Memoria de Proyecto Final de Grado

Grado Multimedia

Ingeniería Web

Autoliquidación de la Tasa Portuaria de Ayudas a la Navegación

Autor: **Francisco Máiquez Beltrán**

Consultor: Ignasi Lorente Puchades

Profesor: Carlos Casado Martínez

11 de junio de 2018

Licencia

[1] Texto completo de la licencia: https://creativecommons.org/licenses/by-nc-nd/3.0/es/

Dedicatoria

Para Asun, mi amor y mi luz.

Tu apoyo y motivación me han impulsado tanto en mis estudios como en mi vida.

También para mi familia y amigos. A todos os he robado algo de dedicación para poder lograr este objetivo. Un fuerte abrazo a todos.

Resumen

Esta aplicación web facilita el proceso de autoliquidación de la Tasa de Ayudas a la Navegación a los titulares de embarcaciones de recreo y a los de buques pesqueros de litoral. Proporciona un formulario web que contempla la casuística de la legislación que regula esta tasa y guía al usuario durante este trámite.
Además, la aplicación proporciona dos entornos de usuario según su perfil: uno para los obligados a la liquidación de esta tasa, y otro para los administrativos del organismo público encargados de gestionar su cobro. El formulario estará disponible para el público general, mientras que para acceder a estos entornos será necesario autentificarse. Mediante estos sitios privados, y dependiendo del rol de usuario, se podrá acceder al histórico de autoliquidaciones, perfiles de usuario y embarcaciones registradas.
Para su desarrollo se emplearán diversos lenguajes de programación, como PHP orientado a objetos, HTML5 y CSS3, y se apoyará en una base de datos MySQL. El uso de frameworks como Laravel y Vue.js aporta un desarrollo eficaz, rápido y guiado por patrones de diseño, al mismo tiempo que la seguridad básica necesaria en este tipo de aplicaciones.

Palabras clave: TFG, Ingeniería Web, PHP, POO, MySQL, Laravel, Vue.js, autoliquidación, tasa.

Abstract

This web application facilitates the self-assessment process of the Aids to Navigation Fee to owners of recreational boats and those of coastal fishing vessels. It provides a web form that includes the casuistry of the legislation that regulates this fee and guides the user through this legal issue.
In addition, the application provides two user environments according to their profile: one for those who are obliged to the payment of this fee, and the other one for civil servants responsible for managing its charge. Although the form will be available to the public, to access these environments it will be necessary to authenticate. Through these private sites, and depending on their role, users will be able to access the history of self-assessments, their own profiles and registered ships.
Several programming languages will be used for its development, such as object-oriented PHP, HTML5 and CSS3, and it will be supported on a MySQL database. The use of frameworks such as Laravel and Vue.js provides effective, rapid and design patterns guided development, and also basic security required in this type of applications.

Keywords: Undergraduate Dissertation, Web Engineering (WE), PHP, OOP, MySQL, Vue.js, self-assessment, fee.

Notaciones y Convenciones

A continuación, se indican las tipografías seleccionadas para cada apartado:

Títulos de la introducción	Verdana 14, negrita
Títulos de los índices	Calibri 20
Títulos de nivel 1	Cambria 20, negrita
Títulos de nivel 2	Cambria 16, negrita
Títulos de nivel 3	Cambria 14
Texto normal	Arial 11
`Texto de código fuente`	Consolas 9
Texto destacado	Arial 11, cursiva
Enlaces de páginas web	Arial 9, subrayado
Cabecera de página	Arial 9
Pie de página	Arial 9
Pie de imágenes y tablas	Arial 9
Notas al pie	Arial 9
Contenido de tablas	Arial 10

Índice general

Índice de figuras y tablas

Índice de figuras

Índice de tablas

1. Introducción

El Real Decreto Legislativo 2/2011[2], de 5 de septiembre, por el que se aprueba el Texto Refundido de la Ley de Puertos del Estado y de la Marina Mercante (LPEMM), establece la regulación de las Autoridades Portuarias, que son los organismos públicos que gestionan los puertos de interés general. Este sistema portuario de titularidad estatal debe ser autosuficiente, por lo que se establecen una serie de tasas para la autofinanciación del mismo.

Una de las anteriores es la Tasa Portuaria de Ayudas a la Navegación, también denominada Tasa T0 o B0, está regulada en las secciones 1ª y 5ª, del capítulo II, título VII del Libro Primero de la citada LPEMM, y es destinada a sufragar todos los costes de la gestión y mantenimiento de las instalaciones visuales, acústicas, eléctricas y radioeléctricas que constituyen el sistema de ayudas a la navegación marítima en el litoral español.

Mientras que solo en algunas de las 28 Autoridades Portuarias (AAPP) existentes, como las de Barcelona, Alicante y Cartagena[3], han establecido un formulario web para la autoliquidación de esta tasa para las embarcaciones deportivas o de recreo y para las embarcaciones de pesca de bajura o litoral, en otras como la de A Coruña[4] solo podemos descargar de su web un formulario en formato PDF, y en el resto no encontramos mención de este trámite en sus sedes electrónicas.

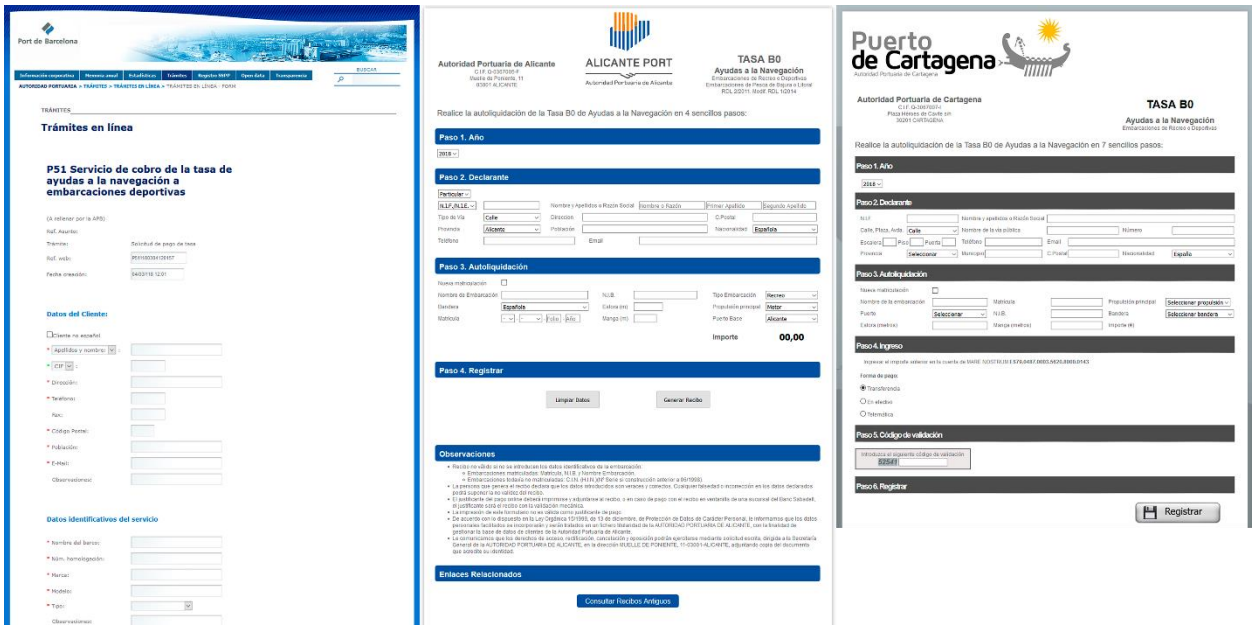


Figura 1. Formulario Tasa T0 de las Autoridades Portuarias de Barcelona, Alicante y Cartagena

[2] Ley 2/2011: https://www.boe.es/buscar/act.php?id=BOE-A-2011-16467

[3] T0 Barcelona: http://www.portdebarcelona.cat/es_ES/web/autoritat-portuaria/cobrament_t0
T0 Alicante: https://secure.puertoalicante.com/index.asp
T0 Cartagena: http://morsa.apc.es/tasab0/inicio.jsp

[4] T0 A Coruña: http://www.puertocoruna.com/es/puerto-servicios/servicios-tasas-tarifas/servicios-apac/tasas-navegacion.html

1.1. Contexto y justificación

La Ley 39/2015 [5], de 1 de octubre, de Procedimiento Administrativo Común de las Administraciones Públicas (LPACAP), en sus artículos 12.1 y 14 establece el derecho de las personas físicas a relacionarse por vía electrónica con la Administración Pública, incluidos los Organismos Públicos como las Autoridades Portuarias.

El artículo 172.1 párrafo 3 de la LPEMM, indica que las tasas portuarias serán objeto de autoliquidación, aunque mientras que el Ministerio de Fomento no desarrolle este procedimiento, las AAPP deben efectuar su liquidación. No obstante, algunas AAPP como la de Barcelona, Alicante y Cartagena, para facilitar la recogida de datos de los usuarios que prefieran relacionarse con estos organismos públicos mediante la vía electrónica, han puesto a su disposición unos formularios vía web que facilitan el cálculo del importe de esta tasa para los casos de embarcaciones de recreo y de pesqueros de litoral.

La autoliquidación se realiza mediante el siguiente proceso: el usuario rellena el formulario y registra su solicitud. De esta forma obtiene un recibo electrónico de autoliquidación con el importe que debe ingresar en la cuenta bancaria de la AP correspondiente. Cuando ese organismo recibe el ingreso, lo coteja con el registro de autoliquidación y emite la factura pertinente.

Esto supone un doble beneficio: para el interesado, puesto que agiliza el proceso de pago de esta Tasa T0, evitando tener que desplazarse hasta la oficina de estos organismos públicos y esperar turno para realizar el trámite; también para las AAPP, ya que se limitan los posibles errores en la introducción de datos por parte del interesado, elimina el soporte en papel de este trámite y supone un control más eficaz del mismo.

Queda patente el posible interés del resto de Autoridades Portuarias de incluir en sus sedes electrónicas la automatización de este proceso de cobro de la Tasa T0 para los casos mencionados. Por tanto, el presente trabajo se orienta a partir de conseguir la asignación de un concurso público de una de estas AAPP para el desarrollo de una extensión que dote a sus portales web de esta funcionalidad.

[5] Ley 39/2015: https://www.boe.es/buscar/act.php?id=BOE-A-2015-10565

1.2. Objetivos

Principales:

a) Dotar a la sede electrónica de una nueva funcionalidad para la autoliquidación de los apartados b y c del artículo 241 de la LPEMM para embarcaciones de pesca de bajura o litoral y para embarcaciones de recreo.
b) Dotar a la sede electrónica de un apartado privado para el usuario en el que pueda consultar y modificar su perfil, así como registrar sus embarcaciones y consultar sus autoliquidaciones anteriores.
c) Dotar a la sede electrónica de un apartado privado para los administrativos de la AP, en el que puedan realizar la gestión del cobro de las autoliquidaciones.

Secundarios:

a) Agilizar del procedimiento de recaudación de la Tasa de Ayudas a la Navegación.
b) Mejorar la imagen de esta AP de cara al ciudadano mediante la simplificación de trámites obligatorios.
c) Atraer mayor tráfico web hacia el dominio de esta AP.

1.3. Enfoque y método seguido

La Autoridad Portuaria sacó a concurso público el desarrollo de un apartado nuevo para su sede electrónica, mediante el cual los obligados al pago de la Tasa de Ayudas a la Navegación puedan realizar este trámite de forma telemática. Suponiendo superada la fase de licitación, finalmente la AP resuelve adjudicarnos el proyecto, por lo que en su desarrollo se debe cumplir lo estipulado en el pliego de condiciones firmado.

Entre las condiciones más importantes se encuentran los objetivos, el presupuesto licitado y aprobado, el alcance y el plazo de ejecución, así como los hitos relevantes.

- Objetivos: Definidos en el apartado anterior.
- Presupuesto: El pactado en la licitación. Se adjunta como **Anexo 1** de esta memoria.
- Alcance: Desarrollo y publicación en los servidores de la sede electrónica de la AP de una aplicación web desarrollada en PHP orientado a objetos, sobre una base de datos MySQL, para la publicación de un formulario de autoliquidación de la Tasa T0, para los casos de embarcaciones de recreo o pesqueros de litoral, así como de los sites privados para usuario y empleado público, donde puedan realizar las tareas indicadas en los objetivos. El **Anexo 2** de esta memoria muestra el detalle de las funcionalidades requeridas. Por último, una formación básica para los empleados de la AP en la gestión de ambos sites privados.
- Plazo de ejecución: Se establece la fecha de entrega del proyecto para el 11/06/2018.
- Hitos y entregables: Los indicados en el apartado Planificación.

Para garantizar un desarrollo rápido de la aplicación, con los lenguajes de programación requeridos (PHP orientado a objetos y MySQL) y los propios del entorno web (HTML5, CSS, JavaScript), con la seguridad de incorporar los patrones de diseño necesarios que garanticen el aprovechamiento de la experiencia previa y la reutilización de soluciones a los problemas de programación de determinados contextos, se hace necesario la elección de un framework de desarrollo que cumpla estos requisitos.

En este sentido, Laravel es un framework de código abierto que garantiza todo lo anterior, está bien documentado y se aprende fácilmente, tiene una comunidad amplia, y además facilita la persistencia de los objetos en la base de datos mediante su ORM llamado Eloquent, simplifica la escritura de código PHP en las vistas mediante Blade, que es un procesador de plantillas, y permite crear pruebas automatizadas para verificar el código desarrollado.

Por otro lado, para facilitar la comunicación entre las vistas y el modelo, Laravel permite el uso de otros frameworks para el front-end, como Angular o Vue. Este último será el que se utilizará para este propósito, por su sencillez e integración optimizada en Laravel.

1.4. Planificación

1.4.1. Recursos necesarios

a) Material:

- Desarrollo en local: PC con servidor web, intérprete PHP y MySQL. Se usará el entorno de desarrollo Laragon para Windows 10.
- IDE (SW de desarrollo): Sublime Text 3 con plugins para Laravel.
- SW Ofimático: Office 365, Visio 2016, Lucidchart (UML clases), GanttProject.

b) Humano:

En este proyecto su autor asumirá los diversos roles para su gestión, desde jefe de proyecto, hasta programador y especialista en audio y video.

1.4.2. Tareas

Las principales fases aplicables a este proyecto de desarrollo son 4:

- Definición y planificación.
- Análisis y diseño.
- Producción.
- Entrega final / Explotación.

1.4.3. Planificación temporal

Los hitos principales vienen establecidos por las fechas de entrega parcial de la documentación de este proyecto, debiendo hacer coincidir la consecución del hito de cada fase con la entrega de la PEC que trabaja dicho apartado.

Según el calendario previsto, este proyecto se inicia con la asignación del concurso público el 21/02/2018 y tiene una ejecución de 75 días hábiles a 8h/día, por lo que obtiene sus resultados el 11/06/2018.

ID	Tarea	Inicio	Días	Fin
1	**TFG Tasa de Ayudas a la Navegación**	**21/02/2018**	**75**	**11/06/2018**
1.1	Definición y Planificación	21/02/2018	10	06/03/2018
1.1.1	Definición de objetivos, funcionalidades y presupuesto	21/02/2018	3	23/02/2018
1.1.2	Elaboración de la memoria (PEC1)	21/02/2018	10	06/03/2018
1.1.3	Planificación inicial	02/03/2018	3	06/03/2018
1.1.4	Entrega PEC1	06/03/2018	1	06/03/2018
1.2	Análisis y Diseño	07/03/2018	18	04/04/2018
1.2.1	Elaboración de la memoria (PEC2)	07/03/2018	18	04/04/2018
1.2.2	Modelo de datos	07/03/2018	2	08/03/2018
1.2.3	Modelo de clases	09/03/2018	3	13/03/2018
1.2.4	Definición de Usabilidad y Arquitectura	14/03/2018	3	16/03/2018
1.2.5	Estudio de documentación de Laravel	20/03/2018	7	28/03/2018
1.2.6	Configuración entorno de desarrollo	20/03/2018	2	21/03/2018
1.2.7	Revisión de la planificación	02/04/2018	3	04/04/2018
1.2.8	Entrega PEC2	04/04/2018	4	04/04/2018
1.3	Producción	05/04/2018	27	14/05/2018
1.3.1	Elaboración de la memoria (PEC3)	05/04/2018	27	14/05/2018
1.3.2	Creación de rutas, controladores y vistas	05/04/2018	4	10/04/2018
1.3.3	Creación de migraciones y seeders	11/04/2018	4	16/04/2018
1.3.4	Creación de pruebas unitarias	17/04/2018	1	17/04/2018
1.3.5	Creación de modelos	18/04/2018	5	24/04/2018
1.3.6	Estudio de documentación de Vue.JS	25/04/2018	7	04/05/2018
1.3.7	Creación del front-end	26/04/2018	6	04/05/2018
1.3.8	Creación vídeo demostración	07/05/2018	3	09/05/2018
1.3.9	Revisión de la planificación	10/05/2018	3	14/05/2018
1.3.10	Entrega PEC3 y publicación vídeo	14/05/2018	1	14/05/2018
1.4	Entrega final / Explotación	15/05/2018	20	11/06/2018
1.4.1	Elaboración de la memoria (Entrega final)	15/05/2018	20	11/06/2018
1.4.2	Tests y corrección de errores	15/05/2018	8	24/05/2018
1.4.3	Pruebas de usabilidad	15/05/2018	3	17/05/2018
1.4.4	Revisión de la planificación	24/05/2018	2	25/05/2018
1.4.5	Elaboración de la presentación	28/05/2018	5	01/06/2018
1.4.6	Elaboración del vídeo	30/05/2018	7	07/06/2018
1.4.7	Elaboración del autoinforme	08/06/2018	2	11/06/2018
1.4.8	Cierre de proyecto, entrega final y publicación del vídeo	11/06/2018	1	11/06/2018

Tabla 1. Tareas de la planificación

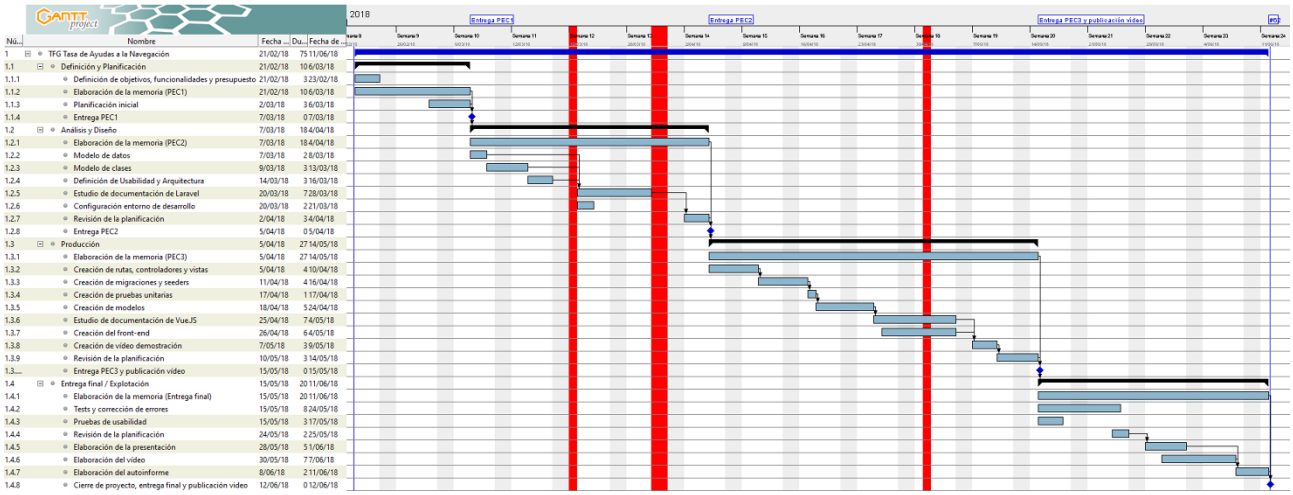


Figura 2. Diagrama de Gantt

1.4.4. Actualización de la planificación: PECs 3 y 4

Durante la fase de Producción que culmina con la entrega de la PEC3 han surgido algunas tareas que no estaban suficientemente dimensionadas en la planificación original, sobre todo como consecuencia de no haber realizado un buen análisis de la documentación de Laravel y Vue. Estas tareas se refieren a dotar a la aplicación de autenticación de usuarios, validación de formularios, interactividad del frontend (elementos reactivos), envío de email, generación de documentos en PDF y adjuntar documentos al formulario. Estas funcionalidades, que se integran en el desarrollo de modelos, controladores y vistas, han provocado un retraso en la producción.

Por otro lado, el orden de las tareas durante de la producción tampoco ha sido el planificado: en primer lugar, se han creado las migraciones y *seeders* en su totalidad, que ha llevado más tiempo del esperado, luego se han creado todos los modelos y las relaciones entre ellos según las restricciones de integridad establecidas en las migraciones, y por último se ha procedido a una iteración en el desarrollo de rutas, controladores, modelos y vistas para cada nuevo apartado de la aplicación, que ha concluido con algunas de estas tareas sin terminar.

Por tanto, se hace necesario una replanificación de las tareas de la última fase, la entrega final, para incluir estas tareas incompletas de la fase anterior. La nueva planificación real de la fase 3 y de la esperada para la fase 4 son como se muestran en la tabla 2, y el diagrama de Gantt resultante es el mostrado en la figura 3.
En esta nueva planificación, para la fase 4 se realiza una revisión de planificación en paralelo al resto de actividades para comprobar su adecuación al tiempo asignado, y se guardan 2 días de margen al final de las actividades para una posible replanificación de imprevistos.

ID	Tarea	Inicio	Días	Fin
1.3	Producción	05/04/2018	27	14/05/2018
1.3.1	Elaboración de la memoria (PEC3)	05/04/2018	27	14/05/2018
1.3.2	Creación de migraciones y seeders	05/04/2018	7	13/04/2018
1.3.3	Creación de modelos	13/04/2018	19	10/05/2018
1.3.4	Creación de rutas, controladores y vistas	18/04/2018	16	10/05/2018
1.3.5	Estudio de documentación de Vue.JS	23/04/2018	13	10/05/2018
1.3.6	Creación del front-end	26/04/2018	10	10/05/2018
1.3.7	Creación vídeo demostración	11/05/2018	1	11/05/2018
1.3.8	Revisión de la planificación	14/05/2018	1	14/05/2018
1.3.9	Entrega PEC3 y publicación vídeo	14/05/2018	1	14/05/2018
1.4	Entrega final / Explotación	15/05/2018	20	11/06/2018
1.4.1	Elaboración de la memoria (Entrega final)	15/05/2018	19	08/06/2018
1.4.2	Revisión de la planificación	15/05/2018	19	08/05/2018
1.4.3	Modificación de modelos	15/05/2018	14	01/06/2018
1.4.4	Modif. y creación de rutas, controladores y vistas	15/05/2018	14	01/06/2018
1.4.5	Pruebas de usabilidad	31/05/2018	2	01/06/2018
1.4.6	Tests y corrección de errores	28/05/2018	5	01/06/2018
1.4.7	Elaboración de la presentación	01/06/2018	2	04/06/2018
1.4.8	Elaboración del vídeo	05/06/2018	3	07/06/2018
1.4.9	Elaboración del autoinforme	07/06/2018	1	07/06/2018
1.4.10	Cierre de proyecto, entrega final y publicación del vídeo	11/06/2018	1	11/06/2018

Tabla 2. Planificación real de la fase 3 y estimada de la fase 4

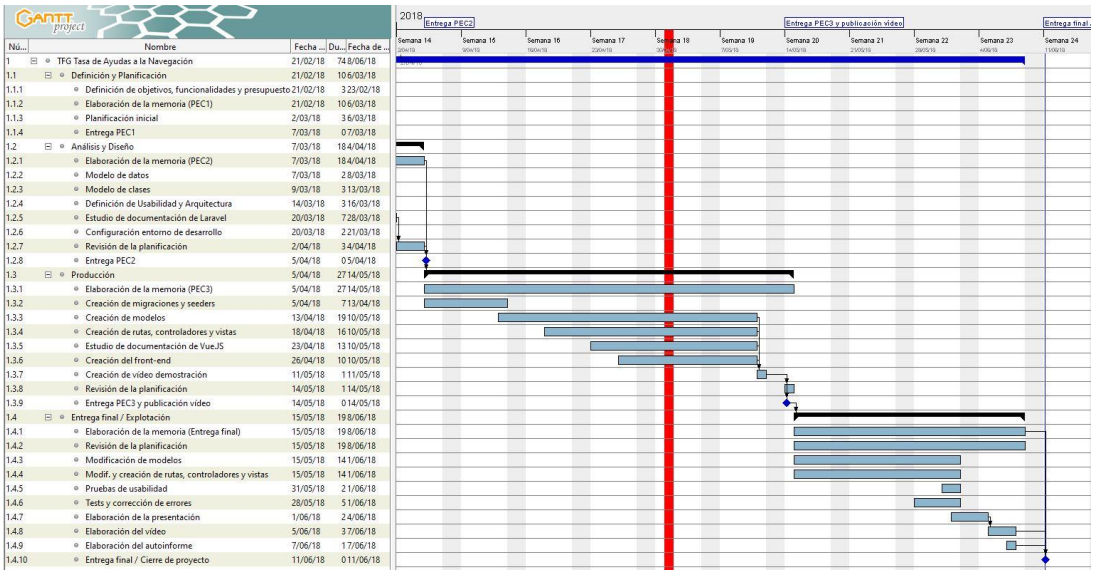


Figura 3. Diagrama de Gantt de la nueva planificación

1.4.5. Seguimiento de la planificación en la fase final

En esta última fase se ha llevado a cabo la replanificación de las tareas pendientes de la PEC anterior. Aun así, el número de funcionalidades a desarrollar en esta aplicación era demasiado elevado para el tiempo real de desarrollo disponible. Según la planificación inicial se dispuso una jornada de 8 horas al día de dedicación, cuando la real en ningún caso ha podido pasar de las 4 horas diarias, ya que trabajo en jornada de mañanas y solo puedo dedicar tiempo al desarrollo por las tardes. Aun dedicando fines de semana completos no he podido asumir el volumen de trabajo propuesto inicialmente.
Los motivos principales han sido la falta de tiempo de desarrollo y el desconocimiento de los dos frameworks principales utilizados: Laravel y Vue, así como la integración de ambos en el Frontend.

Como conclusión, y teniendo en cuenta que el calendario para la entrega es inamovible, se debería haber rebajado el nivel de exigencia en el alcance de este proyecto.

1.5. Sumario de productos obtenidos

Como resultado de este proyecto se obtendrá una aplicación web desarrollada conforme a lo indicado en el pliego de condiciones del concurso público, que se resume en el Anexo 2, Funcionalidades requeridas.

En esta última entrega se cierra el desarrollo a fecha 08/06/2018 para dedicar estos 3 últimos días a terminar la memoria, la elaboración de la presentación, vídeo de defensa y autoinforme.

Los hitos conseguidos son los siguientes:

- Formulario de autoliquidación funcional, desarrollado en Laravel y Vue.
- Validación completa del formulario en el lado del servidor, y básica en el lado del cliente.
- Creación del documento de pago en PDF según los datos introducidos.
- Envío de documentos en el formulario: guardado en el servidor y ruta indicada en clases.
- Envío de correo electrónico al declarante con el documento PDF adjunto.
- Implementación del sistema de autenticación del Laravel.
- Desarrollo de layouts y vistas para el Dashboard.
- Ocultación de secciones en este entorno según rol de usuario.
- Mostrar la información de declarantes, embarcaciones y autoliquidaciones recabadas en el formulario, incluyendo los archivos que se adjuntan en cada uno.

Los hitos pendientes para una aplicación completamente funcional son los siguientes:

- Autoliq. complementaria: cálculo de la diferencia del importe con la liquidación anterior.
- Autocompletar formulario con datos de usuario y embarcación si se ha autenticado.
- Desarrollo completo del Dashboard: Filtrado, búsqueda y paginación de información.
- Acciones según rol: cambiar el estado de una tasa, añadir un comentario, modificar el perfil de declarante, alta de nuevos parámetros de tasa.
- *Middleware* de restricción de acceso para la separación de contenidos en el Dashboard.
- Tarea programada que pase el estado de las tasas a expiradas cumplido el plazo.
- Ajustar la visualización para pantallas pequeñas.

Con el tiempo disponible al día para el desarrollo de este proyecto, no más de 4 horas por las tardes y algunos fines de semana, se calcula que, con el conocimiento adquirido de Laravel y siendo realistas con la estimación de tiempo, en 4 semanas de desarrollo podrían estar terminadas todas las tareas pendientes.

No obstante, se han concentrado los esfuerzos en obtener un producto que pueda usarse, desactivando el registro de usuarios en tanto el Dahsboard no esté completamente operativo. Los administrativos podrían conectarse y obtener la información de las autoliquidaciones recibidas para su seguimiento. En una segunda fase de implantación, un mes más tarde, cuando esté completamente desarrollado el producto, se podría dar acceso a los declarantes.

2. Arquitectura

2.1. Cliente

Los usuarios de la aplicación, tanto los obligados a la autoliquidación de la T0 como los administrativos que gestionan su cobro, accederán a la aplicación mediante un navegador web. El único requerimiento es que este esté actualizado y pueda interpretar los lenguajes utilizados del lado cliente: HTML5, CSS3 y JavaScript.

2.2. Servidor

El pliego de condiciones por el que se nos asigna este proyecto indica que se deben utilizar los servidores puestos a disposición por la AP, en los que ya se encuentran corriendo otras aplicaciones de su Sede Electrónica. Este servicio está diseñado para alta disponibilidad 24x7, con previsión de un incremento importante del número de conexiones.
Disponen de 2 centros de proceso de datos (CPD) distribuidos en edificios separados, conectados mediante fibra óptica. En cada uno de ellos hay un armario rack de 22 U que asegura la disponibilidad del servicio en caso de fallo del otro CPD, configurado con los siguientes equipos:

- *Electrónica de red:* 1 firewall, 1 balanceador de carga, 1 switch de red y 1 switch de datos de fibra óptica.
- *3 servidores físicos:* 1 para un servicio web de 6 instancias distribuidas (3 en cada CPD), a los que llamaremos sW1 y sW2 (servidor web 1 y 2, según el CPD en que se encuentren); 1 para un servidor de aplicaciones (sA1 y sA2) con otras 8 instancias distribuidas en los 2 CPDs; y 1 para un servidor de bases de datos (sBD1 y sBD2) con 2 instancias en configuración activo-pasivo, siendo el sBD1 el activo.
- *1 cabina de discos SAN* (*Storage Area Network*) para cada instancia de sBD, denominadas Cab1 y Cab2, que se replican entre ellas de forma síncrona.

Respecto al HW de los servidores, todos los equipos tienen redundancia de fuentes de alimentación y de tarjetas de red. Para los sW se dispone un procesador Intel Xeon E5 de 4 núcleos a 3,4GHz y 125MB RAM, con posibilidad de ampliar a un segundo procesador y otras 128MB RAM. Los equipos sA y sBD tienen la misma configuración: un procesador Intel E7 a 3,4GHz y 256MB RAM, también con posibilidad de duplicar el procesador y memoria RAM. Para el almacenamiento masivo de estos equipos, se dispone de los LUN (*Logical Unit Number*) necesarios en la SAN y, por tanto, se instalan adaptadores HBA (*Host Bus Adapter*) para conectar los equipos servidor a la red de datos de fibra óptica.

En cuanto a sistema operativo, todos los equipos servidor incorporan Red Hat Enterprise Linux 7.3 (RHEL). Para el servicio web se dispone de Apache 2.4 con intérprete de PHP 7.2, para el servicio de base de datos se instala MySQL Enterprise Edition con MySQL Cluster CGE para la gestión del cluster activo-pasivo, y para el servicio de aplicaciones Java se usa el motor WildFly 11 para Java Enterprise Edition.

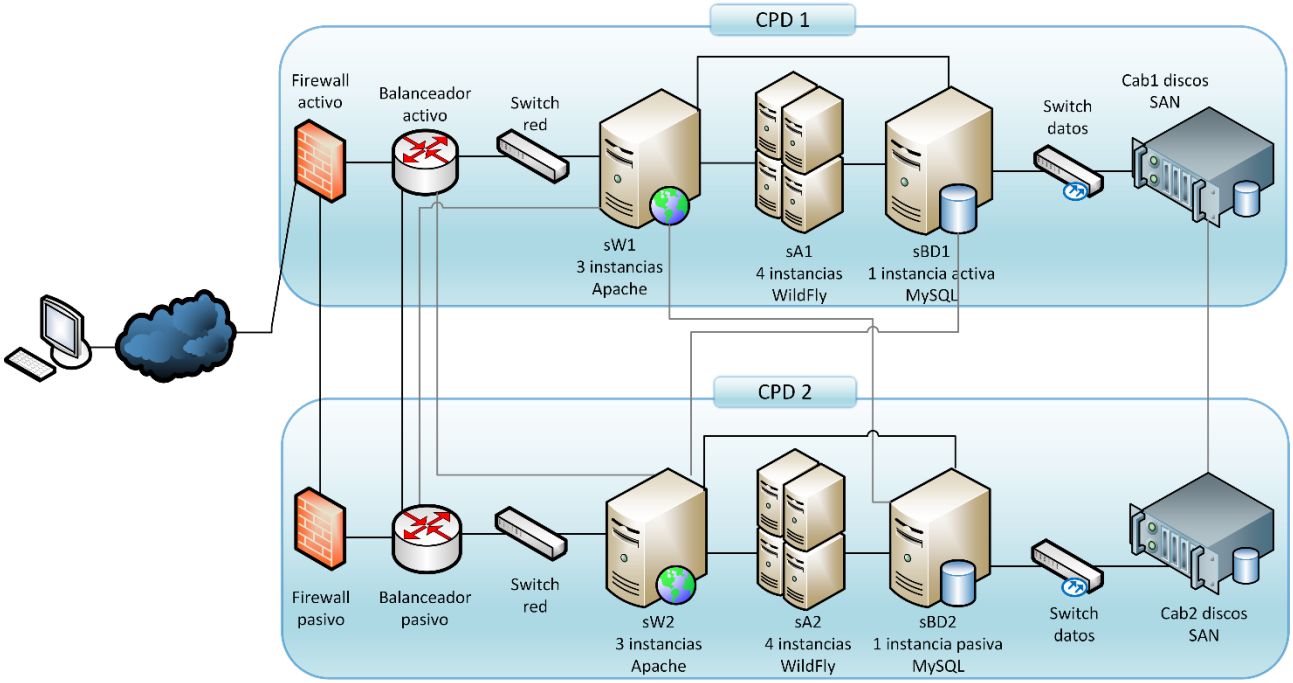


Figura 4. Arquitectura del servicio de Sede Electrónica de la AP

Debido a que la plataforma de servicio es compartida con otras aplicaciones, en el contrato de acuerdo de nivel de servicio (SLA – *Service Level Agreement*) se detalla que es responsabilidad de la AP comprobar que la respuesta del servicio sea eficaz, lo que se puede medir mediante software de monitorización, como:

- Las propias del sistema operativo RHEL: top, vmstat o sysstat.
- Las propias de Apache: apachetop o mod_status.
- Herramientas para WildFly, como WildFly Monitor, de Manage Engine.
- Herramientas para MySQL, como MySQL Enterprise Monitor.
- Otras herramientas externas, como Google Analitycs, Nagios o Zabbix.

Es por tanto responsabilidad de la AP dotar de más recursos al servicio en caso de detectar picos de saturación de memoria o procesador, mediante alguna de las herramientas sugeridas o similares, aumentando el número de procesadores y/o RAM de los servidores saturados, y en caso de tener muchas conexiones concurrentes, el número de instancias distribuidas del servidor web.

Para este proyecto se hará uso del servidor web extendido con PHP y del motor de base de datos MySQL, así como de la red de comunicación y de datos.

2.3. Estructura de la aplicación

Dado que tendremos un servicio implementado en capas cliente/servidor, la aplicación debe estructurarse también en este sentido, ofreciendo una capa para el lado de cliente que contenga las vistas que componen la interfaz de usuario, y por otro lado la capa de servicio.

Esta última, se desglosará en otras capas para la lógica del sistema y para la base de datos, además de las que proporciona el framework Laravel para mapear los datos de los objetos a la base de datos relacional, a través de su ORM denominado Eloquent, y para realizar los enrutamientos, que también son un apartado importante a destacar en este framework.

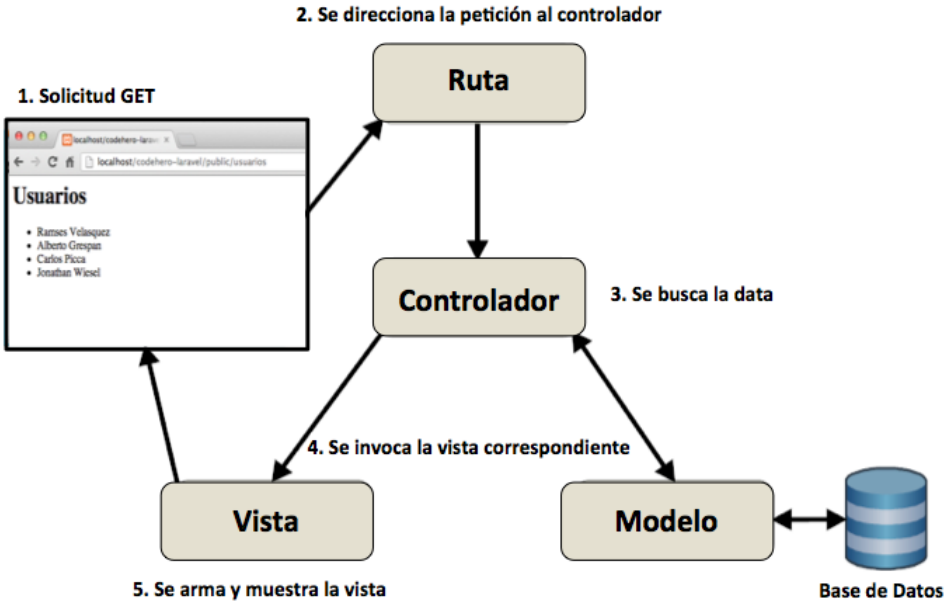


Figura 5. Estructura de una aplicación en Laravel [6]

Como se puede apreciar en la figura 5, la estructura de las aplicaciones en Laravel sigue el patrón de arquitectura MVC, o Modelo-Vista-Controlador, que permite desacoplar la interfaz gráfica del resto del sistema, estableciendo una estructura en capas que permita una separación de responsabilidades, en la que cada componente se encarga de un apartado:

- La vista presenta la información al usuario y recogen su interacción, que es enviada al controlador. Por ejemplo, si un usuario pulsa un botón en una pantalla, no es la vista la encargada de resolver esa petición.
- El controlador decide qué vista debe mostrarse al usuario en cada momento, y también qué ocurre en cada petición del usuario, para lo que se comunica con el modelo.
- El modelo es el responsable de la lógica de negocio, esto es, el comportamiento del sistema en términos del dominio del problema que debe resolver la aplicación.

[6] Figura 5: Imagen tomada del libro online “Laravel-5”, de Richos, publicado en GitBook https://richos.gitbooks.io/laravel-5/content/capitulos/chapter9.html

3. Análisis y Diseño / Diagramas UML

3.1. Historias de usuario

3.1.1. Como declarante, quiero poder solicitar la autoliquidación de la tasa T0 sin tener que desplazarme.

- Crear un formulario online para la recepción en la AP de solicitudes telemáticas de autoliquidación de la tasa T0.
- Los administrativos de la AP deben disponer de una aplicación que funcione back-end de este formulario web para poder consultar las solicitudes de autoliquidación presentadas.

3.1.2. Como declarante, quiero poder aportar la documentación necesaria en la solicitud de autoliquidación de la tasa T0.

- En el formulario online, el declarante debe poder adjuntar la documentación requerida por la AP para la autoliquidación de esta tasa.

3.1.3. Como declarante, acabo de comprar una nueva embarcación y pretendo matricularla, pero en Capitanía Marítima me exigen un comprobante de pago de la Tasa T0 para iniciar el procedimiento.

- El formulario online no debe requerir, en este caso, la información de la embarcación referente al NIB o la matrícula, puesto que esos datos los asigna la Capitanía Marítima después de que se haya pagado esta tasa.

3.1.4. Como declarante, no español, con una embarcación a vela mayor de 12 metros de eslora, quiero poder pagar la tasa T0 desde mi móvil cuando llegue a un puerto español.

- El formulario debe ser responsive para ajustarse a distintos tamaños de pantalla.
- El formulario debe disponer de una opción para indicar el número de días de estancia para las embarcaciones transeúntes.

3.1.5. Como declarante, con una embarcación a motor menor de 9 metros de eslora, quiero poder descargar la factura de liquidación de la tasa T0 cada vez que me sea exigida en Capitanía Marítima o por otra Autoridad Portuaria.

- Los declarantes deben tener acceso a un back-end similar al de los administrativos.
- Este back-end solo mostrará la información que concierne a cada declarante.
- El declarante debe poder consultar la información de autoliquidaciones anteriores.

3.1.6. Como declarante, empresa de alquiler de embarcaciones de recreo, quiero poder realizar la autoliquidación de la tasa T0 de una forma más ágil, sin tener que indicar todos los datos de la empresa o de la embarcación.

- El declarante debe poder registrar sus embarcaciones en el sistema.
- Cuando el declarante realice una autoliquidación nueva con su sesión de usuario abierta, los datos del declarante se autocompletarán con los del perfil del declarante.
- Cuando el declarante realice una autoliquidación nueva con su sesión de usuario abierta, el formulario sugerirá al usuario que seleccione una embarcación de las que tiene registradas para que se autocomplete la información.

3.1.7. Como administrativo, quiero poder consultar y filtrar información sobre declarantes, embarcaciones y autoliquidaciones.

- El sistema debe facilitar al administrativo su tarea, permitiéndole filtrar la información mostrada en base a sus necesidades.

3.1.8. Como administrativo, quiero poder gestionar las autoliquidaciones pendientes para revisar las que ya están pagadas e indicarlo en el sistema.

- Las autoliquidaciones deben tener varios estados dentro del sistema.
- El sistema debe permitir al administrativo hacer una gestión de las autoliquidaciones recibidas, sin poder modificar la información que contiene, de forma que el estado de la autoliquidación varíe según la gestión realizada.
- El sistema debe permitir filtrar el listado de autoliquidaciones por estos estados.

3.1.9. Como administrativo, quiero que el sistema me avise de las autoliquidaciones que estén a punto de expirar el plazo sin haber realizado el pago.

- El sistema debe tener una tarea programada que chequee periódicamente las autoliquidaciones y envíe al administrativo un email con un listado de las autoliquidaciones pendientes de cobro que están próximas a expirar.
- Si el administrativo comprueba que no se ha recibido el pago de estas autoliquidaciones, debe poder realizar una acción sobre las mismas para alertar al declarante sobre este hecho.

3.1.10. Siendo administrativo, quiero poder registrarme en el sistema y realizar una autoliquidación como declarante, puesto que también soy propietario de una embarcación.

- Un administrativo debe poder cambiar de un rol de usuario a otro sin tener que finalizar e iniciar sesión.

3.1.11. Como declarante, debo realizar una autoliquidación complementaria porque en la anterior hubo un error en la introducción de la eslora de mi embarcación.

- El sistema debe permitir realizar una declaración en base a una anterior, de forma que calcule el nuevo importe e indique la diferencia resultante.
- Si la diferencia es a favor del declarante, el sistema facilitar al declarante la introducción los datos de una cuenta bancaria en la que desea recibir el abono.

3.2. Casos de uso

Del análisis de los requisitos para este proyecto (ver Anexo 2 – Funcionalidades requeridas), se deduce que en el modelo hay 2 actores: el Declarante (obligado a la autoliquidación) y el Administrativo (el trabajador de la AP).

Los declarantes pueden autenticarse (login), registrarse, autoliquidar la Tasa T0, modificar su perfil, registrar embarcaciones, consultar embarcaciones y autoliquidaciones anteriores.

Los administrativos se autentican, consultan autoliquidaciones, embarcaciones y declarantes, y gestionan autoliquidaciones, embarcaciones y los parámetros de cuantificación de la Tasa T0.

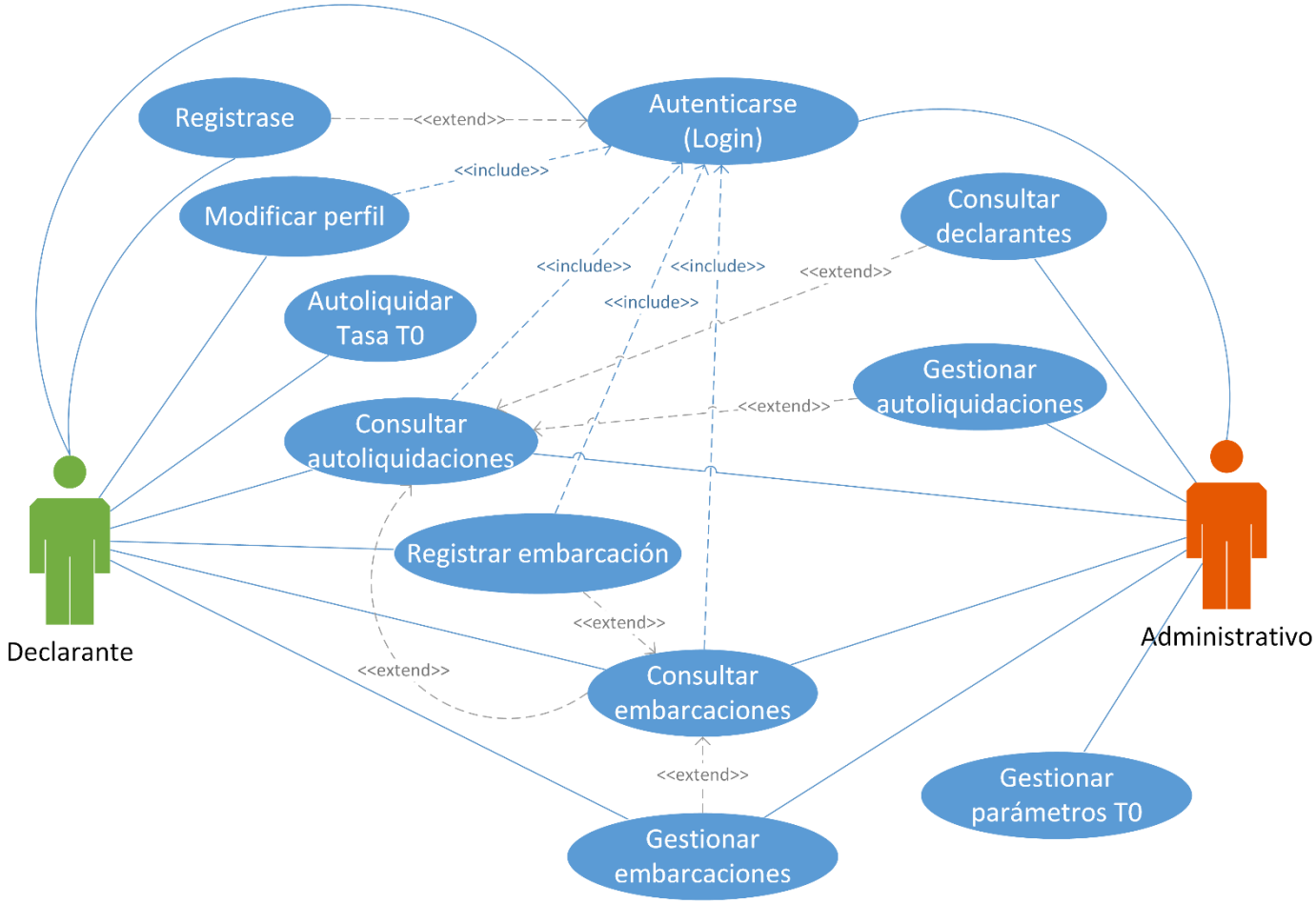


Figura 6. Casos de uso

A continuación, se realiza la especificación textual de los casos de uso detectados en la figura 6, con el fin de obtener su análisis completo.

3.2.1. Caso de uso *Autoliquidar Tasa T0*

Este CU Permite al declarante introducir los datos para realizar una autoliquidación de la tasa T0, pero no dispone de una pasarela de pago electrónico por tarjeta de crédito, por lo que debe realizar este trámite en persona, o mediante una transferencia a través de banca online, con el documento de pago obtenido al registrar la solicitud.

Al administrativo le permite confirmar la recepción del pago y adjuntar la factura correspondiente.

El sistema chequea diariamente (tarea programada a las 07:00 a.m.) que las autoliquidaciones pendientes estén dentro del plazo indicado. En caso contrario, las pone en estado “plazo expirado” y notifica por email al usuario de este hecho y sus consecuencias.

Nombre:	*Autoliquidar Tasa T0*
Actores principales:	Usuario con rol de declarante
Actores de soporte:	Usuario con rol de administrativo
Precondición:	El administrativo debe autenticarse en el sistema. El declarante puede autenticarse o no en el sistema.

Escenario principal de éxito:

1. El sistema muestra la pantalla “Form_T0” con el formulario de autoliquidación de la tasa T0.
2. El declarante lo rellena y selecciona calcular el importe.
3. El sistema muestra el importe en la pantalla “Resultado”.
4. El usuario selecciona registrar solicitud de autoliquidación.
5. El sistema guarda los datos introducidos, pone la autoliquidación en estado “email no confirmado”, genera el documento de pago o de exención y lo adjunta a la autoliquidación.
6. El sistema confirma el email.
7. El sistema muestra la pantalla “Confirmación_autoliquidación”, envía el documento de pago por email al declarante y pone la autoliquidación en estado pendiente.
8. El administrativo consulta la autoliquidación pendiente del declarante.
9. El sistema muestra la pantalla “Detalle_autoliquidación”.
10. El administrativo puede gestionar la autoliquidación.

Escenarios alternativos (extensiones):

2a. El declarante se ha autenticado: sus datos ya aparecen en el formulario. También puede seleccionar una de sus embarcaciones registradas o añadir los datos de una nueva. Selecciona calcular el importe.
  2a1. Vuelve al paso 3.

3a. El sistema detecta información incorrecta o algún campo obligatorio vacío, y muestra la pantalla “Resultado” indicando este hecho.
  3a1. El usuario selección Aceptar.
  3a2. Vuelve al paso 1, con los campos resaltados para localizarlos fácilmente.

3b. El sistema muestra en la pantalla “Resultado” el motivo de la exención del pago de esta tasa.
  3b1. Vuelve al paso 4.

3c. Ya existe una autoliquidación para este año o indefinida para la embarcación indicada. El sistema muestra un mensaje en la pantalla “Resultado” indicando este hecho.
  3b1. Fin del caso de uso.

4a. El usuario selecciona cancelar.
  3b1. Vuelve al paso 1 para que el usuario pueda modificar los datos introducidos.

Pantallas:

- **Form_T0**

  Datos solicitados:

  - Generales: Año, autoliquidación complementaria (si/no) y referencia de autoliquidación inicial, política de protección de datos (enlace y check de aceptación).
  - Declarante: Tipo de documento de identidad, nº del documento, Nombre y apellidos, tipo de vía, nombre de la vía, código postal, provincia, población, email, teléfono y foto.
  - Embarcación: Tipo (pesquero o recreo), Nombre, Matrícula, NIB, Bandera, Propulsión, Eslora, Manga, Puerto base, Nueva matriculación (si/no)

- **Resultado**

  Datos mostrados:

  A) Datos incorrectos o incompletos: Indica qué campos deben ser revisados. Botón Aceptar.

  B) Embarcación exenta de pago: Muestra los datos introducidos del declarante y la embarcación, e indica el motivo de la exención. Botones Cancelar y Registrar.

  C) Importe: Muestra los datos introducidos del declarante y la embarcación, e indica el importe total a ingresar y el número de cuenta de la AP. También desglosa el importe total en los subtotales para la AP y SSSM y muestra los botones Cancelar y Registrar.

- **Confirmación_autoliquidación**

  Datos mostrados:

  Documento PDF con los datos mostrados para la pantalla Resultado en su caso C), salvo los botones Cancelar y Registrar. El documento también indica el número de registro de autoliquidación de Tasa T0, la fecha de fin de plazo para el ingreso y las consecuencias de no realizar el ingreso a tiempo.

- **Detalle_autoliquidación**

  Datos mostrados:

  Declarante: Los de la pantalla Resultado apartado C), salvo el botón Cancelar y Registrar. En su lugar, muestra el botón Volver. Listado de documentos adjuntos.

  Administrativo: Además de lo anterior, paneles con listados de: Otras autoliquidaciones de esta embarcación y Otras autoliquidaciones de este usuario. Botón Modificar (pasa a modo edición).

  Datos solicitados (perfil de Administrativo – modo edición):

  Muestra los mismos datos, añadiendo 4 campos editables: Fecha de pago, check exenta de pago, importe del ingreso recibido y observaciones. Botones Adjuntar y eliminar adjunto, Cancelar y Guardar.

Tabla 3. Caso de uso “Autoliquidar Tasa T0”

La tarea del sistema “Confirmar email” no se ha detallado porque se repite en caso de uso Registrarse, que extiende al de Autenticarse. Para evitar repetir la explicación de esta tarea en los casos de uso anteriores, se ha optado por detallarla aparte.

Cuando el resultado de una autoliquidación es que está exenta, no se termina el CU. De esta forma le llega al administrativo una autoliquidación pendiente con resultado exenta. Así, la AP puede emitir un certificado de exención, en lugar de una factura, y adjuntarla a la autoliquidación, como se verá en el CU Gestionar autoliquidación.

3.2.2. Caso de uso *Autenticarse (login)*

Permite al usuario identificarse en el sistema.

Nombre:	*Autenticarse (login)*
Actores principales:	Declarante y Administrativo
Escenario principal de éxito: 1. El sistema muestra la pantalla “Acceso”, con el formulario de autenticación. 2. El declarante introduce los datos (todos son obligatorios) y selecciona Entrar. 3. El sistema comprueba que los datos son correctos e inicia la sesión de usuario según su perfil. 4. El sistema muestra la pantalla “Dashboard”.	
Escenarios alternativos (extensiones): 2a. El declarante selecciona Cancelar. 2a1. Fin del caso de uso. 3a. El declarante introduce datos incorrectos o deja campos en blanco. 3a1. El sistema muestra la pantalla “Login” indicado este hecho y resaltando los campos implicados. 3a2. Vuelve al paso 2.	
Pantallas: • **Login** Datos solicitados: Usuario y contraseña, ambos obligatorios. Enlace “Registrarse”. • **Dashboard** Datos mostrados – todos los perfiles: nombre de usuario y botón Salir. Datos mostrados – perfil de Declarante: - Acciones del panel menú: Perfil, Embarcaciones, Autoliquidaciones - Panel general: Mensaje de bienvenida Datos mostrados – perfil de Administrativo: - Acciones del panel menú: Declarantes, Embarcaciones, Autoliquidaciones, Parámetros T0. - Panel general: Listado de autoliquidaciones.	

Tabla 4. Caso de uso Autenticarse (login)

3.2.3. Caso de uso *Registrarse*

Permite al usuario registrarse en el sistema. Extiende al CU Autenticarse, en su paso 2.

Nombre:	*Registrarse*
Actores principales:	Declarante
Escenario principal de éxito: 1. El sistema muestra la pantalla “Registro_Usuario” con el formulario del mismo nombre. 2. El declarante introduce los datos (todos son obligatorios, salvo la fotografía) y selecciona Entrar. 3. El sistema comprueba que los datos son correctos y los guarda. Asigna al usuario el perfil Declarante. 4. El sistema confirma el email. 5. El sistema muestra la pantalla “Dashboard”.	

Escenarios alternativos (extensiones): 2a. El declarante selecciona Cancelar. 2a1. Fin del caso de uso. 3a. El declarante introduce datos incorrectos o deja campos en blanco. Otros posibles errores son: la contraseña y su confirmación no coinciden, o el documento de identidad o el email ya existen en el sistema. 3a1. El sistema muestra la pantalla “Registro_Usuario” con el error y resalta los campos implicados. 3a2. Vuelve al paso 2. 3b. El declarante ya existía porque había realizado alguna autoliquidación sin autenticarse previamente. 3a1. El sistema actualiza el perfil de usuario, guarda la contraseña y vuelve al paso 4.
Pantallas: • **Registro_Usuario** Datos solicitados: Nacionalidad, Tipo de documento de identidad, nº del documento, Nombre y apellidos, tipo de vía, nombre de la vía, código postal, provincia, población, email, teléfono, fotografía, contraseña y confirmación de contraseña.

Tabla 5. Caso de uso Registrarse

3.2.4. Tarea *Confirmar email*

Esta tarea aparece en los casos de uso Autoliquidar Tasa T0 y en Registrarse.

Nombre:	*Tarea Confirmar email*
Actores principales:	Sistema
Escenario principal de éxito: 1. El sistema muestra la pantalla modal “Confirmación_email” y envía un email con un link que incluye el código de verificación. 2. El usuario consulta su correo (fuera de esta aplicación) y pincha en el link 3. El sistema recibe la petición http con el código de verificación, que coteja con el almacenado en el perfil de usuario, y marca en el perfil del declarante el check “email comprobado”. 4. El sistema actualiza la pantalla modal “Confirmación_email” con un mensaje de comprobación finalizada con éxito y botón Aceptar.	
Escenarios alternativos (extensiones): 2a. El declarante no pulsa el enlace en su correo en el plazo de 48 horas. 2a1. El sistema elimina el registro de usuario (tarea programada). 2a2. Fin del caso de uso donde se ejecute esta tarea.	
Pantallas: • **Confirmación_email** Datos mostrados: Mensaje al Declarante con las instrucciones para que encuentre el correo electrónico de verificación y pulse en el link que contiene. No hay botón Aceptar ni Cancelar. Cuando el sistema recibe la petición http con el código de verificación, actualiza el mensaje en la pantalla modal.	

Tabla 6. Tarea Confirmar email

3.2.5. Caso de uso *Consultar autoliquidación*

Permite al usuario seleccionar una autoliquidación concreta y consultar su detalle. El perfil de administrativo también puede añadir información y adjuntos a una autoliquidación concreta.

Nombre:	*Consultar autoliquidación*
Actores principales:	Declarante y Administrativo
Precondición:	Tanto el declarante como el administrativo deben autenticarse en el sistema.

Escenario principal de éxito:

1. El sistema muestra la pantalla “Dashboard”, que consta de los paneles menú y general.
2. El usuario selecciona la acción Autoliquidaciones del panel menú.
3. El sistema muestra en el panel general la pantalla “Listado_autoliquidaciones”.
4. El usuario interactúa con el formulario de filtro del listado.
5. El sistema actualiza el listado en cada cambio del formulario de filtro.
6. Vuelve al paso 4 hasta que el usuario selecciona una autoliquidación.
7. El sistema muestra en el panel general la pantalla “Detalle_autoliquidación”.
8. El declarante selecciona volver. El administrativo puede seleccionar volver o modificar.
9. En caso de volver, el sistema vuelve al paso 1.

Escenarios alternativos (extensiones):

2a. El usuario selecciona Salir u otra acción del panel menú del Dashboard.
2a1. Fin del caso de uso.

2b. El administrativo no hace ninguna acción en su Dashboard.
2b1. El panel general por defecto para este usuario es la pantalla “listado_autoliquidaciones”.
2b2. Vuelve al paso 4.

4a. El usuario no realiza ningún filtro.
4a1. Vuelve al paso 6.

Pantallas:

- **Dashboard.** Descrita en el CU Autenticarse (Login)
- **Listado_autoliquidaciones**

  Datos solicitados (filtro) – perfil declarante: Año. Opcional.

  Datos solicitados (filtro) – perfil administrativo: Año, mes, día, Importe desde y hasta, declarante (nombre y apellidos y documento), embarcación (todos los campos de la misma). Todos son opcionales.

  Datos mostrados – perfil declarante: fecha, embarcación (nombre, matrícula, eslora, manga), importe.

  Datos mostrados – perfil administrativo: Ídem que para el declarante, documento y nombre del declarante.

- **Detalle_autoliquidación**

  Detallado en CU Autoliquidar Tasa T0

Tabla 7. Caso de uso Consultar autoliquidación

3.2.6. Caso de uso *Gestionar autoliquidación*

Permite al administrativo modificar una autoliquidación para añadir los datos de pago o exención. El sistema actualiza el estado de la misma. Extensión del CU Consultar Autoliquidación.

Nombre:	*Gestionar autoliquidación*
Actores principales:	Administrativo
Precondición:	El administrativo debe autenticarse en el sistema.

Escenario principal de éxito:

1. El Administrativo consulta una autoliquidación y selecciona modificar.
2. El sistema muestra la pantalla “Detalle_Autoliquidación” en modo edición y solicita al administrativo unos datos u otros según el estado y el resultado de la autoliquidación:
   a) Estado “pendiente”, “expirado” o “email no confirmado” y el resultado es un importe: El administrativo introduce la fecha y el importe del ingreso recibido, y adjunta el documento de factura.
   b) Estado “pendiente” y el resultado es exenta: El administrativo introduce un motivo de exención y adjunta un certificado de exención.
3. El administrativo introduce la información correspondiente y selecciona Guardar.
4. El sistema guarda los cambios y actualiza el estado de la autoliquidación:
   a) En el caso 2a) pasa a “Pagado”.
   b) En el caso 2b) pasa a “Exento”.

Escenarios alternativos (extensiones):

3a. El Administrativo selecciona Cancelar.
  3a1. Vuelve al paso 1.

3b. El sistema detecta información incorrecta o algún campo obligatorio vacío. Pop-up con indicación del error.
  3b1. El usuario cierra el pop-up.
  3b2. Vuelve al paso 2, con los campos resaltados para localizarlos fácilmente.

Pantallas:

- **Dashboard.** Descrita en el CU Autenticarse (Login)
- **Detalle_autoliquidación.** Detallado en CU Autoliquidar Tasa T0

3.2.7. Resto de casos de uso

Los anteriores casos de uso derivaban del CU Autoliquidar Tasa B0, por lo que también se ha realizado una descripción completa de cada uno de ellos. Del resto de casos de uso detectados y mostrados en la figura 6 se realiza una descripción menos detallada.

Tarea *Adjuntar documento*

Permite a un administrativo adjuntar un documento, como la factura de pago o el certificado de exención, al detalle de una autoliquidación en modo edición.

El declarante también puede adjuntar fotografías al registro de embarcación o a su perfil personal.

Escenario de éxito: El administrativo selecciona añadir nuevo adjunto y el sistema muestra el pop-up “Adjuntar”. El usuario selecciona el tipo de documento y el archivo (mediante el cuadro de diálogo Abrir del sistema operativo), y el sistema lo guarda para esta autoliquidación.

Para el declarante no hay un pop-up “Adjuntar”, puesto que solo puede adjuntar documentos de tipo fotografía por lo que, al seleccionar añadir nuevo adjunto, el sistema muestra directamente el cuadro de diálogo Abrir. El usuario elige una foto y el sistema la guarda, con tipo de documento fotografía, para el registro correspondiente (perfil de usuario o registro de embarcación).

Si el tipo de documento es factura, el sistema oculta para el declarante el archivo documento de pago adjunto a la autoliquidación, y si es certificado de exención, el sistema oculta al declarante el documento de exención.

Pantallas: Dashboard, Detalle_autoliquid., Registro_Usuario, Registro_Embarcación y Adjuntar.

Caso de uso *Modificar Perfil*

Permite al declarante modificar los datos personales registrados, salvo el tipo y número de documento de identidad y el nombre y apellidos.

Escenario de éxito: El declarante se autentica en el sistema y selecciona la acción Perfil del panel menú de su dashboard. El sistema actualiza el panel general con la pantalla “Registro_Usuario”. El declarante modifica los datos y selecciona Guardar (también puede Cancelar). El sistema actualiza los datos del declarante. Las autoliquidaciones anteriores no se actualizan con los nuevos datos.

Pantallas: Dashboard y Registro_Usuario.

Caso de uso *Consultar embarcación*

Permite al declarante consultar sus embarcaciones registradas y al administrativo consultar y filtrar el listado de embarcaciones de todos los declarantes.

Escenario de éxito: El usuario se autentica en el sistema y selecciona la acción Embarcaciones del panel menú de su dashboard. El sistema actualiza el panel general con la pantalla “Listado_embarcaciones”.

El declarante puede registrar una embarcación. El administrativo puede filtrar las embarcaciones por cualquier campo y el sistema actualiza el listado en cada cambio.

Cualquiera de ambos tipos de usuario puede seleccionar una embarcación, y el sistema muestra la pantalla “Detalle_Embarcación” en el panel general.

Pantallas: Dashboard, Listado_Embarcaciones,
Detalle_Embarcación:

- Perfil Declarante: Muestra los datos de la embarcación en un panel superior, y en un panel inferior muestra un listado de las autoliquidaciones de esta embarcación. Acciones:
  - Volver. Regresa a la pantalla de bienvenida del Dashboard.
  - Modificar. El declarante puede cambiar datos de la ficha de la embarcación.
  - Baja. El declarante indica que la embarcación se ha dado de baja en Capitanía.
  - Cambio de titular. El declarante indica la venta de la embarcación.
- Perfil Administrativo: Además de los datos y acciones para el perfil declarante, el administrativo puede seleccionar Versiones de embarcación. Mediante esta acción, se muestra la pantalla "Listado_embarcaciones" con el historial de cambios de esta embarcación.

Caso de uso *Registrar Embarcación*

Permite al declarante añadir una embarcación nueva de su propiedad.
El sistema muestra la pantalla "Registrar_Embarcación" en la que solicita los datos de la misma y el porcentaje de titularidad. El declarante introduce los datos y selecciona Guardar. El sistema guarda la nueva embarcación o, si ya estaba registrada, guarda una nueva versión de su ficha y actualiza, si es el caso, el cambio en el porcentaje de titularidad.

Caso de uso *Gestionar Embarcaciones*

Permite al declarante y al administrativo modificar los datos de una embarcación y el porcentaje de titularidad, lo que crea una nueva versión de la ficha.
Extiende al CU Consultar embarcación cuando el usuario selecciona Modificar en la pantalla "Detalle_Embarcación". El sistema muestra la pantalla "Registro_Embarcación" con los datos modificables (todos salvo el NIB). En este punto incluye al CU Registrar embarcación.
Este será un paso necesario en caso de que el declarante deba realizar una autoliquidación complementaria por error en los datos de la embarcación.

Caso de uso *Consultar Declarantes*

El administrativo puede consultar los declarantes de autoliquidaciones.
Una vez autenticado, el administrativo selecciona la acción Declarantes del panel menú de su dashboard. El sistema muestra la pantalla "Listado_Declarantes" en el panel general. En la parte superior del listado puede realizar un filtro por nombre o documento de identidad. El sistema actualiza el listado cada vez que hay un cambio en el filtro.

Caso de uso *Gestionar Parámetros T0*

El administrativo puede añadir los parámetros para el cálculo de esta tasa para cada vez que entre en vigor un cambio de los mismos en los Presupuestos Generales del Estado.

El administrativo selecciona la acción Parámetros T0 del panel menú de su dashboard. El sistema muestra la pantalla “Listado_Parámetros” en el panel general. El usuario selecciona Añadir y el sistema muestra la pantalla “Detalle_Parámetro”. El usuario introduce los datos y el sistema los guarda.

El sistema sabe cuál es el conjunto de parámetro vigente para aplicar a las nuevas autoliquidaciones porque no tiene fecha de fin, en caso de que sea una nueva autoliquidación para el año actual. Si se trata de una autoliquidación para un año anterior, el sistema busca los parámetros que estaban vigentes en esa fecha.

3.3. Diagrama de clases

Del análisis de los casos de uso se detectan las siguientes entidades que forman parte del dominio y que serán representadas por clases.

Usuarios – Clase `User`

Cada usuario tendrá nombre, documento de identidad, tipo de documento (CIF, NIF, NIE, Pasaporte), email y contraseña. Puede haber tipos de usuario, al menos dos: declarante y administrativo. Además, un administrativo puede tener embarcación y tener que pagar la T0, por lo que también puede ser declarante.

De cada usuario también interesa guardar otros datos de su perfil, como la dirección postal completa, teléfono y una fotografía. De la dirección hay datos que siempre se repiten, como el tipo de vía y la provincia, por lo que se crearán clases para estos datos.

Embarcaciones – Clase `Boat`

De cada embarcación hay que guardar el NIB, nombre, matrícula, tipo (lista y pesquero o recreo), propulsión (motor o vela), eslora, manga, puerto base y la fecha del último pago de la T0.

De las embarcaciones hay que guardar todos los cambios de su ficha como versiones de embarcación, por lo que, empleando el patrón de análisis Asociación Histórica, debe haber una clase de fecha que indique el inicio y el fin de cada versión.

Por último, el porcentaje de titularidad de cada usuario sobre cada embarcación es un dato de la relación entre ambos, de la que también se guardará información histórica.

Adjuntos – Clase `Attachment`

Los usuarios, embarcaciones y autoliquidaciones pueden tener documentos adjuntos. Estos documentos pueden ser de varios tipos, como fotografía, documento de pago, factura, documento de exención o certificado de exención.

Parámetros T0 – Clase `T0Parameter`

El sistema debe guardar los parámetros para el cálculo del importe de las autoliquidaciones, como son las cuantías básicas A y B y los coeficientes para cada caso que establece la LPEMM, y que son revisados por los Presupuestos Generales del Estado.

Autoliquidación T0 – Clase `T0Fee`

Es la entidad objeto de esta aplicación. Para cada autoliquidación deben guardarse todos los datos del declarante, de la embarcación y de los parámetros, así como la fecha de registro y de pago, y su estado (email no confirmado, pendiente, expirada, pagada o exenta). Se aplica el patrón de diseño Estado para representarlo.

También debe haber un marcador que indique si es complementaria y de qué autoliquidación anterior.

Teniendo presente todo lo anterior, este es el diagrama estático de análisis resultante.

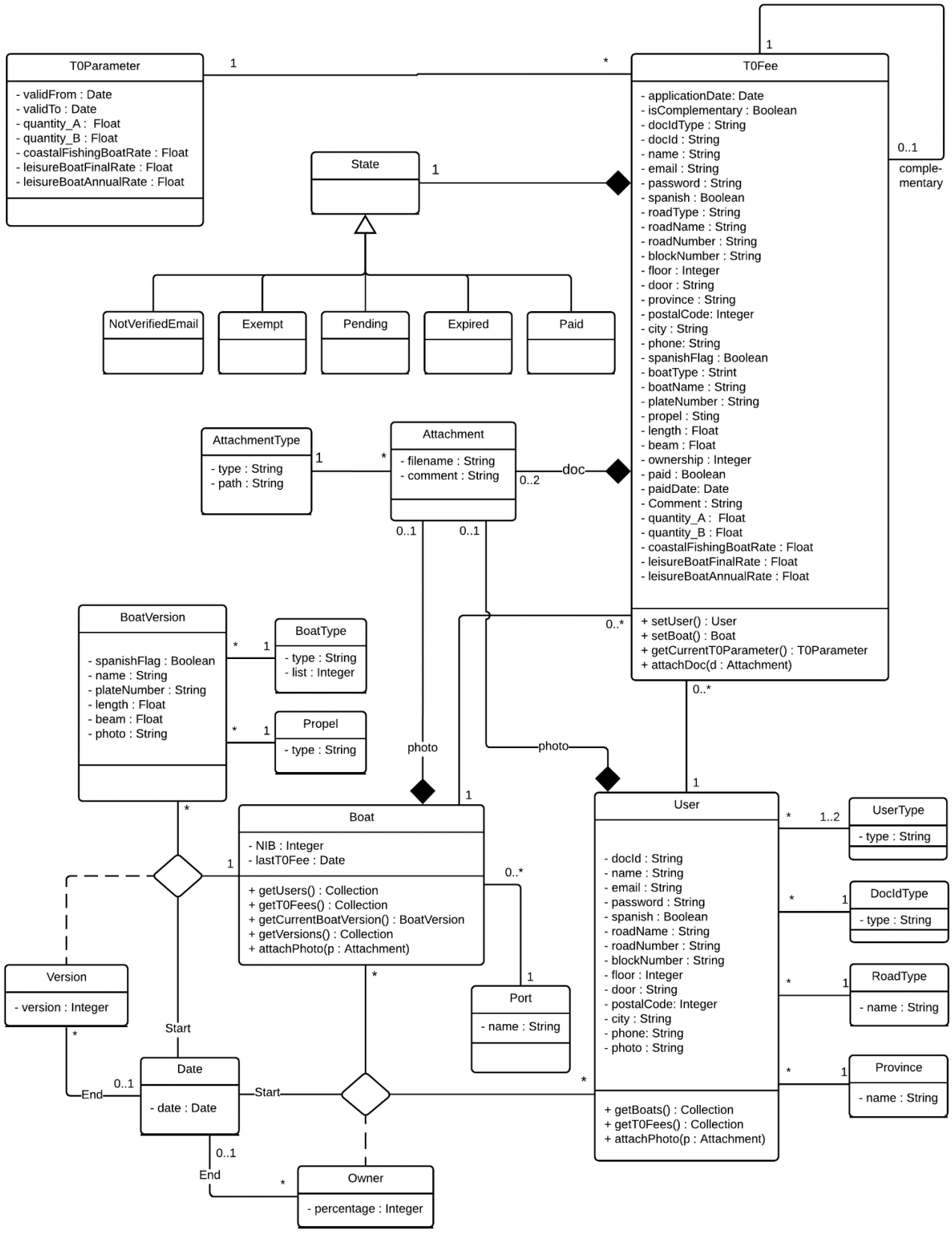


Figura 7. Diagrama de Clases

En la fase de producción, durante la creación de migraciones surgieron ciertos contratiempos con del diagrama de clases presentado, como la implementación de los patrones de diseño historial y estado que, si bien suponen una mejora en la lógica de la aplicación a nivel de modelos, acarrean la creación de tablas innecesarias en la base de datos asociadas a dichos modelos. Todo esto supuso el retraso en la planificación de la creación de migraciones reflejado en el punto 1.4.4.

Finalmente, del compromiso entre simplificar la lógica a nivel de modelo y mantener el menor número de tablas posible, así como de la optimización del diagrama anterior y de la adición de nuevos parámetros y funcionalidades, se ha obtenido el diagrama de la siguiente figura.

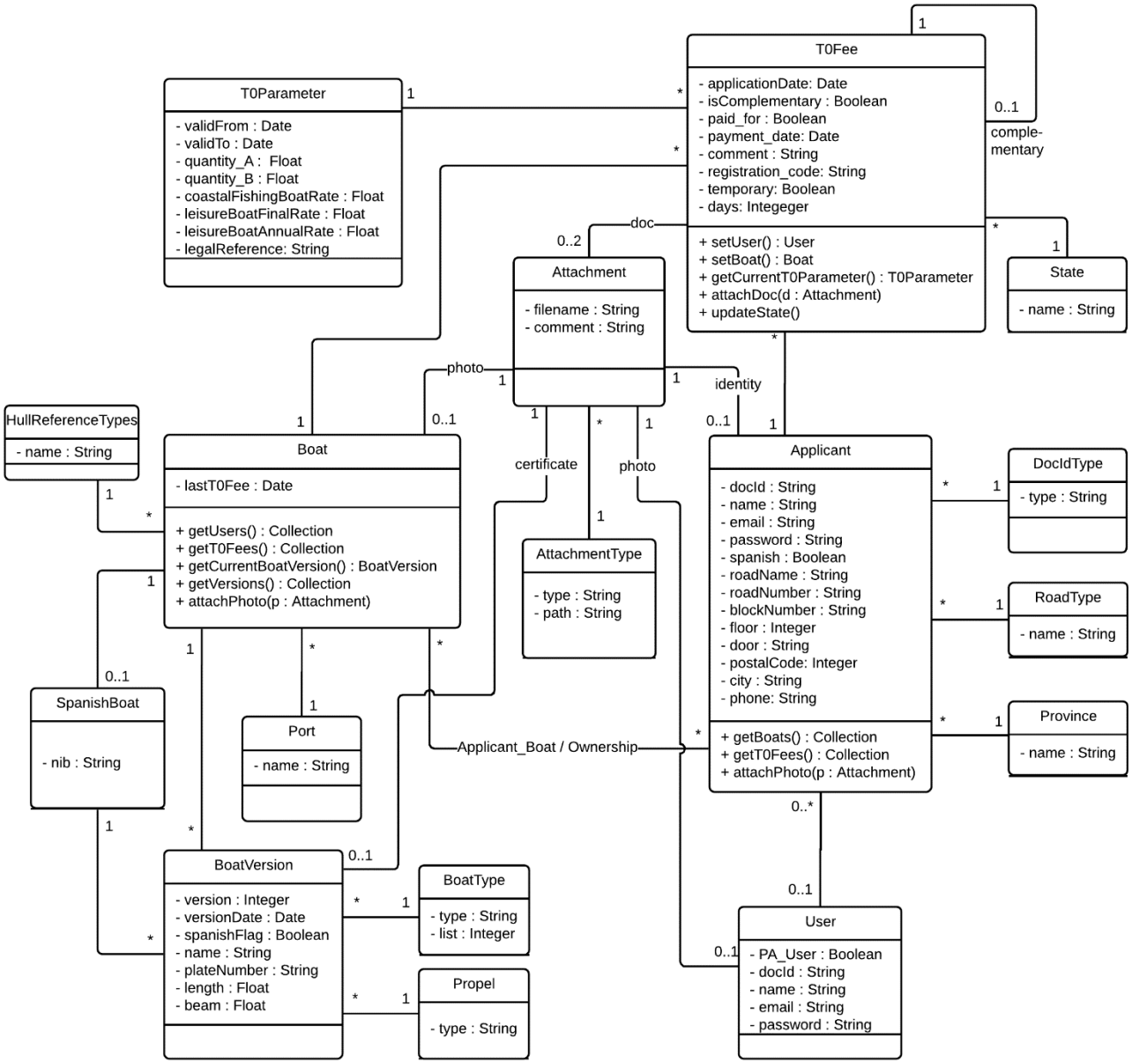


Figura 8. PEC3: Nuevo Diagrama UML de Clases

La relación applicant_boat tiene una tabla pivote de relación n..m, y esta no tiene clase asociada.

Como se explica en la planificación y en los anexos 3 y 4, el desarrollo de software es un proceso cíclico de todas las fases del proyecto, que implica cambios en cada una de ellas. El modelo de clases es una parte fundamental, y aunque se ha tratado de respetar la idea inicial, determinadas decisiones han influido en su diseño final, siempre para una simplificación del mismo.

Puesto que no se espera que esta aplicación vaya a aceptar ningún otro tipo de documento adjunto, se prescinde de las clases `Attachment` y `AttaccmentType`, y se incluye un atributo en las clases que tenían relacionadas para guardar la ruta al archivo en el servidor.

Se añade una relación entre `T0Fee` y `BoatVersión`, y entre esta y `SpanishBoat`.

Se actualizan los atributos y métodos de cada clase al cierre del desarrollo de este proyecto.

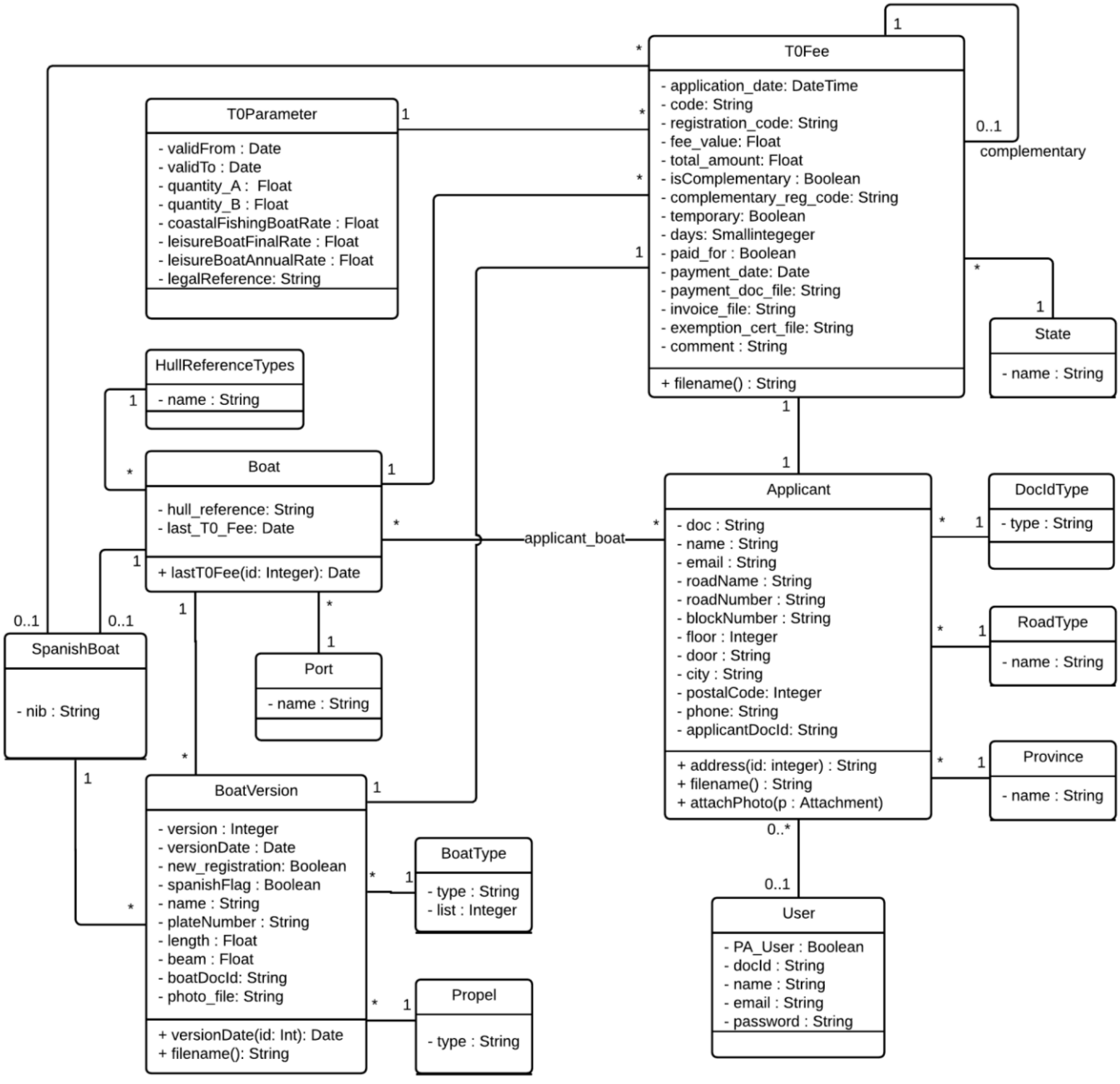


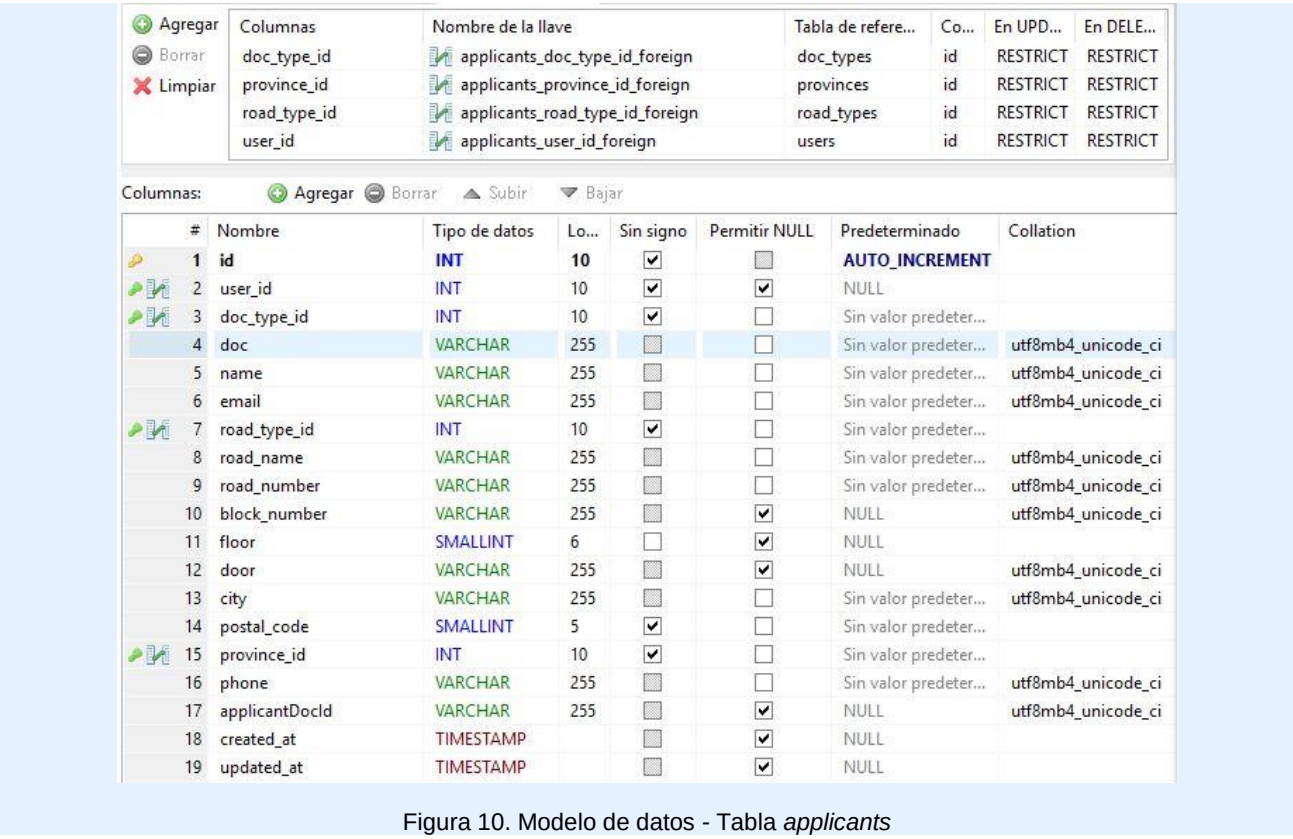
Figura 9. Diagrama UML de Clases final

4. Modelo de datos

Como se ha explicado anteriormente, la aplicación objeto de este proyecto almacenará la información en una base de datos de MySQL. Puesto que la lógica de negocio se desarrollará siguiendo el paradigma de la orientación a objetos, para no perder la información manejada por estos objetos deberá ser almacenada en la base de datos, por lo que resulta necesario el empleo de un ORM que nos asista en la persistencia de esta información.

A este respecto, el framework Laravel ofrece Eloquent, su ORM propio que facilita las operaciones CRUD (*Create*, *Read*, *Update* and *Delete*) en la base de datos con la información de los objetos. Por tanto, Eloquent establece un enlace entre los modelos de Laravel que definen las clases estudiadas en el punto anterior, y las tablas del modelo de datos de MySQL.
Por cada modelo debe existir una tabla con el mismo nombre (en plural, según la convención seguida por este framework), y los campos de cada tabla deben coincidir con el atributo de la clase y su tipo de dato, de forma que cada registro de una tabla coincida con un objeto de la clase.

Por otro lado, Laravel permite gestionar las tablas de la base de datos a través de las migraciones, que se estudian en el capítulo 7.2. de esta memoria. De la ejecución de estas migraciones se establecen las siguientes tablas en la base de datos de MySQL.


Agregar
Borrar
Limpiar

Columnas	Nombre de la llave	Tabla de refere...	Co...	En UPD...	En DELE...
doc_type_id	applicants_doc_type_id_foreign	doc_types	id	RESTRICT	RESTRICT
province_id	applicants_province_id_foreign	provinces	id	RESTRICT	RESTRICT
road_type_id	applicants_road_type_id_foreign	road_types	id	RESTRICT	RESTRICT
user_id	applicants_user_id_foreign	users	id	RESTRICT	RESTRICT

Columnas: Agregar Borrar Subir Bajar

#	Nombre	Tipo de datos	Lo...	Sin signo	Permitir NULL	Predeterminado	Collation
1	id	INT	10	☑	☐	AUTO_INCREMENT	
2	user_id	INT	10	☑	☑	NULL	
3	doc_type_id	INT	10	☑	☐	Sin valor predeter...	
4	doc	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
5	name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
6	email	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
7	road_type_id	INT	10	☑	☐	Sin valor predeter...	
8	road_name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
9	road_number	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
10	block_number	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
11	floor	SMALLINT	6	☐	☑	NULL	
12	door	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
13	city	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
14	postal_code	SMALLINT	5	☑	☐	Sin valor predeter...	
15	province_id	INT	10	☑	☐	Sin valor predeter...	
16	phone	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
17	applicantDocId	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
18	created_at	TIMESTAMP		☐	☑	NULL	
19	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 10. Modelo de datos - Tabla *applicants*

Agregar
Borrar
Limpiar

Columnas	Nombre de la llave	Tabla de referencia	Columnas forá...	En UPDATE	En DELETE
applicant_id	applicant_boat_applicant_i...	applicants	id	RESTRICT	RESTRICT
boat_id	applicant_boat_boat_id_for...	boats	id	RESTRICT	RESTRICT

Columnas: Agregar Borrar Subir Bajar

#	Nombre	Tipo de datos	Longitud...	Sin signo	Permitir NULL	Predeterminado	Collation
1	applicant_id	INT	10	☑	☐	Sin valor predeter...	
2	boat_id	INT	10	☑	☐	Sin valor predeter...	
3	percentage	TINYINT	3	☑	☐	Sin valor predeter...	
4	application_date	DATETIME		☐	☐	2018-05-11 13:05:48	
5	created_at	TIMESTAMP		☐	☑	NULL	
6	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 11. Modelo de datos - Tabla *applicant_boat*

Agregar
Borrar
Limpiar

Columnas	Nombre de la llave	Tabla de referencia	Co...	En UPD...	En DELE...
attachment_id	boats_attachment_id_foreign	attachments	id	RESTRICT	RESTRICT
hull_reference_type_id	boats_hull_reference_type_id_forei...	hull_reference_types	id	RESTRICT	RESTRICT
port_id	boats_port_id_foreign	ports	id	RESTRICT	RESTRICT
spanish_boat_id	boats_spanish_boat_id_foreign	spanish_boats	id	RESTRICT	RESTRICT

Columnas: Agregar Borrar Subir Bajar

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	hull_reference_typ...	INT	10	☑	☐	Sin valor predeter...	
3	hull_reference	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
4	spanish_boat_id	INT	10	☑	☑	NULL	
5	last_T0_fee	DATE		☐	☑	NULL	
6	port_id	INT	10	☑	☐	Sin valor predeter...	
7	attachment_id	INT	10	☑	☑	NULL	
8	created_at	TIMESTAMP		☐	☑	NULL	
9	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 12. Modelo de datos – Tabla *boats*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	type	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	legal_reference	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
4	created_at	TIMESTAMP		☐	☑	NULL	
5	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 13. Modelo de datos - Tabla *boat_types*

Agregar
Borrar
Limpiar

Columnas	Nombre de la llave	Tabla de refere...	Co...	En UPD...	En DELE...
boat_id	boat_versions_boat_id_foreign	boats	id	RESTRICT	RESTRICT
boat_type_id	boat_versions_boat_type_id_foreign	boat_types	id	RESTRICT	RESTRICT
propel_id	boat_versions_propel_id_foreign	propels	id	RESTRICT	RESTRICT
spanish_boat_id	boat_versions_spanish_boat_id_foreign	spanish_boats	id	RESTRICT	RESTRICT

Columnas: Agregar Borrar Subir Bajar

#	Nombre	Tipo de datos	Lo...	Sin signo	Permitir NULL	Predeterminado	Collation
1	id	INT	10	☑	☐	AUTO_INCREMENT	
2	boat_id	INT	10	☑	☐	Sin valor predeter...	
3	boat_type_id	INT	10	☑	☐	Sin valor predeter...	
4	version	SMALLINT	5	☑	☐	Sin valor predeter...	
5	version_date	DATE		☐	☐	Sin valor predeter...	
6	new_registration	TINYINT	1	☐	☐	0	
7	spanish_flag	TINYINT	1	☐	☐	0	
8	spanish_boat_id	INT	10	☑	☑	NULL	
9	name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
10	plate_number	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
11	propel_id	INT	10	☑	☐	Sin valor predeter...	
12	length	DOUBLE	8,2	☐	☐	Sin valor predeter...	
13	beam	DOUBLE	8,2	☐	☐	Sin valor predeter...	
14	boatDocId	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
15	photo_file	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
16	created_at	TIMESTAMP		☐	☑	NULL	
17	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 14. Modelo de datos - Tabla *boat_versions*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	id	INT	10	☑	☐	AUTO_INCREMENT	
2	type	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	created_at	TIMESTAMP		☐	☑	NULL	
4	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 15. Modelo de datos - *doc_types*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	id	INT	10	☑	☐	AUTO_INCREMENT	
2	type	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	description	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
4	legal_reference	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
5	created_at	TIMESTAMP		☐	☑	NULL	
6	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 16. Modelo de datos - Tabla *hull_reference_types*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	id	INT	10	☑	☐	AUTO_INCREMENT	
2	migration	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	batch	INT	11	☐	☐	Sin valor predeter...	

Figura 17. Modelo de datos - Tabla *migrations*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	email	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
2	token	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	created_at	TIMESTAMP		☐	☑	NULL	

Figura 18. Modelo de datos - Tabla *password_resets*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	created_at	TIMESTAMP		☐	☑	NULL	
4	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 19. Modelo de datos - Tabla *ports*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	type	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	t0_length	DOUBLE	8,2	☐	☐	Sin valor predeter...	
4	created_at	TIMESTAMP		☐	☑	NULL	
5	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 20. Modelo de datos - Tabla *propels*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	created_at	TIMESTAMP		☐	☑	NULL	
4	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 21. Modelo de datos - Tabla *provinces*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	type	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	created_at	TIMESTAMP		☐	☑	NULL	
4	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 22. Modelo de datos - Tabla *road_types*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	nib	MEDIUMINT	8	☑	☐	Sin valor predeter...	
3	created_at	TIMESTAMP		☐	☑	NULL	
4	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 23. Modelo de datos - Tabla *spanish_boats*

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	created_at	TIMESTAMP		☐	☑	NULL	
4	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 24. Modelo de datos - Tabla *states*

Agregar
Borrar
Limpiar

Columnas	Nombre de la llave	Tabla de refere...	Co...	En UPD...	En DELE...
applicant_id	t0_fees_applicant_id_foreign	applicants	id	RESTRICT	RESTRICT
boat_id	t0_fees_boat_id_foreign	boats	id	RESTRICT	RESTRICT
boat_version_id	t0_fees_boat_version_id_foreign	boat_versions	id	RESTRICT	RESTRICT
complementary_fee_id	t0_fees_complementary_fee_id_foreign	t0_fees	id	RESTRICT	RESTRICT
spanish_boat_id	t0_fees_spanish_boat_id_foreign	spanish_boats	id	RESTRICT	RESTRICT
state_id	t0_fees_state_id_foreign	states	id	RESTRICT	RESTRICT
t0_parameter_id	t0_fees_t0_parameter_id_foreign	t0_parameters	id	RESTRICT	RESTRICT

Columnas: Agregar Borrar Subir Bajar

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	id	INT	10	☑	☐	AUTO_INCREMENT	
2	t0_parameter_id	INT	10	☑	☐	Sin valor predeter...	
3	applicant_id	INT	10	☑	☐	Sin valor predeter...	
4	boat_id	INT	10	☑	☐	Sin valor predeter...	
5	boat_version_id	INT	10	☑	☐	Sin valor predeter...	
6	spanish_boat_id	INT	10	☑	☑	NULL	
7	state_id	INT	10	☑	☐	Sin valor predeter...	
8	application_date	DATETIME		☐	☐	2018-05-11 13:05:48	
9	code	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
10	registration_code	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
11	is_complementary	TINYINT	1	☐	☐	0	
12	complementary_f...	INT	10	☑	☑	NULL	
13	paid_for	TINYINT	1	☐	☐	0	
14	payment_date	DATE		☐	☑	NULL	
15	comment	MEDIUMTEXT		☐	☑	Sin valor predeter...	utf8mb4_unicode_ci
16	created_at	TIMESTAMP		☐	☑	NULL	
17	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 25. Modelo de datos - Tabla *t0_fees*

```
BEGIN
   DECLARE last_application_date date;
   DECLARE last_code integer;
   SET @last_application_date := (SELECT application_date FROM t0_fees ORDER BY application_date DESC LIMIT 1);

      IF (   ( (SELECT COUNT(*) FROM t0_fees)=0 ) OR (NEW.application_date > @last_application_date)   ) THEN
         SET @last_code = 0;

      ELSE # NEW.application_date <= @last_application_date)
         SET @last_code := (
            SELECT code FROM t0_fees
            WHERE application_date IN (
               SELECT application_date FROM t0_fees
               WHERE application_date = NEW.application_date
            )
            ORDER BY id DESC LIMIT 1
         );

         IF ( ISNULL(@last_code) ) THEN
            SET @last_code = 0;
         END IF;

      END IF;
SET NEW.code = LPAD(@last_code+1, 4, '0');
SET NEW.registration_code = CONCAT('T0-', DATE_FORMAT(NEW.application_date, '%Y%m%d'), '-', NEW.code);
END
```

Figura 26. Modelo de datos - Disparador *t0_fees_before_insert*

Este disparador (*trigger*) se ejecuta antes de cada inserción de un nuevo registro en la tabla `t0_fees`. Su objetivo es generar el código de registro para cada solicitud de autoliquidación, que tiene el formato `T0-YYYYMMDD-XXXX`, donde `YYYYMMDD` es la fecha de la nueva solicitud y `XXXX` es un número entero que se autoincrementa para cada fecha, y vuelve a comenzar por 1 para cada fecha nueva.

Finalmente, este *trigger* es eliminado. Se genera el código de registro en la lógica de la aplicación.

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	legal_reference	VARCHAR	255	☐	☑	NULL	utf8mb4_unicode_ci
3	validFrom	DATE		☐	☐	Sin valor predeter...	
4	validTo	DATE		☐	☑	NULL	
5	quantity_A	DOUBLE	8,2	☐	☐	Sin valor predeter...	
6	quantity_C	DOUBLE	8,2	☐	☐	Sin valor predeter...	
7	coastalFishingBoa...	DOUBLE	8,2	☐	☐	Sin valor predeter...	
8	leisureBoatFinalRate	DOUBLE	8,2	☐	☐	Sin valor predeter...	
9	leisureBoatAnnual...	DOUBLE	8,2	☐	☐	Sin valor predeter...	
10	created_at	TIMESTAMP		☐	☑	NULL	
11	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 27. Modelo de datos - Tabla *t0_parameters*

Agregar
Borrar
Limpiar

Columnas	Nombre de la llave	Tabla de refere...	Co...	En UPD...	En DELE...
attachment_id	users_attachment_id_foreign	attachments	id	RESTRICT	RESTRICT

Columnas: Agregar Borrar Subir Bajar

#	Nombre	Tipo de datos	Longitu...	Sin signo	Permitir NULL	Predeterminado	Collation
1	**id**	**INT**	**10**	☑	☐	**AUTO_INCREMENT**	
2	doc	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
3	name	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
4	email	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
5	password	VARCHAR	255	☐	☐	Sin valor predeter...	utf8mb4_unicode_ci
6	attachment_id	INT	10	☑	☑	NULL	
7	civil_servant	TINYINT	1	☐	☐	0	
8	remember_token	VARCHAR	100	☐	☑	NULL	utf8mb4_unicode_ci
9	created_at	TIMESTAMP		☐	☑	NULL	
10	updated_at	TIMESTAMP		☐	☑	NULL	

Figura 28. Modelo de datos - Tabla *users*

5. Usabilidad / DCU

Esta aplicación nace como un intento de simplificar el proceso de autoliquidación de la Tasa T0, de forma que se facilite al usuario el cálculo del importe según la casuística establecida en la LPEMM. Este es el objetivo que no hay que perder de vista en el desarrollo del apartado visual de la aplicación. Su interfaz debe ser intuitiva y transparente para el usuario, de forma que no entorpezca en ningún sentido el fin para el que el usuario ha iniciado este trámite.

Dado que esta aplicación se integrará dentro de una sede electrónica, se deben respetar los colores y tipografías de la misma para que el usuario, al acceder a la aplicación, perciba una continuidad con el resto del sitio web. Por tanto, se solicitará a la AP la guía de estilo de su sede para aplicarla en el desarrollo del front-end.

Siempre que sea posible se tendrá en cuenta diferenciar claramente los títulos del texto normal, así como usar fuentes de palo seco y con un contraste bueno con el color de fondo que favorezca su legibilidad. Los enlaces, botones y zonas activas estarán también claramente definidas para facilitar al usuario el manejo de la aplicación.

Respecto al funcionamiento de la aplicación, debe ser intuitivo, con metáforas e iconos claramente reconocibles y sin tiempos de espera en los que el usuario no sepa qué está sucediendo. Durante todo el proceso del trámite, el usuario debe ser guiado de forma que no tenga dudas sobre lo que debe realizar en cada apartado de la aplicación.

Es necesario que esta aplicación tenga un diseño coherente, claro, limpio y adaptativo al tamaño de pantalla del dispositivo con el que se utilizará, ya que hoy día es más habitual la navegación web desde dispositivos portátiles, como un smartphone o una tablet, que desde PC o laptop.
Debido a la extensión del formulario, solo se adapta su tamaño para PC o laptop. Queda para una revisión posterior adaptarlo para su visualización en pantallas de pequeño formato, como smartphones o tablets.

6. Configuración del entorno de desarrollo

Para el desarrollo de la aplicación objeto de este proyecto se requiere un servidor web con intérprete de PHP y un motor de MySQL que puedan correr sobre Windows 10 de 64 bits.

Se ha optado por utilizar la suite **Laragon** que, además de proporcionar los requisitos básicos, ofrece otras herramientas muy útiles como la terminal Cmder, HeidiSQL, xDebug, Composer, SSH, Putty, cambiar de versión de PHP, habilitar extensiones de PHP, creación instantánea de proyectos CMS y de Laravel, y la gestión automática de *virtual hosts*.

6.1. Instalación de Laragon

En su web oficial[7] se puede realizar la descarga de la última versión del paquete Laragon Wamp, que incluye todas las herramientas necesarias para el desarrollo de proyectos PHP.

Del proceso de instalación solo hay que destacar que hay que activar una opción para la creación automática de un virtual host para cada proyecto.

Una vez en ejecución, y sin necesidad de tener nociones de composer, desde el menú de Laragon se puede crear un proyecto de Laravel como se muestra en la siguiente figura. El asistente solicitará un nombre para el proyecto, que llamaremos TasaT0, que también se utilizará para crear la base de datos MySQL.

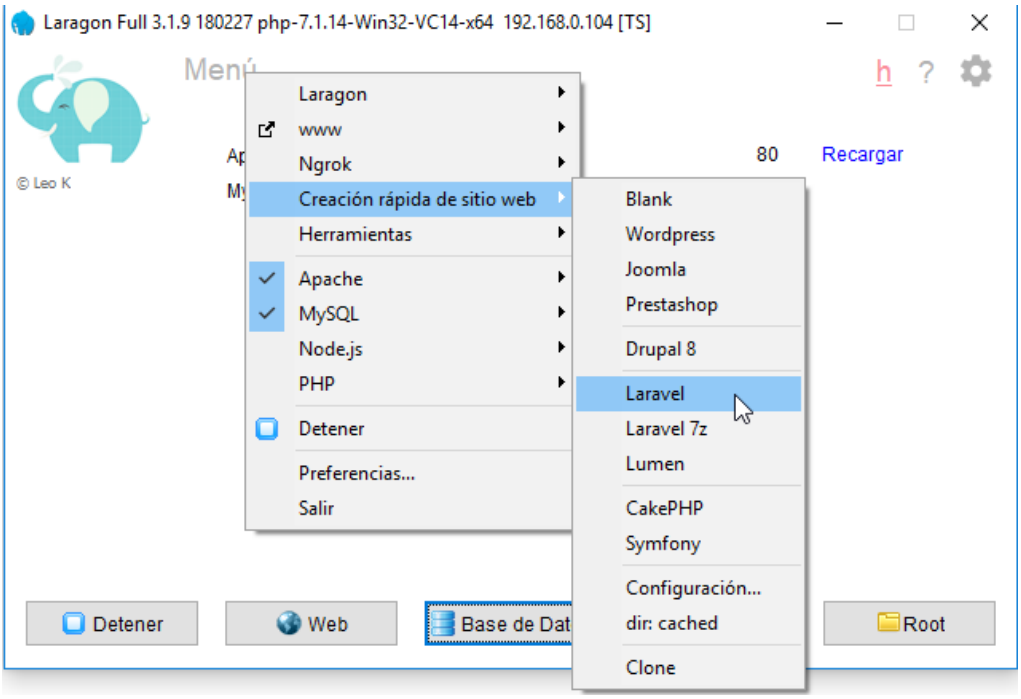


Figura 29. Creación de un proyecto Laravel en Laragon

[7] Web oficial de descarga de Laragon: https://laragon.org/download/

6.2. Estructura de directorios de un proyecto Laravel

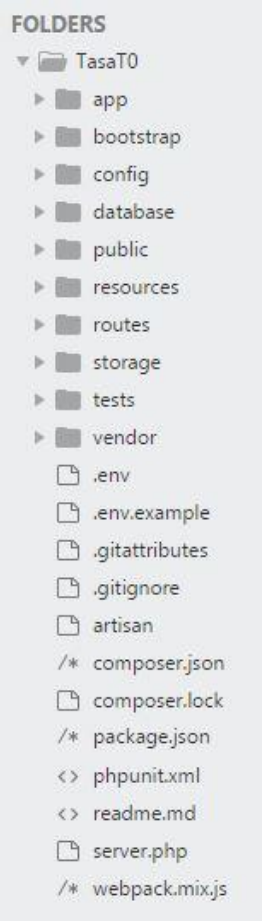


Figura 30. Directorios de un proyecto Laravel

La carpeta *www* de Laragon es la que se utiliza para publicar las webs en este servicio WAMP. Como se ha visto en el apartado anterior, se ha creado una carpeta `www\TasaT0` para que albergue el proyecto de Laravel.

La misión de cada carpeta en este framework está explicada en su web de documentación[8], por lo que aquí solo se nombrarán las que más se van a utilizar.

- Rutas: `TasaT0\routes`
- Modelos: `TasaT0\app`
- Vistas: `TasaT0\resources\views`
- Controladores: `TasaT0\app\Http\Controllers`
- Pruebas: `TasaT0\tests`
- Público: `TasaT0\public`

El directorio público es donde se sitúan los archivos y carpetas que se pueden acceder desde la web, como `index.php`, `.htaccess`, el código css o js.

Todo lo que queda fuera de esta carpeta `public` es privado, y solo accesible por la lógica del framework y de la aplicación.

6.3. Configuración de la base de datos

En primer lugar, hay que crear un usuario con privilegios en la base de datos asignada al proyecto. Para ello se utiliza el cliente HeidiSQL incluido en Laragon, desde el botón *Base de datos* en su panel.

Desde el Administrador de usuarios, se agrega un usuario nuevo al que llamaremos `AP_TasaT0` con contraseña `1234T0` y, a continuación, se añade la base de datos a la que se le permitirá el acceso, llamada `tasat0`, garantizándole todos los permisos.

Por último, para que la aplicación pueda acceder a la base de datos, hay que introducir las credenciales de este usuario en el archivo **`.env`** en la raíz del proyecto. También se aprovecha la edición de este archivo para indicar el nombre de la aplicación y su URL.

[8] Documentación de la Estructura de directorios de Laravel: https://laravel.com/docs/5.6/structure

```
APP_NAME=TasaT0
APP_URL=http://tasat0.test
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=tasat0
DB_USERNAME=AP_TasaT0
DB_PASSWORD=1234T0
```

En el siguiente capítulo se estudiarán las migraciones de Laravel. Ahora se ejecutan las dos migraciones que trae el framework por defecto para probar la conexión. Para ello se abre la consola Cmder mediante el botón Terminal del panel de Laragon.
Situados en la carpeta del proyecto, que por defecto está en `C:\laragon\www\TasaT0`, en el prompt de esta terminal, señalado por el símbolo **λ** precedido de la ruta, se introduce el siguiente comando para ejecutar las migraciones: `php artisan migrate`

```
C:\laragon\www\TasaT0
λ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
```

Ahora se puede comprobar en HeidiSQL que se han creado 3 tablas: `migrations`, `password_resets` y `users`, lo que indica que la aplicación ha podido conectar con la base de datos de MySQL y realizar las operaciones CRUD indicadas en las migraciones.

Sobre los comandos en la terminal, en este proyecto se empleará sobre todo las pruebas automatizadas, que se explican en el capítulo 7, y el comando `php artisan` con diversos argumentos. En el caso anterior, `migrate` se emplea para ejecutar las migraciones, que como se verá más adelante, sirven para establecer la estructura de la base de datos.

Entrando en la terminal `php artisan` sin argumentos, o bien `php artisan list`, se muestra una lista de todos los argumentos disponibles, así como una descripción de la función de cada uno. En la documentación de Laravel[9] puede ampliar más información.

[9] Documentación de Laravel sobre comandos de artisan: https://laravel.com/docs/5.6/artisan

7. Proceso de desarrollo

7.1. Migraciones, *seeders* y *model factories*

Las **migraciones**[10] en Laravel proporcionan un mecanismo para diseñar la estructura de la base de datos que soportará la información de la aplicación. Además, permite por un lado tener control sobre los cambios en dicha estructura, y por otro abstraer a la aplicación del motor de base de datos a utilizar ya que, al ejecutar las migraciones, el framework realizará las operaciones en la base de datos que se configuró en el archivo `.env`. En este caso se usa el motor de MySQL.

El directorio donde se guardan es `TasaT0\database\migrations`, y los archivos se nombran indicando fecha, hora, y nombre de la migración. Este último sigue una convención que indica también el tipo de migración. El *timestamp* en el nombre es importante, porque las migraciones se ejecutan por orden de antigüedad. Si se sigue la convención, al usar el comando `php artisan make:migration` el framework creará el archivo con la estructura básica, aunque también es posible no usar la convención si se usan las opciones del comando, que se pueden revisar en la terminal añadiendo la opción `-h` al comando anterior. En este proyecto se seguirá la convención porque el nombre final del archivo indica lo que se realiza en él.

Las migraciones son clases, con el mismo nombre que el archivo de migración, sin el *timestamp* y en formato *CamelCase*, que extienden la clase `Migration` del framework, y deben definir los métodos `up()` y `down()`, empleados al ejecutar la migración y al deshacerla, respectivamente.

Por ejemplo, la migración para crear la tabla de declarantes usa el mismo nombre del modelo (*Applicant*) para la tabla, pero en plural (*applicants*), y de aquí que se haya usado el inglés para nombrar los modelos, puesto que la construcción del plural en ese idioma es distinta de la del español.

En la figura 31 se puede observar que Laravel emplea el *Facade*[11] `Schema` y su método `create` para crear la tabla, representada por un objeto de la clase `Blueprint`. Hay que usar los métodos de dicho objeto para indicar el nombre y tipo de campos que incluirá la misma.
También se observa la definición de las claves foráneas, haciendo referencia al campo de la tabla con la que se vincula.

[10] Documentación de Laravel sobre migraciones: https://laravel.com/docs/5.6/migrations
[11] Documentación de Laravel sobre Facades: https://laravel.com/docs/5.6/facades

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateApplicantsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('applicants', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned()->nullable();  //FK to users table
            $table->integer('doc_type_id')->unsigned();  //FK to doc_types table
            $table->string('doc');
            $table->string('name');
            $table->string('email');
            $table->integer('road_type_id')->unsigned();  //FK to road_types table
            $table->string('road_name');
            $table->string('road_number');
            $table->string('block_number')->nullable();
            $table->smallInteger('floor')->nullable();
            $table->string('door')->nullable();
            $table->string('city');
            $table->smallInteger('postal_code')->unsigned();
            $table->integer('province_id')->unsigned();  //FK to provinces table
            $table->string('phone');
            $table->string('applicantDocId')->nullable()->default(null);
            $table->timestamps();

            //Restricciones de integridad de clave foránea (FK - Foreign Key)
            //Nota: Antes de esta migración deben ejecutarse las de las tablas
            //      a las que se hace referencia.
            $table->foreign('user_id')->references('id')->on('users');
            $table->foreign('doc_type_id')->references('id')->on('doc_types');
            $table->foreign('road_type_id')->references('id')->on('road_types');
            $table->foreign('province_id')->references('id')->on('provinces');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('applicants');
    }
}
```

Figura 31. Migración CreateApplicantsTable

En este punto hay que recordar que las migraciones se ejecutan por orden alfabético, establecido por el timestamp en su nombre.
Es importante que las tablas maestras con las que se crean restricciones de integridad existan antes de se ejecute la migración de la tabla `applicants`.

Para el desarrollo de este proyecto hay que añadir una migración por cada clase del modelo UML de la figura 9, y para los campos se emplearán los nombres de los atributos señalados.

Para ello se emplearán los siguientes comandos *Artisan*:

`php artisan make:migration nombre_migracion`	Crea una migración, con el nombre según la convención explicada
`php artisan migrate`	Ejecuta las migraciones y las registra el lote en la tabla migrations
`php artisan migrate:rollback`	Deshace solo el último lote de migraciones registradas en la tabla migrations. Elimina la información guardada en las tablas.
`php artisan migrate:reset`	Hace `rollback` de todos los lotes
`php artisan migrate:refresh`	Primero hace `migrate:reset` y luego `migrate`
`php artisan migrate:fresh`	elimina todas las tablas (`drop`) de la BB.DD., y luego hace `migrate`

Tabla 8. Comandos de Artisan para las migraciones

Una relación n..m entre dos tablas en la base de datos requiere de una tabla pivote que guarde los id de estas dos tablas. Esta tabla pivote puede además guardar otra información relacionada con la asociación. Es el caso de la relación entre *applicants* y *boats*. Puesto que no hay una entidad (clase) que represente esta asociación en el modelo UML, la tabla no puede crearse a partir de una migración de un modelo. Por tanto, se incluye la creación de esta tabla en la migración de la tabla *boats*.

Mediante las migraciones también se pueden modificar tablas ya existentes. En entornos en producción solo se deberían realizar migraciones de este tipo, como la utilizada para realizar la asociación reflexiva "*complementary*" en la tabla `t0_fees`.

Los ***seeders***[12] se emplean para añadir datos a las tablas maestras, como la tabla `provinces`, y también datos de prueba o ejemplo al resto de tablas. Son clases que extienden la clase `Seeder`.
Se guardan en `TasaT0\database\seeds` y hay que registrarlos en el archivo `DatabaseSeeder.php` en dicha ruta. Los comandos Artisan que se emplean son:

`php artisan make:seeder ModelSeeder`	Crea un *seeder*
`php artisan db:seed`	Ejecuta los *seeders*
`php migrate:fresh --seed`	Elimina las tablas, las vuelve a crear y ejecuta los *seeders.*

Tabla 9. Comandos de Artisan para los seeders

Para simplificar la adición de datos a las tablas, en primer lugar, se crea un modelo para cada tabla con el comando `php artisan make:model Nombre`, donde Nombre es el mismo que el de la tabla, en singular y en formato C*amelCase*. Los modelos se estudiarán en el capítulo 7.2.

```php
<?php

use App\RoadType;
use Illuminate\Database\Seeder;

class RoadTypeSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        RoadType::create(['type' => 'Calle']);
        RoadType::create(['type' => 'Plaza']);
        RoadType::create(['type' => 'Avenida']);
        RoadType::create(['type' => 'Carretera']);
        RoadType::create(['type' => 'Paraje']);
        RoadType::create(['type' => 'Otro']);
    /*
        Cuando se elija 'Otro', el formulario debe
        indicar al declarante que se incluya el tipo
        de vía delante del nombre de la vía
    */
    }
}
```

Figura 32. RoadTypeSeeder

Para crear un *seeder*, la convención en Laravel es que su nombre debe incluir el nombre del modelo seguido de la palabra *Seeder*, también en formato *CamelCase*.

En la figura 32 se observa el código `RoadTypeSeeder`, que utiliza el modelo `RoadType`, relacionado con la tabla `road_types`, que se ha creado con la migración `CreateRoadTypesTable`.

En primer lugar, se incorpora la sentencia `use` que indica el modelo a utilizar al comienzo del archivo.

En la función `run()` se utiliza el método `create` del modelo, pasándole como parámetro un array con los campos y su contenido. Cada invocación a `create` genera un registro en la tabla.

[12] Documentación de Laravel sobre seeders: https://laravel.com/docs/5.6/seeding

El archivo `DatabaseSeeder.php` es el *seeder* general y sirve para indicar el orden en el que se deben ejecutar los *seeders* que se registran en la función `run()`.
Hay que tener en cuenta que, en primer lugar, hay que alimentar las tablas maestras de las que luego el resto incorporarán alguna clave externa, como tipo de documento o puertos, antes de declarantes y embarcaciones, respectivamente.

```php
<?php

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        /*
        Antes de ejecutar los seeder, se borran las tablas para que no se dupliquen
        los datos. Para ello se crea el método protegido truncateTables, al que se
        le pasa un array con las tablas a vaciar.
        */
        $this->truncateTables([
            'provinces', 'doc_types', 'road_types', 'ports', 'hull_reference_types',
            'boat_types', 'propels', 'states', 't0_parameters', 'users',
        ]);
        //Las tablas de los seeders de datos de prueba se eliminan de este array
        //para que no se elmine su contenido si se ejecutan los seeders para resetear
        //las tablas maestras

        /*
        Registro de seeder:
        Primero los maestros y despues las tablas que dependen de ellos.
        */
        //Tablas maestras
        $this->call(ProvinceSeeder::class);
        $this->call(DocTypeSeeder::class);
        $this->call(RoadTypeSeeder::class);
        $this->call(BoatTypeSeeder::class);
        $this->call(PropelSeeder::class);
        $this->call(PortSeeder::class);
        $this->call(HullReferenceTypeSeeder::class);
        $this->call(T0ParameterSeeder::class);
        $this->call(StateSeeder::class);

        $this->call(UserSeeder::class);  //Usuarios registrados

        // Los siguientes seeders cargan algunos datos para pruebas
        /*
        $this->call(ApplicantSeeder::class);
        $this->call(SpanishBoatSeeder::class);
        $this->call(BoatSeeder::class);
        $this->call(ApplicantBoatSeeder::class);
        $this->call(BoatVersionSeeder::class);
        $this->call(T0FeeSeeder::class);
        */
    }

    protected function truncateTables(array $tables)
    {
        /*
        Primero se desactiva la revisión de llaves externas para que la BBDD permita
        borrar los datos de la tabla vinculada. Después se vuelve a activar.
        */
        DB::statement('SET FOREIGN_KEY_CHECKS = 0;');
            foreach ($tables as $table){
            DB::table($table)->truncate(); //https://laravel.com/docs/5.6/queries#deletes
        }
        DB::statement('SET FOREIGN_KEY_CHECKS = 1;');
    }
}
```

Figura 33. Seeder General - *DatabaseSeeder.php*

Además, cada vez que se ejecutan los *seeder* se incorporan los datos, así que, si se crea un nuevo *seeder* y se vuelve a ejecutar `php artisan db:seed`, los datos de los *seeders* anteriores se duplican.

Para no tener que limpiar las tablas manualmente cada vez que se ejecutan, en este *seeder* general se ha creado un método protegido `truncateTables`, al que se le pasa un *array* que contiene el nombre de las tablas a vaciar, que invoca el método `truncate` de cada tabla para eliminar su contenido y resetear el su `id` a cero.

Para poder realizar este vaciado hay que desactivar el chequeo de llaves foráneas y luego volver a activarlo.

Finalmente, los datos de prueba se han cargado de forma manual, para probar el correcto funcionamiento del formulario, el envío de archivos al servidor, la creación del documento de pago en PDF y su envío al email indicado en el formulario.

7.2. Modelos

Los modelos, según el paradigma de la programación orientada a objetos, están representados por las clases que definen un conjunto de variables (atributos) y de acciones (métodos) sobre las anteriores. En Laravel representan también una tabla del modelo entidad-relación de una base de datos relacional en la que se guardarán los datos de los objetos creados a partir de dichas clases. Los modelos se guardan en la carpeta `TasaT0\app`.

En el apartado anterior se creó un modelo y acto seguido, sin definir un constructor, ni atributos o métodos, se utilizó directamente para la creación de un *seeder*. Esto es posible porque todos los modelos extienden de la clase `Model` del framework, y por tanto disponen de sus métodos.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Boat extends Model
{
    /** Atributos con asignación masiva **/
    protected $fillable = [
        'hull_reference_type_id', 'hull_reference', 'spanish_boat_id',
        'port_id'
    ];

    /** Devuelve la última fecha de liquidación T0 en formato español  **/
    public function lastT0Fee($id){
        return date('d-m-Y', strtotime($this->find($id)->last_T0_fee));
    }


    /****** Relaciones *******/

    /** Cada embarción tiene un puerto base **/
    public function port(){
        return $this->belongsTo(Port::class);
    }

    /** Cada embarcación tiene un tipo de referencia de casco **/
    public function HullReferenceType(){
        return $this->belongsTo(HullReferenceType::class);
    }

    /** Si la embarcación es española, entonces tiene un NIB **/
    public function spanishBoat(){
        return $this->belongsTo(SpanishBoat::class);
    }

    /** Cada boat puede tener muchas versiones **/
    public function boatVersions(){
        return $this->hasMany(BoatVersion::class);
    }

    /** Cada embarcación puede tener varios declarantes,
        cada uno con un porcentaje de titularidad sobre la misma,
        en la fecha de la autoliquidación **/
    public function applicants(){
        /** Usa la tabla pivote applicant_boat **/
        return $this->belongsToMany(Applicant::class)
            ->as('ownership') //alias para el attibuto "pivot"
            ->withPivot('percentage', 'date')
            ->withTimestamps();
    }

    /** Cada embarcación puede tener muchas autoliquidaciones **/
    public function t0Fees(){
        return $this->hasMany(T0Fee::class);
    }
}
```

Figura 34. Modelo *Boat*

Laravel proporciona protección contra la vulnerabilidad *mass-assigment* por defecto por lo que, si se pretende crear objetos a partir de una petición HTTP, como el envío de un formulario web rellenado por un usuario, se debe indicar mediante el atributo protegido `$fillable` el array de los atributos que se aceptarán en esa petición.

En la figura 34 se observa este atributo, y se declara un método para obtener la fecha de la última liquidación de la tasa para cada objeto `boat`. A continuación, se declaran métodos para las relaciones[13] de esta clase con el resto, según el modelo UML de la figura 9. Por ejemplo, así será posible acceder al objeto puerto relacionado con cualquier embarcación.

La relación n..m de entre `Boat` y `Applicant` tiene una declaración especial, en la que se asigna un alias al pivote para poder acceder a los atributos `percentage` y `date` de la tabla pivote entre ambas.

[13] Documentación de Laravel sobre relaciones entre modelos: https://laravel.com/docs/5.6/eloquent-relationships

7.3. Rutas, controladores y vistas

Todas las peticiones HTTP son recibidas por el archivo `index.php`, procesadas por el framework y finalmente enviadas al archivo **`web.php`** de la carpeta `TasaT0\routes`. Es en este archivo de **rutas**[14] en el que se indica qué hacer ante cada una de las peticiones. Por tanto, es en este archivo donde hay que crear una ruta para cada una de las peticiones HTTP que recibirá la aplicación.

```php
<?php

/*
|--------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------
|
| These routes are loaded by the RouteServiceProvider
| within a group which contains the "web" middleware group.
*/

Route::get('/', function(){
    $title = "Tasa T0 - Autoliquidación";
    return view('wellcome', compact('title'));
});

Route::get('/privacy', function(){
    $title = "Política de Privacidad y Protección de Datos";
    return view('privacy', compact('title'));
});

/* Rutas para el Formulario */
Route::get('/FormularioTasaT0',  'T0FormController@showT0Form');
Route::post('/FormularioTasaT0', 'T0FormController@register');


/* Rutas para ficheros */
Route::get('applicant/{id}', 'FileController@getApplicantFile');
Route::get('boat/{id}',      'FileController@getBoatFile');
Route::get('t0fee/{id}',     'FileController@getT0feeFile');


/* Rutas para el Dashboard */

Route::get('/dashboard', 'DashboardController@index');

Route::get('/dashboard/applicants', 'DashboardController@applicants');

Route::get('/dashboard/applicants/{id}', 'DashboardController@applicantShow')
    ->where('id', '[0-9]+');

Route::get('/dashboard/boats', 'DashboardController@boats');

Route::get('/dashboard/boats/{id}', 'DashboardController@boatShow')
    ->where('id', '[0-9]+');;

Route::get('/dashboard/t0fees', 'DashboardController@t0fees');

Route::get('/dashboard/t0fees/{id}', 'DashboardController@t0feeShow')
    ->where('id', '[0-9]+');


/* Rutas para la autenticación */

Auth::routes();
```

Figura 35. Archivo *web.php*

En la figura 35 se puede ver que, ante una petición de tipo `get` para el directorio raíz ('/'), la función anónima del segundo parámetro devuelve la vista `welcome`, definida en el archivo `welcome.blade.php` de la carpeta `TasaT0\resources\views`. Además, se pasa una variable a la vista para mostrar el título de la página.

La mayoría de rutas son dirigidas a una acción de un controlador específico, concatenando el nombre del mismo y su acción mediante el símbolo `@`.

`Auth::routes()` es un helper [15] de Laravel que estable todas las rutas necesarias para la interfaz de autenticación.

Las rutas declaradas se pueden mostrar en la terminal con el comando **`php artisan route:list`**

Por último, indicar que en la figura anterior no se muestra la ruta para `/paymentDoc`, utilizada para el diseño de la vista que sirve de base para documento de pago en PDF. Se le pasa un array asociativo `$data` con la información que necesitará del formulario. Se deja como comentario en este archivo para posteriores cambios en el diseño.

[14] Documentación de Laravel sobre Rutas: https://laravel.com/docs/5.6/routing
[15] Documentación de Laravel sobre helpers: https://laravel.com/docs/5.6/helpers

Un **controlador**[16] es una clase, que extiende la clase `Controller` del framework y cuyos métodos indican qué acciones se deben tomar para cada petición HTTP.

Para crear un controlador, por ejemplo para las peticiones procedentes del panel de control, se emplea el siguiente comando de la terminal: **`php artisan make:controller DashboardController`**
En la carpeta `TasaT0\Http\Controllers` se ha creado el archivo `DashboardController.php`, en el que se define una clase que extiende la clase `Controller` del framework. Ahora hay que definir en esta clase los métodos que responden a las acciones del usuario en las vistas y que se transmiten al controlador a través de una petición HTTP atendida por una ruta.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Applicant;
use App\Boat;
use App\BoatVersion;
use App\T0Fee;
//use Carbon\Carbon;

class DashboardController extends Controller
{

    public function __construct(){
        $this->middleware('auth');  // Accseo solo para usuarios autenticados
    }


    public function index(){
        $title = 'Dashboard';
        return view('dashboard.index', compact('title'));
    }

    public function applicants(){
        $title = 'Listado de declarantes';
        $applicants = Applicant::all()->sortBy('name');
        return view('dashboard.applicants.index', compact('title', 'applicants'));
    }

    public function applicantShow($id){
        $applicant = Applicant::find($id);
        $title = 'Detalle del declarante #'.$applicant->id;
        return view('dashboard.applicants.details', compact('title', 'applicant'));
    }

    public function boats(){
        $title = 'Listado de embarcaciones';
        $boats = Boat::all();
        return view('dashboard.boats.index', compact('title', 'boats'));
    }

    public function boatShow($id){
        $boat = Boat::find($id);
        $title = 'Detalle de la embarcación #'.$boat->id;
        $boatVersions = BoatVersion::where('boat_id', $id)->get();
        return view('dashboard.boats.details', compact('title', 'boat', 'boatVersions'));
    }

    public function t0fees(){
        $title = 'Listado de autoliquidaciones';
        $t0fees = T0Fee::all()->sortByDesc('registration_code');
        return view('dashboard.t0fees.index', compact('title', 't0fees'));
    }

    public function t0feeShow($id){
        $t0fee = T0Fee::find($id);
        $title = 'Detalle de la autoliquidación #'.$t0fee->id;
        return view('dashboard.t0fees.details', compact('title', 't0fee'));
    }
}
```

Figura 36. Controlador *DashboardController.php*

En la figura 36 se puede ver el detalle de las acciones de este controlador a las que se encamina cada petición atendida en el archivo de rutas.
Así, si se navega hacia la url `https://tasat0.test/dahsboard/boats` la petición es atendida por la acción `boats` este controlador, que obtiene una colección con todos los objetos embarcación y asigna un título, y pasa esta información a la vista `index.blade.php` de la carpeta `dahsboard\boats` en el directorio de vistas.

En el método constructor de esta clase se ha especificado que el acceso a las vistas que controla solo se pueda realizar si está autenticado. Por tanto, solo si hay una sesión de usuario abierta se podrá acceder a la url del ejemplo anterior. En caso contrario, la aplicación mostrará la vista para realizar el *login* de usuario como paso previo a mostrar la vista `boats` solicitada.

[16] Documentación de Laravel sobre Controladores: https://laravel.com/docs/5.6/controllers

Las **vistas**[17] contienen la interfaz de usuario, en este caso el código HTML, CSS y Javascript, que se envía al navegador, y se encuentran en la carpeta `TasaT0\resources\views`. Para crear una vista utilizando el motor de plantillas Blade, hay que crear un archivo de extensión `blade.php` e incluir el código HTML.

```
<!doctype html>
<html lang="{{ app()->getLocale() }}">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- CSRF Token -->
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>{{ $title }}</title>

    <!-- Scripts -->
    <script src="{{ asset('js/app.js') }}"></script>

    <!-- Fonts -->
    <link rel="dns-prefetch" href="https://fonts.gstatic.com">
    <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,700" rel="stylesheet">

    <!-- Styles -->
    <link rel="stylesheet" href="{{ asset('css/app.css') }}">
    <link rel="stylesheet" href="{{ asset('css/tasat0.css') }}">
</head>
<body>
    <div class="container">
        <div class="row justify-content-center mt-3 mb-3">
            <div class="col-md-12">
                @if (Route::has('login'))
                    <div class="top-right links">
                        @auth
                            <a href="{{ url('dashboard') }}">Dashboard</a>
                        @else
                            <a href="{{ route('login') }}">{{ __('Login') }}</a>
                            <a href="{{ route('register') }}">{{ __('Register') }}</a>
                        @endauth
                    </div>
                @endif
            </div>
        </div>

        <main>
            <h1 class="display-5">@yield('title')</h1>
            <hr class="bg-info">

            @yield('content')

        </main>

    </div>

    @yield ('script')

</body>
</html>
```

Figura 37. Vista *form.blade.php*

Blade [18] es el motor de plantillas de Laravel, que facilita la inclusión de código php dentro del HTML en las vistas mediante el uso de las dobles llaves `{{código php}}`.

También pone a disposición el uso de directivas que simplifican el uso de los bloques condicionales. Para usarlas hay que anteponer el símbolo arroba `@` al nombre de la directiva, por ejemplo `@foreach` y `@endforeach`.

Este sistema de plantillas también proporciona seguridad al código, ya que el contenido de las variables es escapado de forma automática, evitando así los ataques XSS (*Cross-Site Scripting*).

En la figura 37 se pueden ver algunas de las directivas de Blade, como `@yield`, `@extends`, `@section` y `@endsection`, que permiten establecer un *layout* para facilitar la estructuración del contenido y evitar la repetición de código en las vistas. Así, en la carpeta `views\layouts` se crea el archivo `form.blade.php`, que contiene el diseño elegido para la interfaz. Entonces, en el lugar apropiado de ese diseño se incluye la directiva `@yield` pasándole como argumento un nombre de sección.

Luego, ese contenido se desarrolla en la vista apropiada. Para ello, hay que indicar que la vista debe extender el diseño establecido, mediante la directiva `@extends('layout')`, y después desarrollar el contenido de la sección entre las directivas `@section('contenido')` y `@endsection`.

[17] Documentación de Laravel sobre Vistas: https://laravel.com/docs/5.6/views
[18] Documentación de Laravel sobre Blade: https://laravel.com/docs/5.6/blade

En el capítulo 8 se ahondará en el *layout* de la aplicación.

Por último, señalar que Laravel guarda en caché las vistas compiladas a php plano por Blade, de forma que no deba compilar una vista cada vez que es solicitada. Este caché de vistas compiladas se guarda en la carpeta `TasaT0\storage\framework\views`, y se actualiza de forma automática cada vez que hay un cambio en alguna de ellas.

7.4. Autenticación

Mediante el comando **`php artisan make:auth`**, Laravel genera todas las rutas, vistas y controladores necesarios para el sistema de autenticación de usuarios[19].

```
C:\laragon\www\TasaT0
λ php artisan make:auth
Authentication scaffolding generated successfully.
```

Las vistas de las nuevas rutas están en inglés. Se puede instalar mediante *composer* el paquete de idioma en español de GitHub, siguiendo el método indicado en el repositorio de Laraveles[20].

En la migración de la tabla `users` se añadieron los campos `doc` (string) y `civil_servant` (boolean), para el documento de identidad y para indicar si se trata de un administrativo de la AP. El primero hay que incluirlo en el formulario de registro, y el segundo tiene valor por defecto a `false`.
Para admitir el dato `doc` en el modelo `User` procedente de una petición HTTP, como el formulario de registro de un nuevo usuario, hay que añadir este atributo/campo en el array del atributo protegido `$fillable` del modelo `User`.
Hay que actualizar el diseño de las tablas mediante el comando `php artisan migrate:fresh --seed`

Para añadir el campo `doc` en el formulario de la vista `resources\views\auth\register.blade.php`, se ha seguido el ejemplo indicado en groloop.com[21]: primero se duplica el `div` que contiene la etiqueta y el campo para el nombre y se sustituye `name` por `doc`. Por último, en el controlador `Http\Controllers\Auth\RegisterController.php` se añade en campo `doc` en el array que se pasa como parámetro a los métodos `validator` y `create`.

Como se muestra en la figura 36, para requerir autenticación a determinadas vistas, por ejemplo, al listado de declarantes, hay que incluir en el método constructor del controlador que asiste a dichas vistas la siguiente instrucción: `$this->middleware('auth');`

[19] Documentación de Laravel sobre autenticación: https://laravel.com/docs/5.6/authentication
[20] Traducción del componente de autenticación, por Laraveles: https://github.com/Laraveles/spanish/tree/master
[21] Modificar formulario de registro de usuarios: https://www.groloop.com/laravel-5-4-6-sistema-autenticacion-usuarios/

Por otro lado, para restringir el acceso al contenido del Dashboard según el rol de usuario (parámetro `civil_servant` de `User`), hay que definir un middelware como se indica en el blog[22] de Programar y más. Esto queda pendiente para una revisión posterior de este proyecto.

Queda pendiente modificar el método validator del controlador de autenticación RegisterController para verifique que para el documento de identidad y el email indicados exista una autoliquidación de tasa T0. De esta forma se consigue:

1. Que en el sistema solo haya usuarios registrados que tengan autoliquidaciones.
2. Se omite la verificación de que el documento de identidad y del email son reales, pues en la autoliquidación ya se ha realizado: se valida y adjunta fotografía del documento, se comprueba la existencia del email enviado un correo previo.

[22] Blog: https://programacionymas.com/blog/restringir-acceso-solo-administradores-laravel-usando-middlewares

7.5. Pruebas automatizadas

Las **pruebas**[23] en Laravel se escriben en la carpeta `TasaT0\tests`, y se dividen en dos directorios:

- `tests\Feature` para simular peticiones HTTP
- `test\Unit` para probar partes de la aplicación, por ejemplo, los métodos de una clase.

Las pruebas se ejecutan en la terminal Cmder con el comando **`vendor\bin\phpunit`**
Puesto que se va a ejecutar en muchas ocasiones durante el desarrollo, es preferible definir un alias que lo abrevie, mediante la instrucción **`alias t=vendor\bin\phpunit`** (**`t`** es alias de "test").
Si al ejecutar **`t`** en la terminal no se ejecutan las pruebas automatizadas, se arregla[24] abriendo una ventana de símbolo de sistema en Windows con `cmd.exe`, y a continuación, hay que seleccionar con el botón derecho en la barra de título la opción Propiedades, marcar la casilla "Usar la consola heredada", Aceptar y, por último, reiniciar Cmder.
Las pruebas de tipo *Feature* se crean mediante el comando **`php artisan make:test UsersTests`**
Así, se ha creado un archivo en dicha carpeta en el que se pueden probar las rutas creadas para los usuarios, como el listado o el detalle de un perfil.

Se ha replanificado la realización de pruebas unitarias para la última fase de este proyecto, y finalmente no se incluyen por falta de tiempo. Las pruebas que se crearon al principio se han dejado como ejemplo, pero ya no pasan porque no se han actualizado al activar la autenticación.

7.6. Seguridad

Laravel ofrece por defecto protección contra los siguientes tipos de ataque.

- XSS attacks: https://laravel.com/docs/5.6/blade#displaying-data
- Mass-assignment: https://laravel.com/docs/5.6/eloquent#inserting-and-updating-models
- SQL injection: https://laravel.com/docs/5.6/queries#introduction

También proporciona lo necesario para:

- Autenticación: https://laravel.com/docs/5.6/authentication y https://laravel.com/docs/5.6/passport
- Autorización: https://laravel.com/docs/5.6/authentication
- Cifrado: https://laravel.com/docs/5.6/encryption
- Hashing: https://laravel.com/docs/5.6/hashing
- Reseteo de contraseñas: https://laravel.com/docs/5.6/passwords

[23] Documentación de Laravel sobre pruebas: https://laravel.com/docs/5.6/testing
[24] Solución de Seancheung, en su mensaje de 14/04/2017 en https://github.com/cmderdev/cmder/issues/1257

8. Front-end

8.1. Layout

La figura 37 muestra la vista `form.blade.php`, situada en la carpeta `TasaT0\resources\views\layouts` lugar donde se encuentran las composiciones o *layouts* para esta aplicación, que son 2: la anterior para las vistas que no requieren autenticación, como la de bienvenida, la política de privacidad y el formulario, y `app.blade.php` para las que sí requieren autenticación.
Esta distinción no responde a motivos de seguridad, sino de diseño: las que requieren autenticación son para mostrar el *dashboard*, que tiene una disposición de elementos en pantalla distinta a la del resto de vistas.

Ambos layouts son páginas web simples. En su etiqueta **`head`** incluyen los archivos de javascript y de css necesarios. Laravel utiliza **Bootstrap 4**[25] para conseguir un diseño rápido y elegante, pero todos los archivos js y css necesarios se compilan juntos en un único archivo `app.js` y `app.css` respectivamente, como se verá en el siguiente capítulo sobre la instalación de Bootstrap y Vue.
El archivo `tasat0.css` adicional es una personalización sobre `app.css`, y `pdf.css` es el que se utiliza para personalizar la vista que genera el documento de pago.
Por último, el archivo `app.css` se ha editado para incluir la fuente **Roboto**[26] de Google Fonts.

En el **`body`** se define un `div` de clase `container` que envuelve todo el contenido de la web y se fija en un ancho de 900 pixeles. Queda para una posterior revisión de este proyecto el cambio de escala según el tamaño de pantalla del dispositivo.
A continuación, hay un menú de navegación horizontal en la parte superior derecha de la página, que varía según el usuario este autenticado o no. Para ello se utilizan las directivas de Blade `@guest`, `@else` y `@endguest`, o `@auth`, `@else` y `@endauth`, según el archivo de layout. La primera se utiliza para detectar usuario no autenticados (invitados), mientras que `@auth` detecta los autenticados.
Todavía dentro del `body`, se define la etiqueta **`main`**, en la que se desarrolla el contenido. Parte de este contenido se delega en otras vistas que extienden estos layouts, mediante la directiva `@yield`.
También se utiliza `@section … @show` para definir una parte del contenido que se usa en las vistas del Dashboard, pero no se usa en las vistas del sistema de autenticación de Laravel (login, registro y restablecer contraseña). Por último, también se puede observar un bloque condicional `@if … @endif`.

[25] Web oficial de Bootstrap: https://getbootstrap.com/
[26] Fuente Roboto, de Google Fonts: https://fonts.google.com/specimen/Roboto

8.2. Instalación de Bootstrap y Vue.js

Laravel no obliga a utilizar un preprocesador[27] CSS o Javascript predeterminado. Sin embargo, permite instalar de una forma sencilla Bootstrap y Vue[28] a través de Laravel Mix[29].
Una vez comprobado que en el archivo `package.json` en la raíz del proyecto se encuentran Bootstrap y Vue en las dependencias (también están otros como JQuery), se procede a la instalación mediante el comando **`npm install`**. Este proceso descarga lo necesario de internet y luego lo instala, así que tarda un poco.

Después, como se indica en la documentación de Laravel Mix, se compila mediante el comando **`npm run development`**, y de esta forma se obtienen los archivos `app.js` y `app.css` en la carpeta `public\js` y `public\css` del proyecto. De esta forma ya se tienen todos los archivos necesarios para usar estos componentes en estos dos archivos que, como vimos en el capítulo anterior, se incluyen en los layouts y, por tanto, están disponibles para todas las vistas.

Puesto que Bootstrap es ampliamente conocido, solo mencionar que se hacen uso de sus clases en las etiquetas html. Para ampliar información, consulte la documentación y ejemplos en su web.

Para usar Vue 2 en una vista, hay que incluir una etiqueta `script` al final de la misma, justo antes de cerrar el `body`. A tal fin, se ha dispuesto la directiva `@yield('script')` en los layouts, de forma que cualquier vista en la que se quiera utilizar deberá contener un `@section('script')`.
Por tanto, esta sección deberá tener una etiqueta `<script type="text/javascript">` en la que se define una instancia[30] de la clase Vue, a la que se deben pasar los siguientes parámetros:

1. Elemento **`el`**. Hay que indicar el selector de la etiqueta sobre la que se aplicará el código javascript que se definirá a continuación. En este proyecto se usa en el formulario de la tasa T0, por lo que se le pasa el selector del `id` de su etiqueta: `'#formulario'`.
2. Objeto **`data`**. Sus propiedades coincidirán con los campos del formulario que necesiten ser reactivos, a los que se ata a través del atributo `v-model` en los elementos del formulario. También se pueden añadir otras propiedades que se usaran en los métodos.
   Es importante definir todos los atributos, aunque sea para indicar una cadena vacía.
3. Objeto **`methods`**. Cada una de las funciones javascript con la lógica de la aplicación.
4. Ganchos del ciclo de vida de la instacia: **`hooks`**. En el formulario se utiliza **`beforeMount`**.

También hay otros atributos disponibles, como `v-on` (`@`), `v-if`, `:`, etc.

---

[27] Preporcesadores en Laravel: https://laravel.com/docs/5.6/frontend
[28] Documentación sobre Vue: https://vuejs.org/v2/guide/index.html
[29] Documentación sobre Laravel Mix: https://laravel.com/docs/5.6/mix
[30] Instanciación de Vue: https://vuejs.org/v2/guide/instance.html

8.3. Formulario de autoliquidación

El formulario se dirige a la recaudación de la Tasa T0 para embarcaciones de recreo y para pesqueros de litoral. Estas pueden tener puerto base en alguno de los puertos cuyas ayudas a la navegación están bajo la supervisión de esta AP, en cuyo caso deben pagar esta tasa en esta AP, o pueden estar de paso en alguno de estos puertos, en cuyo caso deben presentar el certificado de tener esta tasa satisfecha o pagarla en esta AP. A las embarcaciones extranjeras se las considera de paso, o transeúntes, y se les cobra solo por el tiempo que vayan a permanecer.

Para determinar la cuantía y periodicidad de pago de esta tasa, hay que atender a la eslora y tipo de propulsión para las embarcaciones de recreo. Si son menores de 12 metros y a vela están exentas del pago de esta tasa. Si son menores de 9 metros y a motor, la tasa es la más elevada, pero pagan una única vez y deben conservar el justificante de pago para mostrarlo cuando se lo exija cualquier organismo. Si son a motor y de eslora igual o mayor a 9 metros, o a vela y eslora mayor o igual a 12 metros, tienen la misma tasa, que es de menor cuantía, pero tienen obligación de pagar anualmente.
Por último, los pesqueros de litoral, independientemente de su eslora y tipo de propulsión, tienen una cuantía fija para esta tasa, y deben pagarla anualmente.
Para la comprensión del funcionamiento del formulario, se recomienda repasar el resumen de la LPEMM que se hace en el **anexo 2**, donde se explica la casuística de cobro de esta tasa.

Para el desarrollo de este formulario se ha dispuesto una única **ruta**, como se puede observar en la figura 35, llamada `/FormularioTasaT0` y que, atendiendo al método de envío, redirige a un método distinto del controlador `T0FormController`. El método `get` invoca a la acción `showT0Form` del mismo, que recaba los datos de los modelos maestros necesarios y construye el array `$fecha_solicitud` que asocia el año con esta misma fecha del año, y así para los 4 años anteriores. El método `post` invoca a la acción `register`, que se explica en este capítulo más adelante.

Los datos obtenidos en el método `showToFrom` del controlador `T0FormController` se pasan a la **vista** `form\T0Form.blade.php` del directorio de vistas de Laravel.
Esta vista extiende del layout `form`, y en su sección `content` únicamente se desarrolla el html del formulario, especificando para cada campo las clases de Bootstrap y los atributos de Vue necesarios, así como los atributos html requeridos para una primera validación del lado del cliente, como los `required`, `min`, `max`, etc.

El primer apartado del formulario son una serie de campos ocultos en los que se guarda la información calculada por Vue sobre el resultado de la autoliquidación. De esta forma se envían de vuelta a Laravel y están disponibles para el documento de pago en PDF.
También se incluye la función `{{csrf_field()}}`[31] de Laravel, que genera otro campo oculto para el token de sesión activa de usuario, que previene de ataques de suplantación de identidad de usuario (**CSRF** – Cross-Site Request Forgery).

El formulario se divide en 6 secciones: las 4 primeras para la introducción de datos de cada sección, la quinta para mostrar el resultado de la autoliquidación, que es donde entra en juego Vue, y sexta para aceptar los términos y registrar el formulario.
Respecto a la validación, en la vista se incluye al final de cada sección un bloque condicional mediante directivas de Blade que comprueba si se el formulario se ha cargado con errores, es decir, ha sido validado insatisfactoriamente en el lado del servidor y se ha redirigido al usuario con la información de los errores detectados. En estos bloques se muestra esta información.
También mediante Blade se introduce la clase `is-invalid` en los campos cuando se ha detectado un error en alguno de ellos. Esto hace que el campo resalte con un borde rojo.
El proceso de validación se explica en el siguiente punto de este capítulo.

En la sección `script` se realiza la instanciación del **objeto Vue**, como se explicaba al final de apartado 8.2 de esta memoria. Este objeto apunta su elemento `el` al formulario completo porque necesita recabar los datos introducidos para el cálculo del resultado, que puede ser una exención del pago o un importe. Este resultado se muestra en la sección 5 del formulario.
Los datos que necesita para el cálculo se definen en el objeto `data`. Para su inicialización hay que tener en cuenta el valor que tenían si en el envío no se supera la validación y se retorna de nuevo al formulario, para que el usuario no deba introducir de nuevo toda la información.
Este punto es el que más problemas ha supuesto, donde parece que la integración de Laravel y Vue tiene su punto flaco. Se han invertido muchas horas en esto, que supuso el retraso en la planificación inicial, y tal vez habría sido más acertado tomar la decisión de no utilizar Vue y continuar con JQuery.
Se ha logrado que el campo Año (`application_date`) tenga este comportamiento mediante Vue. El escoyo fue que, al recargar el formulario, la fecha enviada era unos minutos posterior a la inicial, y por tanto el valor que se asignaba al select-option no era el que tenía, lo que generaba errores. Y aquí es donde entra el problema de integridad entre Laravel y Vue, para pasar el dato antiguo de la variable de sesión disponible en Blade, al parámetro `applicationDate` de Vue.

[31] Documentación de Laravel sobre CSRF: https://laravel.com/docs/5.6/csrf

Finalmente se tuvo que ingeniar la función `appDate_oldRequest()`, previa a la instanciación del objeto Vue y en una etiqueta `script` separada, para que al detectar datos antiguos se inicialice el campo con el valor correcto: la fecha actual con la hora antigua.

A continuación, se definen los métodos. En primer lugar, para el texto que se muestra de ejemplo (`placeholder`) en algunos campos, según el valor seleccionado en otro campo distinto. Después, para desactivar los campos NIB y matrícula, según se marquen los campos bandera española y nueva matriculación. La inicialización del campo bandera española también supuso problemas con los datos de validación pasados en la sesión de usuario. Revisar la historia de usuario 3.1.3.
El método `tasaT0()` calcula el valor de la tasa según el conjunto de parámetros pasados, y en función de si se le pasa un 4º parámetro que determina el número de días de estancia si es transeúnte.
El método `importeTotal()` requiere como primer parámetro el resultado del método anterior, además de la luego la eslora y la manga, y devuelve el importe en función de si es embarcación de recreo o un pesquero.
Para el cálculo de la tasa para embarcaciones transeúntes es necesario el número de días del año, y esto depende de si el año es bisiesto o no. Se puede comprobar para el 2016, que si lo fue.

Los métodos `tasaT0()` e `importeTotal()` son auxiliares de `calcularImporte()`: en un primer lugar este método incluía la lógica de los 2 anteriores, pero mediante la refactorización para la simplificación de su código se llegó a esta conclusión. Este método es el que contiene la lógica de la casuística legal. Según los datos introducidos en el formulario, en primer lugar, resetea el estado de la autoliquidación, después selecciona el conjunto de parámetros en vigor según el año seleccionado, en tercer lugar, asigna el valor a los parámetros de Vue que se pasan a los campos ocultos iniciales, necesarios para pasar la información al documento PDF y simplificar el siguiente apartado, el cálculo del resultado de la autoliquidación utilizando las funciones auxiliares. Por último, establece el estado de la autoliquidación.

La última sección de Vue es un gancho o *Hook* que permite que en cada evento `@keyup` se recalcule el importe y se refresque el DOM.

La sección 5ª del formulario muestra el resultado de la autoliquidación en función del estado de la misma, gracias al atributo `v-show`. Para este fin se disponen las capas *mensaje*, con el texto inicial al cargar el formulario, *exentos* con el texto para este resultado, y *obligados* en la que se indica el importe y la forma de realización del cálculo. Esta última se subdivide a su vez en otras capas según la casuística, ya que el texto a mostrar varía en función de la misma.

Cuando el usuario introduce correctamente todos los datos solicitados, se muestra el resultado y, para poder registrar la solicitud de autoliquidación, debe aceptar la política de privacidad y protección de datos.
El envío del formulario se hace mediante el método POST, según se indica en la etiqueta `form`, a la ruta `FormularioTasaT0` y, como se ha explicado anteriormente, ésta redirige a la acción `register` del controlador `T0FormController`, que recibe como parámetro el objeto `$request` de la clase `T0FormRegRequest`, con la que se lleva a cabo la validación. Esto es explicado con más detalle en el siguiente apartado.
El método `register` de este **controlador** es el que debe procesar la información para crear los objetos del diagrama UML de la figura 9 y, a través del ORM, guardarla en la base de datos.
Puesto que la información debe guardarse en distintas clases/tablas, en primer lugar, se estable una transacción[32] con la base de datos, a la que se le pasa una función anónima con los datos validados. Esto es necesario porque, si ocurriese un error al guardar uno de los últimos objetos en la base de datos (fallo de comunicación, desconexión por mantenimiento, etc.), no se tendría persistencia de toda la información de la autoliquidación, sino solo de los objetos que pudieron ser guardados al principio. Al establecer la transacción, si ha habido un error, se hace un rollback y se elimina el resto de datos guardados.

A continuación se recogen los datos enviados y validados correctamente en el array `$data` mediante `$data=request()->all();` para pasar a construir el código de registro de la autoliquidación. En entregas anteriores de esta memoria, como se puede ver en la figura 26, esto se realizaba mediante un *trigger* de la tabla `t0_fees` en la base de datos, porque tenía sus ventajas, como pasar una parte de la lógica a la base datos ya que podría considerarse un campo calculado. Pero en posteriores iteraciones de desarrollo se concluye que este dato es necesario antes de que se guarde la instancia de `T0Fee`, ya que el resto de instancias también lo requieren. En este punto hay que destacar que el orden de guardado de los objetos es importante, puesto que el objeto de la clase T0Fee debe ser el último por restricciones de integridad de la base de datos.

El código de registro tiene el formato T0-YYYYMMDD-XXXX, donde T0 y los guiones son un texto fijo, YYYY es el año de presentación, el elegido en el formulario, MM el mes, DD el día, y XXXX es un número entero secuencial de 4 cifras, que se completa por ceros por la izquierda. Para construirlo hay que comprobar que la base de datos no esté vacía inicialmente, que tenga solicitudes para el año, mes y día indicados, localizar su último código e incrementarlo en su caso.

[32] Documentación de Laravel sobre transacciones: https://laravel.com/docs/5.6/database#database-transactions

Después se guardan los datos del declarante, que tienen un archivo con el documento de identidad que debe ser asociado. Para que este archivo pueda ser enviado mediante POST, en la etiqueta `form` debe indicarse `enctype="multipart/form-data"`. Este es el motivo de necesitar el código de registro antes de llegar a guardar la instancia de `T0Fee`, ya que para guardar los archivos en el servidor se establece una estructura de directorios con este código para guardar todos los documentos asociados a cada autoliquidación. Todo lo necesario para guardar los archivos recibidos se explica en el apartado 8.4.
Una vez guardado el archivo en el servidor, en el array `$data` se incluye la ruta asociada al nombre de atributo que espera la instancia de `Applicant`. Por último, y valiéndose de la propiedad protegida `$fillable` establecida en los modelos, que solo admite crear nuevas instancias con los datos indicados en este array procedentes de una petición HTTP, se pasa completo el array `$data` al método `create` de la clase `Applicant`, y el objeto resultante se guarda en la variable `$applicant`. Este método tiene la ventaja de guardar al mismo tiempo los datos en la base de datos.

Antes de guardar la instancia de `Boat`, se comprueba si se ha introducido un NIB en el formulario. Si es así, y si este no existía ya, se crea una instancia de `SpanishBoat` para este dato. Si existía ya, se toma su id y se guarda en el array `$data`, puesto que es una clave foránea de la tabla `boats`.
A continuación, se comprueba si ya existe la embarcación. Si no es así, se crea.

Toca el turno de la versión de embarcación. Hay que comprobar si ya existían versiones para esta embarcación, obtener el último número de versión y asignarlo a esta nueva versión. En caso contrario, si no había versiones (es una nueva embarcación) entonces se le asigna la versión 1.
La embarcación también tiene asociado un archivo, que se guarda en la misma carpeta creada anteriormente en el servidor y cuya ruta se almacena en la versión de embarcación.

Creados el declarante y la embarcación, hay que guardar la relación de propiedad entre estos. En este caso, puesto que no hay un modelo para esta relación, se accede directamente a la tabla de la base de datos y se guarda el porcentaje de titularidad y la fecha de registro de la solicitud junto a los ids de los objetos a relacionar.

Por último, hay que guardar los datos de la autoliquidación. Como esta clase puede tener una asociación reflexiva, se comprueba si se ha declarado para guardar los datos. También si chequea si se ha declarado la condición de transeúnte, y si no, se fija el número de días a 0.
Como en la migración de la tabla se estableció el campo donde se guarda la ruta para el documento pdf como `nullable`, en este momento se dispone de toda la información para crear la instancia de `T0Fee`.

Guardada toda la información del formulario, queda pendiente la generación del documento PDF, su envío al email indicado por el declarante y mostrarlo en pantalla como indicación de que el proceso se ha realizado correctamente. La generación del documento se detalla en el apartado 8.5, y el envío por email en el 8.8.
Una vez generado, se guarda en la carpeta para la autoliquidación en el servidor y su ruta se almacena en el objeto T0Fee creado anteriormente. En esta ocasión hay que invocar el método `save()` para que tenga lugar la persistencia de la información en la base de datos.

Puesto que toda la lógica de este método tiene lugar en una función anónima de la transacción que se establece al inicio, fuera del ámbito de esta transacción no hay acceso a las variables internas. Esto suponía un problema, pues desde dentro de la transacción no se podía redirigir el navegador para mostrar el documento PDF generado. La solución adoptada fue devolver el contenido raw del documento PDF como resultado de la transacción a una variable, que es la que envía al navegador para que muestre el documento de pago.

8.4. Validación del formulario

Mediante el comando de consola **`php artisan make:request T0FormRegRequest`** se crea la clase `T0FormRegRequest` en la carpeta `\app\Http\Requests`, que extiende de `FormRequest`.
En esta clase se indican las reglas de validación y los mensajes de error personalizados, mediante los métodos `rules` y `messages` respectivamente, además de la lógica necesaria para la adición de estas reglas de validación. También se debe hacer que el método `authorize` devuelva `true` para permitir a todos los declarantes, registrados o no, que realicen una solicitud de autoliquidación.
Para que el controlador del formulario `T0FormController` utilice esta clase para la validación, se le pasa como parámetro a la acción `register`, que es a la que se dirige la ruta por el método `POST`.

```
public function register(T0FormRegRequest $request) { … }
```

Si la validación no pasa, por defecto se genera una respuesta de redirección a la página anterior: el formulario. Los errores se pasan a la vista del formulario a través de la sesión de usuario, y se muestran en bloques condicionales de Blade (`@if … @endif`) al final de cada sección del formulario.
Cuando se detectan errores, se retorna al formulario mostrando la información previamente introducida, que también es pasada mediante la sesión de usuario. Esto se realiza estableciendo en el atributo `value` de cada `input` lo siguiente: `{{ old('input_name') }}`. Para los campos que se tratan en el controlador mediante Vue, puesto que se deben inicializar al instanciar el objeto Vue, se incorpora la directiva `old` en la inicialización de cada valor.

Para validar el documento de identidad, se utiliza la librería **mpijierro/identity**[33], que proporciona reglas de validación para NIF, NIE y CIF, así como para IBAN bancario. Para instalarla, se introduce en la terminal: **`composer require mpijierro/identity`** y luego se incluye en el archivo `config\app.php` el proveedor de servicios `\MPijierro\Identity\IdentityServiceProvider::class` y el alias para el *facade* `'Identity' => MPijierro\Identity\Facades\Identity::class`.
El pasaporte no se valida, puesto que el formato del número puede ser distinto para cada país.

En el archivo `IdentityServiceProvider.php` de la carpeta `\vendor\mpijierro\src` se ha añadido la regla `identityDoc` para validar que el documento introducido es uno de los tipos dados, aprovechando el método `documentIsValid` definido en la clase `Identity` (archivo `Identity.php` en la misma ruta). Finalmente esta regla no se utiliza (se deja comentada), puesto que se ha añadido en la lógica de inclusión de reglas de la clase `T0RegisterRequest` las condiciones necesarias para detectar el tipo de documento de identidad y aplicar las reglas que ya estaban definidas en `IdentityServiceProvider`: nif, cif o nie. Por ejemplo: `$rules['doc'] = 'required|nif';`

[33] Repositorio GitHub de mpijierro/identity: https://github.com/mpijierro/identity

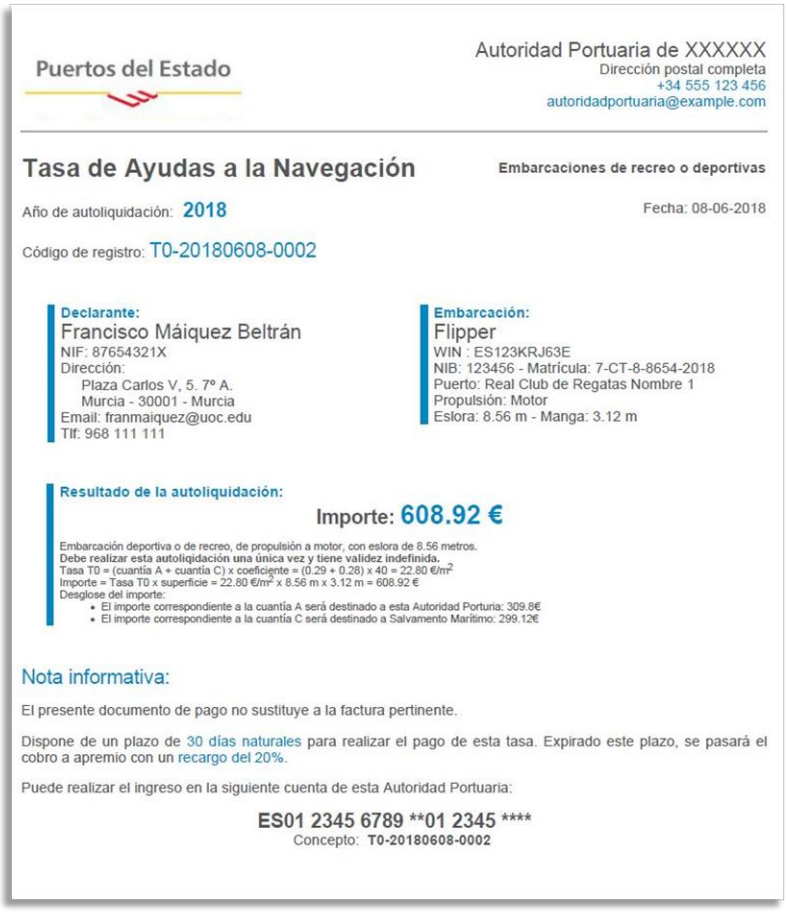
8.5. Generación de documentos PDF

Se utiliza **Laravel-Dompdf**[34], una librería PHP desarrollada por Barry vd. Heuvel que toma un contenido HTML de entrada, en este caso una vista, para la obtención de un documento PDF.

Para su incorporación al proyecto hay que ejecutar el siguiente comando en la terminal, que descarga e instala todo lo necesario: `composer require barryvdh/laravel-dompdf`

Con esto queda registrada la dependencia `"barryvdh/laravel-dompdf": "^0.8.2"` en el apartado *require* de `composer.json`

Por último, para poder utilizarlo en la aplicación, en el archivo `config\app.php` hay que incluir en el apartado `providers` el *service provider* `Barryvdh\DomPDF\ServiceProvider::class`, y en el apartado *aliases* el alias `'PDF' => Barryvdh\DomPDF\Facade::class`


Puertos del Estado

Autoridad Portuaria de XXXXXX
Dirección postal completa
+34 555 123 456
autoridadportuaria@example.com

Tasa de Ayudas a la Navegación — Embarcaciones de recreo o deportivas

Año de autoliquidación: 2018 — Fecha: 08-06-2018

Código de registro: T0-20180608-0002

Declarante:
Francisco Máiquez Beltrán
NIF: 87654321X
Dirección:
Plaza Carlos V, 5. 7º A.
Murcia - 30001 - Murcia
Email: franmaiquez@uoc.edu
Tlf: 968 111 111

Embarcación:
Flipper
WIN : ES123KRJ63E
NIB: 123456 - Matrícula: 7-CT-8-8654-2018
Puerto: Real Club de Regatas Nombre 1
Propulsión: Motor
Eslora: 8.56 m - Manga: 3.12 m

Resultado de la autoliquidación:
Importe: 608.92 €
Embarcación deportiva o de recreo, de propulsión a motor, con eslora de 8.56 metros.
Debe realizar esta autoliqidación una única vez y tiene validez indefinida.
Tasa T0 = (cuantía A + cuantía C) x coeficiente = (0.29 + 0.28) x 40 = 22.80 €/m²
Importe = Tasa T0 x superficie = 22.80 €/m² x 8.56 m x 3.12 m = 608.92 €
Desglose del importe:
- El importe correspondiente a la cuantía A será destinado a esta Autoridad Porturia: 309.8€
- El importe correspondiente a la cuantía C será destinado a Salvamento Marítimo: 299.12€

Nota informativa:

El presente documento de pago no sustituye a la factura pertinente.

Dispone de un plazo de 30 días naturales para realizar el pago de esta tasa. Expirado este plazo, se pasará el cobro a apremio con un recargo del 20%.

Puede realizar el ingreso en la siguiente cuenta de esta Autoridad Portuaria:

ES01 2345 6789 **01 2345 ****
Concepto: T0-20180608-0002

A continuación, hay que diseñar una vista que presente el contenido del documento de pago para las autoliquidaciones. La vista se guarda en `\form\pdf\T0PaymentDoc.blade.php` y hace uso del archivo `pdf.css` ubicado en `\public\css`.

Para ajustar esta vista y hacer pruebas, se crea la ruta `/paymentDoc`, que pasa unos datos de prueba a la vista y le asigna una url. Esta ruta se deja comentada en el archivo de rutas para su revisión.

Figura 38. Documento de pago generado por la aplicación

La configuración de *domdpf* se establece en el archivo `\config\dompdf.php`, y se debe copiar a esta ruta desde el zip descargado del repositorio de GitHub. También se crea la carpeta `fonts` en `\storage` para guardar la fuente `Roboto` de Google Fonts, que es la utilizada para la aplicación.

[34] Repositorio GitHub de barryvdh/laravel-dompdf: https://github.com/barryvdh/laravel-dompdf

El último apartado de la acción `store` del controlador `T0FormController` usa el *facade* `loadView` del alias PDF, esto es, de la clase `Facade` de la librería `DomPDF` de Barryvdh, para generar el objeto de esta librería con la vista diseñada anteriormente, pasándole en esta ocasión los datos recibidos del formulario. A continuación, se emplea el método `stream` para generar el contenido bruto (*raw*) PDF del documento de pago, pasándole como atributo el nombre del fichero para su descarga, que es el código de registro.

```
$pdf = PDF::loadView( 'form.pdf.T0PaymentDoc', compact('data'));
$paymentDoc = $pdf->stream($registration_code.'.pdf');
return $paymentDoc;
```

8.6. Adjuntar documentos

En el archivo `config\filesystems.php` se incluye el siguiente `disk` mediante el que se indica que en la carpeta local del servidor `storage\T0` se guardarán los documentos adjuntos de cada solicitud de autoliquidación de la tasa T0.

```
        'T0' => [
            'driver' => 'local',
            'root' => storage_path('T0'),
        ],
```

Cuando se crea el documento de pago al registrar una nueva autoliquidación, este se guarda en esta ruta mediante la siguiente instrucción:

```
$path=Storage::disk('T0')->put($registration_code.'_RegistroAutoliquidaciónT0.pdf', $paymentDoc);
```

Los parámetros de `put` son el nombre del archivo y su contenido.

En este apartado, además de la documentación de Laravel[35], me ha servido de ayuda el manual[36] de Samuel Oloruntoba publicado el 19/02/18.

8.7. Comprobación del correo electrónico

Este es uno de los hitos que se ha decido no incluir en esta entrega para la obtención de un producto utilizable en esta primera fase de entrega del proyecto. Como se explica en el anexo 4, en 1 mes podría estar desarrollado el alcance inicial, cuyo coste en número de horas excedía del plazo fijado por la fecha de entrega.

[35] Documentación de Laravel sobre tratamiento de archivos: https://laravel.com/docs/5.6/filesystem
[36] Manual sobre el uso de archivos en Laravel: https://scotch.io/tutorials/understanding-and-working-with-files-in-laravel

8.8. Envío de correo electrónico

Para evitar el **envío de correos**[37] de prueba cada vez que se crea una nueva autoliquidación mientras la aplicación se encuentra en modo de desarrollo y, sin embargo, poder comprobar su funcionamiento, se crea cuenta de `mailtrap.io` que es un servicio al que se dirige el correo de salida de la aplicación, mientras no esté en modo producción.

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=84a28222d0386c
MAIL_PASSWORD=2a5277884a7ccf
MAIL_ENCRYPTION=tls
```

Tabla 10. Configuración en `.env` de cuenta de mailtrap

Hay que configurar los datos de esta cuenta en el archivo `.env` como se muestra en la tabla adjunta.

El siguiente paso es crear la vista `notification.blade.php` en la carpeta `resources\views\email` que incluirá el texto del correo.

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
use Illuminate\Contracts\Queue\ShouldQueue;

class SendEmail extends Mailable{

    use Queueable, SerializesModels;

    public $info_email; //se pasa este array a la vista email.notification
    private $path, $filename;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct($info_email, $path, $filename){
        $this->info_email = $info_email;
        $this->path = $path;
        $this->filename = $filename;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build(){
        return $this->from('tasat0@autport.example.com')
                    ->subject('Notificación de autoliquidación de Tasa de
                        Ayudas a la Navegación')
                    ->view('email.notification')
                    ->attach(storage_path($this->path.'/'.$this->filename));
        //$path: ruta completa al archivo $filename desde TasaT0/Storage
    }
}
```

Figura 39. Clase *SendEmail*

A continuación, hay que crear una clase para el envío de correo mediante el comando de terminal

```
php artisan make:mail SendEmail
```

Esto crea la clase `SendEmail` en `app\Mail`, que extiende de la clase `Mailable`.

En esta clase se indica la dirección de correo de `mailtrap`, la vista que va a incluir el texto del correo y el documento a adjuntar. Se pasan 3 parámetros a su constructor, 2 privados y 1 público.

Los privados establecen la ruta y el nombre del archivo a adjuntar al correo, y el público es un array que contiene los datos que se pasan a la vista `email.notification` para mostrarlos en el cuerpo del correo.

Por último, se hace una llamada al *facade* `Mail` en la acción del controlador del formulario que recibe su envío mediante post, es decir `T0FormController@register`

```
Mail::to($data['email'], $data['applicantName'])
                    ->send( new SendEmail( info_email, 'T0/'.$registration_code, $filename) );
```

[37] Documentación de Laravel sobre envío de emails: https://laravel.com/docs/5.6/mail

Mediante los métodos `to` y `send` de este *facade* se le pasan la dirección del destinatario del correo y una instancia de la clase `SendEmail`, con los 3 parámetros que requiere su constructor.

Si se registra una nueva solicitud de autoliquidación, además de visualizar el documento de pago en PDF en el navegador, podemos comprobar en `mailtrap` la recepción del correo con este documento adjunto.

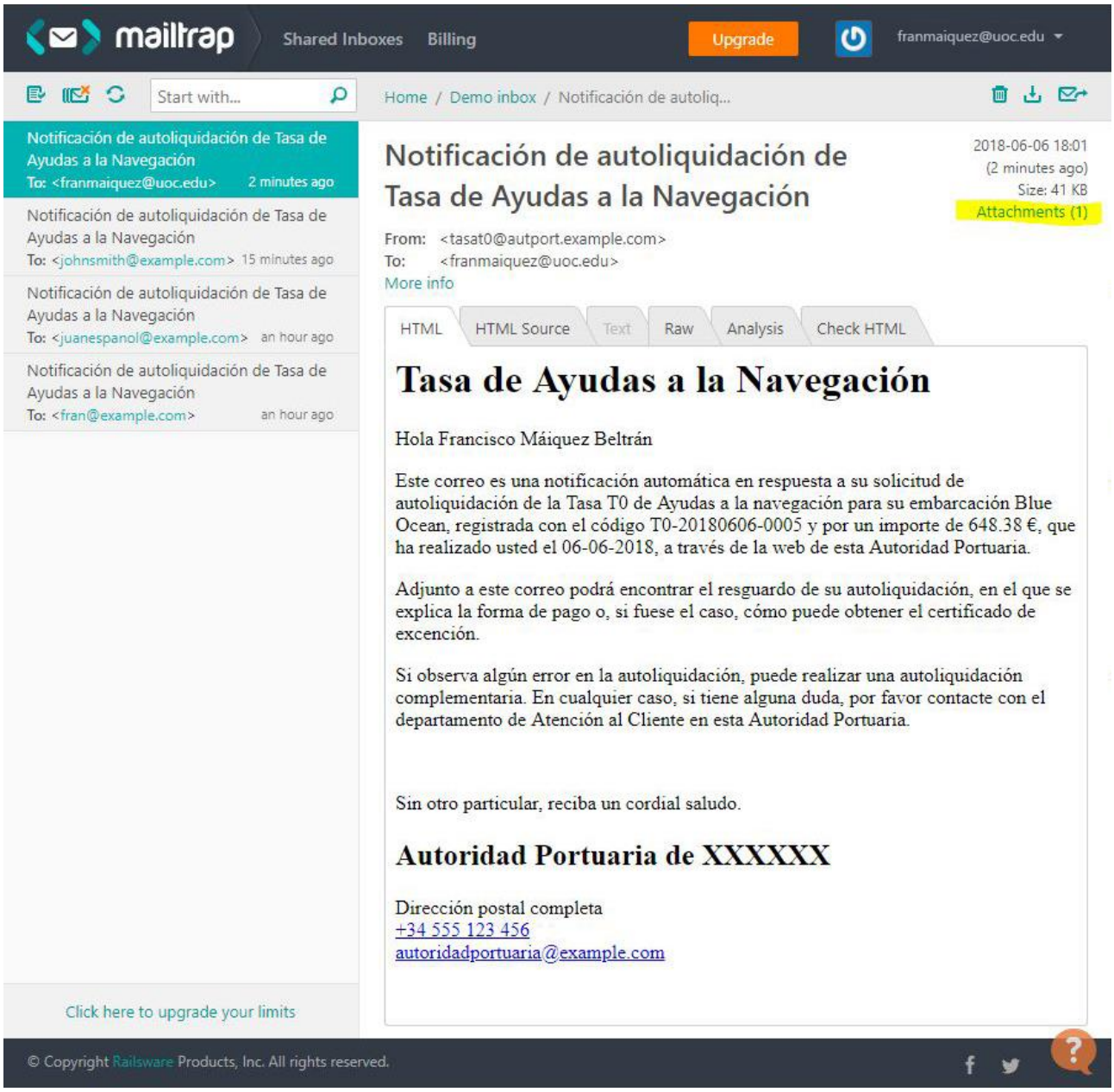


Figura 40. Recepción de correos con adjuntos en *mailtrap*

8.9. Dashboard

El dashboard, también llamado cuadro de mandos o panel de usuario, es un entorno privado al que se accede a través de la autenticación del mismo. A este efecto se disponen los links en el encabezado de la página de bienvenida y del formulario, como se explicó en la sección de layouts.

En el *seeder* del modelo User se crean 2 usuarios, uno para cada rol: declarante o administrativo. Las credenciales son las de la tabla adjunta.

	Administrativo	Declarante
Email	admin@example.com	jgraham@example.com
Contraseña	123456	123456

Tabla 11. Credenciales de acceso al Dashboard

Se han creado 7 **rutas** para las vistas creadas, como se muestra en la figura 35, asociadas cada una a una acción distinta del controlador `DashboardController`, mostrado en la figura 36.
La primera conduce a la pantalla de bienvenida del dashboard. Las 6 restantes son para mostrar el listado y la vista de detalle de declarantes, embarcaciones y autoliquidaciones.

Respecto a los métodos del **controlador**, destacar que los empleados para los listados envían a la vista toda la información para cada una de las tres clases, mientras que para las vistas de detalle el método acepta como parámetro el id recibido por la ruta, que es empleado en la selección del objeto concreto a mostrar.
El método al que se dirige la vista de detalle de embarcaciones de distinto a los otros 2, porque incluye también un listado de las versiones anteriores de la embarcación para la que se ha solicitado el detalle. En la vista de detalle del declarante se puede realizar algo similar para mostrar las embarcaciones que tiene asociadas y las últimas autoliquidaciones pagadas.
En el método constructor se requiere los usuarios estén autenticados para dar acceso a las vistas que se controlan en esta clase.

Las 7 **vistas** a las que conducen las rutas anteriores, a través del respectivo método del controlador, extienden del layout `app` y definen sus secciones de menú y contenido.
En un ciclo de desarrollo previo, el menú se definió como una lista desordenada. Después de consultar los ejemplos de Bootstrap, se optó por eliminar la lista y dejar solo los enlaces, dándoles apariencia de botón mediante las clases seleccionadas.
Con las vistas de listados ha ocurrido algo similar. El primer ciclo de desarrollo se centró en mostrar la información sin tener en cuenta el aspecto, utilizando listas desordenadas. En un ciclo posterior se decidió utilizar tablas también con clases de Bootstrap. El resultado es mucho más visual y ordenado. Las filas de las tablas se generan a partir de un bucle `foreach`, construido con directivas de Blade, sobre la colección de objetos que pasa el controlador a la vista.

En las vistas de listados se ha conseguido el efecto de link continuado para todos los elementos de la fila de la tabla mediante la combinación del uso de la clase `table-hover` en el elemento `table`, que destaca la fila sobre la que se encuentra el cursor, y sobre el elemento `tr`, de la disposición del estilo que modifica el puntero y del evento `onclick` para que redirija a la url del detalle para cada registro.

En el menú de la vista inicial del panel de usuario se incluyen directivas de Blade para mostrar diferentes opciones de menú según el rol de usuario. Concretamente, se sustituye la opción para acceder al listado de declarantes por la opción de perfil personal. No obstante, como ya se ha comentado, esto no es suficiente para limitar el acceso a la información: por ejemplo, un declarante podría tener acceso a la información de otro con tan solo modificar el id en la url.
Para lograr este nivel de restricción de contenidos, es necesario desarrollar un middleware al efecto. Por tanto, en una primera entrega de esta aplicación se impediría el registro de declarantes hasta que no esté esta funcionalidad terminada, mientras que los administrativos de la AP tendrían acceso a toda la información sin restricciones. Para ello bastaría con desactivar en el registro de rutas (comentar en el archivo `web.php`) la ruta para el registro.

Respecto a las vistas de detalle, para declarantes y embarcaciones se distribuye el contenido en 2 columnas: la primera para mostrar la información de detalle solicitada, y la segunda para mostrar una miniatura y el enlace del documento adjunto asociado a cada una de estos objetos, que se encuentran en la carpeta de la autoliquidación en el servidor. El detalle de embarcación, como se explicaba en el controlador, también incluye una tabla para mostrar el listado de versiones anteriores de la embarcación.
El detalle de la autoliquidación es distinto. Muestra una tabla con 4 apartados diferenciados:

- Código de registro. Junto a este se muestra el código de registro de la declaración complementaria, si la tuviese, con un enlace para acceder a la misma.
- Declarante. Muestra su detalle y un enlace al archivo de documento de identidad.
- Embarcación. Idem que para el declarante.
- Resultado. Muestra el importe y el detalle de su cálculo, así como un enlace para la descarga del documento de pago en PDF.

Como se comentaba en el apartado 1.5., no ha dado tiempo a completar la paginación, filtrado y búsqueda de información, y las acciones de usuario según rol.

9. Pruebas de usabilidad

En un proyecto real, estas pruebas se realizarían en una fase beta de la aplicación con los administrativos de la AP que vayan a utilizarla. Se observaría su forma de relacionarse con ella para determinar aspectos como qué acciones se pueden mejorar, que filtros o búsquedas son las más recurrentes para establecerlas por defecto, que sugerencias tienen sobre información que falte o que sobre en listados y vistas de detalle, etc.

Debería hacerse en una fase beta avanzada, que aunque no fuese funcional al 100%, permita a los usuarios tener una idea de cómo va a ser la aplicación. Pero también en una fase lo suficientemente temprana para que si detecta algún problema de base se pueda subsanar dentro del plazo de entrega.

Estas pruebas suponen una ayuda en la detección de posibles bugs ocultos, también una menor resistencia al cambio en los usuarios finales, ya que se tiene en cuenta su opinión en fases de desarrollo, facilita la formación posterior del producto, porque ya tienen una noción del mismo, permite obtener una mayor sensación de calidad en el producto terminado si los usuarios confirman que su opinión se ha tenido en cuenta en el producto final, y conduce finalmente al éxito en el cierre del proyecto. Sin duda, es muy recomendable realizar estas pruebas.

10. Tests y corrección de errores

La intención inicial era desarrollar pruebas unitarias para cada elemento de este proyecto: rutas, vistas, controladores, etc., de forma que se siguiese un desarrollo guiado por pruebas (**TDD**[38] – *Test-Driven Development*). De hecho, se comenzó diseñando pruebas para las vistas en su desarrollo inicial, que dejaron de pasar al introducir la autenticación de usuarios.
Como se ha mencionado en capítulos anteriores, la elevada carga de trabajo como resultado de no haber dimensionado correctamente las horas disponibles y el alcance del proyecto, ha hecho imposible que se siguiese esta metodología de desarrollo.

En su lugar, se ha chequeado todo manualmente, paso a paso, en el navegador, mediante Browser Toolbox de Firefox, que habilita entre otras cosas la consola del navegador y el inspector de página y estilos. Adicionalmente, para chequear los atributos del Vue, se ha recurrido al addon[39] o extensión[40], según el navegador, de Vue Devtools.

En algunas partes de código donde se han tenido problemas en el desarrollo también han dejado comentadas algunas sentencias para la visualización del contenido de variables, como `dd()` para PHP, o `console.log()` para Javascript. Por ejemplo, esto ha sido útil para observar los datos recibidos del formulario, mediante `dd($data)` en el controlador del mismo.

10.1. Bugs detectados

Actualmente la aplicación no chequea que un usuario pueda realizar más de una autoliquidación para la misma embarcación en el mismo año. Es algo poco probable, pero podría suceder.
Para su solución se ha dispuesto el atributo `last_T0_Fee` en el modelo `Boat`, de forma que, cuando el controlador del formulario guarde la información recibida, sobrescriba este campo con la última fecha. Solo faltaría que durante la validación del formulario se compruebe esta condición.

[38] Definición de desarrollo guiado por pruebas: https://www.agilealliance.org/glossary/tdd/
[39] Addon para Mozilla Firefox: https://addons.mozilla.org/es/firefox/addon/vue-js-devtools/
[40] Extensión para Google Chrome: https://chrome.google.com/webstore/search/vue%20devtools

11. Entrega y puesta en producción

Si fuese un proyecto real, en la última fase de entrega del producto se sustituirían las tareas de creación de la presentación y del vídeo de defensa por las siguientes tareas:

- Instalación de Laravel en equipos sW, copia de la carpeta TasaT0 con el contenido de la aplicación, creación de la base de datos en MySQL, creación del usuario de MySQL con permisos en la base de datos.
- Configuración del DNS con el subdominio de la AP para que se acceda a esta aplicación.
- Modificación del archivo `.env`: `APP_ENV=local` → `APP_ENV=production`
- Credenciales de acceso a la base de datos
- Nombre de la aplicación y url real con el subdominio de la AP para esta aplicación
- Configuración de cuenta del servidor SMTP para el envío de correo.
- Modificación del archivo `TasaT0\config\app.php` `'timezone' => 'Europe/Madrid',`
- Revisar el resto de propiedades (producción, nombre de la app, etc.)
- Realización de pruebas
- Entrega de manual de usuario
- Formación a empleados de la AP

12. Conclusiones

12.3. Lecciones aprendidas

- La primera y principal, hacer una planificación realista y con margen del número de horas disponibles de desarrollo respecto al coste en horas del alcance requerido en el proyecto.
- Incrementar el número de horas necesarias si el proyecto incluye el uso de frameworks u otras tecnologías desconocidas para los desarrolladores.
- El desarrollo siguiendo una metodología TDD también exige un número mayor de horas de desarrollo, pero a cambio garantiza un SW optimizado, depurado y sin bugs.

12.4. Logro de los objetivos

Este punto ha sido comentado en diversos apartados de esta memoria, como el capítulo 1.5. El alcance inicial era muy exigente para la cantidad de horas reales disponibles. Por tanto, no se han cubierto todos los hitos previstos.

No obstante, se ha alcanzado un grado de madurez en el que la aplicación es funcional, no con todos los requisitos, pero el formulario se puede utilizar y en el backend se puede hacer seguimiento de la información recibida.

Conforme está y desactivando el registro de usuarios, puesto que hay que desarrollar el middleware para la separación de contenidos por roles en el dashboard, podría ponerse en funcionamiento. El dashboard se quedaría como backend exclusivo para los administrativos.

12.5. Seguimiento de la planificación y metodología

Como se ha explicado, la planificación ha sufrido retrasos por los motivos expuestos. Se ha procedido a la replanificación a lo largo del proyecto, tomando la decisión final de no terminar algunos hitos a cambio de obtener un producto que pueda ser útil en esta fase de desarrollo.

12.6. Líneas de trabajo futuro

En primer lugar, terminar los hitos pendientes de desarrollo para un funcionamiento óptimo, según la definición de alcance inicial. Como se comentaba, con el nivel de conocimiento adquirido del framework en este punto, se estima que en 4 semanas estaría completamente completado.

Posibles mejoras:

- Integración con el software de facturación de la AP para la incorporación, de esta aplicación al ERP de facturación, de la información necesaria para realizar la factura.
- Acceso mediante DNI Electrónico y pago mediante tarjeta de crédito.
- Conexión con web-services de Hacienda y de Fomento, para chequear si el documento de identidad es correcto y pertenece a la persona que se indica en el campo nombre, y para obtener información de las embarcaciones con solo indicar el NIB.
  El interesado debería dar su consentimiento para la consulta de esta información.

13. Glosario

A

AAPP ..... ***Véase*** **AP (plural)**

AP

Autoridad Portuaria 11

B

B0 ***Véase*** **Tasa T0**

C

CamelCase

es.wikipedia.org/wiki/CamelCase 49

CPD

Centro de proceso de datos 19

H

HBA

Adaptador de Bus del Host 19

L

layout

composición, diseño gráfico 56

LPACAP

Ley 39/2015, de 1 de octubre, de Procedimiento Administrativo Común de las Administraciones Públicas 11

LPEMM

Real Decreto Legislativo 2/2011, de 5 de septiembre, por el que se aprueba el Texto Refundido de la Ley de Puertos del Estado y de la Marina Mercante ... 10, 11, 12, 82, 83

LUN

Dirección para una unidad lógica de disco duro 19

M

MVC

Modelo de arquitectura Modelo, Vista y Controlador (Model-View-Controller) 21

O

ORM

Object-Relational Mapping - Mapeo Objeto-Relacional 13

S

SAN

Red de área de almacenamiento 19

SLA

Service Level Agreement 20

SSSM

Sociedad de Salvamento y Seguridad Marítima 83

T

Tasa T0

Tasa Portuaria de Ayudas a la Navegación, regulada por la LPEMM 10

TDD .... *Véase* https://www.agilealliance.org/glossary/tdd/

W

WAMP

Acrónimo de servicios Apache, MySQL y PHP para Windows 47

X

XSS

Cross-Site Scripting 56

14. Bibliografía

1. **Taylor Otwell**. Documentación oficial de Laravel. Laravel.com. Online. Consulta: 20/03/2018. https://laravel.com/docs/5.6

2. Traducción de la documentación oficial de Laravel por la Comunidad de Laravel en español. Laraveles.com. Online. Consulta: 20/03/2018. https://docs.laraveles.com/docs/5.5

3. **Rafael Cuevas** y **Osvaldo Cordova**. Richos. "Introducción a Laravel 5". GitBook. Online. Consulta: 20/03/2018. https://richos.gitbooks.io/laravel-5/content/

4. **Laracasts**. "Laravel 5 Fundamentals". Online. Consulta: 20/03/2018. https://laracasts.com/series/laravel-5-fundamentals

5. **Laracasts**. "Laravel 5.4 From Scratch".Online. Consulta: 20/03/2018. https://laracasts.com/series/laravel-from-scratch-2017

6. **Duilio Palacios**. "Curso de Laravel 5.5 desde cero". Styde.net. Online. Consulta: 20/03/2018. https://styde.net/laravel-5/

7. **Evan You**. Documentación oficial de Vue.js. Vuejs.org. Online. Consulta: 20/03/2018. https://vuejs.org/v2/guide/

8. **Laracasts**. "Learn Vue 2: Step By Step".Online. Consulta: 20/03/2018. https://laracasts.com/series/learn-vue-2-step-by-step

9. **Duilio Palacios**. "Curso de Vue 2". Styde.net. Online. Consulta: 20/03/2018. https://styde.net/curso-de-vue-2/

10. **Nathan Wu**. 2015-2016. "Learning Laravel 5: Building Practical Applications". Learninglaravel.net

11. **Nathan Wu.** 2015-2016. "Laravel 5 Cookbook: Enhance Your Amazing Applications". Learninglaravel.net

12. **Kelt Dockins**. 2017. "Design Patterns in PHP and Laravel". Apress.

13. **Martin Fowler**. 1996. "Analysis Patterns: Reusable Object Models". Adison-Wesley Professional.

14. **Daniel Torrico, Héctor Alonso** y **Marco Marín**. 2012. “Seguridad y calidad en servidores web”. FUOC.

15. **Jordi Pradel** y **Jose Raya**. 2016. “Ingeniería del software”. FUOC. PID_00230156.

16. **Jordi Cabot, Isabel Guitart, Jordi Pradel** y **José A. Raya**. 2010. “Análisis y diseño con patrones”. FUOC. PID_00160153.

17. **Jordi Arnedo** y **Daniel Riera**. 2010. “Diseño y programación orientada a objetos”. FUOC. PID_00160185.

18. **Tona Monjo Palau**. 2011. “Diseño de interfaces multimedia”. FUOC. PID_00159830.

19. “Real Decreto Legislativo 2/2011, de 5 de septiembre, por el que se aprueba el Texto Refundido de la Ley de Puertos del Estado y de la Marina Mercante”. BOE-A-2011-16467. Online. Consulta: 20/03/2018. https://www.boe.es/buscar/act.php?id=BOE-A-2011-16467

20. “Real Decreto 1027/1989, de 28 de julio, sobre abanderamiento, matriculación de buques y registro marítimo”. BOE A-1989-19704. Online. Consulta: 20/03/2018. https://www.boe.es/buscar/doc.php?id=BOE-A-1989-19704

21. “Real Decreto 1435/2010, de 5 de noviembre, por el que se regula el abanderamiento y matriculación de las embarcaciones de recreo en las listas sexta y séptima del registro de matrícula de buques”. BOE-A-2010-17038. Online. Consulta: 20/03/2018. https://www.boe.es/buscar/doc.php?id=BOE-A-2010-17038

22. “Reglamento de Ejecución (UE) 2017/1, de la Comisión, de 3 de enero de 2017, sobre los procedimientos de identificación de embarcaciones con arreglo a la Directiva 2013/53/UE del Parlamento Europeo y del Consejo, relativa a las embarcaciones de recreo y a las motos acuáticas”. BOE. DOUE-L-2017-80001. Online. Consulta: 20/03/2018. http://www.boe.es/buscar/doc.php?id=DOUE-L-2017-80001

23. “Ley 39/2015, de 1 de octubre, del Procedimiento Administrativo Común de las Administraciones Públicas”. BOE-A-2015-10565. Online. Consulta: 20/03/2018. https://www.boe.es/buscar/act.php?id=BOE-A-2015-10565

15. Anexos

Anexo 1. Presupuesto

Tras el análisis del alcance de las funcionalidades requeridas y los recursos de Hardware y conexión a Internet que dispone esta Autoridad Portuaria para el servicio de publicación web de su sede electrónica, servicio en el que se alojará la aplicación objeto de la presente licitación, a continuación, se detalla la oferta económica para el desarrollo del presente presupuesto:

Equipo HW para programación + Licencias sistema operativo	853,25 €
Instalación y configuración del entorno de programación web	100,00 €
Análisis de los modelos de datos y de clases	550,00 €
Diseño front-end	1250,00 €
Desarrollo back-end	1500,00 €
Pruebas unitarias del sistema	500,00 €
Configuración de servidores de publicación web de la sede para el alojamiento de la nueva aplicación	200,00 €
Elaborar manual de uso y formación a empleados públicos	500,00 €
SUBTOTAL	5.453,25 €
IVA (21%)	1.145,18 €
TOTAL	**6.598,43 €**

Este presupuesto está vinculado a que no haya cambios en la normativa legal (como la LPEMM) en la que se basa la lógica de la aplicación web a desarrollar. Si durante el desarrollo de la misma se produce algún cambio en la normativa, se negociará un aumento de presupuesto o la ampliación del plazo de entrega para abordar estos cambios.

Garantía: 2 años tras la entrega de la aplicación.

Contingencias cubiertas: Problemas de funcionamiento de la aplicación.

Contingencias exentas: Mejoras propuestas por esta AP, modificaciones por cambio de normativa legal que regula el procedimiento, problemas por el mal uso de la aplicación, y los que puedan surgir por cambios en la plataforma de servicio para la que ha sido diseñada.

Todas las contingencias cubiertas no tienen coste alguno para esta AP, mientras que las exentas que nos sean reclamadas serán objeto de un nuevo presupuesto para su gestión.

Anexo 2. Funcionalidades requeridas.

Tras el estudio de las funcionalidades requeridas en el pliego de condiciones del concurso público para el desarrollo de la aplicación web objeto de este proyecto, se limita el alcance del mismo a las siguientes funcionalidades:

1. Formulario web de autoliquidación

- El formulario web será visible para el gran público general, independientemente de si se ha autenticado contra el sistema o no.
- El formulario seguirá la casuística de la normativa vigente en el momento del análisis de modelado de la aplicación, para embarcaciones de recreo y para pesqueros de litoral.
- Como resultado del cálculo de cada caso, se mostrará el importe de la Tasa T0 y su desglose respecto a qué organismo público va destinado (AP y SSSM). También indicará en su caso la exención del pago de la tasa.
- En el envío del mismo se comprobará que los campos obligatorios estén completados y la correcta introducción de los campos con formato establecido, como el NIF o el email.

Casuística de cobro:

La LPEMM, en su artículo 137, define y regula el servicio de señalización marítima, en el 163 y 241 bis indica que las tasas recaudadas por las AAPP deben sufragar sus costes y los de la Sociedad de Salvamento y Seguridad Marítima (SSSM), en el 164 se establece que el importe de la T0 es fijo para todo el sistema portuario estatal, y el 171 indica las exenciones del pago de la tasa. Por último, en los artículos 237 a 244 se regula la casuística para el cálculo de la Tasa T0.

A continuación, se resume el valor de esta tasa para cada caso concreto:

Art. 240.2. Cuantía básica A (AAPP): 0,29€. Cuantía básica C (SSSM): 0,28€. A y C son revisables en los Presupuestos Generales del Estado.

Art. 240.1.c) Embarcaciones de pesca de litoral:

- Con base en puerto español: (A+C) * 50 = **28,50€**, en cada año natural.
- Base en puerto no español: 28,50€ / nº días del año * nº de días de estancia

Art. 240.1.d) Embarcaciones de recreo, matriculadas (no inscritas), a motor y eslora ≥ 9 metros, o a vela y eslora > 12 m:

- Con base en puerto español: (A+C) * eslora(m) * manga(m) * 16, en cada año natural.
- Base en puerto no español: [(A+C) * eslora(m) * manga(m) * 16] / nº días año * nº días estancia

Art. 240.1.e) Embarcaciones de recreo, matriculadas (no inscritas), a motor y eslora < 9 m:

- Con base en puerto español: (A+C) * eslora(m) * manga(m) * 40, una sola vez: validez indefinida.
- Base en puerto no español: [(A+C) * eslora(m) * manga(m) * 40] / nº días año * nº días estancia

Art. 171.e) Embarcaciones a vela con eslora ≤ 12 metros:

- Exentas del pago de esta tasa.

2. Sitio web privado para usuarios obligados al pago de esta tasa

- Registro de usuario.
- Modificación del perfil de usuario: fotografía, email, teléfono, dirección y cuenta bancaria.
- Registro de embarcaciones de su titularidad: fotografía, NIB (número de identificación del buque), nombre, matrícula, propulsión, eslora y manga.
- Consulta de autoliquidaciones anteriores.

3. Sitio web para personal de la Autoridad Portuaria

- Modificación de cuantías básicas A y C, así como de los coeficientes de cada caso, para cada actualización de los Presupuestos Generales del Estado.
- Consulta de usuarios, de embarcaciones, de las embarcaciones de cada usuario y de los titulares anteriores de cada embarcación.
- Consulta de autoliquidaciones. Posibilidad de filtrar por año y por importe.
- Marcar autoliquidaciones como pagadas, indicando la fecha de pago.
- Inclusión de la factura o del certificado de exención, según el caso, para su descarga por parte del interesado.

Otras consideraciones

- Las autoliquidaciones tienen 5 **estados** posibles:
    1. email no verificado: Se registra la solicitud y se envía el email de verificación del correo. Cuando el usuario accede al correo y pulsa el botón de verificación en el mensaje, se le envía el documento de pago en pdf y pasa a estado Pendiente.
    2. Exento: La autoliquidación da como resultado esta situación.
    3. Pendiente: Dentro del plazo de pago, sin que este se haya realizado.
    4. Expirado: Fuera del plazo de pago, sin que este se haya realizado.
    5. Pagado: Fin del escenario principal de éxito de los casos de uso.
- Una **autoliquidación** puede ser **complementaria** de otra que no se realizó correctamente, por la indicación de datos incorrectos para su cálculo: tipo de embarcación, eslora, manga, propulsión o año. En este caso, se debe indicar el código de la primera autoliquidación para que el formulario, con los valores correctos introducidos, calcule el importe a pagar o a devolver.
- Las embarcaciones deben admitir cambios de todos sus parámetros, salvo de la referencia del casco y, si fuese española, del NIB, y debe quedar registro de estos cambios.
- Las embarcaciones pueden tener uno o varios propietarios, pero solo uno de ellos puede autoliquidar la tasa en cada año natural.
- Las embarcaciones pueden cambiar de propietario/s a lo largo de su vida útil.

Anexo 3. Cambios respecto a la PEC2.

Tenía previsto eliminar este anexo en la entrega final, pero finalmente he decido incluirlo como prueba de las diversas contingencias y decisiones que se han tomado durante el desarrollo.

Se actualiza la planificación en esta memoria. Como se explica en ese apartado, hay un retraso en el desarrollo, sobre todo debido a la curva de aprendizaje de la integración entre Laravel y Vue.

Queda pendiente la revisión de los casos de uso, ya que, como se ha explicado en la modificación de la planificación, en última fase de este proyecto aún tendré que desarrollar bastantes funcionalidades y, seguramente, vuelva a haber algún cambio.

El siguiente párrafo se vuelve a redactar en la entrega final para su mejor comprensión.

En la PEC3 he concluido que se deben incluir un estado para comprobar si el email indicado en el formulario de autoliquidación ha sido comprobado antes de enviar el documento de pago.

En el registro de usuario se han de indicar email y documento de identidad, y deberá de existir alguna autoliquidación previa con estos datos para que el usuario pueda registrarse en el sistema. De esta forma, cuando el usuario se registra no es necesario comprobar que el email es real, puesto que ya se hizo en el envío del documento de pago de la autoliquidación previa.

El registro de administrativos de esta APC se realizará desde el Dashboard, mediante el usuario administrador inicial, cuyas credenciales son: **admin@example.com** y contraseña: **123456**

Se recomienda eliminar o, al menos, cambiar la contraseña de este usuario.

Respecto al plazo de pago, independientemente de que se reciba el email de confirmación, los 30 días naturales comienzan el día del registro de la solicitud.

Tal como recomienda Ignasi en la revisión de la PEC anterior, se incluyen las historias de usuario.

Respecto al modelo UML de clases, al tratar de implementar un modelo de datos ER a partir del modelo de clases de la PEC2 he tenido problemas para implementar los patrones de diseño. Mi conclusión ha sido que en este proyecto solo aportaban complejidad al añadir más modelos y, por tanto, tablas de la base de datos asociadas a los mismos.

Así pues, he optado por simplificar el modelo de clases al nuevo que aporto en esta PEC3, incluyendo una única fecha para solicitud, declarante, versión de buque y tabla de relación entre declarante y buque, en la que figura también el porcentaje de propiedad. No se indica fecha de inicio y fin de posesión ni de versión de buque porque no es un dato que se aporte en la autoliquidación. En cada solicitud se obtiene la posesión y versión de buque en ese instante, y por eso se indica una sola fecha, de las que se guarda historial como se requería.

En el nuevo modelo de datos también separo los conceptos de usuario y de declarante, ya que un declarante puede enviar una solicitud de autoliquidación sin estar registrado, y un administrativo no tiene por qué tener autoliquidaciones. Por tanto, creo la clase `User` para representar y guardar los datos de usuarios registrados, y separo los datos del declarante en una clase asociada.

El patrón estado también aportaba más complejidad al proyecto de la necesaria. Finalmente controlaré los estados desde la lógica de la aplicación, por lo que solo se requerirá una clase que represente los estados posibles para las solicitudes de autoliquidación.

Para las autoliquidaciones tampoco es necesario guardar toda la información en esta clase. Ésta queda almacenada en los objetos creados y relacionados a través de la clase que representa a la solicitud: `T0Fee`.

Para poder habilitar el cálculo de embarcaciones transeúntes para embarcaciones de bandera no española, es preciso poder registrar este tipo de embarcación, que al no tener matrícula española no tienen NIB. Por tanto, se hace necesario otro valor de referencia para las mismas: el WIN (*watercraft identification number*) es un número de serie grabado en el casco de las embarcaciones. `Boat` y `BoatVersion` se unen a través de una nueva relación a través de la clase `SpanishBoat`, cuyo único atributo es el `nib`. De esta forma se pueden identificar las embarcaciones extranjeras.
Se extrae el atributo nib a una nueva clase porque se da el caso de que en Capitanía Marítima pueden requerir el pago de la T0 como requisito previo al procedimiento de abanderamiento o matriculación de una embarcación extranjera o de una nueva embarcación española. En este momento, la embarcación aún no dispone de NIB ni de matrícula, por lo que el campo nib no puede ser obligatorio y único para la clase `Boat`.

Se incluyen las tablas del modelo de datos ER conforme al nuevo modelo de clases.

Se desarrollan los puntos requeridos para la PEC3.

Anexo 4. Cambios respecto a la PEC3.

En esta última fase he comprendido la importancia de los métodos de desarrollo ágiles, en el que todas las fases de la dirección del proyecto son cíclicas: desde la planificación y el modelado hasta el desarrollo y las pruebas.
También la necesidad de una buena planificación inicial, con un cálculo de horas de trabajo más realista, que hubiera requerido mayor tiempo de familiarización con los frameworks a tratar, sobre todo teniendo en cuenta su total desconocimiento. Había muchas nociones que he tenido que interiorizar, como el modelo MVC y otros, que también he explicado en esta memoria.

En esta última fase, según la nueva planificación del último ciclo de la PEC3, se vuelve a modificar el modelo UML de clases, tal como se explica en ese apartado.

Se termina la validación del formulario de autoliquidación que se comenzó en la PEC anterior.
También se incorpora al framework la librería necesaria para la generación del documento de pago en PDF. Para ello se implementa la vista necesaria y una ruta para visualizar esta vista y optimizar su diseño, como se explica en el apartado correspondiente.

Se implementa lo necesario para incluir archivos en el envío del formulario para que sean guardados en el servidor y que puedan ser consultados fácilmente, a través de la aplicación objeto de este desarrollo, pero también a través del sistema de archivos, estableciendo nombres para directorios y archivos que faciliten esta labor.

Se crea una cuenta en el servicio de correo externo Mailtrap, se configura esta cuenta en Laravel, y se desarrolla la clase y la vista necesaria para poder realizar el envío desde el controlador del formulario del archivo de documento de pago.
No ha sido posible terminar la comprobación del email introducido en el formulario de autoliquidación por falta de tiempo. No obstante, según la información consultada[41] sobre este asunto para su desarrollo, no debería de haber problema en su implementación.

Por último, se optimizan las vistas del Dashboard, se comienza con la separación del contenido de este entorno según el rol de usuario, que se controla mediante el atributo booleano `civil_servant` de la clase `Applicant`.
Cada una de estas tareas supone un nuevo ciclo de desarrollo que repercute en rutas, vistas y controladores existentes, validación del formulario, etc.

[41] Compobación de email. Blog de Programación y más: https://programacionymas.com/blog/confirmar-email-laravel