

MILESTONE 2: NARANAIR



POBPL5: INFORMATIC ENGINEERING DEGREE 3º COURSE

ABSTRACT

This document is the conclusive report of the NaranAir project developed at the University of Mondragon. The main aim of the project is to create a web based simulation of an airport. This involves that the project will be contained in two main segments, the simulation engine and the webpage itself. The link between both of the components is the common database that contains and manages all the information required by the system.

The simulator goal is to manage the planes landing and ground management whilst the webpage aimed at offering a functional and user friendly interface for different users of the resources of the airport (passengers, airlines, airport controllers, etc.). All this was developed using project management tools and strategies both in what involves software development and team administration.

To sum up, actual achievements and misses must be highlighted. The webpage managed to offer a reliable base for a fully developed system and the simulator succeeded in providing a flexible plan guidance reproducer as similar as possible to reality.

RESUMEN

El siguiente documento es el informe concluyente del proyecto NaranAir desarrollado en la Universidad de Mondragón. El objetivo principal del proyecto es crear una simulación Web de un aeropuerto. Esto supone que el proyecto se verá envuelto en dos segmentos principales, la simulación y la página Web. La conexión entre ambos segmentos se lleva a cabo mediante una base de datos común, conteniendo esta toda la información requerida por el sistema.

Respecto a la simulación del sistema, la meta principal es gestionar el tráfico aéreo en el interior del aeropuerto, es decir, dirigir el aterrizaje y despegue de los aviones y el tráfico de estos sobre el suelo. Por otro lado, la página Web tiene como objetivo ofrecer una interfaz funcional y de fácil manejo para diferentes usuarios de los recursos del aeropuerto (pasajeros, aerolíneas, controladores aéreos, etc.). Todo esto ha sido desarrollado con la ayuda de herramientas y estrategias de gestión de proyectos tanto en lo que respecta al desarrollo de software como a la administración del equipo.

Para resumir, los logros reales y los fallos deben ser destacados. La página Web logró ofrecer una base confiable para un sistema completamente desarrollado y el simulador logró un reproductor de orientación de plan flexible lo más similar posible a la realidad.

LABURPENA

Hurrengo dokumentua, Mondragon Unibertsitatean garatutako NaranAir proiektuaren erabateko txostena da. Proiektu honen, helburu nagusia aireportu baten Web simulazioa sortzea da. Hori dela eta, proiektua, bi segmentu garrantzitsu eta ezberdinetan banatuta dago, alde batetik aireportuaren simulazioa eta bestetik, Web orrialdea. Bien arteko konexioa ohiko datu base baten bitartez burutzen da, sistema osoaren informazioa gordeaz eta kudeatuz.

Simulazioari dagokionez, bere asmo nagusia aireportu barneko aire trafikoa gestionatzea da, hau da, hegazkinen lurreratze, aireratze eta lurreko mugimenduak. Beste alde batetik, Web orrialdearen xedea aireportuaren errekurtsoen erabiltzaileentzat (bidaiariak, airebideak, kontrolatzaileak, etc.) interfaze funtzional eta erabilgarri bat eskaintzea da. Guzti hau, proiektuen kudeaketa tresna eta estrategia ezberdinen laguntzaren bidez burutu da, softwarearen garapenean zein taldearen administrazioan eraginda.

Laburbiltzeko, lortze errealak eta huts edo okerrak nabarmendu behar dira. Web orrialdeak oinarri fidagarri bat lortu zuen guztiz garatutako sistema baten lan egiteko eta simulatzailearen bidez, aldiz, errealitatera hurbiltzen den hegaldi planeatzaile bat sortu zen.



INDEX

ABSTRACT	2
RESUMEN	2
LABURPENA	3
1. INTRODUCTION	7
2. PROBLEM DESCRIPTION	8
3. RESEARCH	9
3.1. NECESSITIES	9
3.2. CURRENT STATE	9
4. PROJECT ORGANISATION	10
4.1. OBJECTIVES	10
4.2. TASKS	11
4.3. ROLES	13
5. DEVELOPMENT STRATEGY	14
5.1. REPOSITORY STRATEGY	15
5.2. BRANCH STRATEGY	16
5.3. INTEGRATION STRATEGY	18
5.4. CODING STANDARD	20
5.5. TESTING	21
5.6. DOCUMENTATION	22
6. SIMULATOR	22
6.1. AIRPORT MANAGE LOGISTIC	24
6.2. SYNCHRONIZATION	24
7. WEB APPLICATION	26
7.1. BACKEND	26
7.1.1. TOMCAT SERVLETS	26
7.1.2. STRUTS2	27
7.1.3. ASP.NET	27
7.1.4. DJANGO	27
7.1.5. RAILS	28
7.1.6. STRUCTURE	28
7.1.7. DEPLOYMENT	31
7.2. FRONTEND	32
7.2.1. JQUERY	32
7.2.2. BOOTSTRAP	32
7.2.3. SOCKETIO	33
7.2.4. VISUAL LIBRARIES	34
7.2.5. HTML5	35
7.2.6. CSS3	35
8. DATABASE	35
8.1. ENTITY RELATIONSHIP MODEL	35



8.2.	POSTGRESQL AND JAVA CONECTION.....	37
8.3.	NOTIFICATIONS.....	39
8.4.	BACKUP STRATEGY	40
8.5.	TABLESPACE STRATEGY	40
8.6.	USER STRATEGY	40
8.7.	PROCEDURES	40
8.8.	SECURITY	41
9.	CONCLUSIONS	41
10.	ANNEX	43
10.1.	SOFTWARE SPECIFIC REQUIREMENTS.....	43
10.1.1.	INTRODUCTION	43
10.1.2.	OVERALL DESCRIPTION	44
10.1.3.	SPECIFIC REQUIREMENTS.....	46
10.2.	USE CASES.....	47
10.2.1.	CONTROLLER USE CASES	47
10.2.2.	PASSENGER USE CASES	47
10.2.3.	AIRLINE USE CASES	48
10.2.4.	AIRPORT USER USE CASES	48
10.2.5.	TECHNICIAN USE CASES.....	49
10.2.6.	ADMINISTRATOR USE CASES	49
10.3.	USER RESEARCH	50
10.4.	MOKCUPS	55
10.5.	MODEL VIEW CONTROLLER DIAGRAMS.....	55
10.6.	CODING STANDARDS	56
	REFERENCES	59

IMAGE INDEX

IMAGE 1:	NARANAIR LOGO	7
IMAGE 2:	GANTT DIAGRAM	11
IMAGE 3:	TRELLO WORKFLOW.....	12
IMAGE 4:	TRELLO WORKFLOW TAGS.....	13
IMAGE 5:	TESTPROJECTS REPOSITORY STRATEGY.....	15
IMAGE 6:	POPBL5 REPOSITORY STRATEGY.....	15
IMAGE 7:	NARANAIR GITHUB	16
IMAGE 8:	NARANAIR BRANCH STRATEGY	17
IMAGE 9:	NARANAIR BRANCH STRATEGY WITH FEATURES.....	17
IMAGE 10:	NARANAIR CODACY DASHBOARD	19
IMAGE 11:	NARANAIR CODECOV	19
IMAGE 12:	CONTROLLER SIMULATOR SEQUENCE DIAGRAM	23
IMAGE 11:	MAINTAINANCE SIMULATOR SEQUENCE DIAGRAM	23
IMAGE 12:	FLIGHTCREATOR SIMULATOR SEQUENCE DIAGRAM	23
IMAGE 15:	TILES 3 BEHAVIOUR	29
IMAGE 16:	INTERCEPTORS DIAGRAM	31
IMAGE 17:	NARANAIR RESPONSIVE TABLET SIZE.....	33
IMAGE 18:	NARANAIR RESPONSIVE WEB SIZE	33
IMAGE 19:	C3 ROUTE STATS BAR CHART.	34
IMAGE 20:	DATATABLES ACCOUNT MANAGER.	35
IMAGE 21:	DATABASE ER MODEL.....	36
IMAGE 22:	JAVA PERSISTENCE CLASS.....	37



IMAGE 23: USER CLASS INHERITANCE	38
IMAGE 24: USER CLASS INHERITANCE TRANSLATE TO DATABASE.....	38
IMAGE 25: CONTROLLER USE CASES	47
IMAGE 26: PASSENGER USE CASES	47
IMAGE 27: AIRLINE USE CASES	48
IMAGE 28: AIRPORT USE CASES.....	48
IMAGE 29: TECHNICIAN USE CASES	49
IMAGE 30: ADMINISTRATOR USE CASES	49

1. INTRODUCTION

In this document, it will be explained the NaranAir airport managing simulation software design, prototype and development. Starting from the problem analysis and description, continuing with the design and development and finally the final system results and some technical and methodological conclusions.

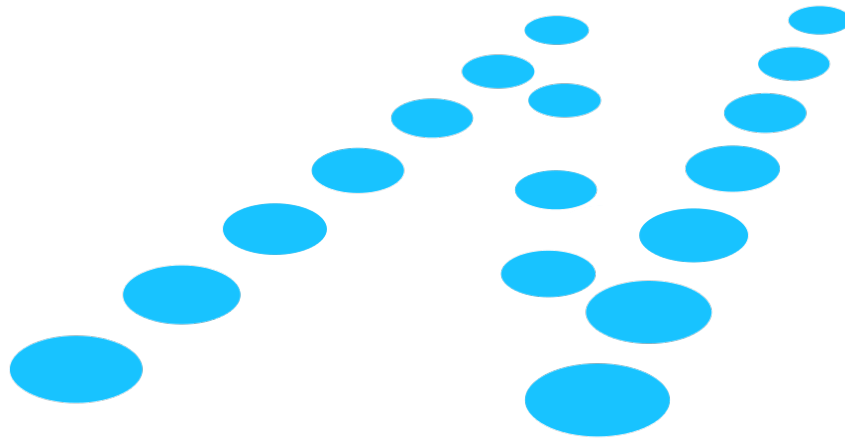


Image 1: NaranAir logo



2. PROBLEM DESCRIPTION

Nowadays airport management systems are insufficient and will be unable to support future demands as airport evolve. Airport manager struggle to balance the challenges of increasing operational efficiency and reducing operation costs.

Considering all explained before, NaranAir has taken an overview of Heathrow airport (London) and has focused the solution on two different aspects of the management of that airport: aviation operations and communication.

Regarding the aviation operations, it is known that among air traffic controllers, stress is the biggest occupational hazard. The job entails a high degree of responsibility and pressure, that is highly dangerous and can result in what is known as “acute episodic incident” or a near mid-air collision. Therefore, NaranAir provides the air traffic controllers with an automatic simulator of the aviation operations in order to improve training as well as the ability to make the right decisions at the right time to keep things moving.

On the other hand, communication in the right moment and format is crucial for the success of the airport. A good communication improves passenger experience and also helps in the airport operations providing information to airport controllers, maintenance technicians, passengers, airport users and airlines. So, for that, NaranAir airport managing system provides different user interfaces to satisfy all the customers, fulfilling all the requirements and analysing each customer or role priorities and most usable devices.

So, in conclusion, NaranAir airport managing system targets the solution on two main applications, one Java application for the air traffic controller automatic simulator and other Web application for the communication between airport users and the improvement of their user experience. With that, NaranAir expects to confront nowadays airport managing systems problems and give a new efficient system reducing operation costs.

3. RESEARCH

Before setting the objectives of the project, first the direction and position of the project had to be defined. For this, the necessities of the area, airport management, current state and possible impact had to be reviewed.

3.1. NECESSITIES

The necessities of the area were analysed by user research, in order to, first, identify the potential users of the application and, second, their needs and conditions. For this, the Personas method was used (explained in the section 10.3) as it was the most suitable one as the resources and time needed to perform more in-depth analysis were lacking. This way, five different types of users were identified, each one of them having own requirements:

- Passengers
- Administrators
- Airport Controllers
- Maintenance Technicians
- Airlines

Regarding to the actual market state, the gap founded by NaranAir was about a simulator that integrates both the public interfaces and the private interfaces in one product.

3.2. CURRENT STATE

Currently most of the airports have a dedicated webpage with different designs all around. If NaranAir manages to define the work pattern of an airport as generically and flexibly as possible the standardization of airport webpages be possible resulting especially useful for small airport with lesser income.

Nowadays, many alternatives are available with a high level of reliability. Most backed by big organisations.

The next ones are the current alternatives that can be find in nowadays market:

- **SITA Airport Management:** It is a suite of integrated applications in a way that all the airports operations can be managed from a central point.
- **IBS Airport Operations:** The most complete suite in the market. Older and more widespread than SITA.
- **Fujitsu:** Offers a big pack of software for airport management.
- **Unisys:** Has a package of software around crew and passage management.
- **Mercator:** Centered around logistics and cargo delivery.

- **Many more:** Most are specialised applications the full list is available [here](#).

4. PROJECT ORGANISATION

NaranAir airport managing system, has defined some main objectives for both applications and has divided and assigned them into different tasks and roles in order to obtain a final good result and project organisation as it is explained in the next 4.1, 4.2 and 4.3 sections.

4.1. OBJECTIVES

NaranAir airport managing system is divided into two main objectives: develop an automatic simulator of the aviation operations and develop a web application for all airport customers.

The automatic simulator, is responsible of managing the whole airport aviation operations, for that, it has to manage all the planes movements, giving them permission for moving, landing or taking off when it is possible and avoid the same operations when it is not possible. Summarizing, the simulator has to be able to manage all the air traffic on the airport and avoid any conflict or problem that can occur between the aviation operations, such as it is explained in the *section 6*.

Regarding the web application, this is aimed at improving the user experience and the communication between all the airport customers or users. For that, different user interfaces will be implemented for each customer. All that interfaces will achieve the requirements of each customer, so that, the functionalities available to the user will vary depending on the role of the logged user. This point will be explained in depth in *section 7*.

In addition, both applications or system parts they need to store all the data somehow, with the aim of being able to share this data between both applications and in this way, be able to communicate between them. To achieve this goal the system will have an object-relational (ORDBMS) ^[1] database where all the needed data and information will be storage.

^[1] *An object relational database management system (ORDBMS) is a database management system with that is similar to a relational database, except that it has an object-oriented database model. This system supports objects, classes and inheritance in database schemas and query language.*

Finally, with the objective of achieving a consistent and stable final product, all the system functionalities and more concretely the riskiest functionalities will be tested by Continuous Integration (CI) ^[2] tools, as it will be explained later in the *section 5.3*.

^[2] Continuous Integration (CI) is a development practice that requires developers to integrate code into shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

4.2. TASKS

The two main objectives explained before, are divided into different tasks, organized from the most general ones to other more specific. The most general ones are organised by a Gantt Diagram, created using [OmniPlan](#) software, where all tasks tracking and duration is reflected. In addition, NaranAir can carry a detailed monitoring of all the tasks and determine at any moment what is the state of the project and how the team is getting with the work.

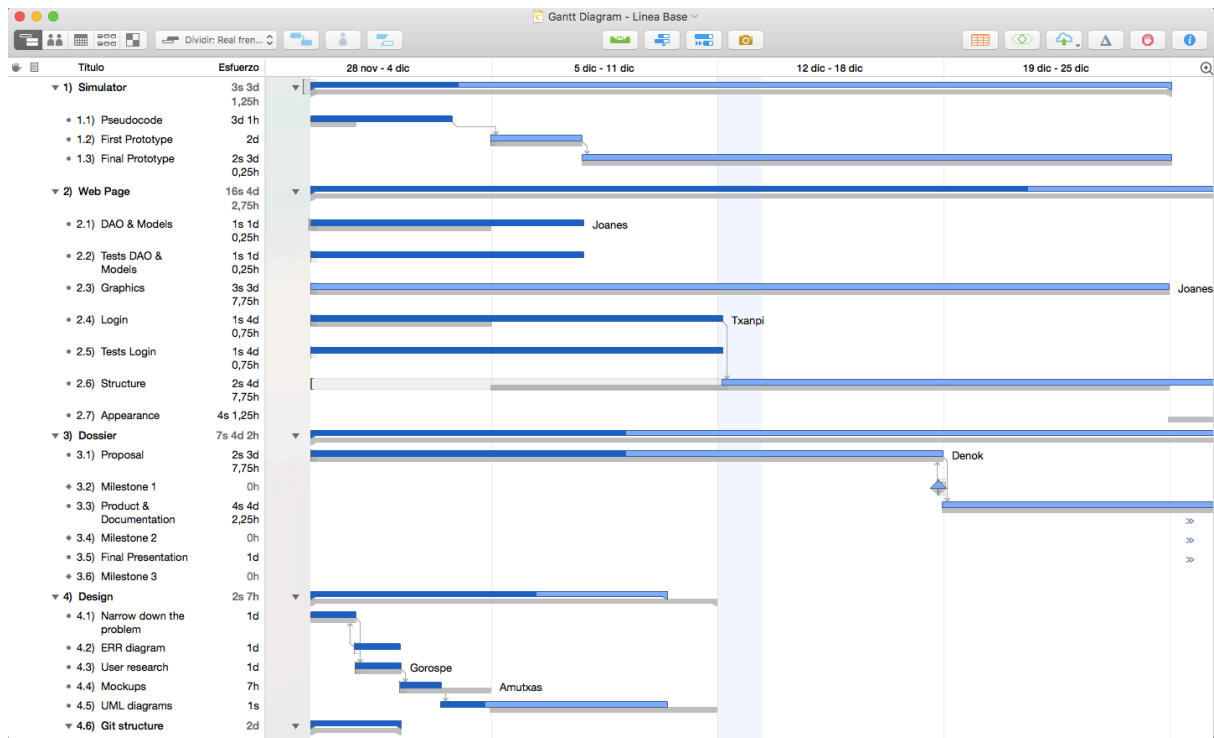


Image 2: Gantt Diagram

As it can be seen in the *Image 2* the general tasks are grouped in some groups, better called Work Packages^[3], such as, simulator, web page, dossier and design. Inside each of them, are defined the general tasks that are going to be performed during the development of the system and their progress. The most important dates, are also reflected in the diagram in order to follow the work state compared with them.

^[3] A Work Package in Project Management is a group of related tasks within a project.

In addition to the Gantt Diagram, for the follow-up of the specific tasks, [Trello](#) software is used, this software allows NaranAir team to declare what is the state of the tasks: To do (“*Egiteko*”), Week tasks (“*Astean egin beharrekoa*”), Doing (“*Egiten*”), Testeatzzen (“*Testing*”), Done (“*Eginda*”), as it can be seen in the *Image 3*. With these descriptions, the tasks can be moved from one to another board depending on their state and can be better organised.

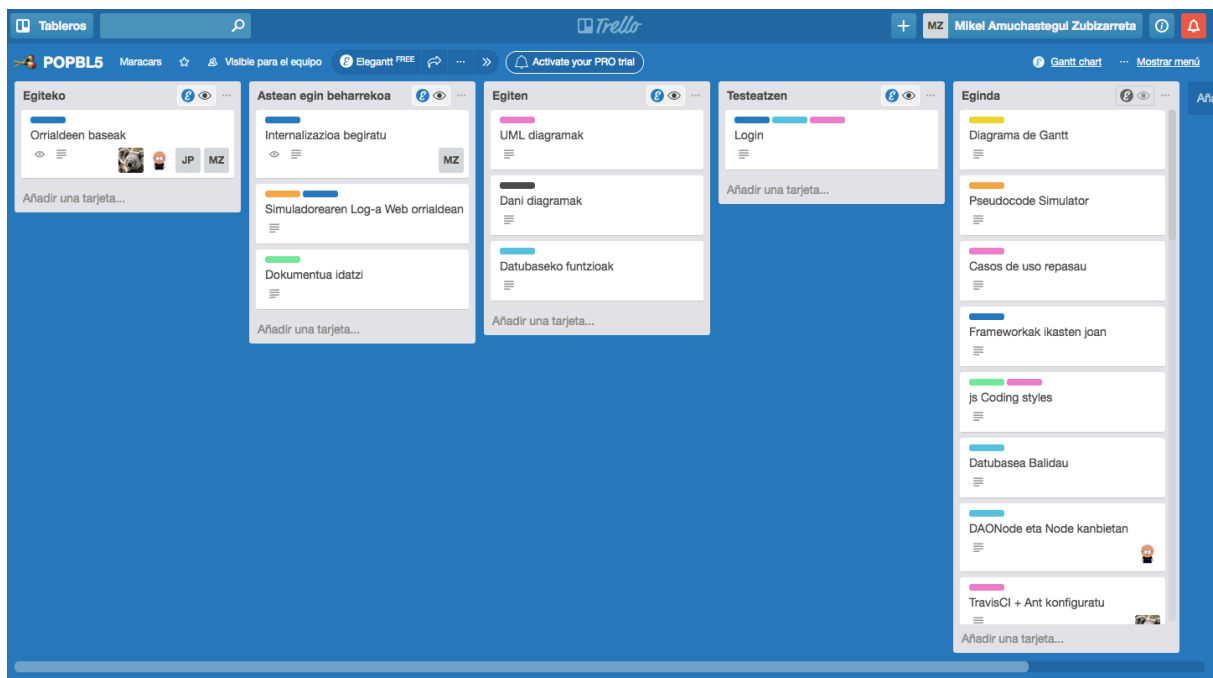


Image 3: Trello Workflow

On the other hand, in order to group also this tasks some labels or tags are added to each of them. With that, the tasks can be filtered depending on the tags and organize the work properly, like it can be seen in the *Image 4*.



Image 4: Trello workflow tags

In conclusion, NaranAir tasks are organised from a high level to a more detailed level, that is, the Gantt Diagram is used to define all the general tasks that are going to be performed during the whole project and the [Trello](#) board is used to keep track of the week more detailed tasks.

4.3. ROLES

In relation to the project roles, these, are defined depending on the Work Packages explained in the *section 4.2*, that is, each group member is the responsible of one Work Package (simulator, web page, dossier or design). However, that does not mean, that the responsible of the group has to take care of all the Work Package tasks, these can be delegated to the other project members, on the condition that, the responsible has to be aware of the status of the tasks delegated. To carry this out, NaranAir has not follow a certain strategy, instead of that, the team has follow the next simple procedure:

1. Define every week the tasks on [Trello](#) workflow.
2. Divide the work and assign each task to a group member, taking into account each member skills.
3. Once the tasks are finished, each member has to be aware of his role situation, that is to say that, has to know what is the situation of the tasks it is responsible for.
4. All this information is reflected to the Gantt Diagram, by the Dossier responsible.

Besides the fact that each task group has a responsible assigned, one group member has the manager or project responsible role. That is, this role, has to be aware every moment about the state of the project, how work is done, what tasks are defined, which of them are done and which not, etc. summarizing, has to follow a



strict and daily follow-up of the project and give feedback to other members every certain lapse of time.

In the next table *Table 1* can be seen an overview about the role assignation of the NaranAir project development.

Group Member	Role
<i>Ander González</i>	Manager and Web Page responsible.
<i>Joanes Plazaola</i>	Design responsible.
<i>Joseba Gorospe</i>	Simulator responsible.
<i>Mikel Amuchastegui</i>	Dossier responsible.

Table 1: NaranAir roles

5. DEVELOPMENT STRATEGY

The NaranAir airport managing system development strategy is defined in the next *sections 5.1, 5.2, 5.3 and 5.4*. For doing that, [Git](#)^[4] is used with a certain repository and branch strategy explained in the *sections 5.1 and 5.2* and all that is complemented with some Continuous Integration tools explained in the *section 5.3*.

^[4] *Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.*

The main reason for choosing [Git](#) instead of other version control systems is that, firstly the team does not want to use a centralised version control system, because it needs to have the chance to clone a copy of the repository on each developer hard drive and not have a unique central copy where commit the changes.

So taking into account these reasons and analysing what are the most used distributed version control systems nowadays ([Git](#) and [Mercurial](#)^[5]) NaranAir choose [Git](#) and [Github](#)^[6] tools for managing the project online. So, in a certain way, the reason for choosing [Git](#) and [Github](#) is related, because, [Github](#) platform does not have support for [Mercurial](#) by default, it is necessary to add a [plugin](#) to work with it, that means it will be indispensable to know how it works this unknown plugin, while at the same time the team will have to become comfortable with [Mercurial](#), also an unknown version control system, whereas, [Git](#) is more familiar and used before in some other projects.

^[5] *Mercurial is a cross-platform, distributed revision-control tool for software developers.*

^[6] Github is a web-based Git repository hosting service.

5.1. REPOSITORY STRATEGY

Starting with the repository^[7] strategy, NaranAir has determined the next strategy shown in the images *Image 5* and *Image 6* as the repository methodology to follow during the project development.

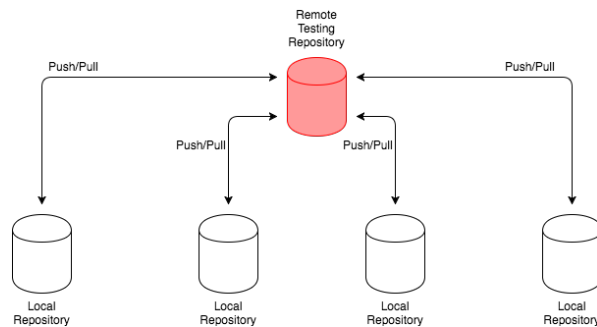


Image 5: TestProjects repository strategy

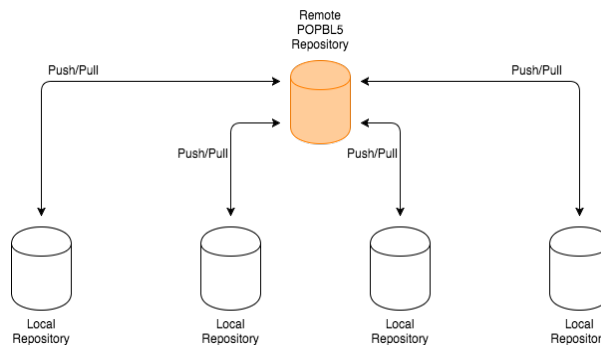


Image 6: POPBL5 repository strategy

^[7] In revision control systems, a repository is an on-disk data structure which stores metadata for a set of files and/or directory structure.

As it is shown in pictures *Image 5* and *Image 6* the repository strategy is divided into two main parts or two different repositories, on one hand it is a remote repository called “TestProjects” destined to push^[8] there all code done as a test^[9], that is, due to the increase of new frameworks and new tools needed in the development of this project, all the code done to test these new tools is uploaded to that repository, with the aim that all the members of the group can reach a general overview about all the new technologies.

^[8] Update content on the remote repository.

^[9] Not JUnit tests, just examples for the new frameworks.

On the other hand, the second repository shown in the *Image 6* is dedicated to the development of the airport managing system as a whole, so, continuing with the same previous structure, each developer has a local repository where he develops and he push and pull ^[10] the code changes to and from the remote repository respectively.

^[10] *Download and update local content from remote repository.*

For the remote repositories management as it is explained in this section introduction [Github](#) platform (shown in the *Image 7*) is used.

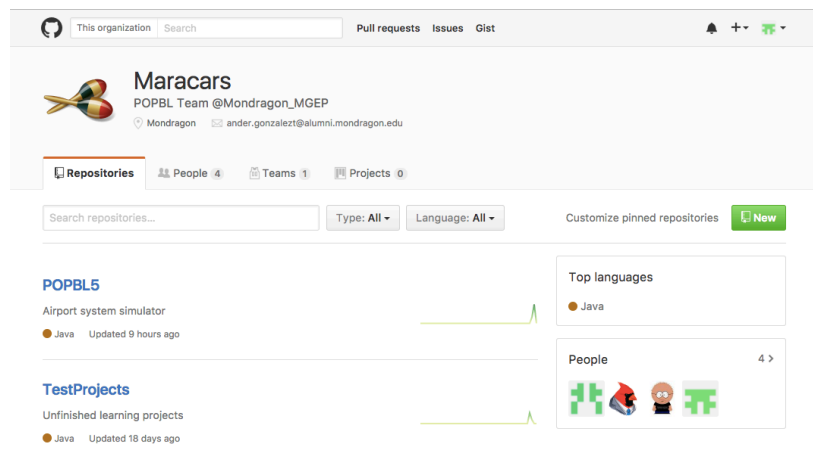


Image 7: NaranAir GitHub

In relation to the management of the Local repositories, two new platforms are used, [TortoiseGit](#) ^[11] for Windows developers and [SourceTree](#) ^[12] for MacOS developers. Both are applications that help working with [Git](#) in a more simpler way, without the need to insert [Git](#) commands directly and with some user-friendly interfaces. [TortoiseGit](#) is used due to the developer's familiarity and because it is a lightweight tool based directly on Git bash console. However, [TortoiseGit](#) does not offer a Mac version so [SourceTree](#) was the most user friendly free git repository management helper found.

^[11] *TortoiseGit is a [Git](#) revision control client, implemented as a Windows shell extension.*

^[12] *SourceTree is a visual [Git](#) and Mercurial client for Mac and Windows.*

5.2. BRANCH STRATEGY

NaranAir airport managing system branch strategy in summarized in the following *Image 8*.



Image 8: NaranAir branch strategy

The branch strategy is determined by two main branches, one “*Master*” branch where they will be the most stable versions of the system, all them tagged and another “*Development*” branch where new functionalities are developed and added, when these new functionalities they reach a stable version, this branch is merged with the master to store there this new version.

Using this schema, the “*Development*” branch has its own working system. If a new feature is going to be developed, a branch will be created from the last commit in “*Development*” and then will be developed separately. This will allow the abandonment of some of this features in case they aren’t useful. After the feature is finished, it will be merged back into “*Development*”. Usually, as this branches are developed by a single developer meaning that the whole development may happen locally and be represented by a single commit on the repository. But in the case where this branches get too long, some cherry picks will be done in order to avoid big changes that will carry on lot of conflicts. So, when creating these new feature branch NaranAir branch tree will have the structure showed



Image 9: NaranAir branch strategy with features

The main reason for choosing this branch strategy is that working with too much branches is more difficult; it takes more time to execute each merge and resolve conflicts if there are significant changes between branches is more complicate. Besides the fact that working with a lot of branches is difficult, development team is not so big as to branch by team, all developers can commit their changes on the same branch and the cost of resolving conflicts in this way is less than the cost of merging conflicts between branches in case of the creation of new more branches, in addition, there are not so much situations where development team needs to work on the same set of files concurrently, because work is divided by functionalities and each functionality works with its set of files.



For all these reasons, this team has used two main branches and one small branch for each functionality in order to have all the work of Git correctly structured.

In addition to all the concepts explained before, this NaranAir branch strategy can suffer from little changes in case of some bugs or problems that can appear on the system during the development process. In case of that, the team, defines that a new branch will be created in order to fix this problem without disturbing other developers, this branch will be denominated as “Hotfix” branch.

5.3. INTEGRATION STRATEGY

Project continuous integration is performed by [Travis CI](#) ^[13] tool, with this tool each check-in into the remote shared repository is verified with an automated build, allowing the team, to detect problems early. To complement the work of this tool [Codacy](#) ^[14] and [Codecov](#) ^[15] are added for automated code reviews, code analytics and code coverage respectively.

^[13] *[Travis CI](#) is a hosted, distributed continuous integration service used to build and test software projects hosted at [GitHub](#).*

^[14] *[Codacy](#) automates code reviews and monitor code quality over time.*

^[15] *[Codecov](#) provides metrics and insights into the results of tests through code coverage reports.*

When choosing a continuous integration tool, some other mechanisms or utensils such as [Jenkins](#) ^[16] were considered by NaranAir team. This tool can be considered the best known among the continuous integration servers but it has some main differences from [Travis CI](#). In this case, the server has to be set up by the developer, whereas [Travis CI](#) is hosted by a third party, this fact means that the server also has to be maintained, that is, this requires more time. On the other hand, [Travis CI](#) is a free tool for open source projects only, while [Jenkins](#) is free for both private and public projects as long as the developer has its own server. Focusing on the usability of both applications, [Travis CI](#) is probably one of the easiest continuous integration servers to get started with, set up is as easy as linking the [GitHub](#) account, this can be considered the key point for deciding between one or another.

^[16] *[Jenkins](#) is an open source automation server written in Java.*

So, taking into account that NaranAir is developing an open source project using [GitHub](#) platform and the facility for setting up it was decided to use [Travis CI](#) tool.

Concerning to [Codacy](#) ^[17] and [Codecov](#) ^[18] tools, the first one offers static analysis, code coverage and metrics information like it is shown on the *Image 10*, whereas the second one is more oriented to code coverage (*Image 11*).

^[17] *Codacy automates code reviews and monitors code quality over time.*

^[18] *Code coverage measurement.*

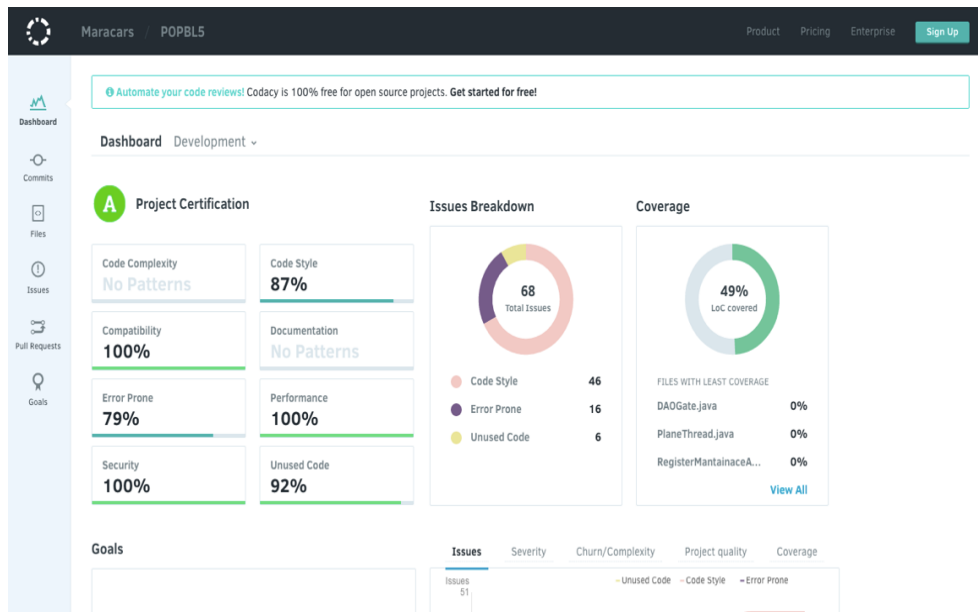


Image 10: NaranAir Codacy dashboard

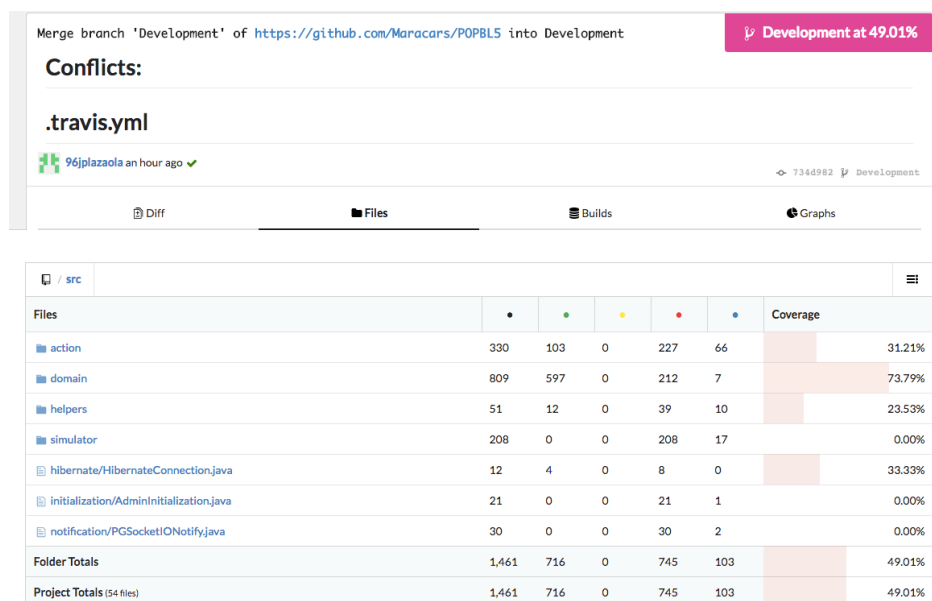


Image 11: NaranAir Codecov



[Codacy](#) and [Codecov](#), both tools are well complemented between them, the first one offers more information about other facts apart from code coverage, such as, issues, code style and other static analysis information. However, [Codecov](#) offers detailed information about the project code coverage for each commit and file, so, a better tracking of the code coverage can be done with this tool.

Summarizing, [Codacy](#) is used oriented to static analysis and [Codecov](#) oriented to testing code coverage, that is, dynamic analysis.

Apart from these two tools, other tools also were taking into account at the time of the automation of the static analysis, using [SonarQube](#)^[19] tool for this objective was valued by the team, but the fact of having to set up the server was a key point for choosing other tools such as [Codacy](#) and [Codecov](#) in this case.

^[19] [Sonarqube](#) is an open source platform for continuous inspection of code quality.

5.4. CODING STANDARD

Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices, and methods for each aspect of a program written in that language. These conventions usually cover file organization, indentation, comments, declarations, statements, whitespace, naming conventions, programming practices, programming principles, programming rules of thumb, architectural best practices, etc. Having coding conventions is very important, they help to improve the readability of the source code and make software maintenance easier.

This section will show the list of the conventions which have been defined in the NaranAir project. These conventions or coding standards have been selected in relation to some code patterns pre-defined by [Codacy](#) tool. These patterns are the next ones: [CSSLint \(CSS\)](#), [Checkstyle \(Java\)](#), [ESLint \(Javascript\)](#), [JSHint \(Javascript\)](#), [PMD Java \(Java\)](#). Inside these patterns [Codacy](#) gives the chance to select some rules from a global pattern list, so, NaranAir has defined what are the most important for the team and select them to add to the [Codacy](#) automated code review, there can be find also all the metrics define by NaranAir development team. The list of all the code style checks defined by NaranAir is shown in the *section 10.6*.

**Note: Each code standard rules can be accessed by clicking in each code pattern link.*

5.5. TESTING

Testing has been a very important part of this project as it helps ensuring that all the changes made in the code do not affect negatively to the application.

At first, the method Test-Driven Development (TDD) ^[20] was used. TDD method is a software development process that is based on doing the test before coding the functionality of the application. It was very effective, as it ensured that all the functionalities were tested, but as there was no member of the group used to this method, it took too much time. For this reason, after one or two weeks proving the TDD method, the group took the decision to discard it and use the common method for coding, first develop the functionality and then test it.

^[20] Test driven development is a software development process that relies on a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only.

Tests are now performed using [JUnit4](#) ^[21] testing library and are implemented on the CI strategy on the project being currently run and checked by [Travis CI's](#) build script. This testing's global coverage is around 60% but covers the main critical areas such as the simulator logic and the [Struts2](#) actions logic and workflow.

^[21] JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated SUnit.

For this testing to be made in a simple way in addition to the previously mentioned, [Mockito](#) was used for mocking ^[22] and also its extension [PowerMockito](#) to enable static class mocking (which makes some tests not to be taken into the coverage count as it statically generates a class on compile time so that the original is never run). This mocks are also accompanied by the usage of [Java's API's Reflection](#) package. This allows the testing of private functions and the setting and getting of private or protected values without changing their software design logic.

^[22] Mocking is used in unit testing when an object under test has dependencies on other (complex) objects. To isolate the behaviour of the object wanted to test other objects are replaced by mocks that simulate the behaviour of the real objects.



5.6. DOCUMENTATION

For the documentation of the code, as it's usually done on java based project, [Javadoc](#) is used in addition to [JAutodoc](#) plugin that auto generates some documentation of dummy functions, such as getters and setters.

Not all the code is manually documented. Some classes that are meant to be self-explanatory such as test classes and some simple private functions. Public functions and classes especially those meant to be interfaced as a facade have longer and more detailed documentation explaining their logic and structure.

All this together makes for a simple documentation that is clear enough and can result useful for a future development and catch up without it, being long, tedious and fall into obviousness. This is why the intensive documentation of all the code was considered detrimental for the quality of the software.

6. SIMULATOR

The NaranAir airport managing system simulator is the framework which controls the functional part of the application, in other words, it takes care of creating flights, simulating them moving the planes, acting as the airport controller giving permission to the planes or even doing the job that a technic maintainer should do, that is revising the planes of the airport. This, helps the application to be more close to reality as it tries to proceed like a real airport is managed.

For the realisation of this system, a design of its behaviour was done in order to have clear what and how will be implemented. In this case, it has been decided doing sequence diagrams showing the general performance of the simulator. The *Image 12* shows how the program will create flights, the *Image 12* shows the behaviour of the controller, who will give permissions to the planes, and finally, the *Image 11* shows how the revisions of the planes will be implemented.

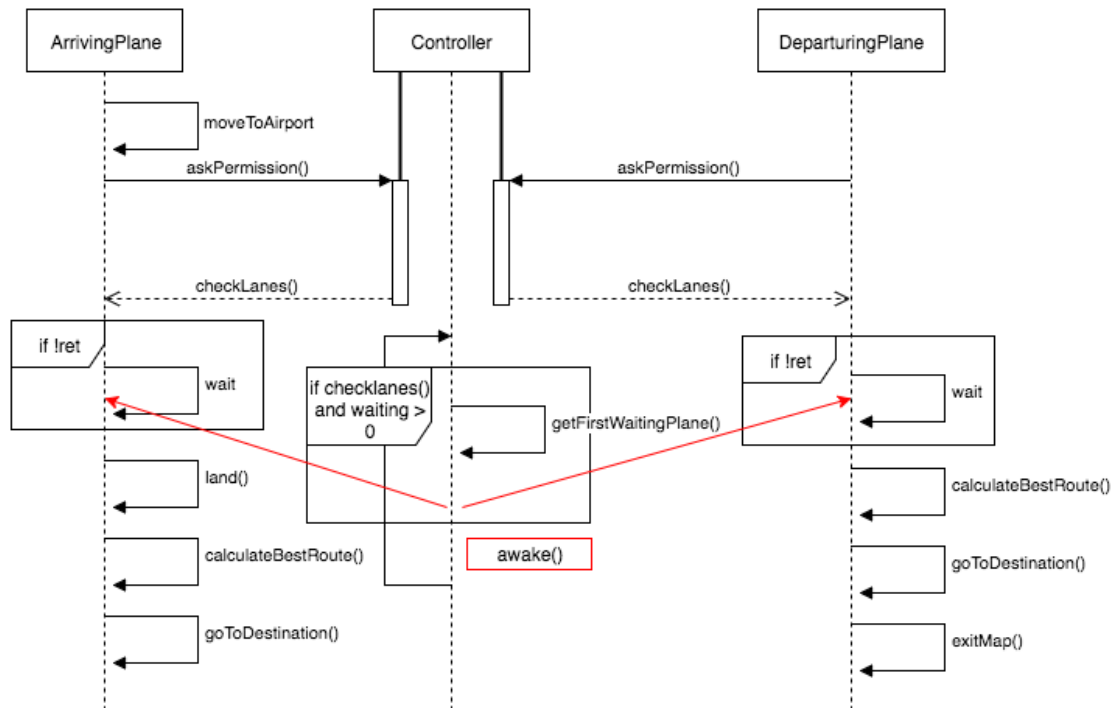


Image 12: Controller simulator sequence diagram

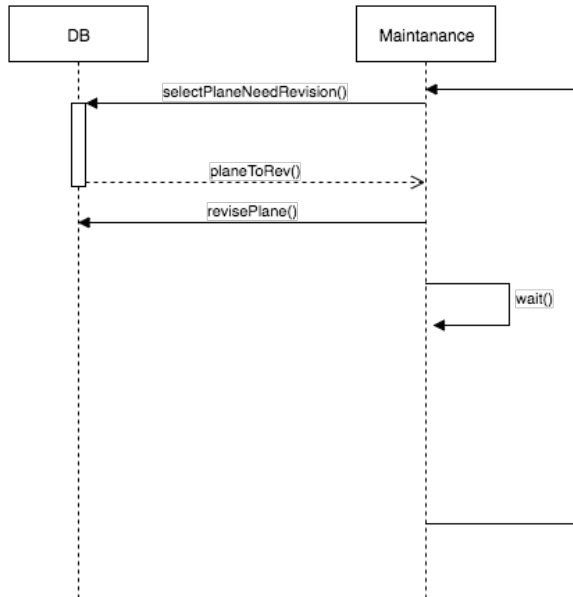


Image 13: Maintenance simulator sequence diagram

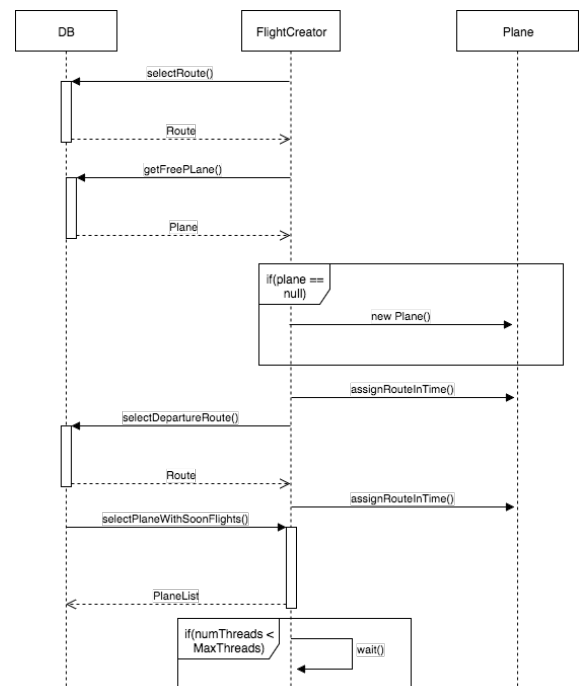


Image 14: FlightCreator simulator sequence diagram

After finishing the design, a first functional prototype is being implemented. This prototype will not fulfil all the requirements and all the functionalities of the final version, but it as it will follow the structure proposed in the design, it will be easy to complete or add new functionalities to it in order to accomplish all the requirements.



In this prototype (*Prototype V1*), the simulator should have implemented the flight creator, airport controller and technical maintenance parts but it won't simulate the flights, move the planes... In addition, a log console will be implemented, exclusively for the prototype to be possible to see what is happening, because taking into account that the planes will not move, without a log, it would be impossible to see what the simulator does. This log will be shown in the web application, so that the user will be able to watch when the flights are created, when a plane is arriving or taking off, when the planes ask permission to the airport controller and when it gives permissions and when a plane is revised.

After the prototype, the movements of the planes, the calculation of the route that each one should take in order to go to the gates and also accurate the parts that were implemented in the Prototype V1 with the aim of reaching a Final Prototype.

6.1. AIRPORT MANAGE LOGISTIC

When it comes to the internal management of the airport, an algorithm called Dijkstra has been implemented and improved in order to calculate the route among the lanes that a plane must follow after landed or before it departures. This algorithm is based on choosing the shortest path between nodes. Taking a source node, it calculates all the shortest routes for the rest of the nodes, we have to take into account that in the original case, the paths were just unidirectional, but the improvement allows Dijkstra to handle bidirectional routes in order to be useful for both, arrival and departing planes.

In addition, it has been implemented a flight programmer that fills a schedule of all the week taken into account the maximum number of flights per hour that the airport can support. This flight programmer will be periodically checking if there is any free time in the schedule and if so, it will create the pertinent flights.

6.2. SYNCHRONIZATION

The synchronization of processes is used in order to resolve some problems which would produce a big impact in the simulator, but they cannot be seen easily in the code, as they probably would appear in the execution. Here is the list of each of the problems with an explanation of what is and how it has been corrected.

In each airport, there can be only a specific number of active planes. If there were more planes than the permitted, there would be a high probability to crash between them. This problem is solved using an `AtomicInteger`, a variable that stores the number of active planes that are in the airport at every moment. An `AtomicInteger` is an integer that can be only changed or even read by one process in each time. This to be possible, the variable has its own predefined function such as `incrementAndGet()`; `get()`; or `decrementAndGet()`; that guarantees that only one process has access to it. In



this case, the FlightCreator class will have the variable, it will check if the number of active planes is less than the maximum and if this condition is true, it will fill check if any plane has to be activated and will activate it. When it activates a plane, it will increment by one the number of active planes, and when the planes action ends, as it has a reference to the variable, it will decrement it by one. It is also implemented a thread pool for having the maximum number of possible active plane threads and not to create more than possible.

In each lane, only one plane is allowed to stay. It is compulsory to ensure that in each lane there is only one plane as if not, there will be an accident between them. To resolve this problem, it has been implemented a Semaphore for each lane so that only one plane can enter to it simultaneously, and if another comes, it will wait till it gets free. Semaphores are signalling mechanisms which can allow one or more process to access to a program section. For that, they have a certain value of tokens and to entering to the section, a process takes one of those tokens (*acquire()*), when it leaves, it returns the token (*release()*), and if there is not any token while a process attempt to enter, it will wait till somebody returns one. In this case, the lanes will have just one token, a plane will take it when entering into it and return when exiting.

The planes need the permission on the airport controller to land or to take off. These requirements have also been solved using Semaphores, having each plane their own semaphore (without tokens at first) and when they want to land or take off, they will ask permission to the AirportController who will check if there is any free lane. If there is any free lane, the airport controller will assign the lane to the plane and it will land/take off straightly, but if all the lanes are busy, the planes will wait in the semaphore. Meanwhile, the airport controller will have a list with the planes that are waiting and it will be constantly checking if there is any lane free. When any of those busy lanes, it will assign it to the first waiting plane and it will give a token to its semaphore. Another solution could be implemented, where the airport controller would have a semaphore and the planes would wait to it, but the selected solution allows the airport controller to have the reference to the planes and not the other way around, so it seems more realistic, as it is not logic a plane to have the reference to the controller.

Another problem, more related to the application than to the airport operation is the synchronization while accessing to the database. It has been noticed that as there are various threads trying to connect at the same time into the database, it sometimes fails. To correct this error, it has been implemented a semaphore acting as a mutex (Mutual Exclusion) which will ensure that each connection to the database is unique and the resting processes wait till the statement is finished.

In order to be able to finish the simulator and the system in a properly and clear way, two thread pools have been implemented. The first contains three threads (FlightCreator, AirportController and AutomaticMaintenance), and the second one, the thread pool commented before and which contains all the threads of the active planes.



When finishing the simulator, all the threads of the first pool are interrupted and with this, the FlightCreator thread will also interrupt the second pool so that there isn't any active thread after finishing the simulator.

Another issue is related to Struts2 way of managing threads. For each action that is executed one instance is created for each session. Usually this makes the synchronization error go but in the case of the tables several requests can happen overlap with the responses. To avoid this every action that was called from a high frequency caller was limited with a mutex in order to avoid concurrent modification of class level data.

7. WEB APPLICATION

The NaranAir airport managing system web application is developed taking into account each system user requirements and needs, based on a responsive web design and taking care of a good data access layer management. In this way, the web application development is explained in the *sections 7.1 and 7.2*.

7.1. BACKEND

Since the beginning, NaranAir, took various frameworks based on different languages into consideration for the backend^[23] development. Finally, the decision was taken measuring the suitability according to the following factors:

^[23] *Data access layer of the web application.*

- **Entry level:** The difficulty related to the implementation as developers may not be familiar with that framework or its base language and that may make the development slower and of a lower quality.
- **Simplicity:** The complexity of the frameworks structure and resulting code.
- **Functionalities:** The number of functionalities that the framework offers.
- **Popularity:** How widely spread it is. This is important as it will make finding resources and help online easier. Also, the gain the developers will get from taking the framework up.

In this process NaranAir considered the following alternatives: [TomcatServlets](#), [Struts2](#), [ASP.NET](#), [Django](#) and [Rails](#). The comparison according to the factors described above is shown in the next *sections 7.1.1 7.1.2 7.1.3 7.1.4 7.1.5*.

7.1.1. TOMCAT SERVLETS

- **Entry level:** The lowest entry level among all the alternatives. All the developers are familiar to the framework.



- **Simplicity:** The base structure is simple but it can become complex very fast as it is the one, that works the closest to the http protocol.
- **Functionalities:** It has the least number of functionalities as it barely has layers over the protocol what means that almost any functionality has to be added through a library.
- **Popularity:** Servlets are widely used although they are usually interfaced in a way that makes their usage less rudimentary.

7.1.2. STRUTS2

[Apache Struts 2](#) is an open-source web application framework for developing [Java EE](#) web applications.

- **Entry level:** Based on the servlet API [Struts](#) is a framework that does not have a huge gap for the developers.
- **Simplicity:** It is simpler to use than servlets as it implements layers over routing and parameter interception.
- **Functionalities:** It has more functionalities than servlets and offers interfaces for modelization, filtering, validation, etc. But still requires extra libraries for other functionalities as database management.
- **Popularity:** It is not very used although it has been funded by [Google](#) with the [Google's Patch Reward program](#).

7.1.3. ASP.NET

[ASP.NET](#) is an open-source server-side web development to produce dynamic web pages.

- **Entry level:** As similarities between [Java](#) and C# are huge the translation from one to the other would not be a huge deal for the developers.
- **Simplicity:** It works in a way that is more similar to [Struts2](#) although it is a completely different framework.
- **Functionalities:** [ASP.NET](#) has a fair amount of functionalities, for instance, offers a login security model based around a membership API giving facilities for managing requests and handling sessions.
- **Popularity:** Being the web backend supported by [Microsoft](#) its use is very frequent.

7.1.4. DJANGO

[Django](#) is a free open-source web framework, written in [Python](#) (Ruby, s.f.), which follows the model-view-template (MVT) architectural pattern.

- **Entry level:** Based on python, the low knowledge about the language and framework could lead the project to a disaster.



- **Simplicity:** Its syntax is not the most complex one, and the implementation of login, register and some other normally used are already implemented, and that is an important point in favour of [Django](#). The routing and parameter interception layers also make easier to implement than other frameworks.
- **Functionalities:** [Django](#) has many functionalities, for instance, the administrator view, that gives the developer the option to manage the tables of the database. [Django](#) has also many implementations of common webs, and that is very important for the developer.
- **Popularity:** Django is one of the most known and used framework of the 3.0 web technologies, but is not still as used as [Java](#), [ASP.NET](#) or others. Used in [Instagram](#), [Spotify](#), [Pinterest](#), [NASA](#), [BitBucket](#), etc.

7.1.5. RAILS

[Rails](#) is a server-side web application framework written in Ruby.

- **Entry level:** Based on [Ruby](#), language that is not too complex, but no one in NaranAir team knows the language, so the team would have to spend some time learning and mastering the framework.
- **Simplicity:** No SQL knowledge is needed to implement a database in [Rails](#), and that makes its implementation easier.
- **Functionalities:** It has more functionalities than servlets and offers interfaces for modelization, filtering, validation, etc. But still requires extra libraries for other functionalities as database management.
- **Popularity:** It is one of the most used 3.0 web technologies, used in [GitHub](#), [Twitch](#), [SoundCloud](#), etc.

7.1.6. STRUCTURE

Taking into account the analysis of the different backend frameworks explained in the *sections 7.1.1, 7.1.2, 7.1.3, 7.1.4, 7.1.5* before [Struts2](#) is the framework used by NaranAir, behind [Apache Tomcat Server](#) and Eclipse IDE for Java EE developers, to control the interaction with the backend through an MVC (Model View Controller) controller, explained in the next *sections 7.1.6.1, 7.1.6.2 and 7.1.6.3*.

In this case the models are java classes that are also persisted into a database using [Hibernate](#) ^[24] (explained in the *Section 8.2*), the view is composed by all the “.jsp” files that are then assembled by [Tiles3](#) (explained in the *section 7.1.6.2*) into the pages displayed to the user and the controllers are the Struts2 actions (where the logic and most of the structure resides).

^[24] *Hibernate is an object-relational mapping framework for the Java language.*

7.1.6.1. MODEL

The model is defined in a [Java](#) package containing all the declarations of the classes that will be used to instantiate the necessary data. Such as planes, airports, users, etc. All this data, as mentioned before, is then persisted through [Hibernate](#) to a [PostgreSQL](#) database where it is stored. To access and alter this data DAO (Data Access Object) classes are used. Although no SQL is used only HQL a SQL based object oriented query language. This way, the database is always up to date with the model and no tedious and error prone [JDBC](#) DAOs have to be created.

7.1.6.2. VIEW

The view is managed by [Tiles 3](#) framework. This framework enables tiling on the views, defining a “.jsp “ as a layout and leaving gaps for the content. Then the elements to fill the gaps are declared on an “.xml” configuration file in a way that redundant “.jsp” code is avoided as it can be shown in the *Image 15*. For instance, instead of creating a sidebar for every page a few are defined and then placed depending on the page that has to be displayed. This way also, one file one time has to be changed to make all the implementations of that tile change.

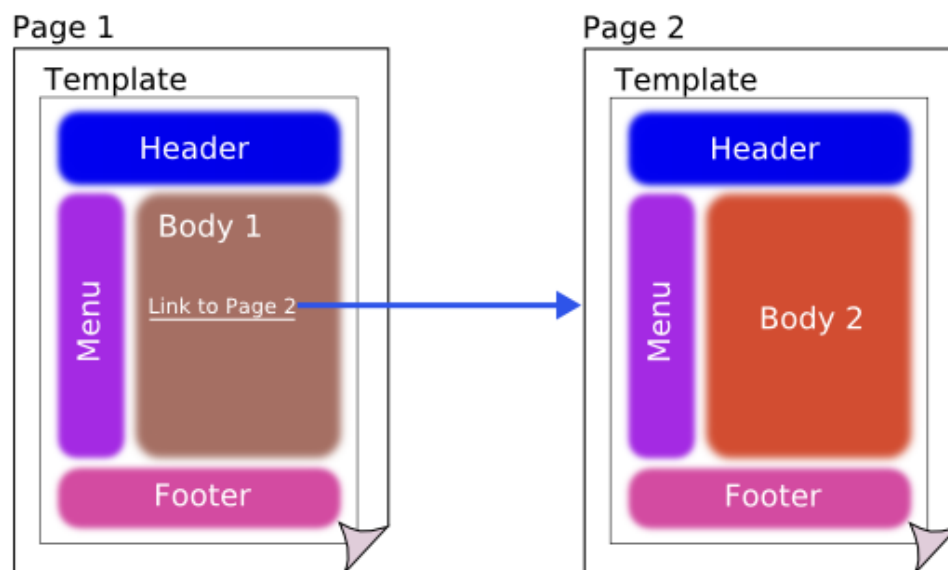


Image 15: Tiles 3 behaviour

7.1.6.3. CONTROLLER

The controller or the user interactions are handled through routing files that can call [Java](#) classes and methods. Each one of this routes defined by a path (relative URL), being <http://maracars.sytes.net/Naranair/u/myUserName's> action route /u/{username} for example is an action. All this considered the user interactions are divided into smaller parts: Common actions (workflow and data feed), additional actions (JSON requests), interceptors, initialization and relation to the simulator.



7.1.6.3.1. COMMON ACTIONS

Common actions are those that control the flow of the webpage, redirecting user to other actions or pages. They also check the user's inputs and output values and push them into the request scope for the view to display or use them. At the moment, all actions classes are atomic, meaning that each class has only one method to be called, the execute method. Although this method's name is defined by contract and thus does not need to be implemented through an interface or defined in the action's xml, NaranAir is extending the ActionSupport class provided by the [Struts2](#) framework, which enables validation and error messages by default.

One example of this actions could be the login action. If the login data is incorrect it will return to the logging page showing the error that happened. Or, if the login is successful, will return to the calling page of the login.

7.1.6.3.2. ADDITIONAL ACTIONS

Additional actions are those that do not provide interaction directly but they are designed to work in a way that enables dynamism in the webpage (avoid reloads). Up to date, this functions are only JSON string requests for the [JavaScript](#) functions all across the system (maps, tables, etc.). Also by an internal convention of NaranAir, this action's Java classes should contain an inner dummy class whose instances contain the data requested so that it is encapsulated and we prevent data leakage to the front end.js

For instance, the [Datatables JavaScript](#) library enables an option to request the displayed rows to the server. This way, [Datatables](#) sends a request to the server where an action will read the request parameters and according to them will filter the data and send it back to the user so that, if the data requested belongs to a 2 million table, the client doesn't have to load all the 2 million entries, but only the 10 or 5 that he requested.

7.1.6.3.3. INTERCEPTORS

Interceptors are a feature included in the Struts2 framework which work like an onion, being the core the action class and each of the layer an interceptor. They work both ways when going into the action and when returning from it. As the *Image 16* diagram shows.

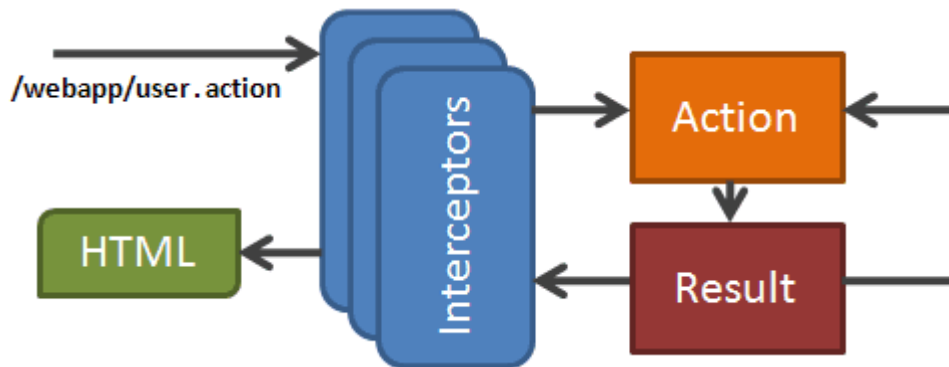


Image 16: Interceptors Diagram

These are used in the default way except from three cases.

Firstly, Internationalization, it is actually an interceptor that replaces the key by the value depending on the actual locale.

Secondly, the user validation checks the logged user and the access rules for the requested action and denies or allows access accordingly, this one is custom.

Finally, the store interceptor enables propagation of messages through two or more actions.

7.1.6.3.4. INITIALIZATION

The server's configuration file enables some classes that implement a certain interface to be executed both on server start and stop and using that interface several initialization tasks are performed such as the simulators initialization, the connection to the database, the default admin user (in case there is no admin) and some values of the database.

7.1.6.3.5. RELATION TO THE SIMULATOR

As mentioned before the simulator is launched within the server on a separate thread allowing interaction between both the simulator and the webpage directly and making the relation between both more consistent. However, all this happens having the option to make them both independent open.

7.1.7. DEPLOYMENT

For the server deployment, a VPS (Virtual Private Server) was hired. For this we used [OVH](#)'s services. In this server, the Web Applications ".war" file was deployed on a Tomcat instance with a PostgreSQL database configured.



As the VPS default hostname, the one provided by the hosting company is a serial code and is not very appealing for the visitors a free DNS to a subdomain was used through [NO-IP](#). The subdomain's name *maracars.sytes.net* and the Web applications base address is <http://maracars.sytes.net/Naranair/>. This has many advantages, the main point being to put the website online and the second one to have a recognizable domain name.

7.2. FRONTEND

The frontend has to render information and provide functionality to the user in the friendliest way as possible. For this to be possible various techniques, methods and resources have to be used. First of all, a user research has been done with the objective of analysing what are NaranAir airport managing system potential users and their requirements (this user research can be seen in the *section 10.3*). After that, Mockups have been designed in order to provide the webpage of the best possible interface (this mockups can be seen in the *section 10.4*). Then once the webpage has reached its first complete version it will undergo several usability and user experience testing so that it can be redesigned and improved. Interactivity can also be achieved through the use and implementation of several JavaScript libraries.

The resources used, apart from HTML5 and CSS, to implement all this from the frontend are the following explained in the *sections 7.2.1 7.2.2 7.2.3 7.2.4*.

7.2.1. JQUERY

Being the most used lightweight ajax library is used in NaranAir as such. Utilized to update the web content dynamically from the client side. It is also used to interact with the backend in order to gather data from the backend. Providing dynamic update without page refresh.

Almost any event that can be triggered on the webpage and some validation functions (username availability check) are done through JQuery.

7.2.2. BOOTSTRAP

Bootstrap is the frontend framework used to enhance the UI (User Interface) of the page, making it more visually appealing, interactive and responsive.

In the next Image 17 and Image 18 can be reflected the responsive NaranAir UI (User Interface) design.

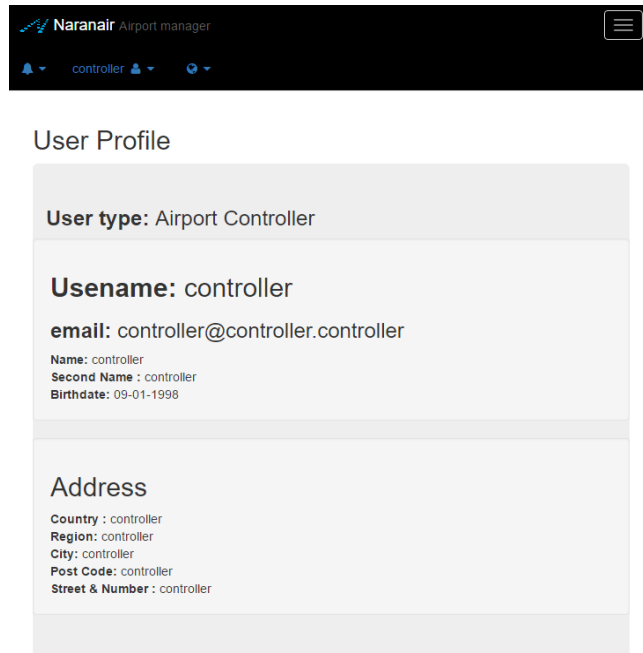


Image 17: NaranAir responsive Tablet size.

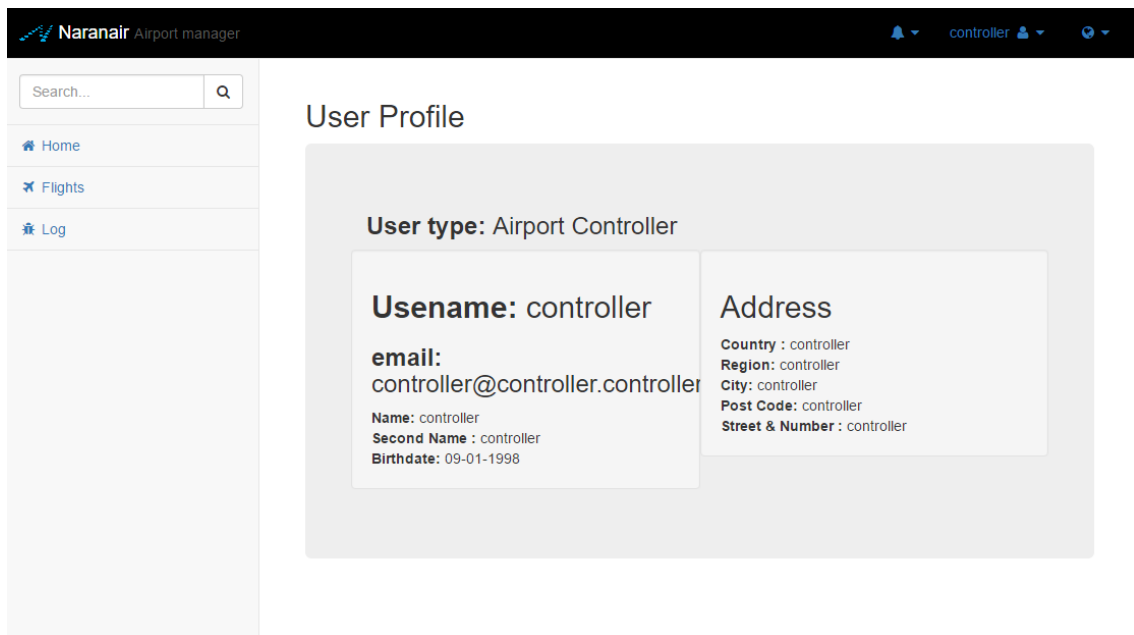


Image 18: NaranAir responsive Web size.

7.2.3. SOCKETIO

Library used to trigger events on the client side directly from the backend ([Struts2](#)). This library provides the optimal RTD (Real Time Data) implementation on the webpages. The alternative being the use of timer-like loops to fetch data every time, SocketIO makes the data updates selective (only the data that has changed is sent) and triggered (the data is sent when the change happens and not constantly).

7.2.4. VISUAL LIBRARIES

7.2.4.1. C3

Used for graphical visualization of data on the webpage also provides interactive components. Data feed comes from the server and graphics are interactive and integrated into the webpage. Also, is directly built on top of the well-known D3.js library enabling plenty options while coding simpler than on D3.js. This provides the perfect balance between functionality and development time invested into the rendering of the data. In the *Image 19* can be seen a little example of the library used in the NaranAir airport managing system.

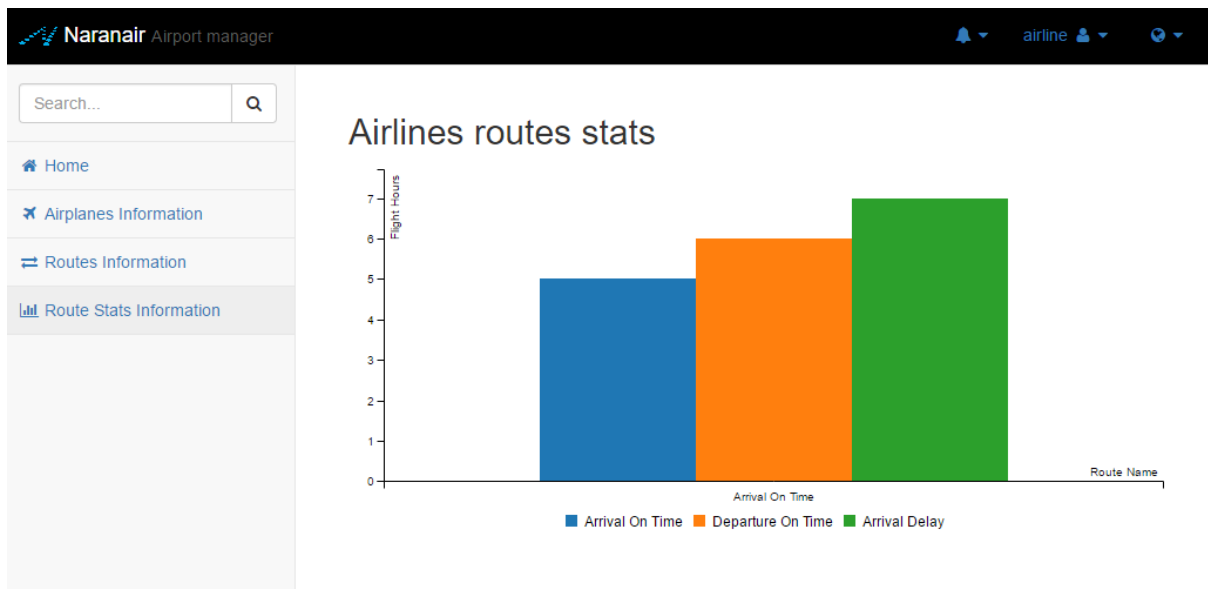


Image 19: C3 Route Stats bar chart.

7.2.4.2. DATATABLES

Used in order to provide tables with data to the user that can be dynamically rendered and filtered. The data from this tables is fetched from the server depending on the requested entries of the table. So, in case a user is looking flights to Germany, for instance, and the flights table contains 100.000 entries of which 10.000 are to Germany but the page only renders 20 at a time the user will only have to load those 20 flights. This, increases both server-side and client-side processing and network optimization, an example can be find in the *Image 20*.

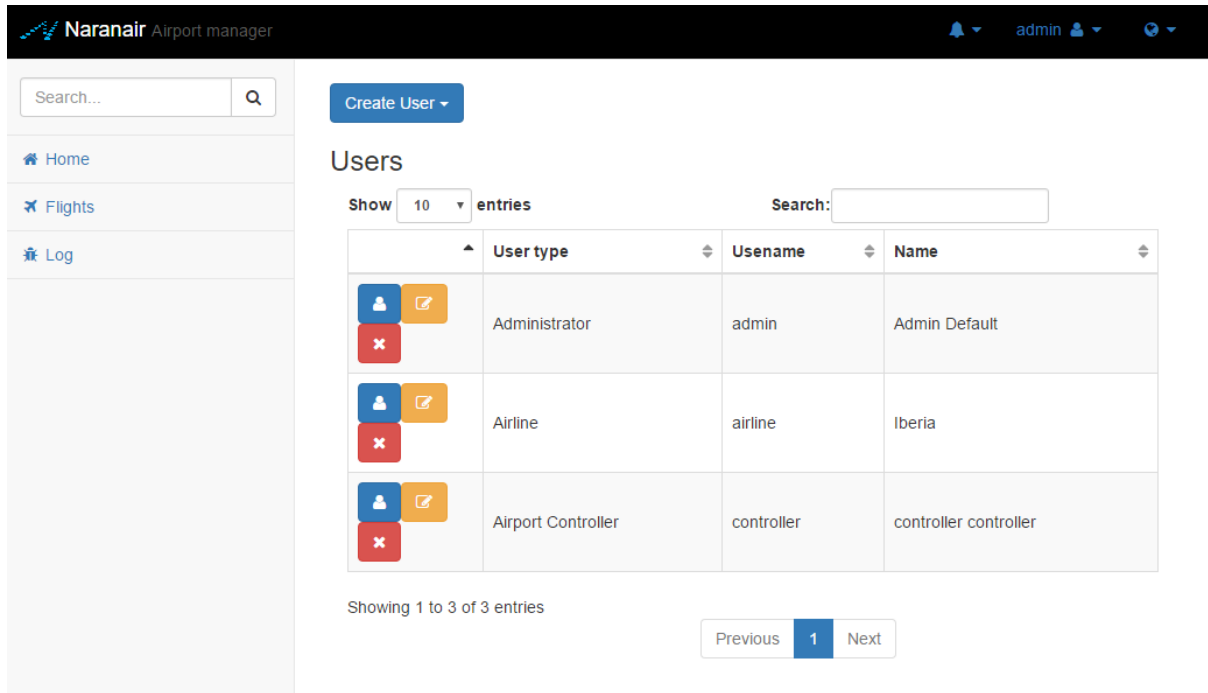


Image 20: DataTables Account Manager.

7.2.5. HTML5

Although most of the source HTML code is embedded in [JavaServer Pages](#) and is therefore generated dynamically each time a page is requested on the server. All the html content generated must meet [W3C standards](#) and although this statement cannot be validated 100% no generated HTML has failed the testing. This testing is done by making randomly selected pages HTML code undergo the [W3C HTML validator](#).

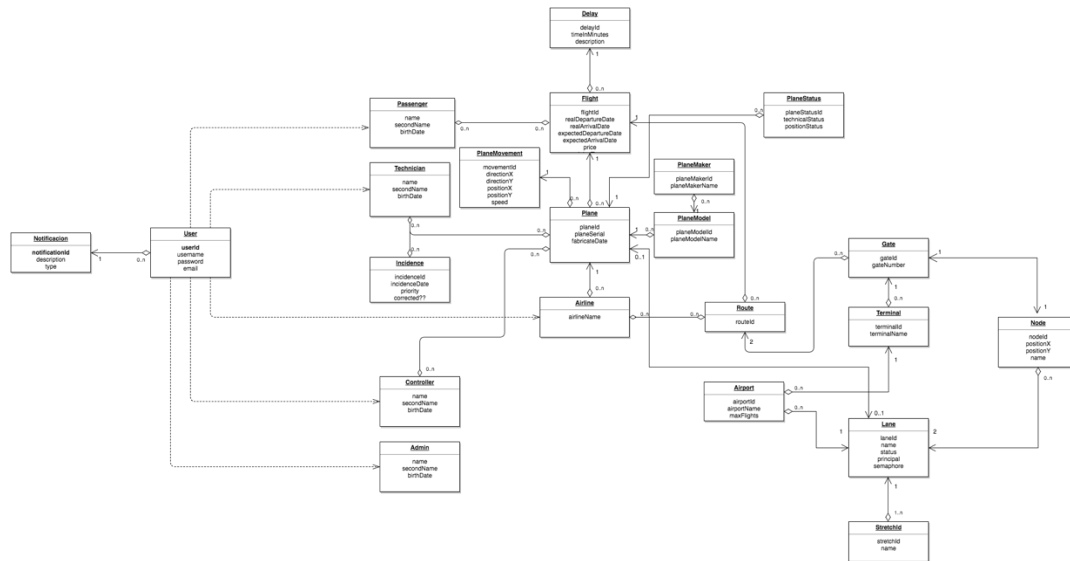
7.2.6. CSS3

Provides styling to the webpage. Most of it, comes from the Bootstrap CSS and a custom layer implemented over it apart from some more custom CSS for specific components such as the console log window.

8. DATABASE

8.1. ENTITY RELATIONSHIP MODEL

NaranAir airport managing system data storage is done using a [PostgreSQL](#) ^[25] object-relational database based on the next entity relational model or diagram shown in the *Image 21*.



**Note: The ER Model image is added to this document in a ZIP file.*

^[25] PostgreSQL is an object-relational database (ORDBMS) with additional “object” features.

Firstly, it was considered using a NoSQL^[26] database apart from [PostgreSQL](#) database due to the need of working with real time data and taking into account that NoSQL databases are faster than other SQL databases such as [PostgreSQL](#). For that, the most used NoSQL databases were analysed and in the first instance it was decided to use [MongoDB](#)^[27] document-oriented database. The main purpose of this, was to save the plane movements or positions in real time of the planes that are fighting and reflect all that in the controller interfaces maps. For all of these reasons, the speed was a key point when storing the data, as long as, it was important also maintain the data integrity of the [PostgreSQL](#) database.

^[26] NoSQL is referred to “non-relational” or “not only SQL” databases, that they provide a mechanism for storage and retrieval of data which is modelled in means other than the tabular relations used in relational databases.

^[27] MongoDB is a free and open-source cross-platform document-oriented database.

Despite that, using [MongoDB](#) would require spreading out NoSQL databases and more concretely [MongoDB](#) knowledge, because it was something unknown, so, new alternatives were considered and finally it was decided to use an unique [PostgreSQL](#) database where every plane movement is saved but not every position, by means of this fact, data speed is not as much a crucial point as before.

Regarding to the data stored in the [PostgreSQL](#) database, all tables and attributes saved are shown in the *Image 21*. As can be seen, there are some singularities, due to the fact that the movements of the planes inside the airport must be managed in order to avoid conflicts and collisions between them, “Lane”, “Node” and “Stretch” database tables and a [Java](#) algorithm. These database tables they mean that, all airport lanes are composed with a starting and ending node, and where one or more lanes are joined a stretch through which a plane can move around is created.

8.2. POSTGRESQL AND JAVA CONECTION

The connection between [PostgreSQL](#) and [Java](#) is performed by [Hibernate](#) framework. This, offers a lot of benefits in comparison with other tools used with the same aim, such as, the well-known and very used before [JDBC API](#). In case of Hibernate, is database independent, that is, supports a large list of database and it provides a powerful query language (HQL) easy to use, similar to SQL and compatible with any database server. These makes the application very portable and flexible, changing database server will not require for any change in the code.

Apart from that, [Hibernate](#) translates every [Java](#) entity persistence class into a database table and creates the pertinent relationships between them, that is, using the Hibernate tagging system with annotations, it takes charge of translating all that information into database tables and relations information. So, in a certain mode, [Hibernate](#) creates the database script automatically saving work to the developer. An example of this can be seen in the *Image 22*.

```
1 package domain.model;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Airport{
7     @Id
8     @GeneratedValue
9     Integer id;
10
11     String name;
12
13     Integer maxFlights;
14
15     @ManyToOne(optional = false)
16     Address address;
17
18     public Integer getId() {
19         return id;
20     }
21
22     public void setId(Integer id) {
23         this.id = id;
24     }
25
26     public String getName() {
27         return name;
28     }
29
30     public void setName(String name) {
31         this.name = name;
32     }
33
34     public Integer getMaxFlights() {
35         return maxFlights;
36     }
37
38     public void setMaxFlights(Integer maxFlights) {
39         this.maxFlights = maxFlights;
40     }
41
42     public Address getAddress() {
43         return address;
44     }
45 }
```

Image 22: Java Persistence class

Other Hibernate functionality used on the NaranAir airport managing system is [Hibernate](#) inheritance, this is, [Hibernate](#) maps the Java inheritance based on three different strategies:

1. A single table to store the entire class hierarchy. It has the advantage of being the best performance option, since it is only necessary to access a table.
2. A table for the hierarchy father with the common attributes and another table for each child class with the concrete attributes. It is the most normalized option and very flexible when new tables and attributes have to be added.
3. An independent table for each type. In this case, each table is independent from the others but the father attributes are repeated in each table. This strategy can have serious performance problems, if we are working with polymorphism.

Taking into account all the strategies explained before, NaranAir airport managing system has decided to use the first functionality in order to implement the user's inheritance, as it is shown in the *Image 23*.

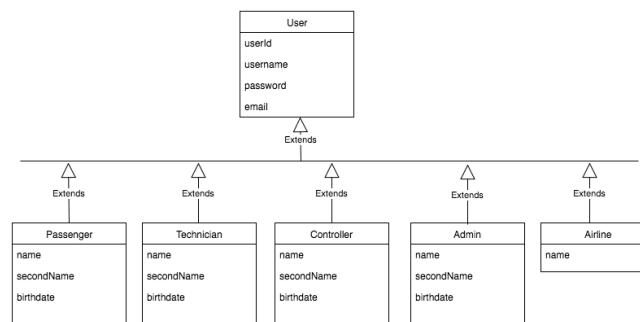


Image 23: User class inheritance

That is translated by [Hibernate](#) to a unique database table with the next structure shown in the *Image 24*.

AppUser	
PK	<u>id</u>
	email
	password
	username
	name
	birthdate
	secondname
	dtype

Image 24: User class inheritance translate to database



The *dtype* column will be the column reference to the type of user (*Passenger, Technician, Controller, Admin or Airline*), in each case the remainder columns are filled depending on the type of user registered on the *dtype* column.

8.3. NOTIFICATIONS

NaranAir airport managing system has the aim of improving the airport user experience and the communication between all them, for that, it has been thought in a notification system.

This notification system, set ups the opportunity for each user to know every moment what is going on in the airport. That is, every user will receive in the web page all the latest information relevant for it when it is log inside the system. For instance, controller will be notified when a plane asks permission to land or take off or a passenger will be notified when one of its flights is going to take off early.

For the implementation of this functionality, the PostgreSQL database has a great weight, has to be able to notify the Java application when some change instructions are executed inside the database. This fact management will be performed by some Triggers which execute the pertinent procedures. This procedures will have a special characteristic, will communicate with the Java program by [PostgreSQL pg_notify\(\)](#) function. This function, sends a notification event together with an optional “payload” string to each client application that has previously executed LISTEN channel for the specified channel name in the current database. The LISTEN channel execution is performed by the Java application using [SocketIO](#) ^[28] JavaScript library, this works as follows: creates a connection with the database and sets the same LISTEN channel as the one specified in the *pg_notify()* function, by this, all the notifications sent from the database are listened on the Java application backend and finally are sent to the Web Application frontend where the JavaScript will deal with it.

^[28] *Socket.IO is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers.*

The same procedure is followed for filling the administration controller log, but in this case the database is not an intermediary on the real time bi-directional communication. The communication is done between the backend and frontend of the Web Application.

**Note: This functionality is not implemented in the first NaranAir airport managing system.*

8.4. BACKUP STRATEGY

The backup strategy followed is the simplest possible, firstly a batch script was created in order to make a backup of the database and after that compressing that file. After that, [pgAgent](#) was installed, with the purpose of executing that script each day at a known time. But, due to some issues with [pgAgent](#), finally, Windows Task Manager has been used to execute the script.

8.5. TABLESPACE STRATEGY

NaranAir airport managing system Tablespace Strategy first idea was to add a tablespace for the plane movements, because this table is changed a lot of times in a short period of time and having a tablespace stored in a faster disk would increment the quality of the database, but, finally, as the server has not any faster disk, it does not make sense implementing it as it won't improve the speed.

8.6. USER STRATEGY

NaranAir database user strategy followed is the simplest that could be, using an own super user that will access to the database. This strategy is not the best way, in fact, the first idea was using different users, each one with their permissions defined previously, and depending on the type of the user registered in the webpage, change the role access to the database. However, NaranAir airport managing system uses the same database connection in both simulator and Web applications, and, as they are running simultaneously, the access to the database must have, at least, all the permissions that the simulator needs (insert, select and delete).

8.7. PROCEDURES

NaranAir airport managing database makes use of only one procedure which is used to select free dates in the schedule where all the timetable of the flights is stored. This procedure, checks the timetable looking for an empty space, taking into account the maximum number of flights that can be on the airport and returns the date in which a flight will arrive or departure.

Actually, there are three different versions of the procedure. One of them checks the number of flights taking into account the airport where their destination or origin is, another one checks just the number of flights, and the last one is the same as the second one but with the difference that it starts counting the flights from the start of the day, and not from the current date. This last version is only used for the demo.

Apart from this procedure no more is used in NaranAir database, that is due to the reason that every other operation needed to perform in the system can be performed only using HQL queries making the system more flexible and portable to use with other



databases. For that same reason, using a lot of procedures in the system would be a bad weight for the system, because for the application would not lead to any change but the procedures would have to be changed to use it with other database systems.

8.8. SECURITY

The user's confidentiality as well as the Web Application security is a key point for the NaranAir airport managing system. For that reason, the registered user's password information is saved on the database using the MD5 algorithm, by this, the password is no longer saved as plain text and is avoided the access to it by any database user.

Also, the [SocketIO](#) broadcast targets are hashed versions of the types of receptor of those messages, so that, not all the users receive all the messages and confidential messages can't be leaked unless the hash code is known. The hashes are MD5 as the previous ones and the private keys in use now are one for the user and another one for the role. So, for instance, an admin named Joe will be listening to the hashed version of (RandomSaltValue + username) and (RandomSaltValue + roleName).

Finally, query management is done in a way that avoids SQL injection attacks. This is especially useful in situations in which a query is performed with some user input that has not been checked.

9. CONCLUSIONS

After finishing the whole project NaranAir airport managing system has drawn some important conclusions that are explained in the following lines.

It must be stated that the project did not fully meet all the objectives that were set at the beginning of the project. The maintenance technician's interface was finally not implemented, neither were its functionalities, both in the simulator and in the web. The deployment of the application was not set as an objective although it was performed because the conclusion of it being paramount for the project's approach.

Also several factors played a detrimental role in the development, the realization of a second smaller project in parallel to this by the Maracars development team and the absence of a team member during almost a week due to an illness.

In order to measure the quantity of the objectives, the SRS document that was written in the analysis part of the project has been revised, checking how many requirements were fulfilled. Around the 60% of the requirements were implemented during the time that the project has lasted. The result is a bit deceptive as the most important requirements are correctly implemented, and the ones that aren't were requirements that add value to the application but were not essential.



The expected impact of the project can be met with some weeks of further development. Although all this requirements were not met, a lot of them were, such as the multi user interface, responsive design, simulator functionality, deployment, real time updates, maps, security, database implementation, tested code, code documentation, online CI, W3C standards, backups, DNS, data rendering and documented analysis and design.

10. ANNEX

10.1. SOFTWARE SPECIFIC REQUIREMENTS

10.1.1. INTRODUCTION

This section gives a short description and overview of everything included in the SRS document.

10.1.1.1. PURPOSE

The purpose of this document is to give a detailed description of the requirements for the NaranAir airport managing system software. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications. This document is primarily intended to be proposed to a customer for its approval and a reference for developing the first version of the system for the development team.

10.1.1.2. SCOPE

The NaranAir airport managing system is a Web application which helps to manage an airport in order to support future demands and help airport managers to balance the challenges of increasing operational efficiency and reducing operational costs. For that the system is focused in resolving two main problems:

1. Aviation operation stress: The controller job entails a high degree of responsibility and pressure, and that is highly dangerous and can result in what is known as “acute episodic incident” or a near mid-air collision. So, trying to solve this problem the system will provide an automatic simulator of the aviation operation to the air controllers, helping them to make the right decisions at the right time to keep things moving.
2. Communication: Communication between all airport users is crucial inside the airport for improving passenger experience and airport employee’s relationship, so the system will provide all useful information visualization to all airport users in order to achieve these goals.

The system will be designed based on responsive design patterns, so it will be suitable and usable in different devices, such as, smartphone, tablet and laptops.



10.1.1.3. OVERVIEW

This document includes three chapters and appendixes: introduction, product general description and the specific requirements of the system.

Firstly, in this first chapter provides a general overview of the SRS document with an introduction.

Secondly, in the second chapter, the NaranAir airport managing system product general description is explained.

Finally, in the last chapter, there will be explained the different specific requirements of the system in a detailed way.

10.1.2. OVERALL DESCRIPTION

This section will give an overview of the whole system, for that different concepts will be explained, firstly, the product perspective, then, the product or system functions, user's characteristics, restrictions, and finally future requirements.

10.1.2.1. PRODUCT PERSPECTIVE

The system Web application will be divided in two main parts divided in different interfaces: one for the managing of the airport and other for the information. The managing interfaces will be used for managing all the information of the airport as a whole and the information interfaces will be used for displaying all the useful information to the airport users.

Since this is data-centric product it will need somewhere to store the data. For that, a database will be used, one for storing real-time data and normal data in which the process time is not crucial.

10.1.2.2. PRODUCT FUNCTIONS

With the managing interfaces, the users will be able to control all the transcendental information about the airport, such as, flights, airplane state, etc. These functionalities will be changed depending on the type of user, for that, a login interface will be provided in order to redirect the users to an interface or another and provide them more or less functionalities or options.

On the other hand, with the information interfaces the airport users will be able to visualize all the flights information of the airport during the day, providing them, useful information, such as, flight timetable, delay, airline information, gate number, etc.



In addition, the final version of the product will have a simulator integrated which will create flights automatically, move the planes, do the functions of the airport controller (ensure the number of flights, give permissions, etc.) and also simulate the revision of the planes. This will ensure that the product will work correctly when implementing in a real airport.

10.1.2.3. USER CHARACTERISTICS

There are five type of users that interact with the system: airport controllers, flight passengers, airport airlines, maintenance technicians and airport users. Each of these are five type of users has different use of the system so each of them has their own requirements.

The airport controllers will use the system in a laptop and they will be able to monitor the flights and visualize information about them, also they will be able to take decisions about landings and take-offs, for that they will need to login the system.

The flight passengers will have the chance to visualize information about the flight when they login the system.

The maintenance technicians will be able to manage airplanes, so they can visualize their information, state and they can take decisions about them, if there is an incidence, if an incidence is solved, etc. For that, they will need to login the system.

Finally, the airport users will be able to visualize information about the flights of the airport during the day (timetable, origin, destination, delay, airline). Also, they will be able to buy a ticket, if they buy a ticket or flight the system will give them a username and password to login the system and view the information about the bought flight.

10.1.2.4. RESTRICTIONS

The NaranAir airport managing system is based on the Heathrow airport (London), so is suitable only to use in a unique airport. Taking into account that, the system is designed based on Heathrow airport statistics and data, so there will be managed 4 different terminals, with one landing lane and another take off lane as shown as in the next picture:



10.1.2.5. ASSUMPTIONS AND DEPENDENCIES

All these functionalities that are offered by NaranAir airport managing system they have dependencies to various factors. The system depends on the network connectivity, so if there it is not network connectivity the system cannot work and airport managing gets difficult to control, also, if the network is not operative the connection to both database cannot be done, so all the transcendental data gets inaccessible.

10.1.3. SPECIFIC REQUIREMENTS

1. Controller requirements

- **SR01:** Airplanes monitoring by map.
- **SR02:** Airplanes monitoring in airport terminals.
- **SR03:** Flights visualization with a filter based on different characteristics.
- **SR04:** Flight information visualization
 - Passengers information
 - Real time location
- **SR05:** Gate/Terminal/Lane information visualization.

2. Passenger requirements

- **SR06:** Flight information visualization.
 - Gate number.
 - Flight number.
 - Airline information.
 - Timetable information.
- **SR07:** Option to buy tickets.

3. Airline requirements

- **SR08:** View information about routes.
- **SR09:** Create new routes.
- **SR10:** View statistic results information:
 - Passengers quantity by flight.
 - Number of hours done flying.
- **SR11:** Airplanes manage
 - Add to new routes.
 - Add new airplanes.
- **SR12:** View employee's information.

4. Airport user requirements

- **SR13:** Visualize flights information.
 - Timetable.
 - Gate number.

10.2. USE CASES

10.2.1. CONTROLLER USE CASES

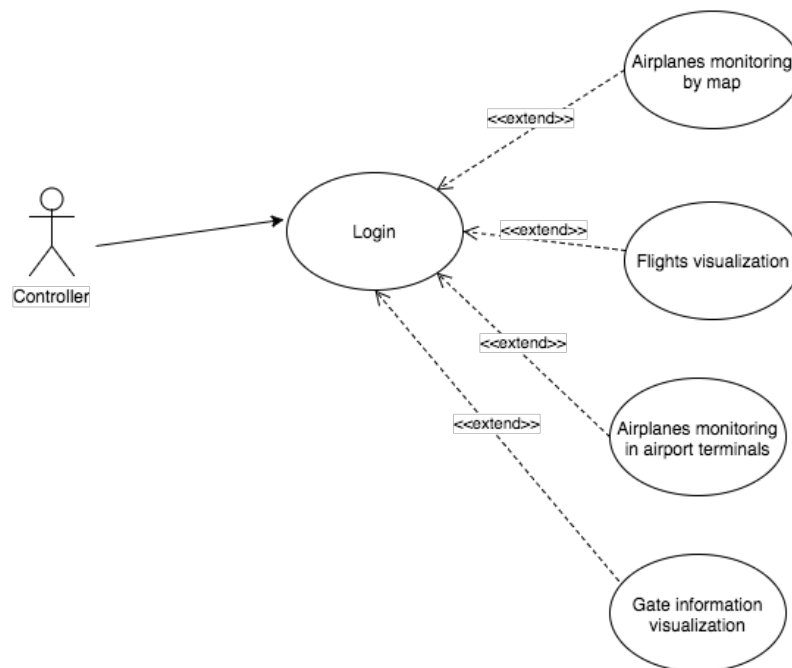


Image 25: Controller use cases

10.2.2. PASSENGER USE CASES

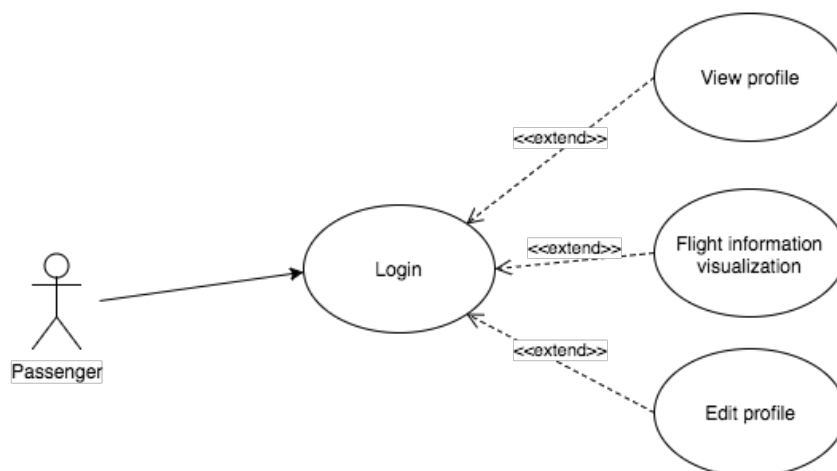


Image 26: Passenger use cases

10.2.3. AIRLINE USE CASES

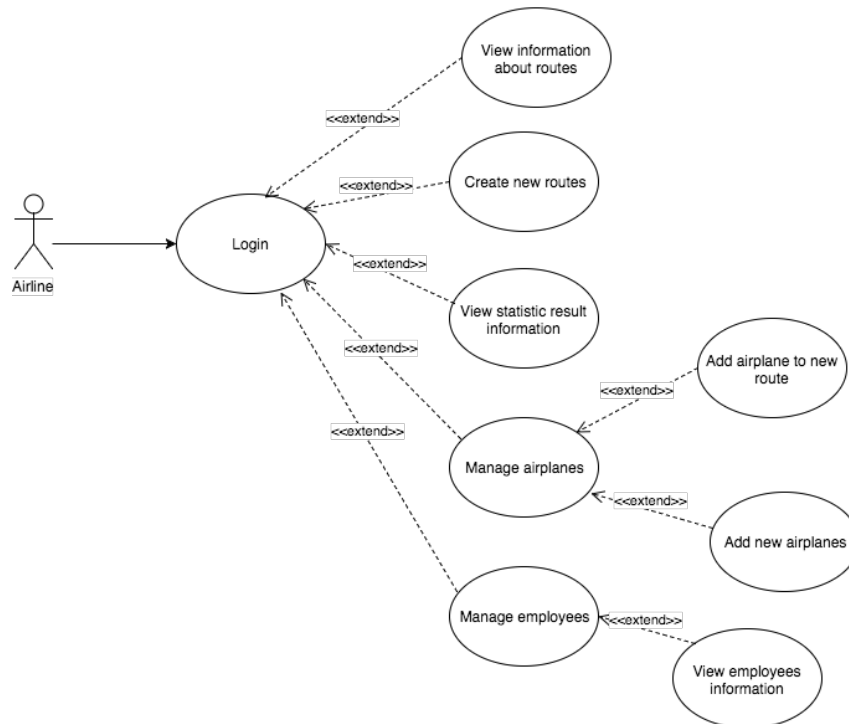


Image 27: Airline use cases

10.2.4. AIRPORT USER USE CASES

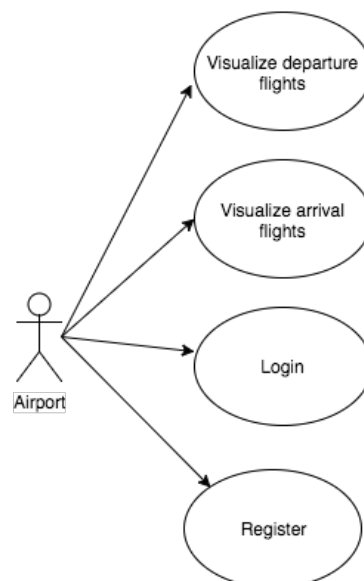


Image 28: Airport use cases

10.2.5. TECHNICIAN USE CASES

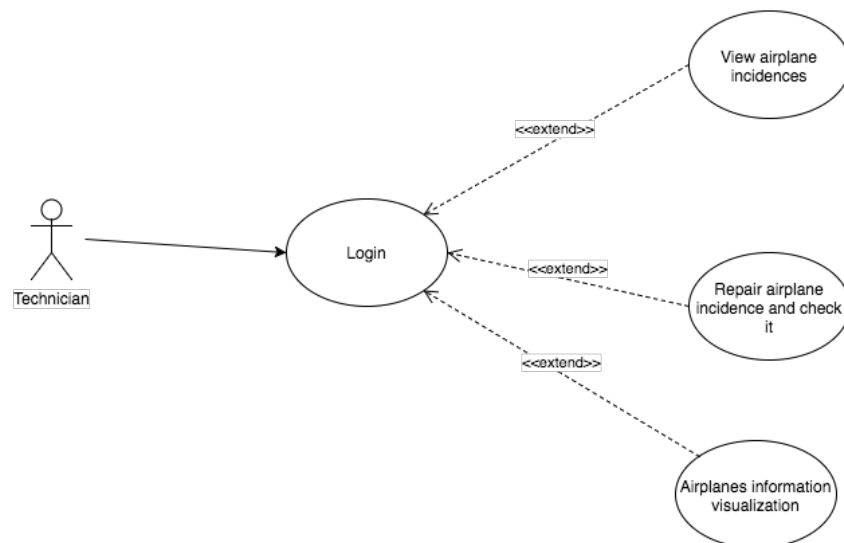


Image 29: Technician use cases

10.2.6. ADMINISTRATOR USE CASES

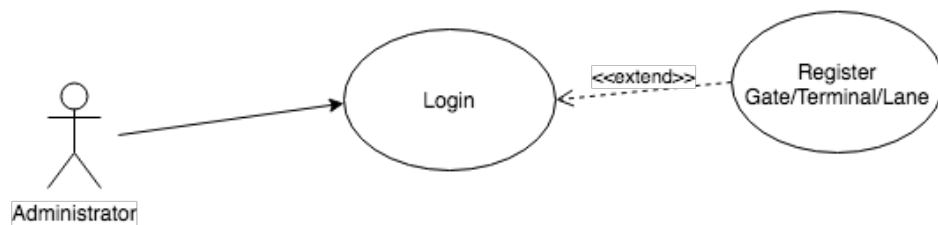





Image 30: Administrator use cases

**Note: The use cases scenarios are added to this document in a ZIP file.*


10.3. USER RESEARCH


Persona:	Airport Controller
Photo:	
Name:	Dave Marshall
Job Title:	Airport Controller in Heathrow's airport
Demographics:	<ul style="list-style-type: none"> • 51 years • Married • Father of 2 children • Has a Ph in Chemical Engineering
Goals:	<p>He has to be focused on controlling all the planes that are near the airport of Heathrow.</p> <p>For that he needs several requirements:</p> <ul style="list-style-type: none"> • A view of where the planes are located (map) • Information of the flights (departure/arrival time, gate num...) •
Skills:	He is comfortable using a computer but he is not an expert on it.

Persona:	Passenger
Photo:	
Name:	Marcelina Huertas
Job Title:	School teacher
Demographics:	<ul style="list-style-type: none"> • 44 years • Married • Mother of 1 children • Degree in Education
Goals:	She is going on a trip to Madrid and she wants to view the information about her flight (departure/arrival time, gate number...)
Skills:	She doesn't have a computer with her, she uses the mobile phone but she is not a skilled user.

Persona:	Airline Manager
Photo:	
Name:	Julienne Curran
Job Title:	Turkish Airlines Manager
Demographics:	<ul style="list-style-type: none"> • 37 years • Single • Master in Administration
Goals:	<p>Her aim is to manage the company as well as possible. For that, she wants some requirements:</p> <ul style="list-style-type: none"> • View statistics of flights. • View all the routes available (possibility to add or remove). • Manage the planes (add and remove them to the database, view information...).
Skills:	She is an expert using all kind of devices, but he usually uses a tablet.

Persona:	Airport (external user)
Photo:	
Name:	Mateo Prats
Job Title:	Retired
Demographics:	<ul style="list-style-type: none"> • 68 years • Married • Father of 3 children • Grandfather of 2 nieces
Goals:	He is going to pick up his son and nieces to the airport and he wants to know the timetable of the airport (see when the flight arrives)
Skills:	He is not a skilled user in any device.

Persona:	Maintenance Technician
Photo:	
Name:	Benjamin Ayivorh
Job Title:	Heathrow airports maintenance technician.
Demographics:	<ul style="list-style-type: none"> • 32 years • Single • Aerospace engineering degree.
Goals:	He has to ensure that all the planes are functional before they take off. For that, he needs to see all the incidences?? That have been reported, and also to add new ones. He also needs to view some information about the airplanes, such as how many hours of flying they have...
Skills:	He is an expert using all kind of devices, but he usually uses a tablet.

Persona:	Airport Administrator
Photo:	
Name:	Anuj Bhardwaj
Job Title:	Heathrow airport manager.
Demographics:	<ul style="list-style-type: none"> • 43 years • Married • Business Master
Goals:	He describes his role as the conductor of the orchestra. He has to manage all the airport, the workers, the gates and the terminals, negotiate with new airlines...
Skills:	He is an expert using all kind of devices.

10.4. MOKCUPS

**Note: NaranAir airport managing system mockups are added to this document in a ZIP file.*

10.5. MODEL VIEW CONTROLLER DIAGRAMS

**Note: NaranAir airport managing system web application model view controller diagrams are added to this document in a ZIP file.*



10.6. CODING STANDARDS

CSS

- Prohibit !important
- Prohibit duplicate properties
- Prohibit empty rules
- Prohibit syntax errors

General Conventions

- A check for TODO comments
- Check for ensuring that for loop control variables are not modified inside the for block
- Check nested (internal) classes/interfaces are declared at the bottom of the class after all method and field declarations
- Checks for empty blocks
- Checks for long anonymous inner classes
- Checks for long lines(150)
- Checks for long source files
- Checks for magic numbers
- Checks for multiple occurrences of the same string literal within a single file
- Checks for redundant modifiers in interface and annotation definitions
- Checks that a token is surrounded by whitespace
- Checks that class which has only private constructors is declared as final
- Checks that each top-level class, interfaces or enum resides in a source file of its own
- Checks that each variable declaration is in its own statement and on its own line
- Checks that long constants are defined with an upper ell
- Checks that there is a newline at the end of each file
- Checks that there is only one statement per line
- Checks the number of methods declared in each type Details
- Checks the number of parameters that a method or constructor has
- Checks the padding of parentheses
- Checks the padding of parentheses for typecasts
- Controls the indentation between comments and surrounding code
- Detects empty statements (standalone ';')
- Finds nested blocks
- Restricts nested boolean operators (&&, ||, &, | and ^) to a specified depth (3)
- Restricts nested for blocks to a specified depth (2)
- Restricts nested if-else blocks to a specified depth (default = 1)
- Restricts nested try-catch-finally blocks to a specified depth (default = 1)
- Restricts return statements to a specified count (2)
- Restricts throws statements to a specified count (4)
- This metric measures the number of instantiations of other classes within the given class
- Throwing java.lang.Error or java.lang.RuntimeException is almost never acceptable



JAVA

- Avoid catch blocks that merely rethrow a caught exception.
- Avoid code containing duplicate String literals
- Avoid instantiating String objects.
- Avoid negation in an assertTrue or assertFalse test.
- Avoid throwing NullPointerExceptions.
- Avoid throwing general exception types.
- Avoid unused import statements.
- Avoid using '==' or '!=' to compare strings.
- Avoid using System.out|err).print.
- Enforce class naming convention to CamelCase
- Enforce equals() to compare objects
- Enforce method naming convention camelCase
- Enforces constants as static final
- Enforces default label to be last on switch
- Enforces each method to not have too many lines of code (30)
- Enforces field declarations at top of the class
- Enforces opposite operator instead of negating
- Enforces package definition
- Enforces small and clear methods(50)
- Enforces the care you should take when looking at class file lengths(750)
- Enforces the declaration of one variable per line
- Enforces the moderate usage of parameters in methods(5)
- JUnit tests should include at least one assertion.
- JUnit assertions should use the three-argument version of assertEquals().
- Prohibits Empty Catch Blocks
- Prohibits branch statement at last part of loop
- Prohibits empty If Statements
- Prohibits empty finally blocks
- Prohibits empty statements that are not inside a loop
- Prohibits empty try blocks
- Prohibits fields when local variable is enough
- Prohibits jumbled loop incrementers
- Prohibits multiple if statements that can combine
- Prohibits non-static initializer block
- Prohibits reassigning values to parameters
- Prohibits returning from a finally block
- Prohibits switch statements without break
- Prohibits switch statements with no default
- Prohibits the implementation of an empty Finalizer
- Prohibits the implementation of empty block statements
- Prohibits the use of Floats for loops
- Prohibits the use of empty switch statements
- Prohibits the use of empty synchronized blocks
- Prohibits the use of unnecessary return statements
- Prohibits unnecessary comparisons in boolean expressions
- Prohibits unsafe static field
- Prohibits unused local variables
- Prohibits unused parameters
- Prohibits unused private fields
- Prohibits unused private methods



- Prohibits useless class
- Prohibits useless if statements
- Prohibits while loops without braces
- Some JUnit framework methods are easy to misspell.
- Unnecessary JUnit test assertion with a boolean literal.
- Use StringBuffer.length() to determine StringBuffer length
- Web applications should not call System.exit().

JavaScript

- Enforce camelCase
- Enforce Curly Brace Conventions
- Enforce coherent Declarations
- Enforce spacing around *
- Enforce Semicolons
- Prohibit Assignment in Conditions
- Prohibit Duplicated Keys
- Prohibit Extra Semicolons
- Prohibit Function Assignment
- Prohibit Functions in Loops
- Prohibit Iterator
- Prohibit Redeclaring Variables
- Prohibit Spaces in Regular Expressions
- Prohibit Switch-Case Fallthrough
- Prohibit Sparse Arrays
- Prohibit Undeclared Variables
- Prohibit Unreachable Code
- Prohibit Use of undefined variable
- Prohibit Variables Deletion
- Prohibit console messages
- Prohibit constant expressions in conditions
- Prohibit debugger
- Prohibit duplicate arguments
- Prohibit duplicate case label
- Prohibit duplicate name in class members
- Prohibit empty Block Statements
- Prohibit extra boolean casts
- Prohibit invalid typeof comparisons
- Prohibit irregular whitespace
- Prohibit malformed Regular Expressions
- Prohibit mixed spaces and tabs
- Prohibit modifying class declarations
- Prohibit modifying const
- Prohibit out of scope variables
- Prohibit the use of variables before defining
- Prohibits Invalid Regular Expressions
- Prohibits high Cyclomatic Complexity(4)
- Quotes Style
- Require Constructors to Use Initial Caps
- Require parentheses in arrow function arguments
- Require space around arrow function's

REFERENCES

- Apache. (1999-2016). *Apache Tomcat*. Retrieved from Apache Tomcat:
<http://tomcat.apache.org/tomcat-8.0-doc/index.html>
- Atlassian. (2016). *SourceTreeApp*. Retrieved from SourceTree:
<https://www.sourcetreeapp.com/>
- Bootstrap. (n.d.). *Bootstrap*. Retrieved from Bootstrap: <http://getbootstrap.com/>
- Bostock, M. (2015). *D3*. Retrieved from D3. Data Driven Documents: <https://d3js.org/>
- C3. (n.d.). *C3js*. Retrieved from C3: <http://c3js.org/reference.html>
- Codacy. (2016). *Codacy*. Retrieved from Codacy automated code review:
<https://www.codacy.com/>
- Codecov. (2016). *Codecov*. Retrieved from Codecov: <https://codecov.io/>
- Confluence. (n.d.). *Confluence*. Retrieved from Apache Struts 2 Documentation:
<https://cwiki.apache.org/confluence/display/WW/Home>
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms* (Second Edicion ed.). S&P Global.
- Django. (2005-2016). *Django*. Retrieved from Django documentation:
<https://docs.djangoproject.com/en/1.10/>
- Eclipse. (2016). *Eclipse*. Retrieved from The Eclipse Foundation open source community website: <https://eclipse.org/>
- Git. (2016). *Git*. Retrieved from Git. Local branching on the cheap: <https://git-scm.com/>
- GitHub. (2007, 10). *GitHub*. Retrieved from GitHub: <https://github.com/>
- Hg-Git. (n.d.). *Hg-Git.GitHub*. Retrieved from The Hg-Git mercurial plugin: <http://hg-git.github.io/>
- Hg-Git. (n.d.). *Hg-Git.GitHub*. Retrieved from The Hg-Git mercurial plugin: <http://hg-git.github.io/>
- JQuery. (2016). *JQuery*. Retrieved from JQuery. Write less, do more.
- Mercurial. (n.d.). *Mercurial*. Retrieved from Project of the Mercurial community:
<https://www.mercurial-scm.org/>
- Microsoft. (2016, 10 14). *Microsoft*. Retrieved from Introduction to ASP.NET Core:
<https://docs.microsoft.com/en-us/aspnet/core/>
- Microsoft. (2016). *Microsoft*. Retrieved from Microsoft:
<https://www.microsoft.com/es-es/>
- MongoDB. (2016). *MongoDB*. Retrieved from MongoDB:
<https://www.mongodb.com/es>
- OmniPlan. (2016, 2 1). *OmniPlan*. Retrieved from OmniPlan:
<https://www.omnigroup.com/omniplan>
- OpenLayers. (n.d.). *OpenLayers*. Retrieved from OpenLayers: <https://openlayers.org/>
- Oracle. (n.d.). *Oracle*. Retrieved from Java EE:
<http://www.oracle.com/technetwork/java/javasee/overview/index.html>
- Oracle. (n.d.). *Oracle*. Retrieved from Java: <https://www.java.com/es/>
- Oracle. (n.d.). *Oracle*. Retrieved from JDBC:
<http://www.oracle.com/technetwork/java/javase/jdbc/index.html>



PostgreSQL. (2009-2013). *PostgreSQL*. Retrieved from PostgreSQL:
<http://www.postgresql.org.es/>

Python. (n.d.). *Python*. Retrieved from Python: <https://www.python.org/>

Redhat. (n.d.). *Hibernate*. Retrieved from Hibernate: <http://hibernate.org/>

Ruby on Rails. (n.d.). *Ruby on Rails*. Retrieved from Ruby on Rails documentation:
<http://api.rubyonrails.org/>

Ruby. (n.d.). *Ruby*. Retrieved from Ruby: <https://www.ruby-lang.org/es/>

SocketIO. (n.d.). *SocketIO*. Retrieved from SocketIO: <http://socket.io/>

SonarQube. (2008-2016). *SonarQube*. Retrieved from SonarQube:
<http://www.sonarqube.org/>

SpyMedia. (2007-2016). *DataTables*. Retrieved from DataTables. Table plug-in for
jQuery: <https://datatables.net/>

The Apache Software Foundation. (2000-2016). *Apache Struts*. Retrieved from Apache
Struts: <https://struts.apache.org/>

TortoiseGit. (2015-2016). *TortoiseGit*. Retrieved from Windows Shell Inteface to Git:
<https://tortoisegit.org/>

Travis CI. (n.d.). *Travis CI*. Retrieved from Travis CI: <https://travis-ci.org/>

Trello. (2016). *Trello*. Retrieved from Trello: <https://trello.com/home>

W3C. (n.d.). *W3C Standards*. Retrieved from W3C Standards:
<https://www.w3.org/standards/>