Trabajo de Fin de Grado

Open Data en el sector marítimo portuario

Open Data in the maritime port sector

Adrián González Hernández

La Laguna, 27 de junio de 2018

D. **Jose Luis Roda García**, con N.I.F. 43.356.123-L, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas, de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“Open Data en el sector marítimo portuario”

ha sido realizada bajo su dirección por D. **Adrián González Hernández**,

con N.I.F. 78574523-Z.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 27 de junio de 2018

Agradecimientos

Primero agradecer a Airam Rodríguez, por haberme dado una oportunidad para comenzar en el mercado laboral, por darme una oportunidad para sacar adelante esta idea, y por todo el camino que nos queda por recorrer.

A mi pareja, por estar siempre ahí, por ser mi mayor apoyo, y no haberme permitido que me rindiera nunca.

A mis compañeros de Híades, con quiénes he pasado tanto tiempo que al final acaban siendo parte de tu familia.

A mi familia, amigos y compañeros de carrera, porque sin ellos no sería la persona que soy hoy.

A José Luis, mi tutor, porque su ayuda ha sido imprescindible para que esto saliera adelante de la mejor manera posible.

Y en general, a todos aquellos que han sido capaces de aguantarme a lo largo de estos años.

Licencia

Resumen

El objetivo de este trabajo ha sido el desarrollo de una herramienta que permita mejorar la seguridad de las operaciones marítimas, centrándose inicialmente en la labor de los prácticos de puerto, pero siendo extensible a cualquier agente implicado en operaciones marítimas o portuarias. Con este fin se ha tratado de aprovechar el potencial que ofrece la filosofía de datos abiertos, y se ha tratado de conseguir la irrupción de las nuevas tecnologías en un sector tan tradicional.

Durante la introducción se abordará el estado actual del sector, se expondrá el marco de colaboración bajo el cual se realizará este proyecto junto con la empresa Híades, y los conceptos necesarios para entender la filosofía Open Data. También se expondrán los objetivos concretos y los resultados esperados: Un portal web para la consulta de datos abiertos de incidencias marítimas, la integración con la solución Amura Pilots, y un prototipo de aplicación que a futuro permita disfrutar de las ventajas de esta solución y sus mejoras en la seguridad de la operativa en dispositivos móviles. A continuación se mostrarán los procesos de análisis, toma de requisitos, y desarrollo que han conducido a la consecución de los objetivos propuestos.

El prototipo desarrollado ofrecerá un conjunto de APIs, implementadas bajo la filosofía Open Data, que permitirá mejorar la transparencia, incrementando de esta forma la seguridad en las operaciones portuarias como el practicaje o el amarre.

Palabras clave: Seguridad Marítima, API, .Net Framework, buque, práctico

Abstract

The objective of this work has been the development of a tool to improve the safety of maritime operations, initially focusing on the work of port pilots, but being extended to any agent involved in maritime or port operations. To this end, we have tried to take advantage of the potential offered by the open data philosophy, and we have tried to achieve the emergence of new technologies in such a traditional sector.

During the introduction, the current state of the sector will be discussed, the framework of collaboration under which this project will be carried out together with the company Híades, and the necessary concepts to understand the Open Data philosophy. The specific objectives and expected results will also be exposed: A web portal for consulting open data on maritime incidents, integration with the Amura Pilots solution, and a prototype application that in the future allows to enjoy the advantages of this solution and its improvements in the security of operations on mobile devices. The following will show the processes of analysis, taking of requirements, and development that have led to the achievement of the proposed objectives.

The developed prototype will offer a set of APIs, implemented under the Open Data philosophy, which will improve transparency, thus increasing security in port operations such as pilotage or mooring.

Keywords: Maritime security, API, .Net Framework, vessel, pilot

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

El mundo marítimo es quizás el más internacional de todos los grandes sectores comerciales del mundo, y uno de los más peligrosos. Parte de esa peligrosidad ha venido desde el hermetismo que existe en este sector para el intercambio y utilización de los datos relativos a las operaciones marítimas entre los profesionales del sector. Por otro lado, en los últimos años se ha comenzado a aceptar que la manera más adecuada de mejorar la seguridad en el mar es a través de la introducción de nuevas herramientas e innovaciones que faciliten y mejoren esa seguridad en la operativa portuaria.

Actualmente, la seguridad en el tráfico marítimo es el objetivo y responsabilidad de los operadores portuarios que brindan su servicio de asesoramiento a las embarcaciones que navegan las aguas restringidas con el objetivo de proteger los recursos naturales, el medio ambiente y los bienes de los ciudadanos.

Bajo esta línea se busca a través de este proyecto desarrollar herramientas tecnológicas que permitan intercambiar datos de gran valor entre los operadores portuarios, concretamente datos relativos a los buques que comprometan la seguridad del atraque, complementada con datos particulares de cada puerto, tales como calados y elementos de especial interés.

1.1 Antecedentes y estado actual

Actualmente ni siquiera entre los puertos de un mismo país se intercambia información relativa a incidencias o accidentes más de los que aparecen en los medios informativos, o los que pueden surgir en los congresos profesionales que se celebran anualmente. En el escenario actual las deficiencias de un buque son localizadas a la llegada del buque a puerto con lo que los profesionales no pueden realizar tareas de prevención de accidentes marítimos.

Por otro lado, la tecnología y la innovación han comenzado a conseguir en los últimos años un avance enorme en productividad en el sector marítimo que ha significado que con las mismas infraestructuras se puede conseguir muchos más movimientos manteniendo los niveles de seguridad. Pero aún este sector está por detrás por ejemplo del sector aéreo, mucho más habituado al uso de tecnologías como Big Data, el internet de las cosas (IoT) o tecnologías móviles que están todavía en sus comienzos.

Con el desarrollo de este proyecto se busca conseguir a través del sistema OpenData MICSI (Maritime Incidents Cloud & Ships Interoperability) un marco tecnológico que permita la utilización y apertura de esa gran cantidad de datos, que abren un modelo disruptivo a nivel internacional donde se podrán aplicar modelos predictivos y mejorar la eficiencia de los servicios Portuarios. Encontramos también que muchos de los servicios prestados por los agentes técnico-marítimos están basados en documentos físicos denominados vouchers, que se trasladan con los datos del buque, y que hacen firmar al cliente. En este caso la integración de este sistema con tecnologías móviles permitirá presentar un reemplazo eficiente tanto en la forma de visualización, como en el registro de los datos de los buques y sus observaciones de seguridad, pudiendo así transmitir en tiempo real toda esta información.

1.2 Marco de colaboración

Es en el análisis de este contexto donde surge la oportunidad de colaboración con la empresa Híades Bussines Patterns, empresa de desarrollo web y móvil, y consultoría informática, establecida en Santa Cruz desde 2010, y que se encuentra en estrecha relación con los agentes portuarios a través del desarrollo de los distintos productos de Amura Solutions, y sobretodo su principal producto Amura Pilots, una solución para la gestión integral de los servicios de practicaje.

Desde Híades consideran que una integración de sus servicios con el desarrollo de este proyecto podría suponer una importante mejora para la seguridad de los prácticos.

1.2.1 ¿En qué consiste el practicaje? Agentes implicados

Antes de conocer más sobre Amura Pilots o el desarrollo de Open Data MICSI, sería interesante conocer los agentes implicados, y cuál es la labor de los prácticos de puerto.

El practicaje es un servicio de asesoramiento, que se utiliza para facilitar y garantizar las condiciones de seguridad de las operaciones de entrada, salida y maniobras náuticas, dentro de los límites de la zona de practicaje. El servicio será prestado por un práctico, a bordo del buque, y comienza desde que se imparten las primeras instrucciones desde la estación de practicaje. Sin embargo, durante el servicio, corresponde al capitán del buque el mando y la dirección de cualquier maniobra.

Por tanto, los prácticos son los agentes encargados de prestar este servicio de asesoramiento, para lo cual deberán realizar las maniobras de embarque y desembarque al buque, las cuales implican un alto riesgo, por lo que se hace imprescindible conocer con el mayor detalle posible las cuestiones relativas a la seguridad de la embarcación, siendo una de las más frecuentes el mal estado de la escalerilla, por la cuál tendrá que acceder el práctico a la nave, sobretodo en buques que proceden de países donde las normativas de seguridad son más laxas.

Por último, comentar que para poder acceder al puesto de práctico no sólo

se requerirá la realización de los exámenes establecidos, sino que además será necesario haber ejercido el mando de buques mayores de 1000 GT's (GT es la unidad que se emplea para medir el tamaño de los buques, a partir de su volumen) durante dos años en los últimos diez.

Otros agentes implicados en la operativa del practicaje son los Consignatarios, quienes actúan en los puertos como intermediarios de los propietarios de los buques, y ejecutan las fases terrestres del transporte marítimo.

Por último, la autoridad portuaria es un organismo público en España, dependiente del Ministerio de Fomento, y que gestiona en coordinación con el ente público Puertos del Estado, los distintos puertos del país. Actualmente 28 autoridades portuarias gestionan los 45 puertos marítimos españoles de interés general. Son por tanto, otra de las partes más interesadas en el control de la seguridad de aquellas embarcaciones que se dispongan a operar en sus puertos.

1.3 Amura Solutions

El desarrollo de Amura surge en 2013 como una oportunidad para cubrir una necesidad específica de la Corporación de Prácticos de Puertos de Tenerife, los cuales necesitan un software capaz de resolver graves problemas tanto de gestión como de eficiencia.

Una vez puesto en marcha el funcionamiento en el puerto de Santa Cruz de Tenerife, se detecta que son muchas otras las corporaciones de prácticos que se enfrentan a estos problemas:

- Descontrol entre lo operativo y lo contable.
- Conflictos con la autoridad portuaria.
- Duplicidad de procesos.
- Problemas operativos de movilidad.
- Conflictos internos y con el ecosistema portuario.
- Conflictos con consignatarios.

Es aquí donde surge Amura Pilots como un producto para la gestión administrativa y operativa, enfocada en la flexibilidad que le permita adaptarse a las distintas corporaciones de prácticos, permitiendo a sus clientes:

- La gestión eficiente del practicaje.
- Planificación de las maniobras de buques.
- Gestión del libro de servicios, con un registro de todas las operaciones realizadas y su estado actual.
- Gestión de embarcaciones, y del personal de la corporación. Permitiendo controlar tareas como la planificación de turnos, el número de servicios de los buques, o las horas trabajadas por sus

empleados.

- Control de cobros y vencimiento de facturas de los distintos consignatarios.
- Un completo sistema de facturación y tarificación.
- Cuadros de mando con indicadores de tráfico, facturación y rendimiento.

1.3.1 Algunos datos sobre Amura

Estos datos muestran por qué resulta tan interesante la colaboración con Amura en este proyecto.

Figura 1-1: Clientes de Amura

Actualmente Amura Pilots está siendo utilizado a diario por las siguientes corporaciones:

- Corporación de prácticos de Puerto de Tenerife SLP
- Prácticos de Melilla
- Corporación de prácticos de Las Palmas
- Prácticos de Valencia
- Pilots. Bahía de Cádiz
- Tarragona Pilots

Aunque a lo largo de 2018 se espera que se incorporen también Prácticos de Fuerteventura, Prácticos de El Hierro, Palma Pilots, y se encuentra en negociaciones con otras corporaciones españolas, así como con prácticos de Holanda, Croacia y Latinoamérica.

Otros datos que reflejan la importancia de la solución:

En 2017 se registraron 70000 maniobras de practicaje en las distintas corporaciones, y se espera que esas cifras crezcan en 2018.

Figura 1-2: Maniobras registradas

Y estos indicadores muestra algunas mejoras en la operativa diaria de los clientes que usan Amura.

Figura 1-3: Indicadores clientes

1.3.2 Un vistazo a Amura

Las siguientes figuras mostrarán algunas capturas de módulos actuales de Amura Pilots:

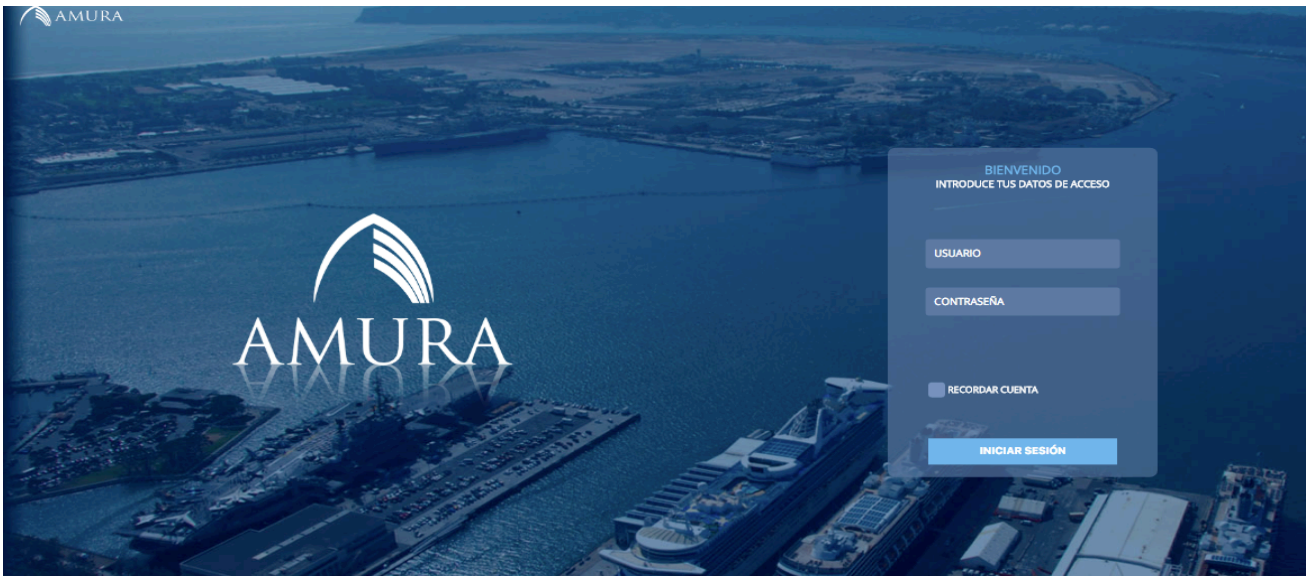


Figura 1-4: Página de Inicio de Amura Pilots

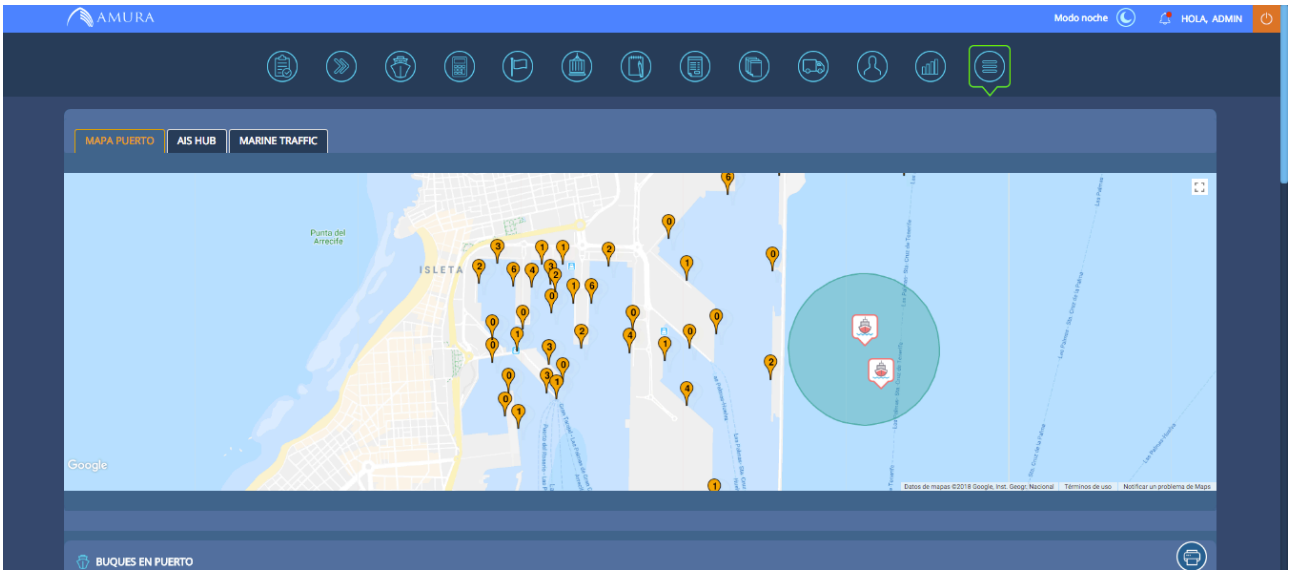


Figura 1-5: Mapas de puerto del Puerto de Las Palmas

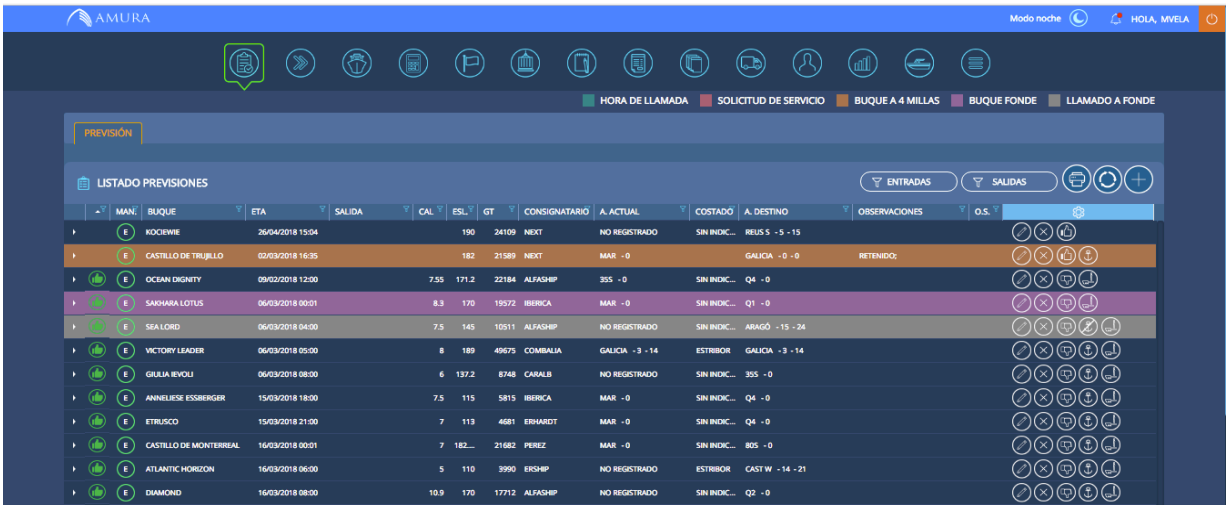


Figura 1-6: Tablón de previsiones

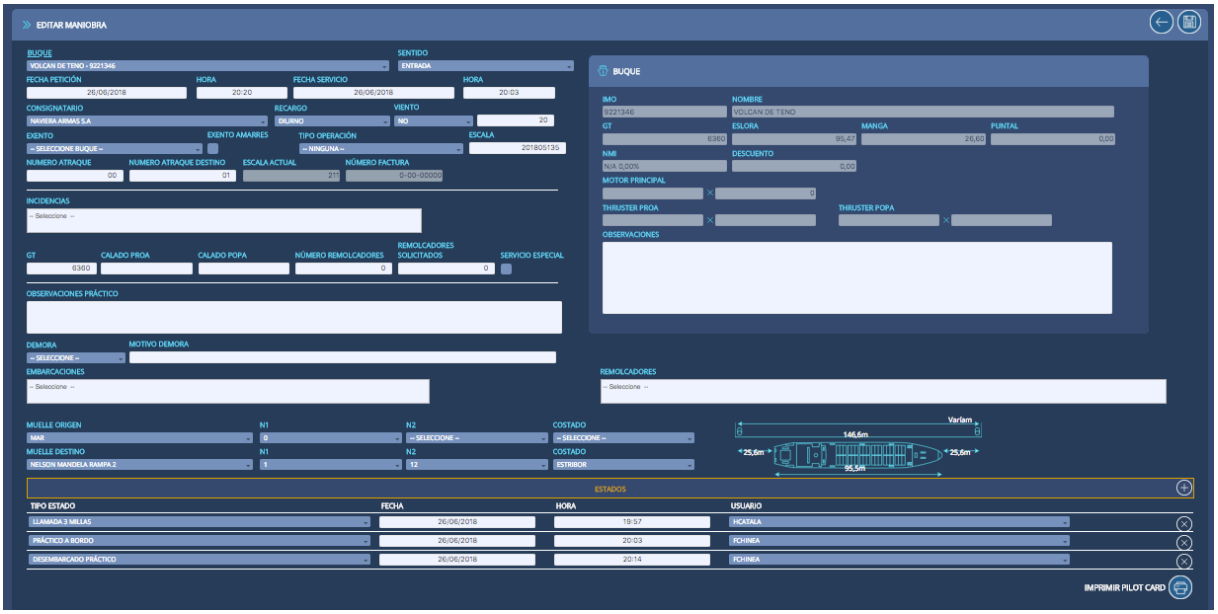


Figura 1-7: Ficha de una maniobra

1.3.3 Otros productos de Amura Solutions

El crecimiento del producto ha permitido a Híades enfocarse en otros agentes dentro del ámbito portuario. Así es cómo han surgido y se encuentran en pruebas Amura Amarradores y Amura Remolcadores, ambas soluciones parten de Amura Pilots, y permitirán la integración con la misma, permitiendo así un ecosistema conectado entre las distintas entidades del mismo puerto. A continuación, algunas pantallas de ambas soluciones:

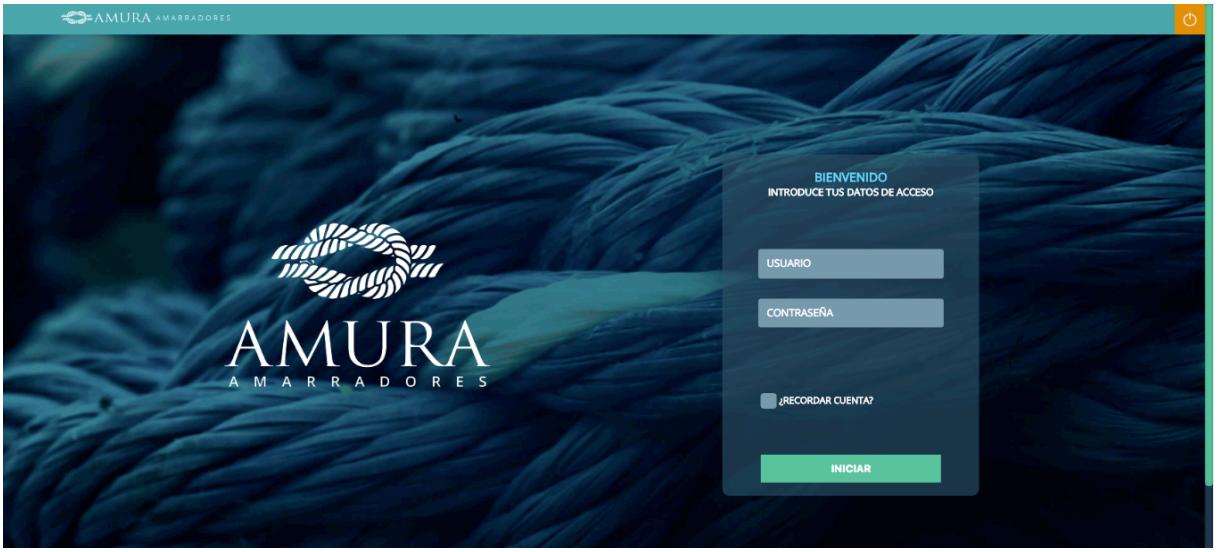


Figura 1-8: Página inicio Amura Amarradores

Figura 1-9: Libro de Servicios de Amura Amarradores

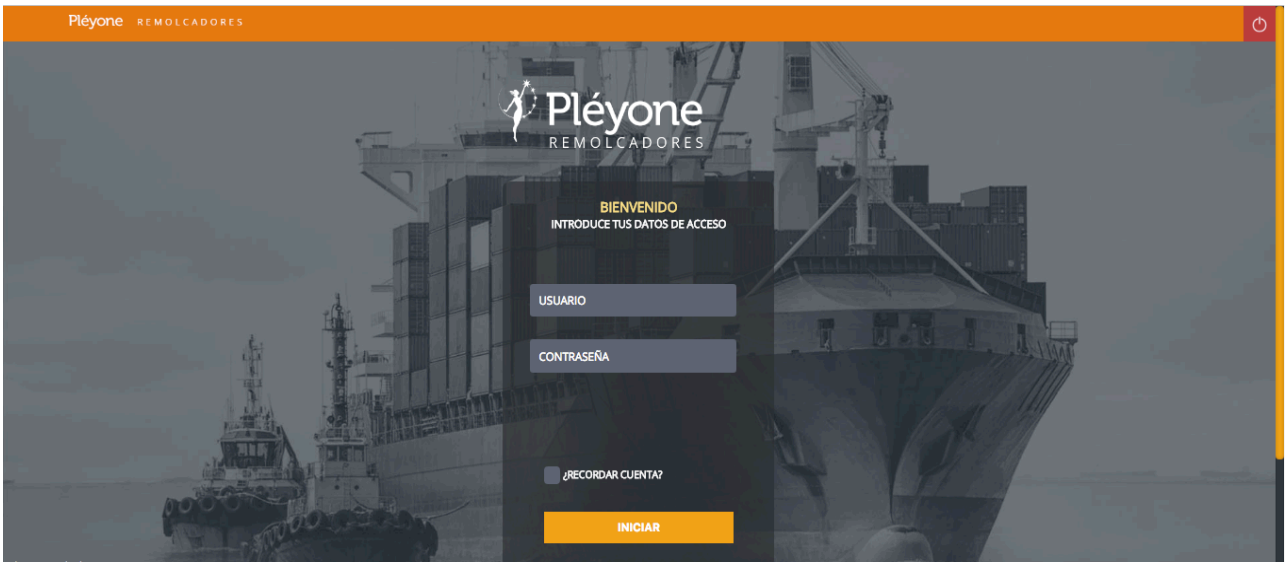


Figura 1-10: Página de inicio de Remolcadores

1.4 Open Data

Conociendo ya el potencial origen de datos, y su importancia en la mejora de la seguridad, el objetivo era facilitar el acceso a cualquier persona o entidad interesada en los mismos, en cualquier momento y de forma sencilla.

De esta forma entendíamos que la filosofía Open Data podía encajar tanto con los objetivos del proyecto como con futuros requisitos de acceso y reutilización que podrían plantear las Autoridades Portuarias en cada caso.

1.4.1 Qué es Open Data

Se considera open data o datos abiertos, a la filosofía cuyo objetivo es el de promover el acceso de forma libre y sin restricciones, a determinados tipos de datos, normalmente generados por Administraciones Públicas. Aunque se espera que con el tiempo haya una mayor apertura de los datos por parte de entidades privadas.

Por tanto, para que los datos puedan considerarse datos abiertos deben cumplir las siguientes consideraciones:

- **Disponibilidad y acceso:** Estos datos deben estar accesibles a través de un protocolo estándar, y que no debe suponer un coste adicional. Además deben volcarse en formatos ampliamente extendidos que faciliten su procesamiento. Debe evitarse cualquier formato propietario, como, por ejemplo, tablas Excel (.xls, .xlsx).
- **Reutilización y redistribución:** Los datos deben proporcionarse de forma que se permita reutilizarlos y redistribuirlos, e incluso integrarlos con otros conjuntos de datos. Para este fin los datos deberían ser lo más detallados, presentarse con el mayor nivel de granularidad posible, y, proporcionarlos sin ningún tipo de filtro o procesamiento.

- **Participación universal:** Cualquier persona interesada debe poder acceder a ellos, sin excepción y sin ningún tipo de restricción comercial. Por tanto, no deberá requerirse ningún tipo de registro o identificación previa.

1.4.2 Open Data en España

Observando los datos de OpenDataBarometer.org, podemos comprobar como en 2016, España ocupaba la 11ª posición a nivel mundial en un barómetro que evalúa cómo los gobiernos están publicando y utilizando datos abiertos para la transparencia, la innovación y el impacto social.

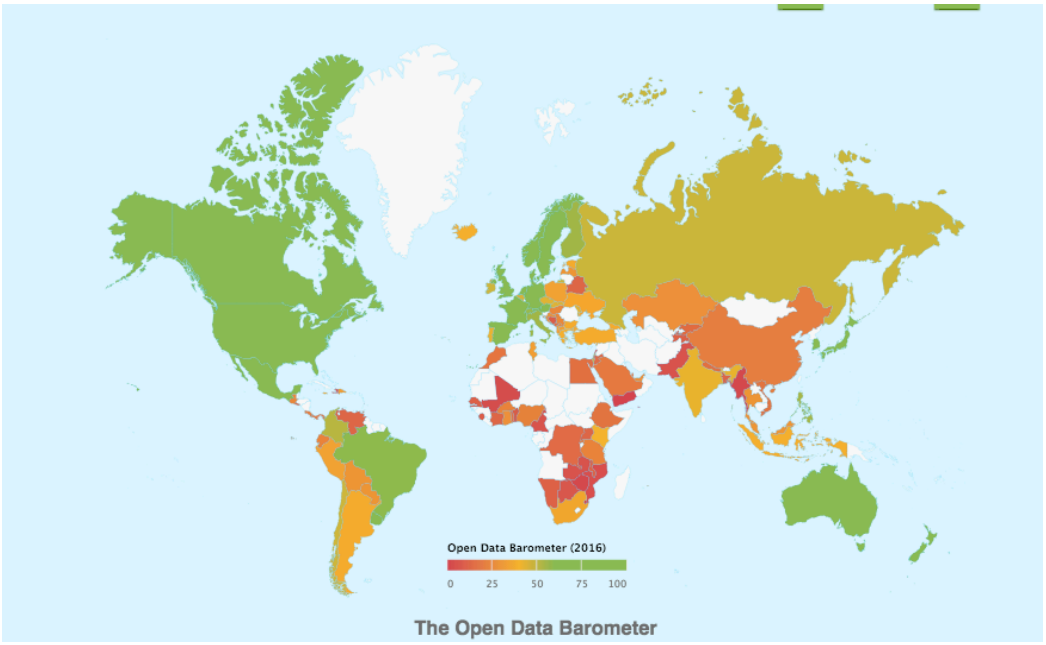


Figura 1-11: Mapa mundial Open Data Barometer 2016

Country	Rank	Score OUT OF 100	Change	Score Trend OVER PAST EDITIONS	Readiness OUT OF 100	Implementation OUT OF 100	Emerging Impact OUT OF 100
United Kingdom See details	1	100	0 -		99	100	94
Canada See details	2	90	2 ▲		96	87	82
France See details	3	85	-1 ▼		100	71	88
United States of America See details	4	82	-2 ▼		96	71	80
Korea See details	5	81	3 ▲		95	59	100
Australia See details	5	81	5 ▲		85	78	78
New Zealand See details	7	79	-1 ▼		92	58	99
Japan See details	8	75	5 ▲		84	60	89
Netherlands See details	8	75	-1 ▼		94	64	68
Norway See details	10	74	7 ▲		77	71	73
Mexico See details	11	73	5 ▲		83	58	88
Spain See details	11	73	2 ▲		81	58	88
Denmark See details	13	71	-8 ▼		67	71	71
Austria See details	14	70	-1 ▼		83	56	78
Sweden See details	14	70	-5 ▼		87	70	47
Germany See details	14	70	-3 ▼		67	69	71

Figura 1-12: Open Data Barometer. Ranking de países

Sin embargo, el sector privado no se encuentra afectado por el marco normativo sobre la reutilización de la información, por lo que, pese a la

irrupción del Internet de las Cosas y el Big Data, existen aún pocas iniciativas de datos abiertos que partan desde instituciones privadas.

A pesar de esto, cifras de 2015 situaban el volumen de negocio de la reutilización de datos abiertos en torno a 1700 millones de euros, y en torno a unos 13000 empleos. Por lo que se mira con optimismo el crecimiento de este sector. [1]

1.5 Objetivos del proyecto

El objetivo principal de este proyecto es la creación de un portal Open Data para la gestión de incidencias marítimas y de buques, el cual se conocerá por la siglas MICSI (del inglés Maritime Incidentes Cloud & Ships Interoperatibility). La función principal de este portal será la de permitir la consulta en tiempo real de las incidencias técnicas u operativas relacionadas con los buques que operen en el área portuaria del interesado.

Al tratarse de una iniciativa cuyos datos tienen su origen en entidades privadas, y no públicas, se hace necesario definir en el marco de cooperación las normas y las formas en que se suministrará esta información, y del mismo modo, en qué forma estará disponible para su utilización y reutilización, garantizando siempre la seguridad en el tratamiento de la información.

Para este proyecto contaremos con la colaboración de la empresa Híades, la cual nos permitirá el acceso a su aplicación Amura Pilots para realizar una integración con Open Data MICSI, gracias a la cual se suministrará información real y actualizada al portal.

De este acuerdo de colaboración surge también la propuesta de una aplicación para dispositivos móviles, que permitirá a los prácticos consultar y registrar información en tiempo real, durante los servicios de practicaje.

En resumen, los objetivos de este proyecto serán:

- Definición y análisis de un sistema en la nube que permita la consulta de datos sobre las incidencias relativas a los buques y a la operativa marítima.
- Desarrollo del sistema mencionado, garantizando que se cumplan los principios de la filosofía Open Data.
- Integración de la solución para el practicaje Amura Pilots con el portal MICSI, de forma que permita el suministro de datos en tiempo real.
- Propuesta de diseño de una aplicación móvil que se integre con Amura Pilots, y que permita en versiones posteriores la integración para el suministro y consulta de información con el portal MICSI.
- Desarrollo de un prototipo inicial de la aplicación para móviles, total o parcialmente funcional.

Capítulo 2
Análisis y Requisitos

La primera fase del desarrollo de este proyecto requería de un análisis de los requisitos funcionales, para lo cual aprovechamos la colaboración con Híades para acceder a sus clientes, quienes nos ayudarán a definir la estructura de los datos que se utilizará para el portal.

El contexto de esta colaboración nos facilitará una serie de infraestructuras tales como entornos de prueba y de producción en los que poder testear el desarrollo. Aunque como condición, se limitarán las tecnologías a utilizar, como se comentará más adelante.

2.1 Análisis de la estructura de datos

Fruto de una serie de reuniones para la toma de requisitos, y del análisis de la estructura actual de la Base de Datos de Amura se establece lo siguiente:

2.1.1 Estructura de Incidencias de Buques

El objeto principal del desarrollo será el registro de incidencias de buques, la cual se determina que deberá cumplir los siguientes requisitos:

Tabla 2-1: Estructura de Incidencia de Buque

Campo	Tipo de dato	Descripción
IMO [2]	*String*	Código de identificación internacional para el Buque.
Código	*String*	Código de incidencia.
Descripción	*String*	
Corporación	*String*	Corporación que proporciona el dato.
Fecha de Alta	*DateTime*	
Fecha de Actualización	*DateTime*	
Resuelta	*Boolean*	¿Se ha resuelto la incidencia?

2.1.2 Estructura de Buques

Se solicita también la creación de una base de datos buques, que permita no sólo consultar las incidencias relacionadas, sino una ficha detallada del mismo. Esta tabla deberá contener los siguientes campos:

Tabla 2-2:Estructura de Ficha de Buque

Campo	Tipo de dato	Descripción
Nombre	*String*	
IMO	*String*	Código de identificación internacional.
MMSI	*String*	Identificación del Servicio Móvil Marítimo.
Distintivo de Llamada	*String*	Código de identificación alternativo al IMO.
Nacionalidad	*String*	Bandera con la que opera el buque.
Tipo de Buque	*String*	
Consignatario	*String*	
Naviera	*String*	
GT	*Integer*	Medida del volumen del buque.
Eslora	*Decimal*	Longitud de pro a popa.
Manga	*Decimal*	Anchura del barco.
Guinda	*Decimal*	Altura de la parte más alta del buque por encima de la línea de flotación.
Puntal	*Decimal*	Altura interior del casco.
Calado Verano	*Decimal*	Distancia vertical entre un punto de la línea de flotación y la línea base del buque.
Construcción	*String*	
Observaciones	*String*	
Fecha de Alta	*DateTime*	
Fecha de Actualización	*DateTime*	
Velocidad	*Decimal*	
Remolcadores Recomendados	*Integer*	Número de remolcadores recomendados en caso de necesitarlos.

Todos estos datos se extraen de un análisis de la estructura de la tabla de

buques de la Base de Datos de Amura.

2.2 Estructura de Metadatos

En España disponemos de una guía de aplicación de la NTI para la Reutilización de Recursos de Información. Esta guía es de obligatorio cumplimiento para la reutilización de Recursos de Administraciones Públicas.

Sin embargo, el ámbito de este proyecto pertenece al sector privado, por lo que no estaríamos obligados a su cumplimiento. No obstante aprovecharemos sus recomendaciones para tratar de definir una estructura para nuestro catálogo de datos.

El esquema propuesto por la NTI sería el siguiente:

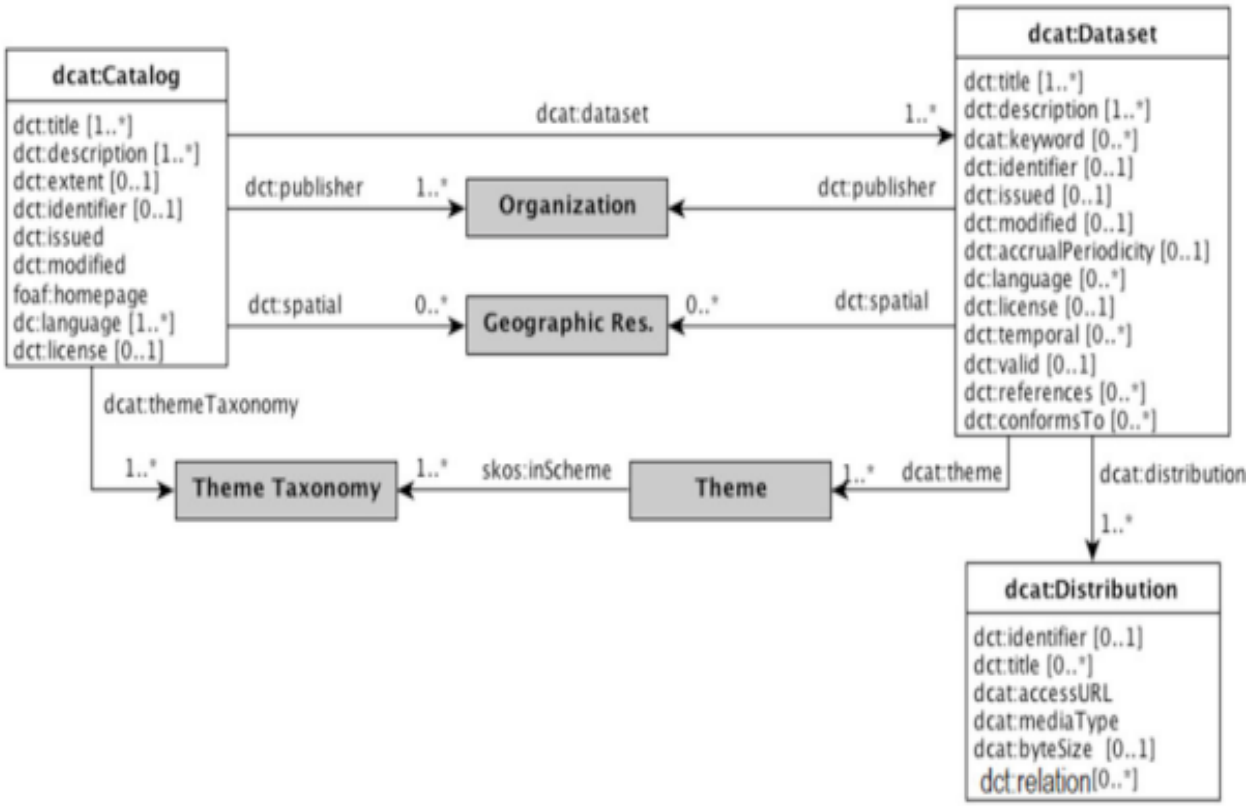


Figura 2-1: Esquema NTI Reutilización de Recursos de Información

Podemos ver como la información se estructura en catálogos, los cuáles se organizan por Organismo, Recurso Geográfico, Temática y Taxonomía de Temas. A su vez, un catálogo contendrá distintos datasets o conjuntos de datos, que serán accesibles a su vez mediante distintos formatos de distribución.

En nuestro caso sólo dispondremos de un catálogo de datos e inicialmente, ofreceremos dos datasets: Incidencias y Buques, aunque de

éste último se podrá consultar el listado completo o simplemente la ficha de un único buque. Así obtenemos la siguiente estructura para los metadatos de nuestro proyecto.

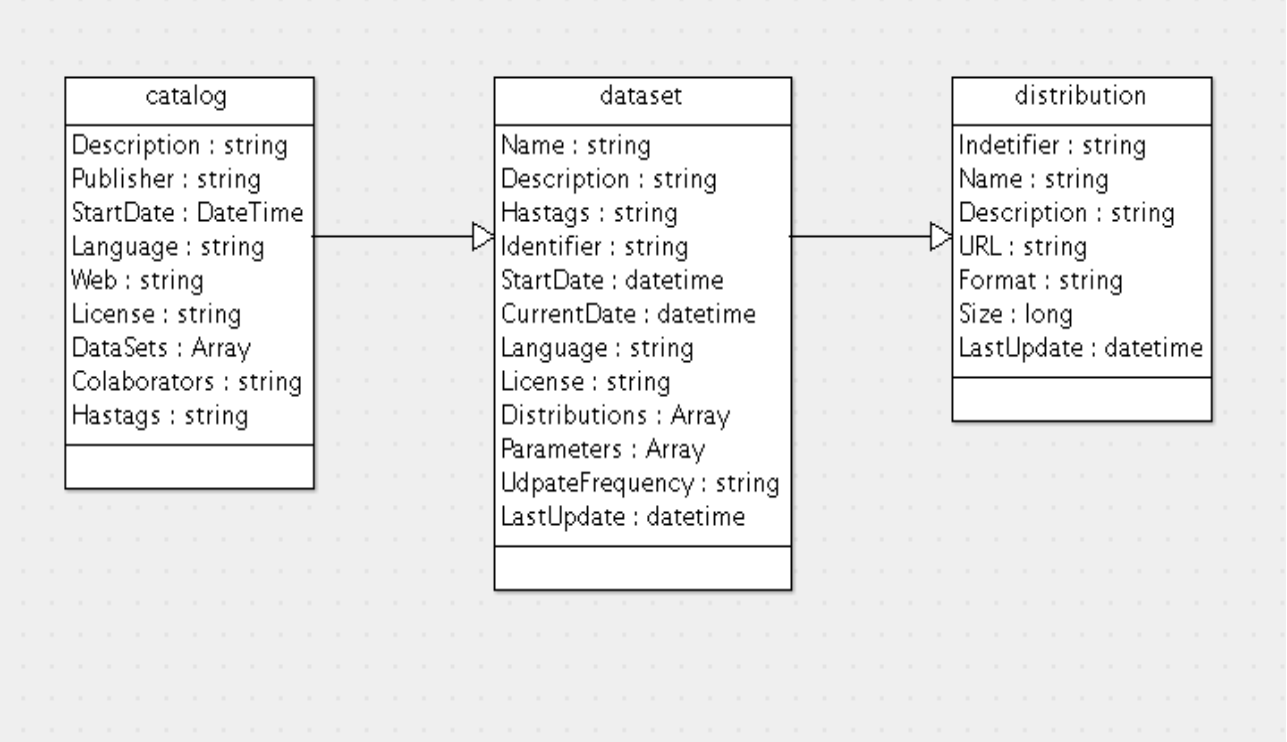


Figura 2-2: Estructura metadatos Open Data MICSI

2.3 Requisitos del Portal Open Data MICSI

Para el acceso a los datos a través de Open Data MICSI hemos determinado dos métodos: el uso de APIs para la utilización de los datos de forma automática, a través de formatos estándar como JSON o CSV, y, a petición de los clientes de Amura, un portal Web donde se les proporciona una interfaz gráfica en la que podrán consultar las incidencias. Aunque en este último caso la reutilización de la información no sea posible.

2.3.1 APIs

Una API(Application Programming Interface) es una interfaz de programación de aplicaciones, es decir, un conjunto de métodos, funciones o clases proporcionadas por un sistema operativo, librería o servicio, para ser utilizados por otro software como una capa de abstracción para la comunicación con el mismo.

En nuestro caso concreto, el desarrollo de las APIs permitirá a otras aplicaciones tanto la consulta de los datos, como, en un ámbito más limitado, la inserción de nuevos datos, sin necesidad de conocer la implementación del proyecto.

2.3.2 Requisitos de la API

Para el desarrollo de las APIs se han establecido los siguientes requisitos:

Tabla 2-3: Requisitos de la API

Requisito	Descripción
Tipo de API	Debe tratarse de una API Rest.
Formatos	Obligatorio: JSON. Opcional: CSV.
Parámetro Obligatorio	La identificación de los buques, tanto para la descarga de incidencias como para las fichas de buque, se realizará a través del IMO del buque, un número asignado por la Organización Marítima Internacional (OMI), y que permite identificar de manera inequívoca una embarcación, por buscar una referencia similar, podríamos considerarlo como la matrícula de un buque, pero a nivel internacional.
Parámetros opcionales	Los datos, en la medida de lo posible podrán obtenerse de forma paginada y ordenada según criterios especificados por parámetros del a petición.

2.3.3 Portal Web

Un portal web es un sitio en Internet, que ofrece al usuario el acceso a una serie de recursos y servicios sobre un tema concreto.

En este caso se ha requerido una interfaz en la que se muestren disponibles todas las incidencias registradas en MICSI, y en la que se puedan utilizar filtros para discriminar la información. Deberá poder usarse desde dispositivos móviles.

Tabla 2-4: Requisitos del portal Web

Requisito	Descripción
Interfaz	Sencilla y usable, evitar información y elementos innecesarios.

Responsive/Adaptativo	Esto es, que el diseño debe ser capaz de adaptarse a la visualización tanto en dispositivos de escritorio como en móviles.
Acceso Libre	La web debe ser accesible sin necesidad de realizar autenticación o registro.
Filtrado de Datos	Debe mostrarse, y por tanto poder filtrarse los datos por los siguientes campos: • Nombre de Buque. • IMO. • Corporación. • Fecha de alta.

2.4 Requisitos tecnológicos

El desarrollo de Amura se ha realizado utilizando el Framework de desarrollo de Microsoft .NET Framework [3] 4.0, el cual utiliza una estructura MVC adaptada para el desarrollo Web, y que utiliza el lenguaje de programación C# como lenguaje principal para el backend de la aplicación, utilizando HTML con sintaxis Razor de ASP.NET y JQuery como librería principal para el uso de JavaScript en el frontend. Para el acceso a datos de Base de Datos utiliza el ORM Entity Framework [4], también integrado dentro de .NET Framework. El control de versiones se realiza en un servidor Subversion propio de Híades.

Por tanto, siguiendo las recomendaciones, se ha optado por desarrollar tanto el portal web como las correspondientes APIs en .NET Framework, pero actualizando la versión del framework a la 4.6.1, y actualizando la versión de JQuery a la 1.11.

Las bases de datos, a su vez, serán implementadas en un Microsoft SQL Server 2014. El IDE que se utilizará para el desarrollo será Visual Studio 2015 Community Edition, y para el control de versiones se utilizará un entorno privado dentro de la plataforma GitHub.

2.5 Requisitos de Infraestructuras

Respecto a las infraestructuras, Híades ha puesto a nuestra disposición un servidor virtual con las siguientes características:

- Microsoft Windows Web Server 2008 R2.
- Internet Information Services 7.
- SQL Server 2014.

Este servidor se utiliza actualmente para entornos de preproducción de Amura, y cumple con los que se han establecido como requisitos mínimos de la aplicación que actualmente son:

- Microsoft Windows Server 2008.
- Internet Information Services 6.
- SQL Server 2010.

Se nos proporciona también un subdominio donde se podrá publicar la aplicación y acceder a ella: http://vessels.amurapilot.com/

Y un entorno de demo de Amura: http://demo.amurapilot.com/

2.6 Requisitos App Pilot Pro

El principal requisito para el desarrollo del prototipo de la app era que ésta debía ser compatible con las plataformas iOS y Android.

Además dada la naturaleza de la operativa del practicaje, la aplicación debía ser lo más ágil y fluida posible, y sus consultas al backend debían estar optimizadas al máximo, ya que la velocidad de conexión durante las maniobras suele ser excesivamente lenta, por la poca cobertura que suele haber en las zonas donde se opera.

De esta forma nos quedaba escoger entre dos opciones: desarrollo de aplicaciones nativas en cada plataforma, o utilización de un framework multiplataforma que permita generar código nativo.

La primera opción queda descartada desde el primer momento, ya que implicaría no sólo la necesidad de aprender las características de los lenguajes y entornos de desarrollo de ambos sistemas operativos, sino que, como mínimo duplicaría los tiempos de desarrollo, al tener que desarrollar una app para Android y otra para iOS.

Por tanto, la opción más viable consistía en encontrar y utilizar un framework multiplataforma que se adaptase a las necesidades del proyecto.

2.6.1 Xamarin

Este framework [5] surge de la mano de Mono, una implementación Open Source de .Net Framework que consiste en una serie de herramientas que permiten al desarrollador crear apps nativas para distintos sistemas operativos móviles, escribiendo el código en C#. Algunas ventajas de Xamarin son:

- Poder utilizar la potencia de C# en dispositivos móviles.
- Aplicaciones nativas con controles nativos de interfaz.
- Rendimiento nativo.
- Comparte alrededor del 90% del código entre plataformas.
- No utiliza Javascript, todo el código es compilado.

- Integración con Visual Studio.
- Un amplio catálogo de librerías en C#
- Ideal para desarrolladores de .Net

2.6.2 NativeScript

Otra potente alternativa para el desarrollo nativo multiplataforma es NativeScript [6]. Este framework desarrollado por Progress, es de código abierto, y permite desarrollar aplicaciones utilizando lenguajes de programación independientes del dispositivo y del sistema operativo.

En NativeScript se puede desarrollar tanto con JavaScript como con TypeScript, aunque también tiene soporte para el desarrollo con Angular y Vue. En cualquiera de los casos, utilizará las mismas APIs que las aplicaciones desarrolladas con Xcode o Android Studio, lo que facilita que cualquier funcionalidad que se agregue a las API nativas esté disponible de inmediato. Otras ventajas de NativeScript son:

- Soporte para librerías nativas de Android e iOS.
- Soporte para Angular 2.0.
- Interfaz de usuario y rendimiento nativos.
- Soporte para librerías de JavaScript.
- Soporte para TypeScript.
- Respaldado por Google y Telerik.
- Soporte para CSS.
- Open Source bajo licencia Apache 2.0.

2.6.3 App Pilot Pro tecnología escogida

Como hemos podido ver en los puntos anteriores, ambas tecnologías cuentan con numerosas ventajas. A favor de Xamarin destacar el uso de C# y su relación con .Net Framework, tecnología en la que se ha desarrollado la solución Amura.

Sin embargo, el poder incorporar una tecnología al alza como Angular, la flexibilidad que nos proporciona la utilización de CSS para el desarrollo de interfaces, y la inmediata disponibilidad de las novedades de cada plataforma a través de las APIs nativas, consigue que nos decantemos por la utilización de NativeScript para el desarrollo de la aplicación.

2.6.4 Requisitos funcionales

La aplicación Amura Pilot Pro pretende ser una extensión de la solución Amura Pilots para dispositivos móviles, por lo que deberá permitir con ciertas limitaciones, realizar la operativa del practicaje. Para ello se requieren las siguientes funciones:

Tabla 2-5: Requisitos funcionales App Pilot Pro

Funcionalidad	Descripción
Login	Los usuarios de la solución Amura Pilots deben poder acceder también en la aplicación Pilot Pro.
Consulta de previsiones	Debe poder acceder al listado de operaciones previstas, y realizar las operaciones correspondientes a la asignación de la maniobra.
Mis servicios	El práctico debe poder consultar desde el móvil, el listado de servicios que ha realizado, y realizar las operaciones correspondientes al embarque y desembarque.
Libro de Servicios	El usuario podrá consultar en todo momento el registro de servicios realizados por la corporación.
Buques	La información de las fichas técnicas de los buques debe estar accesible a través de un buscador, que permitirá al usuario consultar la ficha técnica de la nave seleccionada.
Gestión de estados operativos	Durante la maniobra de practicaje se generan una serie de estados de la operación que son registrados en la aplicación. Estos son por ejemplo el momento del embarque, el desembarque, el aviso a la autoridad portuaria, o las maniobras de fondeo. Estos estados deben poder registrarse desde la aplicación de forma sencilla por parte del usuario

Capítulo 3
Desarrollo

Concluido el proceso de análisis y definición de requisitos, y seleccionadas las tecnologías que se utilizarán para el desarrollo, pasaremos a la fase de Desarrollo.

En este punto abordaremos el Desarrollo del Portal Open Data MICSI, la integración con la solución Amura Pilots, la propuesta de diseño de la App Pilot Pro y el desarrollo de una versión inicial de la misma.

3.1 Desarrollo del Portal Open Data MICSI

Como se indicó en el capítulo anterior, el portal Open Data MICSI se desarrollará con .NET Framework 4.6.1, utilizando para ello el IDE Visual Studio 2015, y SQL Server Express para la gestión de bases de datos en el entorno de pruebas.

3.1.1 Estructura de clases de Modelo

El primer paso para el desarrollo de una API será definir el modelo de datos y las clases que se van a utilizar. Para nuestro caso sólo vamos a requerir el uso de dos modelos de datos: uno para las Incidencias de Buque y otro para los Buques.

Una de las características de .NET Framework, es que podemos definir los modelos de datos de nuestra Base de Datos utilizando clases de C#, las cuales serán trasladadas al esquema de Base de Datos gracias al ORM Entity Framework.

Esto se realiza siguiendo el principio Code-First [7], que consiste en definir primero las estructuras de datos en código, permitiendo que sea el framework quien gestione la creación y la estructura de la Base de Datos.

A continuación, podemos ver la clase que define el modelo de datos para las incidencias de buques.

```
public class IncidenciaBuque
{
    public IncidenciaBuque() { }

    public int IncidenciaBuqueId { get; set; }
    [Required]
    [DisplayName("Buque")]
    public int BuqueId { get; set; }

    public virtual Buque Buque { get; set; }

    [Required]
    [StringLength(10)]
    public string IMO { get; set; }            //"Matrícula" de la Organización Marítima Internacional

    [DisplayName("Código")]
    public string Codigo { get; set; }         //Código de la incidencia, unificar códigos

    [DisplayName("Descripción")]
    public string Descripcion { get; set; }

    public string Observaciones { get; set; }
    [DisplayName("Corporación")]
    public string Corporacion { get; set; }    //Corporación que realiza el envío. Definir código

    [Required]
    [DisplayName("Fecha alta")]
    public DateTime FechaAlta { get; set; }
    [DisplayName("Fecha Actualización")]
    public DateTime? FechaActualizacion { get; set; } //Posibles actualizaciones de la incidencia

    public bool Resuelta { get; set; }
}
```

Figura 3-1: Clase IncidenciaBuque

Como podemos observar, el framework manejará los elementos de una tabla (modelo) como instancias de una clase.

Destacar en este caso, la convención que utiliza Entity Framework para establecer las claves primarias de las tablas, y que consiste en agregar un campo entero que incluye el nombre de la clase con el sufijo *Id*.

Esto nos permitirá también establecer relaciones con otras entidades, como por ejemplo en el caso del buque. Al especificar el campo *BuqueId* se establece una relación de dependencia donde *BuqueId* es una clave externa relacionada con la clave primaria de la tabla Buque. Al añadir un elemento de la clase *Buque*, establecemos que cuando se instancie un objeto de *IncidenciaBuque*, se acompañe también de una instancia del objeto *Buque* relacionado.

A continuación, se presenta la definición de la clase *Buque*, de la cual se han ocultado algunos miembros, ya que estos han sido mencionados en la definición de requisitos, y en este momento no es relevante volver a reproducirlos.

```
public class Buque
{
    public Buque(){}

    public int BuqueId { get; set; }

    ////////////////////////////////////////////////////////////////////////////////////////////////////
    /// DATOS IDENTIFICATIVOS
    ////////////////////////////////////////////////////////////////////////////////////////////////////
    [Required]
    [StringLength(50)]
    public string Nombre { get; set; }

    [StringLength(10)]
    public string IMO { get; set; }             //"Matrícula" de la Organización Marítima Internacional

    [StringLength(10)]
    public string MMSI { get; set; }             //Identificación del Servicio Móvil Marítimo

    [StringLength(10)]
    [DisplayName("Distintivo de llamada")]
    public string DistintivoLLamada { get; set; }

    [DisplayName("Bandera")]
    public string Nacionalidad { get; set; }

    ////////////////////////////////////////////////////////////////////////////////////////////////////
    /// DATOS RELACIONADOS
    ////////////////////////////////////////////////////////////////////////////////////////////////////

    public string TipoBuque { get; set; }

    public string  Consignatario { get; set; }

    public string  Naviera { get; set; }

    ////////////////////////////////////////////////////////////////////////////////////////////////////
    /// CARACTERÍSTICAS TÉCNICAS
    ////////////////////////////////////////////////////////////////////////////////////////////////////
#region CaracteristicasTecnicas
#endregion
    ////////////////////////////////////////////////////////////////////////////////////////////////////
    /// OTROS
    ////////////////////////////////////////////////////////////////////////////////////////////////////
#region OtrosDatos
#endregion
    [ScaffoldColumn(false)]
    [DisplayName("Actualización")]
    public DateTime? FechaActualizacion { get; set; }

    public List<IncidenciaBuque> IncidenciaBuque { get; set; } //Relacionado: Incidencias del Buque
}
```

Figura 3-2: Clase Buque

En esta clase podemos observar también la introducción de lo que se denomina como *DataAnnotations*, esto son etiquetas que se colocan sobre los atributos para definir restricciones o características. Un ejemplo es la anotación *[StringLenth(n)]*, la cual determina tanto en la estructura de la tabla en base de datos, como en los distintos elementos de validación, que esa cadena tendrá como máximo una longitud *n*.

También podemos observar cómo se hace referencia a las incidencias relacionadas con el buque a través del atributo *IncidenciaBuque*, que contiene una lista de objetos de la clase *IncidenciaBuque*. Esto nos permitirá al momento de instanciar un objeto de la clase *Buque* desde la base de

datos, obtener un listado de las incidencias relacionadas, lo cual nos resultará tremendamente útil a la hora de consultarlas para el desarrollo de la API.

3.1.2 Estructura de la Base de Datos

Las clases anteriores, a través de la herramienta Entity Framework generan el esquema en nuestra base de datos SQL Server, para ello, en la configuración del proyecto debemos especificar la cadena de conexión con el servidor, y el nombre de la base de datos. Podemos ver un ejemplo de la cadena de conexión del entorno local:

```
<connectionStrings>
  <add name="AmuraVesselsContext" connectionString="data source=EHI002\SQLEXPRESS;
  initial catalog=AmuraVessels;User Id=sa;Password=*****; MultipleActiveResultSets=True;
  App=EntityFramework" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Figura 3-3: Cadena de conexión del entorno de desarrollo

Además de definir las clases, y la cadena de conexión, debemos definir una clase que se conoce como Contexto de Base de Datos, en la que se establecerán qué clases pertenecen al modelo de datos, además de otros métodos para determinar la configuración de la base de datos, convenciones y las relaciones entre entidades.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("AmuraVesselsContext", throwIfV1Schema: false)
    {
    }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {…
    }
    public static ApplicationDbContext Create()
    {…
    }
    public DbSet<Buque> Buque { get; set; }
    public DbSet<IncidenciaBuque> IncidenciaBuque { get; set; }
}
```

Figura 3-4: Clase de Contexto de la Aplicación

Podemos ver que en el constructor se especifica una cadena que contiene el mismo nombre que hemos utilizado para definir la cadena de conexión. De esta forma se relaciona la conexión de la base de datos con las instancias de esta clase.

Además, nos encontramos con las clases que habíamos definido, acompañadas de la sentencia *DbSet*. Aquí es donde indicamos que la clase *<NombreClase>* deberá aparecer en la base de datos con el nombre indicado a continuación.

Hecho esto, cuando arranquemos el proyecto por primera vez, si no existiera, se creará una base de datos con la siguiente estructura:

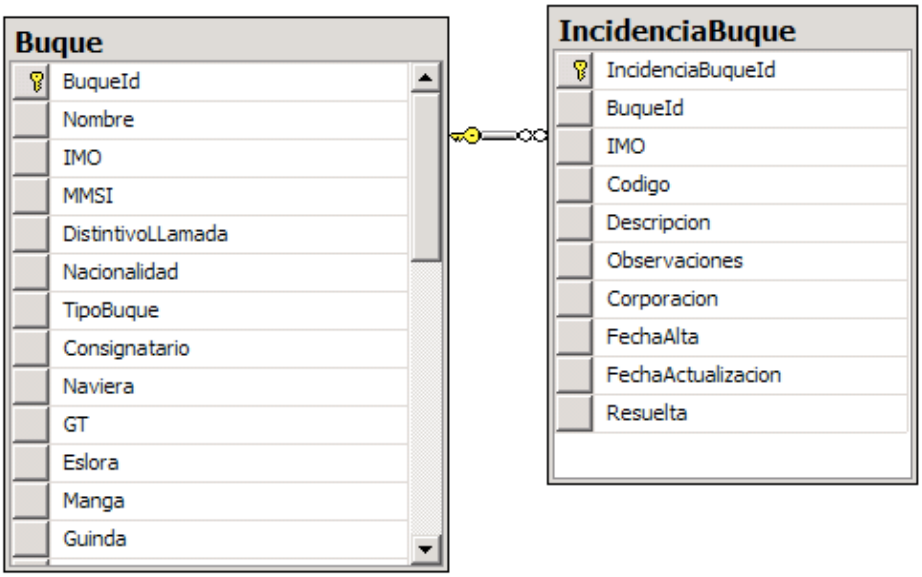


Figura 3-5: Esquema BBDD Buque-IncidenciaBuque

3.1.3 Controlador para APIs

Como habíamos comentado, .NET Framework utiliza un diseño basado en MVC (Modelo, Vista, Controlador). Esto implica que, para cada Vista, la cual define la interfaz de usuario, se asociará un modelo de datos, que es la representación que hayamos escogido para la información, y a su vez habrá un controlador que contenga un método relacionado con la vista y el modelo, que será quien gestione las peticiones y eventos de la vista al modelo.

En nuestro caso, sin embargo, utilizaremos lo que en .NET Framework se conoce como un *APIController*, que es básicamente un controlador cuyos métodos están orientados a la respuesta de peticiones web, concretamente implementaremos peticiones *GET* para el suministro de la información.

La definición del enrutamiento definido por el framework permitirá el acceso a los métodos siguiendo la siguiente estructura en las peticiones.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Configuración y servicios de API web

        // Rutas de API web
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{action}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Figura 3-6: Configuración del enrutamiento de APIs

Por ejemplo, si quisiéramos acceder a un método *ObtenerBuque* dentro de un *APIController* denominado *Buque*, y obtener el buque de *Id* 30, la petición tendría la siguiente forma:

http://nombredeservidor.es/api/Buque/ObtenerBuque/30.

Obviamente, se pueden utilizar más parámetros, y en este caso sólo se aceptarán peticiones *GET*, pero más adelante veremos cómo realizar otras peticiones a los servicios web implementados.

Para el desarrollo del proyecto hemos utilizado un único controlador, que será el encargado de implementar los distintos métodos de las APIs. Como estamos tratando de aplicar los principios de Open Data, los métodos públicos para la obtención de datos deberán estar disponibles sin ningún control de acceso ni autenticación. Para ello utilizaremos la anotación *AllowAnonymous* [8], la cual permite acceder a sus métodos públicos de forma anónima.

```
//[BasicAuthentication]
[AllowAnonymous]
public class BuquesApiController : ApiController
{
    private ApplicationDbContext db = new ApplicationDbContext();


    #region Metadata
```

Figura 3-7: Definición BuquesApiController

3.1.4 Métodos para la obtención de Metadatos

Anteriormente, habíamos comentado la estructura del catálogo de metadatos que se iban a implementar. Llegados a este punto hemos conseguido implementar una serie de métodos públicos, que permitirán

acceder a la información del catálogo y los correspondientes datasets.

En el Controlador no desarrollaremos la implementación de los métodos que construyen los JSON de metadatos, sino que estos se codificarán en una clase independiente, cuyos métodos serán accedidos desde el controlador para ofrecer la respuesta a las peticiones.

```
public class OpenDataUtils
{
    private static string GetServerName()
    {…
    }
    public static object CatalogData()
    {…
    }
    public static object ShipsDataSet() //TODO: Hacer una para buques, otra para buque, otr
    {…
    }
    public static object ShipsJsonDistribution(int length, DateTime? lastUpdate)
    {…
    }
    public static object ShipDataSet()
    {…
    }
    public static object ShipIncidencesDataSets()
    {…
    }
    public static object ShipIncidencesJsonDistribution(int length, DateTime? lastUpdate)
    {…
    }
```

Figura 3-8: Clase OpenDataUtils

Desde el controlador por tanto tendremos disponibles los siguientes métodos:

- **api/BuquesApi/GetCatalogInfo**: Proporciona la información relativa al catálogo


```
{
    "CatalogName": "MICSI Open Data",
    "Description": "",
    "Publisher": "Amura Pilots",
    "StartDate": "01/06/2018",
    "Language": "ES-es",
    "Web": "http://amurapilot.com/",
    "License": "Undefined",
    "DataSet": {
        "Vessels": "vessels.amurapilot.com/Api/BuquesApi/GetVesselsDataSetInfo",
        "VesselCard": "vessels.amurapilot.com/Api/BuquesApi/GetVesselCardDataSetInfo",
        "VesselIncidences": "vessels.amurapilot.com/Api/BuquesApi/GetVesselIncidencesDataSetInfo"
    },
    "Colaborators": "",
    "Hashtags": ""
}
```


Figura 3-9: Respuesta de metadatos de información del catálogo

- **api/BuquesApi/GetVesselsDataSetInfo:** Describe la información relativa al Dataset que permite descargar un listado de todos los buques de la base de datos. Especificando las URLs de las diferentes distribuciones disponibles, así como la configuración de parámetros admitidos.


```
{
    "Name": "Vessel List",
    "Description": "",
    "Hashtags": "",
    "Identifier": "Vessels_DataSet",
    "StartDate": "01/06/2018",
    "CurrentDate": "25/06/2018",
    "Language": "ES-es",
    "License": "Undefined",
    "Distributions": {
        "JSON": "vessels.amurapilot.com/Api/BuquesApi/GetBuques",
        "CSV": "Not Implemented"
    },
    "Parameters": {
        "pageSize": "int, optional",
        "pageNumber": "int, optional",
        "orderBy": "string, optional, not implemented"
    },
    "UpdateFrequency": "Real time",
    "LastUpdate": "01/06/2018"
}
```


Figura 3-10: Respuesta de metadatos del Data Set de Buques

- **api/BuquesApi/GetVesselCardDataSetInfo:** Similar al método anterior, pero en este caso, la información que obtendremos es la ficha de un buque concreto, identificado por su número IMO.


```
{
    "Name": "Vessel Info",
    "Description": "",
    "Hashtags": "",
    "Identifier": "VesselInfo_DataSet",
    "StartDate": "01/06/2018",
    "CurrentDate": "25/06/2018",
    "Language": "ES-es",
    "License": "Undefined",
    "Distributions": {
        "JSON": "vessels.amurapilot.com/Api/BuquesApi/GetBuque",
        "CSV": "Not Implemented"
    },
    "Parameters": {
        "IMO": "string, required"
    },
    "UpdateFrequency": "Real time"
}
```


Figura 3-11: Respuesta de metadatos del Data Set de Ficha de Buque

- **api/BuquesApi/GetVesselIncidencesDataSetInfo:** Estos son los

metadatos referentes a la API principal de este proyecto, que es la que permitirá acceder al histórico de incidencias de un buque, identificado por su número IMO.


```
{
    "Name": "Vessel Incidences",
    "Description": "",
    "Hashtags": "",
    "Identifier": "VesselIncidences_DataSet",
    "StartDate": "01/06/2018",
    "CurrentDate": "25/06/2018",
    "Language": "ES-es",
    "License": "Undefined",
    "Distributions": {
        "JSON": "vessels.amurapilot.com/Api/BuquesApi/GetIncidenciaBuque",
        "CSV": "Not Implemented",
        "Web": "vessels.amurapilot.com/IncidenciaBuque"
    },
    "Parameters": {
        "JSON": {
            "IMO": "string, required",
            "onlyMeta": "bool, optional",
            "pageSize": "int, optional",
            "pageNumber": "int, optional",
            "orderBy": "string, optional, not implemented"
        },
        "Web": {
            "IMO": "string, optional"
        }
    },
    "UpdateFrequency": "Real time",
    "LastUpdate": "25/06/2018"
}
```


Figura 3-12: Respuesta de metadatos del Data Set de Incidencias de Buque

3.1.5 API para Incidencias de Buque

Una vez implementadas las APIs que proporcionaban metadatos, el siguiente paso era desarrollar aquellas que suministren los datos. La primera de todas, la que nos permitirá descargar en formato JSON las incidencias relacionadas con un buque.

- **api/BuquesApi/GetIncidenciaBuque:** Realiza la búsqueda del buque por su número IMO, y devuelve un listado con las incidencias registradas, o en caso de no encontrar ninguna el mensaje “No data”. Junto con los datos de incidencias, se envían los metadatos referentes a la distribución.

  Los parámetros admitidos son los siguientes:

Tabla 3-1: Parámetros de la Api para Incidencias de Buque

Parámetro	Tipo	Descripción
IMO	*String*	Código de identificación internacional del buque. Obligatorio.
onlyMeta	*Boolean*	Determina si se envían los datos o sólo los metadatos. Opcional. Por defecto falso.
pageSize	*Int*	Número de incidencias por página. Opcional.

		Por defecto 0, listado completo.
pageNumber	*Int*	Cuando se solicita la información paginada, el número de página que se quiera descargar. Opcional. Por defecto 0, primera página.
orderBy	*String*	No implementado. La intención es establecer un sistema de ordenación basado en NombreCampo+DESC/ASC. Opcional. Por defecto null.

El código de la API:

```
// GET: api/BuquesApi/GetIncidenciasBuque
[ResponseType(typeof(IncidenciaBuque))]
public IHttpActionResult GetIncidenciaBuque(string IMO, bool onlyMeta = false, int pageSize = 0,
    int pageNumber = 0, string orderBy = null)
{
    Buque buque = db.Buque.Include("IncidenciaBuque")
        .FirstOrDefault(p => p.IMO == IMO);
    if (buque == null)
    {
        return NotFound();
    }

    if (!buque.IncidenciaBuque?.Any() ?? true)
    {
        return Ok(new { Data = "No data"});
    }

    buque.IncidenciaBuque.ForEach(p => p.Buque = null);

    var data = GetDataIncidencias(buque, pageSize, pageNumber);
    var meta = OpenDataUtils.ShipIncidencesJsonDistribution(new JavaScriptSerializer().Serialize(data).Length,
    buque.IncidenciaBuque.Max(p => p.FechaActualizacion));
    if (onlyMeta)
    {
        return Ok(new {MetaData = meta});
    }

    return Ok(new {MetadaData = meta, data = data });
}
```

Figura 3-13: Código de la API GetIncidenciaBuque

El método *GetDataIncidencias* nos devuelve un listado de las incidencias asociadas al buque, paginadas en la forma que se indique en los parámetros, mientras que el método *ShipIncidencesJsonDistribution* de la clase *OpenDataUtils* nos generará los metadatos asociados a la distribución.

Podemos ver a continuación un ejemplo de respuesta de la API, a la siguiente petición:

http://vessels.amurapilot.com/Api/BuquesApi/GetIncidenciaBuque?IMO=9463011&pageSize=1&pageNumber=0

```
{
    "MetadaData": {
        "Indetifier": "Vessels_Incidences_JSON",
        "Name": "Vessels Incidences JSON",
        "Description": "",
        "URL": "vessels.amurapilot.com/Api/BuquesApi/GetIncidenciaBuque",
        "Format": "JSON",
        "Size": 348,
        "LastUpdate": "18/06/2018 12:28:27"
    },
    "data": {
        "incidences": [
            {
                "Buque": null,
                "IncidenciaBuqueId": 1,
                "BuqueId": 579428,
                "IMO": "9463011",
                "Codigo": null,
                "Descripcion": null,
                "Observaciones": "ERROR!",
                "Corporacion": "PDV - PRÁCTICOS DE VALENCIA S.L.P.",
                "FechaAlta": "2018-06-18T12:27:00",
                "FechaActualizacion": "2018-06-18T12:28:01.007",
                "Resuelta": false
            }
        ],
        "pageSize": 1,
        "pageNumber": 0,
        "pages": 2,
        "nextPage": 1
    }
}
```


Figura 3-14: Respuesta de la API GetIncidenciaBuque

3.1.6 API para el listado de Buques

El desarrollo de esta API será idéntico al de la API para incidencias, pero en este caso se devolverá un listado de buques en formato JSON.

- **api/BuquesApi/GetBuques:** Los parámetros de esta API serán los mismos que en la anterior, exceptuando el campo IMO, ya que la intención es obtener un listado de los buques disponibles. Con esta API habrá que tener cuidado a la hora de realizar una petición, ya que actualmente hay registrados unos 83000 buques en la Base de Datos, con lo que una petición que incluya todos los datos podría provocar una excesiva lentitud en el tiempo de respuesta.

Como los parámetros son similares a los de la API anterior, veremos directamente la implementación de la API, en la cual encontraremos también un método para la paginación de los datos, *GetDataBuques*, y otro método de *OpenDataUtils* que devuelve los metadatos de esta distribución. A continuación se mostrará también un ejemplo de respuesta a la petición.

```
// GET: api/GetBuques/
//TODO: Optional order by
[ResponseType(typeof(List<Buque>))]
public IHttpActionResult GetBuques(int pageSize = 0, int pageNumber = 0, string orderBy = null
, bool onlyMeta = false)
{
    List<Buque> buques = db.Buque.Where(p => p.IMO != null).ToList();
    if (!buques.Any())
    {
        return NotFound();
    }

    var length = pageSize != 0 ? buques.GetRange(pageNumber*pageSize,
    pageSize).ToArray().Length : buques.ToArray().Length;
    var data = GetDataBuques(buques, pageSize, pageNumber);
    var meta = OpenDataUtils.ShipsJsonDistribution(length, buques.Max(p => p.FechaActualizacion));
    if (onlyMeta)
    {
        return Ok(new { MetaData = meta });
    }

    return Ok(new { MetadaData = meta, data = data });
}
```

Figura 3-15: Código de la API GetBuques

http://vessels.amurapilot.com/Api/BuquesApi/GetBuques?pageSize=10&pageNumber=0


```
{
    "MetadaData": {
        "Indetifier": "Vessels_JSON",
        "Name": "Vessels JSON",
        "Description": "",
        "URL": "vessels.amurapilot.com/Api/BuquesApi/GetBuques",
        "Format": "JSON",
        "Length": 10,
        "LastUpdate": "N/D"
    },
    "data": {
        "ships": [⟺],
        "pageSize": 10,
        "pageNumber": 0,
        "pages": 8329,
        "nextPage": 1
    }
}
```


Figura 3-16: Respuesta de la Api GetBuques

3.1.7 API para la ficha de Buque

La última API que se desarrollará para este bloque será la que nos permita obtener la ficha técnica de un buque, identificándolo por el código IMO, siendo este el único parámetro que se admitirá en las peticiones.

El método es el más sencillo, ya que no requiere ningún tipo de paginación ni la carga de listados. Para mantener su compatibilidad con Amura, no se ha implementado la carga de metadatos en este caso.

```
// GET: api/GetBuque/5
[ResponseType(typeof(Buque))]
public IHttpActionResult GetBuque(string IMO)
{
    Buque buque = db.Buque
        .FirstOrDefault(p => p.IMO == IMO);
    if (buque == null)
    {
        return NotFound();
    }
    return Ok(buque);
}
```

Figura 3-17: Código de la API GetBuque

3.1.8 Desarrollo del portal Web

El siguiente paso en el desarrollo del proyecto consistía en el desarrollo de un portal a través del cual tuvieran una interfaz con la que consultar las incidencias de los buques. En este caso sí aprovecharemos la estructura MVC, para implementar una vista en HTML, y las operaciones que permitan la carga y visualización de los datos. Comencemos con el Modelo.

Modelo. IncidenciaBuqueGridViewModel:

En este caso crearemos lo que en .Net Framework conocemos como *ViewModel*, que es una clase que implementa una adaptación de una clase de Modelo, para ajustarla a los requisitos de una vista o vistas concretas.

```
public class IncidenciaBuqueGridViewModel
{
    public int IncidenciaBuqueId { get; set; }
    [Required]
    [DisplayName("Buque")]
    public int BuqueId { get; set; }

    [Required]
    [DisplayName("Buque")]
    public string BuqueNombre { get; set; }

    [Required]
    [StringLength(10)]
    public string IMO { get; set; }            //"Matrícula" de la OMI

    public string Observaciones { get; set; }
    [DisplayName("Corporación")]
    public string Corporacion { get; set; }    //Corporación que realiza el envío.

    [Required]
    [DisplayName("Fecha alta")]
    public DateTime FechaAlta { get; set; }
    [DisplayName("Fecha Actualización")]
    public DateTime? FechaActualizacion { get; set; } //Posibles actualizaciones

    public bool Resuelta { get; set; }
}
```

Figura 3-18: Clase IncidenciaBuqueGridViewModel

Controlador. IncidenciaBuqueController:

El controlador dispondrá de una vista, que sólo acepta peticiones *GET*, por lo que no será necesario implementar ningún método adicional. Su única

función será la de establecer un enlace entre la Vista y el modelo de datos. Además se encargará de convertir los objetos del Modelo *IncidenciaBuque* al ViewModel *InciencíaBuqueGridViewModel*.

Este método acepta el IMO como parámetro, para acotar las incidencias a aquellas que pertenezcan al buque especificado.

```
public class IncidenciaBuqueController : Controller
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: IncidenciaBuque
    public ActionResult Index(string IMO = null)
    {
        return View(db.IncidenciaBuque.Where(p => IMO == null || p.IMO == IMO).Select(
            s => new IncidenciaBuqueGridViewModel()
            {
                BuqueId = s.BuqueId,
                BuqueNombre = s.Buque != null ? s.Buque.Nombre : string.Empty,
                IMO =  s.IMO,
                Codigo = s.Codigo,
                Descripcion = s.Descripcion,
                Observaciones = s.Observaciones,
                Corporacion = s.Corporacion,
                FechaAlta =  s.FechaAlta,
                FechaActualizacion = s.FechaActualizacion,
                Resuelta = s.Resuelta

            }).ToList());
    }

    protected override void Dispose(bool disposing)
    {…
    }
}
```

Figura 3-19: Código del controlador IncidenciaBuque

La vista. IncidenciaBuque/Index:

En esta vista se generará una tabla, en la que se insertarán los datos recibidos desde el controlador, con las incidencias registradas. Además, se utilizará la librería para JQuery, JQuery BootGrid [9], con la que se añadirán automáticamente controles de filtrado, selección de columnas visibles, paginación y ordenación a la tabla.

Además, esta librería incluye hojas de estilos CSS que hacen que el diseño de la tabla se adapte al tamaño de la ventana, y puede ser usada desde el móvil.

El resultado lo podemos ver en la siguiente URL:

http://vessels.amurapilot.com/IncidenciaBuque

Figura 3-20: Portal de consulta de Incidencias

3.2 Integración con Amura Pilots

El siguiente hito de este proyecto consistía en realizar una integración del portal con la solución para el practicaje Amura Pilots. Este punto habría que dividirlo en dos partes, la creación de una API de escritura en el proyecto MICSI, y la integración de la estructura en el proyecto Amura.

3.2.1 API de Escritura en el proyecto MICSI

Hasta ahora, todas las APIs que hemos visto utilizaban métodos *GET* para recibir una serie de parámetros y devolver la información solicitada. En este caso, implementaremos un método *POST* que recibirá un objeto del tipo *IncidenciaBuque*, y lo volcará en la base de datos.

Una de las ventajas del *APIController* de .Net Framework, es que permite convertir un objeto recibido por *POST* a una clase del proyecto de forma automática, aunque esto implica que quien envíe la solicitud debe conocer la estructura de la misma, o una estructura compatible. En nuestro caso, en Amura implementaremos exactamente la misma estructura que hemos desarrollado para el portal MICSI, asegurándonos la compatibilidad.

Otro punto a destacar en esta API es que no nos interesa que sea una API abierta como en los casos anteriores, ya que aunque los datos sean abiertos para su utilización, no serán así para el volcado de información, por lo que restringiremos el acceso a esta operación sólo a desarrollos autorizados, a través de las cabeceras *BasicAuthentication*.[10]

```
[ResponseType(typeof(IncidenciaBuque))]
[HttpPost]
[BasicAuthentication]
public IHttpActionResult PostIncidenciaBuque(IncidenciaBuque incidencia)
{
    incidencia.BuqueId = db.Buque.FirstOrDefault(p => p.IMO == incidencia.IMO)?.BuqueId ?? default(int);
    if (!ModelState.IsValid || incidencia.BuqueId == default(int))
    {
        return BadRequest(ModelState);
    }
    db.InsertOrUpdate(incidencia, incidencia.IncidenciaBuqueId, db.IncidenciaBuque);
    db.SaveChanges();
    return CreatedAtRoute("DefaultApi", new { id = incidencia.IncidenciaBuqueId }, incidencia);
}
```

Figura 3-21: Código de la API para la escritura de Incidencias en el Sistema

En el código podemos observar la potencia de Entity Framework, y el lenguaje de consultas que utiliza (LINQ). Lo primero que hacemos al recibir una incidencia, es buscar por el IMO el buque con el cuál se va a relacionar.

En caso de que no se encontrara el buque registrado en el sistema, o la estructura de la incidencia no fuera la correcta, por ejemplo, que no cuente con todos los campos requeridos, se devolvería un estado *HTTP_Bad_Request*.

Las siguientes sentencias son las encargadas de escribir en la base de datos la información que acabamos de recibir, a través de las sentencias *InsertOrUpdate*, para insertar o actualizar una entidad, y *SaveChanges*, para confirmar los cambios en el contexto.

Con esto ya tendríamos disponible una API para la escritura de incidencias en la ruta: **/api/BuquesApi/PostIncidenciaBuque**

3.2.2 Integración en Amura Pilots

Para integrar las incidencias de buques en Amura Pilots, lo primer que debíamos hacer era adaptar la estructura de base de datos. Al ser una aplicación en producción, no podíamos simplemente volver a construir el esquema, por lo que se opta por la solución de generar un script SQL que inserte las tablas necesarias.

```
Create Table IncidenciaBuque
(
    IncidenciaBuqueId int primary key identity(1,1) not null,
    BuqueId int not null references Buque(BuqueId),
    IMO nvarchar(10) not null,
    Codigo nvarchar(10),
    Descripcion nvarchar(max),
    Observaciones nvarchar(max),
    Corporacion nvarchar(max),
    FechaAlta DateTime not null,
    FechaActualizacion DateTime null,
    Resuelta bit not null default 0
)
```

Figura 3-22: Script SQL para la tabla IncidenciaBuque

El siguiente paso, consistía en integrar la clase *IncidenciaBuque* en el Contexto del Modelo de Amura Pilots, para lo cual hemos replicado la misma clase de la Figura 3-1 dentro del código de Amura, y hemos añadido la entrada en el *ModelContext*, haciendo el *DbSet* del mismo modo que hacíamos en el otro proyecto.

Una vez la estructura de la base de datos y el contexto están preparados para operar con las incidencias de Buque, pasaríamos a generar los métodos para la conexión con Open Data MICSI, en una clase independiente para que pueda ser accedida desde cualquier controlador.

Este método recibirá la incidencia generada desde Amura, y realizará las operaciones necesarias para adaptar los datos y realizar el envío.

```
public void PostIncidenciaBuque(IncidenciaBuque incidencia, AmuraContext db)
{
    if (incidencia == null)
        return;
    var incidenciaVessels = new AmuraVessels_IncidenciaBuque();
    incidenciaVessels.IMO = db.Buque.FirstOrDefault(p => p.BuqueId == incidencia.BuqueId)?.IMO
     ?? string.Empty;
    incidenciaVessels.Corporacion = GetDatosCorporacion(db);
    incidenciaVessels.Observaciones = incidencia.Observaciones;
    incidenciaVessels.FechaAlta = Util.GetDateAndTime(incidencia.Fecha, incidencia.Hora)
     ?? DateTime.UtcNow;
    incidenciaVessels.FechaActualizacion = DateTime.UtcNow;
    incidenciaVessels.BuqueId = 0;
    incidenciaVessels.Resuelta = incidencia.Resuelta;
    var server = GetVesselsServer();
    string url = server + "Api/BuquesApi/PostIncidenciaBuque";
    string responseFromServer = CallApi(url, "POST",
     new JavaScriptSerializer().Serialize(incidenciaVessels));
}
```

Figura 3-23: Código del método que envía Incidencias a OpenData MICSI

Finalmente, sólo quedaría integrar estos métodos en la operativa de Amura, para lo cual se ha optado por emplear la ficha de buque, dentro del módulo de Buques de Amura.

Para esta tarea usaremos una herramienta que se encuentra integrada dentro del proyecto Amura, el *Telerik MVC Grid* [11]. Un componente JQuery que permite la creación de tablas, con métodos Ajax para la carga e inserción de datos, así como funciones de filtrado, paginación y ordenación.

Lo primero que debemos hacer es desarrollar los métodos del controlador que se accederán por peticiones Ajax desde el *Grid* en la Vista. En este caso generaremos dos métodos, uno para la carga de las incidencias relacionadas con el buque, y otro para la inserción de nuevas incidencias. Estos métodos utilizarán un *ViewModel*, como ya vimos cuando definíamos la vista para Open Data MICSI, en este caso, para evitar que posibles referencias circulares generen errores en la *serialización* de datos.

```
[GridAction]
public ActionResult selAjax_IncidenciasBuque(int id = 0)
{
    Session[GRID_STATE_HISTORICO_ICIDENCIAS] = this.GridRouteValues();
    return View(new GridModel(GetIncidenciasBuqueList(id)));
}

[GridAction]
public ActionResult insAjax_IncidenciasBuque(IncidenciaBuqueGridViewModel incidenciaBuque)
{
    Session[GRID_STATE_HISTORICO_ICIDENCIAS] = this.GridRouteValues();
    if (ModelState.IsValid)
    {
        db.IncidenciaBuque.Add(incidenciaBuque.ObtenerModeloBase());
        db.SaveChanges();
    }
    try
    {
        var vessels = new AmuraVessels();
        vessels.PostIncidenciaBuque(incidenciaBuque.ObtenerModeloBase(), db);
    }
    catch (Exception ex)
    {
        Log.Save(ex);
    }
    return View(new GridModel(GetIncidenciasBuqueList(incidenciaBuque.BuqueId)));
}
```

Figura 3-24: Métodos Ajax para el Grid de Incidencias de Buque

El primer método simplemente carga el listado de incidencias y lo convierte en un *ViewModel* para que pueda ser gestionado por el *Grid* de Telerik.

En el segundo vemos, como una vez recibido el objeto, se inserta en la base de datos, y se realiza su envío al portal Open Data MICSI, aprovechando el método de la clase que definimos con anterioridad.

Llegados a este punto sólo quedaría integrar el Grid en la vista del formulario de Buques, para lo cual aprovechamos el componente de Telerik, al cual tendremos que especificar:

- ***Model***: *IncidenciaBuqueGridViewModel*.
- ***DataKeys***: Claves principales del modelo, en este caso *IncidenciaBuqueId*
- ***Columns***: Columnas que queremos que se muestren en el *Grid*.
- ***DataBinding***: *dataBinding.Ajax()* especifica que la llamada a los métodos se realizará por Ajax. Aquí especificaremos también los métodos que implementamos en el controlador tanto para la selección de datos, como para la inserción de nuevas incidencias.
- ***ClientEvents***: Eventos predefinidos por el *Grid*, podemos especificar métodos de JavaScript que se disparan cuando se produzca ese evento.

- **Configuración del Grid**: *sortable*(), *pageable*(), *filterable*() son métodos de configuración que sirven para determinar si se permite el filtrado, ordenación, paginación, etc. dentro del *Grid*. También permite establecer configuraciones por defecto, como el orden predeterminado.

```
@(Html.Telerik().Grid<Amura.ViewModels.IncidenciaBuqueGridViewModel>()
.Name("GridIncidencias")
    .DataKeys(keys => keys.Add(c => c.IncidenciaBuqueId).RouteKey("IncidenciaBuqueId"))
    .ToolBar(commands => commands.Insert().ButtonType(GridButtonType.Image)
    .HtmlAttributes(new { @class = "CambiarSpan" })
        .ImageHtmlAttributes(new { @class = "t-icon t-new", title = "Nuevo registro" }))
.Columns(columns =>
{…
})
.Editable(editing => editing.DefaultDataItem(new Amura.ViewModels.IncidenciaBuqueGridViewModel()
{ Fecha = DateTime.Now, Hora = DateTime.Now, BuqueId = Model.BuqueId }).Mode(GridEditMode.InLine))
.EnableCustomBinding(true)
.DataBinding(dataBinding =>
{
    dataBinding.Ajax()…
})
.ClientEvents(events =>
{…
})
.Sortable(sort => sort.OrderBy(order => order.Add(o => o.Fecha).Descending()))
.Pageable(page => page.PageSize(5))
.Filterable().Scrollable().HtmlAttributes(new { @class = "with-chosen" }))
```

Figura 3-25: Código del Grid de Telerik para Incidencias de Buque

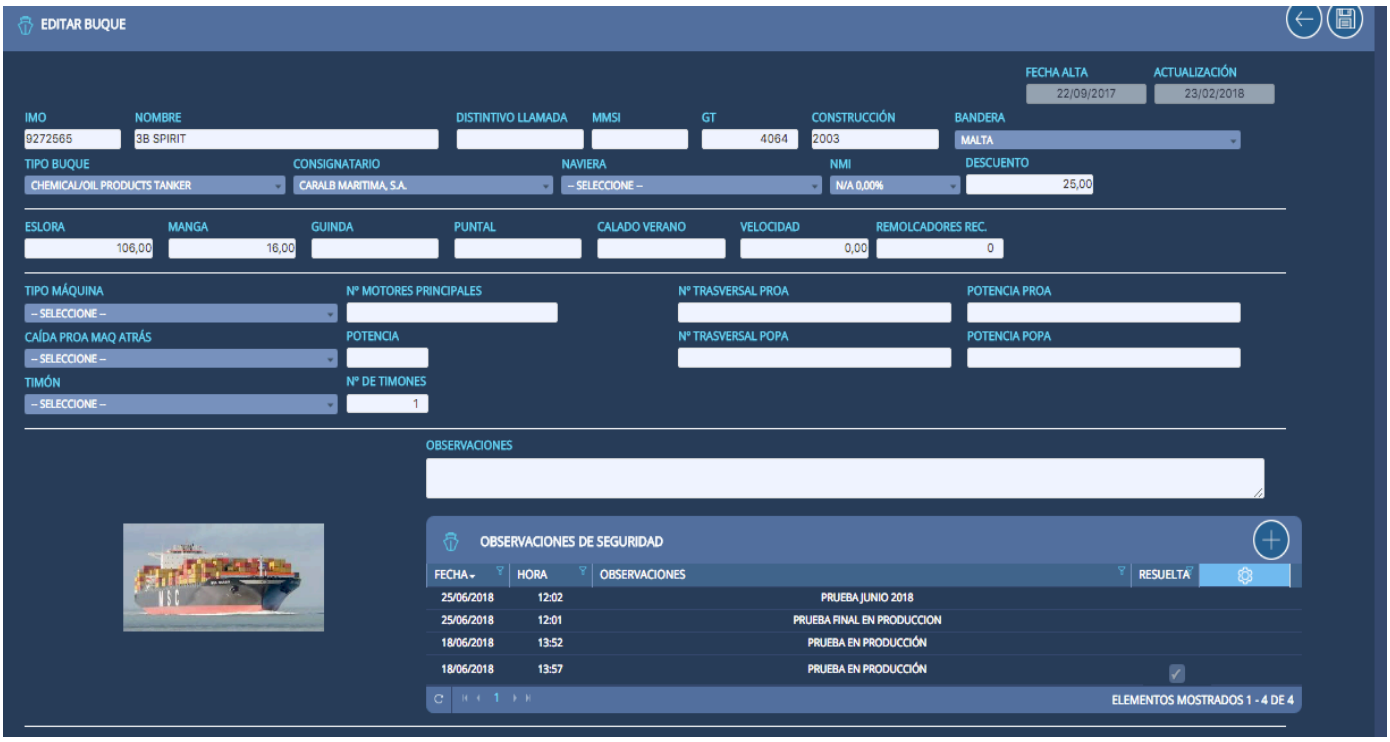


Figura 3-26: Ficha de Buque con integración de Incidencias

3.3 Diseño y desarrollo de la App Amura Pilot Pro

El último objetivo del proyecto es sin duda el más complejo y costoso del mismo, por eso desde un primer momento se plantea como un trabajo a concluir en el futuro, tratando de obtener durante esta etapa del desarrollo las bases para poder ofrecer una continuidad.

Es por eso por lo que esta tarea se centrará en la elaboración de un diseño, y un prototipo, parcialmente funcional de una aplicación multiplataforma para dispositivos móviles.

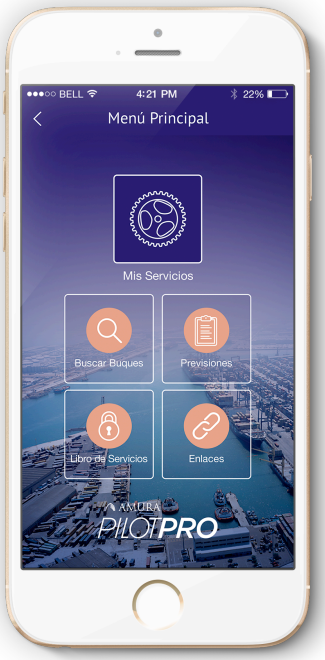
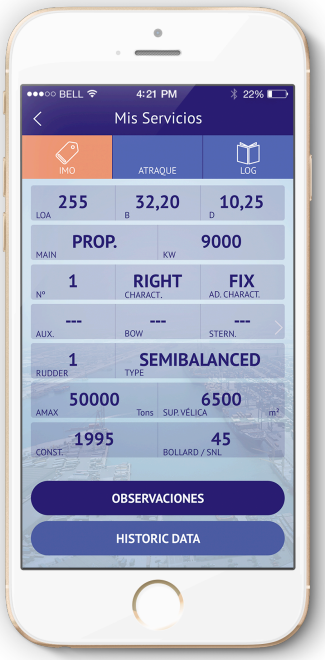
3.3.1 Fase 1: Diseño de la aplicación.

Para esta tarea contábamos con los siguientes elementos:

- Requisitos funcionales especificados por los clientes de la solución Amura Pilots, especificados en este mismo documento en el capítulo 2.6.4
- Una versión anterior de la aplicación, desarrollada con Corona SDK, y que ha sido descartada por la mayoría de las corporaciones, debido al bajo rendimiento que ofrece la tecnología que escogida.

Además, contábamos con la colaboración del departamento de diseño de Híades, que se encargó de generar una galería de maquetas, en base a las propuestas y bocetos que les fuimos facilitando. Con estas maquetas se definen las líneas de diseño a seguir en el desarrollo de la aplicación

Tabla 3-2: Mock-ups de Amura Pilot Pro

Login	Menú Principal	Mis servicios
		

3.3.2 Fase 2: Desarrollo de la aplicación

En el proceso de desarrollo de la aplicación nos hemos encontrado con algunas dificultades que han impedido dar por concluida esta fase, aunque es cierto que se ha cumplido el objetivo de disponer de un prototipo capaz de realizar ciertas fases de la operativa, y que se puede ejecutar en dispositivos tanto Android como iOS.

Las principales dificultades han sido:

- Desconocimiento del lenguaje y del framework: Angular a pesar de todas sus virtudes, se considera que tiene una de las curvas de aprendizaje más pronunciadas si se compara con otras alternativas como por ejemplo Vue.js
- Mala optimización de las APIs: Las APIs para la aplicación comenzaron su desarrollo en torno al año 2014-2015. A pesar de los profundos cambios que se han producido en la aplicación, estos métodos no han sido revisados adecuadamente, para adaptarse al incremento del volumen de datos y a las nuevas operativas del sistema.

A pesar de estas dificultades se han conseguido desarrollar los módulos propuestos, con mayor o menor éxito, por lo que a continuación pasaremos a detallar cada uno de ellos, mencionando aquellas partes que no se han podido completar, o que no funcionan de forma correcta, o en caso contrario, indicando que se encuentra plenamente funcional.

3.3.3 Módulos desarrollados de la app Pilot Pro

Tabla 3-3: Módulos desarrollados de Amura Pilot Pro

Módulo	Estado actual	Descripción e Incidencias
Login	Finalizado	No se produjeron incidencias. La comunicación de los componentes de NativeScript con los de .Net Framework no supuso ningún problema en este caso.
Mis Servicios	No funcional	Aunque el módulo permite acceder al listado de servicios y consultar datos del mismo, desde dispositivos iOS no permite registrar estados del servicio, y el panel de firma del Capitán no se encuentra operativo, debido a un error de la librería utilizada. Además, se produce un error en la animación en la brújula que indica la dirección del viento. En el proceso de embarque, en algunos dispositivos no es posible confirmar la hora, ya que los controles desaparecen, bloqueando la aplicación.

Buques	Parcialmente funcional	La búsqueda de buques funciona, pero la API aún tiene excepciones no controladas que producen errores en el lado de la app. Además algunos datos como el histórico de maniobras no se muestran aún.
Previsiones	Funcional	Este módulo es totalmente operativo, ya que permite la asignación de una previsión. Además en la última revisión se ha adaptado la aplicación al nuevo procedimiento que permite a los prácticos la asignación de varias maniobras, que realizarán de manera consecutiva, sin volver a pasar por la caseta de control.
Libro Servicios	No funcional	Permite la visualización del listado de servicios del puerto, pero actualmente no permite el acceso a los detalles del servicio.
Enlaces	Funcional	Este módulo es meramente informativo, y simplemente muestra un listado de enlaces de interés para los prácticos, como puede ser enlaces a la página de la AEMET o información sobre mareas.

Para finalizar este capítulo y antes de pasar a las conclusiones, una serie de capturas de la aplicación obtenidas desde un dispositivo iOS.

Figura 3-27: Capturas de pantalla de Pilot Pro en iOS

Capítulo 4
Conclusiones y líneas futuras

4.1 Conclusiones

Los objetivos propuestos inicialmente para el proyecto eran, la elaboración de un portal Web para la consulta de incidencias de buques, el desarrollo de una serie de APIs que permitieran la reutilización de esta información de acuerdo a los principios Open Data, la integración de esta herramienta con la solución web para la gestión del practicaje Amura Pilots, y por último el desarrollo de un prototipo de aplicación móvil multiplataforma, que permitiera a los usuarios de Amura disfrutar de sus posibilidades en un entorno de movilidad, permitiendo de esta forma el acceso a información de seguridad en tiempo real.

Se podría decir, a la conclusión de este proyecto que se han cumplido todos y cada uno de los objetivos, siendo quizás el último de estos el que ha dejado un sabor más agridulce, ya que las limitaciones y complicaciones surgidas no han permitido dotar a la aplicación de todas las funcionalidades que se pretendían. Sin embargo, se trataba de un prototipo inicial, y creemos sin duda que un futuro cumplirá con creces las expectativas depositadas en el proyecto.

En cuanto a los beneficios potenciales de este proyecto, seremos plenamente conscientes en el momento en que se puedan poner a disposición de los prácticos usuarios de Amura Pilots. Sin embargo, no tenemos dudas de que, haciendo un uso adecuado de esta solución se podrán disminuir los riesgos en operaciones, ya de por sí peligrosas como son todas aquellas relacionadas con el entorno marítimo, y que en ocasiones se ven empeoradas por la falta de transparencia con la información relativa a la seguridad.

Este proyecto permitirá no sólo la cooperación entre corporaciones de prácticos, sino que permitirá además a otras entidades como las autoridades portuarias, consignatarias o navieras, anticipar estas situaciones, y tomar las medidas correspondientes para solventar o incluso sancionar en las formas que se estimen oportunas.

Sin embargo, todo esto puede chocar con la realidad actual, y es el escaso

interés de entidades privadas en sumarse a iniciativas de datos abiertos, en parte por la falta de definición de unos estándares que definan las formas y garanticen la seguridad, y en parte también por la reticencia a ofrecer datos de forma abierta, para que puedan ser utilizados por otros, sin que esto revierta en un beneficio económico, al menos de forma directa.

En mi opinión, queda aún un largo camino por recorrer hasta que veamos una fuerte irrupción del sector privado en la disposición de sus datos de forma abierta, aunque sí que son cada vez más las empresas que se han dado cuenta de los beneficios que les proporciona la utilización de los datos que ponen a su disposición las administraciones públicas.

4.2 Líneas futuras

En cuanto a las líneas futuras del proyecto podemos establecer dos líneas, la referente al Open Data, y la relacionada con la integración con Amura Solutions.

4.2.1 Líneas futuras en Open Data

Esta línea debería orientarse en ofrecer nuevos formatos de distribución de la información, como podría ser csv o XML.

Además se deberían potenciar las APIs, para que permitan la obtención de consultas personalizadas, permitiendo aplicar filtros, establecer criterios de ordenación, o la selección de las columnas de datos que se quieren obtener.

Desde el portal Web se podría facilitar también la obtención de informes o gráficas, permitiendo generar de manera sencilla distintas estadísticas en base a los datos almacenados.

4.2.2 Líneas futuras en la integración con Amura Pilots

En este caso el objetivo principal debería ser el de potenciar no sólo el registro de información, sino facilitar el acceso a la información generada por otras corporaciones desde el propio entorno de Amura.

Otra forma de hacer crecer el proyecto sería desarrollar la integración con el resto de sistemas de la solución como pueden ser Amura Amarradores o Amura Remolcadores. Esto permitiría aumentar enormemente el tamaño de los orígenes de datos.

La reutilización de la información generada por las propias corporaciones, o el resto de corporaciones implicadas, debería ser aprovechada dentro de los Cuadros de Mandos de Amura, para desarrollar modelos que permitan mejorar la seguridad dentro del puerto.

Por último, sería interesante aprovechar la base que se ha desarrollado con el prototipo de Pilot Pro, para obtener una aplicación plenamente funcional.

Capítulo 5
Summary and Conclusions

The objectives initially proposed for the project were the development of a Web portal for the consultation of vessels incidents, the development of some APIs that would allow the reuse of this information, according to the Open Data principles, the integration of this tool with the web solution for the pilotage management Amura Pilots, and finally the development of a multiplatform mobile application prototype, which would allow Amura users to enjoy their possibilities in a mobility environment, allowing access to security information on real time.

It could be said, at the conclusion of this project, that each and every one of the objectives have been met, perhaps the last of which has left a more bittersweet taste, since the limitations and complications that have arisen have not allowed the application have all the functionalities that were intended. However, it was an initial prototype, and we believe without doubt that a future will more than meet the expectations placed on the project.

As for the potential benefits of this project, we will be fully aware at the moment that they can be made available to the users of Amura Pilots. However, we have no doubt that, by making proper use of this solution, risks in operations, already dangerous, such as those related to the maritime environment, may be reduced, and that these are sometimes worsened by the lack of Transparency with information related to security.

This project will allow not only cooperation between pilots corporations, but also allow other entities such as port authorities, consignees or shipping companies, anticipate these situations, and take appropriate measures to solve or even sanction in the ways deemed appropriate.

However, all this can clash with current reality, and is the lack of interest of private entities in joining open data initiatives, partly due to the lack of definition of standards that define the forms and guarantee security, and in part also by the reticence to offer data in an open manner, so that they can be used by others, without this reverting to an economic benefit, at least directly.

In my opinion, there is still a long way to go until we see a strong irruption of the private sector in the provision of their data in an open manner, although there are more and more companies that have realized the benefits provided

by the use of the open data provided by public administrations.

Capítulo 6
Presupuesto

Tabla 6-1: Presupuesto

	HORAS	PRECIO/HORA	IMPORTE TOTAL
Análisis de Requisitos	**10**	**40**	**400€**
ANÁLISIS - TOMA DE REQUISITOS DE CLIENTE	4		
ANÁLISIS - ANALISIS PROCESOS ACTUALES	6		
ENTREGABLE: DOCUMENTO ESPECIFICACIONES FUNCIONALES			
ENTREGABLE: CRONOGRAMA DESARROLLO TECNOLOGICO			
DESARROLLO TECNOLÓGICO	**310**	**40**	**12400€**
ANÁLISIS FUNCIONAL - ESPECIFICACIONES FUNCIONALES	20		
ANÁLISIS FUNCIONAL - ESTRUCTURA BASE DE DATOS	8		
DESARROLLO SOFTWARE – Portal Open Data	40		
DESARROLLO SOFTWARE – Integración con Amura Pilots	32		
DISEÑO – Diseño de Prototipo App Pilot Pro	24		
DESARROLLO SOFTWARE – Desarrollo de App Pilot Pro	180		
IMPLANTACIÓN – Implantación del entorno integrado de Amura	6		
ENTREGABLE: DOCUMENTO DE CREDECIALES DE ACCESO A USUARIOS FINALES			
ENTREGABLE: DOCUMENTO DE RIESGOS DE IMPLANTACIÓN / PRUEBAS DE RENDIMIENTO			
FORMACION	**12**	**30**	**360€**
FORMACIÓN TÉCNICA - USUARIOS DE LA SOLUCIÓN	12		
ENTREGABLE: MANUAL FUNCIONAL DE LA APLICACIÓN			
ENTREGABLE: INFORME FINAL DE PROYECTO			
	TOTAL		**13160€**

Capítulo 7
Bibliografía

[1] El Open Data ya genera un volumen de negocio de casi 1700 millones. [web] http://economiademallorca.com/not/8938/el-open-data-ya-genera-un-volumen-de-negocio-de-casi-1-700-millones/

[2] Cómo identificar un buque – Número IMO. [web] https://buquesyproductos.blogspot.com/2016/08/como-identificar-un-buque-numero-imo.html

[3] What is .NET?. [web] https://www.microsoft.com/net/learn/what-is-dotnet

[4] Información general sobre Entity Framework. [web] https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/ef/overview

[5] Xamarin Documentation. [web] https://docs.microsoft.com/en-us/xamarin/

[6] NativeScrpipt Documentation. [web] https://docs.nativescript.org/

[7] What is Code-First?. [web] http://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx

[8] MSDN Microsoft. AllowAnonymous. [web] https://msdn.microsoft.com/es-es/library/system.web.configuration.profilepropertysettings.allowanonymous%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396

[9] JQuery BootGrid Documentation. [web] http://www.jquery-bootgrid.com/Documentation

[10] Basic Authentication with Asp.Net WebAPI. [web] https://stevescodingblog.co.uk/basic-authentication-with-asp-net-webapi/

[11] Telerik MVC Grid. [web] https://demos.telerik.com/aspnet-mvc/grid

[12] Autoridad Portuaria de Algeciras. Practicaje [web] http://www.apba.es/practicaje

[13] Wikipedia. Práctico. [web] https://es.wikipedia.org/wiki/Práctico

[14] Prácticos de Puerto. [web] http://www.practicosdepuerto.es/

[15] El arqueo o cómo representar el tamaño de un buque. [web] http://sabemos.es/2015/07/23/el-arqueo-o-como-representar-el-tamano-de-un-buque_4549/

[16] ANESCO. ¿Qué es una empresa consignataria? [web] https://anesco.org/asociaciones-provinciales/que-es-una-empresa-consignataria/

[17] Open Data Handbook. [web] http://opendatahandbook.org/

[18] Los 8 principios básicos de los Datos Abiertos. [web]

https://www.viavansi.com/blog-xnoccio/es/8-principios-de-los-datos-abiertos/

[19] Carta Internacional de Datos Abiertos. [web] https://opendatacharter.net/principles-es/

[20] Open Data, o cómo los datos están cambiando el mundo. [web] https://www.paradigmadigital.com/techbiz/open-data-los-datos-estan-cambiando-mundo/

[21] Open Data Barometer. [web] https://opendatabarometer.org/

[22] Expansión. España, el país europeo más preparado para el 'Open Data'. [web] http://www.expansion.com/economia-digital/innovacion/2017/01/26/588785ede5fdea85468b460a.html

[23] Beneficios y retos de la apertura de los datos corporativos. [web] http://datos.gob.es/es/noticia/beneficios-y-retos-de-la-apertura-de-los-datos-corporativos

[24] Los datos abiertos en el sector privado: ¿asignatura pendiente?. [web] http://datos.gob.es/es/noticia/los-datos-abiertos-en-el-sector-privado-asignatura-pendiente

[25] Guía de aplicación de la Norma Técnica de Interoperabilidad de Reutilización de Recursos de Información. [web] http://datos.gob.es/es/documentacion/guia-de-aplicacion-de-la-norma-tecnica-de-interoperabilidad-de-reutilizacion-de

[26] Xamarin vs. Apache Cordova vs. NativeScript. [web] https://stackshare.io/stackups/apache-cordova-vs-nativescript-vs-xamarin

[27] .NET Framework Notes for Professionals [en línea]. Disponible en: https://goalkicker.com/DotNETFrameworkBook/

[28] Amura. Manual técnico de Amura Pilots. Amura: Manuales de usuario, 2018.