Trabajo Fin de Máster:

**Implementación de una API para alimentar una aplicación web y/o móvil que permita la gestión de solicitudes de personal de la salud a domicilio**

|  |  |  |
| --- | --- | --- |
| Titulación:  Máster Universitario en Desarrollo de Aplicaciones y Servicios Web  Curso académico  2022 – 2023 | Alumno/a: Ortiz Vera Paúl Mauricio  D.N.I: 0804362903  Director/a de TFM: Bueso Martínez Víctor | Convocatoria:  Primera |

**Resumen(abstract)**

**Índice**

[1. Introducción y objetivos 4](#_Toc147444820)

[2. Marco teórico y tecnológico 5](#_Toc147444821)

[Sistemas de solicitud de personal de salud 5](#_Toc147444822)

[Páginas web 5](#_Toc147444823)

[Aplicaciones móviles 6](#_Toc147444824)

[Base de Datos 7](#_Toc147444825)

[Diseño conceptual 9](#_Toc147444826)

[Diseño lógico 11](#_Toc147444827)

[Diseño físico 15](#_Toc147444828)

[Procedimientos almacenados (Store procedure – sp) 19](#_Toc147444829)

[API 19](#_Toc147444830)

[Selección del lenguaje de programación 20](#_Toc147444831)

[Selección del servidor de despliegue 21](#_Toc147444832)

[3. Metodología 22](#_Toc147444833)

[Requisitos de la aplicación 24](#_Toc147444834)

[Casos de Uso 25](#_Toc147444835)

[4. Resultados y discusión 26](#_Toc147444836)

[5. Conclusiones y trabajo futuro 26](#_Toc147444837)

[6. Bibliografía 26](#_Toc147444838)

[7. Anexos. 26](#_Toc147444839)

1. Introducción y objetivos

Ecuador se ha convertido en el país menos seguro de Latinoamérica, en consecuencia, la mayoría de los ecuatorianos se sientan inseguros al caminar por la noche, esto se debe al clima de violencia generado por las pandillas y el crimen organizado.

En encuestas realizadas en el año 2022 los ecuatorianos pierden la confianza en el sistema de justicia y en la policía, adicional el 64% de los ecuatorianos se sienten inseguros frente al 35% que no ven el problema de inseguridad. (GALLUP, 2023)

Debido a la gran inseguridad que sufre el país (Ecuador) las personas prefieren quedarse en casa salvaguardando su seguridad, por lo que el tener una aplicación que permita a los usuarios recibir cuidados médicos ya sean de doctores, enfermeros, psicólogos, etc. desde la seguridad de su hogar, proporcionaría un gran aporte a la sociedad.

El tener la base de los APIs desarrollados permitiría la posterior creación de la aplicación, esto generará que independientemente de su arquitectura pueda conectarse a los servicios y consumirlos correctamente.

**Objetivo general:**

Desarrollar una API que permita a alimentar una aplicación web y/o móvil de solicitud de profesionales de la salud para proporcionar toda la información que esta requiera para funcionar correctamente.

**Objetivos específicos:**

* Diseñar una base de datos que proporcione escalabilidad y buen rendimiento a la aplicación.
* Implementar MVC para facilitar el mantenimiento y la añadidura de nuevas funcionalidades a los servicios
* Proporcionar servicios Rest que permitan el correcto funcionamiento de la aplicación.

1. Marco teórico y tecnológico

Es importante realizar una investigación de si existen sistemas que permitan la gestión de solicitudes de personal de la salud ya sean a domicilio o no, tener el conocimiento de estos sistemas se podrán verificar las funcionalidades principales que estas disponen para determinar el mejor camino a seguir para la aplicación.

Sistemas de solicitud de personal de salud

La mayoría de las formas de solicitar personal de la salud a domicilio son a través de páginas web con sistemas de formularios, más no un sistema completo que permita gestionar estas solicitudes. A continuación, se describen algunos de estos:

**Páginas web**

* **DevitaMedical:**

Es una página que ofrece servicios médicos a pacientes con síndromes dolorosos tanto crónicos como agudos, teniendo atención 24/7, entre sus servicios médicos tenemos medicina general, pediatría, traumatología, ginecología, gastroenterología, cardiología y urología. El servicio de agendamiento de citas a domicilio no se realiza a través de ningún formulario, se lo hace directamente vía WhatsApp. (Devita Medical, 2023)

* **Metrored**

Es una red de servicios ambulatorios que opera en Ecuador, trabaja principalmente con redes de seguros, y ofrece la afiliación a estos en caso de no disponer uno, es posible acceder a sus servicios sin necesidad de tener un seguro activo. Dispone una gran cantidad de servicios médicos, entre ellos, medicina general, medicina interna, ginecología, endocrinología etc.

A pesar de la gran cantidad de servicios médicos que dispone sólo tienen los siguientes disponibles para domicilio: Médico, laboratorio, pruebas PCR, prueba anti spike, procedimientos (electrocardiograma, audiometría, espirometría, etc.), fisioterapia y enfermeros. Para poder acceder a los servicios a domicilio se lo debe hacer a través de un formulario. (Metrored, 2023)

* **Veris**

Se encargan de tratar lesiones o enfermedades de atención inmediata, teniendo en cuenta que estas no conlleven riesgo vital o secuelas funcionales graves en los pacientes. No se puede solicitar el servicio a domicilio si los requerimientos del cliente no son lesiones o enfermedades. Para acceder al servicio a domicilio sólo se lo puede hacer a través de llamada telefónica. (Veris, 2022)

**Aplicaciones móviles**

* **Doctoralia:**

Es una plataforma que sólo funciona en España, esta permite pedir citas online y presenciales con especialistas médicos. Cuenta con más de 60.000 especialistas, y se pueden vincular seguros médicos para agilitar dichos trámites. Entre los servicios disponibles no existe uno que tenga que ver con la asistencia de personal de salud a domicilio. Dispone de una página web y una aplicación móvil para acceder a los servicios. (Doctoralia, 2023)

* **Doctor On Demand**

Es un proveedor virtual de cuidados médicos tanto para el cuerpo como para la mente, incluyendo cuidados urgentes, cuidados mentales, entre otros, opera principalmente en el distritito de Columbia. Esta plataforma cuenta con una página web y aplicaciones móviles tanto en Android como en iOS. Se pueden reservar citas a través de una de estas, pero es preciso acercarse al lugar de la cita, y no dispone de servicio a domicilio. (Doctor On Demand, 2023)

* **knok**

Es una aplicación móvil que ofrece atención médica tanto a través de video llamadas como de visitas a domicilio, puede ser usada por cualquier tipo de paciente, pero es especialmente útil para niños, ancianos y enfermos crónicos. Esta aplicación permite visualizar el CV del médico y las labores previas que ha tenido. El servicio a domicilio sólo se encuentra disponible en Madrid. (knok, 2023)

Existen varios sistemas de atención médica en el mercado, iniciando desde una página web que ofrece el agendamiento vía llamadas telefónicas hasta aplicaciones complejas, entre estas es preciso resaltar que las aplicaciones que permiten la administración completa de estas solicitudes sólo están disponibles en ciertas regiones.

Base de Datos

Para continuar con la investigación es necesario seleccionar un buen sistema gestor de bases de datos (SGBD) que cumpla con las necesidades para el tipo de sistema que estamos implementando, debido al tipo de sistema es necesario una base de datos relacional, a continuación, se describen dos de los SGBD más utilizados en la actualidad.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parámetros | MySQL | | | PostgreSQL | SQL Server |
| Arquitectura | | Un solo proceso y relacional | Multiproceso y objeto-relacional | | Un solo proceso |
| Tipos de datos | | char, varchar, binary, varbinary, tinyblob, tinytext, text, blob, mediumtext, mediumblob, longtext, longblob, enum, set, bit, tinyint, bool, boolean, smallint, mediumint, int, integer, bigint, float, float, double, double precision, decimal, dec, date, datetime, timestamp, time, year | bigint, bigserial, bit, bit varying, boolean, box, bytea, character, character varying, cidr, circle, date, double precision, inet, integer, interval, json, jsonb, line, lseg, macaddr, macaddr8, money, numeric, path, pg\_lsn, pg\_snapshot, point, polygon, real, smallint, smallserial, serial, text, time, timestamp, tsquery, tsvector, txid\_snapshot, uuid, xml | | bigint, numeric, bit, smallint, decimal, smallmoney, int, tinyint, money, float, real, date, datetimeoffset, datetime2, smalldatetime, datetime, time, char, varchar, text, nchar, nvarchar, ntext, binary, varbinary, image, cursor, rowversion, hierarchyid, uniqueidentifier, sql\_variant, xml, Spatial Geometry Types, Spatial Geography Types, table |
| Indexes | | Primarily B-tree, hash, R-tree e index invertidos | B-tree, GiST, hash, BRIN y SP-GiST | | Hash, memory-optimized Nonclustered, Clustered, Nonclustered, Unique, Columnstore, Index with included columns, Index on computed columns, Filtered, Spatial, XML, Full-text |
| Rendimiento | | Capacidad para alto volumen de lecturas | Capacidad para alto volumen de lecturas y escrituras | | Capacidad para alto volumen de lecturas y escrituras |
| Seguridad | | Control de acceso y conexión encriptadas | Control de acceso y múltiples conexiones encriptadas | | Control de autorización y autenticación, múltiples conexiones cifradas |
| Soporte | | Soporte de la comunidad y por contratos de proveedores | Soporte de la comunidad en la versión por defecto y soporte por compañías que lanzaron versiones personalizadas | | Soporte ofrecido directamente por Microsoft y la comunidad |

En base a las diferencias antes descritas se decide utilizar PostgreSQL como el SGBD de la aplicación puesto que cuenta con ventajas como una gran capacidad de lectura y escritura que permitirá manejar un gran volumen de datos en la aplicación.

**Diseño conceptual**

Para el buen análisis de la base de datos se debe realizar un diseño conceptual precioso que cubra todas las necesidades que tendrá el sistema, si se detecta algún cambio en la base de datos posterior a este punto sería necesario volver y corregir el inconveniente, por ello este diseño se ha realizado con rigurosidad.

* **Identificar entidades**

**USUARIO:** persona que va a hacer Login en la aplicación.

**CLIENTE:** persona que crea la solicitud.

**PROFESIONAL:** persona (profesional de la salud) que acepta la solicitud.

**SOLICITUD:** requerimiento que levanta el cliente para ser aceptado por el profesional.

* **Identificar las relaciones**

**SER:** Relación entre USUARIO y CLIENTE; Un USUARIO puede o no ser un cliente. Un CLIENTE es uno y solo un USUARIO

**REGISTRA:** Relación entre USUARIO y PROFESIONAL; Un USUARIO puede o no ser un PROFESIONAL. Un PROFESIONAL es uno y solo un USUARIO.

**SOLICITA:** Relación entre CLIENTE y SOLICITUD; Un CLIENTE puede solicitar ninguna o muchas SOLICITUDES. Una SOLICITUD se puede solicitar por una y solo una persona.

**ACEPTA:** Relación entre PROFESIONAL y SOLICITUD; Un PROFESIONAL acepta ninguno o muchas SOLICITUDES. Una SOLICITUD puede ser aceptada por ninguno o un PROFESIONAL.

* **Identificar los atributos y asociarlos a entidades y relaciones**

**USUARIO:** login, contraseña, nombre, cedula, pais, sexo, ciudad, direccion, telefono, correo, fechaNacimiento, fechaModificacion, fechaRegistro, rol, estado.

**CLIENTE:** id, enfermedad, documentacion, estado.

**PROFESIONAL:** codigo, tipo, fechaModificacion, documentación, estado.

**SOLICITUD:** numero, direccion, latitude, longitude, tipo, comentario, fechaRegistro, valorPropuesto, valorAceptado, estado.

* **Determinar los dominios de los tributos.**

Dominio del atributo estado de la entidad USUARIO: [A, P, E]

Dominio del atributo sexo de la entidad USUARIO: [M, F]

Dominio del atributo tipo de la entidad CLIENTE: [A, P, E]

Dominio del atributo documentacion de la entidad CLIENTE: [A, P, D]

Dominio del atributo estado de la entidad PROFESIONAL: [A, P, S, E]

Dominio del atributo documentacion de la entidad PROFESIONAL: [A, P, D]

Dominio del atributo estado de la entidad SOLICITUD: [A, P, E]

* **Determinar los identificadores**

**USUARIO:** login

**CLIENTE:** id

**PROFESIONAL:** codigo

**SOLICITUD:** codigo

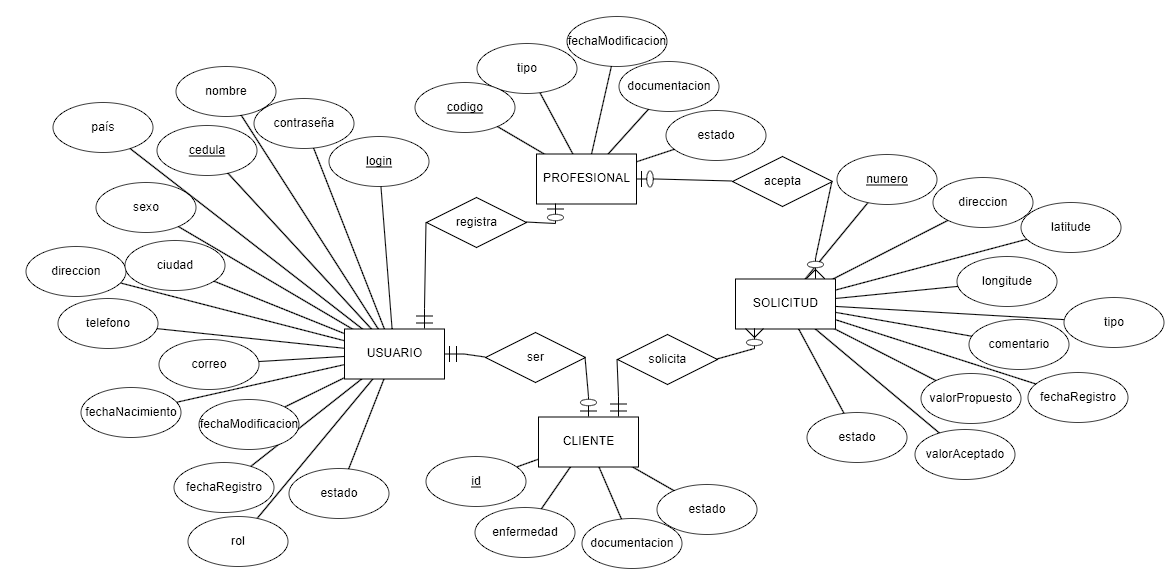


Ilustración diseño conceptual

**Diseño lógico**

Posterior al diseño conceptual se requiere generar el diseño lógico transformando las entidades en tablas, esto permitirá una visión un poco más técnica de la base de datos.

* **Transformar entidades en tablas.**

**USUARIO:** (login, contrasena, nombre, cedula, pais, sexo, ciudad, direccion, telefono, correo, fechaNacimiento, fechaModificacion, fechaRegistro, rol, estado) PK:{login}

**CLIENTE:** (id, enfermedad, documentacion, estado) PK:{id}

**PROFESIONAL:** (codigo, tipo, fechaModificacion, documentación, estado) PK:{codigo}

**SOLICITUD:** (numero, dirección, latitude, longitude, tipo, comentario, fechaRegistro, valorPropuesto, valorAceptado, estado) PK:{numero}

* **Transformar relaciones en tablas y aportar la integridad referencial (las reglas de comportamiento de las claves ajenas)**

**Relaciones m:n**

En el modelo de este sistema no existen relaciones de muchos a muchos, por ello no existe ninguna transformación en este apartado.

**Relaciones 1:1 y 1:N**

**USUARIO:** (login, contrasena, nombre, cedula, pais, sexo, ciudad, direccion, telefono, correo, fechaNacimiento, fechaModificacion, fechaRegistro, rol, estado)

* + PK: {login}

**CLIENTE:** (id, enfermedad, documentacion, estado, login)

* + PK: {id}
  + FK: {CLIENTE.login es clave ajena a USUARIO.login}

**PROFESIONAL:** (codigo, tipo, fechaModificacion, documentación, estado, login)

* + PK: {codigo}
  + FK: {PROFESIONAL.login es clave ajena a USUARIO.login}

**SOLICITUD:** (numero, dirección, latitude, longitude, tipo, comentario, fechaRegistro, valorPropuesto, valorAceptado, estado, id, codigo)

* + PK: {numero}
  + FK: {SOLICITUD.id es clave ajena a CLIENTE.id}
  + FK: {SOLICITUD.codigo es clave ajena a PROFESIONAL.codigo}

**Reglas de comportamiento de las claves ajenas**

**CLIENTE:** (id, enfermedad, documentacion, estado, login)

* PK: {id}
* Dominio de estado: [A, P, E]
* Dominio de documentacion: [A, P, D]
* FK: {CLIENTE.login es clave ajena a USUARIO.login}
  + ¿Acepta nulos?: NO
  + Actualizar: PROPAGAR
  + Borrar: RESTRINGIR

**PROFESIONAL:** (codigo, tipo, fechaModificacion, documentación, estado, login)

* PK: {codigo}
* Dominio de estado: [A, P, S, E]
* Dominio de documentacion: [A, P, D]
* FK: {PROFESIONAL.login es clave ajena a USUARIO.login}
  + ¿Acepta nulos?: NO
  + Actualizar: PROPAGAR
  + Borrar: RESTRINGIR

**SOLICITUD:** (numero, dirección, latitude, longitude, tipo, comentario, fechaRegistro, valorPropuesto, valorAceptado, estado, id, codigo)

* PK: {numero}
* Dominio de estado: [A, P, E]
* FK: {SOLICITUD.id es clave ajena a CLIENTE.id}
  + ¿Acepta nulos?: NO
  + Actualizar: PROPAGAR
  + Borrar: RESTRINGIR
* FK: {SOLICITUD.codigo es clave ajena a PROFESIONAL.codigo}
  + ¿Acepta nulos?: SI
  + Actualizar: PROPAGAR
  + Borrar: RESTRINGIR
* **Normalización(1FN,2FN,3FN).**

**Primera forma normal (1FN):**

No existe relación entre los atributos de la misma tabla por lo que se define que el diseño se encuentra en la 1FN.

**Segunda forma normal (2FN):**

Una vez realizada la comprobación se identifica que el diseño se encuentra en la 2FN puesto que ninguno de los atributos no-principales son funcionalmente dependientes de una parte o subconjunto propio de una clave candidata.

**Tercera forma normal (3FN):**

Al realizada la comprobación se determina que se encuentra en la 3FN puesto que los atributos no-principales no dependen transitivamente de la clave primaria (PK), y además posteriormente se verificó que no se encuentra en la 2FN y en la 1FN

* **Desnormalización**

Debido a que el diseño desde un inicio cumplía con las 3 formas normales no es necesario realizar la desnormalización.

* **Esquema lógico**

|  |
| --- |
| **USUARIO:** (login, contrasena, nombre, cedula, pais, sexo, ciudad, direccion, telefono, correo, fechaNacimiento, fechaModificacion, fechaRegistro, rol, estado)  PK: {login}  Dominio de estado: [A, P, E]  Dominio de sexo: [M, F]  Dominio de rol: [C, R, A]  **CLIENTE:** (id, enfermedad, documentacion, estado, login)  PK: {id}  Dominio de estado: [A, P, E]  Dominio de documentacion: [A, P, D]  FK: {CLIENTE.login es clave ajena a USUARIO.login}  ¿Acepta nulos?: NO  Actualizar: PROPAGAR  Borrar: RESTRINGIR  **PROFESIONAL:** (codigo, tipo, fechaModificacion, documentación, estado, login)  PK: {id}  Dominio de estado: [A, P, S, E]  Dominio de documentacion: [A, P, D]  FK: {PROFESIONAL.login es clave ajena a USUARIO.login}  ¿Acepta nulos?: NO  Actualizar: PROPAGAR  Borrar: RESTRINGIR  **SOLICITUD:** (numero, dirección, latitude, longitude, tipo, comentario, fechaRegistro, valorPropuesto, valorAceptado, estado, id, codigo)  PK: {id}  Dominio de tipo: [A, P, E]  FK: {SOLICITUD.id es clave ajena a CLIENTE.id}  ¿Acepta nulos?: NO  Actualizar: PROPAGAR  Borrar: RESTRINGIR  FK: {SOLICITUD.codigo es clave ajena a PROFESIONAL.codigo}  ¿Acepta nulos?: SI  Actualizar: PROPAGAR  Borrar: RESTRINGIR |

**Diseño físico**

Una vez terminado el diseño lógico se realiza la traducción de este al diseño físico, esto se lo realiza utilizando las sentencias necesarias para la base de datos escogida, que en el caso del sistema actual es PostgreSQL, teniendo en cuenta la traducción sería la siguiente.

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Aplicacion: TFM - Sistema médico \*/  /\* Proposito : Creacion de base de datos \*/  /\* Fecha : 15/09/2023 \*/  /\* Autor : Paúl Ortiz Vera \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  use fastmedical  go  create table usuario(  login varchar(50) primary key,  contrasena varchar(1000) null,  nombre varchar(100) not null,  sexo char(1) not null DEFAULT 'M',  pais varchar(50) not null,  ciudad varchar(100) not null,  direccion varchar(200) not null,  telefono varchar(13) not null,  correo varchar(50) not null,  fecha\_nacimiento date not null,  fecha\_registro date not null,  fecha\_modificacion date NULL,  rol char(1) not null DEFAULT 'C',  estado char(1) not null DEFAULT 'P'  );  create table cliente(  id int identity primary key,  enfermedad varchar(150) not null,  documentacion char(1) not null DEFAULT 'P',  comentario varchar(300) null, --agregado  fecha\_modificacion date null, --agregado  estado char(1) not null DEFAULT 'P',  login varchar(50) not null,  constraint fk\_cliente\_login foreign key (login) references usuario(login) on update cascade  );  create table profesional(  codigo int identity primary key,  tipo char(1) not null,  especialidad char(1) null, --agregado  documentacion char(1) not null DEFAULT 'P',  comentario varchar(300) null, --agregado  fecha\_modificacion date null,  estado char(1) not null DEFAULT 'P',  login varchar(50) not null,  constraint fk\_profesional\_login foreign key (login) references usuario(login) on update cascade  );  create table solicitud(  numero int identity primary key,  direccion varchar(200) not null,  latitude varchar(30) not null,  longitude varchar(30) not null,  tipo varchar(150) not null,  valor\_propuesto money not null,  valor\_aceptado money null,  descripcion varchar(300) not null,  fecha\_registro date not null,  fecha\_modificacion date null,  comentario varchar(300) null,  estado char(1) not null DEFAULT 'P',  id int not null,  codigo int NULL,  constraint fk\_solicitud\_cliente foreign key (id) references cliente(id) on update cascade,  constraint fk\_solicitud\_profesional foreign key (codigo) references profesional(codigo) on update no action  );  create table errores(  numero int primary key,  mensaje varchar(300) not null,  severidad int not null  ); |

En el modelado previo no existe la tabla errores, puesto que esta no forma parte directa de las relaciones de las tablas, pero se la decidió añadir como buena practica de programación debido a que permite un manejo sencillo y fácil manteamiento a los errores de la aplicación.

**Scripts**

Además del script de creación de la base de datos fue necesaria la creación de script que permitieran poblar las tablas con datos parametrizados.

Debido a que la aplicación necesita un manejo de roles se creó un script (roles.sql) que permitiera la creación del usuario administrador, este usuario es el único que puede asignar roles a otros usuarios existentes.

Se decidió no permitir que este usuario administrador pueda asignar a otros usuarios el rol de administrador, de forma que si desea crear otro usuario con este rol se lo deberá realizar por base de datos, esto permite aumentar la seguridad.

El script de errores **(errores.sql)** se encarga de insertar la parametrización de los errores, de forma pueden ser modificados según lo requiera el sistema, así como agregar nuevos.

**Procedimientos almacenados (Store procedure – SP)**

Para aumentar la seguridad y restringir el acceso al a información del a base de datos se optó por la creación de procedimientos almacenados, estos se dividen por la funcionalidad requerida, teniendo dos SP para cada tabla (usuario, profesional, cliente y solicitud), uno para la consulta y otro que se encargará del mantenimiento (creación, actualización y eliminación) de dichas tablas. Adicional un SP que se encargará de todo lo referente al Login de la aplicación.

Cada uno de los SP cuentan con una estructura que provee seguridad y fácil mantenimiento. A cada SP se le agregaron los parámetros @i\_operación y @i\_modo permitiendo dividir cada una de las ejecuciones. Estos también cuentan con una estructura para contener la información de este.

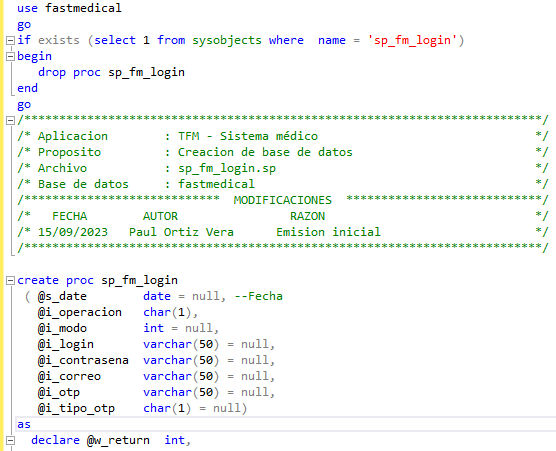


Ilustración Estructura de SPs

Estos también cuentan con un control de errores estándar, permitiendo acceder a la tabla errores para extraer la información del error deseado, esto se logró a través de la creación del sp\_fm\_error, cabe resaltar que esta estructura es similar para los demás sp.

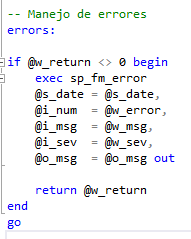


Ilustración Manejo de errores

Los procedimientos almacenados poseen instrucciones que facilitarán la extracción y propagación de la información hacia el API Rest, adicional estas instrucciones facilitarán que la aplicación web o aplicación móvil agregue lógica de negocio en una capa intermedia. Por lo que se debe tener en cuenta que cada uno de estos se comportará como el usuario lo requiera. A continuación, se describen las funcionalidades de cada uno de estos:

* **sp\_fm\_login:** Permite la autenticación de la aplicación.
* **sp\_fm\_error:** Se encarga de la propagación de los errores.
* **sp\_fm\_cons\_usuario:** Devuelve las consultas de la tabla usuario
* **sp\_fm\_adm\_usuario:** Permite realizar operaciones sobre la tabla usuario, así como también el cambio de estado y la aprobación de la documentación.
* **sp\_fm\_cons\_profesional:** Devuelve las consultas de la tabla profesional.
* **sp\_fm\_adm\_profesional:** Provee las operaciones para la administración de la tabla profesional, el cambio de estado y la aprobación de la documentación.
* **sp\_fm\_cons\_cliente:** Permite consultar los datos del a tabla cliente.
* **sp\_fm\_adm\_cliente:** Permite realizar las operaciones, cambio de estado y aprobación de la documentación correspondiente a la tabla cliente.
* **sp\_fm\_cons\_solicitud:** Entrega la información de la tabla solicitud.
* **sp\_fm\_adm\_solicitud:** Permite la administración de las solicitudes, incluyendo aceptar y completar las mismas.

API

Para el desarrollo del api se necesario investigar previamente qué lenguaje de programación es el más indicado para los requerimientos de los usuarios, posterior a ello también es necesario seleccionar un servidor de despliegue que se acople a dichos requerimientos.

**Selección del lenguaje de programación**

Existen muchos lenguajes de programación en la actualidad, por lo que es necesario investigarlos a fondo, y realizar comparativas de estos antes de seleccionar uno, teniendo en cuenta que dicha selección se debe basar en las necesidades de nuestros clientes.

Entre los lenguajes de programación más utilizados tenemos los siguientes, de los cuales se describen sus características más resaltables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Parámetros | Java | | | NodeJS | | PHP |
| Tipo de lenguaje | | Compilado | Scripting | | Scripting | |
| Entorno de ejecución | | Se puede ejecutar tanto del lado del cliente como del servidor, siendo este último el más utilizado | Se ejecuta del lado del servidor | | Se ejecuta del lado del servidor | |
| Tipo de ejecución | | Manejan Non-blocking (no bloqueante). Cuentan con ejecución multihilo y entornos reactivos de forma rápida | | | No cuenta con Non-blocking. La ejecución es sobre procesos y no por hilos (sobrecarga la memoria) | |
| Escalabilidad | | El mantenimiento es fácil conforme crece la aplicación | Aumenta el consumo de recursos conforme crece la aplicación | | Fácil mantenimiento y soporte a pesar del crecimiento de la aplicación | |
| Velocidad | | Excelente manejo de la concurrencia a mayor cantidad de solicitudes | Rápido en aplicaciones pequeñas | | Balanceado | |
| Seguridad | | Todo su entorno es seguro, tanto bibliotecas como frameworks. | Cuenta con un entorno seguro | | Las versiones anteriores son más seguras | |

Luego de realizar la comparativa de los lenguajes de programación, y en base a los requerimientos del sistema se selecciona como lenguaje para el desarrollo de la aplicación Java, puesto que una aplicación de este tipo podría llegar a tener un gran volumen de solicitudes, teniendo en cuenta que tanto clientes como profesionales de la salud estarían ingresando constantemente.

Adicional se selecciona este lenguaje puesto que posee una compatibilidad con grandes cantidades de bases de datos, soporte en línea y entorno que provee seguridad tanto en sus librerías como en los distintos frameworks.

Existen muchos frameworks de java en el mercado, pero para el desarrollo de esta aplicación se selección Spring Boot puesto que permite crear aplicaciones con gran facilidad, de fácil escalabilidad y mantenimiento, sin dejar de lado que tiene un gran soporte por la comunidad.

**Selección del servidor de despliegue**

Una vez que la aplicación sea culminada es necesario que esta sea accesible, ya sea desplegándola en un servidor privado, en un servidor que se encuentre en la nube o una plataforma como servicio.

En el caso en particular es mucho más conveniente desplegar el API en una plataforma como servicio, puesto que ahorra todo el tema de manejo de servidores y configuraciones, por ello se selecciona Amason Web Services (AWS) la cual es una plataforma en la nube que cuenta con más de 200 servicios que permiten a los desarrolladores y empresas gestionar, desarrollar, entregar y monitorear aplicaciones. (AWS, 2023)

**Creación del proyecto**

Para la creación del proyecto se utilizó Sprint Initializr, una herramienta oficial de Sprint que permite crear la aplicación base directamente desde su página web.

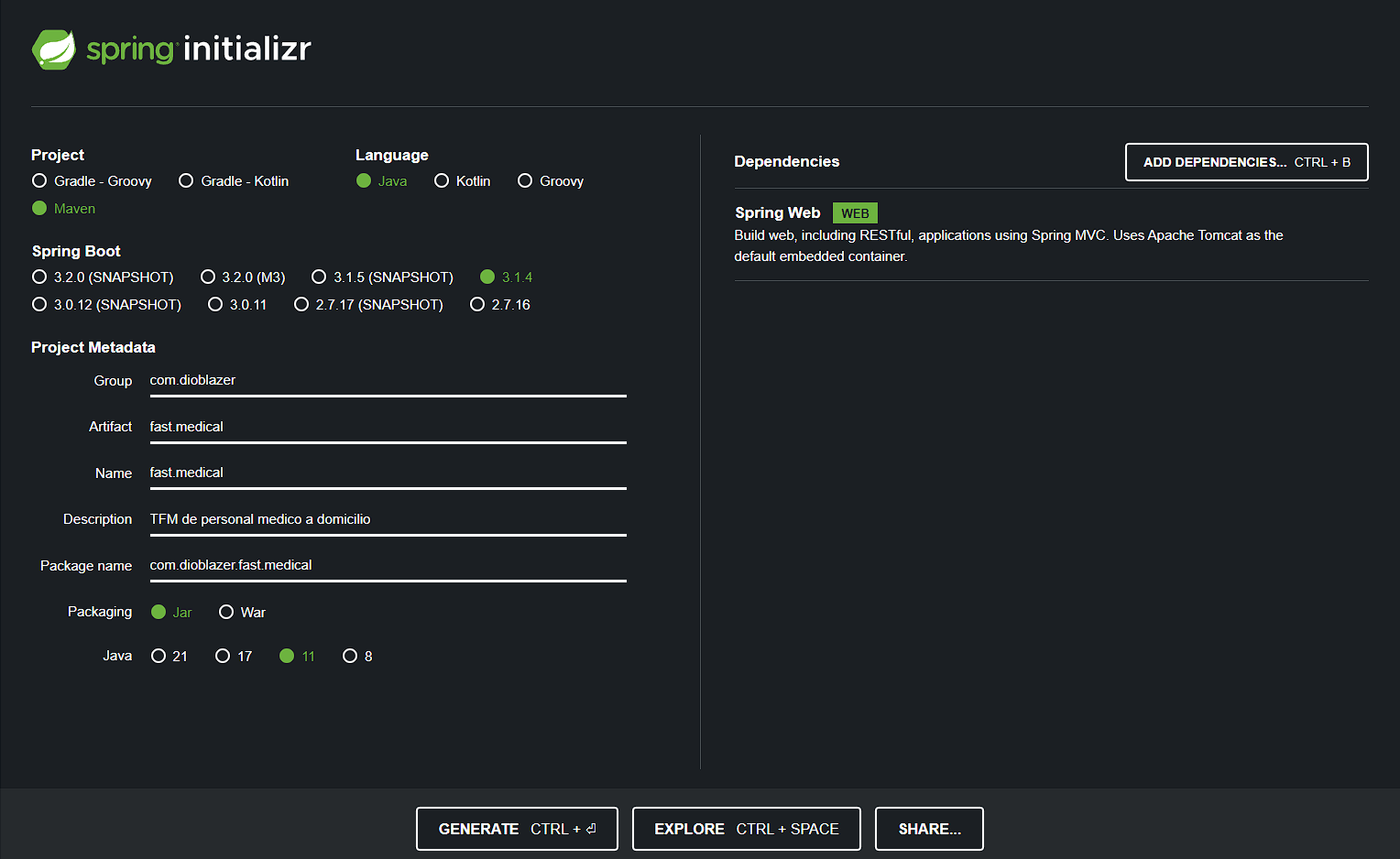


Ilustración Spring Initializr

Esta permite agregar las dependencias requeridas para el proyecto directamente, para este se agregaron las siguientes:

* Spring web: permite crear aplicaciones web basadas en Spring MVC.
* Lombok: permite reducir código repetitivo con la ayuda de anotaciones.
* JDBC Api: permite la gestión de la conexión con la base de datos.
* Spring Boot DevTools: provee un reinicio rápido de las aplicaciones web.
* MS SQL Server Driver: controlador para la base de datos SQL Server.
* Spring Security: provee el marco para la autenticación de las aplicaciones web.

Para el correcto manejo del código se generó una estructura de código, misma que permite dividir los repositorios (gestionan el acceso a la base de datos), servicios (Conectan los endpoints con los repositorios), controladores (exponen los servicios), modelos (representan los campos de las tablas), utilidades (contiene las constantes y componentes que se usan todo el proyecto) y seguridades (contiene toda la configuración de acceso al sistema).

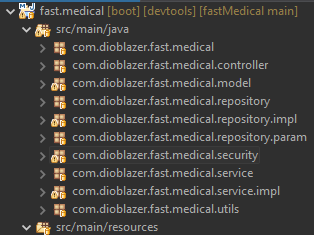


Ilustración Estructura del proyecto

**Seguridad**

Es necesario proveer a la aplicación de seguridad que permita la autenticación de los usuarios que pueden o no tener acceso a la misma. Basándonos en la complejidad del sistema y en la apertura que esta tiene para ser consumida por una aplicación web o móvil, se procede a utilizar Json Web Token (JWT) como método de autenticación.

JWT es un estándar basado en JSON de código abierto, que permite la autenticación de los clientes basándose en un token, esta es programada por un período de tiempo elegido por el desarrollador del sistema conteniendo la identidad y los privilegios del cliente. El token es firmado por una clave almacenada en el servidor, permitiendo que ambas partes puedan verificar si el token es válido y legítimo. (Boldú, 2019)

Para el proyecto se definieron dos filtros que gestionan el acceso a la aplicación:

* Uno para la autenticación, el cual genera el token si los datos (usuario y contraseña) que envió el cliente son correctos, como añadidura este utiliza el nombre del cliente como extra para la generación del token.

En el proceso de validación de las credenciales el sistema realiza la verificación a través de una consulta que se ejecuta directamente desde la autenticación. Para filtrar la información y evitar posibles ataques este se ejecuta el **sp\_fm\_login** en una operación que sólo devuelve el usuario, contraseña y el nombre del cliente (que se usa para la creación del token como se mencionó previamente).

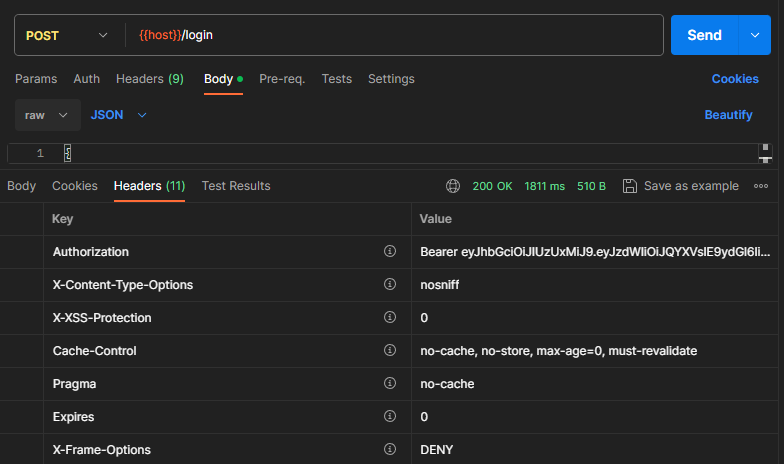


Ilustración Autenticación con Postman

* Para la autorización se gestiona el acceso a la aplicación con el token que se generó previamente, este es validado por el sistema para permitir el acceso al cliente.

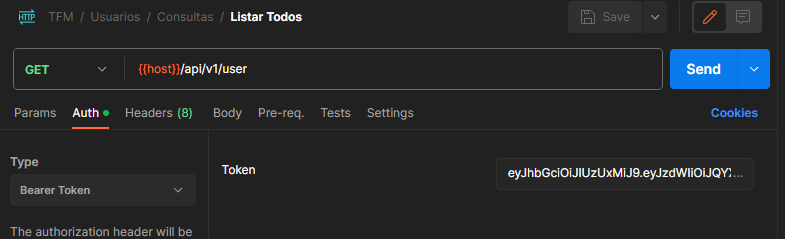


Ilustración Autorización con Postman

**Estructura**

Se aplicó una estructura de conexión que va desde el controlador finalizando en el repositorio que ejecuta la operación en la base de datos, este flujo para el acceso a los datos es esencial puesto que facilita le mantenimiento de la aplicación y aumenta su seguridad.

**Generales:** Con la finalidad de reducir la duplicidad de código se crearon varios paquetes mismos que contienen clases que ayudan a la optimización del código, así como la administración de los componentes, generando que modificaciones específicas se repliquen a otras partes del código.

* + **Constants**: Es una clase tipo enum que integra las constantes que pueden ser utilizadas dentro del proyecto, como son los nombres de los SPs

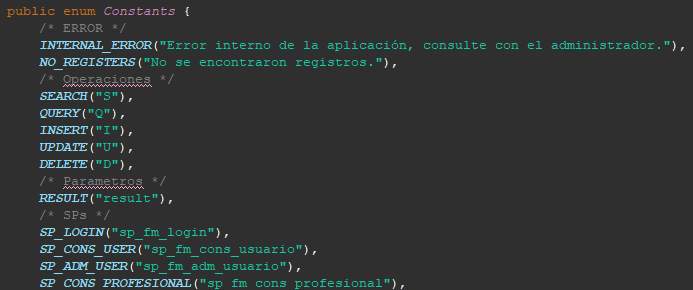


Ilustración fragmento de Constants

* + **GeneralParam:** En esta clase se agrupan los parámetros que son utilizados en la mayoría de los SP, cada repositorio tiene su propio archivo de parámetros, pero para los que son repetitivos se los agrupó en el antes mencionado.

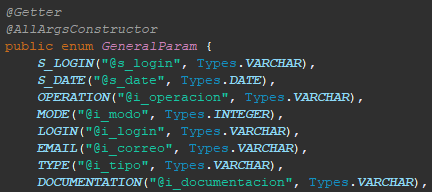
****

Ilustración fragmento de GeneralParam

**Repositorios:** En este se agregó toda la lógica necesaria para realizar las conexiones con la base de datos, creando una clase para cada entidad. Entre las estrategias utilizadas como buena práctica de programación se agregaron dos métodos para realizar las conexiones a los dos SPs de cada uno:

* + **execQueryUser:** este método permite ejecutar los los SPs de consulta.
  + **execAdmUser:** ejecuta las operaciones de administración sobre las tablas.
  + **findOneByLogin:** este método es especial puesto que sólo se encuentra en el repositorio de usuarios, este se utiliza para obtener las credenciales y el nombre del usuario que se está intentando ingresar a la aplicación.

Para la conexión de los repositorios se utilizó **Java EE Database Connectivity** o **JDBC** como es conocido usualmente, esta es un API que permite realizar las operaciones sobre la base de datos, es exclusivo para Java y utiliza el lenguaje SQL para ejecutarlas sobre la base de datos a la que se realiza la conexión.

Entre todos los objetos que tiene **JDBC** se utilizó **SimpleJdbcCall,** este es un objeto que permite realizar la representación a una llamada de un SP o una función almacenada (store función).

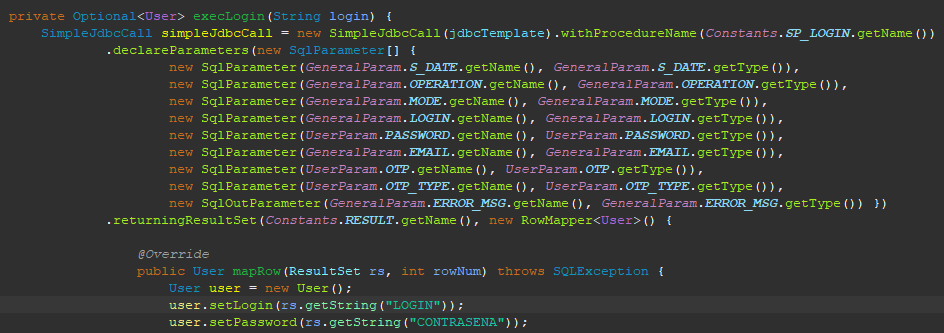


Ilustración fragmento de código de método de execLogin con SimpleJdbcCall

**Servicios:** Estos permiten realizar la conexión a los repositorios, adicional se pueden programar el uso de mapeadores (mappers) para transformar la información que devuelven los repositorios u otro tipo de modelo, en el proyecto actual no fue necesaria esta práctica.

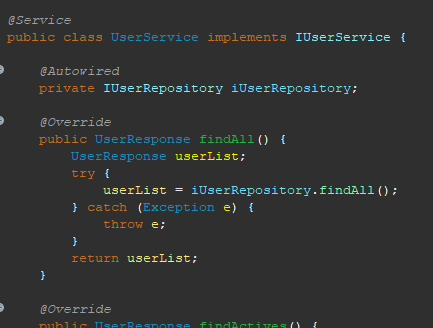


Ilustración fragmento de código de UserService

**Controladores:** En los controladores se realizó la exposición de los servicios rest, al igual que los repositorios y los servicios, cada entidad cuenta con su propio controlador, dividiendo las ejecuciones por operaciones, el tener varios servicios expuestos en vez de uno solo para consultar y/o administrar la información aumenta la seguridad y evita el riego de scripting en la aplicación.



Ilustración fragmento de código de UserController

Para el controlador de los servicios de la tabla usuarios se tiene un caso especial en el que se agregó un método denominado **encryptPass**, este permite realizar la encriptación de la contraseña con **BCryptPasswordEncoder** el cual utiliza la función BCrypt para la encriptación de contraseñas.

**Herramientas utilizadas**

Las herramientas de programación pueden proveer un ambiente de desarrollo ágil y seguro, con la finalidad de optimizar el proceso de desarrollo y la gestión del código se utilizaron varias de estas.

**GIT:** es un sistema de control de versiones de código abierto que permite aumentar la velocidad y eficiencia tanto de proyectos pequeños como grandes.

**GitHub:** es una plataforma de desarrollo para alojar, construir, escalar y entregar código fuente.

**SQL Server Management Studio:** también conocido como SSMS, es un entorno que permite la administración de la infraestructura de SQL, permitiendo acceder a componentes como SQL Server, Azure SQL Managed Instance, entre otros.

**Spring Tool Suite 4 for Eclipse:** es un entorno de desarrollo integrado basado en eclipse que facilita la programación con el framework Spring.

**Postman:** es una plataforma para desarrollar y usar APIs simplificando cada paso de el ciclo de vida de las APIs, permitiendo la integración de varios desarrolladores en la misma.

1. Metodología

Para el desarrollo del TFM se diseñó un diagrama de Gantt con la finalidad de organizar las actividades a desarrollar y ocupar de manera eficiente el trabajo. Para el desarrollo de este diagrama se utilizó la aplicación en línea ClickUp, misma que permite la creación de proyectos que pueden ser compartidos a través de un enlace para su visualización y/o modificación. A continuación, se muestra el diagrama generado para las actividades

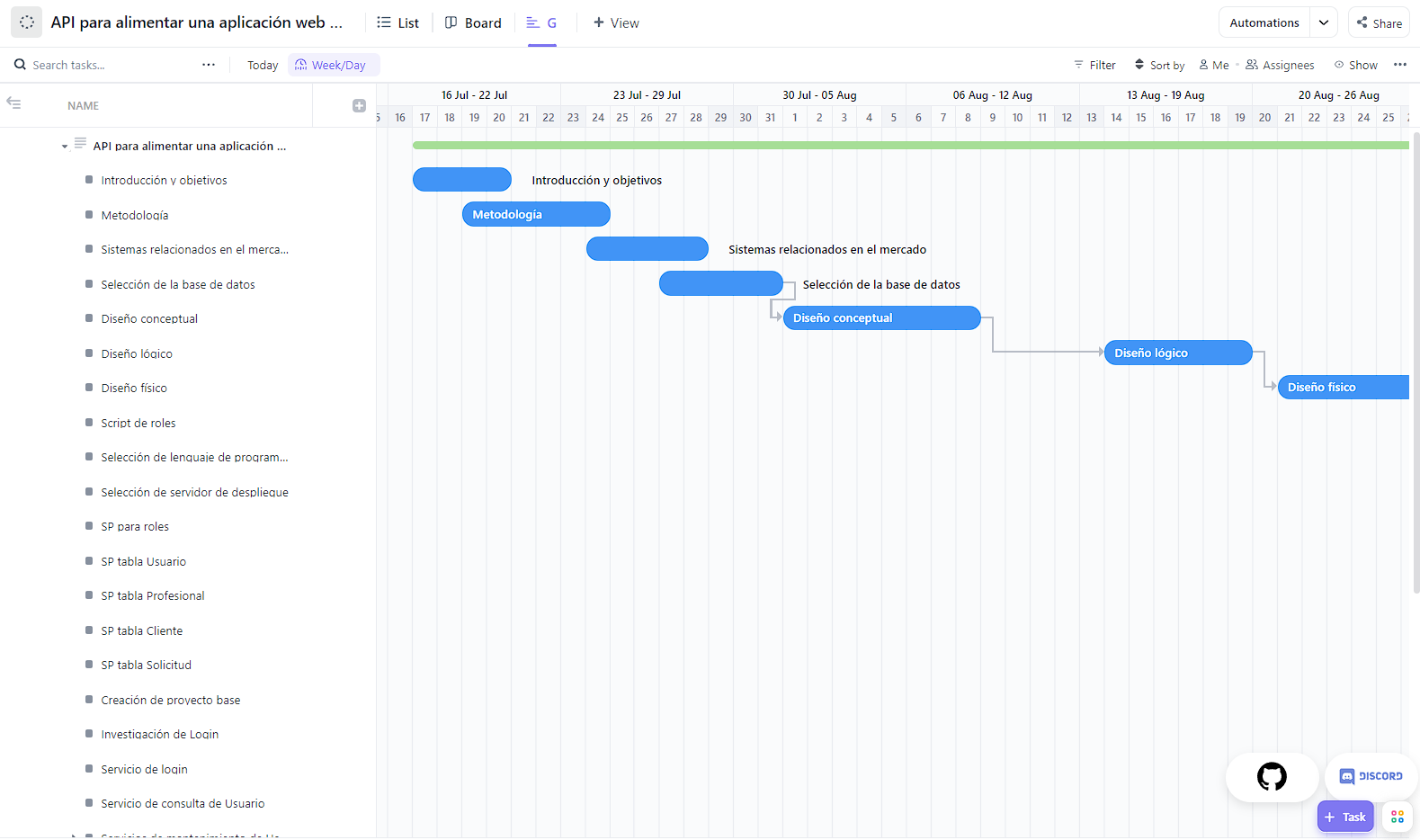


Ilustración diagrama ge Gantt

En la planificación del TFM se excluyó los domingos como día de trabajo, por lo que se podrá observar que algunas de las actividades se pueden desplazar hasta la siguiente semana.

Se utiliza la metodología Scrum, con la finalidad de realizar un proceso de desarrollo con entregables funcionales, por lo que la división de las tareas se realizó por sprint de un mes, iniciando el 17 y terminando el 17 de cada mes. Para el desarrollo de los tableros se utilizó la herramienta Trello, misma que nos permite crear tableros para cada sprint. A continuación, se muestra el tablero del sprint 1 y del sprint 2.

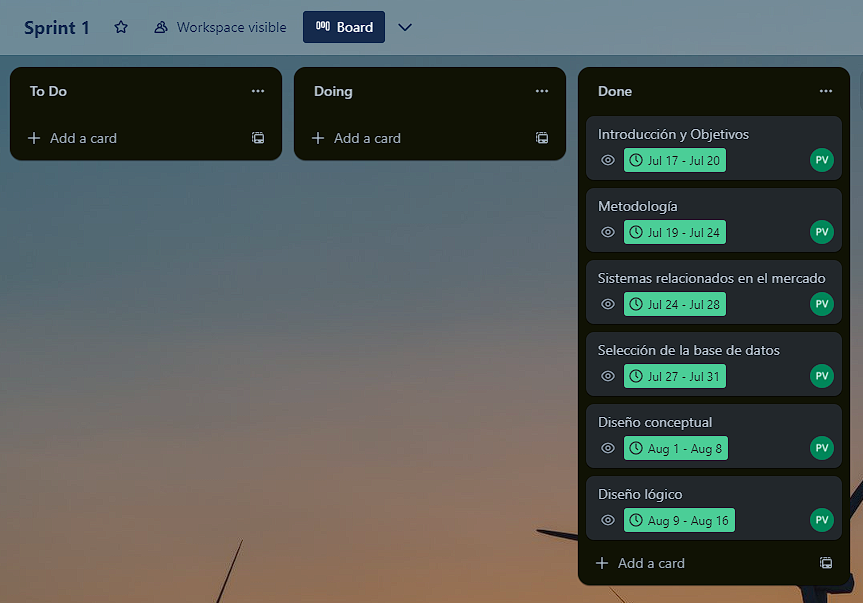


Ilustración tablero Sprint 1

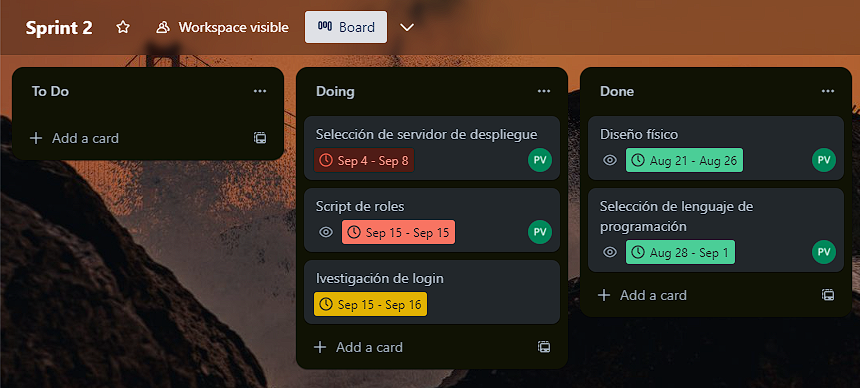


Ilustración tablero sprint 2

Requisitos de la aplicación

Es necesario realizar un modelado de requisitos de forma que se puedan cubrir cada una de las necesidades que tendrá el usuario al momento de utilizar la aplicación, en base a esto tenemos los siguientes requisitos.

Con la finalidad de simplificar los requisitos en estos se conocerá al “API de gestión de solicitudes de personal de la salud a domicilio” simplemente como “sistema”.

* R1. El sistema permitirá al usuario registrarse en la aplicación.
* R2. El sistema permitirá al usuario realizar el ingreso a la aplicación.
* R3. El sistema permitirá al usuario seleccionar si desea ser un cliente o personal de la salud.
* R4. El usuario con rol de cliente podrá crear solicitudes.
* R5. El sistema permitirá al usuario con rol profesional listar las solicitudes relacionadas con el tipo.
* R6. El usuario con rol de profesional podrá aceptar las solicitudes.
* R7. Cuando el usuario intente crear una nueva solicitud con el mismo tipo de una que haya creado previamente el sistema impedirá dicha creación.
* R8. El usuario con rol de cliente podrá administrar las solicitades que haya creado.
* R9. El usuario con rol de profesional podrá cancelar una solicitud previamente aceptada.
* R10. El sistema permitirá al usuario modificar sus datos luego de haberse registrado.
* R11. El sistema le permitirá al usuario con rol profesional filtrar las solicitudes por tipo.
* R12. El usuario con rol administrador tendrá acceso a todo el sistema.
* R13. El usuario con rol revisor podrá cambiar el estado de los profesionales y clientes.
* R14. El usuario con rol revisor podrá modificar el campo documentación de los profesionales y clientes.
* R15. El usuario con rol cliente o profesional puede marcar una solicitud con estado incompleta.
* R16. El usuario con rol revisor podrá revisar las solicitudes marcadas como incompletas.

Cada requisito cuenta con un identificar de forma que se los pueda relacionar directamente en la etapa de modelado.

Modelos conceptuales de requisitos

El modelado de los requisitos se realizó con casos de uso puesto que estos permiten describir las funcionalidades de forma que cualquier persona los pueda interpretar. Debido a la gran cantidad de requisitos a continuación se muestra el modelado de dos de ellos.

* R10. El sistema permitirá al usuario modificar sus datos luego de haberse registrado.

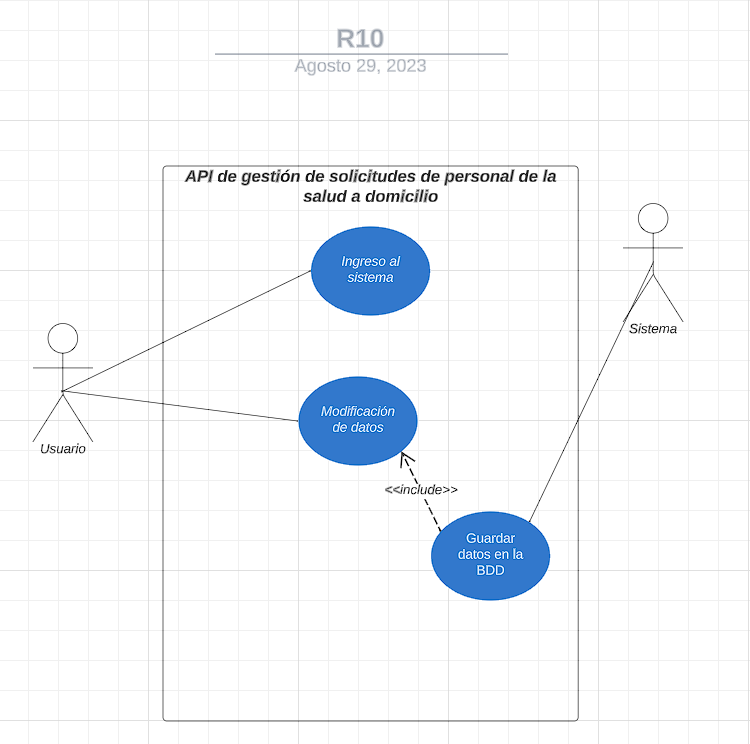


Ilustración caso de uso de requisito R10.

* R14. El usuario con rol revisor podrá modificar el campo documentación de los profesionales y clientes.

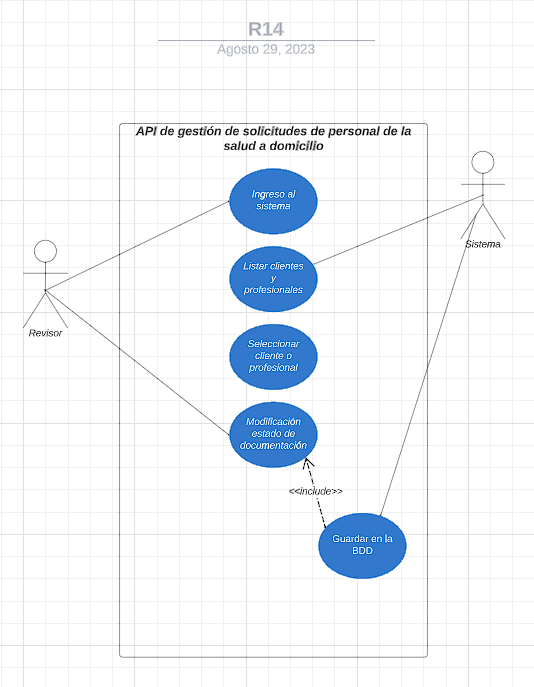


Ilustración caso de uso de requisito R14

1. Resultados y discusión

El presente trabajo de fin de máster nos deja con una API desarrollado en el lenguaje java con Spring Boot como framework y con una base de datos completamente funcional desarrollada en SQL Server.

Esta API podrá ser utilizada por un ente que requiera desarrollar una aplicación relacionada con la salud y más específicamente por aquellos que requieran exponer una aplicación para proveer servidores de la salud a domicilio, esto requerirá que se realice la programación del frontend necesario.

Es preciso entender que esta provee las bases necesarias para poder desarrolla una aplicación de solicitud de profesionales de la salud a domicilio, misma estructura que permitirá la implementación de las reglas de negocio que se requiera para realizar la programación de frontend.

El código fuente de esta se encuentra subido a GitHub como se menciono previamente, por lo que podrá ser accedido

Como se mencionó previamente se utilizó la herramienta Postman para poder probar los servicios rest expuesto de la aplicación, a continuación, se adjuntarán funcionalidades del API desde la herramienta antes mencionada. Dejando como resultado una colección de Postman que contienen las solicitudes necesarias para probar cada endpoint desarrollado.

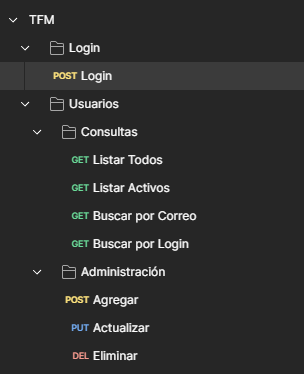


Ilustración Colección de Postman

Para poder acceder a la aplicación se debe realizar la autenticación de la misma, mediante el endpoint ‘/login’, este nos proveerá el token que requeriremos para la ejecución de los demás servicios. Para el ejemplo actual se ingresará con el

1. Conclusiones y trabajo futuro
2. Bibliografía
3. Anexos.

Anexo 1