esta característica en mente.

1. **RF4**. La funcionalidad de reservas debería estar correctamente diseñada para soportar altos picos de carga sin perjudicar la performance, haciendo uso de la técnica de escalabilidad horizontal.

Interoperabilidad

La interoperabilidad es un AC responsable por la transmisión de datos y su intercambio con otros sistemas externos. Un sistema bien diseñado facilita la integración con sistemas de terceros.[3]

1. **RF6**. VacPlanner debía permitir la integración con distintos mecanismos de mensajería e identificación civil para poder adaptarse con un costo de cambio bajo (Ver modificabilidad) a la situación de cada país.

2. **RF8**. Se solicitaba como un requerimiento deseado la capacidad de cambiar el proveedor de logs con facilidad, integrando con sencillez las distintas interfaces.

Seguridad

El atributo de calidad Seguridad es una medida de la habilidad de un sistema de proteger la información y los datos del acceso desautorizado.

1. **RF1**. La configuración de un plan de vacunación debía ser llevada a cabo por aquellos roles debidamente autorizados. En este sentido, nuestra aplicación tuvo en consideración la necesidad de requerir permisos para realizar las distintas operaciones.

2. **RF2**. El suministro de vacunas debía ser únicamente disponibilizado para aquellos usuarios vacunadores debidamente autenticados.

3. **RF5** Algunas de las consultas complejas sobre la base de datos debían ser accesibles únicamente por los administradores del sistema, para poder evaluar el desarrollo de la campaña y tomar decisiones estratégicas en este sentido.

4. **RF6** La información de los usuarios que solicitan agenda para vacunarse es sumamente confidencial y sensible, por lo que el sistema debía permitir la integración con los servicios de autenticación de cada país, pero haciendo un uso responsable de la misma. Nuestra solución no conserva ningún tipo de información personal más allá de la esencial para identificar al usuario.

Disponibilidad

La disponibilidad es parte de otro AC denominado "fiabilidad", y suele ser expresado como la relación entre el tiempo en que el sistema está disponible sobre el tiempo total de trabajo.
Esto suele ser un atributo importante en la medida que los sistemas prestan un servicio esencial, ya que a mayor necesidad del mismo, mayor es el impacto de una potencial caída o baja del sistema.[2]

- **RF4** La funcionalidad de agenda cumple una función crítica, por lo que se esperaba estuviera disponible la mayor parte del tiempo, sin necesidad de darla de baja para realizar cambios en la lógica de negocio, realizar mantenimiento, o consultar datos de la base de datos. Además, se esperaba un buen manejo de errores para evitar caídas inesperadas.

Rendimiento (Performance)

Performance es uno de los atributos de calidad centrales. Se trata de caracterizar los eventos que pueden ocurrir (y cuando), y el tiempo de respuesta del sistema a los mismos.[3].

1. **RF4.** La funcionalidad principal del sistema es la prestación de agenda de reservas de cara a la ciudadanía. Estas reservas debían poder realizarse prácticamente en tiempo real, con un tiempo máximo de respuesta admitido de 5 minutos. Pasado este tiempo, ya no se consideraría correcta la solicitud y se debería descartar la solicitud.

2. **RF7.** La herramienta de generación de estadísticas VacQueryTool tenía una especificación clara sobre el rendimiento esperado. Se esperaba una latencia menor a los 2 segundos en promedio para consultas complejas.

Ortogonalidad

Habiendo realizado el análisis anterior, fácilmente llegamos a esta matriz de ortogonalidad, dónde podemos ver un cruce directo entre los RF detectados en la sección 1.2 y los distintos ACs a considerar.

	PERFORMANCE	DISPONIBILIDAD	MODIFICABILIDAD	ESCALABILIDAD	SEGURIDAD	INTEROPERABILIDAD	PERFORMANCE
RF1			✓		✓		
RF2					✓		
RF3			✓				
RF4		✓	✓	✓			✓
RF5					✓		
RF6			✓		✓	✓	
RF7	✓						✓
RF8			✓			✓	

Figura 2.1: Matriz de Ortogonalidad RF/AC

2.2.2. Restricciones

El otro tipo de Requerimientos no funcionales que forman parte de la especificación de un sistema son las restricciones. Estas son aquellas condiciones que se deben aceptar e incorporar al diseño, como umbrales estrictos de conducta para ciertas

prestaciones.

En este documentos optamos por listar los requerimientos específicos a cómo debe funcionar el sistema en la sección anterior, relacionándolos con los ACs. Sin embargo, también hay otro tipo de restricciones, aquellas más vinculadas al negocio o ambiente en el que se ejecuta el proyecto y los factores externos, las cuales analizaremos a continuación:

1. Tecnología Node.js. Una de las restricciones planteadas por el "cliente"para este proyecto fue la elección de la tecnología. Se seleccionó el framework de javascript Node.js [4] como lenguaje de programación para llevar a cabo el proyecto.

2. Tiempos. Otra restricción del proyecto fue ajustarse al cronograma de trabajo del curso. A pesar de que técnicamente se dio inicio al mismo en el mes de Abril, fuimos incorporando herramientas para lograr el propósito en las semanas siguientes, por lo que el márgen de tiempo para realizar el trabajo terminó siendo acotado.

3. Forma de trabajo virtual. Debido a las circumstancias actuales, la totalidad del proyecto tuvo que ser llevada adelante por medios virtuales. No fue posible en ningún momento la modalidad de trabajo presencial, lo que supuso un desafío extra para un proyecto de semejante magnitud, requiriendo mejor organización, comunicación y división de tareas.

4. Composición de los equipos. Aunque hubo libertad para elegir los equipos, una restricción era el número de 3 integrantes por equipo.

3. Descripción de Arquitectura

Además de contar con una arquitectura robusta, diseñada teniendo en cuenta los requerimientos funcionales y no funcionales de un sistema, hay un factor más igual de importante que podría definir el éxito o el fracaso de su implementación: La eficacia con la que es comunicada. Por esta razón, dedicaremos esta sección a la presentación del diseño logrado mediante el reconocido estándar Views and Beyond, del SIE (Software Engineering Institute) de Carnegie Mellon University.

Views and Beyond, o V&B es uno de los distintos estándares que apelan a uniformizar la manera de comunicar arquitecturas de sistemas. Como el nombre lo indica, este enfoque "utiliza el concepto de Vista como el principio de organización principal para documentar una arquitectura. Una vista representa un conjunto de elementos de un sistema y las relaciones existentes entre estos. El principio principal de V&B es que documentar una arquitectura incluye documentar las vistas importantes, pero también aquella información que excede a las mismas, aplicando a más de una."[1]

3.1. Vista de Módulos

3.1.1. Vista de Descomposición

Representación Primaria

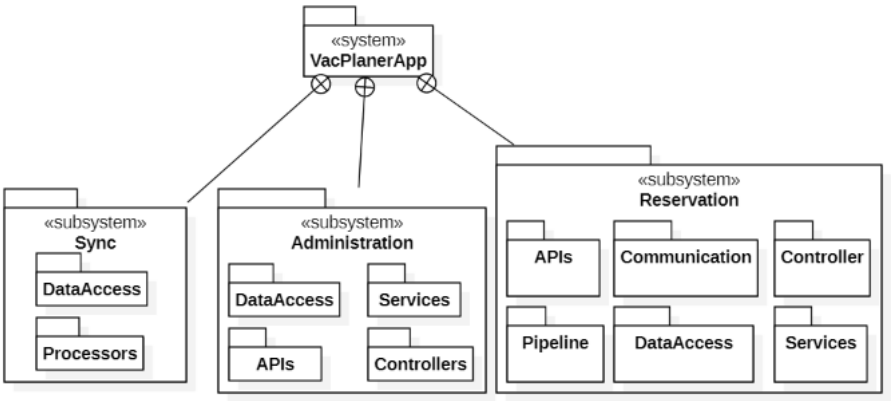

Figura 3.1: Diagrama de Descomposición del Sistema

Catálogo de Elementos

Elemento	Tipo	Responsabilidad
Reservation	Subsistema	Provee el servicio de agenda a los usuarios finales
Administration	SubSistema	Contiene las apis que permiten gestionar los planes de vacunación, junto a toda la lógica de negocio para los requerimientos funcionales especificados en la sección 1.1
Sync	SubSistema	Se encarga exclusivamente de sincronizar datos entre distintas partes del sistema. En particular, se encarga de adaptar los modelos de datos para alimentar la base de datos de consultas cada vez que hay un nuevo evento (actos vacunatorios, nuevas reservas pendientes, o reservas pendientes que se asignan).
DataAccess	Paquete	Contiene las clases de acceso a bases de datos, con la lógica necesaria.
Processors	Paquete	Contiene los workers que procesan los objetos de la MQ de forma asincrónica.
APIs	Paquete	Contiene las clases de APIs, con rutas y middlewares seleccionados.
Services	Paquete	Agrupa los servicios que funcionan independientemente del ciclo de vida de las APIs. Cronjobs, clases estáticas, funcionalidaes específicas, entre otros.
Controllers	Paquete	Contiene las clases de Controladores, donde se ubica la lógica de negocio de cada entidad.
Pipeline	Módulo	Adaptación del Pipeline visto en clase. Contiene filtros de validación para las reservas cargados dinámicamente.
Communication	Módulo	Permite al subsistema de reservas comunicar al resto del sistema las nuevas reservas mediante una MQ, difiriendo en el tiempo el momento del impacto de las reservas en la base de datos.

Decisiones de Diseño

- **Subsistema Reservation**: Teniendo en cuenta lo esencial del RF4. y los ACs analizados para el mismo (Disponibilidad, Escalabilidad horizontal y Performance), se decidió independizar este servicio en un subsistema aparte, que pudiera replicarse cuantas veces fuera necesario y así poder soportar una alta carga. Además, al independizarlo de las otras partes del sistema, disminuyen las probabilidades de bajas del servicio por problemas en otras partes, mejorando la disponibilidad.

- **Subsistema Sync**: Para cumplir con el RF7 optamos por utilizar el patrón **CQRS**[5], que permite independizar las lecturas pesadas sobre la base de datos de las escrituras. Era necesario no quitar poder de cómputo ni ocupar la base de datos en lecturas pesadas mientras los demás sistemas operaban (reservas y administración). Por esta razón, optamos por crear un subsistema independiente que sincronizara las bases de datos, adaptando los modelos a otros modelos más adecuados para consultas pesadas.

  Este subsistema tiene dentro suyo un paquete de procesadores que se encargan de recibir y procesar con mucha cantidad de instancias, los mensajes que llegan por las MQs de otras partes del sistema.

- **DataAccess**: Contamos con dos bases de datos independientes, **countryDB** (encargada de almacenar toda la información relativa al plan de vacunación, como reservas, cupos, usuarios, configuraciones particulares, etc.) y **querydatabase** para almacenar los datos adaptados especialmente para las consultas pesadas del RF7. Las clases de DataAccess contienen la lógica necesaria para interactuar con cada una de ellas.

3.2. Vista de Componentes y Conectores

3.2.1. Componentes y Conectores - Reservation System

Representación Primaria (Reservation System)

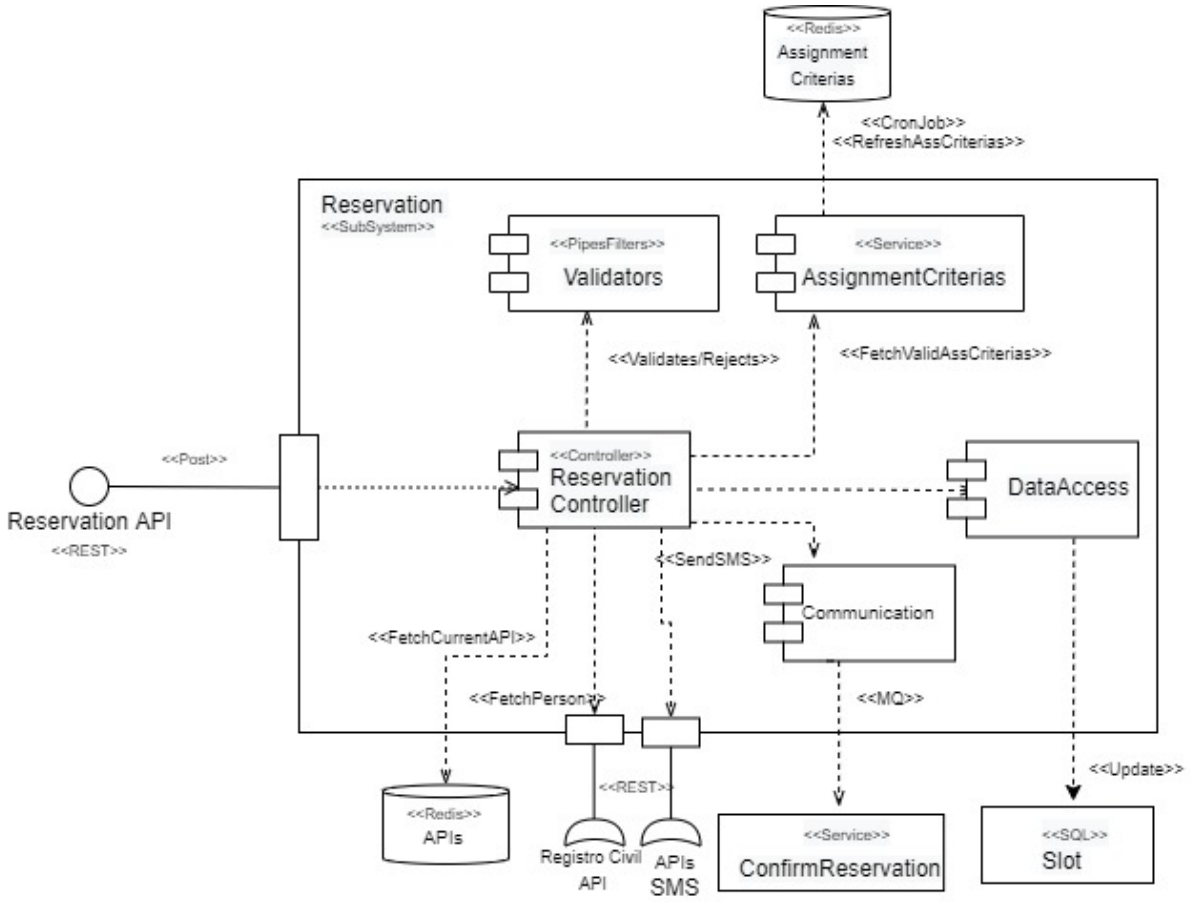

Figura 3.2: Vista Interna. Subsistema Reservations

Catálogo de Elementos

Comp/Conector	Tipo	Descripcion
AssignmentCriterias	Componente	Este servicio se encarga de consultar cada un tiempo X los criterios de asignación actualmente existentes en el sistema, desde la caché compartida Redis. Luego, guarda esta lista de criterios en memoria y cada vez que llega una nueva reserva se utiliza esta copia.
Validators	Componente	Esta implementación de Filters sincrónica permite obtener una validación del objeto de reserva utilizando los filtros definidos mediante la api de configuración. Este componente ejecuta en segundo plano un cronjob que cada cierto tiempo X, actualiza los filtros existentes en una carpeta determinada del FileSystem (por defecto "pipeline/filters")

Decisiones de Diseño

- **APIs en Redis**. Dos de las responsabilidades del proceso de reservas dependen de las apis dinámicamente configuradas en el sistema: Registro Civil Api, y apis de SMS. Por esta razón, optamos por tener en el caché distribuido Redis las apis actualmente utilizadas en el sistema. Así logramos aumentar la velocidad de recuperación de las mismas.

- **Assignment Criterias**: Una de las decisiones de diseño que más nos acercó a cumplir con el AC de modificabilidad en el RF1, fue la decisión de manejar los "criterios de asignaciónçomo una entidad más. Los criterios son objetos que tienen un nombre y un predicado que toma como parámetro una entidad Persona y devuelve verdadero o falso.

  Estos criterios son los que definen a qué categoría de personas se va a vacunar en un determinado periodo de vacunación. Por ende, nuestro subsistema de reservas debía tener un acceso rápido a todos los criterios de asignación cada vez que llegara una nueva reserva, para ver con cuáles cumplía y así poder buscar un vacunatorio que le correspondiera.

  Para compartir esta información de forma rápida y a través de las múltiples instancias replicadas de este subistemas, optamos por mantener esa información en un caché distribuido, **Redis**.

  Esto nos permitió hacer extremadamente extensible el requerimiento de poder agregar nuevos criterios de asignación para los vacunatorios, sin comprometer la performance.

- **Pipeline**: Otra fortaleza de nuestra aplicación fue la modificabilidad permitida en el RF1 respecto a las validaciones sobre las reservas que cada país deseara implementar. De forma similar a con los "criterios de asignación", definimos "validaciones de reserva"que se pueden agregar a través de la api de configuración. Estos no son más un *nombre de campo*, un *predicado*, y un *mensaje de error* a devolver en caso de que una nueva reserva no cumpla un requsito.

  Nuestra API de configuración escribe en una carpeta de "filters"del filesystem cada vez que se le ingresa un nuevo filtro, y luego el subsistema de reservas recarga el pipeline dinámicamente cada un tiempo configurable (por defecto 30 minutos), con los filtros allí presentes.

  Nuevamente, favorecimos la modificabilidad sin comprometer la performance con esta decisión de diseño.

- **Communication**: Buscando cumplir con el AC de performance en el RF4, nuestro subsistema de reservas difiere la inserción de la entidad "Reserva."[en] la base de datos a un momento del tiempo posterior, haciéndo uso de una MQ. Esta lógica se encuentra en el módulo **communication**.

  Esto quiere decir que cuando se recibe una petición de reserva, lo único que hace el subsistema para poder responder velozmente al usuario, es restar un cupo en la tabla de cupos (en un vacunatorio que vacune en la fecha solicitada, y a personas de su categoría), y despacha un mensaje a la cola para que se procese en otro momento por otro componente.

- **Tabla Slot**: Una decisión clave que nos permitió superar el desafío de latencia esperada para el servicio de Reservas, fue el uso de una tabla Cupos. Teniendo en cuenta que este subsistema podría tener múltiples instancias ejecutando en paralelo, era necesario evitar las condiciones de carrera en el acceso a las bases de datos. Además, para poder lograr un buen rendimiento, no era aceptable acceder más de una vez a la base de datos para realizar una reserva. Por esta razón, tuvimos que diseñar una técnica que mediante un único acceso: **1)** Determinara si existía un cupo válido para asignarle al usuario que se pretendía agendar, y **2)** Reservara ese cupo para asegurarse que ninguna otra instancia del sistema lo reservara en el mismo momento.

  La idea que tuvo nuestro equipo para resolver este problema, fue hacer uso de un recurso de la base de datos relacional POSTGRES, llamado "**RETURNING**"[6]. Básicamente, en las consultas UPDATE de postgres, este recurso retorna la/s filas modificadas, evitando así tener que volver a acceder haciendo otra consulta SELECT. Haciéndo un diseño inteligente de esta tabla, donde cada fila representa un turno de un día de un periodo de vacunación, pudimos lograr nuestro objetivo.

3.2.2. Componentes y Conectores - Asignación de Cupos

Representación Primaria (Administration System)

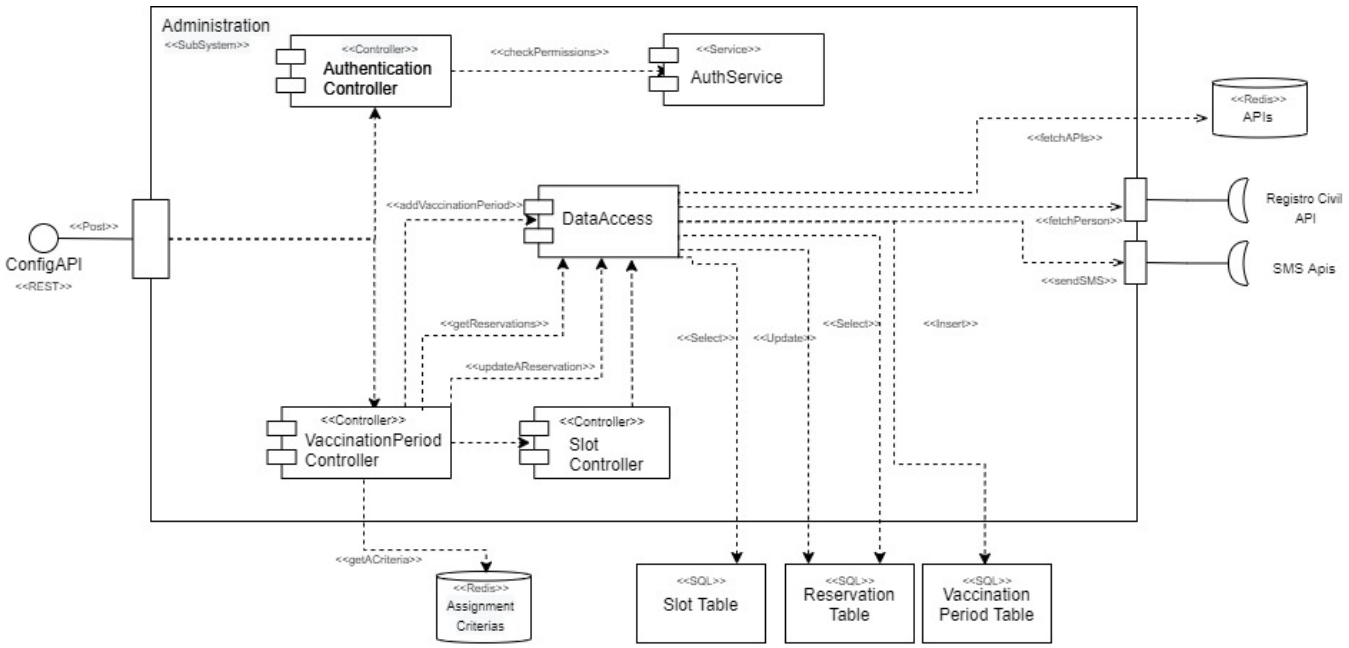

Figura 3.3: Vista Interna. Subsistema Administration

Catálogo de Elementos

Elemento	Tipo	Responsabilidad
AuthenticationService	Componente	Provee el servicio de autenticación basado en JWT
VacPeriodController	Componente	Provee la lógica de gestión de la entidad Periodo de Vacunación. Entre otras responsabilidades, tiene la del algoritmo de Agregar nuevos cupos.
SlotController	Componente	Provee la lógica de gestión de la entidad Cupos

Decisiones de Diseño

- **Algoritmo de Asignación**. Una de las funcionalidades claves de este sistema fue el algoritmo de asignación de nuevos cupos. En el sistema de administración, cuando a través de la api se añaden nuevos cupos a un periodo de vacunación, este es el flujo definido para lograr cumplir con el RF 4 correctamente.

  Como es sabido, algunas personas pueden no haber sido asignadas en una primera instancia si no se encontró ningún periodo de vacunación adecuado para sus categorías, o con sus preferencias de horario y lugar. Por esta razón, las personas pendientes de asignar son las primeras tenidas en cuenta a la hora de asignarse nuevos cupos. El algoritmo ejecuta los siguientes pasos para cumplir este requerimiento:

  1. Busca en Redis el criterio de asignación del periodo de vacunación al que se va a agregar cupos.
  2. Realiza una consulta a **countryDB**, a la tabla de reservas para traer aquellas reservas pendientes de asignar para el mismo día, zona y estado. También se incluyen aquí todas las personas a las que no se les pudo agendar una reserva y cuya fecha solicitada ya transcurrió.
  3. Utilizando el dni de las personas existente en las entidades "reserva", se recupera la lista de personas pendientes de ser asignadas de la api de Registro Civil de turno.
  4. Se ejecuta el filtro para ver cuántas de estas personas cumplen el criterio de asignación del periodo de vacunación al que se están agregando cupos, y por cada una de ellas, mientras queden cupos, se actualiza su reserva a estado **asignada** en la base de datos, y se despacha un evento a la MQ de reservas asignadas, para que sean sincronizadas por el Sistema de Sincronización.
  5. A quienes se les asignó un cupo, se les envía un SMS haciendo uso de las apis de sms actuales del sistema (recuperadas de Redis en el paso numero **1**
  6. Los cupos sobrantes se dividen entre todos los turnos del periodo de vacunación y se actualiza este registro en la base de datos

- **Acceso restringido**. Al igual que en todos los demás endpoints de la api de configuración, se realiza un chequeo sobre la request HTTP de la existencia de un token JWT válido (encriptado con la llave pública disponible, y no expirado).Este control se realiza con un middleware de la libería Koa para integración con JWT. De no existir el mismo, se rechaza inmediatamente la solicitud. Si existiera el token, se desencripta para evaluar si tiene los permisos necesarios. De existir el permiso "ABM de Periodos de Vacunación", se le permite la ejecución. De lo contrario, se rechaza la misma con un error **403 Unauthorized**.

  Creemos que la decisión de utilizar JWT como mecanismo de seguridad en las partes del sistema que así lo requerían, fue una decisión acertada, que nos evitó tener que gestionar sesiones y sobrecargar la base de datos con solicitudes constantes. Esta decisión acompaña el AC de la performance, cuidando la Seguridad, para cumplir el RF3.

- **Despache de Eventos**. Como se especificó anteriormente, para cumplir con el RF7 (Generación de Estadísticas), una de las decisiones de diseño que tomamos fue la de utilizar un mecanismo de sincronización de bases de datos independiente del resto del sistema. Como la información necesaria para alimentar la "queryDatabase"surge de este proceso, se recurrió a un mecanismo de comunicación asincrónica, las colas de mensaje. Utilizando Bull como proveedor de MQ basado en Redis, despachamos mensajes cada vez que una reserva no-asignada pasaba a estado asignada. Esto evitó que comprometieramos la performance general del sistema y la base de datos principal.

3.2.3. Componentes y Conectores - Filtros sobre Reservas

Representación Primaria (Administration System)

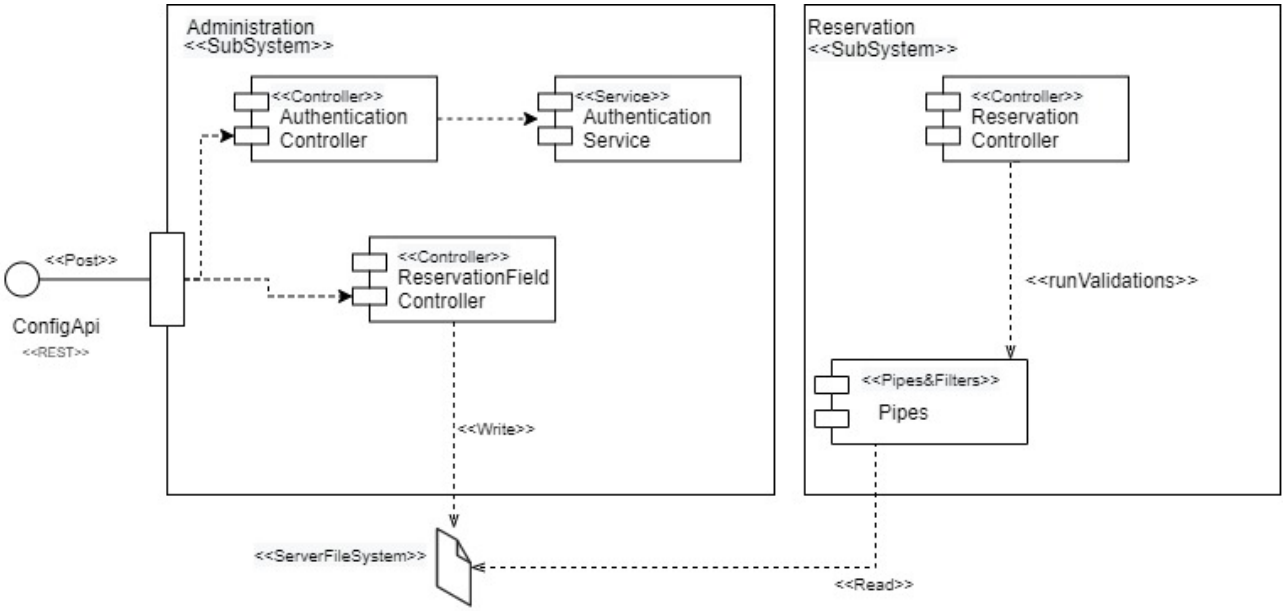

Figura 3.4: Vista Interna. Subsistema Administration

Decisiones de Diseño

- **Carga de Filtros**. Como ya se mencionó en la página 18, cuando se hizo mención al **Pipeline**, para validar la correctitud de las reservas según los criterios establecidos en cada país, nuestro sistema de reservas utiliza una instancia de filtros encadenados sincrónicamente en memoria, similar a una implementación de Pipes and Filters. Para cargar estos filtros optamos por utilizar la misma api de configuración que se puede utilizar para configurar el resto de los atributos del sistema. Así es que cuando llega un nuevo filtro al sistema, se escribe al FileSystem del Sistema de Reservas, en la carpeta establecida como predeterminada para recuperarlos luego.

  Nota: Si ya existiera un filtro con el mismo nombre, se sobreescribirá actualizando el archivo.

3.3. Vista de Asignación

3.3.1. Vista de Despliegue

Representación Primaria (Administration System)

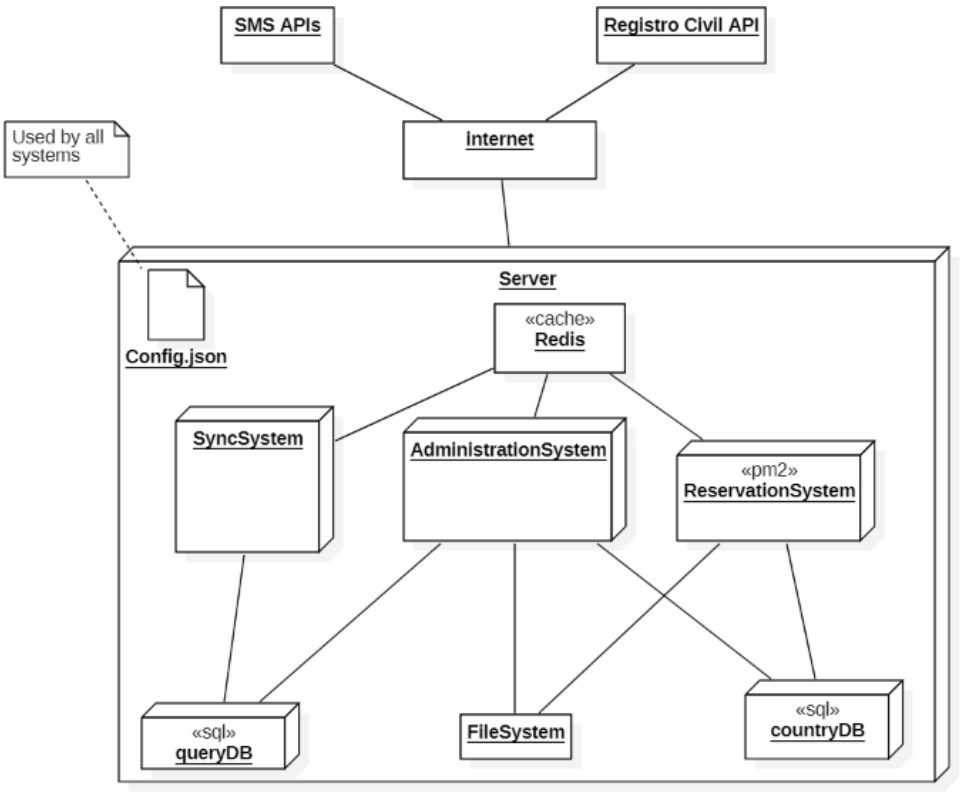

Figura 3.5: Vista de Despliegue

Catálogo de Elementos

Elemento	Tipo	Caracteristicas	Descripcion
pm2	Nodos	Dependiente de los Cores de la CPU	Esta tecnología permite la replicación de instancias del sistema de Reservas (Clustering), favoreciendo el AC de Performance.
config.json	Componente	Cambios en config se ven reflejados luego de reiniciar el sistema	Archivo utilizado para configurar: Puertos, Roles y sus permisos, configuracion de bases de datos, instancias de procesadores de MQ, y vida útil de los tokens,
Redis	Componente	puerto: 6379	Se utiliza como caché distribuida entre procesos
queryDB	Componente	Postgres SQL	Se utiliza para almacenar la información relativa a las consultas pesadas
countryDB	Componente	Postgres SQL	Se utiliza para gestionar las entidades del plan de vacunación

Nota: Anteriormente ya se hizo referencia a los nodos (Cada uno de los 3 sistemas).
Nota 2: Utilizamos Docker para levantar los servicios Postrgres, Redis, y PGAdmin (Cliente Postgres), facilitándo la coordinación del equipo en cuanto a la distribución de los servicios. Se adjunta un archivo **Readme.md** en el repositorio el cual contiene instrucciones para ejecutar dichos servicios.

4. Conclusiones

4.1. Reflexión Final

Ya habiendo culminado el proyecto **VacPlaner**, procedemos a dejar unas reflexiones finales sobre lo que fue el obligatorio. En poco menos de tres meses, diseñamos y desarrollamos un backend modular de alta complejidad utilizando Node.js, un framework con el que ninguno de los integrantes del equipo tenía experiencia previa. Para lograr esto tuvimos que familiarizarnos con una serie de tecnologías novedosas, como Docker para la configuración de los ambientes, motores de bases de datos como Postgres, MySQL y Redis, librerías de Node para cumplir variadas funciones, como Winston, Sequelize, JWT, Bull, pg, Nodemon, entre otras. Además, tuvimos el desafío real de desarrollar un sistema complejo utilizando el paradigma del paralelismo y familiarizándonos con toda la complejidad intrínseca al mundo de la concurrencia y las apis basadas en promesas. En definitiva, en cuestión de meses transcurrimos por una serie de aprendizajes que nos dejan orgullosos del trabajo realizado, y sinceramente creemos que nos hicieron crecer en nuestra faceta profesional.

Aunque somos conscientes de que existen puntos de mejora (como los que detallamos en la sección de bugs y las **Notas** a lo largo de la documentación), quedamos muy conformes con nuestras decisiones de diseño, que ya consideramos lo suficientemente bien fundadas como para elevar la calidad de nuestros productos. En la última etapa del proyecto, nos motivó el desafío de lograr una integración exitosa de los distintos módulos desarrollados independientemente, y ver funcionar el sistema como originalmente lo imaginábamos. Además, en lo que corresponde al diseño arquitectónico, la aplicación de patrones y tácticas nos permitió percibir desde otra perspectiva el mundo del desarrollo de software, que previamente entendíamos como "el proceso de escribir código"principalmente.

Creemos que logramos un producto final que satisface los requerimientos especificados y en la mayor medida posible respeta los criterios de aceptación definidos en la rúbrica. También tuvimos la experiencia y el desafío de aplicar metodologías de desarrollo profesionales en el proceso de trabajo, como **GitFlow** como proceso de SCM para el manejo consistente del repositorio y el control de versiones. Este último punto se vio particularmente afectado en los últimos días, cuando tuvimos que acelerar el ritmo de desarrollo, y eso complicó en alguna instancia el uso de **ramas** y **features**. Siguiendo por la misma línea, tomamos como premisa el desarrollo completo en **idioma Inglés**, incluyendo comentarios y commits, a excepción de los

mensajes de cara al usuario.

Una vez más evaluamos positivo nuestro desempeño como equipo de trabajo, ya que pudimos distribuir las responsabilidades de forma equitativa y acompañarnos en el enfrentamiento de nuevas tecnologías.

Bibliografía

[1] https://resources.sei.cmu.edu. Views and beyond collection.

[2] Nikolay Ashanin. Quality attributes in software architecture.

[3] Eben Hewitt, *Architectural Patterns and Tactics*.

[4] Node.js. Acerca de node.

[5] Martin Fowler. Cqrs.

[6] Postgres. Postgres sql update.