Documento de diseño de software

PSI Covid Control

Version 1.0

Presentado por :
Juan Esteban Cardona
Santiago Melo
Cristian Andrés Carabalí

Índice general

1 Introducción

1.1. Propósito

El propósito de este documento es presentar una descripción detallada de los diseños de la aplicación PSI COVID Control, sistema web para la gestión de accesos y aforo por COVID-19.

Además de documentar el trabajo realizado por el equipo de desarrollo sobre la arquitectura de software y las diferentes vistas: casos de uso, lógica, de proceso, física y despliegue, contiene todo el proceso e investigación realizada por el equipo para escoger de manera adecuada la arquitectura de software y demás decisiones requeridas para realizar un apropiado diseño del sistema que va a implementarse.

Por último, este documento podría usarse para modificar o actualizar el diseño de la aplicación.

1.2. Alcance

Lo que se busca al realizar el diseño del sistema desde las diferentes vistas y su arquitectura es lograr un mayor entendimiento y tener la información necesaria para pasar al proceso de implementación de este, además de tener la respectiva documentación y razones justificadas de las decisiones tomadas por el equipo de desarrollo.

Por otro lado, se tiene como objetivo hacer un análisis y estudio de los distintos requisitos no funcionales que afectan la arquitectura de este sistema, y así, poder priorizarlos y hacer una comparación acorde con lo que se desea implementar.

1.3. Definiciones, acrónimos y abreviaciones

- API: Es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.
- Back end: Es la capa de acceso a datos de un software, que no es directamente accesible por los usuarios.
- Evento: Ocurrencia o cambio significativo en el estado del hardware o software del sistema.

- Front end: Parte de un programa o dispositivo a la que un usuario puede acceder directamente.

1.4. Referencias

[1]"What Is Service-Oriented Architecture?". Medium.com. https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec [Accessed 22 october 2020]

[2] M. Richards "Chapter 4. Microservices Architecture Pattern". Oreilly.com. https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch04.html [Accessed 22 october 2020]

[3] "What is event-driven architecture?". Redhat.com. https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture [Accessed 22 october 2020]

[4]"What is an application architecture?". Redhat.com. https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture [Accessed 22 october 2020]

[5] "3-Tier Architecture: A Complete Overview". Jinfonet.com. https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/ [Accessed 22 october 2020]

[6] J. Rabelo. " Three-Tier Architecture " Techopedia.com.https://www.techopedia.com/definition/24649/three-tier-architecture [Accessed 22 october 2020]

[7] "What is a Relational Database Management System?". Codeacademy.com. https://www.codecademy.com/articles/what-is-rdbms-sql [Accessed 28 october 2020]

[8] A. Brody "SQL Vs NoSQL: The Differences Explained". Blog.panoply.io. https://blog.panoply.io/sql-or-nosql-that-is-the-question [Accessed 28 october 2020]

[9] R. Wang and Z. Yang, "SQL vs NoSQL: A Performance Comparison". [Online].

Available: https://www.cs.rochester.edu/courses/261/fall2017/termpaper/submissions/06/Paper.pdf [Accessed 22 october 2020]

[10] "Graph database: bases de datos para una interconexión eficiente". [Online]. Available: https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/graph-database/ [Accessed november 5 2020]

2 Representación arquitectónica

2.1. Atributos de calidad

Según los requisitos no funcionales obtenidos por el equipo de desarrollo se decidió escoger los siguientes atributos de calidad:

- Rendimiento
- Disponibilidad
- Escalabilidad

Estos atributos satisfacen los requisitos no funcionales tales como RNF-05, RNF-06, RNF-07, RNF-08, RNF-10. Los cuales se refieren a tiempos de respuesta, disponibilidad y escalabilidad de la aplicación. Con estos 3 atributos y una previa investigación se procede a la elección de la arquitectura para el sistema.

2.2. Arquitecturas candidatas

2.2.1. Micro-servicios

Independiente de la topología o estilo de implementación que se elija existen varios conceptos comunes que se aplican a este patrón de arquitectura general. El primer concepto es la noción de unidades desplegadas por separado, esto se refiere a que cada componente de la arquitectura microservicios se implementa como una unidad separada, lo que permite incrementar la escalabilidad y un alto grado de desacoplamiento de aplicaciones y componentes dentro de la aplicación.

El concepto más relevante en este patrón es la noción de componente de servicio. En vez de pensar en servicios dentro de la arquitectura de microservicios es mejor pensar en componentes del servicio, que pueden variar su granularidad desde un solo módulo hasta gran parte de la aplicación. Los componentes del servicio contienen uno o más módulos que representan una función de un solo propósito o una parte independiente de una aplicación grande de negocios. Diseñar el nivel correcto de granularidad de los componentes del servicio es uno de los mayores desafíos de esta arquitectura.

Por otro lado, también se tiene otro concepto importante y es el de arquitectura distribuida, lo que significa que todos los componentes dentro de la arquitectura están completamente desacoplados entre si y se accede a ellos a través de algún tipo de protocolo de acceso remoto. Gracias a ello es que se logra su característica de escalabilidad.

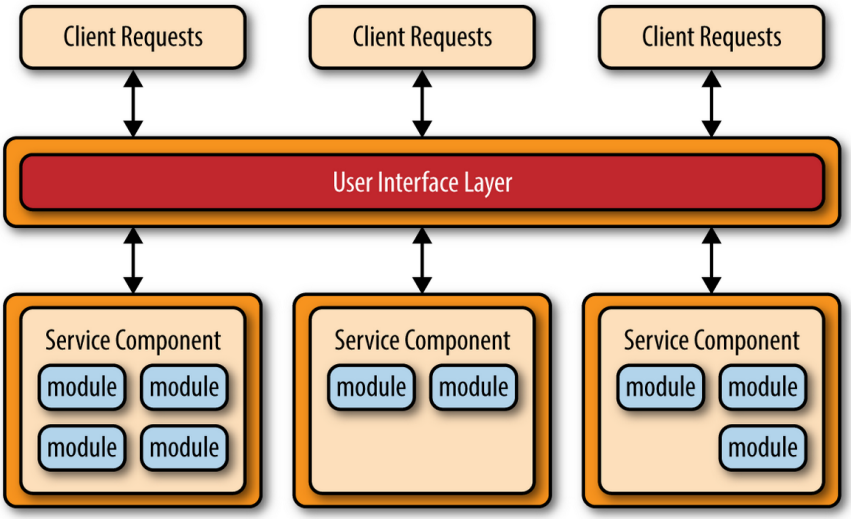


Figura 2.1: Arquitectura básica de microservicios

2.2.2. Capas (3 capas)

Una arquitectura de 3 capas se compone de tres "niveles.º "capas", que, a menudo, se usan en aplicaciones como un tipo específico de sistema cliente-servidor. Brinda muchos beneficios para los entornos de desarrollo al modularizar la interfaz de usuario, la lógica comercial y las capas de almacenamiento de datos. Hacer esto brinda mayor flexibilidad a los equipos de desarrollo al permitir actualizar una parte especifica de la aplicación independientemente de las otras partes. Los 3 niveles de esta arquitectura son los siguientes:

- Presentación: Esta capa es el front end y es la interfaz de usuario, esta interfaz de usuario es a menudo gráfica y accesible a través de un navegador web. Normalmente este nivel se basa en tecnologías web como HTML5, Javascript, CSS y se comunica con otras capas a través de APIs.
- Aplicación: Esta capa contiene la lógica de negocio que controla la funcionalidad central de la aplicación realizando un procesamiento detallado. Normalmente este nivel está codificado en lenguajes de programación como Pythom, Java, C++, etc.
- Datos: Esta capa comprende el sistema de almacenamiento de datos / base de datos y la capa de acceso a los datos. Se usan sistemas como MySQL, Oracle, MongoDB, etc. La capa de aplicación accede a los datos a través de APIs.

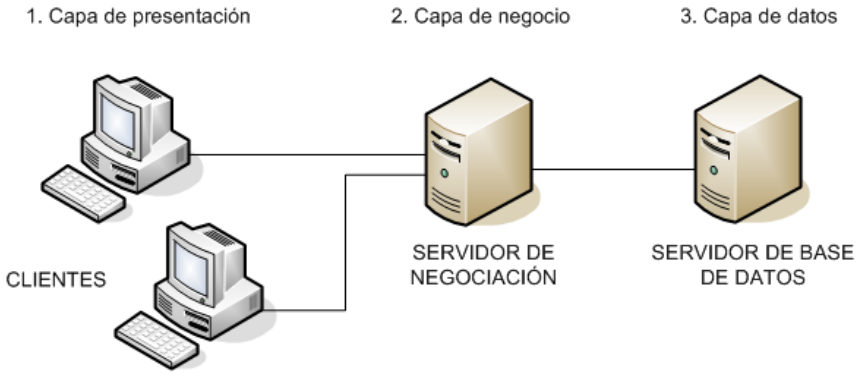


Figura 2.2: Arquitectura 3 capas

El uso de esta arquitectura brinda beneficios tales velocidad de desarrollo, escalabilidad, rendimiento y disponibilidad gracias a la modularización de diferentes niveles de la aplicación.

2.2.3. Orientado a servicios

La arquitectura orientada a servicios es un estilo de diseño de software los servicios son brindados a demás componentes a través de la red por aplicación del componente. Aquí los servicios se comunican entre sí a través del paso de datos o coordinación de actividades por otros servicios.

Existen tres roles en cada uno de los componentes básicos de esta arquitectura, proveedor de servicios; corredor de servicios, registro de servicios, depósito de servicios; y solicitante / consumidor de servicios.

En este tipo de arquitectura hay una buena disponibilidad ya que diferentes servicios están atentos peticiones y la falla de unos no compromete la de otros, adicional a esto es una arquitectura escalable ya que su crecimiento dependerá de cargar mas servicios; diferentes servidores correrán en un mismo ambiente.
Por último un aspecto importante para resaltar es la mantenibilidad ya que por hecho de que los servicios son independientes, esto nos lleva a la posibilidad de manejar actualizaciones sin alterar otros servicios.

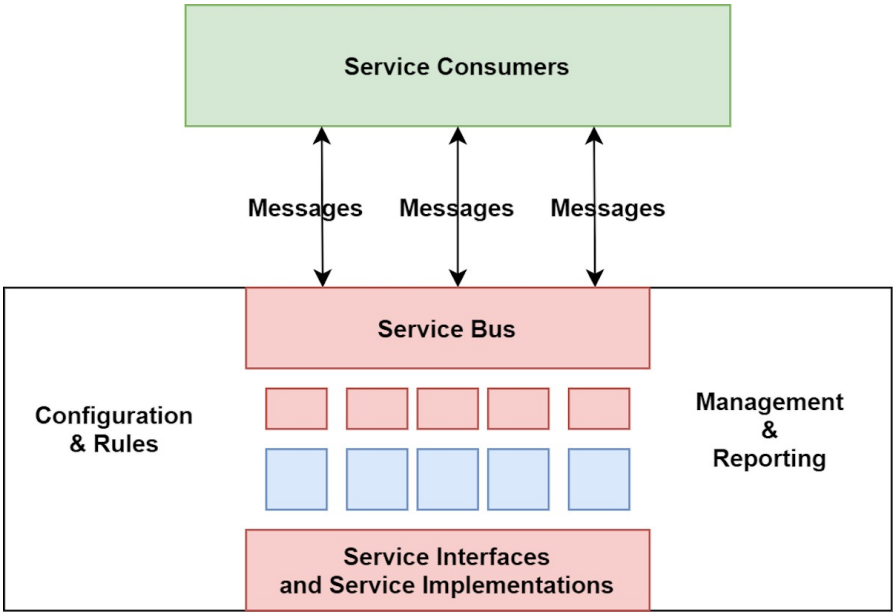


Figura 2.3: Arquitectura SOA

2.2.4. Orientado a eventos

Esta arquitectura esta formada por productores y consumidores de eventos. El productor detecta o percibe un evento y lo representa como un mensaje. No conoce al consumidor del evento ni el resultado de un evento. Después que se ha detectado un evento, el productor lo transmite a los consumidores a través de canales, donde una plataforma de procesamiento de eventos procesa el evento de forma asincrónica.

Los consumidores de eventos deben estar informados cuando ocurre un evento. Pueden procesar el evento o solo pueden verse afectado por el. Por otro lado, la plataforma de procesamiento de eventos ejecutará la respuesta correcta a un evento y enviará la actividad a los consumidores adecuados.

Esta arquitectura puede ayudar a lograr un sistema flexible que puede adaptarse a los cambios y tomar decisiones en tiempo real, además de mejorar la escalabilidad y capacidad de respuesta de las aplicaciones y acceso a datos. Los eventos se capturan a medida que ocurren a partir de fuentes como dispositivos, aplicaciones y redes de Internet de las cosas, permitiendo a los productores y consumidores de evento compartir información de estado y respuesta en tiempo real.

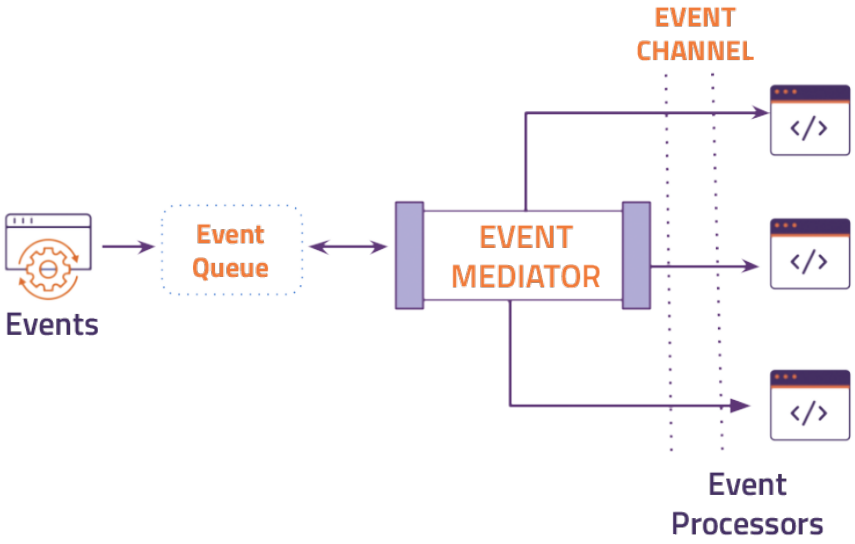


Figura 2.4: Arquitectura event driven (orientado a objetos)

2.3. Comparación y elección de arquitectura

A partir de las arquitecturas presentadas anteriormente se realiza una tabla para comparar los atributos de calidad en cada una de estas:

Arquitectura	Rendimiento	Disponibilidad	Escalabilidad
Micro-servicios	Al tener diferentes puntos de comunicación por cada servicio, esto genera una latencia en las peticiones.	Al tener desacoplamiento la pérdida de un módulo no afecta todo el sistema.	Al estar independientes los servicios pueden escalar verticalmente sin causar problemas en el sistema.
3 Capas	La latencia de comunicación se disminuye al tener estos 3 niveles, ya que se pueden invocar capas que están directamente debajo de ellas.	Al poder tener diferentes módulos en las distintas capas los cuales pueden estar albergados en un número variado de servidores asegura alta disponibilidad pero es muy costoso, aún así, dado que esta arquitectura es monolítica una pérdida de algún módulo involucra pérdida de gran parte del sistema.	Debido a que no está granular el sistema y es una arquitectura monolítica la escalabilidad es compleja.
Orientado a servicios (SOA)	Dado que hay servicios encargados de la distribución de tareas a los diferentes servicios, no hay mucha latencia de comunicación aunque no es el mejor rendimiento cuando es una aplicación con alto nivel de transferencia.	Ya que hay granularidad de los servicios del sistema, hay una alta disponibilidad, pero no tanta como la que posee micro servicios.	Diferentes servidores pueden funcionar en un mismo ambiente del sistema, por esto es escalable, pero no es tan granular comparado a micro-servicios lo que dificulta un poco más la escalabilidad.
Orientado a eventos	Al tener el sistema bien desacoplado y el enrutador de eventos permite una buena capacidad de respuesta de las aplicaciones y el acceso a datos, aún así, dado el desacoplamiento se aumenta la latencia.	Al tener desacoplados los servicios, cada uno solo conoce el enrutador de eventos, no entre ellos, lo que permite que si un servicio falla el resto siga funcionando, aún así, una falla en el enrutador de eventos podría generar una pérdida importante del sistema.	Altamente escalable gracias al desacoplamiento de servicios. Añadir nuevos servicios(máquinas) no afecta al resto gracias al sistema de enrutador de eventos usado.

Con este análisis se procede a realizar una votación en cada arquitectura y por cada atributo de calidad. Se tienen 10 puntos distribuidos de la siguiente forma: 4 para disponibilidad (es decir, lo consideramos como el atributos más importante dados los requisitos no funcionales mencionados anteriormente), 3 para rendimiento y 3 para escalabilidad. A cada uno de estos atributos se les asigna un porcentaje del total de puntos (de 1 a 10, por ejemplo 1 punto equivale al 10 % de puntos de una columna determinada) como se muestra a continuación:

Arquitectura	Rendimiento 3 puntos	Disponibilidad 4 Puntos	Escalabilidad 3 Puntos
Micro-servicios	5	10	10
3 Capas	9	5	3
Orientado a servicios (SOA)	7	7	6
Orientado a eventos	6	8	7

Los resultados obtenidos son:

- Micro-servicios: 8.5
- 3 Capas: 5.6
- SOA: 6.7
- Orientado a eventos: 7.1

Ganando micro servicios, por lo cual el equipo de desarrollo optó por usar esta arquitectura para el proyecto. A continuación se presenta un diagrama detallado de esta arquitectura:

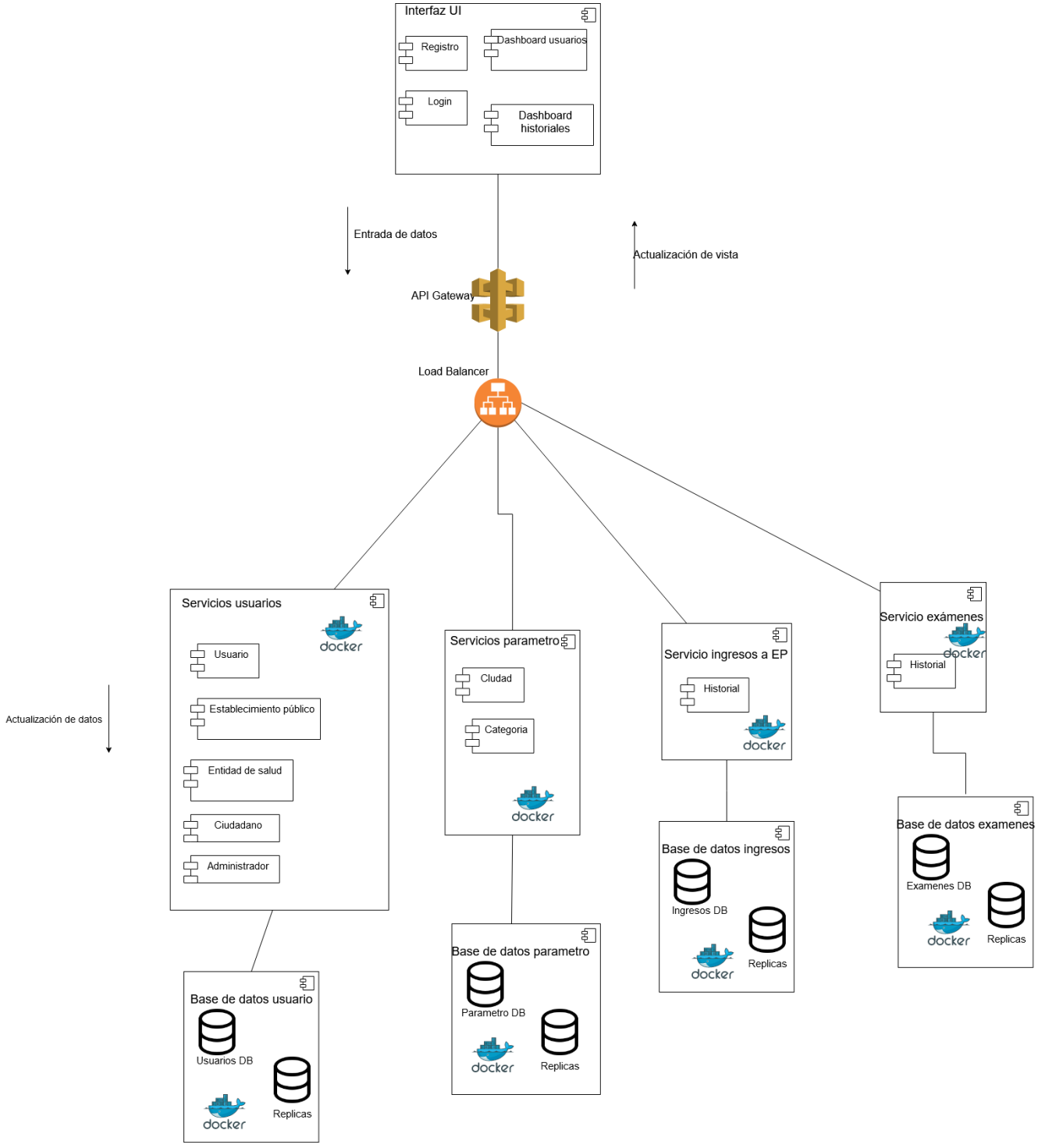


Figura 2.5: Diagrama detallado arquitectura microservicios para el sistema

Como se puede observar se optó por tener 4 servicios distintos para el sistema, el primero de ellos es el de usuarios, el cual se refriere a todo lo relacionado con registro e inicio de sesión en la página web, modificación de datos personales de usuarios, listado de usuarios, activación y desactivación de cuentas. El segundo es para la gestión de parámetros del sistema, que involucran barrios, municipios, departamentos, categorías de establecimientos públicos y período de cuarentena. Luego se tiene el servicio de ingresos que involucra todo el manejo del historial de ingresos de los distintos ciudadanos y el registro de estos para entrar a un establecimiento público. Por último se tiene el servicio de exámenes médicos, el cual se refiere a todo el proceso para registrar un examen médico y para llevar el historial de estos.

3 Objetivos y restricciones de la arquitectura

Con la arquitectura escogida se tiene como objetivo que cumpla con los atributos de calidad mencionados anteriormente, principalmente disponibilidad (esta se da gracias al desacoplamiento de la arquitectura). Como se mostró en el diagrama detallado el equipo opta por realizar replicas de las bases de datos en los servidores por cada uno de los 4 servicios propuestos, es decir, aplicar redundancia por si alguna base de datos se cae, esto asegura una mayor disponibilidad además de la brindada por la granularidad. Por otro lado, el balanceador de cargas permite aumentar el nivel de disponibilidad gracias a que utiliza los micro servicios que se encuentran aptos para generar respuesta a las peticiones.

En el diagrama detallado se puede ver el simbolo de docker, lo que significa que cada servicio va a estar dentro de su propio contenedor. Se hace uso de este para tener un ambiente aislado y hacer un despliegue de forma rápida.

Aunque en rendimiento no tiene la mejor puntuación respecto a las otras arquitecturas satisface lo necesario para cumplir de forma satisfactoria con este atributo de calidad, siendo esta su principal limitación.

La limitación mencionada se debe al nivel de granuralidad del sistema, que causa latencia en las peticiones realizadas, aun así, en los otros dos atributos de calidad supera a las otras arquitecturas, esto gracias a la granuralidad mencionada anteriormente, al tener dividido de esta forma el sistema permite una mayor disponibilidad y escalamiento vertical.

4 Vista lógica

4.1. Base de datos

Para el desarrollo del proyecto se tuvo en cuenta principalmente los siguientes atributos para la elección de una base de datos:

- Escalabilidad
- Disponibilidad
- Rendimiento

Al igual que en el proceso de elección de arquitectura de software se tienen en cuenta estos atributos según los requisitos no funcionales del sistema. Con el fin de hacer una correcta elección se procede a comparar bases de datos relacionales versus no relaciones.

4.1.1. Base de datos relacional

Estas bases de datos proporcionan acceso a puntos de datos que están relacionados entre sí. Se basan en el modelo relacional, una forma sencilla de representar datos en tablas. Cada fila de la tabla es un registro con un ID único llamado llave. Por otro lado, las columnas de la tabla contiene atributos de los datos y, por lo general, cada registro tiene un valor para cada atributo, lo que facilita el establecimiento de las relaciones entre los puntos de datos.

Las razones por las cuales se puede considerar usar una base de datos relacional son:

- Es necesario las propiedades ACID (atomicidad, consistencia, aislamiento, durabilidad). El cumplimiento ACID reduce las anomalías y protege la integridad de su base de datos.
- Los datos están estructurados y no cambian, es decir, no se van agregar variedad de tipos de datos con el tiempo.
- Se tiene una gran cantidad de transacciones y puedan ser complicadas.

4.1.2. Base de datos no relacional

Estas bases de datos no son tabulares y almacenan datos de manera diferente a tablas relacionales. Los tipos de modelo de datos principales son: basada en documentos,

llave valor, grafos, entre otras. Proporcionan esquemas flexibles y escalan fácilmente con grandes cantidades de datos.

Las razones por las cuales se puede considerar usar una base de datos no relacional son:

- Se va a desarrollar una aplicación con Big Data.
- Se va a trabajar con sistemas distribuidos.
- Las aplicaciones necesitan manejar una gran cantidad de datos para crear una experiencia personalizada que sea diferente y cambie con frecuencia para cada usuario.
- Aplicaciones web comerciales con catálogo de productos que cambia con frecuencia.

4.1.3. Comparación y elección de base de datos

A partir de los tipos de base de datos mencionados anteriormente se realiza una tabla para comparar cada una con los respectivos atributos de calidad, teniendo en cuenta que las no relacionales son un conjunto grande se decidió usar llave valor, documentos y grafos:

Base de datos	**Rendimiento**	**Disponibilidad**	**Escalabilidad**
Relacional	Tiene un buen rendimiento cuando se necesitan consultas complejas, depende también de la velocidad del disco.	Puede no estar disponible todo el tiempo debido a que se enfoca como prioridad en la consistencia.	El escalamiento se hace normalmente vertical, ya que horizontal es complicado.
Documentos	Con los almacenes de documentos la información no se distribuye en varias tablas y esto mejora el rendimiento, aun así, para consultas complejas puede no ser tan bueno. (que los datos no estén tan relacionados)	Disponibilidad alta gracias a la redundancia de datos.	Altamente escalable, a través de sharding distribuyendo datos en varias máquinas.
Llave valor	Dado la estructura simple del modelo el rendimiento es alto, los datos se acceden vía una llave específica, operaciones de lectura y escritura se hacen rápidamente.	Su simplicidad en el modelo permite una alta disponibilidad al poder distribuir los datos.	Al igual que en el rendimiento gracias a su simplicidad la escalabilidad es flexible.
Grafos	Tiene alto rendimiento para búsquedas, la complejidad y cantidad de datos no perjudica el proceso de búsqueda. La velocidad de búsqueda depende únicamente del número de relaciones concretas .	Dado su estructura flexible y sus resultados en tiempo real tiene una alta disponibilidad.	Es difícil de escalar, por estar diseñado para arquitecturas con un solo servidor.

Con este análisis se procede a realizar una votación en cada base de datos y por cada atributo de calidad según las funciones de la aplicación que requieren el uso de estas; la primera de ella se le llamo usuarios, se refiere a los roles existentes en el sistema, es decir, guardar la información para registro e inicio de sesión. Con ingresos se refiere a el registro de ingresos de los ciudadanos en los distintos establecimientos públicos, exámenes al registro de exámenes médicos por parte de las entidades de salud y por último ciudad, el cual indica los parámetros del sistema (categorías, tipos de documento, período de cuarentena, barrios).

Los puntos son iguales a la votación mostrada para elección de arquitectura (de 1 a 10 por cada atributo de calidad), de igual manera se distribuyen 10 puntos entre los 3 atributos de calidad y por cada función de la aplicación explicada anteriormente, quedando de la siguiente forma:

Usuarios				
Base de datos	**Rendimiento 4 Puntos**	**Disponibilidad 3 Puntos**	**Escalabilidad 3 Puntos**	**Total**
Relacional	10	5	5	7
Documentos	4	8	8	6.4
Llave valor	5	8	8	6.8
Grafos	7	6	5	6.1
Ingresos				
Base de datos	**Rendimiento 3 Puntos**	**Disponibilidad 4 Puntos**	**Escalabilidad 3 Puntos**	**Total**
Relacional	10	5	5	6.5
Documentos	4	8	8	6.8
Llave valor	5	8	8	7.1
Grafos	7	6	5	6
Exámenes				
Base de datos	**Rendimiento 4 Puntos**	**Disponibilidad 2 Puntos**	**Escalabilidad 4 Puntos**	**Total**
Relacional	10	5	5	7
Documentos	4	8	8	6.4
Llave valor	5	8	8	6.8
Grafos	7	6	5	6
Ciudad (parámetros del sistema)				
Base de datos	**Rendimiento 4 Puntos**	**Disponibilidad 5 Puntos**	**Escalabilidad 1 Punto**	**Total**
Relacional	10	5	5	5.5
Documentos	4	8	8	7.6
Llave valor	5	8	8	7.7
Grafos	7	6	5	5.7

A partir de esto se aprecia que se usarían dos tipos de base de datos distinta, llave valor y relacional, aún así, dado que hacer este híbrido tiene un costo en rendimiento, además una implementación compleja y de cuidado se opto por usar una base de datos no relacional basada en documentos para todo el sistema, en este caso MongoDB, dado que realizar la implementación híbrida lleva prácticamente a los costos generales de implementar todo en MongoDB. Además, esta base de datos permite operaciones de escritura y lectura de registros rápidas, operaciones principales de la aplicación.

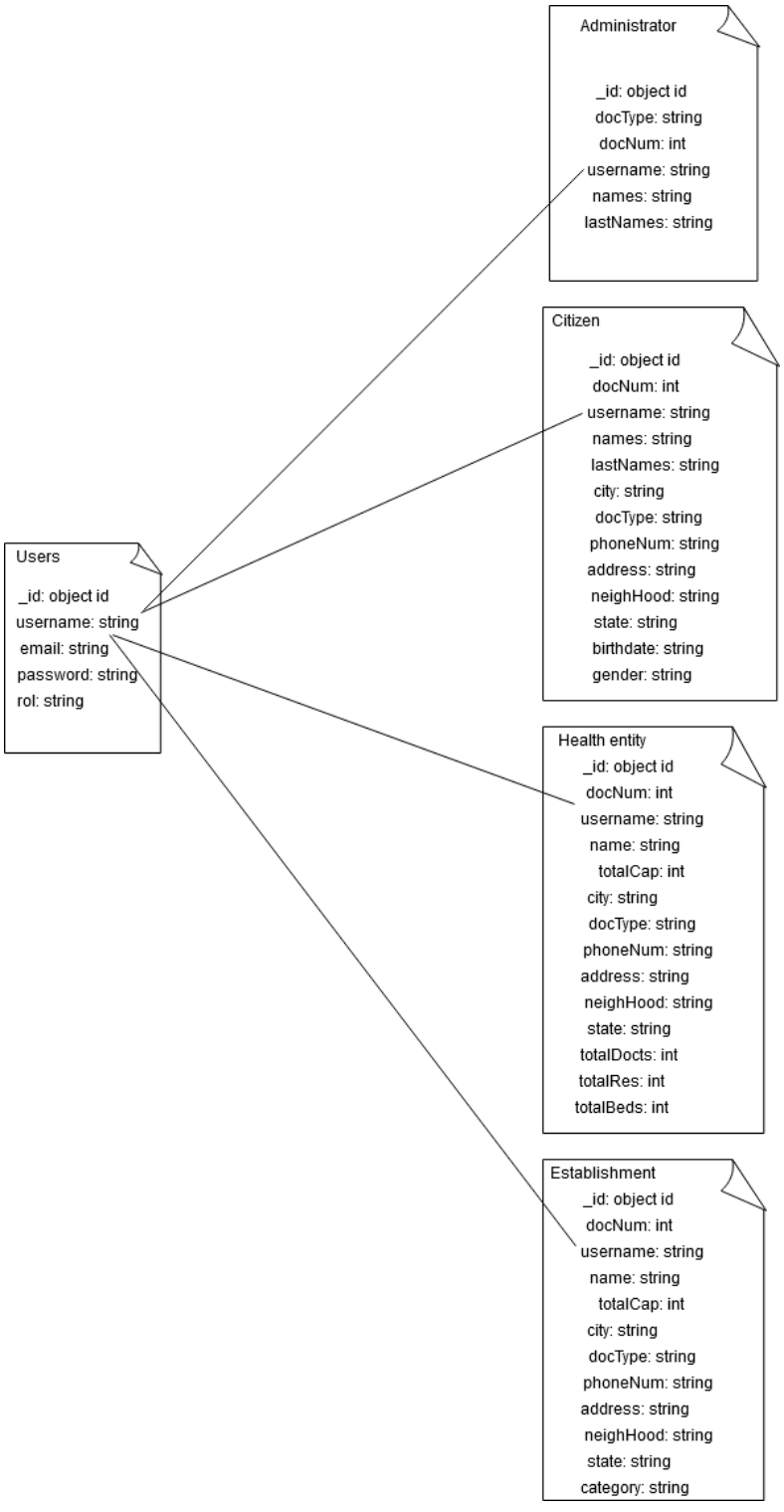
4.1.4. Diseño de bases de datos

Dado que la arquitectura es por microservicios, se implementan cuatro bases de datos de la siguiente forma:

1.UsersDB, la cual sirve para realizar las operaciones de registro e inicio de sesión en la aplicación, cuenta con las siguientes colecciones:

- administrator
- citizen
- establishment
- healthEntity
- user

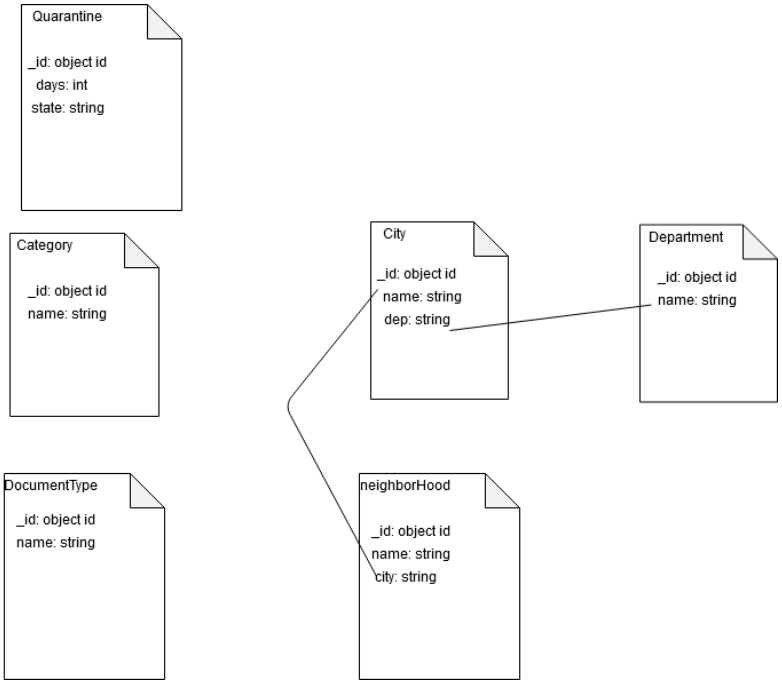
La estructura de los documentos es la siguiente:


Administrator
_id: object id
docType: string
docNum: int
username: string
names: string
lastNames: string
Citizen
_id: object id
docNum: int
username: string
names: string
lastNames: string
city: string
docType: string
phoneNum: string
address: string
neighHood: string
state: string
birthdate: string
gender: string
Users
_id: object id
username: string
email: string
password: string
rol: string
Health entity
_id: object id
docNum: int
username: string
name: string
totalCap: int
city: string
docType: string
phoneNum: string
address: string
neighHood: string
state: string
totalDocts: int
totalRes: int
totalBeds: int
Establishment
_id: object id
docNum: int
username: string
name: string
totalCap: int
city: string
docType: string
phoneNum: string
address: string
neighHood: string
state: string
category: string

2.ParametersDB, la cual sirve para guardar los parámetros requeridos en el sistema, cuenta con las siguientes colecciones:

- category
- city
- department
- documentType
- neighborHood
- quarentine

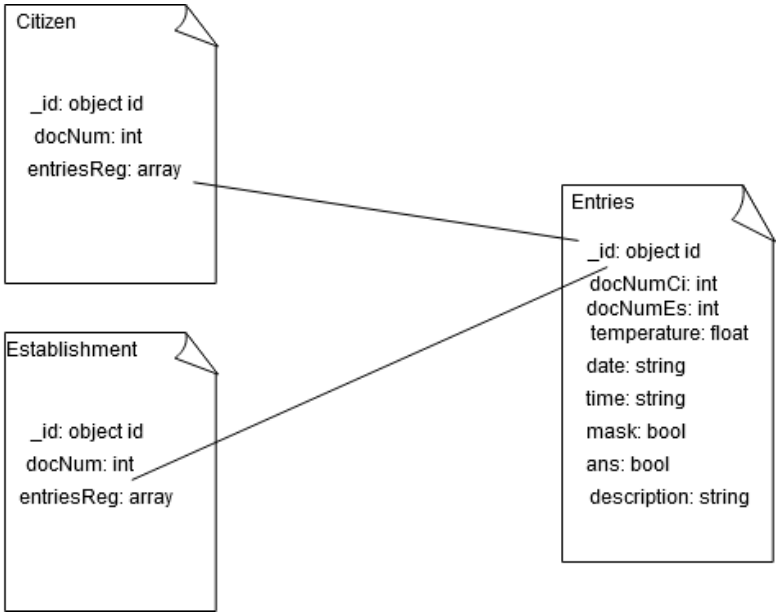
La estructura de los documentos es la siguiente:



3.EntryDB, la cual sirve para guardar los registros de las entradas de los ciudadanos a los establecimientos públicos, cuenta con las siguientes colecciones:

- citizen
- entries
- establishment

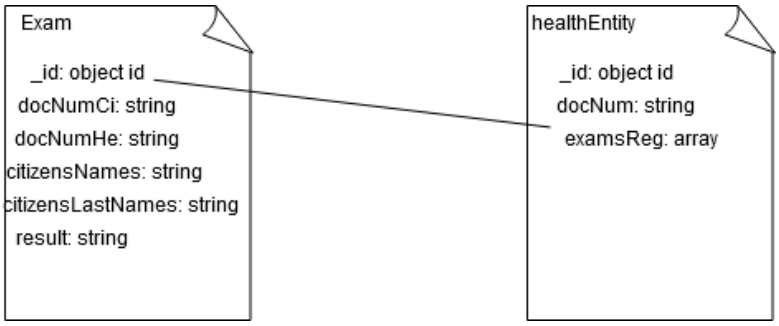
La estructura de los documentos es la siguiente:



4.ExamsDB, la cual sirve para guardar los exámenes médicos registrados por las entidades de salud, cuenta con las siguientes colecciones:

- healthEntity
- exam

La estructura de los documentos es la siguiente:



4.2. Diagramas de clases

El diagrama de clases muestra la relación entre las distintas clases requeridas en el sistema por cada microservicio, por esta razón se deciden presentar por separado, es decir, un diagrama de clases por cada microservicio (4 en total). Se encuentra adjunto al paquete de la entrega.

4.3. Diagrama de secuencia

Se realizó un diagrama de secuencia por cada rol, de la siguiente manera:

- Administrador: Editar nombre de categoría.
- Ciudadano: Generar código QR.
- Entidad de salud: Búsqueda examen por fecha.
- Establecimiento público: Búsqueda de visitas por género.

Se encuentra adjunto al paquete de la entrega.

5 Vista de procesos

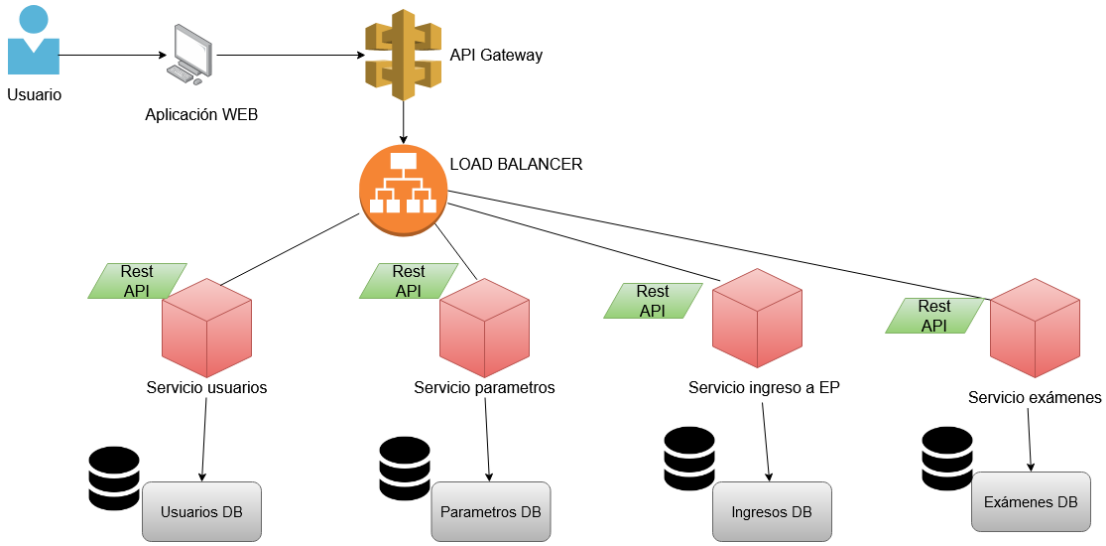
5.1. Diagrama de actividades

Se realizó un diagrama de actividades por cada rol, de la siguiente manera:

- Administrador: Editar nombre de categoría.
- Ciudadano: Generar código QR.
- Entidad de salud: Búsqueda examen por fecha.
- Establecimiento público: Búsqueda de visitas por género.

Se encuentra adjunto al paquete de la entrega.

6 Vista física


Usuario
Aplicación WEB
API Gateway
LOAD BALANCER
Rest API
Servicio usuarios
Rest API
Servicio parametros
Rest API
Servicio ingreso a EP
Rest API
Servicio exámenes
Usuarios DB
Parametros DB
Ingresos DB
Exámenes DB

7 Vista de despliegue

Se encuentra adjunta la imagen con esta vista dentro del paquete de la entrega, al elegir microservicios se procede a colocar servidores cada uno de los 4 mencionados anteriormente, junto a su base de datos.

8 Vista interfaz de usuario

El front end se realiza en Angular, se encuentra adjunto dentro de la carpeta front de la entrega.

9 Anexos

9.1. Documentos externos

- Casos de uso: En la carpeta casos de uso dentro de documentation se encuentra un diagrama por cada rol en el sistema y la respectiva documentación actualizado respecto a la primera entrega.
- Diagrama de contexto: Este archivo es una imagen del diagrama de contexto del sistema corregido respecto a la primera entrega, se encuentra dentro de la carpeta documentation.
- Especificación de requisitos: Esta carpeta contiene dos archivos excel, uno con la especificación de los requisitos y otro para la validación de estos actualizados respecto a la primera entrega.
- Diagrama de paquetes: en la carpeta documentation se encuentra actualizado el diagrama de paquetes respecto a la primera entrega.
- Actas: Esta carpeta contiene las actas que llevan el registro de las reuniones con el cliente.
- Prototipo: Backend mostrando las operaciones de registro, iniciar sesión, listar todas las cuentas, y la opción para activar.
- Vista lógica: El diagrama de clases es una imagen que se encuentra en la carpeta documentation, los diagramas de secuencia se encuentran en la carpeta llamada diagramas de secuencia dentro de documentation, se realizó uno por cada rol.
- Vista de procesos: Los diagramas de actividades se encuentran dentro de la carpeta llamada diagrama de actividades dentro de documentation, se realizó uno por cada rol.
- Vista de despliegue: el diagrama de despliegue se encuentra dentro de la carpeta documentation, es una imagen.
- Diagrama de componentes: se encuentra dentro de la carpeta documentation, en una imagen.
- Interfaces de usuario: todo el front-end de la aplicación se encuentra dentro de la carpeta front realizado en angular.
- Diagrama de clases: se encuentra dentro de la carpeta documentation, en una imagen.