PROYECTO FIN DE CICLO - DAM

Gestión integral: Envío de paquetería

Fase 2: Diseño del proyecto

Rolando Caldas Sánchez

Índice

Índice de contenido

Gestión integral: Envío de paquetería - DAM

1. Identificación de las necesidades del proyecto

1.1 Contextualización

El proyecto a realizar se enmarca dentro de las necesidades específicas del sector de mensajería urgente. La realización del mismo nace como fruto de la necesidad interna de una empresa de mensajería de optimizar y actualizar la gestión de sus recursos.

Tras observar el funcionamiento operativo de otras empresas del sector como MRW, Seur o Correos se detectaron múltiples mejoras a realizar en el servicio y carencias que los servicios de las empresas de referencia tienen y pueden ser solucionados.

El objetivo que cubre el proyecto a realizar es poder llevar con éxito el paso de una gestión en papel del proceso de mensajería a una gestión digital y en tiempo real. Con esta demanda de mejora, se busca mejorar tanto los tiempos de la gestión como el servicio post-venta ofrecido a los clientes, pudiendo darles información en tiempo real del estado de su encargo.

La empresa analizó la situación y la conveniencia o no de solicitar a una empresa del sector informático la realización del proyecto. Esta opción fue descartada, tomando la decisión estratégica de incorporar un departamento de desarrollo como parte activa de la organización, para la realización de la solución y su mantenimiento futuro.

Para la toma de las decisiones estratégicas se ha realizado un pequeño estudio de mercado intentando localizar las soluciones existentes en el mercado, ya sea para realizar una implantación directa o un mejora sobre un sistema ya existente. Durante la búsqueda de soluciones se tomó en consideración valorar las siguientes soluciones:

- **Paquetería.One**
- **Gestión5 SQL: Paquetería y Logística**
- **Recawin**
- **JR Gestión de paquetería**

Las soluciones analizadas tuvieron que ser descartadas debido a la inconveniencia de ser implementadas. Actualmente, los servicios de mensajería estándar buscan la unificación de tarifas y servicio, de forma que se aplica una tarifa por bulto a sus clientes, en lugar de precios diferentes en función de tamaños y pesos. Siguiendo esta tendencia, la empresa de mensajería tiene unos pesos y tamaños máximos por bulto, de manera que si un paquete excede esos tamaños no puede ser entregada por la empresa. Del mismo modo, el precio por paquete es único, para conseguir ser más eficientes y rápidos en el servicio de atención al cliente y para simplificar el proceso de contratación del servicio.

Esta necesidad choca frontalmente con las soluciones estudiadas, puesto que éstas designan gran parte de los recursos de la solución a la gestión de tarifas.

Otra carencia de las soluciones analizadas es la falta de una solución multiplataforma y de movilidad, que permita a los transportistas informar en tiempo real y de forma clara, el estado de los pedidos. Además, las soluciones se basan en entorno de escritorio y Windows, algo que limita el cambio de software en la empresa de mensajería. Aunque esto podría interpretarse como requisitos de la solución y no como un problema, los problemas existentes con la privacidad de la información en la última versión del sistema operativo Windows, hace que la empresa no quiera una solución que fuere una plataforma concreta.

Además de estas soluciones, se descartó intentar implementar las soluciones de otras empresas líderes del sector como MRW o Seur debido a que la solución de movilidad está supeditada a unos dispositivos concretos, algo que se pretende evitar puesto que se quiere tener la libertad de poder actualizar los dispositivos de forma sencilla.

1.2 Justificación

Tras analizar las posibles soluciones del mercado, se toma la decisión estratégica de desarrollar la solución de manera interna. Esta solución se soporta en los siguientes pilares:

- La solución a realizar tiene que estar pensado y basado en la movilidad.
- La solución debe conformarse por diferentes desarrollos de software que estén comunicados entre sí en tiempo real.
- La solución debe permitir eliminar todo soporte en papel para la realización del servicio.
- La solución no debe realizar labores de facturación, puesto que para ello ya existe un ERP.
- La solución debe utilizar tecnologías libres exclusivamente.

Durante el proceso de análisis del problema y la solución, surgió la necesidad de decidir si el proyecto a realizar quedaría como un desarrollo interno o si la solución debería plantearse como un software a comercializar. La decisión tomada en este sentido fue la de realizar la solución como un desarrollo interno no comercializable en primera instancia. Esta decisión se toma principalmente escuchando al departamento de marketing, quien considera que el software a desarrollar puede ser utilizado para obtener una ventaja sobre la competencia, utilizándola como una herramienta diferenciadora de venta del servicio. A pesar de ello, como la competencia es intensa y las mejoras son continuas, una vez la solución esté implementada y sea un punto diferenciador del servicio, se creará una marca comercial para el software, para poder comercializarlo como producto de software a empresas del sector que trabajen en provincias diferentes a la de la empresa de transporte.

1.3 Otros aspectos

Como el desarrollo va a realizarse dentro de la empresa de transporte, no se creará ninguna empresa para tal fin. Lo que sí se procederá es a crear un departamento de desarrollo conformado por cuatro desarrolladores de diferentes perfiles:

- 1 Jefe de proyecto
- 1 Analista programador
- 1 Programador backend senior
- 1 Programador frontend junior

Para poder formar el departamento, la empresa modificará sus datos de actividad económica incorporando la de “Actividades de consultoría informática”, estando los cuatro desarrolladores contratados bajo el convenio de empresas de consultoría y estudio de mercados y la opinión. Esta decisión se toma debido a que no sólo se realizará el desarrollo de software sino que para poder realizar el desarrollo tendrán que ejercer también como consultores informáticos, especialmente cuando el producto llegue a su fase de comercialización.

Para el desarrollo del proyecto se solicitarán ayudas públicas para proyectos de I+D+i a través del IGAPE (Instituto Gallego de Promoción Económica) y los contratos a los desarrolladores serán indefinidos celebrados bajo la cláusula específica de apoyo a los emprendedores.

Los nuevos desarrolladores pasarán a trabajar en la nave de la empresa de transportes, habilitando para ello una sala en la nave, dotada de ventilación, calefacción, servicio propio y dispensador de agua.

Se comprarán cuatro mesas de trabajo, con sus correspondientes sillas, todas con un plus de ergonomía, especialmente importante debido a la cantidad de horas que los nuevos empleados pasarán en el trabajo.

La empresa comprará nuevos equipos informáticos, uno para cada empleado, con las siguientes características:

- Procesador i7-4790K 4.0Ghz
- Memoria G.Skill Trident X DDR3 2400 PC3-19200 32GB 4x8GB CL10
- Disco duro Samsung 850 Pro SSD Series 1TB
- GeForce GTX 980 4GB DDR3
- Teclado y ratón Microsoft Sculpt Ergonomic Desktop
- Monitor con tecnología Anti-Blue Light y Flicker Free

La motivación de las características del equipo es doble: Por un lado dotar al empleado de los recursos necesarios para hacer funcionar con fluidez las herramientas de desarrollo y, por otro, incorporar soluciones ergonómicas teniendo en cuenta los riesgos laborables.

Para llevar a cabo el desarrollo del proyecto, se cuenta con un guión base sobre el cual poder trabajar:

1. Seleccionar los profesionales encargados de desarrollar la solución.
2. Con el equipo formado, analizar el proyecto con el jefe de proyecto y el analista programador, para realizar los ajustes necesarios.
3. Establecer el cronograma de trabajo, desde la fase de análisis.
4. Realizar el análisis y diseño del proyecto.
5. Iniciar la ejecución de proyecto.
6. Fase de pruebas.
7. Presentación del software: demo.
8. Implementar el software con dos transportistas: Experiencia piloto.
9. Realizar las correcciones necesarias tras el resultado de la experiencia piloto.
10. Implementación de la solución a gran escala.

El presente documento plasmará las fases de diseño, ejecución y seguimiento y control, quedando el resto fuera del alcance del presente proyecto.

2. Diseño del proyecto

2.1 Contenido

Teniendo claro el proyecto que quiere realizarse debemos comprobar si es viable o no desde el punto de vista técnico.

Actualmente los sistemas de información han evolucionado rápidamente hacia una interconexión global. Además, la red móvil tiene una cobertura 3G casi absoluta en las grandes ciudades. Esto hace que, la solución objetivo, pueda ser puesto en funcionamiento.

La existencia de lenguajes de alto nivel y de profesionales con dilatada experiencia en dichos lenguajes hace que desde el punto de vista humano no exista problemas en poder realizar el proyecto. El mayor problema se encontraría en la parte de movilidad del proyecto, pudiendo llegar a ser un problema crítico e insalvable en el supuesto de que la aplicación móvil tuviese que desarrollarse para varias plataformas.

Esto hace que se tenga que analizar con cuidado el desarrollo de la aplicación móvil. Afortunadamente existen varios proyectos de software que pretenden llevar el concepto de multiplataforma a los dispositivos móviles tales como Cordova, PhoneGap o Appcelerator entre otros.

Estas soluciones son una puerta de esperanza que hacen viables múltiples proyectos. En el caso que nos ocupa, se valora utilizar Cordova para poder desarrollar una aplicación híbrida que pueda ser multiplataforma de manera extremadamente sencilla.

Es preciso comprobar si la aplicación puede desarrollarse en HTML5 y con Cordova. Para ello se tienen que cotejar las funcionalidades deseadas. Las funcionalidades que deben poder cubrirse son:

- Escaneo de códigos QR
- Intercambio de datos vía API
- Geolocalización
- Acceso a mapas
- Notificaciones Push

Aunque actualmente sólo se contempla la funcionalidad de escaneo de códigos QR, el resto de funcionalidades deben ser tenidas en cuenta para una implementación futura en caso de éxito, por lo que la aplicación desarrollada hoy tiene que ser capaz de incorporar todas las funcionalidades indicadas.

Cordova posee plugins que permiten realizar todas las funcionalidades indicadas sin problema.

Por lo tanto, se considera viable llevar a cabo el proyecto.

2.2 Objetivos y recursos

Los objetivos del proyecto son sencillos y claros: Se busca aumentar el activo de la

Los objetivos del proyecto son sencillos y claros: Se busca aumentar el activo de la empresa aportando una solución software que permita, a largo plazo, agilizar a nivel provincial, autonómico e incluso nacional la gestión de los envíos de paquetería.

Como el desarrollo de un aplicación de estas características es largo y repleto de opciones y posibilidades, se centran los esfuerzos en crear una solución que sirva para lanzar una experiencia piloto a nivel local.

Respecto a los recursos, en el apartado 1.3 del presente documento ya se detallan los recursos tanto de software como de personal; por lo que en este apartado contemplaremos los recursos software:

Todo el software a utilizar en el proyecto tiene que ser software libre, abierto o gratuito:

- **Sistema operativo**: Kubuntu
- **Sistema operativo servidores**: Debian
- **IDE**: Netbeans (Para la webapp y app móvil) y LiClipse (API)
- **Navegador Web**: Chromium
- **Máquina Virtual**: Vagrant + VirtualBox
- **Diagramas**: Dia
- **Suite ofimática**: LibreOffice 5

2.3 Viabilidad económica

Como el proyecto es para uso interno y aumento del activo de la empresa, no se cuenta con unas limitaciones claras de presupuesto como en el caso de que su objetivo fuese ser comercializado y lograr rentabilidad económica.

Sin embargo, es preciso limitar económicamente el proyecto tanto para no descuadrar las cuentas de la empresa como para evitar el derroche de recursos.

En base a lo especificado en el primer epígrafe del presente documento, podemos realizar el siguiente presupuesto económico, planteando un ciclo de desarrollo de un año.

Costes capital humano

- 1 Jefe de proyecto: 30.000 EUR
- 1 Analista programador: 28.000 EUR
- 1 Programador backend senior: 24.000 EUR
- 1 Programador frontend junior 15.000 EUR

El coste anual en personal ascendería a 97.000 EUR. Este coste se considera

excesivamente elevado y de difícil justificación para un proyecto interno. Se decide revisar a la baja esta partida, eliminando parte del equipo que se desea incorporar.

Debido a que desde la empresa se ve factible crear una experiencia piloto con una funcionalidad limitada y, gradualmente, ir complementado la solución, se decide unificar la figura de Jefe de proyecto y Analista programador, junto con la substitución del programador junior por un becario a través del FEUGA quedando el coste de capital humano de la siguiente manera:

- 1 Jefe de proyecto / AP: 30.000 EUR
- 1 Programador backend senior: 24.000 EUR
- 1 Becario FEUGA: 4.000 EUR

Quedando el coste de capital humano en 58000 EUR. Este coste no incluye las cuotas a la seguridad social, puesto que la empresa se acoge durante el primer año a una subvención sobre la totalidad de las cuotas a la seguridad social, al contratar a desarrolladores actualmente sin empleo.

Coste de equipo

Servidores web

Por úlitmo, aunque no hay costes asociados en cuanto a software, como se van a desarrollar dos aplicaciones del lado del servidor (API y WebApp), se considera oportuno contratar dos servidores web en uno de los principales centros de datos europeos por un año.

Se valoran las diferentes opciones y se selecciona el servidor de la gama para PYMES de Ovh (soyoustart) E3-SSD-3 que tiene un coste anual de 480 EUR/año cada uno.

Características servidores:

Procesador	Intel Xeon E3 1245v2
Cores/Threads	4 cores/ 8 threads
Frecuencia	3.4 GHz+
RAM	32GB DDR3
Discos	3x 120 GB SSD
RAID	Soft
Conexión de red	1 Gbps
Ancho de banda	250 Mbps
Tráfico	Ilimitado

Presupuesto anual

- Personal: 58.000 EUR
- Equipos: 5.000 EUR
- Servidores: 480 EUR
- Total: 63.480 EUR

Financiación

A pesar de que el presupuesto es ajustado y aceptable, como el proyecto puede encajar dentro de un proyecto de I+D+i, se solicita una ayuda al desarrollo a modo de subvención a través del IGAPE por 50.000 EUR

2.4 Modelo de solución

La realización de la solución conlleva tener que valorar y modelizar las diferentes aplicaciones que, en su conjunto, conforman la solución. Por lo tanto, se han modelizado las siguientes aplicaciones:

1. **API**: Se va a generar una aplicación cuya única finalidad es la de servir de modelo o capa de datos al resto de aplicaciones. Para poder realizar este objetivo de forma eficiente, se va a establecer una interfaz de comunicación siguiendo los principios de una API CRUD
2. **WebApp**: Esta aplicación tiene como finalidad proporcionar una interfaz amigable al personal administrativo que les permita gestionar el proceso de envío de paquetería.
3. **App móvil**: Se va a crear una pequeña aplicación móvil que utilizarán los transportistas, de forma que desde su teléfono móvil se comuniquen en tiempo real con la aplicación API, pudiendo acceder a los envíos pendientes e informar de la entrega exitosa, o no.

Al tratarse de tres aplicaciones diferentes, analizaremos cada solución por separado, empezando por la API puesto que es de la que se sirven tanto la WebApp como la App móvil.

DESARROLLO API

La aplicación que va a funcionar como API es la única que va a almacenar la información del negocio, por lo que es en esta parte de la solución en la que analizaremos el modelo de datos a través de un diagrama E/R

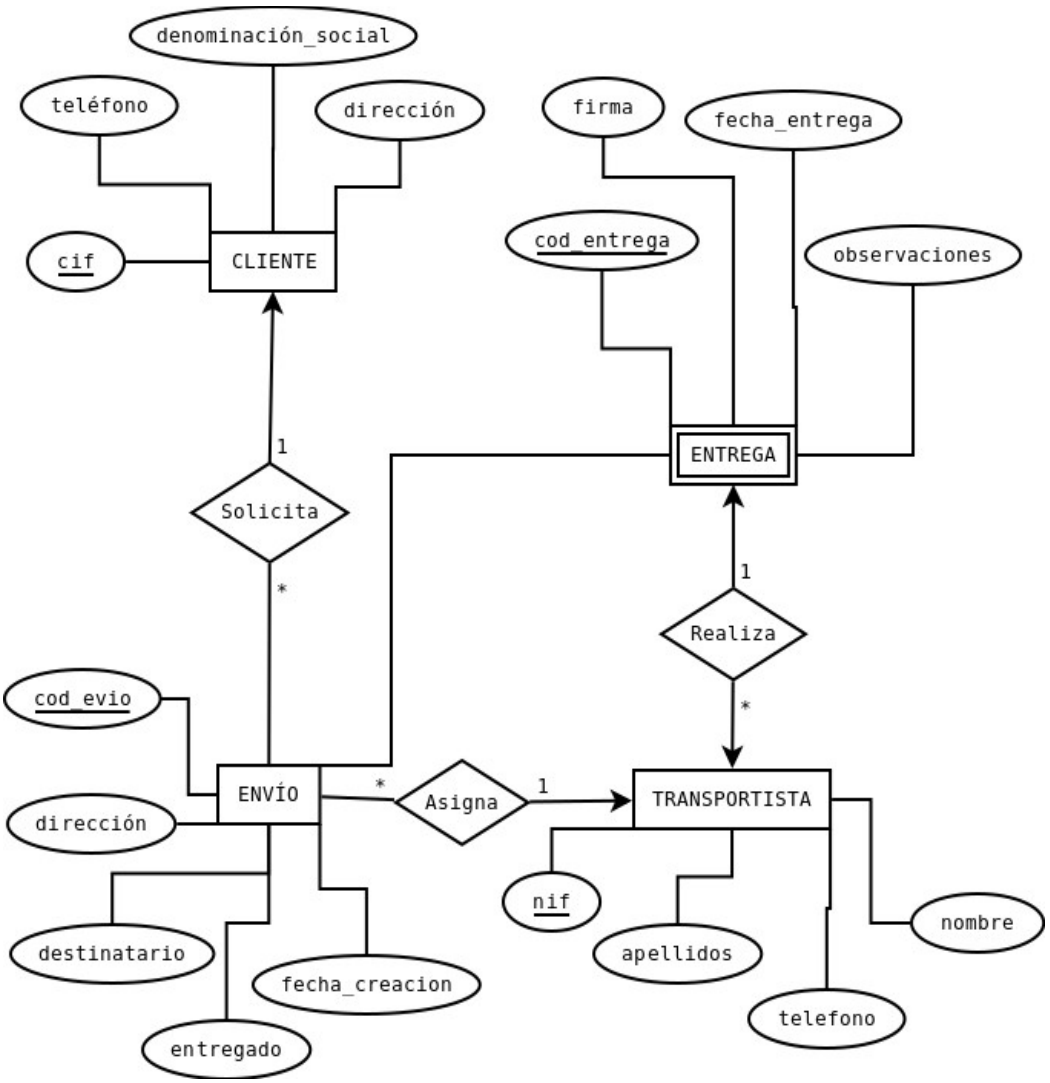
Modelo E/R

Para poder realizar el modelo de E/R se analiza la información con la que trabaja la empresa y se llega a la conclusión de que, para la experiencia piloto, la API debe permitir acceder, crear, editar y modificar la siguiente información:

- Clientes: Todo envío es solicitado por una empresa o autónomo, por lo que debe existir una base de datos de clientes que sea gestionable. Para agilizar el proceso se decide utilizar la información mínima necesaria, puesto que recabar más datos en la API sería duplicar información, puesto que los detalles de cada cliente estarán en la ficha gestionada por el ERP que tiene la empresa.
- Transportistas: Es necesario tener una base de datos de transportistas, puesto que éstos son los encargados de realizar las entregas. Son personal de la empresa.
- Envíos: Los clientes solicitan que se envíe un paquete a un tercero, por lo que los envíos son el eje central de la solución.
- Entrega: Es necesario tener constancia de que el envío se entregó correctamente o de si ha existido alguna incidencia. Se entiende entrega por le proceso de ir a entregar el paquete, por lo que puede darse el caso de que un mismo envío tenga dos entregas, la primera podría ser no exitosa (por lo que

indicaría como observaciones, por ejemplo, que el destinatario no estaba disponible) y la segunda sí (por lo que tendría la firma de recepción del destinatario).

Con las entidades y sus relaciones identificadas de forma general, incluimos el modelo E/R inicial con las entidades, sus atributos y las relaciones existentes entre las diferentes entidades:



CLIENTE (cif, denominción_social, dirección, teléfono)

ENVÍO (cod_envio, destinatario, dirección, fecha_creación, cif, nif)

TRANSPORTISTA(nif, nombre, apellidos, telefono)

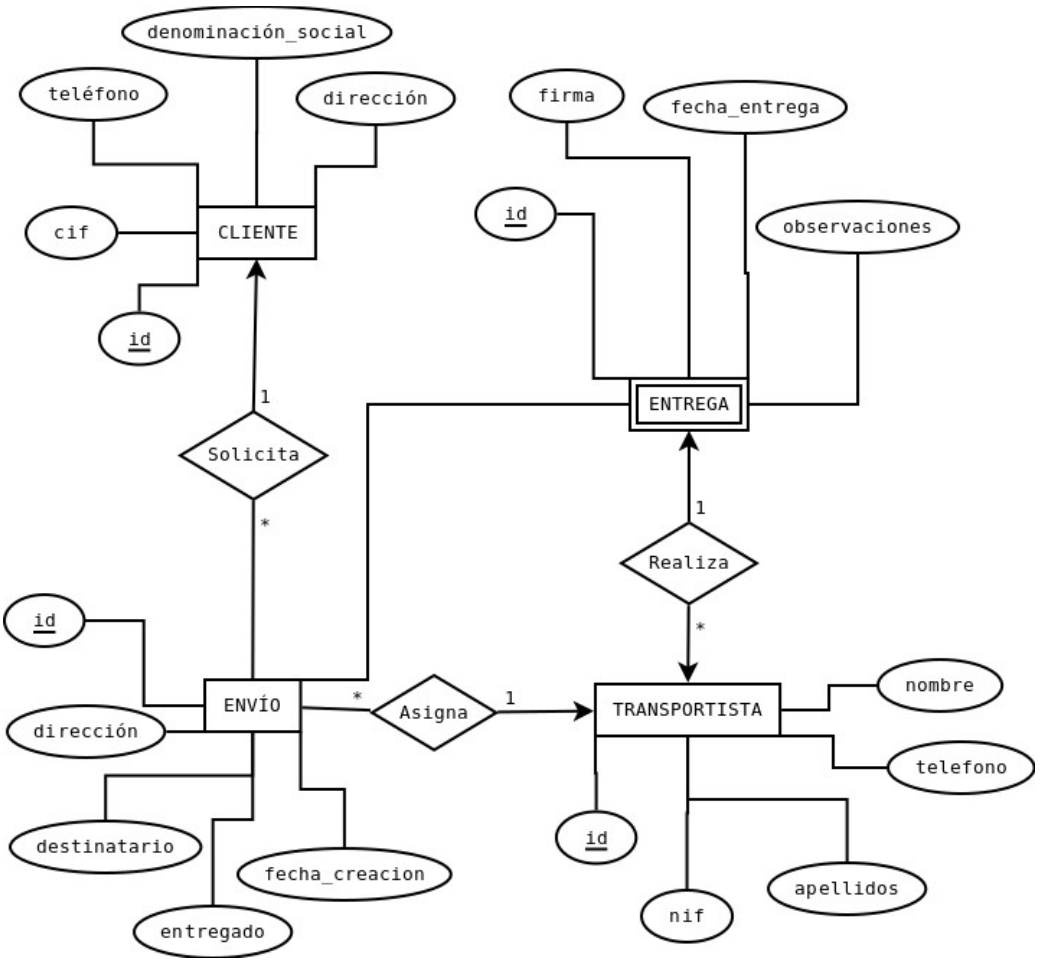
ENTREGA (cod_entrega, cod_envio, firma, fecha_entrega, observaciones, nif)

A pesar de que se considera la solución idónea, este modelo debe ser modificado puesto que se valora una premisa que prevalece sobre la solución resultante del modelo E/R.

Como el tiempo y los recursos son limitados, se tomó como decisión estratégica apoyar todo el desarrollo en diferentes frameworks consolidados que nos permitan avanzar con celeridad. Para realizar la API, se realizó una investigación sobre los diferentes frameworks que ofrecen una solución robusta para una API CRUD. La conclusión fue el framework Django, programado en Python, y en su aplicación “Django REST framework”

Django, de forma nativa, utiliza como SGBD sqLite. Una de las características de este framework es que para cada tabla crea, automáticamente, un campo id auto incremental como clave primaria de la tabla. A pesar de que es posible modificar esta conducta, al hacerlo es necesario encargarse, por código, del valor autoincremental en caso de que la clave primaria deba tener esa condición. Además, las diferentes clases sobre las cuales se asienta el ORM y Django REST Framework asumen la existencia de este campo como clave primaria.

Así pues, se toma la decisión de modificar nuestro modelo E/R para adecuarlo a las premisas que el Django proporciona.



Casos de uso

Con la estructura de la base de datos determinada, se analiza las acciones que desde la API se deben permitir llevar a cabo.

Nuestra aplicación API es una API CRUD, o sea: Create, Read, Update, Delete. Por lo tanto, nuestra API, al menos, debe permitir realizar tales acciones a las diferentes entidades.

Aunque se puede ver en el diagrama de casos de uso, debido a su extensión también los enumeramos a continuación:

- **Entidad cliente**: Obtener clientes, crear cliente, editar cliente, información cliente, borrar cliente.
- **Entidad transportista**: Obtener transportistas, crear transportista, editar transportista, información transportista, borrar transportista.
- **Entidad envío**: Obtener envíos, crear envío, editar envío, información envío, borrar envío.
- **Entidad entrega**: Obtener entregas, crear entrega, editar entrega, información entrega, borrar entrega.

Además de estos casos de uso básicos, para mejorar el rendimiento de las aplicaciones que se conectan a la API se contemplan los siguientes casos de uso:

- **Envíos del Cliente**: Envíos asociados a un cliente concreto.
- **Entregas del envío**: Entregas asociadas a un envío concreto.
- **Envíos asociados al transportista**: Todos los envíos asignados a un transportista.
- **Envíos realizados por el transportista**: Todos los envíos asociados a un transportista que fueron entregados satisfactoriamente.
- **Envíos pendientes del transportista**: Todos los envíos asociados a un transportista que todavía no han podido se entregados.

En todos los casos de uso, se toma por actor la figura “Conector API” que no es más que el modo de denominar al software que solicita a conexión con la API.

La API está protegida, por lo que sólo con un usuario y contraseña existente en la API se puede consultar, por lo que todos los casos de uso incluyen el caso de uso de “Validar acción”.

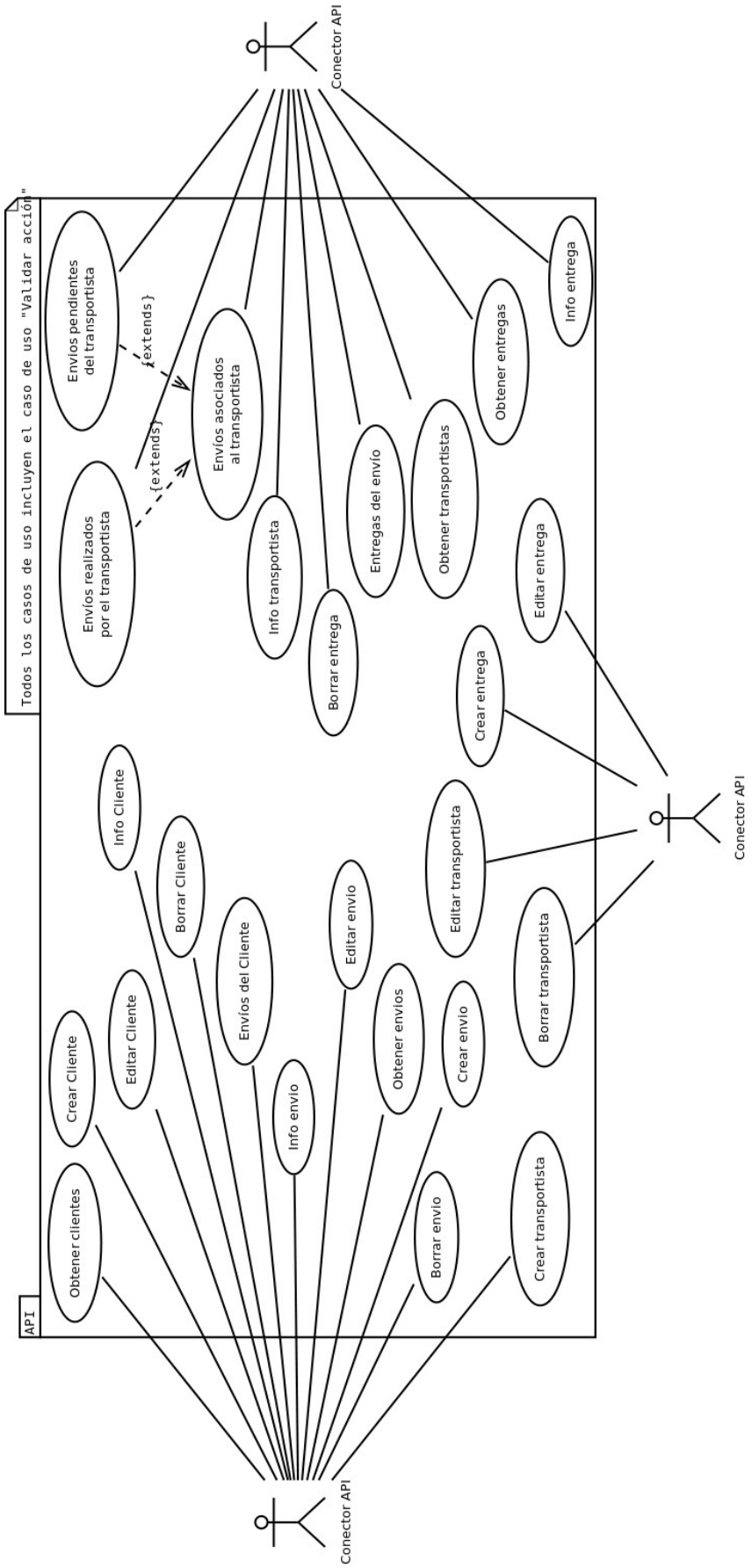

Conector API
Todos los casos de uso incluyen el caso de uso "Validar acción"
Envíos pendientes del transportista
{extends}
Envíos asociados al transportista
Info entrega
Obtener entregas
Envíos realizados por el transportista
{extends}
Info transportista
Entregas del envío
Obtener transportistas
Editar entrega
Borrar entrega
Crear entrega
Info Cliente
Borrar Cliente
Editar transportista
Conector API
Crear Cliente
Editar Cliente
Envíos del Cliente
Editar envío
Borrar transportista
Info envío
Obtener envíos
Crear envío
Obtener clientes
Borrar envío
Crear transportista
API
Conector API

Diagrama UML

Para elaborar el diagrama UML de la API, se estudió tanto Django como su REST Framework para evitar realizar clases y código ya realizado. Django sigue la filosofía DRY (Don't Repeat Yourself). Esto es especialmente importante en el modelo de datos, puesto que Django cuenta con un potente ORM. Gracias a este ORM el código a realizar para crear un modelo de datos se limita a pocas líneas.

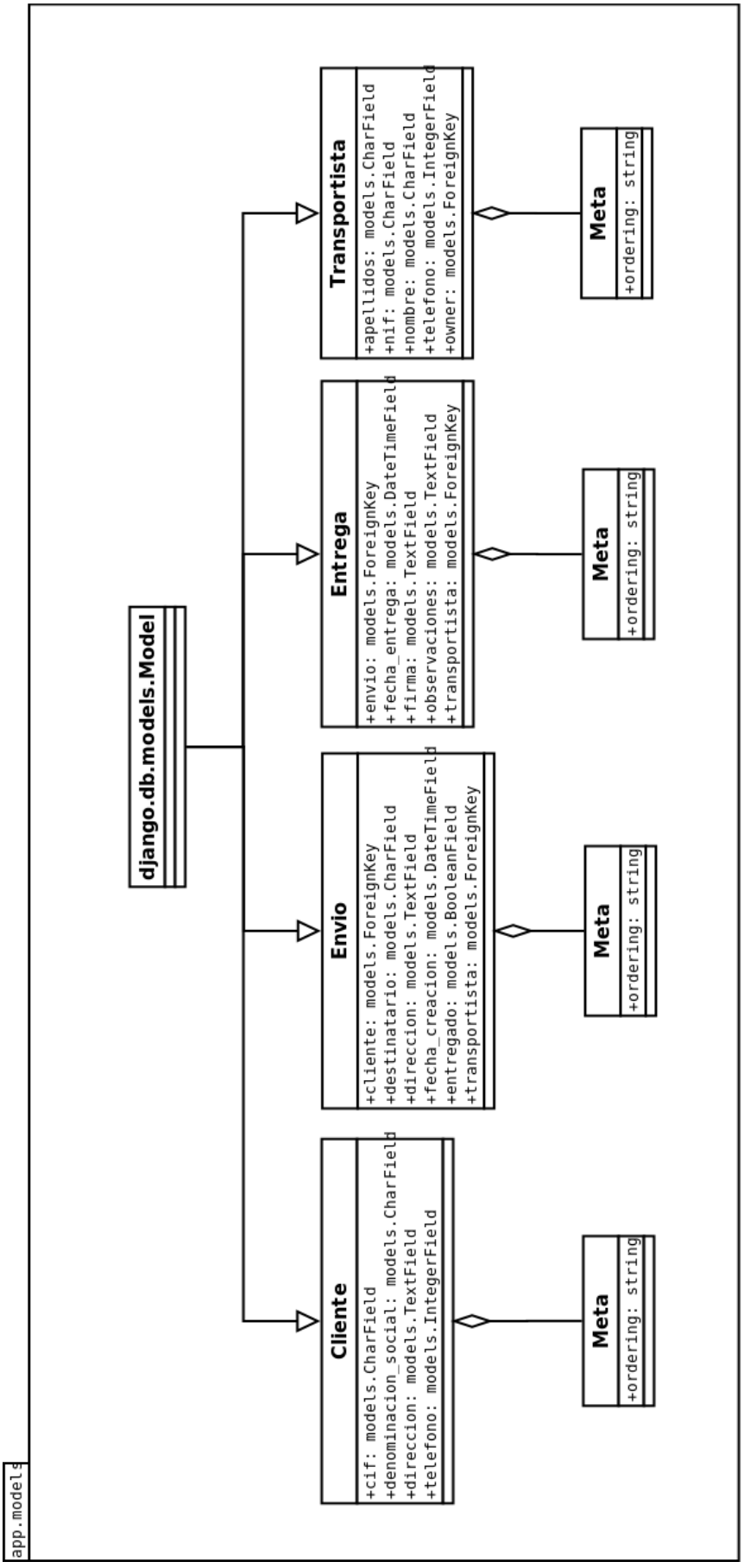
Como en el modelo de datos se almacena el NIF/CIF, es necesario codificar una clase que se encargue de validar el NIF o CIF facilitados.

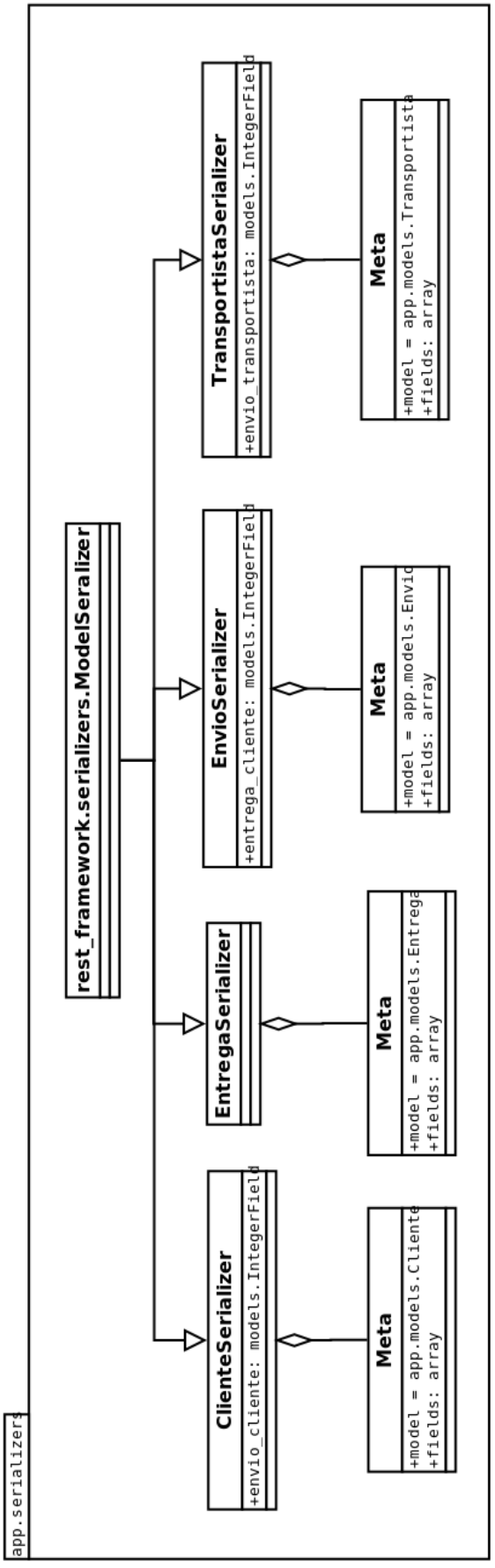
Además del modelo de datos, el diagrama UML tiene que contemplar la capa de vistas y serializers, elementos necesarios para integrar la solución en el Django Rest Framework.

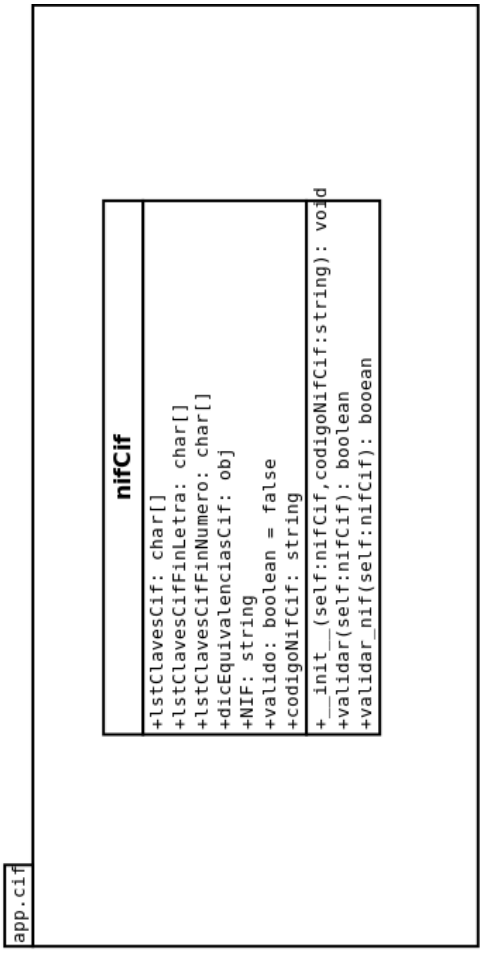
Por último, como las acciones con la API tienen que estar sujetos a una validación de permisos, se debe desarrollar una clase que realice las acción de validación.

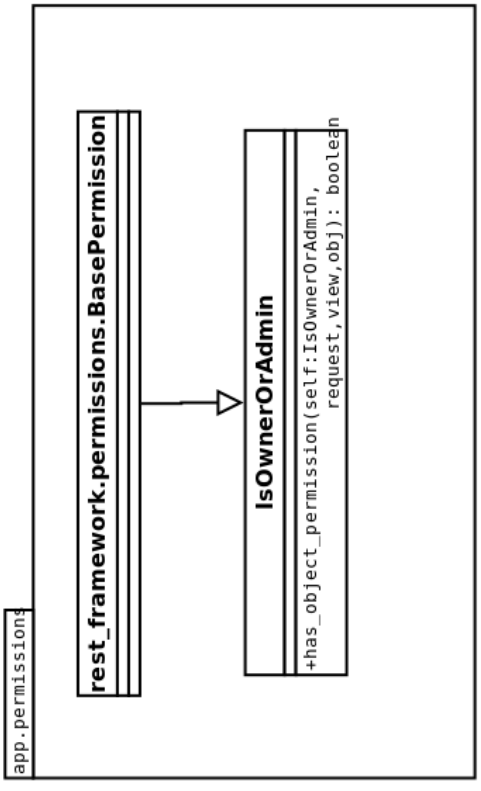
Para facilitar la lectura del diagrama UML, se han agrupado las diferentes clases a desarrollar en función de su funcionalidad:

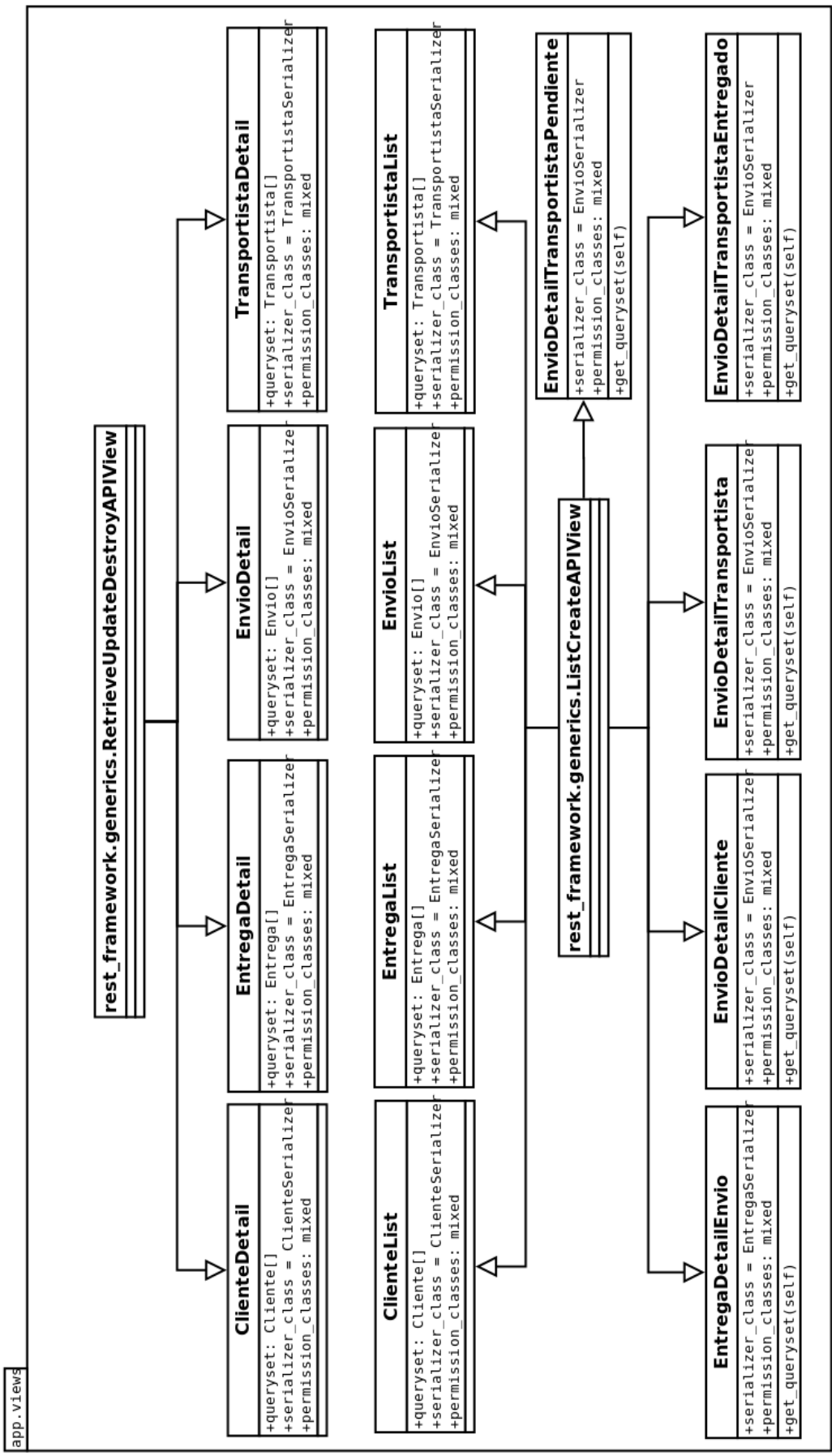
- app.models: Las clases que conformarían el modelo de datos.
- app.serializers: Las clases que se encargan de serializar/deserializar la información a mostrar/recibir en JSON o XML
- app.views: Las clases de vistas. En Django Rest Framework las vistas no sólo son visualización de datos, sino que también contienen lógica de programaión.
- app.permissions: Para validar las acciones.
- app.cif: Para validar nif/cif


app.models
django.db.models.Model
Cliente
+cif: models.CharField
+denominacion_social: models.CharField
+direccion: models.TextField
+telefono: models.IntegerField
Meta
+ordering: string
Envio
+cliente: models.ForeignKey
+destinatario: models.CharField
+direccion: models.TextField
+fecha_creacion: models.DateTimeField
+entregado: models.BooleanField
+transportista: models.ForeignKey
Meta
+ordering: string
Entrega
+envio: models.ForeignKey
+fecha_entrega: models.DateTimeField
+firma: models.TextField
+observaciones: models.TextField
+transportista: models.ForeignKey
Meta
+ordering: string
Transportista
+apellidos: models.CharField
+nif: models.CharField
+nombre: models.CharField
+telefono: models.IntegerField
+owner: models.ForeignKey
Meta
+ordering: string


app.serializers
rest_framework.serializers.ModelSeralizer
ClienteSerializer
+envio_cliente: models.IntegerFiel
Meta
+model = app.models.Cliente
+fields: array
EntregaSerializer
Meta
+model = app.models.Entrega
+fields: array
EnvioSerializer
+entrega_cliente: models.IntegerFiel
Meta
+model = app.models.Envio
+fields: array
TransportistaSerializer
+envio_transportista: models.IntegerFiel
Meta
+model = app.models.Transportista
+fields: array


app.cif
nifCif
+lstClavesCif: char[]
+lstClavesCifFinLetra: char[]
+lstClavesCifFinNumero: char[]
+dicEquivalenciasCif: obj
+NIF: string
+valido: boolean = false
+codigoNifCif: string
+__init__(self:nifCif,codigoNifCif:string): void
+validar(self:nifCif): boolean
+validar_nif(self:nifCif): booean


app.permissions
rest_framework.permissions.BasePermission
IsOwnerOrAdmin
+has_object_permission(self:IsOwnerOrAdmin,
request,view,obj): boolean


app.views
rest_framework.generics.RetrieveUpdateDestroyAPIView
rest_framework.generics.ListCreateAPIView
ClienteDetail
+queryset: Cliente[]
+serializer_class = ClienteSerializer
+permission_classes: mixed
EntregaDetail
+queryset: Entrega[]
+serializer_class = EntregaSerializer
+permission_classes: mixed
EnvioDetail
+queryset: Envio[]
+serializer_class = EnvioSerializer
+permission_classes: mixed
TransportistaDetail
+queryset: Transportista[]
+serializer_class = TransportistaSerializer
+permission_classes: mixed
ClienteList
+queryset: Cliente[]
+serializer_class = ClienteSerializer
+permission_classes: mixed
EntregaList
+queryset: Entrega[]
+serializer_class = EntregaSerializer
+permission_classes: mixed
EnvioList
+queryset: Envio[]
+serializer_class = EnvioSerializer
+permission_classes: mixed
TransportistaList
+queryset: Transportista[]
+serializer_class = TransportistaSerializer
+permission_classes: mixed
EnvioDetailTransportistaPendiente
+serializer_class = EnvioSerializer
+permission_classes: mixed
+get_queryset(self)
EntregaDetailEnvio
+serializer_class = EntregaSerializer
+permission_classes: mixed
+get_queryset(self)
EnvioDetailCliente
+serializer_class = EnvioSerializer
+permission_classes: mixed
+get_queryset(self)
EnvioDetailTransportista
+serializer_class = EnvioSerializer
+permission_classes: mixed
+get_queryset(self)
EnvioDetailTransportistaEntregado
+serializer_class = EnvioSerializer
+permission_classes: mixed
+get_queryset(self)

DESARROLLO WEBAPP

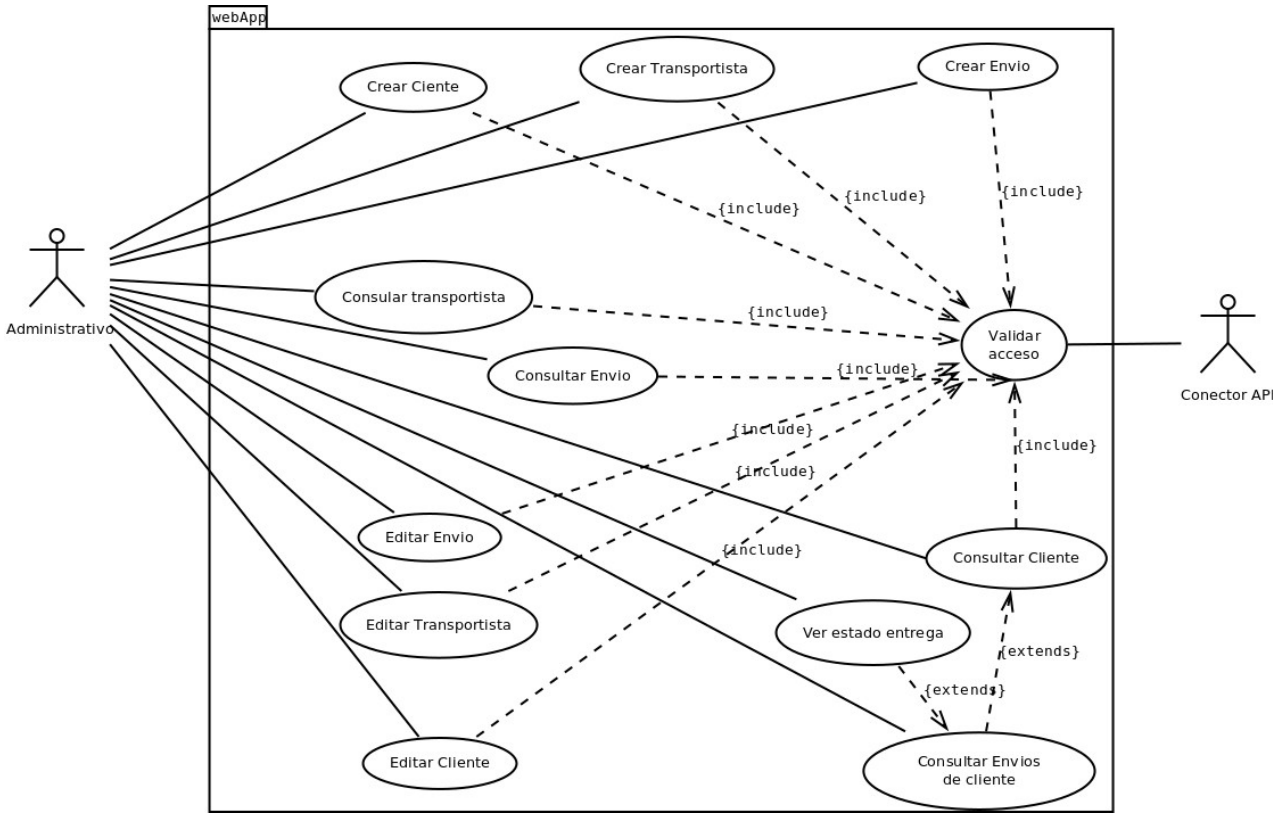
Para desarrollar la aplicación web que se utilizará desde la administración, también se ha buscado entre las diferentes soluciones web para localizar un framework en el que basar el desarrollo.

En este punto se tuvo que decidir si vitaminar la API y dotarla de interfaz web, puesto que Django es un framework para desarrollo web, o bien optar por una solución independiente. Se decidió utilizar una aplicación independiente para mantener de forma independiente las diferentes partes de la solución.

El framework seleccionado para la webapp es Laravel, en su versión 5, un framework de PHP orientado a objetos que se caracteriza por ser de los más punteros para la realización de aplicaciones modernas.

Casos de uso

Al tratarse de la aplicación que sirve de interfaz entre el administrativo y la API, en los casos de uso consideramos a ambos como agentes.

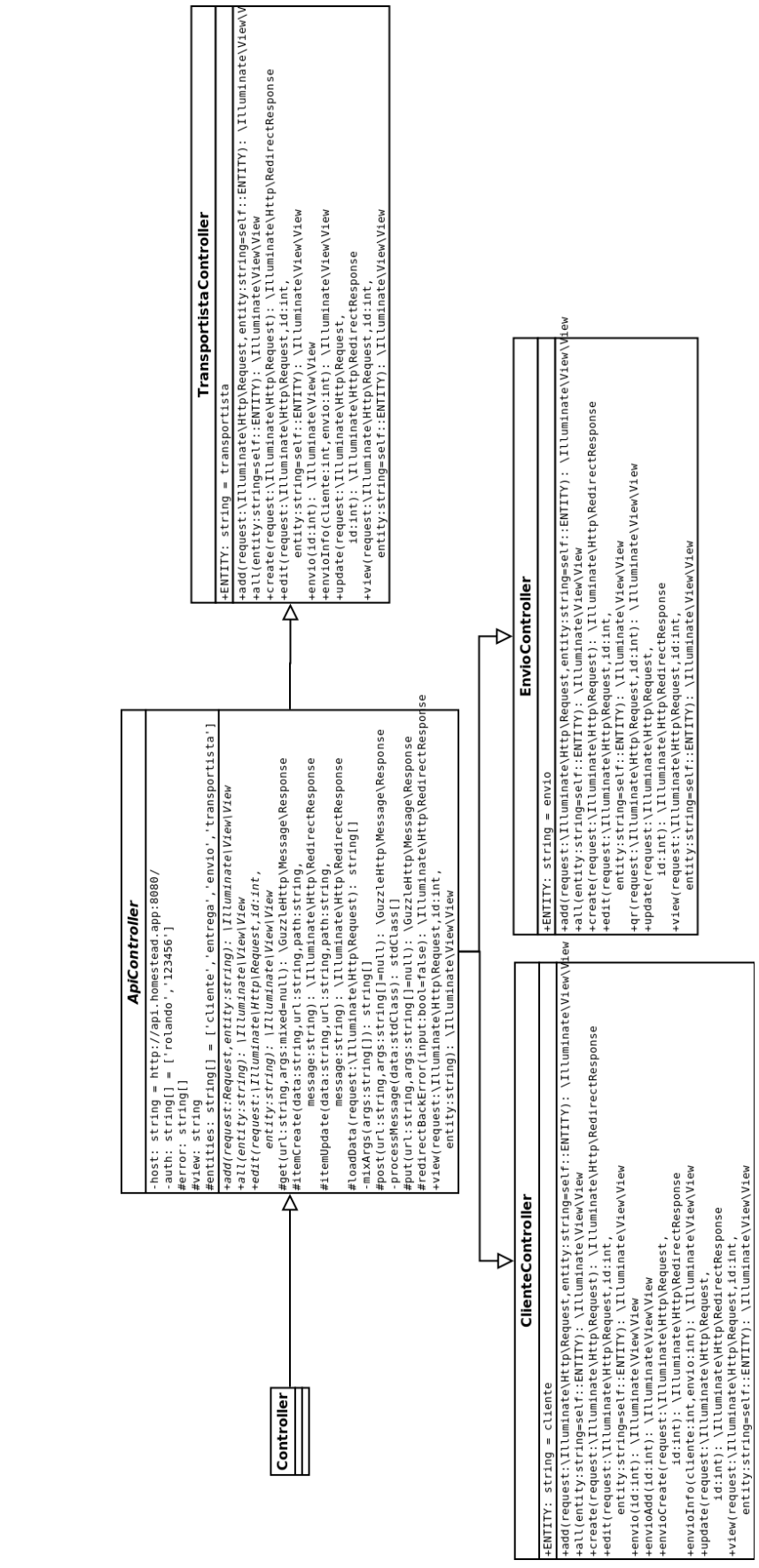


Se puede observar cómo en los casos de uso no se contempla la gestión de las entregas. Esto es así porque se considera responsabilidad del transportista, por lo que sólo él debe poder realizar tareas que alteren las entregas.

Diagrama UML

Laravel es un framework que implementa una visión evolucionada del MVC. A pesar de ello, nuestra solución sólo contempla la creación de clases que actúen a modo de controlador, puesto que las vistas no son clases (utiliza el gestor de plantillas Blade) y el modelo no existe como tal, puesto que tienen que ser conexiones a la API.

Lo que sí se contempla es la creación de una clase abstracta base que tenga la funcionalidad común, puesto que existirá un controlador por entidad gestionada por la aplicación y gran parte de la funcionalidad es compartida (como la conexión con la API).


ApiController
-host: string = http://api.homestead.app:8080/
-auth: string[] = ['rolando', '123456']
#error: string[]
#view: string
#entities: string[] = ['cliente','entrega','envio','transportista']
+add(request:Request,entity:string): \Illuminate\View\View
+all(entity:string): \Illuminate\View\View
+edit(request:\Illuminate\Http\Request,id:int,
entity:string): \Illuminate\View\View
#get(url:string,args:mixed=null): \GuzzleHttp\Message\Response
#itemCreate(data:string,url:string,path:string,
message:string) : \Illuminate\Http\RedirectResponse
#itemUpdate(data:string,url:string,path:string,
message:string) : \Illuminate\Http\RedirectResponse
#loadData(request:\Illuminate\Http\Request): string[]
-mixArgs(args:string[]): string[]
#post(url:string,args:string[]=null): \GuzzleHttp\Message\Response
-processMessage(data:stdClass): stdClass[]
#put(url:string,args:string[]=null): \GuzzleHttp\Message\Response
#redirectBackError(input:bool=false): \Illuminate\Http\RedirectResponse
+view(request:\Illuminate\Http\Request,id:int,
entity:string): \Illuminate\View\View
TransportistaController
+ENTITY: string = transportista
+add(request:\Illuminate\Http\Request,entity:string=self::ENTITY): \Illuminate\View\V
+all(entity:string=self::ENTITY): \Illuminate\View\View
+create(request:\Illuminate\Http\Request): \Illuminate\Http\RedirectResponse
+edit(request:\Illuminate\Http\Request,id:int,
entity:string=self::ENTITY): \Illuminate\View\View
+envio(id:int): \Illuminate\View\View
+envioInfo(cliente:int,envio:int): \Illuminate\View\View
+update(request:\Illuminate\Http\Request,
id:int): \Illuminate\Http\RedirectResponse
+view(request:\Illuminate\Http\Request,id:int,
entity:string=self::ENTITY): \Illuminate\View\View
EnvioController
+ENTITY: string = envio
+add(request:\Illuminate\Http\Request,entity:string=self::ENTITY): \Illuminate\View\View
+all(entity:string=self::ENTITY): \Illuminate\View\View
+create(request:\Illuminate\Http\Request): \Illuminate\Http\RedirectResponse
+edit(request:\Illuminate\Http\Request,id:int,
entity:string=self::ENTITY): \Illuminate\View\View
+qr(request:\Illuminate\Http\Request,id:int): \Illuminate\View\View
+update(request:\Illuminate\Http\Request,
id:int): \Illuminate\Http\RedirectResponse
+view(request:\Illuminate\Http\Request,id:int
entity:string=self::ENTITY): \Illuminate\View\View
ClienteController
+ENTITY: string = cliente
+add(request:\Illuminate\Http\Request,entity:string=self::ENTITY): \Illuminate\View\View
+all(entity:string=self::ENTITY): \Illuminate\View\View
+create(request:\Illuminate\Http\Request): \Illuminate\Http\RedirectResponse
+edit(request:\Illuminate\Http\Request,id:int,
entity:string=self::ENTITY): \Illuminate\View\View
+envio(id:int): \Illuminate\View\View
+envioAdd(id:int): \Illuminate\View\View
+envioCreate(request:\Illuminate\Http\Request,
id:int): \Illuminate\Http\RedirectResponse
+envioInfo(cliente:int,envio:int): \Illuminate\View\View
+update(request:\Illuminate\Http\Request,
id:int): \Illuminate\Http\RedirectResponse
+view(request:\Illuminate\Http\Request,id:int,
entity:string=self::ENTITY): \Illuminate\View\View
Controller

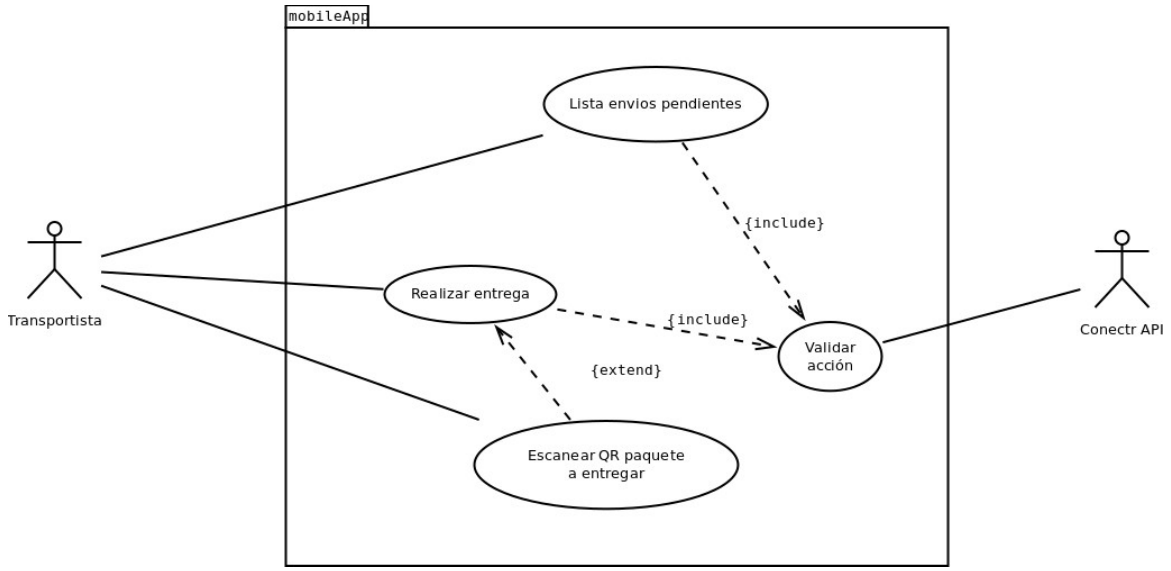
DESARROLLO APP MÓVIL

Para desarrollar la aplicación móvil se toma la decisión de realizarla con una solución híbrida en HTML5. Esto permitirá aprovechar el mismo desarrollo tanto para Android como para iPhone, Windows, Blackberry, FirefoxOS o UbuntuPhone.

Para el desarrollo de la aplicación en HTML5 se utilizará el framework AngularJS y para su implementación como aplicación “nativa” será Cordova el escogido.

Casos de uso

Al tratarse de la aplicación que sirve de interfaz entre el administrativo y la API, en los casos de uso consideramos a ambos como agentes.



Como el transportista sólo puede ver los envíos pendientes y alimentar el sistema de entregas, los casos de uso se limitan a los indicados.

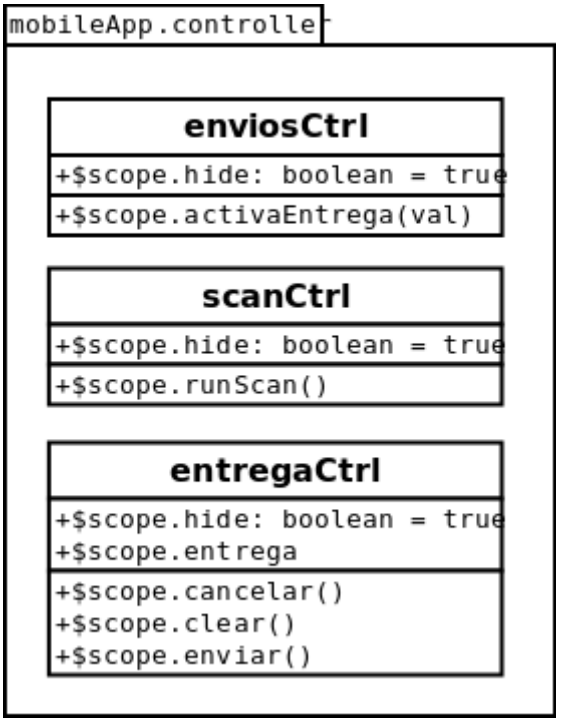
Para facilitar el trabajo al transportista, se incluye como caso de uso el escaneo del código QR que está pegado en cada paquete, de forma que la aplicación le lleve automáticamente a la entrega de dicho envío.

Diagrama UML

Como la aplicación móvil se realiza en HTML5, el lenguaje de programación es JavaScript (AngularJS es un framework de JavaScript) no podemos contar con un diagrama UML al uso.

Aunque en JS existen objetos, no se trata de un lenguaje basado en clases sino que se basa en prototipos.

Lo que se muestra a continuación es el diagrama UML dónde se refleja los controladores de AngularJS que deben ser desarrollados. Las vistas no existen como tales, puesto que es el etiquetado HTML incluido en el index.html



1. Ejecución del proyecto

1.1 Planificación temporal

- Identificar las tareas dependientes.
- Identificar los recursos necesarios en cada tarea.
- Permisos y autorizaciones necesarias.
- Identificar protocolo de actuación en cada fase.

1.1 Riesgos

- Identificación de riesgos.
- Creación de plan de prevención de riesgos.

1.1 Revisión de recursos

- Asignación de recursos (materiales y humanos) y temporización.
- Revisión del presupuesto diseñado en la fase anterior.

1.1 Documentación de ejecución

- Ejecución del proyecto.
    - ✓ Ficheros de configuración
    - ✓ Características técnicas
    - ✓ Código fuente
    - ✓ Base de datos implementada
    - ✓ Configuración del entorno de desarrollo
    - ✓ En general:
        - Texto: Descripción de las implementaciones realizadas
        - Capturas de pantallas de las implementaciones realizadas
- Manuales finales:
    - o Manuales de usuario
    - o Manuales de instalación
    - o Manuales de Configuración y administración

1. Seguimiento y control

1.1 Valoración del proyecto

- Definir medios de evaluar el proyecto.
- Definir los indicadores de calidad del proyecto.
- Elaboración del protocolo de evaluación con el cliente y su documentación.

1.1 Incidencias

- Definir un protocolo para resolución de incidencias.
    - ✓ Recopilación de información
    - ✓ Posible solución
    - ✓ Registro

1.1 Cambios

- Adaptación a los cambios y registro:
    - ✓ Migración
    - ✓ Actualización
    - ✓ Escalabilidad
    - ✓ Mejoras

1.1 Pruebas y soporte

- Registro de las pruebas realizadas:
    - ✓ de unitarias
    - ✓ de carga
    - ✓ de interfaz

- Elaboración de una acuerdo de servicio.