



Horizon 2020

Project acronym: **LASH FIRE**

Project full title: **Legislative Assessment for Safety Hazard of Fire and Innovations in Ro-ro ship Environment**

Grant Agreement No: **814975**

Coordinator: **RISE Research Institutes of Sweden**



Deliverable D08.3

Development of fire hazard mapping visualization tool with fire hazard matching integrated

January 2022

Dissemination level: **Public**

Abstract

This deliverable covers the requirements, the specification and the technologies used for the implementation and testing phases of a Fire Hazard Matching tool one of the main results of the LASHFIRE project related to automatic screening and management of cargo fire hazards.

Since the different types of cargo that can be transported or rolled onboard ro-ro cargo and ro-ro passenger ships is limitless, focus is on the cargo types that possess the most frequent issues, cargo that is classified as hazardous or possess new types of dangers to passengers, crew and ships, e.g. new types of alternative powers for vehicles.

The Fire Hazard Matching tool it is a software that will enable the visualization of risky 'hot' zones and different hazard types of cargo as a support element to identify hazards associated to each zone of the ship according to the cargo unit's position at planning and real level.

The Fire Hazard Matching tool is able to evaluate fire risk associated to all cargo units of a given ship loading configuration using an easy-to-use graphical interface that can be run both in computers and hand-held devices (mobile phones or tablets). It is developed as a standalone visualization and interaction module for the Stowage Planning Tool, the overall software tool result of action 8-A.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 814975

The information contained in this deliverable reflects only the view(s) of the author(s). The Agency (CINEA) is not responsible for any use that may be made of the information it contains.

The information contained in this report is subject to change without notice and should not be construed as a commitment by any members of the LASH FIRE consortium. In the event of any software or algorithms being described in this report, the LASH FIRE consortium assumes no responsibility for the use or inability to use any of its software or algorithms. The information is provided without any warranty of any kind and the LASH FIRE consortium expressly disclaims all implied warranties, including but not limited to the implied warranties of merchantability and fitness for a particular use.

© COPYRIGHT 2019 The LASH FIRE Consortium

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the LASH FIRE consortium. In addition, to such written permission to copy, acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced. All rights reserved.

Document data

Document Title:	D08.3– Development of fire hazard mapping visualization tool with fire hazard matching integrated		
Work Package:	WP08 – Ignition		
Related Task(s):	T08.5		
Dissemination level:	Public		
Deliverable type:	[R]		
Lead beneficiary:	16 – CIMNE		
Responsible author:	Ángel Priegue		
Co-authors:	África Marrero, Sergio Valero		
Date of delivery:	2022-01-31		
References:	D08.1, D08.2		
Approved by	Erik Styhr Petersen on 2022-01-19	Robert Rylander on 2021-08-11	Maria Hjohlman on 2022-01-31

Involved partners

No.	Short name	Full name of Partner	Name and contact info of persons involved
16	CIMNE	Centre Internacional de Mètodes Numèrics en Enginyeria	Ángel Priegue, cruchi@cimne.upc.edu África Marrero, amarrero@cimne.upc.edu Sergio Valero, svalero@cimne.upc.edu

Document history

Version	Date	Prepared by	Description
1	2020-07-15	CIM	Draft of structure
2	2021-07-30	CIM	Ready-to-review version
3	2021-10-29	CIM	Reviewed version
4	2021-12-23	CIM	Updated version for review
5	2022-01-31	CIM	Final version

Contents

1	Executive summary	5
1.1	Problem definition.....	5
1.2	Technical approach.....	5
1.3	Results and achievements.....	6
1.4	Contribution to LASH FIRE objectives.....	6
1.5	Exploitation and implementation.....	6
2	List of symbols and abbreviations	7
3	Introduction.....	9
4	Requirements	11
4.1	Definition of conditions for cargo fire hazard	13
4.2	Requirements on UX.....	17
4.3	Technologies for ignition prevention detectors.....	18
4.4	Use Cases and Functional Requirements	21
4.5	Non-functional requirements.....	29
4.6	Cyber Security requirements	29
5	Software Architecture	31
5.1	Data Layer.....	31
5.2	Application Layer	32
5.3	Presentation Layer	33
6	Implementation.....	35
7	Software Design.....	36
7.1	Design and Architecture	36
7.2	Data Design.....	38
7.3	Sequence Diagrams	41
8	Software Implementation	45
8.1	Web Client implementation	45
8.1.1	Web User Interface	45
8.2	Back-end Services implementation	48
8.3	APIs.....	49
9	Testing	52
9.1	Unit Tests.....	52
9.2	System Integration Tests.....	54
10	Conclusion	56

11	References	57
12	Indexes	58
12.1	Index of tables	58
12.2	Index of figures	58

1 Executive summary

This deliverable focus on all the steps of the development process (specification, design, implementation and testing) of the Fire Hazard Matching tool, the main result of T08.5 in the context of action 8-A of WP8.

According to the grant agreement the tool needs to include:

- Fire hazard matching by association of specific hazards to cargo units, according to input from T08.2 and T08.3,
- Development of integration software, fed by the database (T08.2) and with consideration to the identification technologies selected (T08.3).
- Development of hazard mapping tool with visualization of risky 'hot' zones and different hazard types of cargo.
- Support element to identify hazards associated to each zone of the ship according to the cargo unit's position (at planning and real level).
- Development of software to integrate feeds from identification of units on the deck and calculation of their associated hazard probabilities.

In brief, the Fire Hazard Matching tool, is a software that enables the visualization and identification of hazards associated to each zone of the ship according to the characteristics of the cargo units located in there.

The tool is able to evaluate fire risk associated to all cargo units of a given ship loading configuration using an easy-to-use graphical interface that can be run both in computers and hand-held devices (mobile phones or tablets).

1.1 Problem definition

Results from previous projects such as EMSA FIRESAFE indicated that it was clear that a lot of unwanted incidents on the cargo spaces of Ro-ro cargo/Ro-pax passenger ships are due to malfunction of so-called Reefer-units.

One of the goals of WP08 and the main goal of action 8-A is to develop and demonstrate a technical solution for automatic screening of cargo to identify fire hazards and develop, utilize and experimentally validate a digital logistics management tool featuring risk-based load planning. This solution should be able to identify and screen cargo for known hazards or anomalies and display the position of the cargo/vehicle on when is parked/secured on-board a ships deck.

As of today, no system as the one proposed here is on the market, neither has one been identified to be under development. There are different project and solutions around the world, that could be part of a solution, but there are no standards to lean on. They are stand-alone and not to the project's knowledge designed with interoperability in mind.

Usage of such technologies can enhance the current fire safety routines and systems around the handling of cargo/vehicles. The growing number of alternative powered vehicles (APV) will make the situation on-board the ship more complex than it is today in regard to the ship's fire safety. The mixture of hazards that will make it more complicated to extinguish a fire, since the means of controlling and extinguish the fire might need combinations of solutions and techniques.

1.2 Technical approach

This report describes the requirements, specification, codification and testing phases of the software development process of the Fire Hazard Matching Tool.

Action 8-A aims to create a system for automatic screening of cargo to identify fire hazards featuring risk-based load planning. The approach to fulfil this goal is described as a system of systems, where many subsystems have some operational overlaps where they can hand over information and together solve challenges in the operational context of loading/off-loading a ship.

1.3 Results and achievements

This deliverable includes the specification of the Fire Hazard Matching Tool for risk management of cargo. The matching and mapping tool associates specific hazards to cargo units according to data inputs from T08.2 (implemented through interfaces with the algorithm that distributes the cargo supporting fire risk management) and T08.3 (Assessment of cargo identification technologies).

The tool shows information useful to the users, e.g. position of cargo, type of cargo stated in the cargo manifest and possible hazards connected with the type of cargo etc. Information obtained from sensors can also be used to assist both the crew in the fire control room as well as the fire fighters on the deck via portable devices e.g. rugged smartphone/tablet.

In this deliverable we show all the steps of the development process (specification, design, implementation and testing) of the Fire Hazard Matching tool,. Some of the requirements are not implemented in the scope of the Fire Hazard Matching tool and will be part of the implementation of the later Visualization Aid in task T08.6 that will be presented in D08.4 Stowage planning optimization and visualization aid.

1.4 Contribution to LASH FIRE objectives

The Fire Hazard Matching tool is part of the system of systems approach with open interfaces to other systems, and in LASH FIRE the interoperability with the other WPs and a future data exchange with other systems.

In LASH FIRE WP08, the objective of the action A is to develop and demonstrate a technical solution for automatic screening of cargo to identify fire hazards and develop, utilize and experimentally validate a digital logistics management tool featuring risk-based load planning. This report covers the different software technologies that will be used for the implementation and testing of the fire hazard matching tool, that will be a useful and practical software tool to achieve this objective.
Exploitation and implementation

The fire hazard matching tool is a system that could be interconnected to other systems to meet the challenge that the stevedores and/or the crew on-board have every day when identifying cargo and its corresponding hazards, secure loading operation and stowage of such goods/vehicle, all in all to increase the safety of passengers, crew and the ship by enabling early detection of fire hazards or anomalies.

1.5 Exploitation and implementation

The Fire Hazard Matching tool, will be integrated as the basis of the Stowage Planning tool, that will add capabilities such as optimization and simulation that will be added in the Visualization Aid (task T08.6) into the software which is the final goal of action 8-A and one of the main results of the LASHFIRE project related to automatic screening and management of cargo fire hazards.

The final tool (Fire Hazard Matching tool + Visualization Aid) will be demonstrated in a controlled environment in the LASH FIRE Pilots that will take place during the last year of the project.

2 List of symbols and abbreviations

AFV	Alternative Fuel Vehicle
AJAX	Asynchronous JavaScript And XML
API	Application programming interface
APV	Alternative Powered Vehicle
ASP.NET	Active Server Pages .net
BLL	Business Logic Layer
CSS	Cascading Style Sheets
DBMS	DataBase Management System
EMSA	European Maritime Safety Administration
ER	Entity–relationship
ETA	Estimated Time of Arrival
GDPR	General Data Protection
GPS	Global Positioning
HMI	Human Machine Interface
HTML	HyperText Mark-up Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ICE	Internal combustion engines
I/O	Input/Output
ISO	International Standard Org
IMO	International Maritime Organization
JSON	JavaScript Object Notation
MVC	Model View Controller
ORM	Object-Relational Mapper
REST	Representational state transfer
RFID	Radio-frequency identification
Ro-ro	Roll On Roll Off cargo ship
Ro-pax	Roll On Roll Off passenger ship
SIT	System integration testing

SoS	System of systems
SQL	Standard Query Language
TDS	Tabular Data Stream
UX	User experience

3 Introduction

Main author of the chapter: Angel Priegue, CIM

Results from previous projects such as EMSA FIRESAFE (Ref LASH FIRE D4.01 Review of accident causes and hazard identification report) indicated that it was clear that a lot of unwanted incidents on the cargo spaces of Ro-ro cargo/Ro-pax passenger ships are due to malfunction of so-called Reefer-units.

There are other issues that could be detected if we integrate feeds from identification of units on the deck the cargo with cargo manifest and other administrative information and calculate associated hazard probabilities during sea voyage. This would enhance the effectiveness of a traditional firefighting patrol rounds.

The objective of task T08.5 is the integration & development of the software modules necessary for the implementation of the fire hazard matching and mapping tool. This deliverable describes the steps in the software development cycle achieved to implement the Fire Hazard Matching tool, a tool able to evaluate fire risk associated to all cargo units of a given ship loading configuration using an easy-to-use graphical interface that can be run both in computers and hand-held devices (mobile phones or tablets).

The tool is developed as a standalone visualization and interaction module for the Stowage Planning Tool, the overall software tool result of action 8-A. The following figure show the components that are part of the risk-based stowage planning tool:

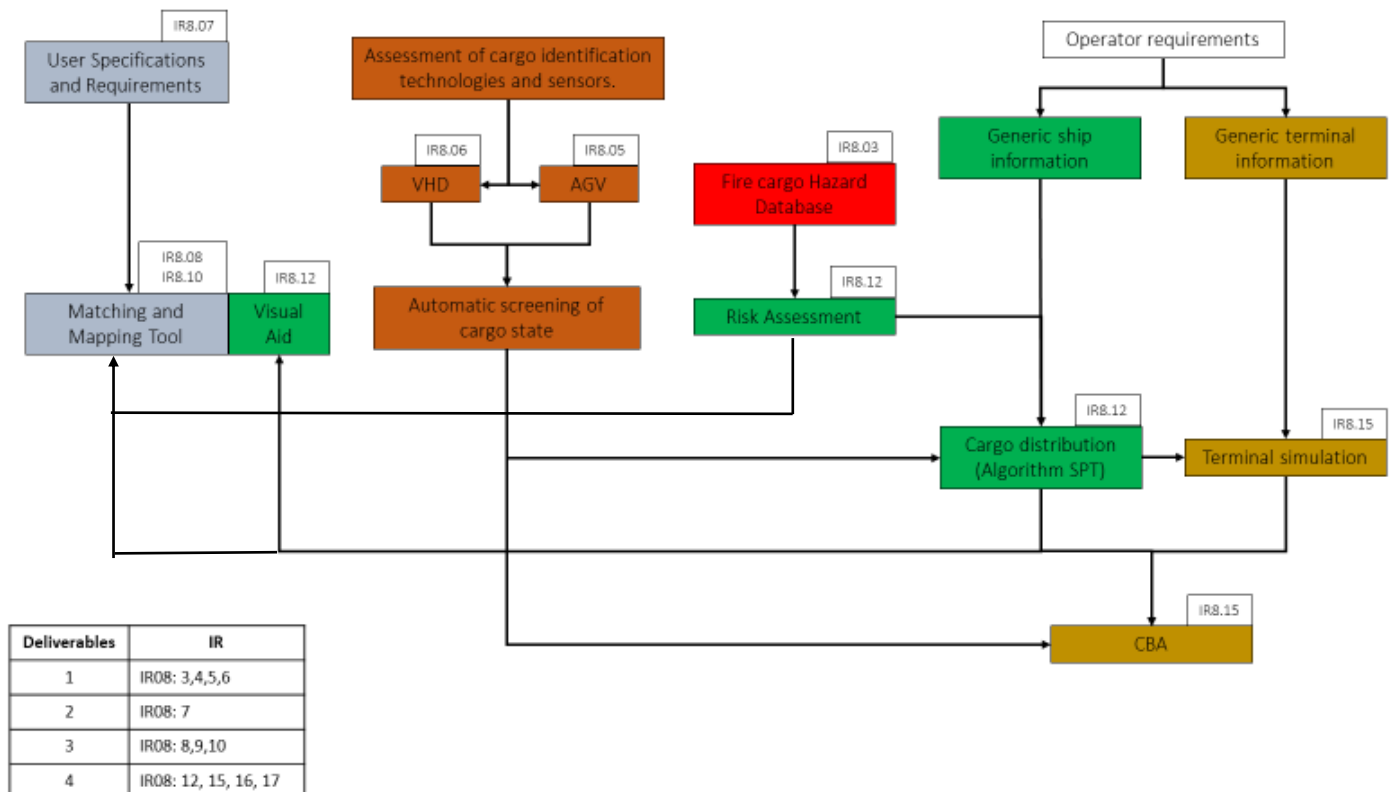


Figure 1 SW/HW components of the risk-based load planning tool

The contents of this deliverable D08.3 are located in the blue boxes in the left of the figure. The deliverable also will include requirements on technology for ignition prevention detectors, evaluation parameters and scenario definition for further prototyping and testing.

The figure also shows that the functionalities of the Matching and Mapping Tool and Visualization Aid are built as a continuum block. The final result of the development will result in a standalone visualization and interaction module for the Stowage Planning Tool, the overall software tool result of action 8-A. In this sense the work is split between the Fire Hazard Matching Tool (Task T08.5) and Visualization Aid (Task T08.6). Some of the functionalities of the final tool will only be developed in the scope of task T08.6 as they require more interaction with modules that are not yet in place. This deliverable will show all requirements of the tool but will develop as use cases only the ones that fit into the scope of task T08.5.

Matching and mapping tool (TASK T08.5) will display:

- Ship information
- Cargo information
- Dangerous Goods information
- The result of the initial risk assessment
- Fire Risk level of cargo units for each deck.

The mobile system screening a cargo deck is intended to reduce the risk of fires erupting during the voyage. By sending information of temperatures and fire hazards in combination with their location to the crew fires can be prevented at an early stage.

Even if the VHD and AGV modules are not yet in place at this moment the tool has implemented the functionality to display this kind of information. Images and data shown are realistic examples and not obtained from real devices.

Visualization Aid (TASK T08.6) will start from the tool developed in Task T08.5. The tool will be updated to show Stowage Planning Tool results and cargo status in real time using VHD and AGV real input. Functionalities related to manual tracking of the cargo units will also be implemented in the scope of this task.

Task T08.6 involves other software modules besides the Visualization Aid. In fact, an alternative way of working with the Stowage Planning Tool, a plug-in for existing software used by ship owners, will be implemented as part of the task.

4 Requirements

Main author of the chapter: Angel Priegue, CIM

In this section we will describe the initial conditions and set the different requirements of the fire hazard matching and mapping tool, including definition of conditions, user experience, functional and non-functional requirements.

According to the grant agreement the tool needs to include:

- Fire hazard matching by association of specific hazards to cargo units, according to input from T08.2 and T08.3,
- Development of integration software, fed by the database (T08.2) and with consideration to the identification technologies selected (T08.3).
- Development of hazard mapping tool with visualization of risky 'hot' zones and different hazard types of cargo.
- Support element to identify hazards associated to each zone of the ship according to the cargo unit's position (at planning and real level).
- Development of software to integrate feeds from identification of units on the deck and calculation of their associated hazard probabilities.

In brief, the Fire Hazard Matching tool, is a software that enables the visualization and identification of hazards associated to a given ship loading configuration according to the characteristics of the cargo units located in there.

The task starting point and basic requirement for its implementation consists of inputs from *T08.2 Definition of conditions for cargo fire hazard management and database construction* and *T08.3 Assessment of cargo identification technologies*.

A list of requirements for the Stowage Planning Tool were obtained after several project meetings interviews with people working in the Ro-Pax industry and as a result of meetings and workshops of the LASHFIRE project.

We present the full list of requirements for the final tool that needs to be implemented for the standalone display of the Stowage Planning Tool at the end of the action. Only some of them will be met in the implementation of the Fire Hazard Matching Tool in the scope of task 08.5.

1. *The system will provide the user with information on the selected load unit.*
2. *The system shall offer the user the possibility to confirm the location of a loading unit on a parking slot of a deck.*
3. *The system shall always offer the visualization of the current status of the cargo unit.*
4. *The system shall offer the user the possibility to extract the current cargo manifest.*
5. *The system shall offer the user the visualization of a unit load and its individual risk level.*
6. *The system shall provide the user with the visualization of the risk level of cargo located on each deck*
7. *The system shall provide the user with the visualization of the overall risk level of the ship's cargo*
8. *The system shall offer the user the possibility to visualize the information provided by external systems such as sensors and drones, for a cargo unit.*
9. *The system shall provide cargo location safety recommendations for each cargo unit.*
10. *The system shall provide the user with an incident table if the load cannot be located where expected.*
11. *The system shall offer the user the possibility to display deck plans of all the predefined ships.*

12. The system shall offer the user the possibility to visualize the voyage history of a given ship.
13. The system shall offer in real status the % of the loading process (Cargo currently confirmed and loaded on ship/total cargo to be loaded on ship), at deck and ship level.
14. The system shall offer users the possibility to do manual annotations on each loading unit.
15. The system shall provide users with the possibility to do manual annotations on each deck.
16. The system shall provide users with the ability to do manual annotations on the overall ship's view.
17. The system shall visually identify dangerous goods from other non-dangerous goods by colour coding.
18. The system shall be an interactive system.
19. The system shall allow marking of the unit load being loaded and unloaded.
20. The system shall be capable of running on mobile devices, adapting its visualization and navigability accordingly.
21. The system must be able to display information of a cargo unit in such a way that all users can identify the cargo.
22. The system must include an incident log.
23. The system shall allow searches within the loads
24. The software must show different screens/functions depending on the user who uses it.
25. When loading a new load configuration, any previously loaded configuration shall be stored in the log, informing the user beforehand.

Requirements 1,3,5,6,7,8,11,17,18,20,21,23,24,25 will be implemented as part of the Fire hazard Matching Tool in the scope of task 08.5 and will be specified and designed as use cases in this deliverable.

The other requirements will be implemented in the Visualization Aid in the scope of task 08.6 and will be specified in deliverable D08.4.

The next figure represents the distribution of software modules between the Fire Hazard Matching Tool and Visualization Aid.

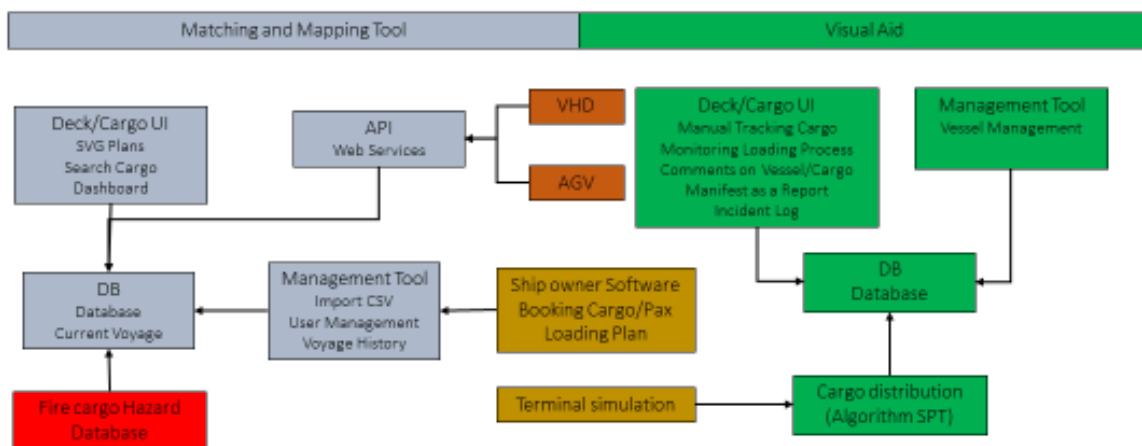


Figure 2 Distribution of modules of the risk-based load planning tool

4.1 Definition of conditions for cargo fire hazard

The main inputs of the system in order to define fire hazard conditions are the Hazard Database (developed in T08.2), Cargo Manifest and Loading Plan (provided by the ship operator), the sensor systems installed on the ship (defined in T08.3) and the collection of deck layouts of the ship (collected from WP5). More information on those inputs can be found in deliverable D08.1 Definition and parametrization of critical fire hazards, classification of cargoes, transport units, engines, fuels and ships and identification methodologies.

Basically, with these 4 input streams (Hazard DB, Cargo Manifest & Loading Plan, Sensors and Deck Layouts), the operator already has enough information to determine in the **Fire Hazard Matching Tool** application the degree of hazard that every cargo unit has and its location within the ship.

The starting key aspect of the software will be the cargo fire hazard management and database created in T08.2 where the system will check the conditions for cargo fire hazard. This previous work enables the possibility of developing tasks T08.5 and T08.6.

Hazard database identifies fire hazards associated to different types of cargo, in particular for hazards increasing the probability of ignition and such potentially causing large consequences.

The database will consider consolidation of regulatory, environmental, operational and shipyard requirements and establishment of necessary functions of a cargo fire hazard screening and management solution.

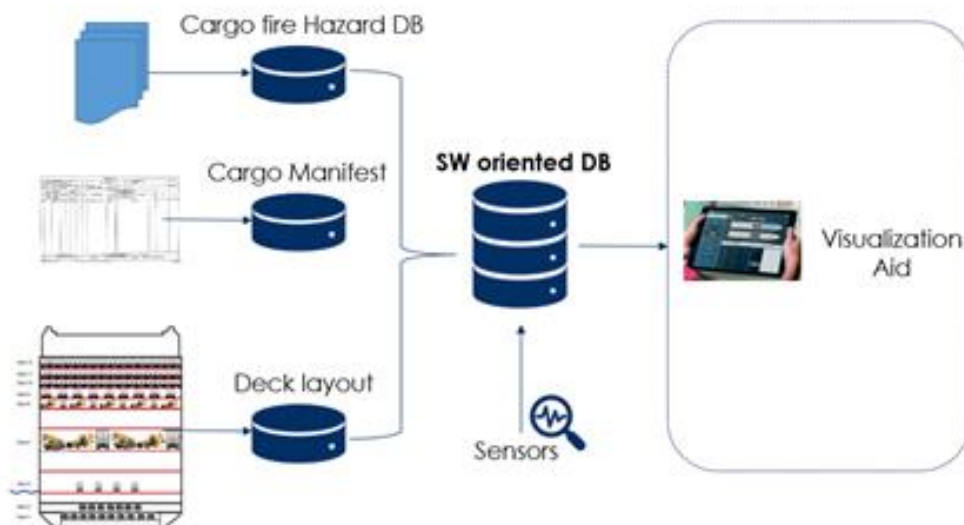


Figure 3 Cargo fire hazard database diagram

The Cargo Manifest, is the document where you will find the list of vehicles that are going to board the ship. Cargo Manifest includes the definition of the type of vehicle, with its technical characteristics such as its dimensions, type of engine, number of wheels, refrigeration capacity if it has that capacity and also the type of cargo it carries such as dangerous goods (explosives, gases, flammable liquids, flammable solids, corrosive substances,) and the type of packaging of the load.

The sensor system, allows us to take a snapshot of the state of the vehicle at the time of passing through the detector arc, mainly indicating the temperature of some critical elements in the vehicle (engine, wheels, ...), providing us with one more element of information to determine the degree of danger. In addition to the detector arch there are also mini-robots, which circulate below the vehicles already parked giving information to the system constantly.

Deck Layouts are obtained from information at the CAD level of the different decks that the boat has, allowing to see the spaces intended for loading, and the arrangement of the different elements of the ship.

Preliminary data provided from WP04 and study of fire accident investigations also complement the database, as well as heavy vehicle fire incident data and a summary of the most common vehicle fire origins and the results from interviewing ship operators and stevedores and an overview of current practices and accident statistics in other relevant fields like tunnels or car parking.

From the point of view of the fire hazard matching tool, the hazard database is only a data source that contains rules to establish the level of danger of the cargo in case of fire. The tool will display in different colors and highlight different risk scenarios according to the conditions

Visually there are 3 levels of fire danger catalogued as (high, medium and low), represented with 3 colors (red, yellow and green) respectively. These three levels are based on a combination of two elements, **frequency** and **severity**, (internally, each one of them is subdivided into 5 sublevels). Considering the frequency as the probability of fire being started at the cargo unit and the severity that depends on the type of material and the history of fires produced in the past.

To see more detailed rules of the Hazard database go to IR8.03

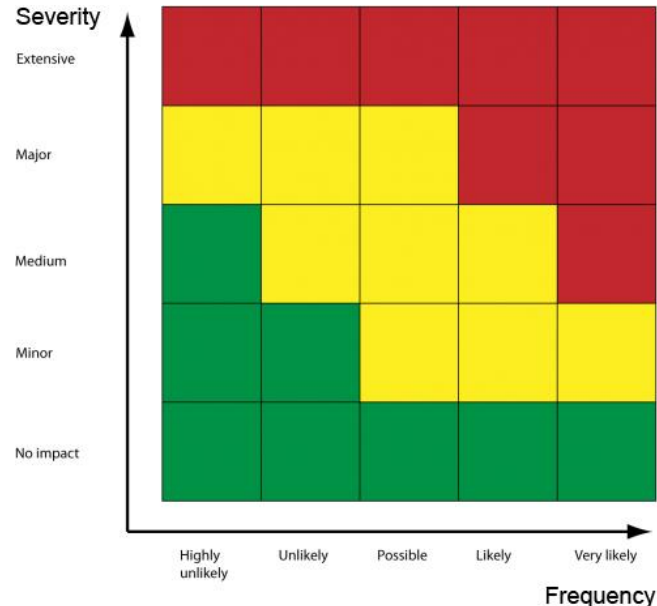


Figure 4. Hazard Levels

In order to properly assess whether fire accidents in each cargo unit are relevant, we must know how many units of that type of cargo are transported in order to evaluate their impact.

For this purpose, information on the cargo transported during one year was requested from the operators and a pattern was obtained of what is the average of each type of cargo transported in a typical ship.

This helps us to develop the frequency of occurrence of an accident per unit of cargo.

The pattern used is the one shown below (the general data of the cargo transported during a year is confidential, that is why only the pattern resulting from the analysis is shown), in order to perform the Risk Assessment, we have only selected from the pattern, the type of cargo where the Fire Hazard Database shows that an accident has occurred in the same.

$$\text{Frequency of occurrence } i = \frac{\sum_{j=0}^n \text{Fire accidents}_i}{\text{Units } i \text{ on the Lash Fire general ship} \cdot \sum_{j=0}^n \text{routes}}$$

i : Type of cargo

j: years (2013/2020)

General Cargo:

Table 1. Typical General Cargo Units carried by a ship

	Typical Units carried by a ship
Conventional vehicle Bus	2
Conventional vehicle Car	63
Conventional vehicle Truck	41
New energy carrier Electrical vehicle	4
Reefer unit Value	8
Special vehicle trailer	32
Special vehicle RVs	10
Special vehicle Tractor	4

Dangerous Goods:

Table 2. Typical Dangerous Cargo Units carried by a ship

	Typical Units carried by a ship
Dangerous goods Corrosive substances	1
Dangerous goods Explosive	1
Dangerous goods Flammable liquid	1
Dangerous goods Flammable solid	1
Dangerous goods Gas	1
Dangerous goods Miscellaneous dangerous substances and articles	1
Dangerous goods Undeclared DG	1

For each type of load, the causes that have produced the fire in the load have been analysed, classifying them in causes of origin:

- Electrical
- Overheating
- Leakage of liquids
- Mechanical
- Other

In addition to each type of failure, the severity of the accidents has been analysed, within each category, for the severity the following scale has been used.

According to IMO definitions (IMO's "Casualty Investigation Code" in its updated version and IMO Circular MSC-MEPC.3/Circ.3), severity is classified into the following levels: Marine accident is considered any marine casualty or marine incident. An accident does not include a deliberate act or omission, with the intention to cause harm to the safety of a ship, an individual or the environment.

Accidents may be classified (in order of severity) as follows:

- very serious marine casualties
- serious marine casualties
- less serious casualties
- marine incidents
- near miss

The results of the analysis can be summarized in the following table:

Table 3. Risk Levels General Cargo

Cargo unit (general cargo)	Frequency of occurrence	Frequency of occurrence per NM	Base Risk Level
Reefer unit Value	8,1577E-07	2,2129E-16	100%
Conventional vehicle Bus	4,7586E-07	1,2909E-16	80%
Conventional vehicle Truck	1,5168E-07	4,1146E-17	60%
Special vehicle RVs	7,9311E-08	2,1514E-17	40%
Conventional vehicle Car	6,9077E-08	1,8738E-17	40%
Special vehicle Tractor	3,9655E-08	1,0757E-17	30%
New energy carrier Electrical vehicle*	2,9741E-08	8,0679E-18	30%
Special vehicle trailer	3,8376E-09	1,041E-18	10%

Table 4. Risk Levels Dangerous Cargo

Cargo unit (dangerous cargo)	Frequency of occurrence	Frequency of occurrence per NM	Base Risk Level
Dangerous goods Flammable solid	5,9483E-07	1,6136E-16	90%
Dangerous goods Flammable liquid	3,569E-07	9,6815E-17	70%
Dangerous goods Miscellaneous dangerous substances and articles	2,3793E-07	6,4543E-17	60%
Dangerous goods Corrosive substances	1,1897E-07	3,2272E-17	50%
Dangerous goods Explosive	1,1897E-07	3,2272E-17	50%
Dangerous goods Gas	1,1897E-07	3,2272E-17	50%
Dangerous goods Undeclared DG	1,1897E-07	3,2272E-17	50%

The base risk level is the measure used in the Fire Hazard Matching Tool to display visually the different risk levels of each cargo unit in the ship.

More details on the risk assessment calculations can be found in D08.4 Stowage planning optimization and visualization aid.

4.2 Requirements on UX

The IMO agreed in March 2015 on a *Guideline on Software Quality Assurance and Human-Centred Design for e-navigation*[1]. This document points to the importance of using human-centred design principles in development of new software systems that are to be handled by humans. Not only the technical aspects but also the user experience (UX) should be in focus.

Good usability is defined as the “extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”(ISO 9241-11:2018).

Good usability is essential created with good design developed in an iterative design process together with the end users. The human-machine interface (HMI) of the Fire hazard matching tool will be developed according to this process:

- 1) Research the context of use (ship visits, observation of fire drills and interviews with stakeholders),
- 2) specify the user and organizational requirements,
- 3) produce design solutions (first simple paper prototypes, later mid-fidelity and high-fidelity mock-ups),
- 4) evaluate design against the requirements. (ISO9241-11:2018)

The **Fire Hazard Matching Tool** shall be designed following the processes and methods described in ISO 9241-210 (Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems).

4.3 Technologies for ignition prevention detectors

The objective of technologies for ignition prevention is to develop further and adapt approved systems which are working on land based roads and tunnels. One system to detect hot spots, incandescent systems or smoldering fires on vehicles before entering for example tunnels.

Sensor based systems should be adapted further from existing and well-known land based industrialized applications to the maritime industry. Possible sources of ignition, incandescent systems on vehicles or smoldering fires on vehicles have to be detected automatically with a so-called Vehicle Hot Spot Detection system (VHD) when entering ferries.

The second system detects vehicles transporting hazardous materials. Hazardous materials have an impact on fire safety on-board right when vehicles enter a ship. Knowing the number and type of vehicles transporting hazardous materials on board is important for the voyage of the ship.

While two different sensor principles read independently of each other, the identification of the hazardous material placard is achieved. The system is called “detection of vehicles transporting hazardous materials” (VDG). This ensures the read rate is increased considerably and misinterpretations are avoided. The generated data out of both sensor systems has to be linked to a cloud-based platform and used for the boarding procedures.

The detection systems have to be able to work on different kind of ferries, like ro-pax, ro-ro as well as on car/vehicle carriers. Different ferries have different loading ramps and open decks with harsh environmental conditions like salt spray water from the sea, vibrations from the engines. The detection systems must not interfere with the bridge radio frequency, especially during the voyage. The sensor systems have to withstand the conditions and work in the next harbor, probably.

Special trailers, trucks and heating or cooling systems on trailers are different objects than those known from road applications. Sensor systems and the algorithms to generate images have to be modified and extended for identification of different types of vehicles entering the ship.

It is foreseen to start with the VHD system and install it in a first step on the port side of a ferry terminal to detect hot spots on vehicles. The focus is to extend the internal library of vehicles in the evaluation software of the VHD. Especially in ports, different type of vehicles and trailers with different heating or cooling systems, are present and entering ferries. In a second step, the VHD system can be mounted directly on a ferry to see how the system works directly on board on real cases.

Different type of vehicles, trailers, trucks and cooling -/ heating systems are in use and the sensor system will be tested under real conditions. The evaluation algorithms to generate images from the vehicles will be extended.

If a sensor system is tested, approved of the reliability and performance during the duration of the project, the outcome can be that an early detection system of hotspots and clear identification of dangerous goods will be a system to prevent fire on board of ferries.

The online generated data will be sent to a cloud-based data recording system. Shipping documents can be checked against the real data on-line. In addition, vehicles carrying dangerous goods can be automatically guided to pre-defined loading zones on board where special firefighting systems are in place.

MOBILE SYSTEM

To prevent fires from starting in vehicles or cargo loaded into a ship a solution is intended to monitor both cargo and transports to find deviating temperatures and cargo that has been identified as fire hazards. This information is to be shared for fire prevention in time or to know where efforts need to be focused.

A two-system solution is proposed based on one stationary system placed before boarding and another mobile system operating inside of the ship. When a transport approaches for boarding it will pass under the stationary system making it possible to detect abnormal heat at a time when it is still possible to prevent it from boarding the ship until it has been further investigated. The mobile solution will continuously travel through the cargo hold and first map out where objects categorized as fire hazards are on a map and then measure temperatures. If there is a temperature reading that stands out it can then be reported alongside location to the ship's crew for further inspection

There are two drone-based suggestions for the mobile system, either a ground moving drone, or an aerial drone will be used. Some of the positives with having a drone that only moves on the ground is that it removes some of the safety risks and complexity. On the other hand, it also requires enough space for it to be able to move underneath or on the side of the vehicles and cargo. Some of the main challenges for such a prototype are:

- The possibilities for communication are not guaranteed to be constant
- The drones need to be able to come close to the vehicles and cargo for measurements
- The weather/smoke/sprinklers can have a large effect on the sensors
- Maneuverability among vehicles independent of parking

One structure alternative for these devices, is the Zumo 32U4 [6]. It is a small, about 100 x 100 mm, and cheap robot that can be useful for early testing. It may although be too small to mount a LiDAR, other sensors and processing equipment that might be necessary. Another possibility would be to use an MLT-JR tracked robot, which even though larger than the Zumo alternative provide a sturdier frame better able to carry sensors and handle uneven terrain with its size of around 27.31 x 31.75 x 10.16 cm, where the highest point comes from the track wheels.

It would also be possible to refurbish a robot that is already on the market, such as a cleaning robot Neato D7. There are many components that would have to be removed e.g. the whole computational unit. It would, however, provide a completely assembled system with LiDAR, battery, motors and frame.

A solution based on three Lithium-Ion Polymer (LIPO) batteries is suggested. Two of which will power the motors and one of them will exclusively power the control card. This design guarantees an isolation between motors and controller in case of undesired power peaks from the motors

It is expected that at least some of the functionalities for the drone such as the navigation or the object detection will be possible to speed up with a GPU compared to a CPU. A processing unit similar to a Jetson board is, therefore, preferable compared to only CPU units.

One alternative is Jetson Xavier NX. It is a small sized GPU and CPU unit with a size of 45 x 69.6 mm. The working memory is 8 GB and total storage is 16 GB. Although the storage is not impressive the drone is not intended to store much information other than the system needed. It can run on between 10 and 15 W. There might be a need for a fan specifically for the processing unit, which would increase the size and power consumption. To minimize the size of the processing unit it could be of interest to use cables to attach it perpendicular to the motherboard. It is also possible to purchase it mounted on

a motherboard and including a fan. With this setup the full dimensions will instead be 103 x 90.5 x 34.66 mm.

Another alternative would be to use an NVIDIA Jetson TX2 module. The Jetson TX2 is a 7.5-watt computer similar to the Xavier but slightly less powerful. It is, however, smaller when coupled to a ConnectTech Quasar breakout module the size is 87 x 53 x 53 mm. There is also an integrated WiFi functionality.

Whatever the type of drones, they will include suitable technologies to work properly in environments like ro-ro spaces:

LiDAR

The intention is to use a LiDAR sensor to provide the necessary information for e.g. mapping, navigation and collision avoidance. If the LiDAR would be used only for navigation it could be possible to have a limited angle, given that it could provide enough information for a SLAM system to recognize its current position. For a collision avoidance system, however, a limited angle would likely not be enough as it will have to avoid not only objects in front but from all angles and in the case of an aerial drone above and below as well.

Among the considered alternatives are Ouster OS1 LiDAR [1], Slamtec RPLIDAR A3 [2] and YDLIDAR G4 [3]. Both the YDLIDAR and the RPLIDAR reports a height of 4 cm, while it is larger for the OS1 LiDAR. The OS1 claims to reach a range up to 60-120 m depending on reflectivity, while RPLIDAR have a range between 10-25 m and YDLIDAR 16 m. Given that the drone will operate indoors in small environments, such as, underneath cars it is likely that a range of 10-15 m will be sufficient. Further testing is still needed though as the navigation might not be able to gain sufficient information in those ranges if no objects are close enough. The rotation rates are between 12 Hz(YDLIDAR) and up to 20 Hz (OS1, RPLIDAR). Given that the sampling rate is high enough the maximum delay with which an object should be detected is about the rotation rate. There is also a delay in processing and sending the required data, which will be added to the total reaction time of the systems in the drone dependent on a LiDAR. The YDLIDAR can handle temperatures between 0 and 50 Celsius, RPLIDAR have similar temperatures but a maximum temp of 45 Celsius instead. The OS1 also handles maximum temperatures of 50 Celsius but have -20 as a minimum going, as far as, -40 with a specific cold resistant model. On an open deck a minimum temperature of 0 Celsius is likely too low, it could, however be sufficient for indoor applications.

Camera RGB

A simple sensor type to use for object detection is the RGB camera. It is a common sensor for the purpose, it can also be both cheap and easily accessible. While the requirements on the camera are very low and there is an abundance of choices, specific models will not be covered here.

Heat sensor

Heat detection is a core functionality requirement and, thus, a sensor to detect temperatures will be integrated into the system. One of the options would be to use an IR camera. Among the potential candidates is a FLIR ONE PRO ANDROID [4]. It has a resolution of 160 x 120 pixels and heat detection range from -20 to 400°C. It is mounted on a mobile phone and have a temperature sensitivity of 70 mK. It is possible that, although small the requirement of mounting it on a mobile phone could create a total height that is too large. Therefore, a second candidate is the FLIR A65 [5] that does not need to be mounted on a mobile phone. It has a size of 104.1 x 49.6 x 46.6 mm, a resolution of 640 x 512 pixels and sensitivity of 50 mK. A third possibility would be to use a radiometric capable LongWave

Infra-Red (LWIR) micro thermal camera module from FLIR, model Lepton 3.5. Which also have the potential to be mounted on a PureThermal-2 FLIR Lepton smart I/O module from GroupGets. The final dimensions of this setup are 30 x 22 x 8 mm.

4.4 Use Cases and Functional Requirements

A simplified description of the functionality of the hazard mapping tool would be: given a certain ship deck configuration and cargo manifest, the hazard matching tool will identify and detect the potential risks associated to current loading plan and show this information to the operators of the ship using hand-held devices.

The functional requirements for the application are a subset of the list of the requirements of the application (Chapter 4 – Requirements):

- 1. The system will provide the user with information on the selected load unit.*
- 3. The system shall always offer the visualization of the current status of the cargo unit.*
- 5. The system shall offer the user the visualization of a unit load and its individual risk level.*
- 6. The system shall provide the user with the visualization of the risk level of cargo located on each deck.*
- 7. The system shall provide the user with the visualization of the overall risk level of the ship's cargo.*
- 8. The system shall offer the user the possibility to visualize the information provided by external systems such as sensors and drones, for a cargo unit.*
- 17. The system shall visually identify dangerous goods from other non-dangerous goods by colour coding.*
- 21. The system must be able to display information of a cargo unit in such a way that all users can identify the cargo.*
- 23. The system shall allow searches within the loads.*
- 24. The software must show different screens/functions depending on the user who uses it.*
- 25. When loading a new load configuration, any previously loaded configuration shall be stored in the log, informing the user beforehand.*

Below is a definition of the different actors involved in the cargo of the ship, according to their responsibility.

The different types of actors / roles that participate in the system are described below. By classifying the actors according to their role in the system, it can be clearly seen that we have users who actually interact with the application and other actors who are information providers, but both are equally important.





 Users	<p>Although in real scenarios we can consider a considerable number of actors (Cargo/Travel office staff, Loading Officer, Deck crew, Chief deck officer, Bosun, Captain, ...) for the software system we will perform an abstraction to 2 types of access levels.</p> <p>Initially we will only consider 2 user levels with access to the software system. A normal user, who will only be able to perform read operations and will not be able to modify any data and an administrator user that will be able to make all kind of operations available by the system.</p>
 Database systems  Cloud platforms	<p>Cargo Fire Hazard DB is a database where the fire hazard level related to every cargo unit is stored according to the type of vehicle and goods transported.</p> <p>Cargo Manifest DB, this database is initially generated by the Charges/Travel office, and it defines the characteristics of the vehicles to be loaded.</p> <p>SW oriented DB, this database is hosted in the backend of the system and is the primary database of the application. This is where the arrangement of the vehicles within the ship, users who can access and the log of the actions performed are stored.</p> <p>Deck layout, in the backend will also host the different geometric layouts of the ships, so that they can be represented by the tool.</p>
 Sensors	<p>Sensors are considered as actors that do not interact with the system as human users, but feed information to the tool. Through an API provided by the system, the sensor provider will have an end point to fill the SW oriented database.</p>

Table 5. Types of users

The use cases of **normal users** (read operations), **administrator users** (write operations) and **sensor users** (input of data from external sensors) are defined as described in figures 3, 4 and 5.

Normal User

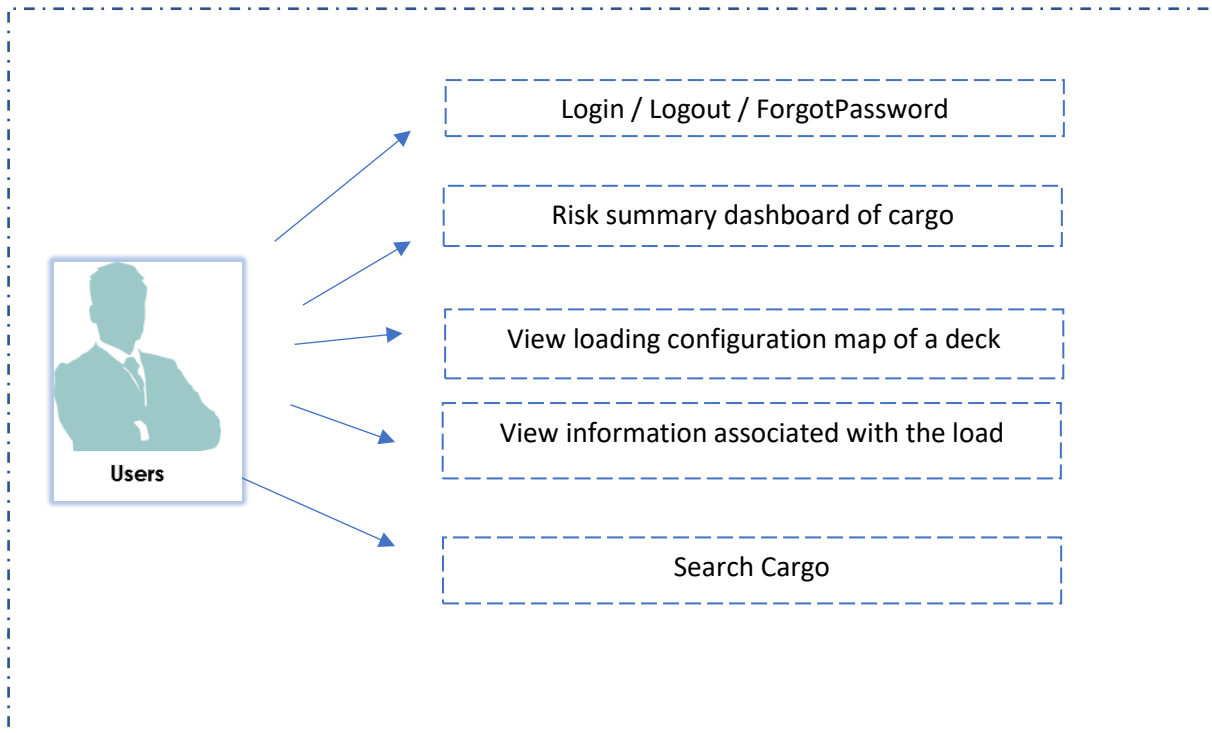


Figure 5. Normal user use cases

Administrator User

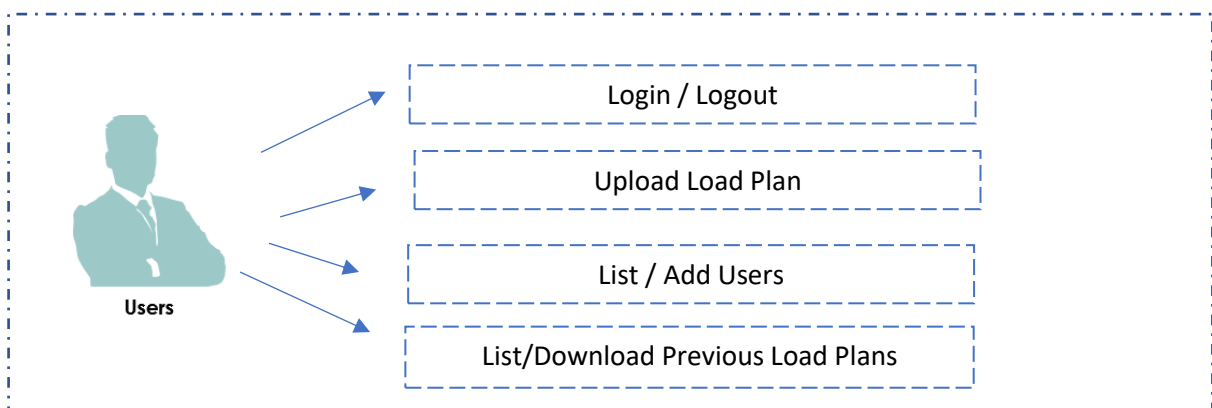


Figure 6. Administrator user use cases

Sensor User

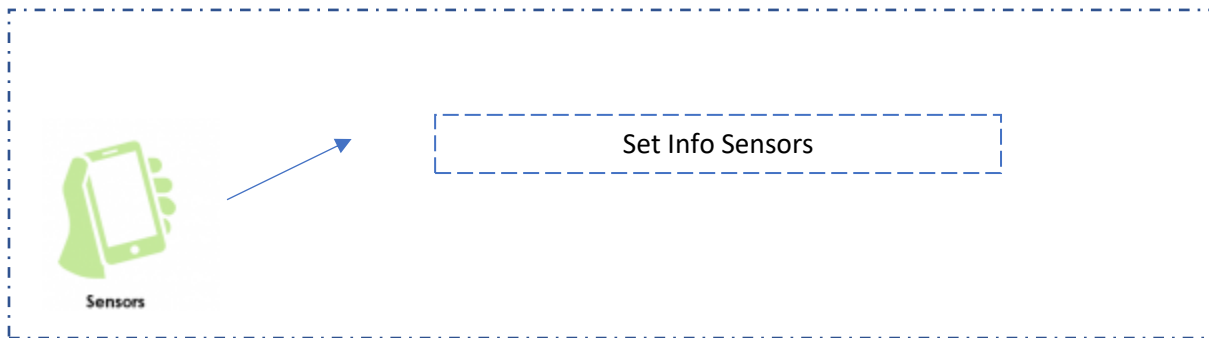


Figure 7. Sensor user use cases

4.1.2 Normal user use case contracts

The following tables (6 to 12) show the specification of the use cases of a normal user. Normal users have access only to read operations.

Table 6. Use Case 01

CA-01	Login	
Precondition		
Description	The user, by entering his user credentials and password, is able to access his private area.	
Normal Sequence	Step	Action
	1	The user enters his credentials and decides if he wants to save his data for later logins and requests to authenticate.
	2	The system checks if the credentials are correct or not.
	3	The user access to the private area.
Postcondition	The user is authenticated in the private area with their data and, if user requested to save credentials, avoid future login.	
Exceptions	3. The user receives an error message and remains on the login screen.	
Comments		

Table 7. Use Case 02

CA-02	Logout	
Precondition	The user is previously logged into the system, after entering the username / password.	
Description	The system closes the user session, redirecting him back to the login screen.	
Normal Sequence	Step	Action
	1	The user with a session already started asks the system to exit the system by means of an action.
	2	The system redirects the user to the login screen.
Postcondition	The user, who had a session open in the system initially, stops being so by deleting his data from the session created previously and becomes ready to start again.	
Exceptions		
Comments		

Table 8. Use Case 03

CA-03	Forgot Password	
Precondition		
Description	The user recovers his password by entering his email address. Later, he receives an email with his password.	
Normal Sequence	Step	Action
	1	The user asks the system to retrieve the password.
	2	The system asks the user to enter their email.
	3	The user enters his email.
	4	The system sends an email with the user's password.
Postcondition	The user receives an email with the password and thus can go to the login screen.	
Exceptions	4 The system reports that this user is not registered.	
Comments		

Table 9. Use Case 04

CA-04	Summary dashboard of cargo/risk	
Precondition	The user is previously logged into the system, after entering the user / password.	
Description	The user sees a summary of the cargo carried by the ship and a total score of the fire risk, along with a graph of the total cargo units according to their level of fire hazard	
Normal Sequence	Step	Action
	1	The user requests to see the dashboard
	2	The user sees a summary of the cargo carried by the ship and a total score of the fire risk, along with a graph of the total cargo units according to their level of fire danger.
Postcondition	The user will be able to see the following information: <ul style="list-style-type: none"> • Total No of bookings / Total No. of chassis/export cars / Total Length (m), Total goods (Kg), Total Weight (Kg). • Global score of fire risk / graph of the total cargo units according to risk level. • Detailed summary according to vehicle (table) 	
Comments	This use case implements the following requirements: 7. The system shall provide the user with the visualization of the overall risk level of the ship's cargo	

Table 10. Use Case 05

CA-05	View load map of a deck	
Precondition	The user is previously logged into the system, after entering the user / password and has already requested to see a deck.	
Description	The user wants to view a load plan of a deck including the visualization of risky 'hot' zones and different hazard types of cargo as a support element to identify hazards associated to each zone of the ship according to the cargo unit's position.	
Normal Sequence	Step	Action

	1	The user requests to view one of the decks of the ship.
	2	The user can see the load plan of the selected deck with risk levels of every cargo unit in the deck with a color map.
	3	The user can drag / zoom in / zoom out the deck layout being viewed, relative to the center.
Postcondition	The user visualizes the deck selected, risk levels of cargo units are shown as a color scale from red to green. Every zoom action makes 50% bigger / smaller the deck layout that he was visualizing.	
Exceptions		
Comments	<p>This use case implements the following requirements:</p> <p>5.The system shall offer the user the visualization of a unit load and its individual risk level.</p> <p>6. The system shall provide the user with the visualization of the risk level of cargo located on each deck</p> <p>11. The system shall offer the user the possibility to display deck plans of all the predefined ships.</p> <p>17.The system shall visually identify dangerous goods from other non-dangerous goods by colour coding.</p>	

Table 11. Use Case 06

CA-06	View Cargo Info	
Precondition	The user is previously logged into the system, after entering the user / password and has already requested to see a deck.	
Description	The user asks on the deck that he is viewing an element of the cargo, in order to see its information.	
Normal Sequence	Step	Action
	1	The user requests to see the information associated with a cargo.
	2	<p>The system displays the associated information.</p> <ul style="list-style-type: none"> - General Information (BookingID, Type Vehicule, Total Weight, ...) - Sensor Information (Hot-spot, vehicle, passenger area, loading area, engine, Wheel hub, exhaust, picture) - Fire Hazard Information (Observations)
Postcondition	The user sees in a grouped way all the information associated with the cargo for which the action is required.	
Exceptions		
Comments	<p>This use case implements the following requirements:</p> <p>1.The system will provide the user with information on the selected load unit.</p> <p>3. The system shall always offer the visualization of the current status of the cargo unit</p> <p>8.The system shall offer the user the possibility to visualize the information provided by external systems such as sensors and drones, for a cargo unit.</p> <p>21.The system must be able to display information of a cargo unit in such a way that all users can identify the cargo.</p>	

Table 12. Use Case 07

CA-07	SearchCargo	
Precondition	The user is previously logged into the system, after entering the user / password and has already requested to see a deck.	
Description	The user requested to the system to see the information associated with a cargo.	
Normal Sequence	Step	Action
	1	The user makes the request to search for cargo.
	2	The system asks the user for the bookingID and offers the option to search for it in the current deck, or in any deck.
	3	The system finds the requested information and displays it.
Postcondition	The user can see the information associated with the requested booking ID.	
Exceptions		
Comments	This use case implements the following requirements : 23.The system shall allow searches within the loads	

4.1.3 Administrator user use case contracts

Next, we show the functionalities of each user-type actor, knowing that the administrator user can perform the operations that allow him more than the actions of normal users.

Table 13. Use Case 08

CA-08	UploadLoadPlan	
Precondition	The user is previously logged into the system, after entering the username / password. View your admin menu.	
Description		
Normal Sequence	Step	Action
	1	The user requests to add the load plan to the repository.
	2	The system offers the user a form where to upload a CSV file
	3	User selects CSV file
Postcondition	The CSV file is uploaded to the system and integrated as a new source of information to the system.	
Exceptions	The CSV file does not have the default format.	
Comments	This use case implements the following requirements : 25.When loading a new load configuration, any previously loaded configuration shall be stored in the log, informing the user beforehand.	

Table 14. Use Case 09

CA-09	List / Add User	
Precondition	The user is previously logged into the system, after entering the username / password. View your admin menu.	
Description	The user can see the existing list of user of the system and can create a new user.	
Normal Sequence	Step	Action
	0	User list the user system.
	1	User selects create new user.

	2	The system displays the user creation form.
	3	The user indicates his login, his password and the type of company / ship he belongs to
Postcondition	The system creates a user with the data entered, to be able to log in.	
Exceptions	User login already exists.	
Comments		

Table 15. Use Case 10

CA-10	List/Download Previous Load Plans	
Precondition	The user is previously logged into the system, after entering the username / password. View your admin menu.	
Description		
Normal Sequence	Step	Action
	1	The user list of previous load plans.
	2	The user selects one load plan.
	3	The user download the select load plan.
Postcondition	The system generates a response with the load plan as a CSV file.	
Exceptions		
Comments		

4.1.4 Sensor user use case contracts

As we have commented before, although the sensors are not "users" properly speaking, they are who are actors in the system because they interact.

Table 16. Use Case 11

CA-11	SetInfoSensor	
Precondition		
Description	The sensor arch establishes a connection with the central server to store the information sent in the database	
Normal Sequence	Step	Action
	1	The sensor arch makes the call to the communication API, with parameters idCargo (identifier Cargo), and with parameter info (information associated with the cargo)
	2	Check idCargo exists
	3	Save info data in the database
Postcondition	The information associated with the idCargo is stored in the database so that it can be consulted later.	
Exceptions		
Comments		

4.5 Non-functional requirements

Non-functional requirements have major influence on architectural design. They describe the quality attributes of a system. The following requirements have been considered for the definition of the architecture of the tool:

- **Scalability:** The application must be built on the basis of an evolutionary and incremental development, in such a way that the new functionalities and related requirements can be incorporated affecting the existing code in the least possible way, for this, aspects of component reuse must be incorporated.
- **Security:** The application must avoid the typical WEB attacks, such as Cross Site Scripting (XSS), SQL injection, ... which is based on the introduction by the user of malicious scripts, therefore, all user input will be validated. Other types of attacks such as brute force targeting the user login will be prevented by blocking the login for a period of time.
In addition to the user credentials, they will be saved in HASH mode and therefore a physical attack on the server would not specifically reveal the user's password.
- **Availability:** The application should perform over a period of extended time. Response times of the application must not degrade the longer the application is available. The application must be able to support a continuous level of availability under levels of normal operating volumes and concurrency, with no application performance degradation over a period of time between planned application restarts.
- **Extensibility:** The application architecture will allow new kind of inputs (hazard database conditions, ship types, ...). These changes could be made in a fairly agile way and will not require extra programming.
- **Portability:** The application is designed to run on handheld devices (mobiles or tablets)
- **Compliance to policies and standards:** GDPR must be implemented.
- **Architecture:** The application should try to work as much as possible offline. That is, once the User is authenticated and loaded the manifest into memory, he must move through the different covers without consulting Internet.

4.6 Cyber Security requirements

Confidentiality, integrity and availability, also known as the CIA triad[3], is a model designed to guide policies for information security within an organization. The model is also sometimes referred to as the AIC triad (availability, integrity and confidentiality) to avoid confusion with the Central Intelligence Agency. Although elements of the triad are three of the most foundational and crucial cybersecurity needs, experts believe the CIA triad needs an upgrade to stay effective.

The three key concepts that form the CIA triad are:

- **Confidentiality:** To protect sensitive and private information from unauthorized access. Protecting confidentiality is dependent on being able to define and enforce certain access levels for information. In some cases, doing this involves separating information into various collections that are organized by who needs access to the information and how sensitive that information actually is. Some of the most common means used to manage confidentiality include access control lists, volume and file encryption, and Unix file permissions.
- **Integrity:** Data integrity is what the "I" in CIA Triad stands for. This is an essential component of the CIA Triad and designed to protect data from deletion or modification from any

unauthorized party, and it ensures that when an authorized person makes a change that should not have been made the damage can be reversed.

- **Availability:** This is the final component of the CIA Triad and refers to the actual availability of your data. Authentication mechanisms, access channels and systems all have to work properly for the information they protect and ensure it's available when it is needed.

With each letter representing a foundational principle in cybersecurity, the importance of the CIA triad security model speaks for itself. Confidentiality, integrity and availability together are considered the three most important concepts within information security.

The developer team of the hazard mapping tool will consider these three principles together to guide the development of security policies for the tool. When evaluating needs and use cases, the CIA triad will help developers to ask focused questions about how value is being provided in those three key areas. Thinking of the CIA triad's three concepts together as an interconnected system, rather than as independent concepts, can help us understand the relationships between the three.

Specifically, we can ensure that we meet the 3 fundamental criteria of the CIA, basically because confidentiality (access to only authorized information) is related to the fact of accessing the web through the HTTPS protocol, which is a protocol end-to-end encryption security, which together with the login using unique access credentials allow us to guarantee that users can only access the resources they have requested, if and only, if they are authorized. The integrity of the data (the security that the data has not been manipulated) is maintained by protecting the information with a HASH in each of the rows of the DB, in addition the TLS protocol itself is responsible for verifying the integrity of all transmitted data and finally availability (access to information when the user needs it) is guaranteed by having a Web server with 24/7 characteristics, this logically does not prevent possible DOS attacks, but it minimizes them.

5 Software Architecture

Main author of the chapter: Angel Priegue, CIM

In this section we will describe the software architecture selected for the implementation of the fire hazard matching and mapping tool. We describe the three different layers of the 3-layer architecture pattern, a well-established software application architecture that organizes applications into three logical and physical computing layers or tiers: the presentation layer, or user interface; the application layer, where data is processed; and the data layer, where the data associated with the application is stored and managed.

There are many benefits to using a 3-layer architecture including speed of development, scalability, performance, and availability. Modularizing different tiers of an application gives development team the ability to develop and enhance a product with greater speed than developing a singular code base because a specific layer can be upgraded with minimal impact on the other layers

An extra chapter will be added regarding security, the application of techniques that assess, mitigate, and protect any software system from vulnerabilities.

5.1 Data Layer

The data layer comprises of the database system and the different software modules wrapping the access to the database, which is an organized collection of data, generally stored and accessed electronically from a computer system.

The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyse the data. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system".

The DBMS that will be used in the hazard matching and mapping tool will be SQL Server.

In fact, today we could have chosen practically any DBMS that is server and relational, since large volumes of data are not expected and therefore the query speed of any DBMS will be sufficient. Also SQL Server is proven to be a good partner for ASP NET technology.

The data layer will also include the use of an object-relational mapper (ORM), a code library that automates the transfer of data stored in relational database tables into objects that are more commonly used in application code.

The ORM will provide a high-level abstraction upon a relational database that allows developers to write native code using the application programming language instead of writing SQL statements to create, read, update and delete data in the database.

In any case, the access to the data is done through an ORM (Object – relational mapping), this makes us independent from the final DBMS and therefore it is easily replaceable. The fact of having selected SQL Server has been for convenience since we have a lot of experience.

The connection layer between the different clients (sensors / web browser) is through the API layer, which is a light layer with the Web Services necessary to interoperate with the system. This layer guarantees the security of the resources that the user can access. Therefore, clients never access directly with the database, but through the API, and more internally it is the API layer that

communicates with the ORM in order not to interact directly with the database and become independent as we have said before the final DBMS.

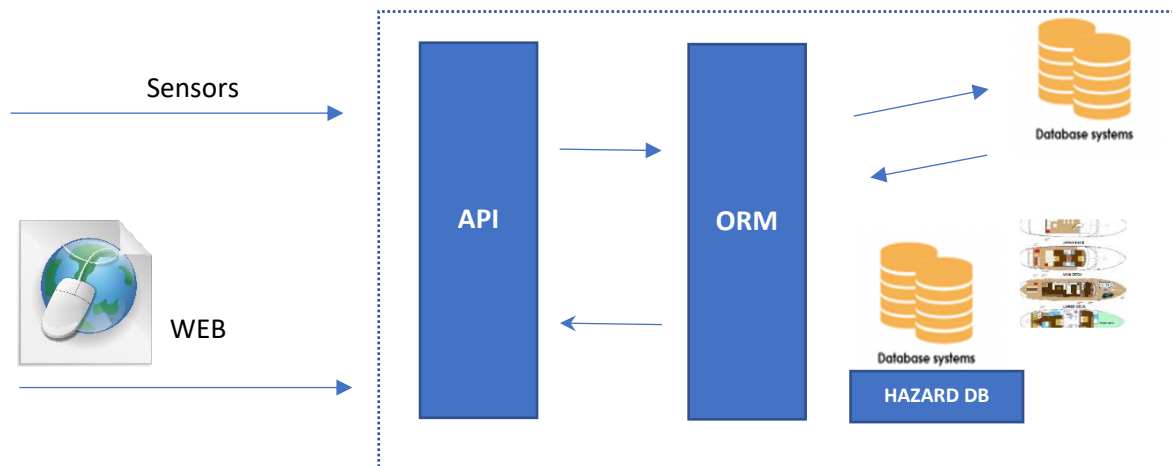


Figure 8. Data Layer

5.2 Application Layer

The application layer contains the functional business logic which drives an application's core capabilities. In this particular case, we will use ASP.NET which is a software framework for Web development, which has been shown to be efficient and allows in a way the decoupling of the view and the design. In addition, as it belongs to the .NET framework, it has the ability to be easily extensible and the possibility of developing in an object-oriented programming language such as C#.

It is known that for the client part Javascript is usually used, however, we will also use a framework that allows us to isolate ourselves in a certain way from the peculiarities of different browsers (JQUERY).

These frameworks combine very well with each other and from ASP.NET they can be downloaded as Nuget packages, which makes it easier to create a fairly complete solution based on modules already created and tested, therefore, they allow us to focus on business logic, abstracting a little of the complexity of making a multi-device application, which reduces development and testing time.

Of course, several other software frameworks could also be suitable for the implementation of the software but CIMNE has a wide experience in this technology and the selection of a previously known technology enables to speed up the development process.

5.3 Presentation Layer

Among the different technologies to implement the Fire Hazard Matching Tool presentation layer, the first thing is to determine whether it will be a native application of the client device (Windows Form, Android App, iPhone App, ...) or a web application, to be used in a browser.

As a rule, native applications have a number of a priori advantages, because they are usually faster at performing some tasks, because they do not depend so much on the server, because they have enough computing power to perform these client operations, this is because native applications have greater control of the device and therefore can access customer resources more directly and efficiently, such as access to sensors, or exceed the local storage limits imposed for security reasons in web applications. However, not only are there advantages to choosing the development of a native application, there are also drawbacks. The first is very clear, the application is linked to a particular operating system and a given version, which implies that client devices are linked. That is, the development of a web application, makes us gain in **portability**, since any desktop/mobile device has a web browser. Logically this problem could be solving with the development of different clients, but this **would make the** development and testing times of the different platforms more cost. It is true that there is also fragmentation in web browsers, but the use of frameworks helps to solve these problems

Performance, would be the point that could break the balance, but in the case at hand we must consider that access to the server of both a native and web application, would be comparable since reading the layouts of the ships, such as storing the information would go through the same communication API. Then the only performance problem, could be the visualization/handling of the layouts used however the use of HTML5 allows the SVG tag that is a container for SVG files, which allows viewing SVG files natively, with which the performance is optimal or the tasks that we want to perform, suffice it to say that broadly speaking what we need is a small "Google Maps" of the ship's cargo, where the usual operations are usually putting information associated with a point, zooming in on an area, checking listings.

As for the **deployment/update** of the different software versions there is no color since no market is required for publication, nor are there problems of re-compatibility with previous versions, since no essential type of data is stored on the client.

Another point to consider is the use of the **application in offline** mode, but already for some time with the appearance of HTML5, you can make use of an element called localStorage that has nothing to do with the old cookies, since its space limitation is no longer 4Kb but 10Mb, it has no expiration time and the stored information is not sent in each request, which saves heaviness with the server. Stored information is usually saved in JSON format, and in it we could save all the application information. A priori we do not know the volume of the data to be stored and we do not know if the 10Mb limit is sufficient (depends on the size of the geometries of the decks of the ships), but there are frameworks based on *this technology such as Indexed DB* [4] where not only does the space of the data to be stored increase, it also allows access through SQL and with all the characteristics of the relational model, such as referential integrity, atomicity, isolation and of course durability.

By using HTML5 and CSS3, software teams are developing and expanding web content and web apps with an aim to create well-defined and accurate web pages and web systems that can access on different devices, web browsers, and operating systems. HTML5 and CSS3 are the modern standard in the web development for business that develop and deploy online content and web applications.

By using both of these tools in web development one can optimize users web experience, providing a fast and simple user experience to users using mobile devices.

A mock-up of the presentation layer of the tool has been designed by NTNU to guide the different layouts and screens of the system. Mock-up can be accessed at:

<https://projects.invisionapp.com/share/THHNGHCGJZC#/screens/292891686>

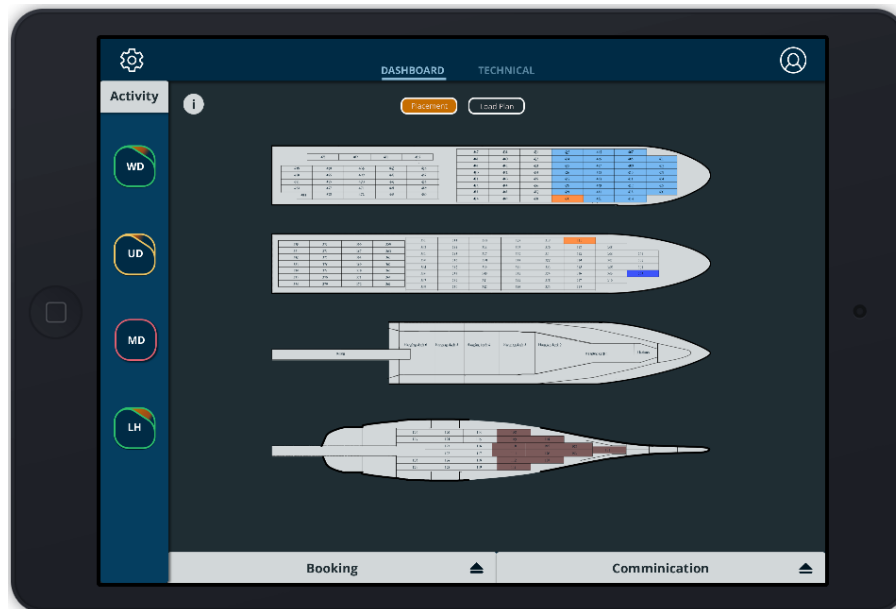


Figure 9 Mock-up of the presentation layer

6 Implementation

Main author of the chapter: Angel Priegue, CIM

The development of hazard mapping tool aims to build an easy to use application with visualization of risky 'hot' zones and different hazard types of cargo that can run on tablets or mobile phones on real life ship loading scenarios.

The software needs also to integrate feeds from sensors devoted to the thermal scan of cargo units on the deck and probabilities and knowledge from the fire hazard database where the calculation of cargo associated hazard probabilities is performed.

The methodological process that has been used to successfully overcome the implementation and deployment of the fire hazard matching tool is the classic Software Development Life Cycle[1].

The Software Development Life Cycle, is a set of steps used to create software applications. These steps divide the development process into tasks that can then be assigned, completed, and measured. It's typically divided into six steps: Analysis, Design, Implementation, Testing, Deployment and Maintenance. These are the core components recommended for all software development projects.

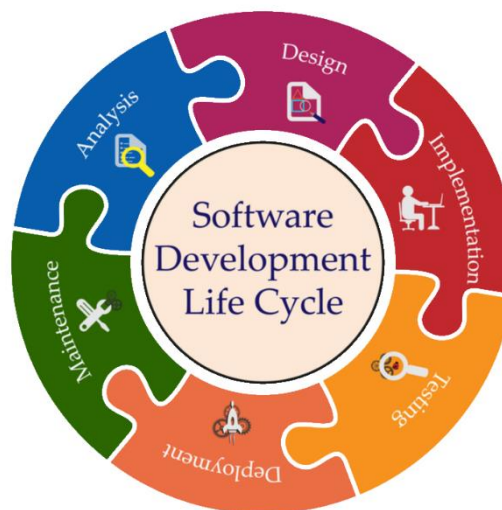


Figure 10 Software Development Life Cycle diagram

This Life Cycle allows a fine-grain analysis of each step of the process and helps to maximize efficiency at each stage and achieve the goals by identifying inefficiencies and higher costs and fixing them to run smoothly.

The scope of this report will only cover Implementation and Testing phases, can run concurrently with the Development phase, since developers need to fix errors that occur during testing.

7 Software Design

Main author of the chapter: Angel Priegue, CIM

In this section we will describe the architectural pattern used to design the different software modules, the design of the structure of the internal database and the materialization of the use-cases in sequence diagrams describing the operation of the fire hazard matching tool.

Since the early definition of task T08.5 during the definition of Action 8-A CIMNE has thought of HTML and JavaScript, the technologies used to develop web pages on the Internet, as the most suitable platforms to build the Fire Hazard Visualization Tool.

The characteristics of modern web applications (they now allow local storage and processing of data) and the responsive design to adapt to any screen size made them a good choice to implement a system that has to run both in a LCD screen/computer in the bridge and on hand-held devices in operational levels.

One of the big advantages of web technologies is that almost every device has today the possibility to show web content natively indifferently of the Operative System that is running on the device.

Of course, several other software frameworks could also be suitable for the implementation of the software but CIMNE has a wide experience in this technology and the selection of a previously known technology enables to speed up the development process.

Due to this previous baggage of experience in the development team, CIMNE has selected Microsoft ASP.NET framework to build the web components. CIMNE has more than 10 years of experience in this framework which is one of the most popular ones. Unlike in the past, the selection of Microsoft as the provider of technology does not imply a closed system in the matter of open-source since Microsoft has embraced open-source community in the last years.

7.1 Design and Architecture

The architecture of the fire hazard matching tool will be Model-View-Controller. In order to implement it in a software programming technology we will use MVC[8], a complete framework for building web applications and APIs using the Model-View-Controller design pattern.

The Model View Controller (MVC) architecture pattern separates an application into three main component groups: models, views, and controllers. This pattern allows you to achieve separation of interests. With this pattern, user requests are routed to a controller that is responsible for working with the model to perform user actions or retrieve query results. The controller chooses the view to display to the user and provides any model data that is required.

The following diagram shows the three main components and which one refers to the others:

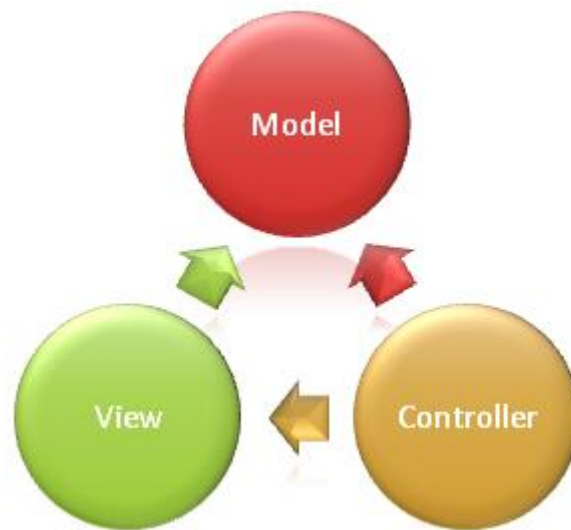


Figure 11 Software Development Life Cycle diagram

This outline of responsibilities makes it easier to scale applications because it's easier to code, debug, and test something (model, view, or controller) that has a single job. It is more difficult to update, test, and debug code that has dependencies spread across two or more of these three areas.

The view and controller depend on the model. However, the model does not depend on the view or the controller. This is one of the main advantages of separation. This separation allows you to build and test the model regardless of the visual presentation.

The model in an MVC application represents the state of the application and any business logic or operations that the application must perform. Business logic must be encapsulated in the model, along with any deployment logic to preserve the state of the application. Strongly-typed views typically use types designed to contain the data to display in that view.

Views are responsible for presenting content through the user interface. They use a view engine to insert code and HTML markup. There should be minimal logic between views and any logic in them must be related to content display.

Controllers are the components that control user interaction, work with the model, and ultimately select a view to render. In an MVC application, the view displays only information; the controller controls and responds to the interaction and data that users enter. In the MVC pattern, the controller is the initial entry point that is responsible for selecting which model types to work with and which views to render hence its name because it controls how the application responds to a particular request.

Then, in the following diagram and without losing sight of the idea of the MVC, the architecture at the module level used in the application is shown:

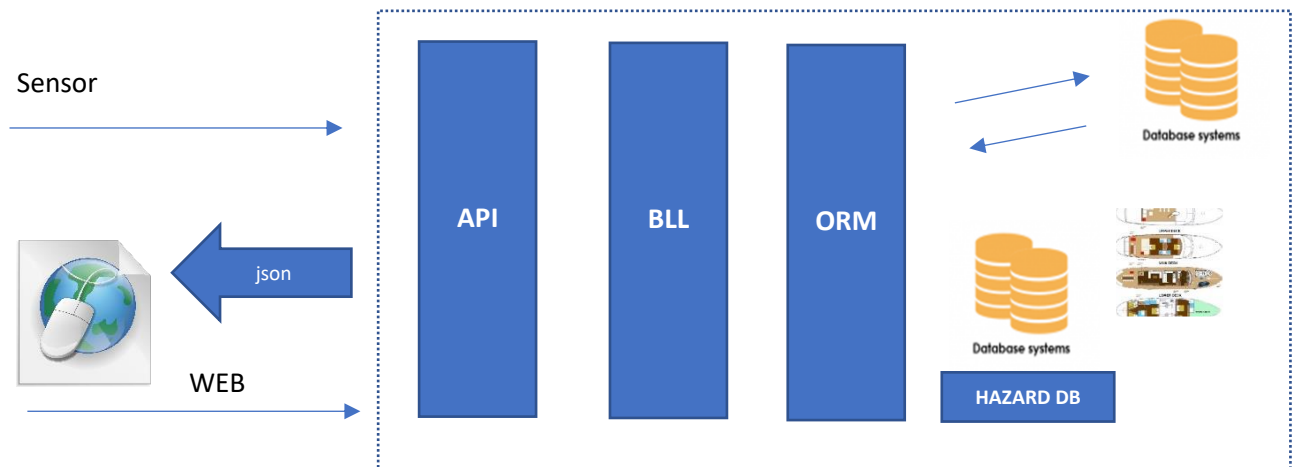


Figure 12 Design architecture

At the module level, it is composed of an API input layer, which is the gateway to interoperability with the different clients (browser / sensor).

This first layer is very thin because it simply collects parameters and immediately calls its upper layer, which is the business logic, the request is processed and returns the result in JSON, which is a standard widely used in the use of WEB APIs.

The next layer would be the Business Logic Layer (BLL) that interacts at the data model level with an Object Relational Mapping (ORM) library and where the bulk of the processing is carried out and the data persistence is carried out in the Database Management System (DBMS).

7.2 Data Design

The Fire hazard matching tool has different data entry streams:

Cargo Manifest initially the data is in a PDF, but needs to be exported to an electronic format to be able to be stored in a database, for this reason a unified entry of the data is required, since each shipping company has its own way of managing information about the load. To solve this problem, the original PDF file is exported to a CSV, with a specific format. The CSV format will be as follows:

TypeCargo	CargoID's	Length	Height	Weight Total	Weight Goods	Description	Num deck	Location

- **TypeCargo:**
 - Reefer unit Value
 - Conventional vehicle Bus
 - Conventional vehicle Truck
 - Special vehicle RVs
 - Conventional vehicle Car
 - Special vehicle Tractor
 - New energy carrier Electrical vehicle
 - Special vehicle trailer
 - Dangerous goods Flammable solid
 - Dangerous goods Flammable liquid

- Dangerous goods Miscellaneous dangerous substances and articles
 - Dangerous goods Corrosive substances
 - Dangerous goods Explosive
 - Dangerous goods Gas
 - Dangerous goods Undeclared DG
- **CargoID's:** BookingID and number registration.
 - **Length:** Vehicle length in meters.
 - **Height:** Vehicle height in meters.
 - **Total Weight:** Weight in tons (vehicle + cargo).
 - **Goods Weight:** Weight in tons (cargo).
 - **Description:** This is a free text field to put a more detailed description of the load and it is optional.
 - **Num deck:** Reference ID of the deck.
 - **Location:** Coordinates (x1,y1,x2,y2,x3,y3,x4,y4) of the location of the cargo unit with respect to the SVG format deck plan (1792x500).

Decks, the original format of the covers are .DWG files (AUTOCAD), they require an export to a type of object that can be vector, that is, it supports Zoom In and Zoom Out without losing quality and that can be viewed by browser standard, it is because the initial cover files are converted to SVG format.

Sensors, the information from the sensors, can be dumped into the system by accessing an API provided by the Fire hazard matching tool.

Cargo Fire Hazard DB is a database where the hazardous relationship of a possible fire is stored according to the type of vehicle and goods transported.

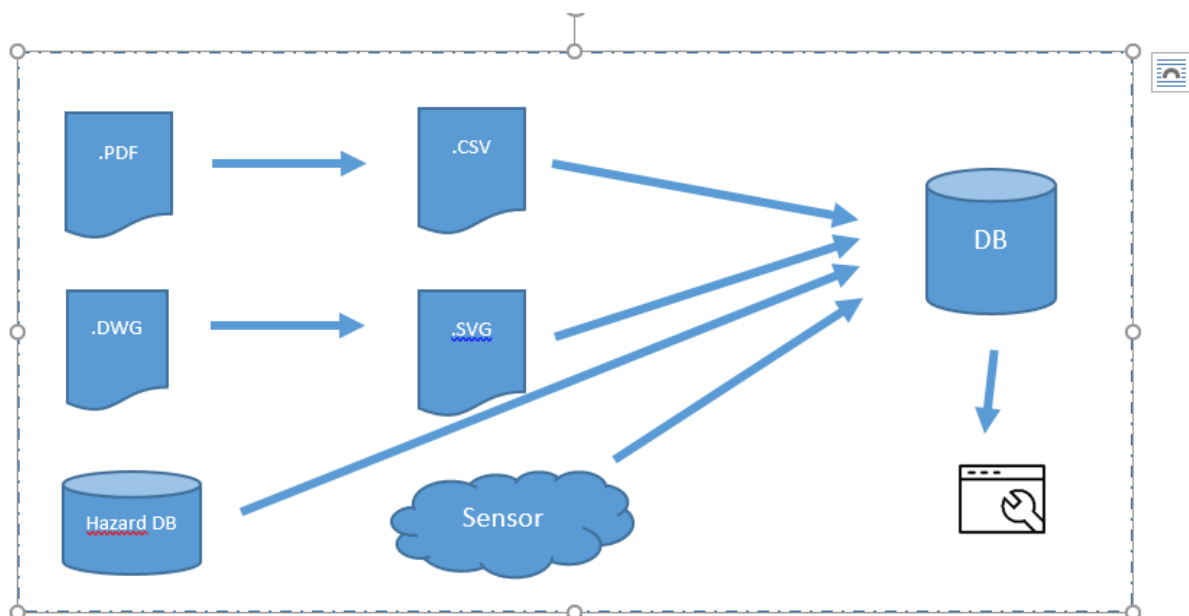


Figure 13 Data design

Finally, all these data input streams will be stored in a central database, here we can see the ER model of the DB.

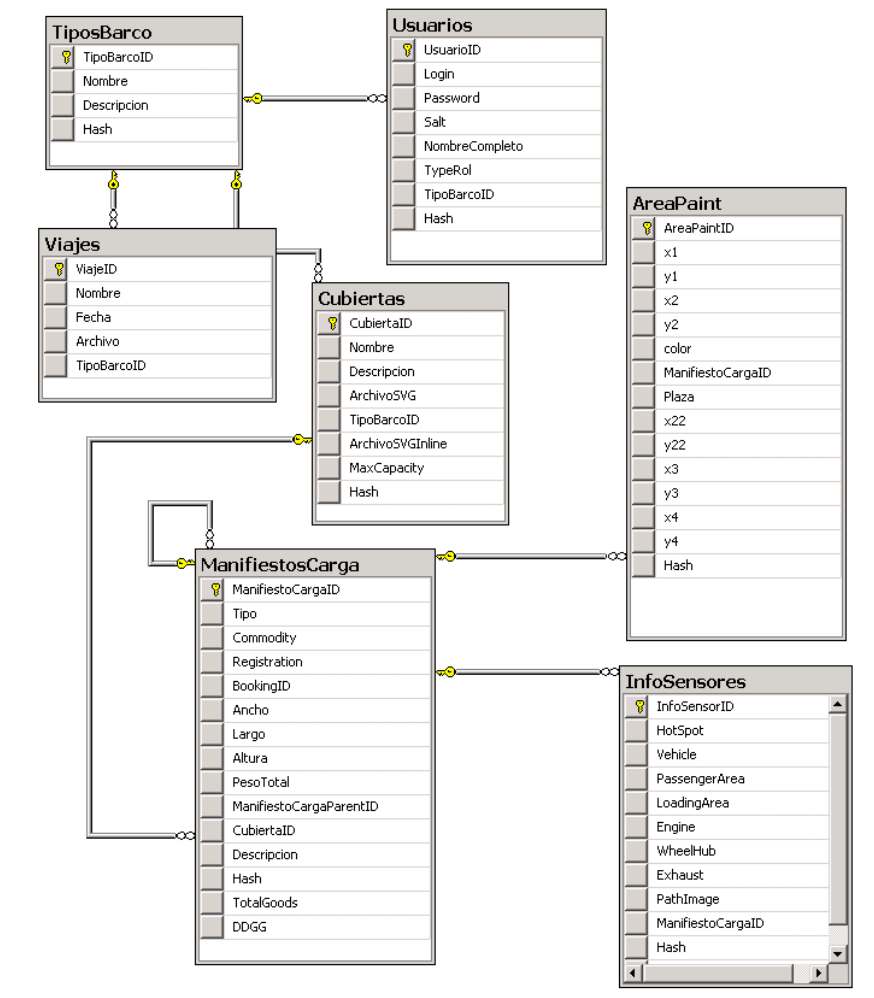


Figure 14 SQL Server Database diagram

Communication between the WEB browser and the central database will be done through a communication API, which will expose the data in JSON format. This will make deserialization on the client possible very quickly, since Javascript allows it natively. Having this communication API makes it easier, if necessary, to be able to work offline, since the data can be stored locally using Index DB[7]

7.3 Sequence Diagrams

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

UploadLoadPlan

The following sequence diagram corresponds to the UploadLoadPlan use case, related to the initial data load derived from the cargo manifest. As you can see in the diagram, this initial load is done by the ship owner, and the input format is a CSV file.

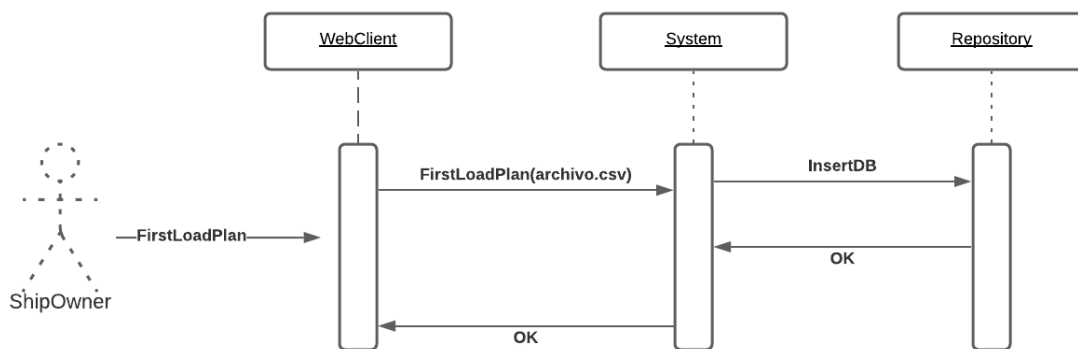


Figure 15 Sequence diagram of the cargo loading process

SetInfoSensor

The following sequence diagram corresponds to the SetInfoSensor use case which is related to the initial load of data from the sensors relative to the initial cargo.

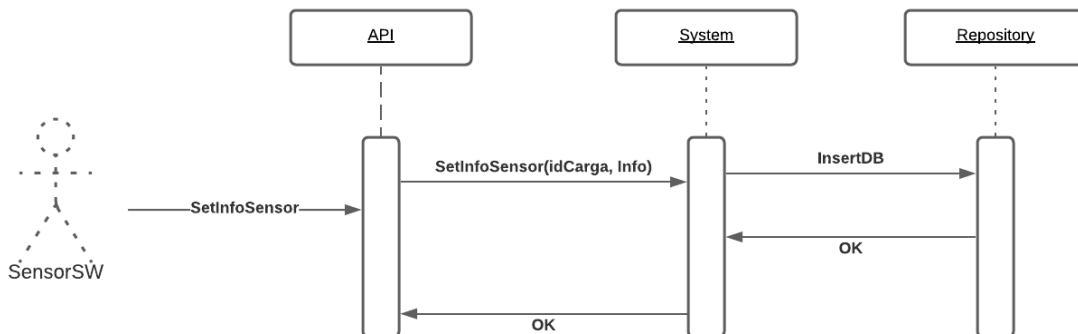


Figure 16 Sequence diagram of the set info sensor

Login

The following sequence diagram corresponds to the Login use case, to the Web Client platform.

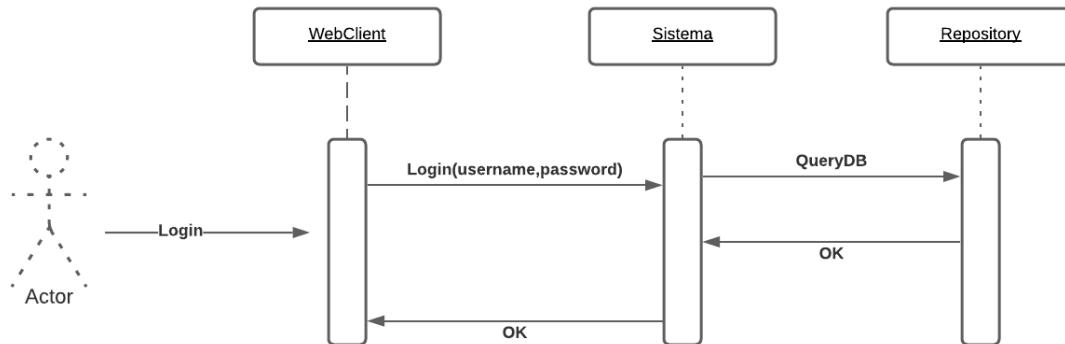


Figure 17 Sequence diagram of the Login

ForGotPassword

The following sequence diagram corresponds to the **ForGotPassword** use case, to the Web Client platform. As its name indicates, it helps the user to recover their password in case they have forgotten by sending an email

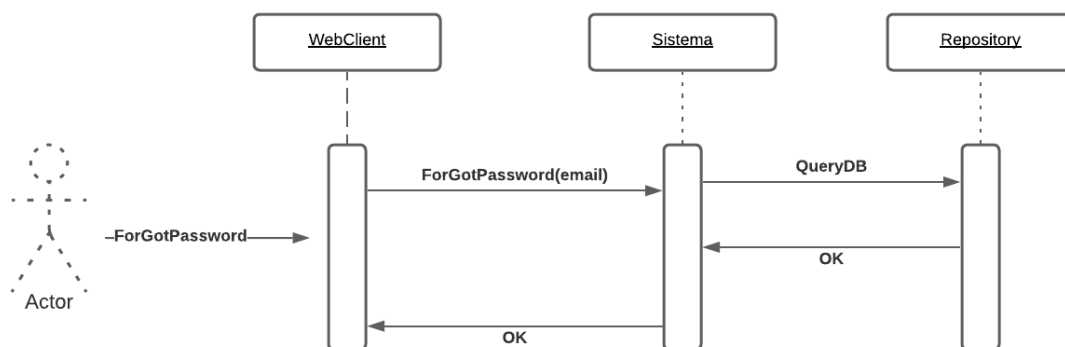


Figure 18 Sequence diagram of the forgotpassword

View Deck / Zoom In / Drag Deck / View Cargo Info

The following sequence diagram corresponds to the use cases of visualization of the decks, information associated with the cargo and the zoom and drag of the deck designs. As you can see, once the data is downloaded from the server, all the information is managed on the client, so this allows use in offline mode.

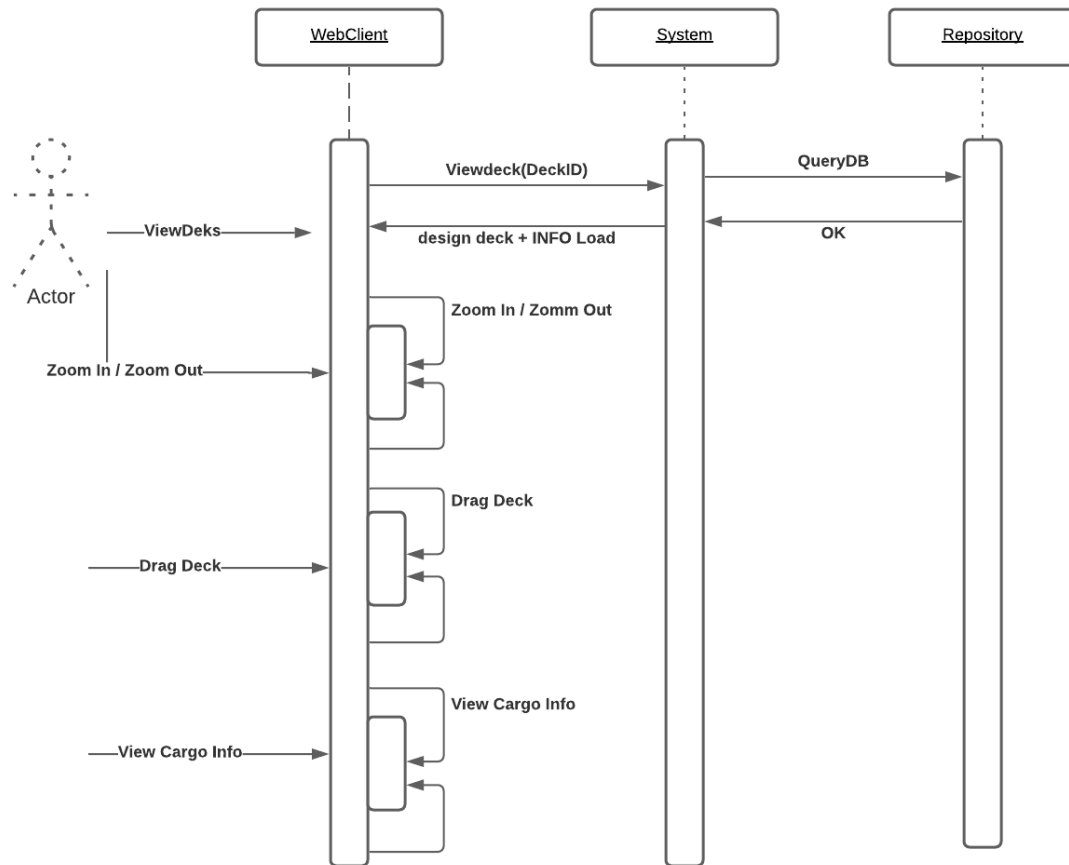


Figure 19 Sequence diagram of the Sequence diagram of the View Deck / Zoom In / Drag Deck / View Cargo Info

Dashboard The following sequence diagram corresponds to the dashboard use case, which allows you to see the summary of the total cargo according to their type, and a fire risk assessment graph.

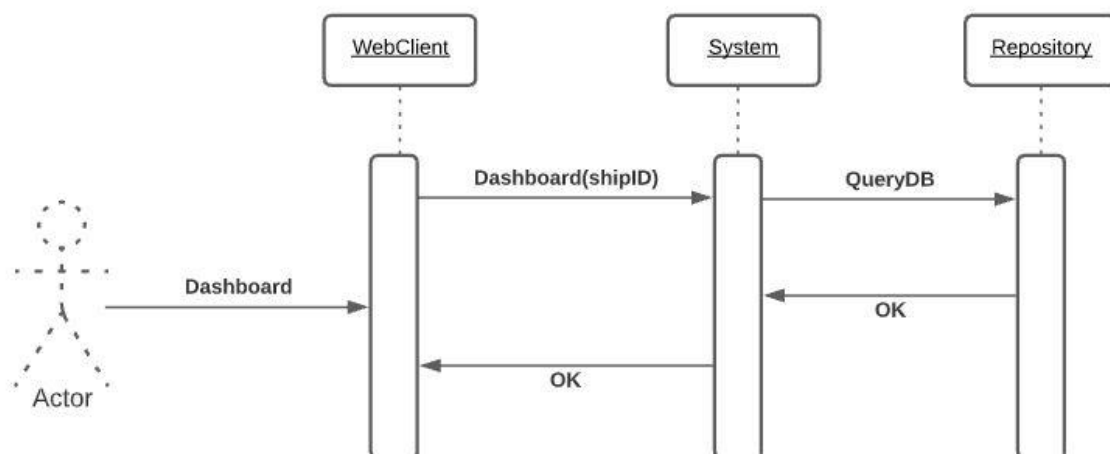


Figure 20 Sequence diagram of dashboard

SearchCargo(NumRegister)

The following sequence diagram corresponds to the Searchcargo use case, which allows the user to locate a cargo located on the ship and thus be able to consult its associated information more quickly. The search input parameter would be the NumRegister.

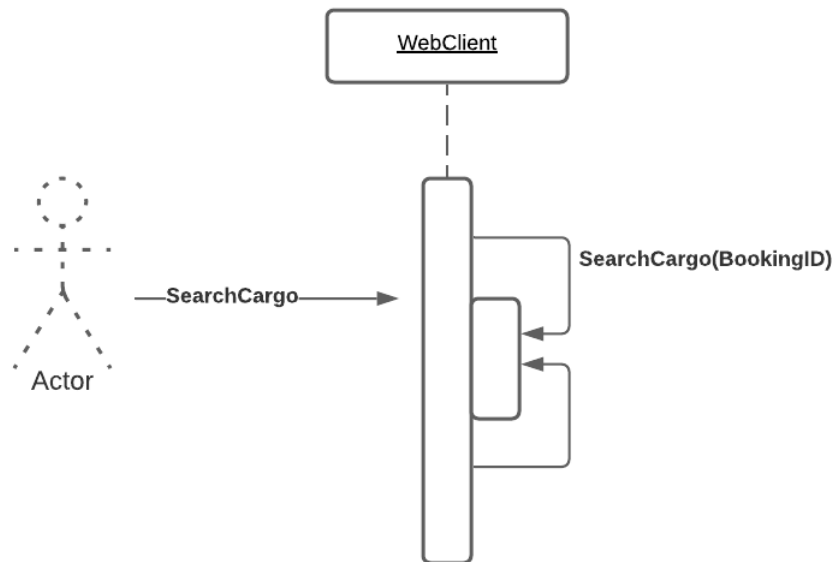


Figure 21 Sequence diagram of SearchCargo

8 Software Implementation

Main author of the chapter: Angel Priegue, CIM

In this section we will describe the technologies that will be used to implement the main elements of the fire hazard matching tool.

8.1 Web Client implementation

The web client for the fire hazard matching tool will use the Open Web Platform, which is the collection of open technologies which enable the Web.

Being basically a visualization tool, it focuses a lot on the capabilities that HTML5 offers to be able to visualize embedded SVG-type objects in a `</canvas>` tag without the need for any type of plug-in. This allows you to have the basic functionalities of zooming or moving the design on the canvas.

For the client side we also use one of the most popular frameworks today for the design of Web applications, such as JQUERY, which has the advantage of being compatible with different browsers and versions, for us it is an important point since it makes us independent from the version of Javascript that the Tablet has or the Javascript version of the PC. In addition, it allows us to access the API through AJAX and therefore makes it much easier for us to deserialize JSON objects.

8.1.1 Web User Interface

Access to the main screen is done through a login section that allows the user through their credentials (login and password) to access its content. Remember password option is allowed to make it easier to use and typical remember password functionality.

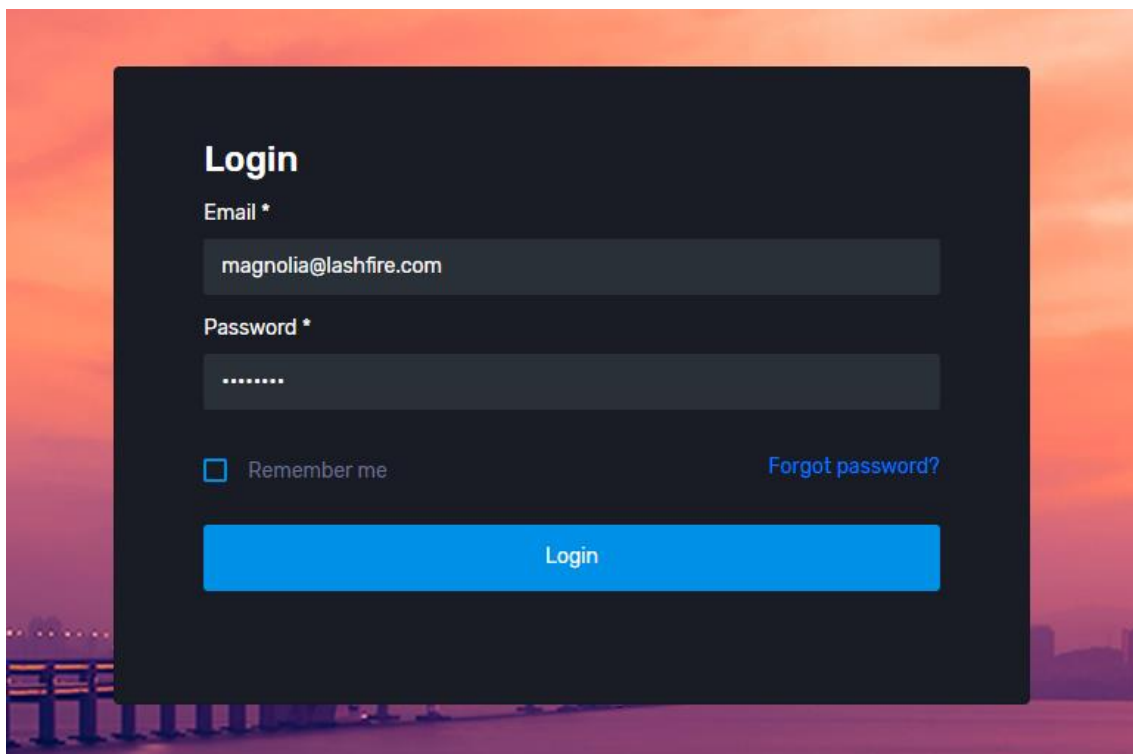


Figure 22 Login screen

The next main screen of the application would be the Summary of cargo/risk dashboard screen, where once logged in with his credentials he could see the menu in (left menu). At the top you can also see both the name of the ship to which the user is associated, and at the top right the name of the user who has logged into the application. By clicking on the user's name you can logout.

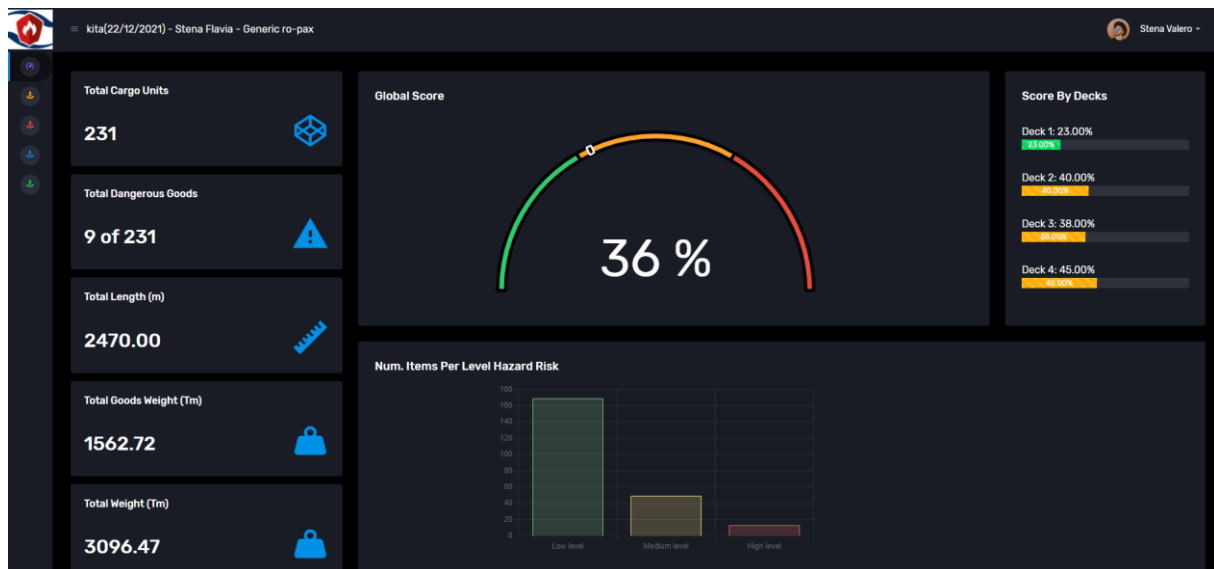


Figure 23 Dashboard screen

From dashboard screen the user can see a summary of the ship's cargo and an overall fire risk score. The information in this dashboard is displayed in a very visual way so that the alert level can be quickly seen according to the cargo of the ship. Specifically, the data that can be viewed are the following:

- Total No of bookings / Total No. of chassis/export cars / Total Length (m), Total goods (Kg), Total Weight (Kg).
- Global score of fire risk / graph of the total load units according to risk level.
- Detailed summary according to vehicle type (table)

The next main screen would be the work screen with the covers that is accessed by clicking on the anchor icon. where you can mainly see the vector diagram of the deck selected, being able to Zoom In & Zoom Out, without losing quality, since the possibility that HTML5 offers to embed SVG objects is taken advantage of.

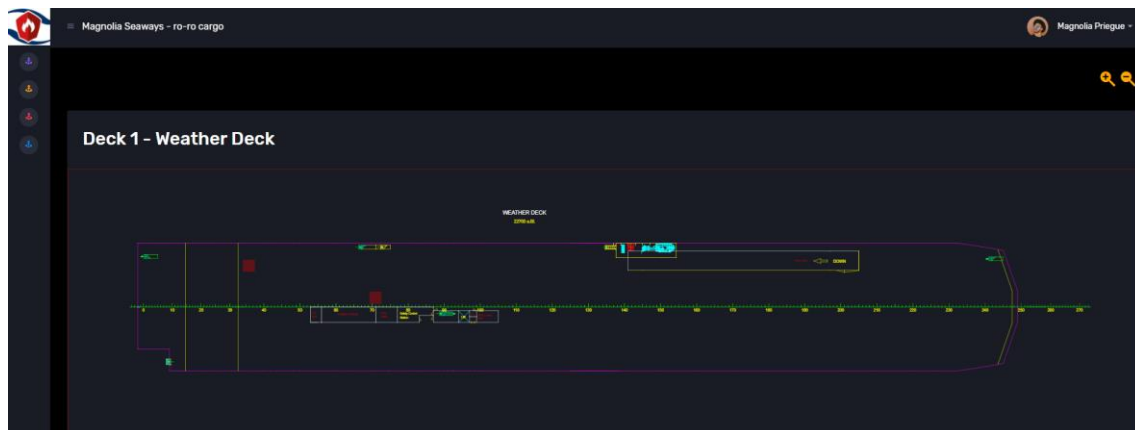


Figure 24 Deck view

In more detail, you can see the load on the deck, how it is distributed and the associated fire risk, using a color map. For more information about color map you can see IR8.10 section 4.2 Hazard Database.

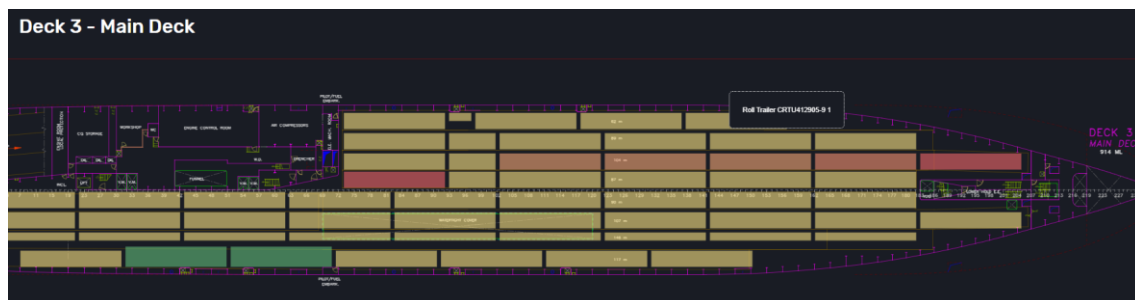


Figure 25 View of cargo on a deck

By clicking on a specific load we can see its associated information, the information to be displayed comes from 3 sources of information: cargo manifest, load sensing and observations indicating the reason why that load has been catalogued with a type determined risk.

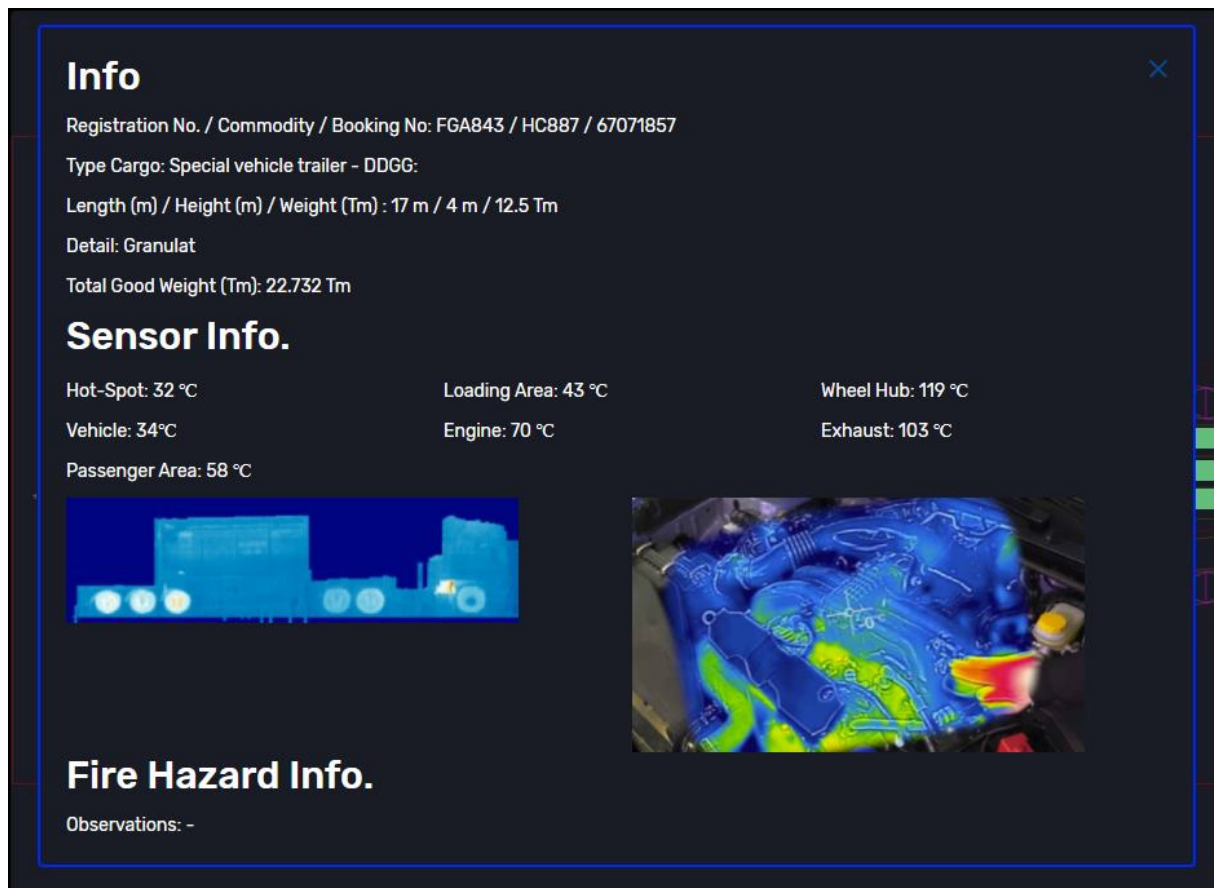


Figure 26 Detail of the cargo file.

8.2 Back-end Services implementation

Backend services handle the ‘behind-the-scenes’ functionality of the web application. It connects the web to a database, manages user connections, and powers the web application itself. Backend services work in tandem with the front-end web client to deliver the functionality to the end user.

The language that will be used to send messages between the back-end services and front-end web client of the fire hazard matching tool will be JSON.

JSON (JavaScript Object Notation)[5] is a lightweight data-interchange format. It is easy for humans to read and write and it is easy for machines to parse and generate. JSON is based on a subset of the JavaScript Programming Language but is a text format that is completely language independent which make it an ideal data-interchange language.

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language. When exchanging data between a browser and a server, the data can only be text.

JSON allows to convert any JavaScript object into JSON text and send it to the server. We can also convert any JSON text received from the server into objects. This way we can work with the data as objects, with no complicated parsing and translations.

Next, a simplified example where the separation between layers and the responsibility of each one of them is very clearly seen. We have a GetInfoAll input ENDPOINT, which makes a call to the MODEL, and parses the response to JSON format.

```
[WebMethod()]
public static String GetInfoAll()
{
    try
    {
        // ACCESS TO Model
        var cubiertaRepository = new LashFireBLL.Repository.DB.CubiertaRepositoryDB();
        listaCubiertasPorBarco = cubiertaRepository.GetCubiertas(tipoBarco);

        // Create OBJECT Return
        var response = new LashFireBLL.DTO.RespuestaWS<List<Object>>>();
        response.responseStatus = 0;
        response.message = Newtonsoft.Json.JsonConvert.SerializeObject(listaCubiertasPorBarco);
        response.error = String.Empty;

        // RETURN JSON Format
        return Newtonsoft.Json.JsonConvert.SerializeObject(response);
    }
    catch (Exception ex)
    {
        // Create OBJECT Return
        var response = new LashFireBLL.DTO.RespuestaWS<List<Object>>>();
        response.responseStatus = -1;
        response.message = String.Empty;
        response.error = "ERROR"

        // RETURN JSON Format
        return Newtonsoft.Json.JsonConvert.SerializeObject(response);
    }
}
```

8.3 APIs

A good way to maintain cross-layer responsibility for the architectural model used (MVC) and provide interoperability with other external software is to provide a webAPI communication.

Web APIs are the defined interfaces through which interactions happen between an application and other external applications that use its assets.

Certainly, the WebAPI of communication can be done in any server language, in our case they are made in ASP.NET, since interoperability is achieved through the use of JavaScript Object Notation (JSON) which is a lightweight and easy to understand format.

In the spirit of making a good documentation of the API created and an easy way to understand and test by third parties the use of the API, it has been used (Swagger [1]) which allows in a fast way and always with the same format, to see all the methods offered.

In the figure below we show, according to the typical swagger interface, the methods exposed in our API.

<https://lashfire.cimne.com/external/swagger/ui/index>

API_KEY = "25b2601e-76f2-418d-93eb-1d57bc981fe1";

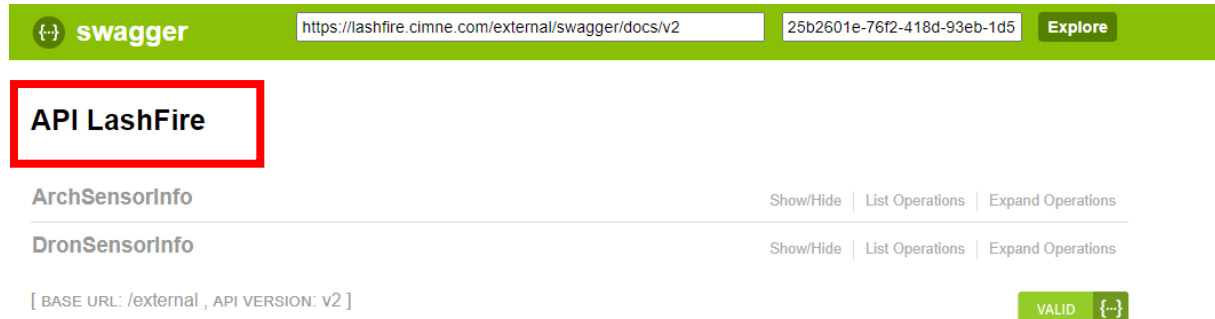


Figure 27 Swagger UI – List of methods API

ArchSensorInfo : This endpoint is used by the VHD arch sensor to have interoperability with the application. It has method POST that allows to dump the information of the sensorics given a load identifier (registrationID)

Parameters:

registrationID: *Identifier of the cargo unit*

json (HotSpot, Vehicle, PassengerArea, LoadingArea, Engine, WheelHub, Exhaust): *Temperatures obtained by the VHD of different key points of the vehicle.*

urlFoto: *URL of the image obtained by the VHD*

/ Example ArchSensorInfo */*

POST

<http://lashfire.cimne.com/external/api/ArchSensorInfo>

Params by BODY

```
{
  "registrationID": "FGA843",
  "json": "{
    \"HotSpot\": 12,
    \"Vehicle\": 23,
    \"PassengerArea\": 34,
    \"LoadingArea\": 45,
    \"Engine\": 56,
    \"WheelHub\": 67,
    \"Exhaust\": 78
  }",
  "urlFoto": "http://lashfire.cimne.com/data/Stena/Sensor.PNG"
}
```

DronSensorInfo: This endpoint is used by the AGV drones to have interoperability with the application. It has method POST that allows to dump the images obtained given a load identifier(registrationID)

Parameters:

registrationID: *Identifier of the cargo unit*

urlFoto: *URL of the image obtained by the AGV*

/ Example DronSensorInfo */*

POST


<http://lashfire.cimne.com/external/api/DronSensorInfo>

Params by BODY

```
{
  "registrationID": "FGA843",
  "urlFoto": "https://atyges.es/wp-content/uploads/2020/02/anafi_thermal_1_atyges.jpg"
}
```

This communication API is mostly consumed by AJAX clients, with the exception of the Sensor package that acts as a gateway for sensor data. Obviously, the application has many more use cases, but not all use cases are exposed by the API.

Here's an example capture of what the methods look like, with their input parameters and output format.

 swagger
<https://lashfire.cimne.com/external/swagger/docs/v2>
25b2601e-76f2-418d-93eb-1d5 [Explore](#)

API LashFire

ArchSensorInfo Show/Hide | List Operations | Expand Operations

POST </api/ArchSensorInfo> Save info arch sensor into RegistrationID

Response Class (Status 200)
OK

Model | Example Value

```
{}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
data	<pre>{ "registrationID": "FGA843", "json": "{ \"HotSpot\": 12, \"Vehicle\": 23, \"PassengerArea\": 34, \"LoadingArea\": 45, \"Engine\": 56, \"WheelHub\": 67, \"Exhaust\": 78 }", "urlFoto": "http://lashfire.cimne.com/data/Stena/Sensor.PNG" }</pre>	Info arch sensor to send.	body	Model Example Value
				<pre>{ "registrationID": "string", "json": "string", "urlFoto": "string" }</pre>

Parameter content type: application/json

Try it out!

Figure 28 Example </external/api/DronSensorInfo>

9 Testing

Main author of the chapter: Angel Priegue, CIM

Software testing defines the activities to check whether the actual results match the expected results and to ensure that the software system is defect free. Testing involves execution of a software components or system components to evaluate one or more functions of the tool. Tests will be devoted to identify errors, gaps or missing requirements in contrary to the actual requirements.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding failures and verifying that the software product is fit for use.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments
- achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically attempts to execute a program or application with the intent of finding failures due to software faults. The job of testing is an iterative process as when one fault is fixed, it can illuminate other failures due to deeper faults, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.

9.1 Unit Tests

Software testing is the process where individual units/components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

Unit tests are typically automated tests written and run by software developers to ensure that a section of an application, known as the "unit", meets its design and behaves as intended.

In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. By writing tests first for the smallest testable units, then the compound behaviours between those, one can build up comprehensive tests for complex applications.

To isolate issues that may arise, each test case should be tested independently. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation.

During development, a software developer may code criteria, or results that are known to be good, into the test to verify the unit's correctness. During test case execution, frameworks log tests that fail any criterion and report them in a summary.

Writing and maintaining unit tests can be made faster by using parameterized tests, which allow the execution of one test multiple times with different input sets, thus reducing test code duplication. Parameterized tests are supported by testing softwares like JUnit[6] or XUnit. Suitable parameters for the unit tests may be supplied manually or in some cases are automatically generated by the test framework.

To perform unit tests, we have used MSTest for the business layer classes, since it is perfectly integrated into our IDE, and it is not necessary to leave it to run the tests.

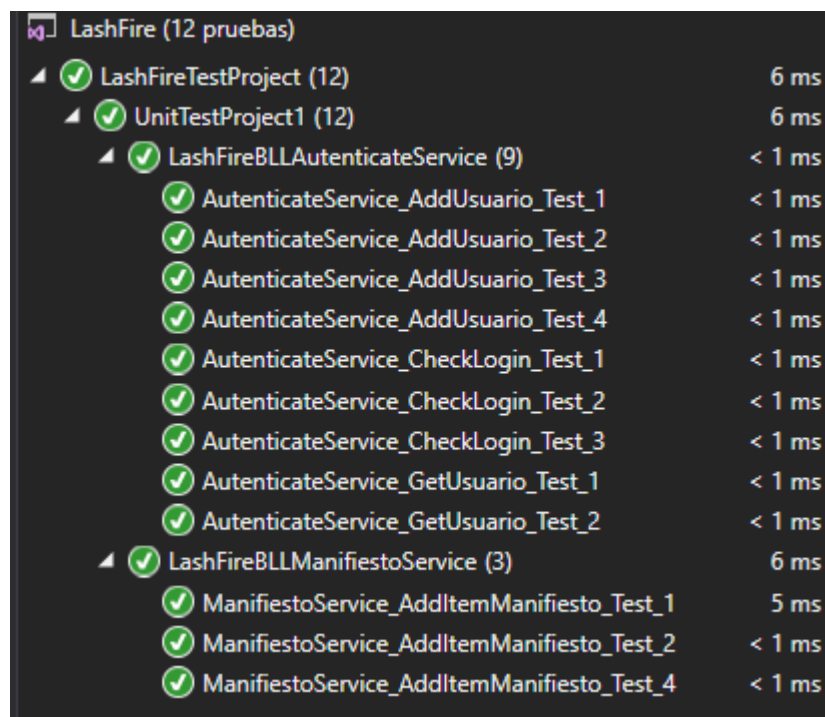


Figure 29 Fragment Unit Test from Visual Studio

At the API level, however, for testing we have used the facility provided (SWAGGER) since it has a "Try it Out" button for each of the methods.

AllInfo Show/Hide List Operations Expand Operations

GET /api/AllInfo

Response Class (Status 200)
string

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
tipoBarco	<input type="text" value="2"/>		query	integer

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:61628/api/AllInfo?tipoBarco=2'
```

Request URL

```
http://localhost:61628/api/AllInfo?tipoBarco=2
```

Response Body

```
{
  "CubiertoID": 2,
  "Nombre": "Deck 1",
  "Descripcion": "Weather Deck",
  "ArchivoSVG": "data/MagnoliaSeaways/WeatherDeck-Mode"
}
```

Response Code

```
200
```

Response Headers

```
{
  "cache-control": "no-cache",
  "content-length": "1304343",
  "content-type": "application/json; charset=utf-8",
  "date": "Wed, 23 Jun 2021 15:53:59 GMT",
  "expires": "-1",
  "pragma": "no-cache",
  "server": "Microsoft-IIS/10.0",
  "x-aspnet-version": "4.0.30319",
  "x-powered-by": "ASP.NET",
  "x-sourcefiles": "?UTF-8?B?QzpcUHJveWVjdG9zXExhc2hGaXJlXExhc2hGaXJlXEFQSVxhcG1cQWxsS5w5mbw==?="
}
```

Figure 30 Example response from /api/AllInfo

9.2 System Integration Tests

In software development, individual units are often combined as a group. The purpose of system integration testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing

System integration testing (SIT) involves the overall testing of a complete system of many subsystem components or elements. The system under test may be composed of hardware, or software, or hardware with embedded software, or hardware/software with human-in-the-loop testing.

SIT consists, initially, of the "process of assembling the constituent parts of a system in a logical, cost-effective way, comprehensively checking system execution (all nominal & exceptional paths), and including a full functional check-out." Following integration, system test is a process of "verifying that the system meets its requirements, and validating that the system performs in accordance with the customer or user expectations."

In the context of software systems and software engineering, system integration testing is a testing process that exercises a software system's coexistence with others. With multiple integrated systems, assuming that each have already passed system testing, SIT proceeds to test their required interactions.

Finally, we will have Acceptance System Tests where the system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

To carry out the integration tests, we have used Selenium [2] which is a software test environment oriented to web applications, it has the advantage that it does not require code to automate the tests but it is easy to use by making a short video of the actions that the end user would perform.

10 Conclusion

Main author of the chapter: Angel Priegue, CIM

This report describes the different steps in the software development process of the Fire Hazard Matching tool, developed as the main result of T08.5 in the context of action 8-A of WP8.

Following its requirements, the software enables the visualization and identification of hazards associated to each zone of the ship according to the characteristics of the cargo units located there.

The tool is able to evaluate fire risk associated to all cargo units of a given ship loading configuration using an easy-to-use graphical interface that can be run both in computers and hand-held devices (mobile phones or tablets).

The tool is part of the LASH FIRE action 8-A system of systems approach with open interfaces to other systems, and in LASH FIRE the interoperability with the other WPs and a future data exchange with other systems.

Some of the visual UI requirements of the tool are not implemented in the Fire Hazard Matching tool in the scope of task T08.5 as they will be implemented in the final result of the action, the Visualization Aid in task T08.6 (Ref. D08.4 Stowage planning optimization and visualization aid).

This report only defines the description of software technologies that have been used for the implementation and testing of the fire hazard matching tool.

Final details and real working screens of the fire hazard matching tool is presented in in D08.2 Fire hazard mapping visualization tool with fire hazard matching integrated.

11 References

[1] IMO Publications and Documents. (n.d.). *Annex - Guideline on Software Quality Assurance and Human Centred-Design for E-Navigation*.

https://www.imorules.com/MSCCIRC_1512_ANN.html

[2] Mozilla Web Docs. (n.d.). *HTML5*.

<https://developer.mozilla.org/es/docs/HTML/HTML5>

[3] Imperial College London. (n.d.). *The CIA principle*.

<https://www.doc.ic.ac.uk/~ajs300/security/CIA.htm>

[4] Index DB

https://developer.mozilla.org/es/docs/Web/API/IndexedDB_API

[5] OpenSDLC. (n.d.). *Systems Engineering and Software Development Life Cycle Framework*.

http://opensdlc.org/mediawiki/index.php/Main_Page

[6] Microsoft. (n.d.). *ASP.NET MVC Pattern*.

<https://dotnet.microsoft.com/apps/aspnet/mvc>

[7] Mozilla Web Docs. (n.d.). *HTML5*.

<https://developer.mozilla.org/es/docs/HTML/HTML5>

[8] Mozilla Web Docs. (n.d.). *CSS: Cascading Style Sheets*.

<https://developer.mozilla.org/en-US/docs/Web/CSS>

[9] JSON.org. (n.d.). *Introducing JSON*.

<https://www.json.org/json-en.html>

[10] JUnit.org. (n.d.). *JUnit5*.

<https://junit.org/junit5/>

[11] Index DB

https://developer.mozilla.org/es/docs/Web/API/IndexedDB_API

12 Indexes

12.1 Index of tables

Table 1. Typical General Cargo Units carried by a ship	15
Table 2. Typical Dangerous Cargo Units carried by a ship	15
Table 3. Risk Levels General Cargo	16
Table 4. Risk Levels Dangerous Cargo	17
Table 5. Types of users	22
Table 6. Use Case 01.....	24
Table 7. Use Case 02.....	24
Table 8. Use Case 03.....	25
Table 9. Use Case 04.....	25
Table 10. Use Case 05.....	25
Table 11. Use Case 06.....	26
Table 12. Use Case 07.....	27
Table 13. Use Case 08.....	27
Table 14. Use Case 09.....	27
Table 15. Use Case 10.....	28
Table 16. Use Case 11.....	28

12.2 Index of figures

Figure 1 SW/HW components of the risk-based load planning tool.....	9
Figure 2 Distribution of modules of the risk-based load planning tool.....	12
Figure 3 Cargo fire hazard database diagram	13
Figure 4. Hazard Levels.....	14
Figure 5. Normal user use cases.....	23
Figure 6. Administrator user use cases	23
Figure 7. Sensor user use cases	24
Figure 8. Data Layer.....	32
Figure 9 Mock-up of the presentation layer	34
Figure 10 Software Development Life Cycle diagram	35
Figure 11 Software Development Life Cycle diagram	37
Figure 12 Design architecture	38
Figure 13 Data design.....	39
Figure 14 SQL Server Database diagram	40
Figure 15 Sequence diagram of the cargo loading process.....	41
Figure 16 Sequence diagram of the set info sensor.....	41
Figure 17 Sequence diagram of the Login	42
Figure 18 Sequence diagram of the forgotpassword	42
Figure 19 Sequence diagram of the Sequence diagram of the View Deck / Zoom In / Drag Deck / View Cargo Info	43
Figure 20 Sequence diagram of dashboard.....	43
Figure 21 Sequence diagram of SearchCargo.....	44
Figure 22 Login screen	45
Figure 23 Dashboard screen	46
Figure 24 Deck view.....	47
Figure 25 View of cargo on a deck.....	47
Figure 26 Detail of the cargo file	48
Figure 27 Swagger UI – List of methods API.....	50

<i>Figure 28 Example /external/api/DronSensorInfo</i>	<i>51</i>
<i>Figure 29 Fragment Unit Test from Visual Studio</i>	<i>53</i>
<i>Figure 30 Example response from /api/AllInfo.....</i>	<i>54</i>