

**División de Electrónica y Computación**

Departamento de Ciencias Computacionales

**Ingeniería en Computación**

**PROYECTO MODULAR**

**ON MY WAY**

**para acreditar los módulos**

MÓDULO I. ARQUITECTURA Y PROGRAMACIÓN DE SISTEMAS

MODULO II. SISTEMAS INTELIGENTES

MODULO III. SISTEMAS DISTRIBUIDOS

**P r e s e n t a**

SANCHEZ GONZALEZ ALDO, RODRIGO HERRERA ORNELAS, LEONARDO

**A s e s o r**

RAMOS BARAJAS, ARMANDO

Guadalajara, Jalisco, 11 de Abril de 2019

ÍNDICE

[CONTENIDO 3](#_Toc10113384)

[INTRODUCCIÓN 3](#_Toc10113385)

[PLANTEMIENTO DEL PROBLEMA 3](#_Toc10113386)

[JUSTIFICACIÓN 4](#_Toc10113387)

[ANTECEDENTES ESTADO DEL ARTE 5](#_Toc10113388)

[HIPÓTESIS 6](#_Toc10113389)

[METODOLOGÍA 7](#_Toc10113390)

[DISEÑO Y FASE DEL PROTOTIPADO DEL PROYECTO MODULAR 8](#_Toc10113391)

[1. Diagrama Entidad Relación 8](#_Toc10113392)

[2. Diccionario de datos del sistema. 13](#_Toc10113393)

[3. Diagramas de clases del sistema 25](#_Toc10113394)

[3.1 Controladores de Autenticación 25](#_Toc10113395)

[3.2 Controladores del núcleo del sistema 28](#_Toc10113396)

[3.3 Clases de tipo servicio 30](#_Toc10113397)

[3.4 Modelos del sistema 33](#_Toc10113398)

[CONCLUSIONES 35](#_Toc10113399)

[REFERENCIAS BIBLIOGRÁFICAS 35](#_Toc10113400)

**CONTENIDO**

INTRODUCCIÓN

Actualmente, existen compañías que se dedican a realizar el traslado de paquetes de una parte a otra de la ciudad, país o incluso del mundo. El problema con estas compañías es que no ofrecen demasiadas comodidades al momento de contratar sus servicios. Muchas de estas, no ofrecen al usuario la posibilidad de ver qué pasó con el paquete que se quiere enviar o recibir, si es que se ha perdido, si es que llegará con un retraso, etc. Provocando en ciertas ocasiones, desconfianza a los usuarios.

El principal problema que se puede percibir es que, si alguien desea enviar un paquete, en la mayoría de las veces, es necesario acudir directamente a la oficina de la compañía por medio de la cual se desea enviar, muchas de estas compañías muchas veces no ofrecen servicio a domicilio obligando al usuario a que tenga que ir de manera personal a la oficina.

Con el panorama anteriormente descrito, podemos observar que es necesario y de gran ayuda tener una aplicación, que nos ayude a conocer los detalles de nuestro envío, si es que el paquete no podrá llegar por alguna razón, por cuales lugares este ha pasado, qué otros lugares faltan por recorrer y cómo podríamos tener un contacto directo con el repartidor para que este nos comunique cuando haya algún imprevisto o algún tipo de retraso.

Teniendo ya una base de datos con información referente a los tipos de paquetes que se han entregado, el tipo de público al que se les está mandando y otros datos de interés, es posible realizar regresiones matemáticas para poder aplicar predicciones que sean de interés para los clientes de la aplicación o para los administradores de la misma.

PLANTEMIENTO DEL PROBLEMA

En muchas ocasiones, al momento de enviar un paquete, consideramos que el usuario desearía tener una plataforma, de preferencia móvil, donde sea posible solicitar a alguien que recoja su paquete y lo envíe a otra persona, sería mejor aún, si la empresa que administra la aplicación se hiciera responsable si es que el paquete resultó dañado o en el peor de los casos, extraviado. Además de esto, sería útil tener la posibilidad de rastrear el paquete que se está enviando, un rastreo que incluye principalmente, por qué lugares ha pasado el paquete y a donde es que se dirige ahora.

Otro caso particular, es cuando un usuario desea hacer un pedido a corta distancia, es decir, que una aplicación móvil le diera la oportunidad de poder mandar un paquete dentro de la misma área metropolitana, cosa que las empresas de paquetería no hacen.

Muchas empresas de paquetería, ofrecen a sus usuarios una manera de saber dónde está su paquete actualmente, sin embargo resultan deficientes al momento de que el usuario trata de buscar información más detallada, sin mencionar que la comunicación con la empresa repartidora es, la mayoría de las veces, muy lenta y no ofrecen una manera de comunicarse con el repartidor para que él mismo nos diga el estado del pedido que el usuario ha hecho.

Los principales puntos en los que creemos que la industria de la paquetería debería mejorar, son los siguientes:

* Ofrecer al cliente la posibilidad de pedir a alguien que entregue un paquete, a corta distancia (50 kilómetros a la redonda por ejemplo), a distancia media (125 kilómetros a la redonda) y a larga distancia (250 kilómetros a la redonda) o incluso mas lejos, como muchas empresas de paquetería ya lo hacen.
* Ofrecer una aplicación móvil con la finalidad de que esta sirva como herramienta para administrar las entregas que se han hecho.
* Ofrecer al cliente una comunicación directa con el repartidor de su paquete por medio de mensajería dentro de la aplicación.
* Ofrecer al cliente un rastreo explícito de dónde ha estado y a donde irá su paquete, desde el momento en que el pedido haya sido aceptado para enviar hasta que el producto llega a la puerta del cliente.
* Ofrecer a los usuarios tanto enviar como recibir productos, es decir, no es necesario tener un tipo de cuenta diferente para poder solicitar un envió, o solicitar un repartidor y enviar un producto.

JUSTIFICACIÓN

Hoy en día, en especial para las personas que viven en ciudades grandes y pobladas, es mucho más cómodo que el mayor número de servicios posibles se puedan solicitar mediante aplicaciones en un teléfono móvil, uno de estos servicios es el de paquetería, el poder enviar paquetes y recibir de un sitio a otro es una tarea bastante común que hoy en día no ha sido tan modernizada.

Por otro lado, el modernizar un servicio como el de paquetería, para que esté disponible a manera de aplicación móvil, traería consigo diversos cambios en el modelo de negocio. Estos cambios podrían ser:

* Ya no es necesario que la empresa tenga que invertir en un transporte especial para el traslado de paquetes, siempre y cuando se le remunere bien económicamente al repartidor por usar su propio transporte. Lo que representaría un ahorro significativo para la empresa propietaria de la aplicación.
* Los clientes podrían pedir de manera instantánea un repartidor que lleve su paquete y este será atendido inmediatamente siempre y cuando haya un repartidor que trabaje el número de kilómetros que este paquete va a recorrer.
* **OnMyWay** pretende abarcar un mercado tanto municipal, como de entregas a larga distancia, logrando de esta manera, un número de usuarios mas elevado, ya que, las empresas dedicadas a mensajería y paquetería hoy en día, usualmente no trabajan a distancias tan cortas, y otras aplicaciones disponibles en el mercado para trasladar encargos pequeños, muchas veces no trabajan a distancias tan largas.
* En una aplicación móvil sería posible hacer que los usuarios califiquen el servicio que se les está brindando, y esas calificaciones pueden ser visibles para el usuario al abrir la aplicación, siendo esta una ventaja sobre las empresas de paquetería actuales en donde, el usuario que requiere el servicio tendría que meterse al portal web y ver la evaluación del servicio, en el caso de que estas empresas ofrezcan esa posibilidad.
* El hecho de poder ver una evaluación de los repartidores con base en las entregas que este ha realizado, le brinda al cliente, tanto el que envía como el que recibe, mayor información. Al haber mayor información el cliente podría sentirse mas confiado al momento de dejar su paquete en manos de la empresa o del repartidor asignado.
* El informe estadístico que brinda la aplicación permite a los administradores de la misma conocer en qué ciudades el uso de la aplicación esta siendo mas elevado, y en que ciudades se podría necesitar algún tipo de campaña publicitaria para elevar el número de usuarios.

Estos cambios en el modelo de negocio tradicional, podrían brindarle al usuario una mejor experiencia al momento de requerir este servicio. Además creemos que, si bien puede haber desventajas al momento de modernizar de esta manera un servicio como el de paquetería, este nuevo modelo de negocio es suficiente para que el usuario se sienta seguro de enviar y recibir paquetes de esta manera.

Las desventajas que puede traer el brindar el servicio de paquetería de esta manera, no distan mucho de las amenazas que pueden surgir con el modelo de negocio tradicional, es por eso que se tiene en consideración que es necesario contratar el mismo tipo de servicios que una empresa de paquetería tradicional. Servicios tales como asesoría legal en caso de extravío de paquetes, un proceso de selección adecuado para los repartidores, pruebas psicométricas, etc.

Los usuarios de las empresas de paqueterías aun dependen de varios factores que podrían evitarse al dejar que ellos tengan interacción por medio de una aplicación móvil, tales factores incluyen por ejemplo:

* Tener que llevar el paquete por ti mismo a la oficina.
* Tener que marcar continuamente a la oficina para saber qué pasó con el paquete que aún no ha llegado.
* Retraso significativo de los paquetes sin informar al cliente de lo sucedido.

ANTECEDENTES ESTADO DEL ARTE

# **Acerca del negocio tradicional de mensajería y paquetería:**

El servicio de mensajería de paquetes y documentos es demandado por prácticamente todos los eslabones de una cadena productiva. Como en todo servicio, su éxito dependerá de la eficiencia y calidad con que se brinde.

"Hay más de tres mil empresas que ofrecen el servicio de envío de documentos y paquetes. Estos negocios, además, se dedican a distribuir, almacenar y gestionar todo tipo de mercancías. Se trata de un sector muy dinámico que crece de forma importante (10 por ciento anual), a consecuencia de las tendencias globales que, gracias a la generalización de procesos como los de **just in time** o los de **cero inventario** que manejan las industrias, requieren de la subcontratación de estos servicios", afirma Manuel Armendáriz Etchegaray, presidente de la Asociación Mexicana de Mensajería y Paquetería, AC (AMMPAC).

En México, las empresas de mensajería trabajan bajo un sistema intermodal (es decir, combinan más de un tipo de transporte, generalmente el terrestre y el aéreo), existiendo diversos tipos de operadores.

Considerando las condiciones de precios, tarifas, cantidades, medidas, cargos, plazos o fechas, aparte de la entrega a domicilio, el servicio tradicional se puede prestar en distintas modalidades como:

* **Envío o flete normal:** documentar los requerimientos.
* **Envío o flete por cobrar:** el pago lo realiza el destinatario al recibir el flete.Los requerimientos son codificados.
* **Ocurre:** el destinatario acude a recoger los envíos en las oficinas del negocio.
* **Entrega a detalle:** una vez entregado se revisa cada uno de los artículos o documentos que fueron enviados, y puede ser servicio a domicilio o servicio ocurre.
* **Recolección:** se recoge la mercancía en el domicilio del remitente.
* **Acuse de recibo:** se ofrece a aquéllos clientes que requieren de un comprobante del envío, y es un servicio adicional ya que implica volver a visitar al cliente para entregarle el acuse.
* **Recolección:** se recoge la mercancía en el domicilio del remitente.
* **Seguro:** dependiendo del valor de la mercancía que se declare, se cobrará la prima correspondiente para proteger los envíos de algún siniestro, como daños, robos o percances de tránsito.

Se distinguen principalmente dos tipos de cliente: particulares y empresas. Las necesidades de estos clientes pueden ser eventuales "como el público que acude al mostrador para hacer un único envío" o permanentes, que sería el caso de un hotel que requiere del envío de la factura al lugar de origen de determinado huésped.

**Acerca del reciente éxito de las aplicaciones móviles:**

A medida que crece la difusión de los teléfonos celulares, aumenta también la popularidad de las aplicaciones móviles. Según un estudio llevado a cabo en Alemania por [Bitkom](http://www.bitkom.org/" \t "/home/lherrera/Documents\\x/_blank), más de 15 millones de usuarios han descargado ya aplicaciones en sus dispositivos móviles. Esta cifra constituye un crecimiento del 50% con respecto al año anterior.

René Schuster, miembro de la directiva de Bitkom, asegura que al mercado de las aplicaciones móviles le queda aún mucho para tocar techo. “Este mercado no ha hecho sino empezar a andar y tiene muchísimo potencial”, explica Schuster.

Los usuarios de teléfonos celulares tienen de media 17 aplicaciones en sus móviles. La proporción de uso de estos programas aumenta a medida que desciende la edad media del consumidor. Así, los usuarios de entre 14 y 29 años tienen instaladas una media de 22 aplicaciones móviles en sus smartphones, informa Horizont.

HIPÓTESIS

Uno de los objetivos de los sistemas distribuidos es que, estos permiten que varias aplicaciones puedan comunicarse y compartirse información entre ellas. Existen diversas formas de comunicación entre dispositivos, sin embargo, un caso de uso muy común hoy en día es que las aplicaciones envíen a otro programa información a través de una **API**, lo cual es básicamente un conjunto de funciones de alto nivel que proveen acceso a funciones de un software. Las **APIs** son usadas frecuentemente en comunicación cliente-servidor, aunque su uso no esta estrictamente limitado a este campo.

La regresión estadística nos permite determinar el grado de dependencia de las series de valores X e Y, prediciendo el valor Y estimado que se obtendría para un valor X que no esté en la distribución.

La importancia de las distribuciones bidimensionales radica en investigar cómo influye una variable sobre la otra. Esta puede ser una dependencia causa efecto, por ejemplo, la cantidad de lluvia (causa), da lugar a un aumento de la producción agrícola (efecto). O bien, el aumento del precio de un bien, da lugar a una disminución de la cantidad demandada del mismo.

Tomando en cuenta esta información podemos aplicar ambos enfoques para desarrollar una plataforma móvil que haga uso de sistemas distribuidos para comunicarse con otro tipo de servicios externos y asimismo haga uso de regresión lineal para poder hacer predicciones con la información recogida apropiadamente y de esta manera brindarle al usuario una experiencia satisfactoria.

La plataforma móvil pretende comunicarse con un servidor web por medio de una **API** para obtener recursos de una base de datos remota y además pretende usar el método de **regresión lineal** para poder brindarle al usuario de la aplicación un historial de sus ventas en meses pasados y brindarle una predicción del número de posibles ventas en los próximos meses. Los datos con los que se realizará la regresión lineal se obtienen desde la base de datos remota y se sacaran por medio de funciones de la **API.**

Se le brindará también a los administradores de la aplicación un módulo para realizar **regresión lineal** de las entregas hechas por municipio, con la intención de que se tenga un conocimiento de en qué partes de la república es conveniente hacer un esfuerzo por elevar el número de usuarios.

METODOLOGÍA

Como equipo, hemos escogido ciertas metodologías para resolver las tareas que conforman el proyecto.

Como metodología de desarrollo de software, optamos por el **modelo en V**, el cual define un procedimiento uniforme para el desarrollo de sistemas computacionales o diversos tipos de productos tecnológicos. Esta metodología es una representación gráfica del ciclo de vida del desarrollo de sistemas. Este describe las actividades y resultados que deben producirse durante el desarrollo del producto. El lado izquierdo de la V representa la descomposición de las necesidades y la creación de las especificaciones del sistema. El lado derecho de la V representa la integración de las piezas y su verificación. “V” significa verificación y validación.

Las fases de esta metodología incluyen:

* Recoger requerimientos del cliente
* Documentar los requerimientos
* Traducir requerimientos en un lenguaje técnico de manera que puedan ser codificados
* Los requerimientos son codificados
* Las pruebas unitarias se comparan contra los requerimientos a nivel software
* Las pruebas de componentes se comparan contra los requerimientos técnicos
* Las pruebas del sistema se comparan contra los requerimientos a alto nivel
* Las pruebas de aceptación se comparan contra lo que el cliente dice que el quisiera tener, es decir, en este punto se valida que el sistema hace lo que el cliente espera que haga.

Al definir los pasos anteriores, podemos ver que para cada paso del lado izquierdo de la V (primeros 4 puntos) existe su contra parte del lado derecho (últimos 4 puntos).

La metodología de diseño de software, como ya se mencionó anteriormente es **MVC (Model View Controller)**. Esta metodología, que actualmente está en su auge, es un tipo de arquitectura de software que dicta una serie de normas a seguir para comunicar las distintas partes que conforman una aplicación.

Básicamente, en **MVC**, **los modelos** (lo que en modelo Entidad - Relación se conoce como entidades) son los encargados de hacer operaciones con la base de datos, a través de algún lenguaje de consultas o en su defecto, la herramienta utilizada para hacer estas consultas (ORM). **Los controladores**, son clases encargadas de funcionar como intermediarios entre los modelos y las vistas. Los controladores, en un panorama ideal, deben solamente recibir información de la vista hacia el modelo, para que el modelo sepa con que datos tiene que trabajar, y debe también de mandar datos desde el modelo hacia la vista para que la vista sepa que información tiene que mostrar al cliente.

**Laravel**, usa de manera interna muchos patrones de diseño que hacen el desarrollo más sencillo y abstracto, entre estos patrones comunes se encuentran: **Factory, Singleton, Decorator, Iterator, etc.**

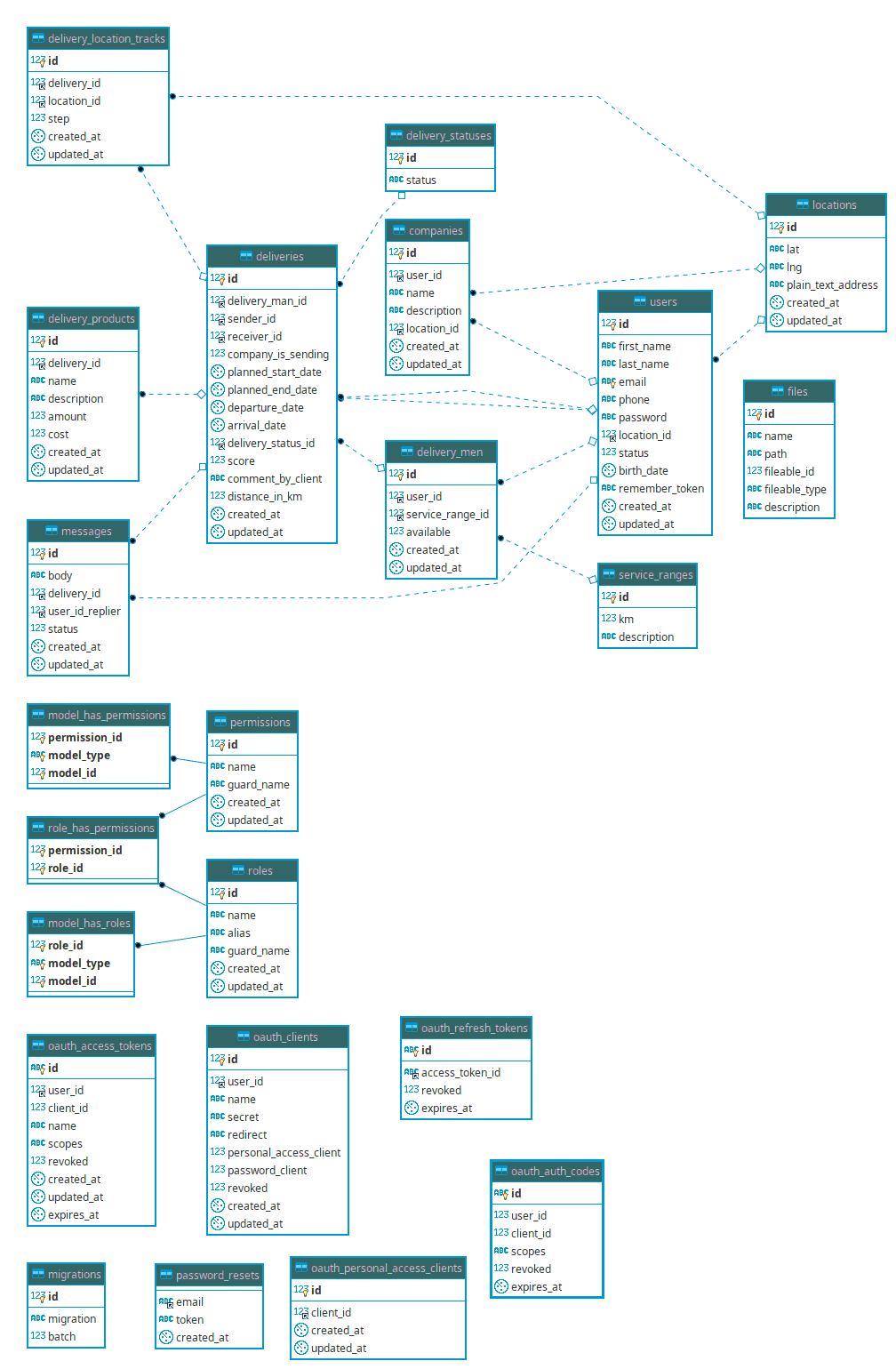
DISEÑO Y FASE DEL PROTOTIPADO DEL PROYECTO MODULAR

1. Diagrama Entidad Relación

Lo primero que se mostrará es un diagrama entidad-relación que representa cómo está organizado el proyecto a nivel base de datos. Se explicaran las tablas divididas por módulos y cada conjunto de tablas mostrado contendrá una explicación de la razón de ser de cada tabla en el sistema.

1.1 Tablas primordiales del sistema.

La primera parte del diagrama entidad relación, modela el núcleo de **onMyWay**, es decir, las tablas esenciales para que el sistema cumpla con su funcionamiento.



**users:** esta tabla almacena la información general de todos los usuarios registrados en el sistema. Independientemente del rol que ellos tengan. Todos los usuarios que pasan por el sistema se registran en esta tabla. Esta tabla guarda información como el correo electrónico, contraseña, nombre y apellidos, etc.

**locations:** almacena las coordenadas de la localización que el usuario registrado especificó como su localización de registro. Ademas de guardar las coordenadas, esta tabla reserva también un campo para escribir el domicilio en texto plano del usuario. Es decir, la calle, colonia, ciudad, estado y país donde esté viva. Este campo se reservó con el fin poder tener más información para realizar estadísticas con la información del sistema.

**companies:** las compañías, son otro tipo de rol en el sistema. Una compañía, al igual que un usuario normal (a excepción de un repartidor) puede enviar y recibir paquetes, la diferencia es que, como compañía, es necesario capturar otro tipo de información como el nombre de la compañía y una pequeña descripción. Esta tabla contiene una llave foránea a un registro en la tabla de usuarios, para ligar a la compañía con el usuario que la posee.

**delivery\_men:** al igual que la entidad **companies**, esta entidad representa otro rol en el sistema, en esta tabla se guarda información pertinente al número de kilómetros que el repartidor trabaja y si los repartidores están disponibles o no. (Cuando un repartidor está en el proceso de realizar una entrega, el repartidor no debería aparecer como disponible para realizar otras entregas).

**files:** esta tabla guarda las fotos de perfil de los usuarios registrados en el sistema y las imágenes de los productos que pertenecen a una entrega. Sin embargo, está pensada como una tabla polimórfica para guardar todo tipo de archivos ligados a todo tipo de entidad en el sistema.

**service\_ranges:** con esta tabla se busca almacenar las distancias disponibles para que los repartidores escojan para trabajar. Los valores para esta tabla son:

* Local (50 km a la redonda)
* Corta (125 km a la redonda)
* Media (250 km a la redonda)
* Media Larga (500 km a la redonda)
* Larga (700 km a la redonda)
* Muy Larga (1000 km a la redonda)

**deliveries:** la tabla más importante del sistema. Esta almacena todo lo referente a las entregas. Almacena su estatus (entregada, cancelada, en proceso, etc.), el repartidor asignado a ella, el cliente que enviar, el cliente que recibe y otra información importante como las fechas de salida y la fecha de llegada del paquete. Podemos observar en la figura mostrada anteriormente, que esta es la tabla que se encuentra relacionada con más entidades.

**delivery\_statuses:** esta tabla guarda los tipos de estatus que las entregas pueden tener. Los valores para esta tabla son:

* Creando
* No Asignado
* No Iniciado
* En Progreso
* Entregado
* Cancelado

**delivery\_location\_tracks:** cuando una entrega está siendo realizada, esta tabla es necesaria para guardar los lugares que el repartidor de esa entrega ha recorrido. Podemos observar en la figura anterior que se tiene una relación con la tabla de **deliveries** y otra con la tabla de **locations**, de esta manera hemos definido una relación de **n** a **m** entre esas dos entidades.

**delivery\_products:** las entregas tienen que tener una imagen asociada, con la finalidad de que ambos, el usuario que envía y el que reciba puedan ver una imagen real del producto involucrado en una entrega. Esta tabla identifica los productos y su información de interes como su costo, y los relaciona con entregas.

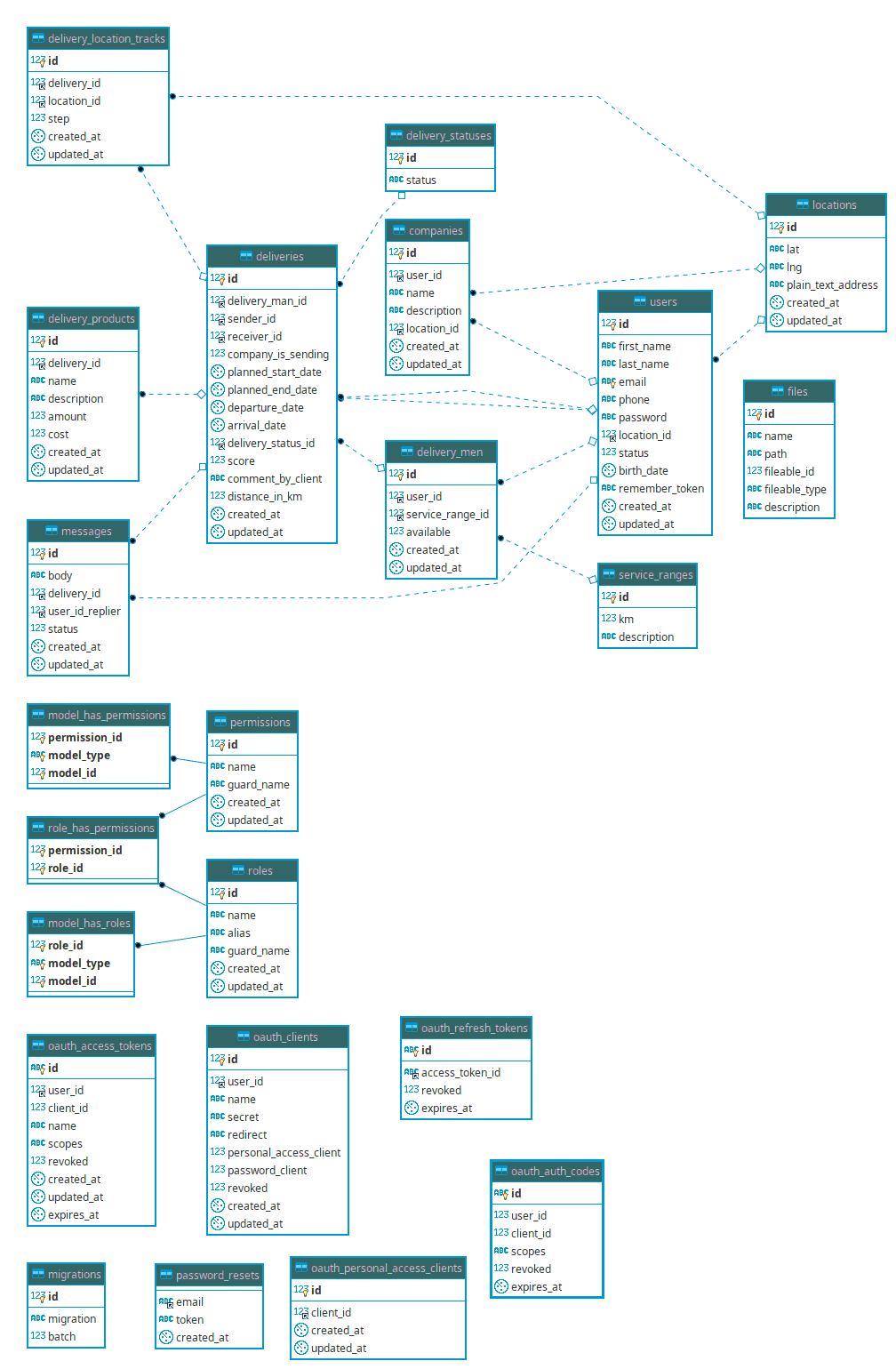
**messages:** existe un sistema de mensajería instantánea entre el cliente y el administrador del sistema. Esta tabla guarda las conversaciones entre repartidores y clientes.

1.2 Sistema de Roles y Permisos

El conjunto de tablas que se muestra a continuación pretende representar la administración de roles y permisos dentro del sistema. En la aplicación se tienen solo cuatro roles:

* Administrador
* Repartidor
* Cliente
* Compañía

Es importante mencionar que, el sistema de roles y permisos fue un paquete externo a nuestro sistema llamado Spatie, instalado y montado en **Laravel** para hacerlo funcionar de acuerdo a nuestras necesidades. Por lo que este conjunto de tablas no fue pensado e ideado por nosotros. Sin embargo sabemos cómo funciona internamente y se llegó a un acuerdo sobre escoger este paquete externo en lugar de otros disponibles.



**model\_has\_permission:** esta tabla guarda la relación entre un usuario y los permisos que este posee. Se le nombra **model** al usuario ya que esta tabla es pensada para ser polimórfica y funcionar con distintos tipos de clases ademas de un usuario.

**role\_has\_permission:** esta tabla guarda la relación entre un rol y los permisos que este posee. Muy similar a la manera en que funciona **model\_has\_permission.**

**model\_has\_role:** tabla encargada de relacionar un usuario con los roles que este posee, esta tabla también pretende ser polimórfica y funcionar para todo tipo de entidades ademas de usuario.

**permissions:** esta tabla modela los permisos que se tienen en el sistema. Permisos que pueden ser aplicables tanto a usuarios como a roles. Los permisos generalmente representan pequeñas acciones como **guardar usuario, editar compañía, eliminar folio, etc**. Y por lo general, es un rol específico quien hace uso de todas estas acciones.

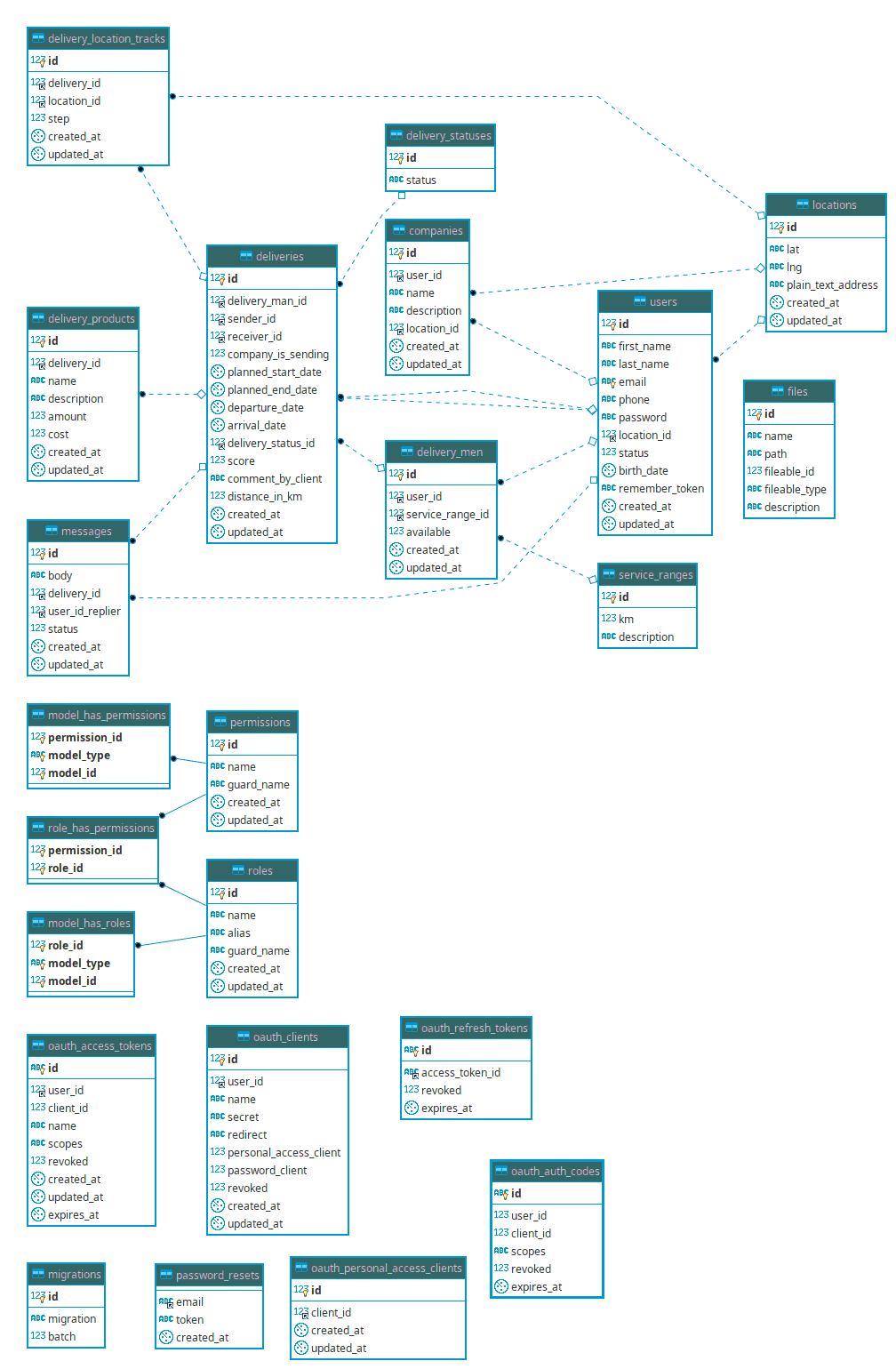
**roles:** en esta tabla se modelan todos los tipos de usuario que estarán usando el sistema. Generalmente un rol, posee un conjunto casi único de permisos diferente al de los demás roles, sin embargo, frecuentemente se da el caso en que muchos permisos puedan ser compartidos por otros roles.

Teniendo esta estructura de roles y permisos nos aseguramos de que el sistema puede adaptarse sin problemas a nuevos roles y permisos que puedan surgir más delante, y que tendremos un manejo más libre de los permisos usados en el sistemas, al poder estos ser asignados a un rol o a un usuario, y en el caso de usar polimorfismo, a cualquier otra entidad.

1.3 Sistema de autenticación

Ademas de las tablas que representan el núcleo del sistema y el sistema de roles y permisos, se tiene también un sistema de autenticación, que modela todo lo necesario para guardar las sesiones del usuario y su tiempo de actividad en el sistema. Sirve ademas como módulo de registro y restablecimiento de contraseñas de los usuarios.

Este módulo al igual que el de roles y permisos, tiene que ser instalado ya que no viene incluido en el framework. Es necesario ejecutar **composer install passport** para instalar el plugin en nuestro sistema y una vez que se tenga instalado ejecutar **php artisan passport:install** para inicializar el módulo y crear las tablas necesarias en nuestra base de datos.



**migrations:** esta tabla es parte del framework de **laravel** y es usada para guardar el registro de los cambios a la base de datos para poder hacer **rollback** y deshacer las operaciones ya hechas si es necesario.

**password\_resets:** esta tabla también es parte de **laravel**, y guarda el registro de los usuarios que solicitan un cambio de contraseña en el sistema. Una vez logrado el cambio de contraseña se eliminar la fila correspondiente a esa solicitud.

1. Diccionario de datos del sistema.

|  |  |  |
| --- | --- | --- |
| **companies** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| user\_id | int(10) | llave foránea al usuario al que pertenece la compañía |
| name | varchar(255) | nombre de la compañía |
| description | text | descripción de la compañía |
| location\_id | int(10) | Llave foránea al registro que guarda la localización de la empresa. |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de ultima actualización |

|  |  |  |
| --- | --- | --- |
| **deliveries** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| delivery\_man\_id | int(10) | Llave foránea que referencía al repartidor que ha enviado un paquete |
| sender\_id | int(10) | llave foránea que referencía al usuario que envía un paquete |
| receiver\_id | Int(10) | llave foránea que referencía al usuario que recibe un paquete |
| company\_is\_sending | tinyint(1) | campo de tipo bandera que representa si una entrega esta siendo enviada por un usuario o una compañía |
| planned\_start\_date | datetime | la fecha en la que se tiene que un paquete salga del punto de partida |
| planned\_end\_date | datetime | la fecha en la que se tiene planeado que un paquete llegue a su punto final |
| departure\_date | datetime | la fecha real en la que el paquete sale del punto de partida |
| arrival\_date | datetime | la fecha real en la que el paquete llega a su punto final |
| delivery\_status\_id | int(10) | llave foránea al status en el que se encuentra la entrega |
| score | decimal(8, 2) | guarda la evaluación de la entrega por parte del cliente que recibe el paquete |
| comment\_by\_client | text | comentario del cliente que recibe el paquete hacia la entrega en general |
| distance\_in\_km | decimal(8,2) | distancia en kilómetros recorrida durante una entrega |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **delivery\_location\_tracks** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| delivery\_id | int(10) | llave foránea a la entrega que se esta siguiendo |
| location\_id | int(10) | llave foránea a la localización en donde se encuentra la entrega |
| step | varchar(255) | numero de destinos diferentes que un paquete ha recorrido |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **delivery\_men** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| user\_id | int(10) | Llave foránea al usuario al que esta asociado un repartidor |
| service\_range\_id | int(10) | Llave foránea al campo que define el numero de kilómetros que un repartidor esta dispuesto a recorrer |
| available | tinyint(1) | campo booleano que verifica si un repartidor se encuentra disponible para asignar a una entrega |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **delivery\_products** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| delivery\_id | int(10) | llave foránea a la entrega a la que un producto pertenece |
| name | varchar(255) | nombre del producto |
| description | text | descripción del producto |
| amount | decimal(18,2) | cantidad de productos |
| cost | decimal(8,2) | costo del producto a ser enviado |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **delivery\_statuses** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| status | varchar(500) | nombre del estatus |

|  |  |  |
| --- | --- | --- |
| **files** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| name | varchar(255) | nombre del archivo |
| path | varchar(255) | ruta del archivo |
| fileable\_id | int(10) | id de la entidad a la que pertenece el archivo |
| fileable\_type | int(10) | nombre de la entidad a la que pertenece el archivo |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **locations** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| lat | varchar(255) | latitud de la localización |
| lng | varchar(255) | Iongitud de la localización |
| plain\_text\_address | varchar(255) | Domicilio en texto plano correspondiente a las coodernadas brindadas |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **messages** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| body | text | cuerpo del mensaje |
| delivery\_id | int(10) | llave foránea que referencía la entrega a la que el mensaje pertenece |
| user\_id\_replier | int(10) | llave foránea al usuario que envía el mensaje |
| created\_at | timestamp | Fecha de creación |
| updated\_at | timestamp | Fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **migrations** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| migration | varchar(255) | nombre del archivo donde se encuentra la función a ser llamada al ejecutar y hacer rollback de una migración |
| bacth | int(11) | Iote de transacciones pendientes ejecutadas |

|  |  |  |
| --- | --- | --- |
| **model\_has\_permission** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| permission\_id | int(10) | permiso al que se hace referencia |
| model\_type | varchar(255) | nombre de la entidad a la que se le esta asignando el permiso |
| model\_id | int(10) | id de la entidad a la que se hace referencia |

|  |  |  |
| --- | --- | --- |
| **model\_has\_roles** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| role\_id | int(10) | campo llave primaria auto incrementable |
| model\_type | varchar(255) | nombre de la entidad a la que se le esta asignando el rol |
| model\_id | int(10) | id de la entidad a la que se hace referencia |

|  |  |  |
| --- | --- | --- |
| **oauth\_access\_tokens** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| user\_id | int(11) | id del usuario |
| client\_id | int(10) | id del cliente |
| name | varchar(255) | nombre del cliente |
| scopes | text | ámbitos a los que pertenece el cliente |
| revoked | tinyint(1) | bandera que indica si el cliente es rechazado o no |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |
| expires\_at | datetime | fecha de expiración del token |

|  |  |  |
| --- | --- | --- |
| **oauth\_auth\_codes** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| user\_id | int(11) | id del usuario |
| client\_id | int(10) | id del cliente |
| scopes | text | ámbitos a los que pertenece el cliente |
| revoked | tinyint(1) | bandera que indica si el cliente es rechazado o no |
| expires\_at | datetime | fecha de expiración del token |

|  |  |  |  |
| --- | --- | --- | --- |
| **oauth\_clients** | | | |
| **Campo** | | **Tipo (longitud)** | **Descripción** |
| id | | int(10) | campo llave primaria auto incrementable |
| user\_id | | int(11) | id del usuario |
| name | | varchar(255) | nombre del cliente |
| personal\_access\_client | | tinyint(1) | bandera que indica si el cliente es personal o no |
| password\_client | | tinyint(1) | bandera que indica si es un password client |
| revoked | | tinyint(1) | bandera que indica si el cliente es rechazado o no |
| created\_at | | timestamp | fecha de creación |
| updated\_at | | timestamp | fecha de actualización |
| expires\_at | | datetime | fecha de expiración del token |
| **oauth\_personal\_access\_client** | | | |
| **Campo** | **Tipo (longitud)** | | **Descripción** |
| id | int(10) | | campo llave primaria auto incrementable |
| client\_id | int(10) | | id del cliente |
| created\_at | timestamp | | fecha de creación |
| updated\_at | timestamp | | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **oauth\_refresh\_tokens** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| access\_token\_id | int(11) | id de la llave de acceso |
| revoked | tinyint(1) | bandera que indica si el cliente es rechazado o no |
| expires\_at | datetime | fecha de expiración del token |

|  |  |  |
| --- | --- | --- |
| **password\_resets** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| email | varchar(255) | correo electrónico del usuario que intenta recuperar su contraseña |
| token | varchar(255) | token de recuperación de contraseña |
| created\_at | timestamp | fecha de creación |
| **permissions** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| name | varchar(255) | nombre del permiso |
| guard\_name | varchar(255) | alias del permiso |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **role\_has\_permissions** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| permission\_id | int(10) | permiso al que se hace referencia |
| role\_id | int(10) | Rol al que se hace referencia |

|  |  |  |
| --- | --- | --- |
| **roles** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| name | varchar(255) | nombre del rol |
| alias | varchar(255) | alias del rol |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

|  |  |  |
| --- | --- | --- |
| **service\_ranges** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| km | int(11) | numero de kilómetros que abarca el rango |
| description | varchar(255) | descripción del rango de trabajo |

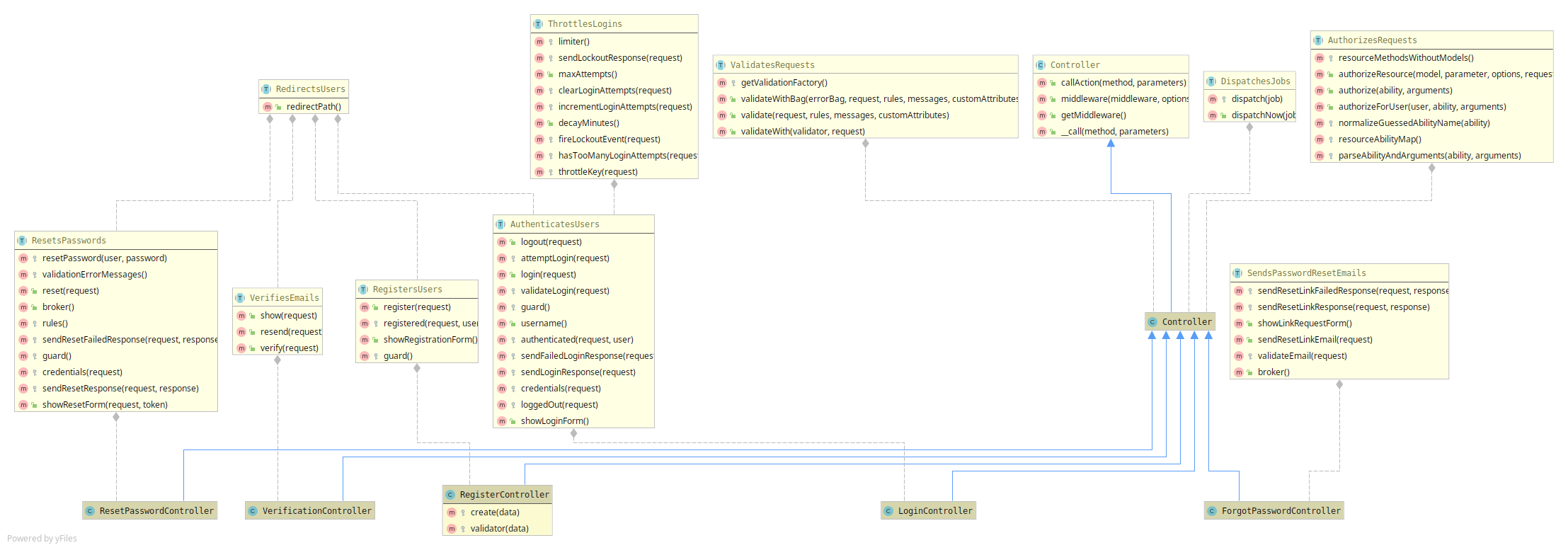
|  |  |  |
| --- | --- | --- |
| **users** | | |
| **Campo** | **Tipo (longitud)** | **Descripción** |
| id | int(10) | campo llave primaria auto incrementable |
| first\_name | varchar(255) | nombre del usuario |
| last\_name | varchar(255) | apellido del usuario |
| email | varchar(255) | correo electrónico del usuario |
| phone | varchar(255) | teléfono del usuario |
| password | varchar(255) | contraseña del usuario |
| location\_id | int(10) | id de la localización del usuario |
| status | tinyint(1) | bandera que indica si el usuario esta activo o no |
| birth\_date | date | fecha de nacimiento del usuario |
| remember\_token | varchar(100) | Campo que guarda el token del usuario durante la sesión |
| created\_at | timestamp | fecha de creación |
| updated\_at | timestamp | fecha de actualización |

1. Diagramas de clases del sistema

A continuación se procederá a mostrar el diseño de las clases utilizadas en el sistema. Cabe mencionar que, ya que el diseño del proyecto se hizo mediante un framework, solo mostraremos las clases necesarias para entender el funcionamiento y comportamiento del sistema. De otra forma tendríamos que mostrar el diagrama de clases de todo el framework, lo que estaría completamente fuera del ámbito de este documento.

* 1. Controladores de Autenticación

Uno de los módulos usados por nuestro sistema, es el módulo de autenticación de usuarios, que puede ser descrito con la jerarquía de clases mostrada en la siguientes imágenes

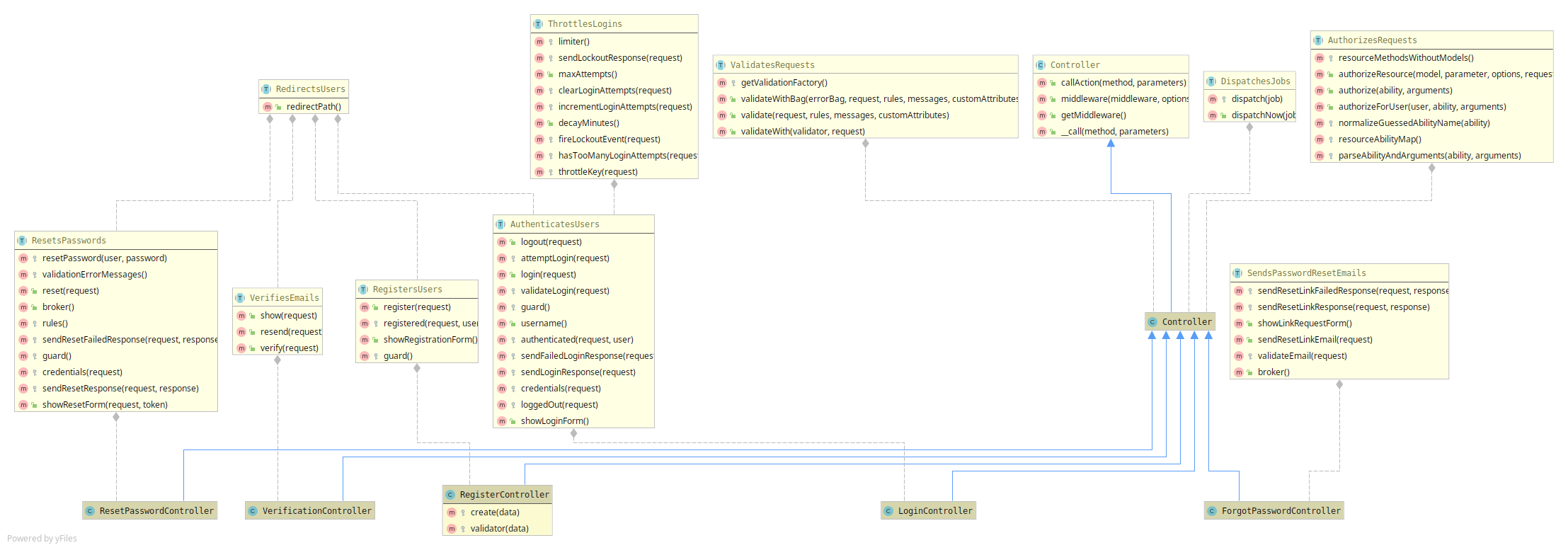


En la imagen de la parte superior, podemos ver 3 clases que modelan:

* **Controller:** el **controlador** base del cual todos los controladores van a extender para heredar sus propiedades y atributos. Vemos que esta clase implementa la interfaz **DispatchesJobs, AuthorizesRequests** y **ValidatesRequests**.
* **LoginController:** el controlador que maneja las peticiones correspondientes al inicio de sesión de usuarios dentro del sistema. Esta clase implementa la interfaz **AuthenticateUsers.**
* **ForgotPasswordController:** esta clase se encarga de manejar las peticiones correspondientes al olvido de contraseña por parte del usuario. Podemos ver que implementa además, la interfaz **SendsPasswordResetEmails**, que nos obliga a implementar un conjunto de métodos que definen como la clases va a responder cuando estos eventos sean llamados.

Tenemos también 4 interfaces, que representan diferentes acciones que las clases pueden implementar a su conveniencia. De esta manera es fácil mantener clases que estén abiertas a la extensión pero cerradas a la modificación. Las interfaces usadas en la imagen de arriba son:

* **SendsPasswordResetEmails:** las clases que implementan esta interfaz están comprometidas a definir todos los métodos esta tiene. Podemos ver que todos esos métodos corresponden a operaciones que se tiene que hacer cuando se le envía un correo a un usuario con un enlace para que este sea capaz de **restablecer su contraseña.**
* **AuthorizeRequests:** podemos observar que este conjunto de métodos, modelan lo que una clase que implementa esta interfaz, tiene que hacer para autorizar validar solicitudes para obtener algún tipo de información de nuestro sistema. Evidentemente la clase **Controller** tiene que implementar todos estos métodos, ya que, los controladores están pensados para encargarse de todo lo que tenga que ver con solicitudes de usuario.
* **DispatchesJobs:** este interfaz, como parte del núcleo de **Laravel**, tiene un uso fuera del ámbito de este sistema, sin embargo se sabe que todos los controladores tienen que implementarla.
* **Controller:** modela las acciones referentes a **middlewares,** los cuales son un mecanismo para averiguar si el usuario que intenta realizar una acción, cuenta con ciertos requisitos previos.
* **ValidatesRequests:** interfaz que se encarga de modelar las acciones que se necesitan hacer para validar que la información entrante de los usuarios, cumpla con un formato adecuado para poder ser manipulada.



En la imagen de la parte superior, se sigue mostrando más sobre las clases e interfaces que se tiene para la autenticación del usuario.

Se tienen ahora 3 clases, las cuales son:

* **RegisterController:** controlador donde se atienden las solicitudes correspondientes al registro de usuarios en el sistema. Esta clase hereda de **Controller** e implementa la interfaz **RegistersUsers**.
* **ResetPasswordController:** esta clase también extiende de **Controller** y sirve para atender solicitudes referentes al restablecimiento de contraseñas en el sistema. Esta clase implementa la interfaz **ResetsPassword.**
* **VerificationController:** este controlador viene integrado en el código de **Laravel,** y por la impletanción de la interfaz **VerifiesEmails,** podemos ver que este controlador realiza realiza funciones que tienen que ver con el correcto envío y verificación de correos electrónicos.

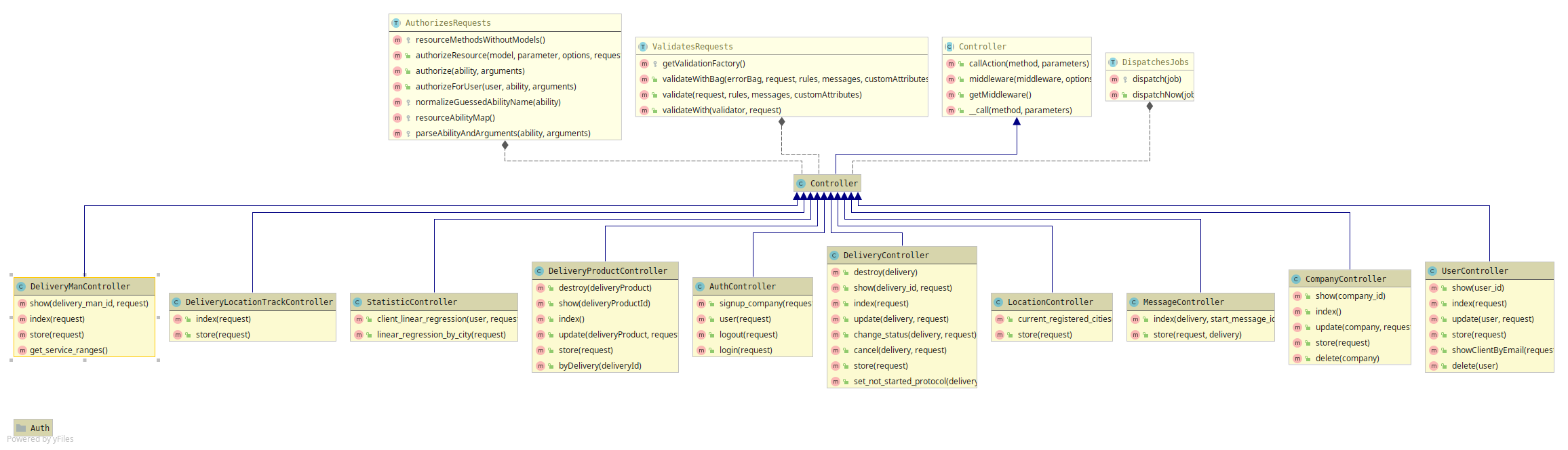
Se tienen además, en esta parte diversas interfaces que los controladores usan. Estas interfaces son:

* **RedirectsUsers:** las clases que implementan esta interfaz saben de qué manera tiene que actuar cuando haya que redireccionar a los usuarios en caso de éxito o fracaso en sus solicitudes.
* **ThrottlesLogins:** las clases que implementan esta interfaz saben cómo comportarse cuando suceda algún comportamiento especial al momento de hacer un inicio de sesión dentro del sistema. Un ejemplo de un comportamiento especial podría ser cuando el usuario intenta iniciar sesión de manera fallida más de las veces permitidas.
* **ResetsPasswords:** esta interfaz representa el conjunto de funciones que una clase tiene que implementar manejar el restablecimiento de contraseñas dentro del sistema.
* **VerifiesEmail:** las clases que implementan esta interfaz, están obligadas a implementar una función para verificar que los correos se hayan enviado de manera exitosa y otra función para reenviar el correo, en caso de que la verificación no haya sido exitosa
* **RegistersUsers:** esta interfaz declara las funciones que tienen que ser implementadas para registrar usuarios dentro del sistema.
* **AuthenticatesUsers:** esta interfaz declara el conjunto de funciones necesarias para realizar el proceso de autenticación de un usuario dentro del sistema. El proceso de autenticación incluye guardar las variables de sesión necesarias para tener al usuario activo dentro del sistema durante el tiempo que nosotros hayamos especificado en la configuración del framework.
  1. Controladores del núcleo del sistema

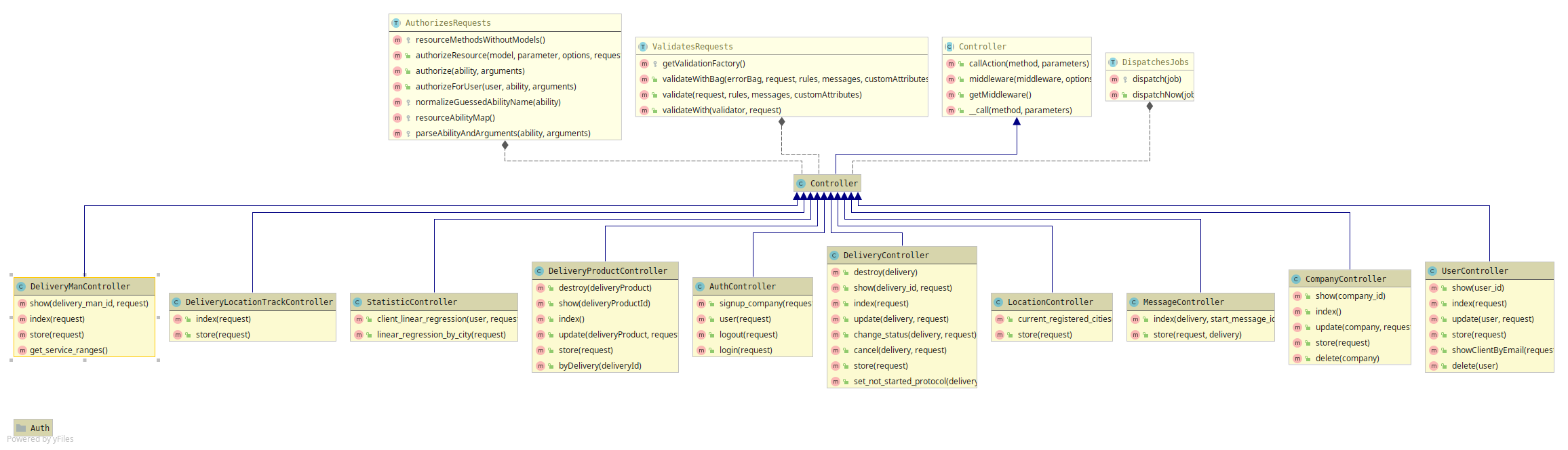
A continuación se presentara el diagrama de clases que hacen que el **onMyWay** cumpla su principal funcionalidad.

En esta sección, se observará desde el nombre de los archivos, que estas clases son mas acordes a la problemática que **onMyWay** resuelve.

Todos los controladores extienden de la clase **Controller** y esta clase a su vez, implementa las interfaces: **AuthorizeRequests, ValidatesRequests, Controller** y **DispatchesJobs** que ya fueron explicadas con anterioridad.



* **AuthController:** las funciones de este controlador no corresponden al núcleo de **Laravel,** fueron codificadas como parte del desarrollo de este proyecto, y lo que hacen básicamente es registrar usuarios y compañías, manejar su inicio y salida de sesión. La diferencia entre este controlador y todas las demás entidades que manejan autenticación es que, aquí se trabaja directamente con clases del núcleo del sistema, es decir, clases como **Deliveries, Locations, Users,** etc.
* **DeliveryController:** este controlador recibe todas las peticiones que corresponden al módulo de entregas de **onMyWay.**
* **LocationController:** este controlador recibe todas las peticiones que tiene que ver con registro y manejo de localizaciones en el sistema.
* **MesageController:** esta clase maneja las peticiones concernientes a mensajes de texto entre el cliente que recibe o envía una entrega con el administrador del sistema.
* **CompanyController:** esta clase maneja las solicitudes que el usuario realiza para poder extraer información de sus compañías, o para hacer operaciones como inserción actualización y borrado de datos en las mismas.
* **UserController:** siendo uno de los controladores mas extensos, este maneja las peticiones que se hacen para guardar, consultar, editar y eliminar la información de los usuarios del sistema.



* **DeliveryManController:** aquí se manejan las solicitudes del usuario de la aplicación a todo lo que tenga que ver con los repartidores. Este controlador realiza tareas básicas como mostrar, crear, editar y borrar un repartidor del sistema.
* **DeliveryLocationTrackController:** como se observa en la última imagen adjunta, este controlador solo tiene dos métodos, los cuales sirven para mostrar todas las localizaciones por las que un paquete en específico ha pasado y para guardar un punto en donde el paquete se encuentra actualmente.
* **StatisticsController:** este controlador maneja peticiones que tengas que ver con calculos de **regresión lineal** en el sistema. Se observa en la última imagen adjunta que se tiene disponibles los métodos para calcular la regresión lineal de las entregas del cliente y la regresión lineal de entregas por municipio.
  1. Clases de tipo servicio

Otra parte muy importante de **onMyWay** son las clases de tipo **servicio** que implementaron.

Decidimos enfocar nuestras clases de tipo **servicio** como intermediarios entre los **controladores** y los **modelos**.

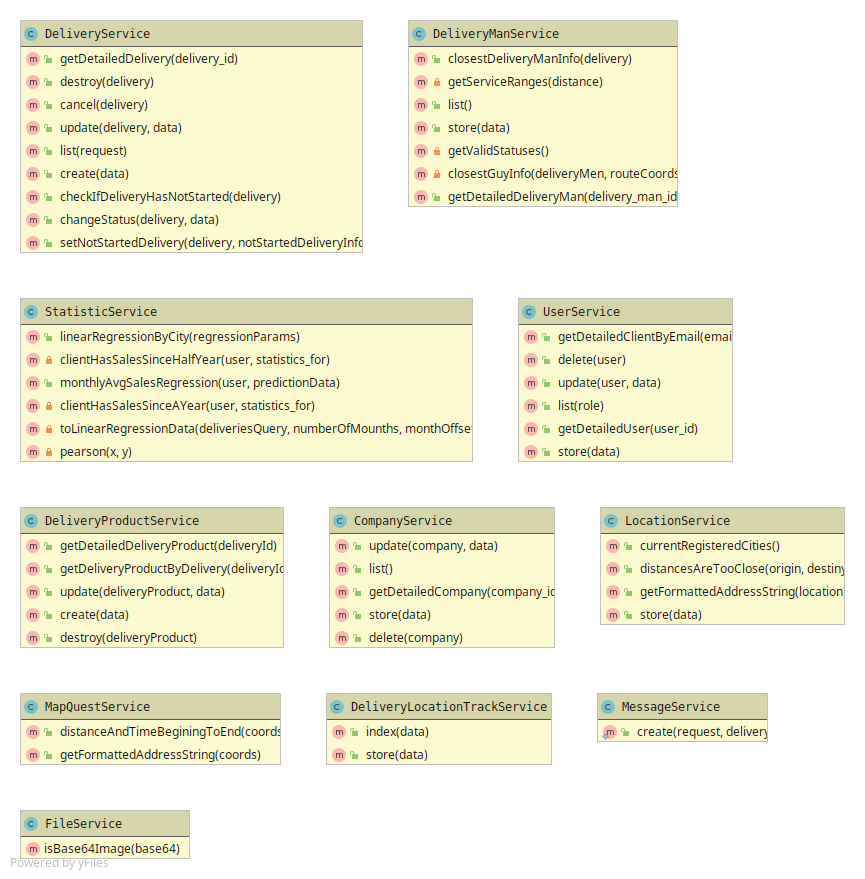
Como equipo, llegamos a la conclusión de que, colocar la lógica del negocio en los controladores, no es una buena opción para tener un buen mantenimiento y mejora futura del proyecto, esto debido a que los controladores no son clases pensadas para ser instanciadas, sino para ser mandadas llamar cuando un usuario realice una petición de una operación al servidor. Por lo que los métodos definidos en el controlador no podrían, o al menos sería incorrecto, usarlos en algún otro lugar.

Sin embargo, también llegamos a la conclusión de que, colocar demasiada lógica dentro de los modelos también representaría una desventaja. Esto debido a que, comúnmente, se dan casos donde es necesario que existan métodos para enviar correo de bienvenida a un usuario (por poner un ejemplo), conectarse con una API, para obtener algún tipo de información, entre otras operaciones parecidas. Como equipo, no creemos que ese tipo de operaciones deberían estar definidos en algún modelo, si fuera así, le estaríamos dando más acoplamiento a nuestros modelos, es decir, le estaríamos dando más responsabilidades de las que realmente le corresponden, y en el momento en que una clase, de las que están incluidas en nuestros modelos, cambia su forma de ser usada, tendríamos que cambiar todas las referencias a esa misma clase en todo el proyecto.

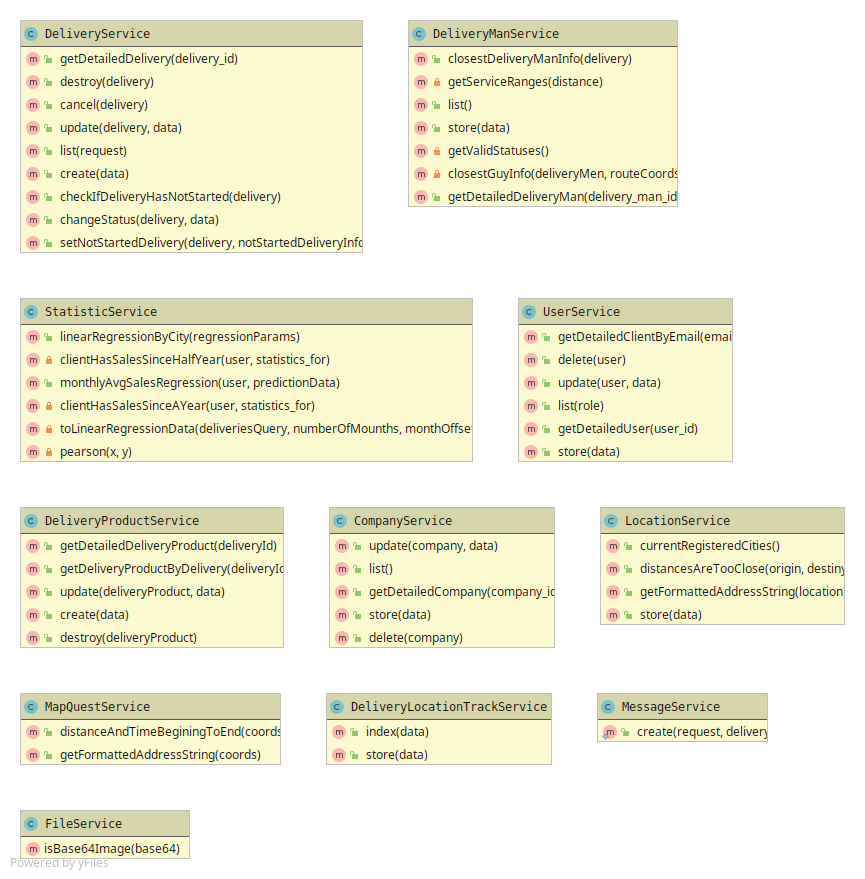
Es por eso que se decidió poner una capa intermedia entre el modelo y el controlador llamado **servicios.**

Esta capa lo que hace es, tomar la información que llega al controlador, información que el usuario mandó desde algún dispositivo, y trabajar con ella y hacer las operaciones necesarias, esta capa sería la ideal para realizar instanciaciones de **modelos** en incluso otros servicios.

Cada controlador del núcleo de **onMyWay** tiene un servicio correspondiente, este servicio se pasa al controlador en tiempo de ejecución por medio de **inyección de dependencias.** Si bien se podría realizar la instanciación de dichos servicios donde sea necesario en el controlador, como equipo llegamos a la conclusión de que el evitar instanciaciones, a la larga trae beneficios debido a que las clases quedan más desacopladas entre sí.



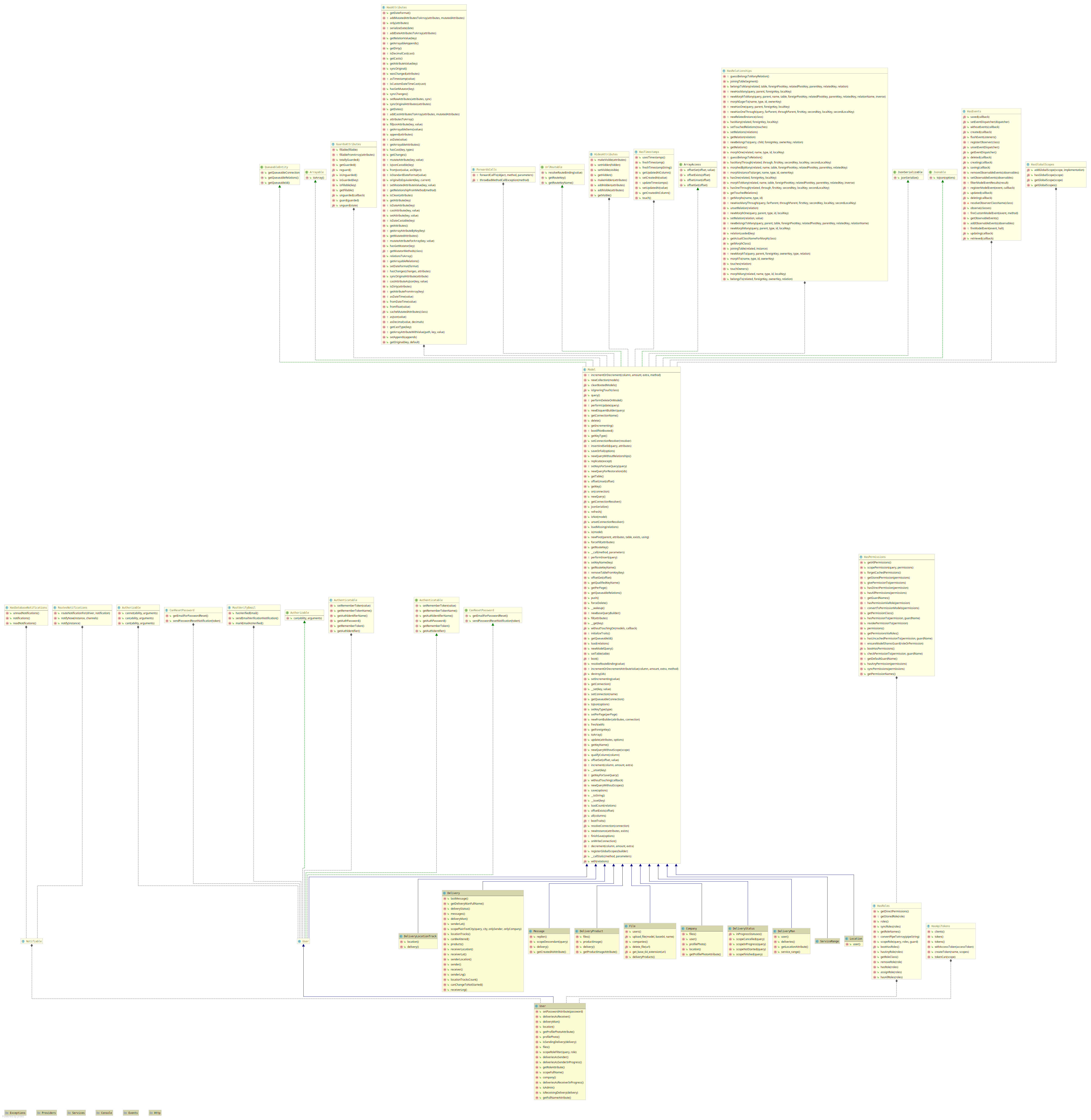
* **DeliveryService:** esta clase está incluida en **DeliveryController** y sirve para realizar la lógica de las operaciones que realiza en controlador de entregas. Recordemos que el controlador solo recibe y envía información de vuelta al usuario, el servicio en este caso es quien opera con la información que el controlador recibe.
* **DeliveryManService:** esta clase realiza la lógica de negocio que tiene que ver con repartidores en el sistema, este servicio realiza operaciones importantes como buscar un repartidor disponible para una entrega.
* **StatisticService:** este servicio realiza las regresiones lineales con los parámetros que el usuario nos haya mandado.
* **UserService:** este servicio realiza la lógica de las operaciones que tienen que ver con administrar usuarios en el sistema.



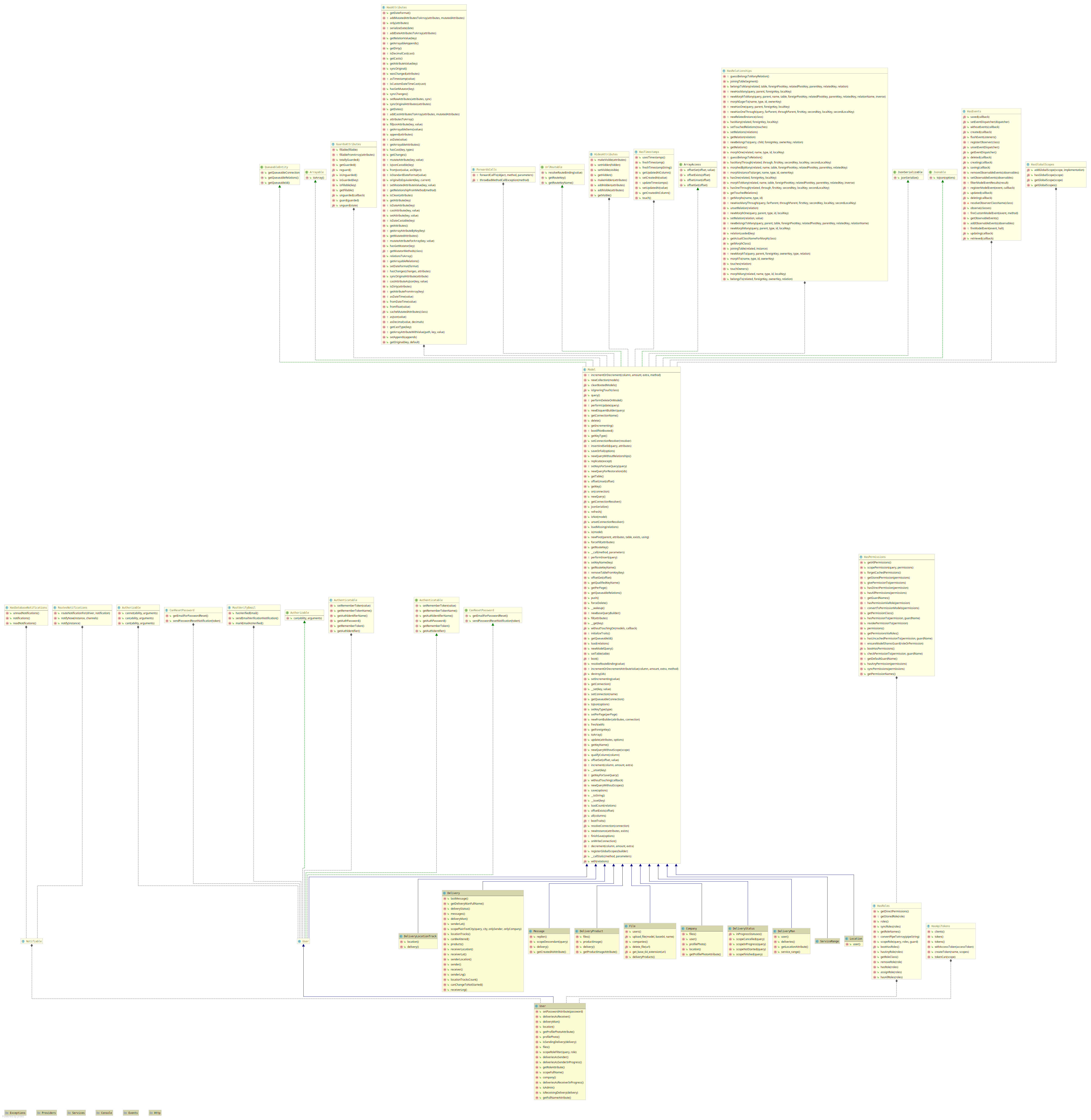
* **DeliveryProductService:** este servicio realiza la lógica de negocio que tiene que ver con los productos de una entrega.
* **CompanyService:** la información de las compañías que pasan por el sistema, es manejada en esta clase.
* **LocationService:** este servicio realiza la lógica de negocio que tiene que ver con las localizaciones, ya sea del usuario, de la compañía o de los lugares por los que un paquete ha pasado.
* **MapQuestService:** este servicio realiza la lógica de negocio que tiene que ver con peticiones a la API de **Mapquest,** una API que nos brinda información de geolocalización en distintos formatos.
* **DeliveryLocationTrackService:** este servicio realiza la lógica de negocio que tiene que ver con los productos de una entrega.
* **MessageService:** este servicio realiza la lógica de negocio que tiene que ver con los productos de una entrega.
* **FileService:** este se encarga del almacenamiento de archivos en el sistema.
  1. Modelos del sistema

Los modelos son clases que abstraen entidades concretas de la base de datos.

A continuación, se mostrará un diagrama de los modelos que se necesitan en **onMyWay.**

****

* **DeliveryLocationTrack:** representa una ubicación en la que el producto se encuentra en un momento específico.
* **Delivery:** representa un viaje, en cualquier tipo de estatus, tanto si ya ha sido enviado, recibido, cancelado, etc.
* **Message:** representa un mensaje de texto, con su respectivo cuerpo de mensaje, emisor y receptor.
* **DeliveryProduct:** representa un producto relacionado a una entrega.
* **User:** representa un usuario, de cualquier rol.

****

* **File:** representa cualquier tipo de archivo guardado en el sistema.
* **Company:** representa y abstrae una compañía registrada.
* **DeliveryStatus:** representa posibles estados que puede tener la entrega.
* **DeliveryMan:** representa un repartidor dentro del sistema.
* **ServiceRange:** representa un rango de trabajo, que el repartidor puede escoger para trabajar a la redonda el número de kilómetros seleccionado.
* **Location:** representa una localización, que pueden ser localizaciones donde el cliente o e repartidor vivan, o localizaciones donde se encuentra una entrega.

CONCLUSIONES

La aplicación **onMyWay** brinda un servicio (el servicio de paquetería) de manera más cómoda que la forma tradicional de realizarlo. Además estar desarrollado bajo una arquitectura que hace posible su crecimiento y el agregamiento de nuevos módulos y funcionalidades.

Al hacer un balance entre las empresas de paquetería tradicionales, o las aplicaciones para hacer mandados dentro del mismo municipio que existen ya que en mercado, el cliente se sentiría mas cómodo usando **onMyWay**, principalmente por tener todo lo que este necesita dentro de la misma plataforma. Cualquier tipo de intermediario entre el cliente y el servicio que se está realizando es invisible al cliente. El cliente solo vería la aplicación, los datos que esta informa y si tuviera algún problema, por medio de la aplicación podría quejarse o comunicarse con soporte.

**OnMyWay** es un proyecto rentable y que puede tener un seguimiento futuro. A largo plazo se pueden pensar en otro tipo de medidas que pueden ser implementadas para mejorar la experiencia del usuario en la aplicación. Estas medidas podrían incluir, el hecho de invertir en algún tipo de vehículo especial para los paquetes que viajan distancias grandes, invertir en campañas publicitarias para mejorar la cartera de clientes, etc.

El hecho de que el servicio pueda ser calificado directamente por el cliente tiene como consecuencia que el usuario pueda basarse en el historial de entregas, comentarios y sugerencias para decidir si este se siente seguro de usar o no la aplicación. Al tener la aplicación buenas críticas y un historial de entregas satisfactorias, el usuario se sentirá mas atraído a usar la aplicación y sentirse menos desconfiado respecto a las desventajas de la modernización de un servicio como este puede traer.

REFERENCIAS BIBLIOGRÁFICAS

Laravel (2018). Controllers. 30 de Mayo del 2019, de Taylor Otwell. Sitio web: <https://laravel.com/docs/5.8/controllers>

Laravel (2018). Passport. 30 de Mayo del 2019, de Taylor Otwell. Sitio web: <https://laravel.com/docs/5.8/passport>

RESTFULAPI (2017) REST. 30 de Mayo del 2019, de RESTFULAPI. Sitio web: <https://restfulapi.net/>

IONIC. Ionic Framework. 30 de Mayo del 2019, de Ionic Framework. Sitio web: <https://ionicframework.com/>

IBM. Model-View-Controller design pattern. 30 de Mayo del 2019, de IBM. Sitio web: <https://www.ibm.com/support/knowledgecenter/en/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm>

Github. Spatie/laravel-permission. 30 de Mayo del 2019, de Spatie. Sitio web: <https://github.com/spatie/laravel-permission>