

Grau en Enginyeria Informàtica de Gestió i Sistemes d'Informació

BUSINESS ANALYTICS PLATFORM OF INFLUX CONTROL OF PEOPLE IN ENCLOSURES

Memòria

Jan Jiménez Serra
TUTOR: Xavier Font

4rt

Abstract

Security on closed sites is one of, if not the most important concern that owners and managers of enclosures should have in mind. One of the most crucial knowledge concerning the safety of people in a closed space is the occupation relative to the maximum capacity.

This project is about the design, building and implementation a full-stack web application which gives this information in a manner which ensures the best user experience. It shows the real-time occupation of a site, control of the influx of people, and provides business analytics attractive for the client.

Resumen

La seguridad en sitios cerrados es una las preocupaciones más importantes que deben tener en cuenta los propietarios y administradores de recintos. Uno de los conocimientos más importantes sobre la seguridad de las personas en un espacio cerrado es la ocupación relativa a la capacidad máxima.

Este proyecto consiste en el diseño, la construcción y la implementación de una aplicación web completa que proporciona esta información de una manera que garantiza la mejor experiencia de usuario. El producto final muestra la ocupación en tiempo real de un sitio, el control de la afluencia de personas y proporciona análisis de negocios atractivos para el cliente.

Resum

La seguretat en llocs tancats és una les preocupacions més importants que han de tenir en compte els propietaris i administradors de recintes. Un dels coneixements més importants sobre la seguretat de les persones en un espai tancat és l'ocupació relativa a la capacitat màxima.

Aquest projecte consisteix en el disseny, la construcció i la implementació d'una aplicació web completa que proporciona aquesta informació d'una manera que garanteix la millor experiència d'usuari. El producte final mostra l'ocupació a temps real d'un lloc, el control de l'afluència de persones i proporciona anàlisi de negocis atractius per al client.

Table of contents

| | |
|--|----|
| 1. Introduction | 1 |
| 2. Previous Study: context, background and information necessities | 3 |
| 2.1. Background..... | 3 |
| 2.2. Context..... | 5 |
| 2.2.1. Market | 5 |
| 2.2.2. Web Applications..... | 5 |
| 2.2.3. Database | 6 |
| 2.2.4. Backend..... | 7 |
| 2.2.5. Frontend | 7 |
| 2.2.5. Sensors for counting people..... | 8 |
| 2.2.6. Deep Learning..... | 9 |
| 2.2.7 Image recognition | 20 |
| 2.2.8 Gender and age recognition | 24 |
| 2.3. Information needs | 25 |
| 3. Goals and scope..... | 27 |
| 3.1. Goals of the product | 28 |
| 3.2. Goals of the client..... | 29 |
| 3.3. Definition of functional and technological requirements | 29 |
| 3.4. Target or potential audience | 33 |
| 4. Methodology | 34 |
| 4.1. Programming Languages, Libraries and Frameworks..... | 35 |
| 4.1.1 MVC | 38 |
| 5. Development | 41 |
| 5.1. Database..... | 41 |
| 5.2. Sensor Installation and Data Retrieving | 46 |
| 5.3. Backend | 50 |
| 5.3.1. Model | 52 |
| 5.3.2. Repository | 53 |
| 5.3.3. Controller | 54 |
| 5.3.4. Security | 55 |

| | |
|--|-----|
| 5.3.5. Payload | 55 |
| 5.3.6. Security..... | 57 |
| 5.3.7. Software Engineering | 58 |
| 5.4. API..... | 66 |
| 5.5. Frontend | 68 |
| 5.5.1. React General Classes | 69 |
| 5.5.2. Source folder structure | 70 |
| 5.5.3. App.js | 70 |
| 5.5.4. Routing..... | 73 |
| 5.5.5. Layout..... | 75 |
| 5.5.6. Assets | 77 |
| 5.5.7. Components..... | 78 |
| 5.5.8. Login | 79 |
| 5.5.9. API Calls | 80 |
| 5.5.10. Views..... | 81 |
| 5.5.11. Responsiveness..... | 90 |
| 5.6. Deloyment..... | 91 |
| 5.7. Gender and age recognition | 92 |
| 5.7.1. Create the training data | 93 |
| 5.7.2. Training the neural network | 97 |
| 6. Conclusions | 104 |
| 7. Bibliography | 105 |

1. Introduction

There have been many cases of incidents related to the improper control of a place's capacity, such as events where the fast evacuation of people is needed. Not knowing the true occupation of an enclosure can lead to dangerous and sometimes even fatal accidents. This project is aimed to solve this problem in an easy-to-implement, precise, and visual approach.

The project consists of the design and construction of a web application for the control of the capacity, influx and occupation control of people in an enclosure.

The application also presents meaningful business analytics data for the clients, which are the owners, managers or whoever is in charge of this kind of knowledge:

- How many people visit your establishment?
- How many of your visits become sales?
- Do you know the capacity of your business in real time?
- Do you know where customer traffic is concentrated?

Those are questions that the application answers. The client that decides to acquire the product is able to extract data from a period of time and that data be shown in different types of media forms, while also knowing the quantity of people occupying their business place in real time with little percentage of error, ensuring that the occupation of their place is secure at all times.

The application also has an alarm system which warns the users whenever the capacity is near, by a defined percentage, to the maximum capacity.

To acquire the data, it is first needed to sensorize the entrances of the site. There is a need to be flexible with the different kind of inputs that the application has deal with. Usually the input comes from a third-party infrared sensor which already has an interface to define entrances and departures, but sometimes it has to extract data from an IP camera or a thermal camera (so there is better night vision) and use software to define the entrance and exit lines, and image recognition to know how many people cross those lines.

It needs to be said that the goal of the project is not to extract personal data from the public that access the enclosure, but to provide general data for the client's business decision making. In no case the system saves or shares individual's images without their consent.

The application will be developed for NexoTech S.L, a security business focused on videosurveillance. One of the principal goals of the product is to guarantee the safety of the enclosures displaying with precision the occupation in real time. The project is named NexoCount as requested by the company.

Additionally, there is the goal to take the application's business analytics providing further. It is intended to extract interesting information from the cameras like gender and age classification of the people on the site. For that to work, deep learning has to come into the equation, there was a state-of-the-art research of gender and age recognition on video streams. After that, a neural network trained with a dataset is used for the final recognized predictions and posterior storing and displaying of the information extracted.

2. Previous Study: context, background and information necessities

The project was developed with a profound knowledge imported from the work experience.

2.1. Background

Business internship

It is necessary for the context of the project to explain the internship at the company NexoTech, as it is them who asked and sponsored the project and for whom the final product will be for, as they will sell it and distribute it. There was a participation in a series of small projects relevant to the realization of this project:

- Website: Update of the business website. Rebuilding of the website which was outdated and presented high deficiencies, with the intention to improve the user experience and the SEO visibility. The new website was developed with 1&1 CMS. The website can be found at the date 24th of April 2019 at: <https://www.nexotech.es>.
- QlikView: It is a platform of business discovery that offers an auto service business intelligence for every kind of business or organization users. QlikView was relevant for having an introduction to BI software. Mainly involved in doing consulting for the enterprise *Circuit de Catalunya*, in a portal that showed ticket data of events like *Formula1* and *MotoGP* in a visual manner.
- Genetec: Platform of videosurveillance and video analysis leader in the market which combines systems of IP security in a unique interface for the simplicity of its operations. As a principal service of the business, there was a period of familiarization with the platform in case one of the specialized technicians were not available, so it was possible to still provide the service.

NexoTech Company

NexoTech S.L. was created in 2004 as a company specialized in the development and marketing of video surveillance, security, integration and TIC consulting solutions. From access control, video surveillance and automatic recognition of license plates to communications, intrusions and systems of video analysis and integration with third-party systems.

They have central offices in Mataró (Barcelona) and, as of 2012, in Santiago de Chile as the basis of operations for the development of projects in Latin America for both the public and the private sector.

NexoTech has obtained several recognitions throughout its business career, such as the 1st Cre@tic Prize 2004, the 1st Bancaja 2005 Award for Young Entrepreneurs, among others, and also has the certification of leading manufacturers in the sector such as Genetec, Bosch, Panasonic, Flir, Zebra or Intel and collaborates with several universities and institutions.

They have a multidisciplinary team of professionals from different areas such as engineering, security, IT, telecommunications, automation and management. So resources on any kind of problematic from different departments will not be scarce.

The projects are oriented to provide advanced IT SOLUTIONS that evolve with the needs of our customers, contributing to increase the efficiency in their business processes.

NexoTech is registered as a private security company at R.E.S.C. (Special Register of Security Companies of Catalonia) with n°2017 / 006.

Need of the project

At the end of summer 2018, the board of directors of the company proposed the development of the application that starts in this project.

The proposed idea was to create business analytics platform with the information of the occupation on enclosures for the already clients of videosurveillance.

The identification and counting of gender and age with deep learning was proposed by the student, also related to one of the proposals from the professor body.

2.2. Context

2.2.1. Market

Currently there are several companies that offer similar solutions of occupation control. The differentiation that gives value to the application is the customization for each client, where it adapts depending on the hardware that is used and geographic difficulties where the enclosure is found. Moreover, the classification of genders donates a unique value that cannot be found in the rest of the market.

2.2.2. Web Applications

Nowadays, most of software applications are web applications. A web application is an application that users can use accessing a web server via Internet or intranet using a browser. In other words, it is an application (Software) that is codified in a language supported by web browsers in which the execution is entrusted to the browser. It may also be defined as a client-server computer program which the client (including the user interface and client-side logic) runs in a web browser.

Throughout the degree, programming of web application and technologies used for it were taught. Although many different architectures for *WebApps* such as *serverless* apps or progressive apps exist; the standard backend, frontend and database architecture is used for this project that can be seen on figure 2.1.

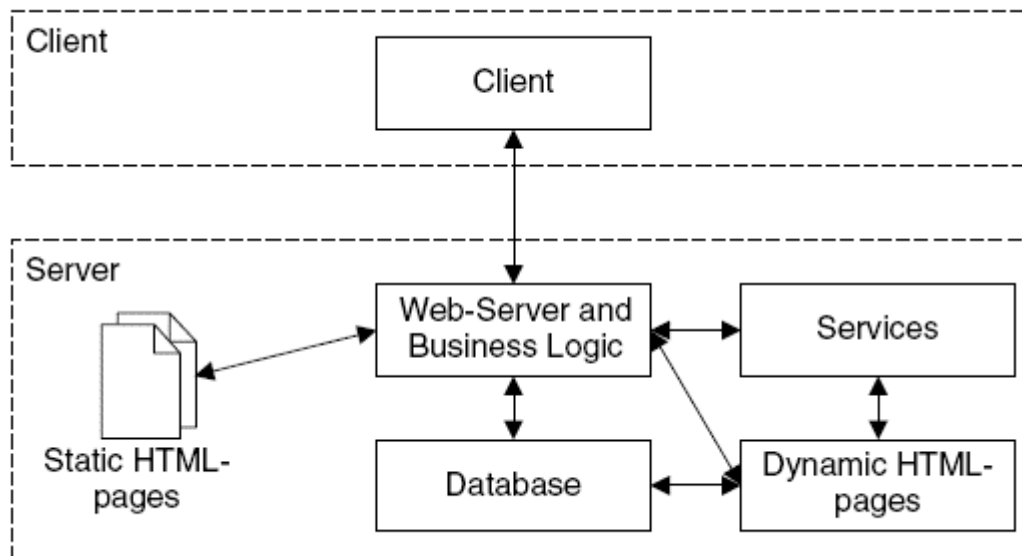


Figure 2.1: Client-server basic architecture structure. Source: <https://gyires.inf.unideb.hu/GyBITT/08/ch04.html>

2.2.3. Database

The database is where the application stores its data. A database is used by an organization as a method of storing, managing and retrieving information. Modern databases are managed using a database management system (DBMS). It is imperative to use a DBMS that allows all the application parts to access easily, as the sensor will also have to send data to it.

There are many types of databases, for instance NoSQL or object-oriented database, each with its own advantages and perks. For this project a relational database is used, as it is the more widespread and used type of database.

SQL is a programming language for Relational Databases. There are also a lot of database languages that are variations of it, but for this project MySQL is used for its simplicity and easy connection with the sensors.

The database is stored in a specialized server.

2.2.4. Backend

Using a client-server environment, the server or backend references the logic of the *WebApp*. Backend is the data access layer of a software or any device, which is not directly accessible by users.

Some of the programming languages of backend are Python, PHP, Ruby, C# and Java, each of the above have different frameworks to create a web backend that work better according to the type of project you are developing. For example Django, Laravel, Ruby On Rails and ASP.Net [1].

In this project Java is used, with Spring Boot and other frameworks and libraries for building the API so the frontend can communicate with it. Both used for its commonness, large community that supports them and personal knowledge of them.

2.2.5. Frontend

Frontend is the part of a program or device that a user can access directly. It involves web design and development technologies that run in the browser and that are responsible for interactivity with users.

HTML, CSS and JavaScript are the main frontend languages, from which a number of frameworks and libraries emerge that expand their capabilities to create any type of user interface. React, Redux, Angular, Bootstrap, Foundation, LESS, Sass, Stylus and PostCSS are some of them.

ReactJS is used for this project. React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

2.2.6. Sensors for counting people

On the market, plentiful of solutions for counting people are available [2]. Those can be divided by three main technologies:

- **Movement sensors:** It uses a camera. It relies on movement of pixels to recognize people and then track that person's direction to determine whether they enter or exit the enclosure. It is not the best solution as it has the disadvantage of poor low-light performance and it can have problems distinguishing movement by people and other movements from the environment.
- **Image recognition:** It also uses a camera. This solution involves machine learning, it reads frames from a video stream to recognize a person and determines whether that person crosses a pre-established line for entries and another for exits. It does have good accuracy and allows for other image recognition like gender and age to be implemented.
- **Infrared sensors:** It similarly to the image recognition as it recognizes the people but in this case with infrared points. It has the best accuracy but it cannot be used for image recognition so each entry has to be paired with a camera if the user is interested in purchasing the gender and age recognition.

This project uses a combination of infrared sensors and cameras for image recognition, the sensor is called *Gazelle 2* by *Irisys*, and was given by the company *NexoTech*.

Gazelle 2 is powered by *Irisys*' infrared technology and using body heat to track the presence and direction of movement, the *Gazelle 2* has proven reliability of over 98%, and can generate valuable analytics data for end users. It is also suitable for a multitude of applications, including People Counting, Smart Buildings, Security and more.

It has fast and easy installation, wide opening capability, stability, scalability and privacy.

A small program is built by the head of technology department to catch the sensor's information and send it to the database so the main application can then retrieve the information and show it to the user. Although it is not an intrinsic part of the project, part of the code will be explained later on.

More about the installation process and data retrieving will be explained in the *Development* section.

2.2.7. Deep Learning

A goal has been set to get more information of the people entering the enclosures, it is wanted to classify the people by gender and age, for business analytics purposes. It will be done by image recognition on the cameras at the entrances. The solution more used and developed nowadays for image recognition is based on deep learning, as it offers better results than non-deep neural networks.

Although it is not an essential part of the application, it takes a big portion of the time to research and implement a modest solution.

On the next paragraphs there will be an explanation of what deep learning is about and current solutions for the problem faced.

2.2.7.1 What is deep learning?

Deep learning is a machine learning technique that teaches computers to do what humans do in a natural way: learn by way of example. Deep learning is a key technology behind autonomous cars, which allows them to recognize, for example, the stop symbol, or distinguish a pedestrian from a street lamp. It is the key to voice control in consumer devices, such as telephones, tablets, televisions and hand-held speakers. Deep learning is receiving a lot of attention, and deservedly so [3].

In deep learning, a computer model (or computer model) learns to perform classification tasks directly from images, text or sound. The models of deep learning can achieve cutting-edge precision, which sometimes exceeds the performance at the human level. The models are formed by the use of a large set of labeled data architectures and neural networks that contain many layers.

For the project it is essential to achieve high precision. Deep learning achieves recognition accuracy at levels higher than ever. This allows consumer devices to respond to user

expectations, and is critical for critical security applications, such as seating control or standalone cars. Recent advances in deep learning have improved to the point where it is possible to overcome humans on some tasks such as classifying objects in images.

Although deep learning was theorized for the first time in the 1980s, there are two main reasons why it has recently become useful: deep learning requires large amounts of labeled information. For example, the development of driverless cars requires millions of images and thousands of hours of video. For the project, a sufficiently large dataset will be required to have a high accuracy.

On the other hand, deep learning requires great computer power. High-performance GPUs have an efficient parallel architecture for deep learning. When combined with clusters or cloud computing, this allows the development teams to reduce the training time of a deep learning network from weeks to hours or less.

Most of the deep learning methods use neural network architectures, so deep learning models are often referred to as deep neural networks.

The term "deep" or "deep" usually refers to the number of layers hidden in the neuronal network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have up to 150.

Deep learning models are designed to mitigate the need for great combinations of tagged (or labeled data) and neural architecture that capture characteristics directly from the data without the need for information extractions with manual functions.

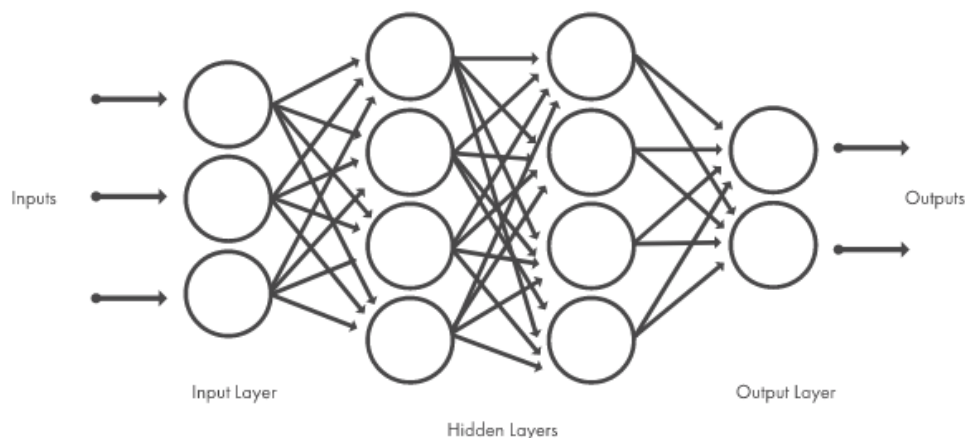


Figure 2.2: Neural networks, organized into layers that consist of a set of interconnected nodes. Networks can have dozens or hundreds of hidden layers. Source: <https://es.mathworks.com/discovery/deep-learning.html>

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN or ConvNet). A CNN converts input data into functions learned to detect a series of features, and uses 2D convoluted layers, making this architecture ideal for processing 2D data, such as images.

CNN eliminates the need for feature extraction with manual functions, so it is not necessary to identify the characteristics used to classify images. CNN works with extracting features directly from images. The relevant features are not previously taught; they learn while the network integrates them into a collection of images. This automatic extraction of features makes deep learning models very accurate for computational vision tasks, such as object classification.

CNNs learn to detect different markers (or features) of an image using dozens or hundreds of hidden layers. Each hidden layer increases the complexity of the image features learned. For example, the first hidden layer could learn to detect the edges, and the latter learn how to detect more complex shapes that adapt specifically to the shape of the object we try to recognize.

2.2.7.2 Inside Artificial Intelligence

Inventors have always dreamed of creating machines that they think [4]. This desire is found from at least the time of ancient Greece. The mythical representations Pygmalion, Daedalus and Hefesto can be interpreted as legendary inventors, and Galatea, Talos and Pandora can be considered artificial life (Ovidio and Martín, 2004; Sparkes, 1996; Tandy, 1997).

When programmable computers were conceived for the first time, people wondered if these machines could become intelligent, more than one hundred years before one was built (Lovelace, 1842). Today, artistic intelligence (Artificial Intelligence or AI) is a prosperous field with many practical applications and active research topics. We see using intelligent software to automate routine work, understand speech or images, do medical diagnoses and support basic scientific research.

In the early stages of artistic intelligence, the field quickly addressed and resolved problems that are intellectually difficult for humans, but relatively easy for computers, problems that can be described by a list of formal and mathematical rules. The real challenge to artificial intelligence was to solve the tasks that are easy for people to do, formal problems that we solve intuitively, that they feel automatic, such as recognizing words, faces in images or the sex of a person.

The solution used in this project is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relationship with simpler concepts. When collecting the knowledge of the experience, this approach avoids the need for human operators to formally specify all the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts creating simpler ones. If we draw a graph that shows how these concepts are constructed to each other, the graph is deep, with many layers. For this reason, we call this approach to the deep learning of AI.

Many of the first hits of the AI occurred in relatively sterile and formal environments and did not require computers to have a lot of knowledge about the world. For example, the IBM Deep Blue chess system defeated Garry Kasparov in 1997 (Hsu, 2002).

Ironically, the abstract and formal tasks that are among the most important mental tasks for a human being are the easiest ones for a computer. Computers have been able to defeat even the best human chess player, but have recently begun to carry out some of the abilities of human beings to recognize objects or discourse.

Various artificial intelligence projects have attempted to adhere to a difficult knowledge of the world in formal languages, that is, hard-coded. A computer can automatically reason for declarations in these formal languages using logical inference rules. This is known as knowledge base approach for handmade intelligence. None of these projects has resulted in greater success.

The challenges facing coded knowledge-based systems imply that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This ability is known as automatic learning or machine learning (ML). The introduction of the ML will allow computers to address problems that imply knowledge of the real world and make

decisions that seem subjective. A simple ML algorithm called logistic regression or logistic regression can determine if cesarean is recommended (Mor-Yosef et al., 1990). A simple ML algorithm named Naive Bayes can separate the legitimate email from spam.

The performance of these simple machine learning algorithms depends to a great extent on the representation of the data that is given. For example, when logistic regression is used to recommend cesarean section, the AI system does not examine the patient directly. Instead, the doctor informs the system of several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the patient's representation is known as a feature. A set of features from the people entering and leaving the enclosure for the program to give a solution.

This dependence on representations is a general phenomenon that appears throughout computer science and even everyday life. In computing, operations such as searching for a data collection can increase exponentially if the collection is structured and indexed intelligently. People can easily make arithmetic in Arabic numerals, but finds that arithmetic in Roman numerals consumes much longer. It is not surprising that the choice of representation has a huge effect on the performance of ML algorithms.

Many artificial intelligence tasks can be solved by designing the correct set of features that can be extracted for this task, providing these features with a simple machine learning algorithm. For example, a useful feature for speech identification is an estimate of the size of the speaker's voice tube. This feature gives a strong clue about whether the speaker is a man, a woman or a child.

However, for many tasks, it is difficult to know what features are to be extracted. For example, for this project, the deep learning algorithm will have to detect humans in photographs. We know that humans have legs, so you might like to use the presence of a leg as a feature. Unfortunately, it is worthy to describe exactly what a leg looks like in terms of pixel values. A leg has a simple geometric shape, but its image can be complicated by the shadows falling on the foot.

A solution to this problem is to use machine learning to discover not only the representation mapping on the output, but also the representation itself. This approach is known as representation learning or representation learning. Well-known representations often

produce much better performance than what you can get with a manually rendered design. They also allow AI systems to quickly adapt to new tasks, with minimal human intervention.

A representation learning algorithm can discover a good set of functions for a simple task in a matter of minutes, or for a complex task in hours to months. The manual design of functions for a complex task requires a great deal of human time and effort; decades can take place for a whole community of researchers. The prime example of a representation learning algorithm is the autoencoder. An autoencoder is the combination of an encoder function, which converts the input data into a different representation, and a decoder function, which converts the new representation into the original format.

When we design features or algorithms for the learning characteristics, our general objective is to separate the variation factors that explain the observed data. In this context, we use the word "factors" simply to refer to separate sources of influence; the factors usually are not combined by the multiplication. Often, these factors are not amounts that are directly observed. Instead, they may exist as unobserved objects or unobserved forces in the physical world that obtain observable amounts.

Of course, it can be very difficult to extract abstract high-level and abstract functions from raw data. Many of these variation factors, such as the accent of a speaker, can be identified only by using a sophisticated and almost human understanding of the data. When it is almost as difficult to obtain a representation as to solve the original problem, the learning of the representation does not seem, at first sight, to help us.

The deep learning solves this central problem in the representation learning by means of the introduction of representations that are expressed in simpler terms. Deep learning allows the computer to build complex concepts based on simpler concepts. Figure 2.3 shows how a deep learning system can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which in turn are defined in terms of edges.

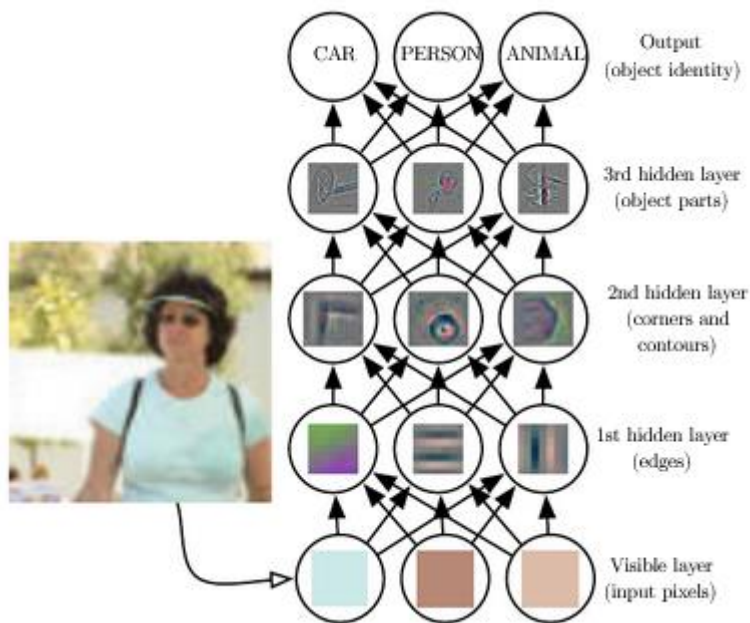


Figure 2.3: Illustration of a deep learning model. It is difficult for a computer to understand the meaning of unprocessed sensory input data, we see this image becomes represented as a collection of pixel values. Source: <https://www.deeplearningbook.org/contents/intro.html>

The prime example of a deep learning model is the multilayer or multilayer perceptron (MLP) receiver. An MLP is only a mathematical function that assigns a set of input values to the output values. The function is formed by forming many simpler functions. We can think that each application of a different mathematical function provides a new representation of the input.

The idea of learning the right representation of the data provides a perspective on deep learning. Another perspective on deep learning is that depth allows the computer to learn a computer program in many steps. Each layer of the representation can be considered as the state of computer memory, then executing another set of instructions in parallel. The networks with greater depth can execute more instructions in sequence.

Sequential instructions have a great power because later instructions can refer to the results of the previous instructions. According to this vision of deep learning, not all information in activations of a layer necessarily encode the variation factors that explain the entry. The representation also stores status information that helps execute a program that can make sense in the entry. This status information could be analogous to a counter or pointer in a

traditional computer program. It has nothing to do with the content of the entry specifically, but it helps the model to organize its processing.

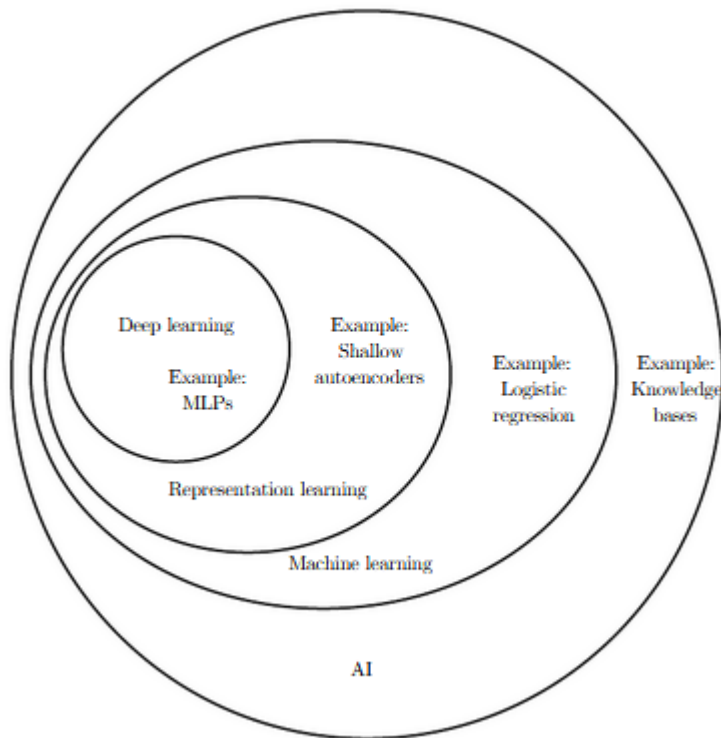


Figure 2.4: A Venn diagram that shows how deep learning is a kind of representation learning, which in turn is a kind of machine learning. Source: <https://www.deeplearningbook.org/contents/intro.html>

To summarize, deep learning is an AI approach. Specifically, it is a type of machine learning, a technique that allows to improve computer systems with experience and data. We know that machine learning is the only viable approach to build AI systems that can operate in complex real-world environments. The deep learning is a particular type of machine learning that achieves great power and flexibility representing the world as a hierarchy of leveled concepts, with each concept defined in relation to simpler concepts and more abstract representations calculated in less abstract terms. Figure 2.4 illustrates the relationship between these different AI disciplines.

2.2.7.3 Historical trends in Deep Learning

It is easier to understand deep learning with a historical context. Instead of providing a detailed history of deep learning, you can identify some key trends [5]:

- Deep learning has had a long and rich history, but has gone through many names, reflecting different philosophical points of view.
- Deep learning has become more useful as the amount of available training data has increased.
- Deep learning models have grown in terms of time, since the computer infrastructure (both hardware and software) for enhanced learning.
- Deep learning has solved increasingly complex applications with increasing precision over time.

2.2.7.4 Types of Deep Learning Neural Networks

Publishing a complete list of deep neural networks or deep neural networks is impractical, as there are many variants and combinations. In addition, it is a constantly evolving field.

Many times, many of these networks are combined in batteries or layers to give better results than a single network. For example, an advanced network of resources such as MLP that uses selected functions of an autoencoder can be more effective than MLP alone.

The following are some of the main networks, along with a brief description and some cases of use for each one of them [6]:

Multilayer Perceptrons (MLP)

They are the most basic networks and feed the entries to create results. They consist of an input layer and a layer of output and many layers and hidden neurons interlaced between the layers of entry and exit. In general, they use an activation function Non-linear like ReLu or Tanh and calculate the losses (the difference between the real output and the computed output) such as Mean Square Error (MSE), Logloss. This loss spreads backwards to adjust weights and training to minimize losses or make models more accurate.

This network is usually the initial phase of building a more sophisticated deep network and can be used for any supervised regression or classification problems, such as sales prediction, fatality, probability of response, etc. For example, in the bibliography one can find a code of Python that identifies diabetes of non-diabetic cases through Tensorflow. The data used here are PIMA Indian Diabetes data [7].

Autoencoders (AE)

Stacked autoencoders are unsupervised learning algorithms for function learning, dimension reduction and external detection, figure 2.5. The concept of the AE is quite simple: here input vectors are used to calculate the output vectors, but the output vectors are the same as the input vectors. The reconstruction error is calculated and the data points with the superior reconstruction error are assumed to be higher. AEs are also used for the recognition of voice and images. For example detection fraud detection by credit card through autoencoders in Keras[8].

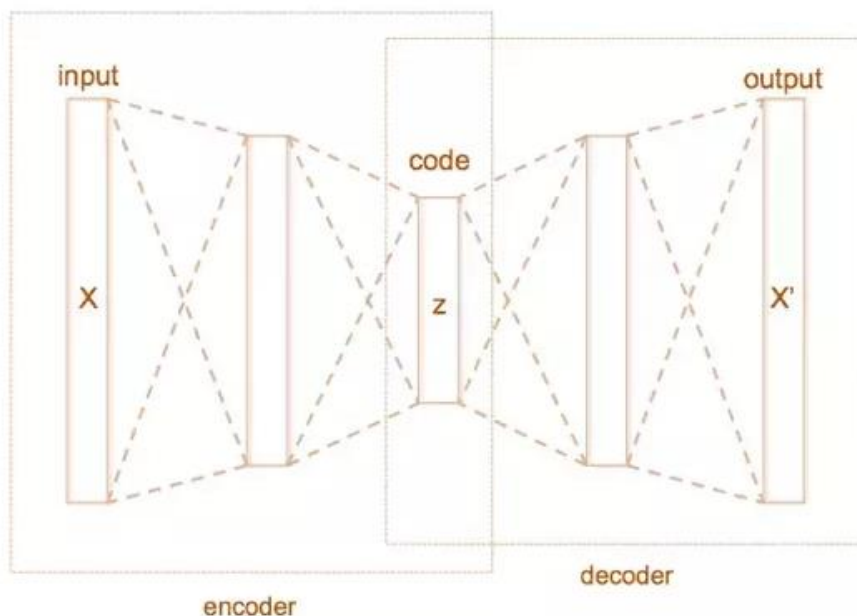


Figura 2.5: Scheme that shows the operation of the autoencoders.

Convolution Neural Network (CNN)

Convolution Neural Networks (CNN) significantly improves feed forward network capabilities such as MLP by inserting convolution layers, Figure 2.6. They are especially suitable for spatial data, the recognition of objects and the analysis of image analysis through structures of multidimensional neurons. One of the main reasons for the popularity of deep learning lately is due to CNN. Some of the common uses of CNN that surround us are: cars of own conduction, drones, vision by computer, analysis of text, etc. It will possibly be the best option for gender recognition based on people's images. An example of digit recognition by hand can be found in the bibliography [9].

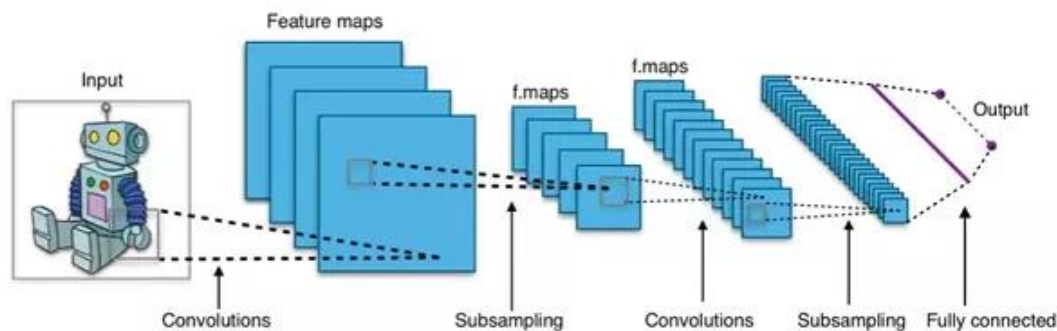


Figura 2.6: Scheme that shows the operation of CNN.

Recurrent Neural Network (RNN)

RNNs are also a feed forward network, however, with recurring memory loops that take the input from previous layers or states and / or the same. This gives them unique ability to model along the time dimension and the arbitrary sequence of events and entries. One of the most common types of the RNN model is the short-term memory network (LSTM). Figure 2.7 shows how an algorithm that uses RNN when receiving a new entry knows the most likely output in relation to the words that it previously received, the example shows a word completion system.

RNNs are used for the analysis of sequenced data, such as time series, feelings analysis, NLP, language translation, speech recognition, image capture and sequence recognition, among others. An example of LSTM can also be found in the bibliography [10].

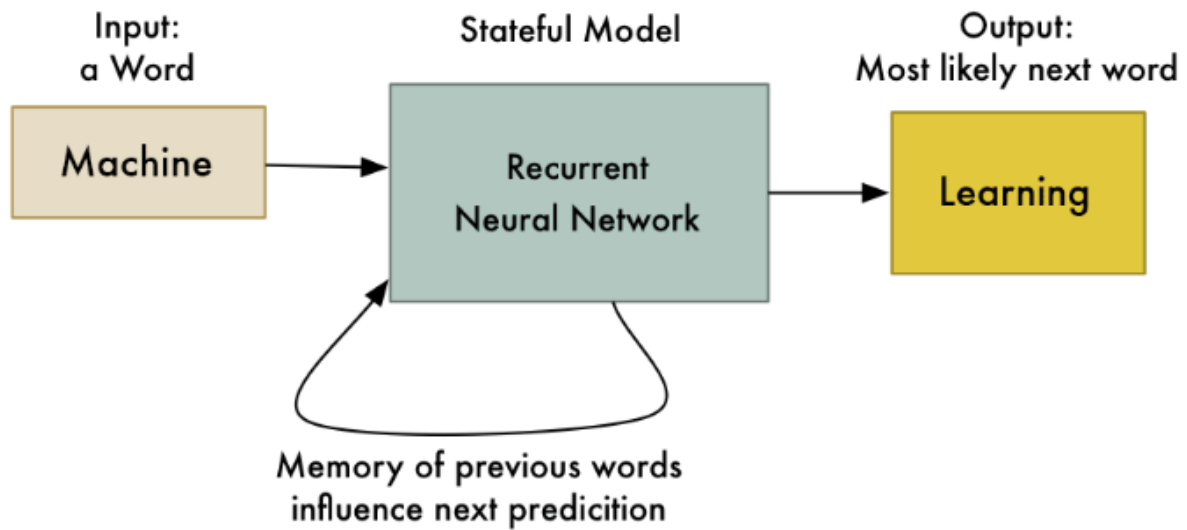


Figure 2.7: Operation of an RNN. The program receives a word, the program looks to the memory of the previous words to influence the following prediction. The output is the next possible word.

2.2.8. Image recognition

Deep learning has dominated completely the vision by computer in the last years, obtaining maximum scores in many tasks and its related competitions. In recent years, deep learning techniques have allowed for rapid progress in this competition, even surpassing human performance.

The best known of these computer vision skills is ImageNet [11]. The task of ImageNet's competition investigates the researchers with the creation of a model that more precisely classifies the images given in the data set.

Almost every year since 2012 has given us great advances in the development of models of deep learning for the task of image classification. Due to its large-scale and challenging data, ImageNet's challenge has been the main point of reference for measuring progress.

Some of the main architectures that made this progress possible are the following.

- AlexNet

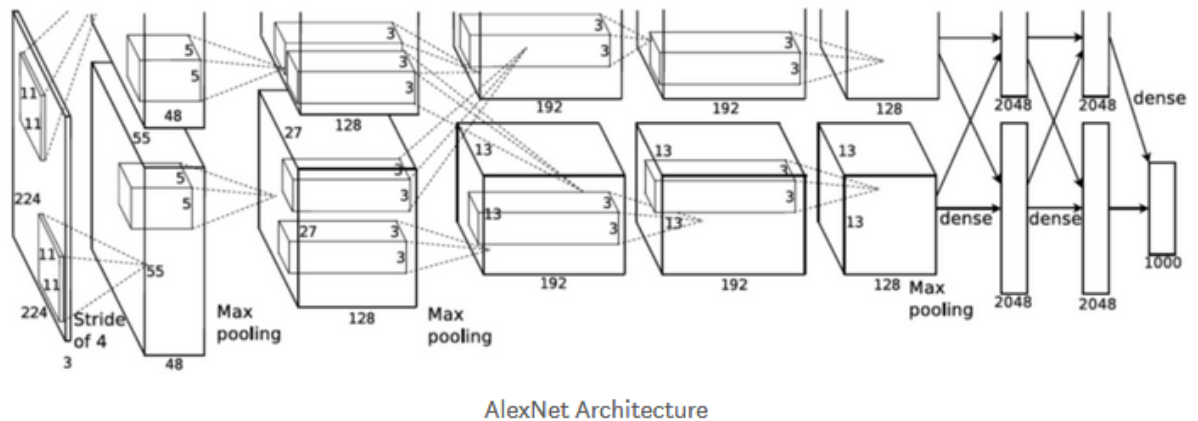


Figura 1.6: AlexNet architecture.

As of 2012, a document from the University of Toronto was published at the NIPS. This document was ImageNet Classification with deep networks of Convolutions [12]. It would become one of the most influential documents in the field after achieving a 50% reduction in the error rate in the ImageNet Challenge, which was an unprecedented move at that time. The architecture of the AlexNet neural network of the paper is shown above.

The document proposes to use a deep convolutional neural network (CNN) for the task of image classification. It was relatively simple compared to those being used today.

- VGGNet

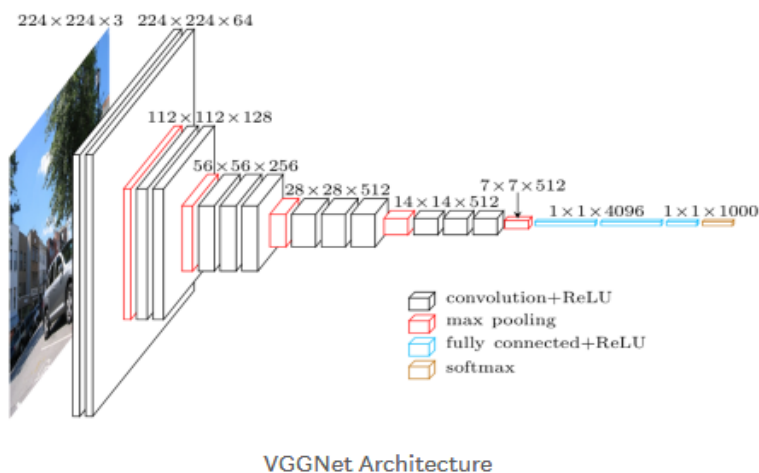


Figure 2.8: VGGNet architecture.

The VGGNet role "Convolved deep-neuronal neural networks for large-scale image recognition" appeared in 2014[13], further expanding the ideas of using a deep network

with many convolutions and ReLUs. The neural network architecture for VGGNet on paper is shown in Figure 2.8. His main idea was that he really did not need any extravagant tricks to get great precision. Only a deep network with many small 3x3 convolutions and non-linearities would be sufficient. These repetitive block batteries of the same size as in the previous figure are direct results of using 3x3 batteries.

- **GoogLeNet and the Inception module**

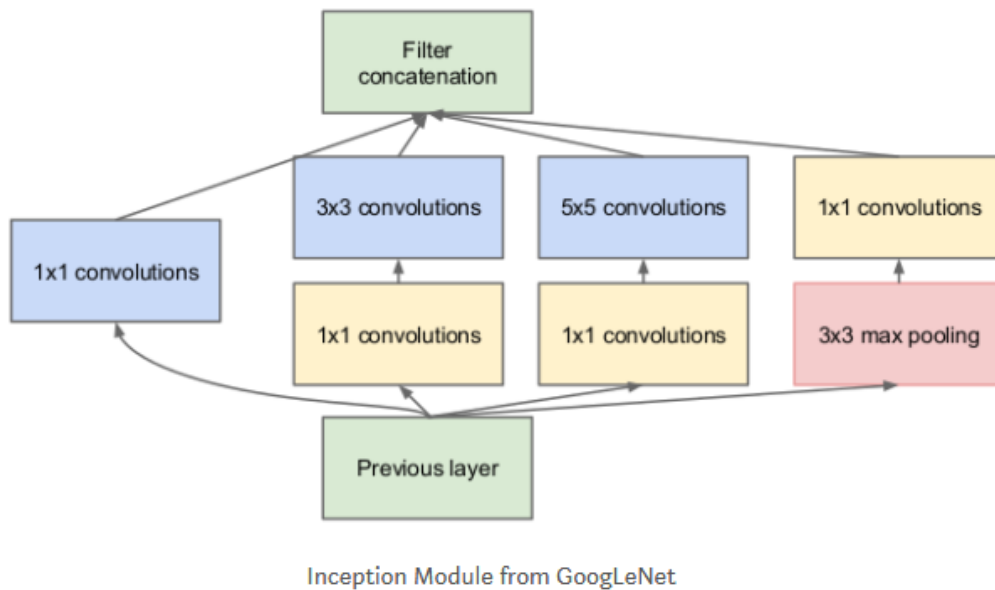


Figura 2.9: GoogLeNet architecture.

The architecture of GoogLeNet was the first to really address the issue of computational resources along with large-scale processing in the document "Going Beyond Convolutions"[14]. As we continue to make our rating networks deeper and deeper we reach a point where we are using a lot of memory. Additionally, in the past, different computational filter sizes have been proposed: from 1x1 to 11x11; how do you decide what? The startup module and GoogLeNet addresses all these problems.

- **ResNet**

Since its initial publication in 2015 with the document "Deep Deep Deepening for Image Recognition"[15], ResNets has created important improvements in accuracy in many computer vision tasks. ResNet architecture was the first to pass the level of humans in

ImageNet, and its main contribution to residual learning is often used by default in many state-of-the-art networks.

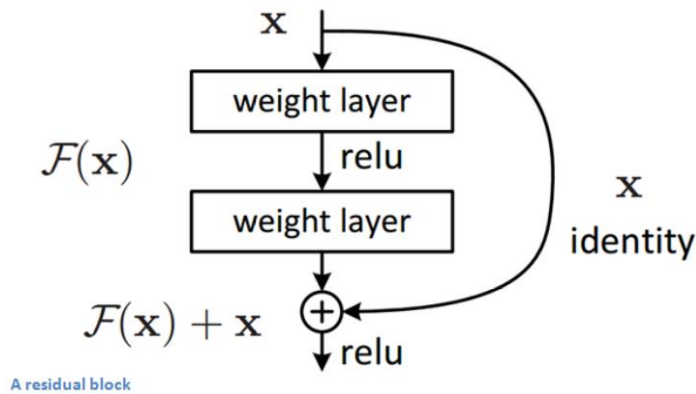


Figura 2.10: Residual block

- **DenseNet**

The shortcut connections were extracted with the introduction of DenseNets from the paper "Densely Known Convolved Networks"[16]. DenseNets broadens the idea of direct access connections but has a much denser connectivity than ResNet.

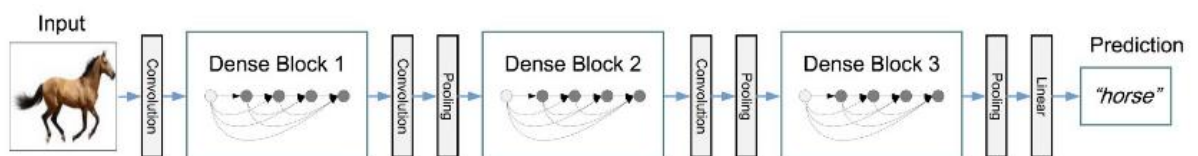


Figura 2.11: DenseNet Architecture.

- **Wide Residual Networks**

Deep residual networks were shown to be able to scale up to thousands of layers and still have improving performance. However, each fraction of a percent of improved accuracy costs nearly doubling the number of layers, and so training very deep residual networks has a problem of diminishing feature reuse, which makes these networks very slow to train. To tackle these problems, the paper Wide Residual Networks (WRNs)[17] conducts a detailed experimental study on the architecture of ResNet blocks, based on which it proposes a novel architecture where it decreases depth and increase width of

residual networks. It calls the resulting WRNs and show that these are far superior over their commonly used thin and very deep counterparts. For example, it demonstrates that even a simple 16-layer-deep wide residual network outperforms in accuracy and efficiency all previous deep residual networks, including thousand-layer-deep networks, achieving new state-of-the-art results on CIFAR-10, CIFAR-100 and SVHN.

It is used in this project for gender and age recognition, as it offers better results and accuracy.

These are the main architectures that have been the backbone of progress in the classification of images in recent years. Great progress has been made and it is exciting to see, since it allows us to solve many real-world problems with this new technology.

This project aims to understand the evolution of image recognition with deep learning, how to apply it to the problem and if possible, move the search to the next step.

2.2.9 Gender and age recognition

Artificial intelligence has tried in various ways to distinguish the sex of a person, such as using voice as the main characteristic [18]. Other solutions involve reading images, which is used for the project. Those images can be obtained with either a normal or thermal camera.

The use of automatic gender recognition is more widely used and demanded than ever in various programs and hardware, especially due to the growth of websites in social networking and social networks. However, performance of gender and age recognition software on real-life face images are not optimal.

It has been explored that, through the learning and classification method and with the use of the technique of convolutional deep neural networks (D-CNN), satisfactory growth in performance can be achieved in gender classification tasks [19].

Technical reading methods have also been found to distinguish hands from women or men with thermal vision [20]-[22].

The project is looking at the most appropriate solution to the conditions in which the vision of the cameras is found.

A solution of a Wide Residual Network is used as the main neural network that will predict gender and age on images, which can be described as a wider CNN. It will be trained with the datasets IMDB-WIKI [23] and UTKFace [24]. Both are datasets of images with the identified gender and age of the people that appear on those images.

After the neural network is trained it can be used on images of faces. To get the cropped images of the faces we need to first identify those faces. For face identification we use the library *dlib* from python which has methods that identify them. When we have the faces cropped we then predict the gender and age. More about the process will be explained in the “Development” chapter.

2.3. Information needs

The project requires a lot of research to use the technologies and define the most appropriate structures for the application. In addition to studying machine learning and specifically deep learning for the gender and age recognition.

The project requires knowledge of basic calculation, algebra and statistics. For the web application, technologies used were taught during the degree. For the deep learning program, it is advisable to know Python, since most frameworks use one of these languages and it will be used for age and gender recognition. That is why the courses of Python of Udacity or Coursera are useful.

For deep learning, the Coursera course by Andrew Nguyen is studied [25]. It explains how deep neural networks work on the inside and how to make and improve them on Python. It is highly recommended as it is the basis of this project age and gender recognition program.

Some of the most popular frameworks and dependencies for developing deep learning in Python and that are used in this project are:

1. Keras2.0+ (Keras Deep Learning that compiles above Tensorflow)
2. Tensorflow Bookstore from Google.
3. scipy, numpy, Pandas, tqdm, tables, h5py
4. dlib
5. OpenCV3

A set of classified image data is also needed in situations where the application is used to train the gender recognition algorithm. Kaggle, as the industry standard, is the portal where the appropriate dataset is located, it is where UTKFace can be found.

3. Goals and scope

The main goal of the project is developing a full-stack web application that shows in real time the occupation of an enclosure.

Other important features that headline the application are a reporting system and an alarm system.

In the reporting system you can extract data from a set of filters, such as date, time or access. The reporting system, after the user fills all the configuration forms, creates a report with the data filtered. The data will be shown in different types of ways, like graphs and tables. That report can be downloaded in different formats, in concrete CSV, PDF or as a spreadsheet.

The alarm system contains a configuration page, where the application lets users create and configure different types of alarms. Those being an alarm that triggers when the occupation exceeds a determined warning and danger percentage, or an alarms that notices the system of disconnections of the sensors. If the configured alarms are triggered, an email will be sent to whomever the user decides, and the alarm will be posted on a incidents page as a log of triggered alarms.

The project is sponsored, as specified in the "Context and Background" section, for the company Nexotech S.L., a security company that wishes to develop an occupation control software for its clients.

How does the occupation control system work?

- First, a new client wants to have an occupation control in its enclosure, this enclosure is called a *center* within the application.
- The *center* has at least one *zone* or area, usually called "interior", although you can add all the zones you want.
- Finally, each *zone* has *access* where people can enter and leave the *center*, in each *access* there is a sensor to count inputs and outputs, the difference of these entries and exits and the sum of this result gives the total capacity of the center.

Each *access*, apart from the people counting sensor, also has a camera for reading images and extracting other information.

The web application is built as a standard full-stack *WebApp*, it has its front-end, back-end, and database. The technologies used on each part of the project are specified on the next chapter.

It should be said that the process of sending the information from the sensor to the database will not be part of the project, only the extraction of this data to be displayed in the web application.

Once the web application is completed, there is a research and developing period to implement a control, not only from the number of people on the enclosure, but also the classification of that people by gender and age.

Here are the lists of goals of the final product and project:

3.1. Goals of the product

- Add new *centers* with their corresponding *zones* and *access*, as well as a maximum capacity for each zone.
- Add users of the *centers*.
- Show thoroughly and visually the real-time occupation of the *center*.
- Show the entries and exits of each *access* and *zone*.
- Creating reports of the historical capacity of the enclosure, can be filtered by range of dates, times, days of the week, grouping by time period (occupation of each minute, time, day, etc.), and specific data of a *zone* or *access*.
- Be able to download reports with a visually pleasing format in PDF, CSV or spreadsheet.
- Ability to setup general information of the *center*.

- Add, edit, or delete users with their corresponding roles.
- Create a system of alarms and incidents once they jump if the capacity exceeds the maximum or near capacity, also if a sensor is disconnected.
- Classifying people by gender and age by recognizing images with the cameras located in the access.
- Show the amount of occupation of men and women with the information obtained in the previous objective.

3.2. Goals of the client

- Keep track of the people that are in your enclosure.
- Gain confidence knowing the occupation at all times, which is currently required by law in Spain [26].
- Find out which time slots more people access your site.
- Have at all times the security that the system is as accurately as possible.
- Know information about the kind of people that access their site, thus gaining information for their decision making.

3.3. Definition of functional and technological requirements

Functional Requirements

The functional requirements of the web application are described below. The functional requirements establish the behavior of the software. Interaction between system and environment.

- 1) The system must provide the company's technicians to insert a new center with: name, maximum capacity, zones, access and user administrator by the client.
- 2) The system must show a login form so that the user can access the app: center name, user and password must be defined.
- 3) The system must show relevant capacity information in a dashboard as an initial screen.
 - a) The system will show a graph that shows the occupation relative to the maximum capacity of the center.
 - b) The system will show a table with the breakdown of the accumulated capacity of entries and exits of each *zone*. In addition to the interval (difference between the last data sent by the access sensors and the previous accumulated one).
 - c) The system will show a flow chart of the entries and exits of each access.
 - d) The system will show the percentage of fullness of the center. If it exceeds a specific percentage of danger, it will be of different color.
- 4) The system must inform which user and center is the one logged in at that time.
- 5) The system will show if the access sensors are connected or disconnected.
- 6) The system must provide a configuration screen for the Administrators:
 - a) The center configuration and user management can only be executed by users with Administrator roles.
 - b) The system will allow to modify the type of schedule: party or continuous. If it is continuous, it will allow you to set an everyday capacity reset time.
 - c) The system will allow to change the name of zones and access, of each *zone* it will allow to define the maximum capacity, being the sum of these the maximum total capacity of the center.
 - d) The system will allow you to define a percentage of safe, warning and danger levels of occupation percentage.

- 7) The system must provide a screen for reporting:
 - a) The system must allow a report to be generated:
 - i) The system must allow selecting the range of dates when the data is wanted.
 - ii) The system must allow filtering for hours.
 - iii) The system must allow filtering for days of the week.
 - iv) The system must allow filtering by grouping of minute, hour, day, week, month and year.
 - v) The system must allow data from the complete center, specific zones, or specific access.
 - b) The system will show a graph of entries and exits in the defined groups.
 - c) The system will display a table of entries and exits in the defined groups.
 - d) The system must allow the report to be generated in PDF format.
 - i) The system will show the report in the application with the filter data and center name with the corresponding chart and table in PDF format.
 - ii) The system will allow you to download the PDF.
 - e) The system will allow you to download the table in spreadsheet format.
 - f) The system will allow you to download the table in CSV format.
- 8) The system must provide a list of incidents: from indicating capacity increases above specific percentages to alarms. It will also show manual capacity resets and if a sensor is switched on/off.
- 9) The system must distinguish and count gender and age range of the people who access the enclosure
 - a) The system must have image recognition on the cameras of the access.

- b) The system must distinguish between the gender and age range of the people identified by the system.
- c) The system must count the people of each gender and age range.
- d) The system should graphically show the occupation of each gender and age range.

Non-functional requirements

Non-functional requirements define how the system should be.

Security and data logic

1. The system will have to provide different levels of access to the application depending on the role of each user (Administrator, User, Super Administrator).
2. Permissions to access the system may only be modified by Administrators.
3. The system will use encrypted tokens for user authentication.
4. The system must have persistence.
5. In the case of not having an internet connection, the system must guarantee its persistence (it will keep the information in databases so that when it is arranged to be able to do the transaction)
6. Passwords will be saved with a database encryption.
7. It will not be possible to access the API if the user is not logged in, except the login API.

Usability

1. The interface must be intuitive and follow UML patterns.
2. The system must notify the user of the reasons why any error occurs.

Product

3. The web application must be compatible with all major browsers: Internet Explorer, Mozilla Firefox, Safari, Microsoft Edge, Google Chrome.
4. The interface must be completely responsive with screens of any size, computer, mobile or tablet.
5. The application must load as quickly as possible

3.4. Target or potential audience

The potential audience of the application is discotheques or leisure centers where it is necessary by law to monitor the capacity of the enclosure.

Business analytics information the app displays is another attraction for this type of client. Gender differentiation is a very interesting addition for these types of venues and their managers.

Other clients are libraries or other public leisure centers. The company NexoTech has confirmed clients that will use the application as Pacha Ibiza (disco), Conde de Godó (tennis competition) or the Festa del Cel (speed aircraft competition).

4. Methodology

The strategy of choice of solutions and technologies of this project is done following a process of exhaustive research of the state of the art and all alternative solutions. Several factors are taken into account, such as prior knowledge and ease with a specific technology, economic feasibility and the solution that has faster integration.

A web application is built in contrast to other types of applications as it is easy to access in every device with an internet connection and browser. Responsiveness is crucial as the application must bring a perfect user experience in every screen size.

In the case of software development, agile methodology, specifically *Scrum* [27], is used. Partial and regular deliveries of the production application are carried out, prioritizing deployment for the benefit they bring to the project recipient.

Scrum methodology is especially suitable for complex-environment projects, where results are needed as soon as possible, where the requirements are changing or not defined, and where innovation, competitiveness, flexibility and productivity are fundamental.

In Scrum a project is executed in short and fixed-term temporary cycles (iterations that are usually 2 weeks, although in some teams it is 3 and up to 4 weeks, maximum limit of real product feedback and reflection). Each iteration must provide a complete result, an increase in the final product that can be delivered with the minimum effort to the client when requested.

The activities that are carried out in Scrum are the following (the indicated times are for iterations of 2 weeks called Sprints): Selection of requirements, iteration planning, execution of the iteration and finally the inspection and possible adaptation. It will be necessary to inform the company about the advances in each Sprint.

Tasks control tools are used such as Trello, which allows you to organize tasks in TODOs, develop or develop, facts and once they are reviewed or reviewed, once revised they are done or completed.

The IDE for the web development that is used is Visual Code of Windows. The reasoning for the choice is that Visual Code combines the simplicity of a source code editor with

powerful developer tools, such as ending and debugging of IntelliSense codes. It is an editor that adapts to the main technologies and languages. The cycle edit-build-debug without friction implies less time playing with your environment and more time running on your ideas. In addition to having personal experience with IDE.

Once the requirements contemplated in the web application have been completed and we go on to find solutions for gender classification, information is sought according to criteria of public scientific articles on the syllabus. The Kaggle page is used to find datasets for the training of the deep learning model that is used.

Websites such as StackOverflow for software development problems or Medium for relevant articles are also very useful.

It is essential that Git is used as a version control system software, as it is the industry standard and provides fluency with the software. The repository is stored in the BitBucket web hosting version based on the website owned by Atlassian. BitBucket was used instead of other hosting services such as GitHub as it allows to make private repositories for free, and as the project is developed by a single person, it is appropriate. But nowadays GitHub also offers private repositories for free so the project was switched to it.

Lastly, it goes without saying that all information that does not come from reliable and proven sources is discarded.

4.1. Programming Languages, Libraries and Frameworks

A handful of programming languages, libraries and frameworks were used in every other layer of the application.

For the persistence layer, as said on the context chapter, MySQL is used as the DBSM:

- **MySQL**

MySQL is a relational database management system which uses the SQL (Structured Query Language) language.

It has become very popular thanks to its speed when executing inquiries, in the elaboration of web applications, in the environment of free software.

MySQL can be used in all kinds of applications (web, desktop, or other) free and freely under the terms of the GPL license.

It is also one of the components of the LAMP architecture (of Linux - operating system) and the WAMP (of Windows - operating system). The following three abbreviations (A, M and P) refer to the set consisting of Apache (web server), MySQL (database) and PHP (programming language). A platform for building websites using free software.

For the data processing, business logic layer or server, the main programming language used is Java, Spring will be used as the framework for creating the API, Spring security for the logging in and out of the application and have all around security on the app, plus other tools like Maven for putting the server app together and other dependencies too:

- **Java**

Java is a general, concurrent, object-oriented programming language that was designed specifically to have as little implementation dependencies as possible. Its intention is to allow application developers to write the program once and execute it on any device (known as "write once, run anywhere"), which means that the code that is executed on a platform It does not have to be compiled to run in another. Java is one of the most popular programming languages in use, particularly for client-web server applications, with about ten million users reported.

That's why it is used as the server programming language, as well as familiarity with the technology as it is the principal language taught at the degree for learning programming.

- **Spring**

Spring is a framework for the development of applications and control investment container, open source for the Java platform.

It is the standard framework for building APIs on client-server applications like this one. A RESTful web service is build that provides HTTP communication between both parts of the app.

REST or Representational State Transfer is an architectural style for designing distributed systems. It is not a standard but a set of constraints, such as being stateless, having a client/server relationship, and a uniform interface. REST is not strictly related to HTTP, but it is most commonly associated with it.

Spring security is used to the log in or sign up, and creating tokens and closing API URLs in case that the user is not logged in.

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.

- Other tools, plug-ins or dependencies

Maven as a software tool for the management and construction of Java projects.

Tomcat implements the JavaServer Pages (JSP) specifications of the Sun Microsystems servlet and server, and it provides the app a Java code environment for running in cooperation with a web server.

Other dependencies like Java Mail API is used for sending mails, or Gson for JSON manipulation.

Finally, In case of data presentation layer that works as the user interface, the main languages are classic web-based languages: HTML, CSS and JavaScript. But for easier and better structure, React is used as the main framework with SCSS as the CSS framework. NPM for downloading all packages needed.

- React

React is a JavaScript library for building user interfaces. It is maintained by Facebook and the community of free software, more than a thousand different developers have participated in the project.

React tries to help developers build applications that use data that changes all the time. Its objective is to be simple, declarative and easy to combine. React only handles the

user interface in an application; React is the “view” in a context in which the MVC pattern (Model-View-Controller) or MVVM (Model-view-view model) is used. It can also be used with React-based extensions that take care of the non-UI parts (which are not part of the user interface) of a web application.

- SCSS

Since SCSS is a CSS extension, everything that works in CSS works in SCSS. This means that for a Sass user to understand it, they need only understand how the Sass extensions work. Most of these, such as variables, parent references, and directives work the same; the only difference is that SCSS requires semicolons and brackets instead of newlines and indentation.

- NPM

NPM is the default package handler for Node.js, a JavaScript execution environment.

It is an essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

4.1.1 MVC

A variation of the Model-View-Controller software architecture is used in this project. That variation being a web application with a client-server structure.

Model–View–Controller (usually known as MVC) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts.

- Model: The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

- View: Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- Controller: Accepts input and converts it to commands for the model or view.

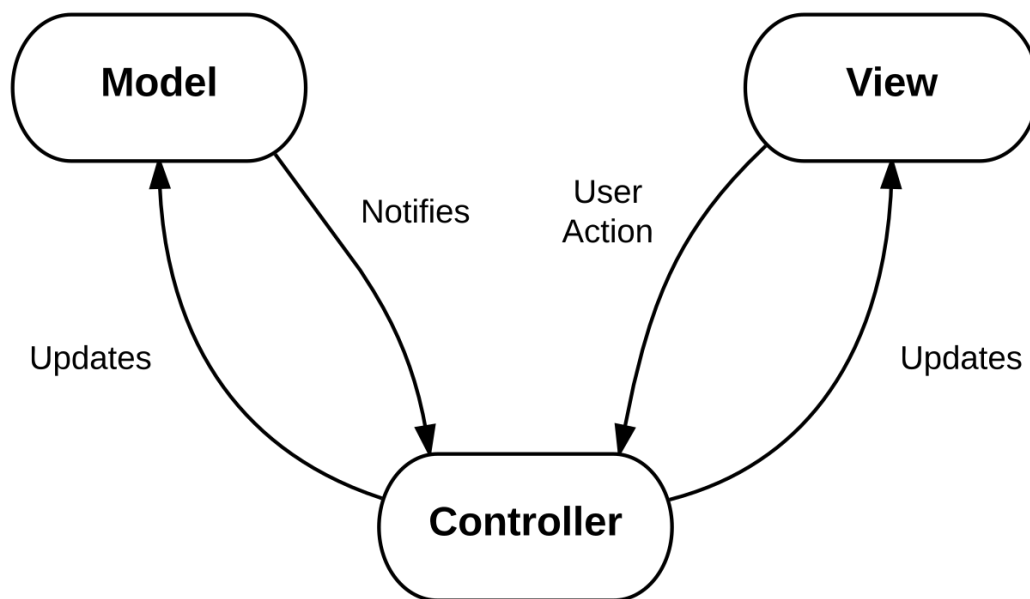


Figure 4.1: MVC basic structure. Source: <https://commons.wikimedia.org/wiki/File:MVC-basic.svg>.

Although originally developed for desktop computing, MVC has been widely adopted as an architecture for World Wide Web applications in major programming languages. Several web frameworks have been created that enforce the pattern. These software frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.

In this project case, the View is clearly the client application, although it has a few functionalities that could be considered as a controller. Mainly the controller is the server application, as well as the model controller with the actual persistence being on the database.

5. Development

The development of the application was, as stated at the methodology, pushed through different sprints. In this section it is explained the process of developing the application, with a first part describing the design and implementation of the database, after that the backend, and at last the frontend.

5.1. Database

The database development was focused on designing it in the most efficient way. The requirements must be the first concern in mind when designing the database, for that, there was an extensive period of thought before the final layout.

First of all, it is important to have the context mapped out so we know which words to use on the database, those words must be understandable for all of the business layers that will have contact with the application.

This design process is also known as domain-driven design. The words used will all be part of an ubiquitous language (UL). UL is a term part of DDD that means creating a vocabulary where all the people in touch of the app will be able to understand, and where the programmers will also have to, and can use easily.

Unless the table is a weak entity [28], all the tables have an autoincremental ID.

Here are all the terms in the ubiquitous language that also are an individual database table:

- Center

In the middle of the database design, there is a term that stands out, the enclosure or closed site, known in the ubiquitous language as the *center*.

The *center* is determined by the client that purchases the application, for each client, a *center* is build.

The *center* is connected, directly or indirectly to most, if not all the other tables on the database. When the user accesses the application, it accesses the *center* information.

The attributes will consist of:

1. Name
 2. Capacity: Maximum capacity of the *center*, it will be calculated on the data processing layer by the sum of the capacity of the *zones* that are part of the center.
 3. Occupation: Real-time occupation of the *center* calculated by the sum of the calculated occupation of the *zones* of the *center*.
 4. Warning Occupation: Percentage of warning occupation related to the maximum capacity. Default will be 70%.
 5. Danger Occupation: Percentage of dangerous occupation related to the maximum capacity. Default will be 95%.
 6. Reset time: Time of the day where the occupation will be resetted to zero.
 7. Manual reset: Boolean where if true, it will mean that the user made a petition to reset manually the occupation.
 8. Direction: Physical direction of the *center*.
 9. Phone Number: Contact of the client.
 10. Logo: Logo of the client or *center*.
 - 11.
- Zone

Each *center* can be divided by *zones*, a *center* will have at least one *zone*. The *zone* can be part of another bigger *zone*, that being the parent *zone*.

Attributes:

1. Name
2. Center ID: Center to which it belongs.
3. Capacity: Maximum capacity of the *zone*.
4. Occupation
5. Parent: *zoneId* of the *zone* it belongs to, if it has no parent *zone*, it is null.

- Access

The *access* are the entrances and exits of the center, it puts a name to the sensors. They are connected with the *zone* which they belong to, a future feature will be to use the same *access* on different *zones*. For now one *access* is only part of one *zone*.

The sensors will send data in a small interval to the database, specifically to the entity *recordAccess* which has two lines for the entry and the exit, known as *firstLine* and *secondLine*.

The reasoning behind not naming the lines simply “entry” and “exit” is because of scalability, as the direction of entry and exit could change if the same *access* is in two opposite side *zones*.

The *access* has attributes of the *accumulatedFirstLine* and *accumulatedSecondLine* for the sum of the *firstLine* and *secondLine* of all the *recordAccess* from that *access* since the last reset, the subtraction of them will be the occupation inside the *access*. The order of the operation will be determined by the attribute *io* or IN/OUT.

Full list of the attributes:

1. Name
2. Zone ID
3. Occupation
4. Io
5. Accumulated first line
6. Accumulated second line
7. Active: Boolean that indicates if the sensor is on/off.

- User

Each *user* of the application will have at least one *center*, but the same *user* can be on many *centers*. Same structure with the roles, those roles being administrator, super administrator or normal user, each with different capabilities and access to the app.

The administrator will be the one able to create, edit or delete *users* for its *center*.

There is also an attribute for language in case a feature of compatibility with different languages is implemented to the app in the future.

Attributes:

1. Username
 2. Created at: When it was created.
 3. Updated at: Datetime of when, If it is updated.
 4. Password
 5. Language
- Alarm

Entity for the *alarms*, those alarms can be about the occupation, or if a sensor disconnects or reconnects. If the alarms are triggered an *incident* entity will be created.

The *alarm* has the *center* to which it belongs.

If the type of *alarm* is of occupation, there are attributes for the integer of percentage (from zero to one hundred), and if it is more or less than the real percentage of occupation. For example if the *alarm* is “more than 50%”, in case that the occupation exceeds 50% of the maximum capacity, the *alarm* will be triggered.

Attributes:

1. Typealarm
2. Center ID
3. Zone ID
4. Morelessthan: Three-state value, “=”, “<” or “>”.
5. Percentage: From zero to one hundred.
6. Email: Email to send *alarm* if triggered.
7. Sent: boolean, true if the *alarm* was sent or false if not.

After the introduction to the entities that shape the database, at figure 5.1 the full database design is displayed.

It must be noted that, as it is explained in the chapter of methodology, the type of database is relation, so weak entities will be created to cover n-to-n relationships.

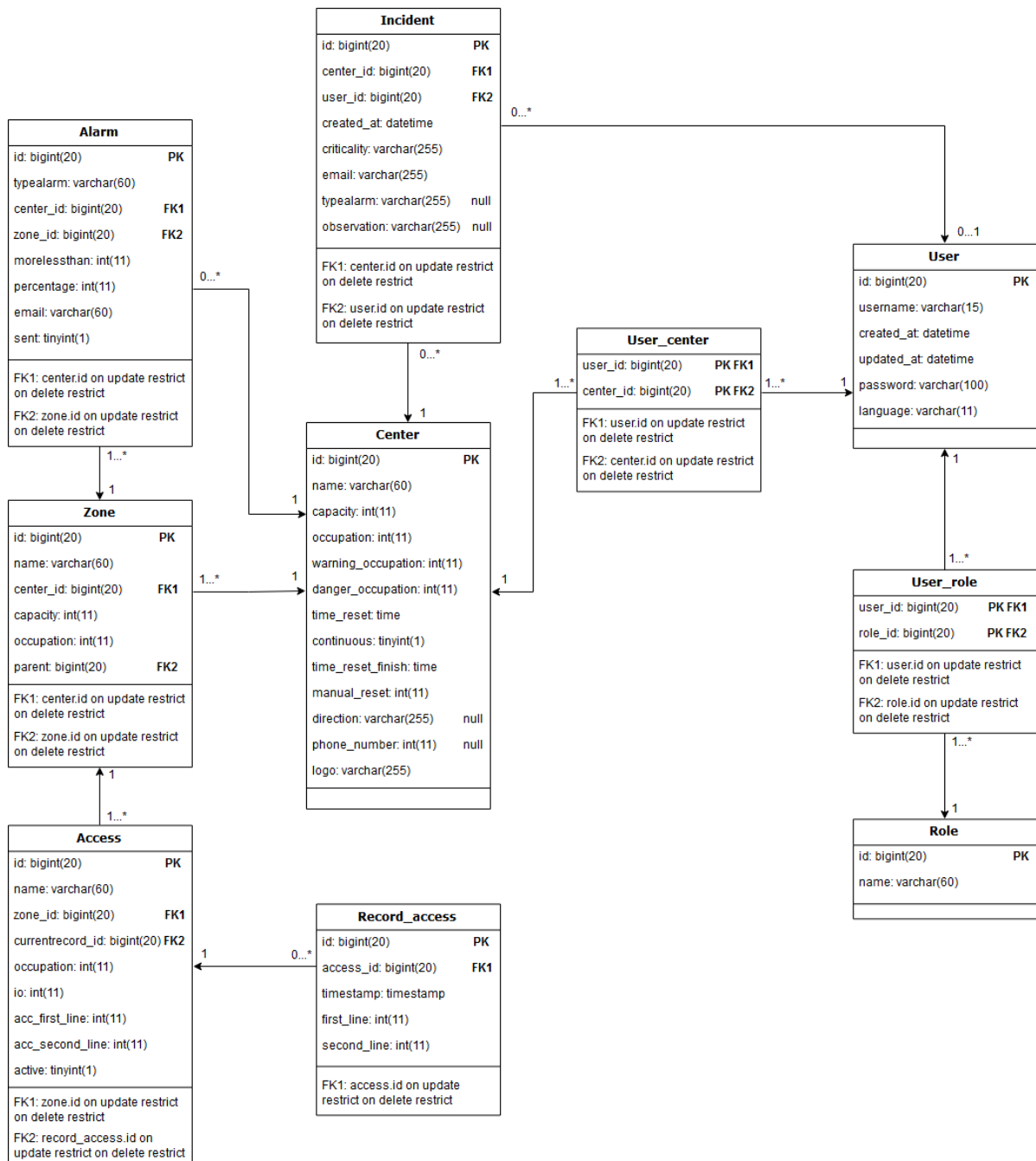


Figure 5.1: Design of the database of the project

5.2. Sensor Installation and Data Retrieving

For the sensor installation it will be better understood if we explain a real case. There are real-life implementations of the application on some sites. One of those is the nightclub *Pacha* (Ibiza, Spain).

There are five entrances in this enclosure, each with a infrared sensor and a camera. Both the sensors and the cameras are connected to the internal network and accessed externally with the IP.

On figure 5.2 you can see an example of the network structure (at the time it only had four entrances):

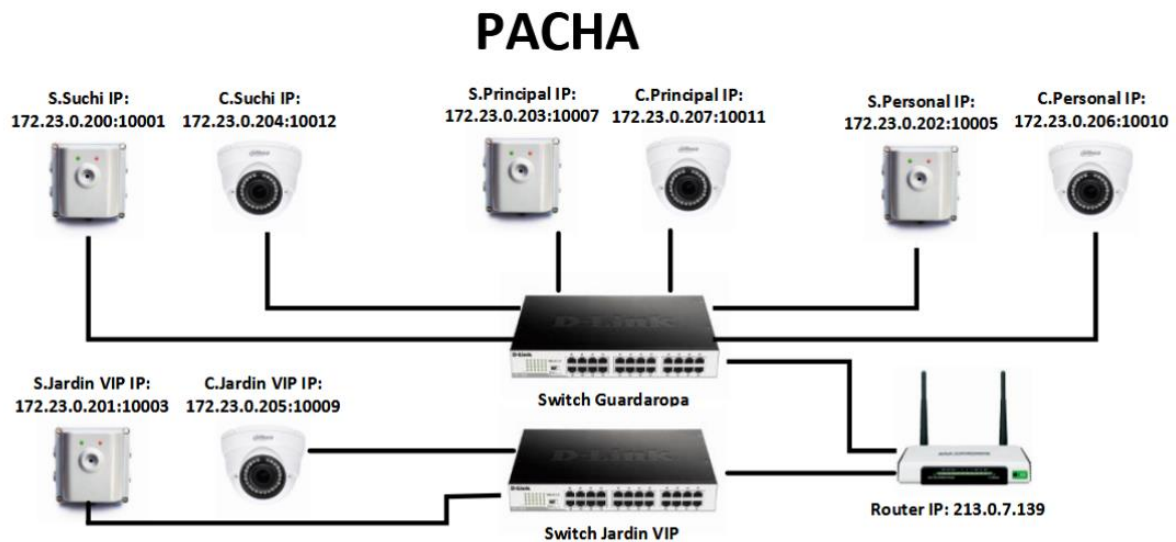


Figure 5.2: Design of the network at Pacha Ibiza

The sensor is connected through an ethernet cable only for power, using a PoE injector [29](Power over Ethernet), the sensor connects to the ethernet of the PoE, with the power connected to the other port.

On figure 5.3 you can see an example of a sensor and camera installation at *Pacha*.



Figure 5.3: Image of a sensor and camera installation at an access of Pacha Ibiza

The sensor is positioned ninety degrees perpendicularly to the ground, with the sensor pointing to the ground, so it gets an overhead view of the people passing.

The sensor has a setup tool that allows us to calibrate it, where you have to input the height from the ground to the sensor. As well as determining the line of entry and of exit where it will sum one if a person crosses the entry line, and subtract one if a person crosses the exit line. Figure 5.4 shows how the interface looks.

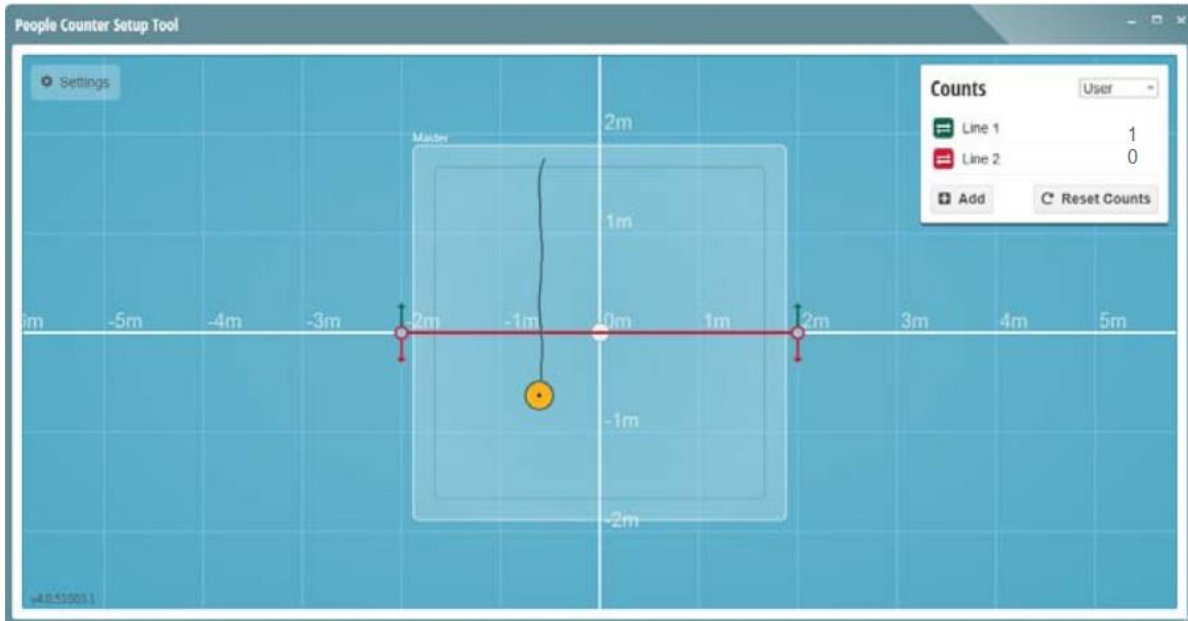


Figure 5.4: Web interface of the IP of an infrared sensor *Gazelle*.

Both of the lines will probably be set at the same place as people go in and out in the same space.

The camera helps us to check that the sensor is working properly. At figure 5.5 you can see a visual representation of it. Where “Line 1” is the entry, and “Line 2” the exit.

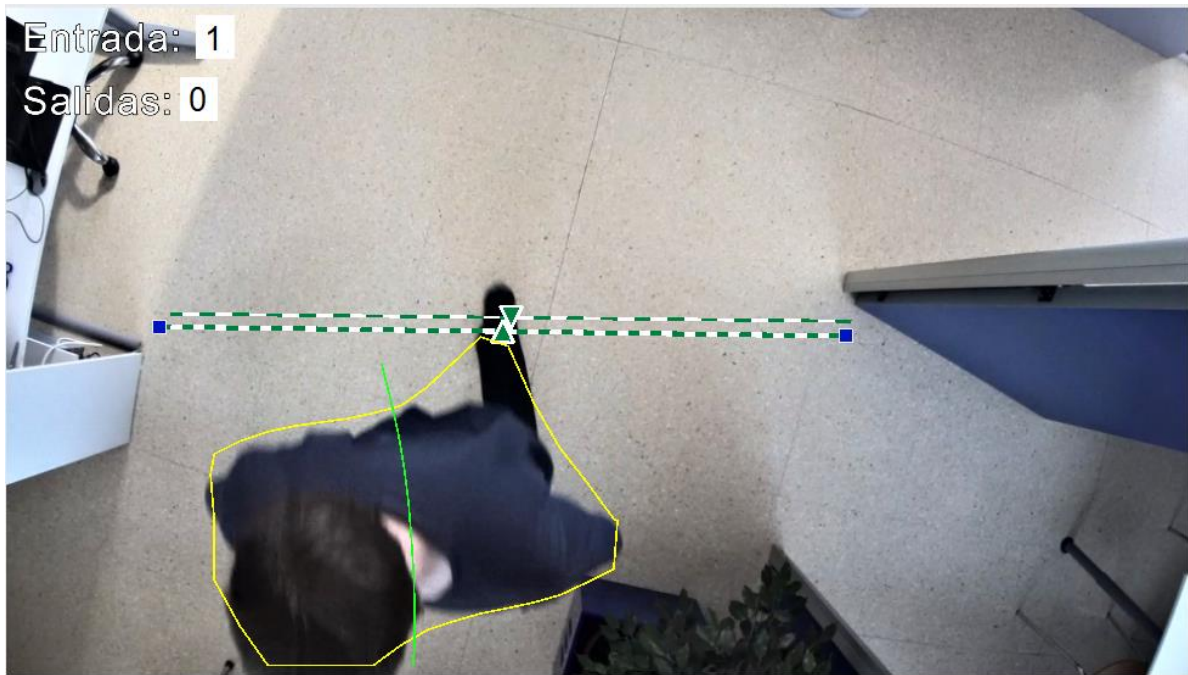


Figure 5.5: Image of a person passing by an overhead camera with entry and exit lines.

In case that the user wants gender and age recognition, the camera will have to be set on a better angle where the face is seeable. Otherwise the camera is just there to check there are not any abnormalities with the sensor counting.

When the sensors are installed, the data that they obtain needs to be inserted into the database. For that, a simple external *Java* program, not in the main backend of the WebApp, has the task to connect to sensors *APIs* and extract the data.

This program works as a server where the sensors connect through IP protocol.

In the figure 5.6 below we can see the part of the code that runs every second for each sensor that connects to the server:

```
Count livecount = blackfin.GetCurrentCount();
if (livecount != null) {
    if (DEBUG)System.out.println("Got Counts " + deviceName);

    //insert into record acces table. Actual_value - previous_value
    int ins1=livecount.countLines.get(0).intValue()-line1;
    int ins2=livecount.countLines.get(1).intValue()-line2;
    if (DEBUG)System.out.println("lcount1: " + livecount.countLines.get(0) + " lcount2: " + livecount.countLines.get(1));
    if (DEBUG)System.out.println("incr1: " + ins1 + " incr2: " + ins2);
    if (!DEBUG) {
        String query =
            "INSERT INTO `record_access` (`id`, `timestamp`, `access_id`, `first_line`, `second_line`) +
            "VALUES (NULL, CURRENT_TIMESTAMP, '" + deviceName + "', '" + ins1 + "', '" + ins2 + "')";
        MySQLDriver.getDriver().executeUpdate(query);
    }
    //update the incremental value
    line1=livecount.countLines.get(0).intValue();
    line2=livecount.countLines.get(1).intValue();

    if (DEBUG)System.out.println("Hora "+hora+":"+minutos + " count 1: " + livecount.countLines.get(0).toString() +
        " count 2: " + livecount.countLines.get(1).toString());

    while (rs.next()) {
        if (DEBUG) System.out.println(deviceName+ " - Hora "+hora+":"+minutos + " count 1: " + livecount.countLines.get(0).toString() +
            " count 2: " + livecount.countLines.get(1).toString());
        //update the last record_access and the accumulate record
        MySQLDriver.getDriver().executeUpdate(
            "UPDATE `access` SET `currentrecord_id` = '" + rs.getInt(1) +
            "' , `acc_first_line` = '" + livecount.countLines.get(0).toString()+
            "' , `acc_second_line` = '" + livecount.countLines.get(1).toString()+
            "' WHERE `access`.`id` = " + deviceName + " ;");
    }
}
}
```

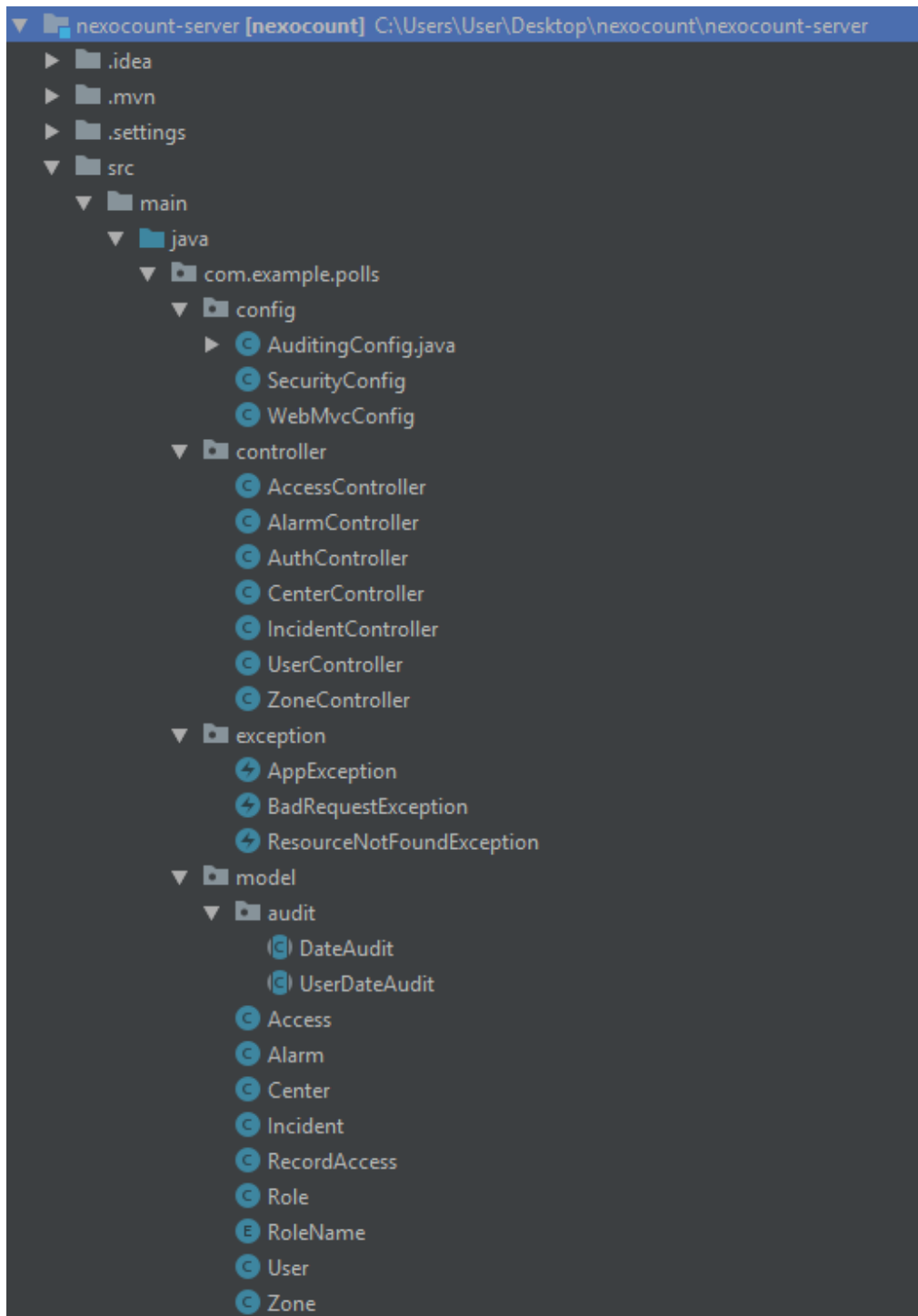
Figure 5.6: Code from the sensors data retrieving program.

It checks if there are new values from the sensor from the last time it was checked and if yes, inserts the data in a new *recordAccess* with the sensor name (*deviceName*) as the *accessId*. It also updates the accumulated *firstLine* and *secondLine* with the sum from the value that it had plus the new data from the sensor.

5.3. Backend

For the backend of the application Java is used as the main programming language, with Spring as the framework for database connection and building the REST API.

Figure 5.7 shows the structure of said backend:



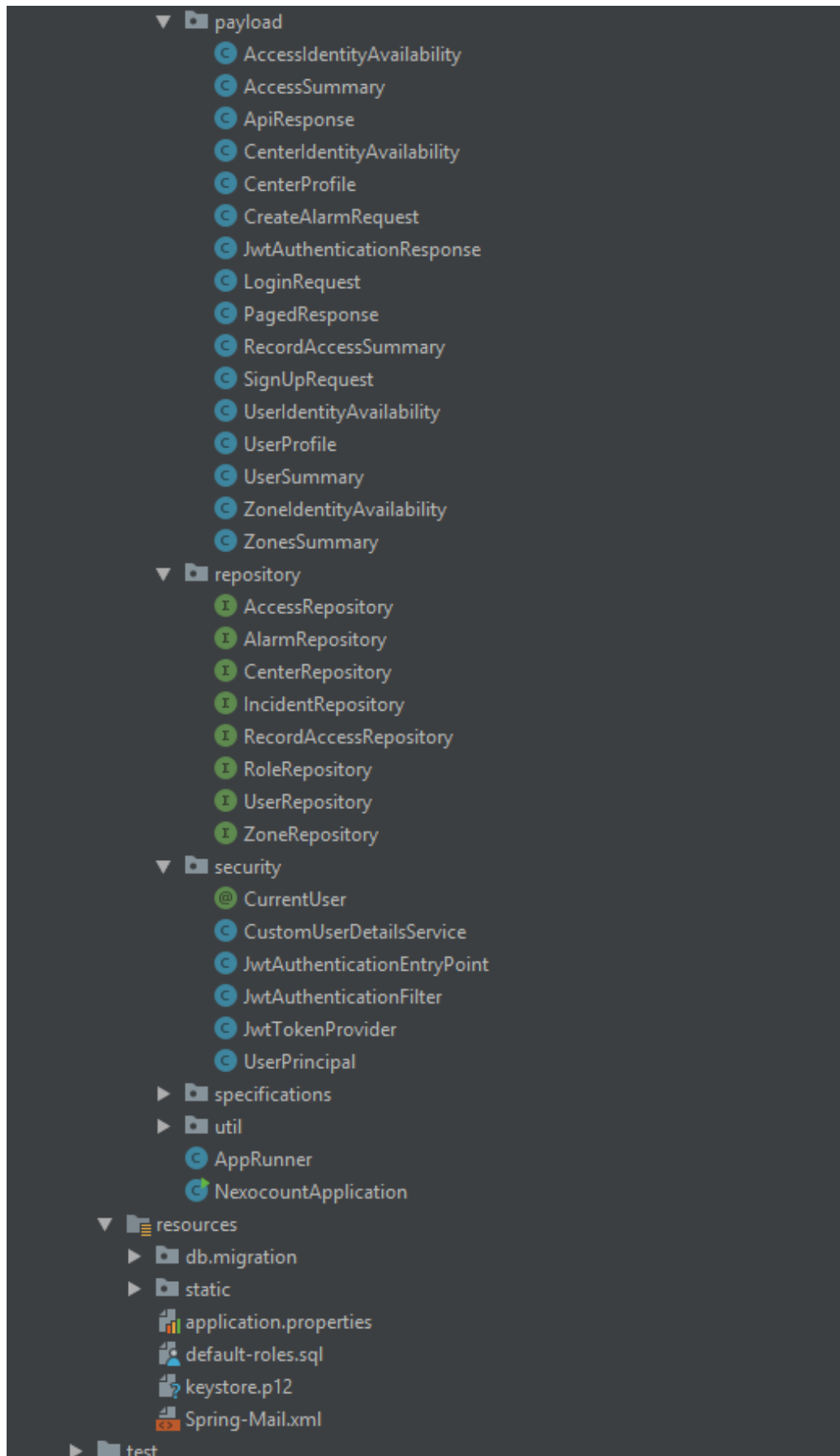


Figure 5.7: Folder structure of the backend.

It follows the spring framework guidelines of how to structure an application.

5.3.1. Model

The model folder determines the database tables, so for every database table there is a model class, for example in the class *Zone.java* seen at figure 5.8:

```
package com.example.nexocount.model;

import ...

@Entity
@Table(name = "zones")
public class Zone {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 60)
    private String name;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "center_id", nullable = false)
    private Center center;

    private Integer capacity;
    private Integer occupation;

    @ManyToOne(fetch = FetchType.LAZY, optional = true)
    @JoinColumn(name = "parent", nullable = true)
    private Zone parent;

    public Zone() {
    }

    public Zone(String name, Center center, Integer capacity, Integer occupation, Zone parent) {
        this.name = name;
        this.center = center;
        this.capacity = capacity;
        this.occupation = occupation;
        this.parent = parent;
    }
}
```

Figure 5.7: Code of the class *Zone.java*.

As you can see, there are some annotations before the class begins. First, there is the *@Entity*, which is part of JPA [30](Java Persistence JPA). JPA greatly simplifies Java persistence and provides an object-relational mapping approach that lets you declaratively define how to map Java objects to relational database tables in a standard, portable way.

Entities represent persistent data stored in a relational database automatically using container-managed persistence. They are persistent because their data is stored persistently in some form of data storage system, such as a database, in this case the MySQL database defined in the previous chapter.

The other annotation defines which table is the one that should be mapped into this class, it is not necessary as Spring does it automatically, but is always good for easier readability.

Otherwise it is a standard java class with its attributes, constructor, getters and setters, with the particularity of having to annotate with “@ManyToOne” the attributes that relate to another class.

5.3.2. Repository

This classes define the data access objects (DAO). The DAO layer usually consists of a lot of boilerplate code that can and should be simplified. The advantages of such a simplification are many: a decrease in the number of artifacts that need to be defined and maintained, consistency of data access patterns and consistency of configuration.

Spring Data takes this simplification one step forward and makes it possible to remove the DAO implementations entirely – the interface of the DAO is now the only artifact that needs to be explicitly defined.

In order to start leveraging the Spring Data programming model with JPA, a DAO interface needs to extend the JPA specific *Repository* interface – *JpaRepository*. This will enable Spring Data to find this interface and automatically create an implementation for it.

By extending the interface we get the most relevant CRUD methods for standard data access available in a standard DAO out of the box.

For each model class there is a repository to handle its persistence.

5.3.3. Controller

On the MVC pattern, if the *Model* is already defined and the *View* is the frontend, what is missing is the *Controller*. This controller works also as a *RestController*, *RESTful* applications like this one are designed to be service-oriented and return raw data (JSON/XML typically). The *Controller* is generally expected to send data directly via the HTTP response.

Example of a controller class, *AuthController.java* at figure 5.8:

```

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    UserRepository userRepository;

    @Autowired
    CenterRepository centerRepository;

    @Autowired
    RoleRepository roleRepository;

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    JwtTokenProvider tokenProvider;

    @PostMapping("/signin")
    public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {
        if(!centerRepository.existsByName(loginRequest.getCenter())) {
            return new ResponseEntity(new ApiResponse( success: false, message: "Nombre del centro incorrecto!"),
                HttpStatus.BAD_REQUEST);
        }

        User user = userRepository.findByUsername(loginRequest.getUsername())
            .orElseThrow(() -> new ResourceNotFoundException("User", "username", loginRequest.getUsername()));

        Set<Center> centersOfUser = user.getCenters();
        boolean hasCenter = false;
        for(Center center: centersOfUser){
            if(center.getName().equalsIgnoreCase(loginRequest.getCenter())){
                hasCenter = true;
            }
        }

        Center center = centerRepository.findByName(loginRequest.getCenter())
            .orElseThrow(() -> new ResourceNotFoundException("Center", "name", loginRequest.getCenter()));
        user.setCurrentCenter(center);
        User result = userRepository.save(user);

        if(!hasCenter){
            return new ResponseEntity(new ApiResponse( success: false, message: "El usuario no pertenece a este centro."),
                HttpStatus.BAD_REQUEST);
        }

        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                loginRequest.getUsername(),
                loginRequest.getPassword()
            )
        );

        SecurityContextHolder.getContext().setAuthentication(authentication);

        String jwt = tokenProvider.generateToken(authentication);
        return ResponseEntity.ok(new JwtAuthenticationResponse(jwt));
    }
}

```


Figure 5.8: Code of the class *AuthController.java*.

The annotations in this class tell us that this is a *RestController* class and it is mapped to “*api/auth*”. In the class, there is a method which returns a *ResponseEntity* and is mapped to a *POST* request thus any URL call ending with “*signin*” would be routed by the *DispatcherServlet* to the *authenticateUser* method in the *AuthController*.

5.3.4. Security

Here are the classes strictly for security functionalities. In the application, Spring security is used to ensure maximum user security. For example encryption of passwords, token authentication or URL authorization.

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.

5.3.5. Payload

Payload classes are convenient classes that are useful for direct communication between the frontend and the backend.

For example, on the sign up, we cannot create a *User* class directly, as the user only inputs the *centername*, *username* and *password*. For that, we need a class to easily handle that input, that class is *SignUpRequest*, which you can see at figure 5.9:

```

public class SignUpRequest {
    @NotBlank
    @Size(min = 3, max = 15)
    private String username;

    @NotBlank
    @Size(min = 4, max = 20)
    private String password;

    @NotBlank
    @Size(min = 3, max = 60)
    private String center;

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    public String getCenter() { return center; }

    public void setCenter(String center) { this.center = center; }
}

```

Figure 5.9: Code of the class *SignUpRequest.java*.

With this class, the frontend can send directly the values with a JSON and with the annotation “@RequestBody” automatically create an instance of the class *SignUpRequest*.

In the method *registerUser* of the class *AuthController* we can see the use of this method (figure5.10):

```

@PostMapping("/signup")
public ResponseEntity<?> registerUser(@Valid @RequestBody SignUpRequest signUpRequest) {
    if (userRepository.existsByUsername(signUpRequest.getUsername())) {
        return new ResponseEntity(new ApiResponse( success: false, message: "Username is already taken!",
        HttpStatus.BAD_REQUEST);
    }

    Center center = centerRepository.findByName(signUpRequest.getCenter())
        .orElseThrow(() -> new ResourceNotFoundException("Center", "name", signUpRequest.getCenter()));

    // Creating user's account
    User user = new User(signUpRequest.getUsername(), signUpRequest.getPassword(), center, language: "es", center);

    user.setPassword(passwordEncoder.encode(user.getPassword()));

    Role userRole = roleRepository.findByName(RoleName.ROLE_USER)
        .orElseThrow(() -> new AppException("User Role not set."));

    user.setRoles(Collections.singleton(userRole));

    User result = userRepository.save(user);

    URI location = ServletUriComponentsBuilder
        .fromCurrentContextPath().path("/users/{username}")
        .buildAndExpand(result.getUsername()).toUri();

    return ResponseEntity.created(location).body(new ApiResponse( success: true, message: "User registered successfully"));
}

```

Figure 5.10: Code of the class *AuthController.java*.

As you can observe, it allows us to directly have the input of the user as a class so we can more easily manipulate the data.

5.3.6. Security

A web application needs to be secure, as it may contain personal data from the users. For this application Spring Security is used. It integrates effortlessly to the Spring framework.

Spring Security saves a user for each session on the application. Known as “CurrentUser”, it is an interface which can be loaded whenever it is needed. To create the “CurrentUser” first it is needed to log in, when the user tries to login Spring Security gives a token if it successfully logged in (figure 5.11):

```
@Component
public class JwtTokenProvider {

    private static final Logger logger = LoggerFactory.getLogger(JwtTokenProvider.class);

    @Value("${app.jwtSecret}")
    private String jwtSecret;

    @Value("${app.jwtExpirationInMs}")
    private int jwtExpirationInMs;

    public String generateToken(Authentication authentication) {

        UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();

        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + jwtExpirationInMs);

        return Jwts.builder()
            .setSubject(Long.toString(userPrincipal.getId()))
            .setIssuedAt(new Date())
            .setExpiration(expiryDate)
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
    }

    public Long getUserIdFromJWT(String token) {
        Claims claims = Jwts.parser()
            .setSigningKey(jwtSecret)
            .parseClaimsJws(token)
            .getBody();

        return Long.parseLong(claims.getSubject());
    }
}
```

Figure 5.11: Code of the class `JwtTokenProvider.java`.

The token it is then saved to the local storage of the browser, so whenever the user opens a new tab, it is not needed to sign in again. When the user logs out, the token is then deleted from the local storage.

That token is a combination between the username and the password, it is fully encrypted for safety reliability.

Those are the main folders of the backend, there are also utility classes for application constants, and a mail sender which checks every ten seconds if there is a new alarm that got triggered and sends an email with information of it.

5.3.7. Software Engineering

Some of the functions that were firstly done were rushed as basic functionalities of the application needed to be in production under rather short deadlines for some basic functionalities.

When the app was in production it became exponentially clear that it needed better software engineering on some methods. An example of it is the *getRecords()* method on the class *CenterController.java*.

The *getRecords()* gets a request from the report creator page of the frontend, it will be explained in more detail on the frontend chapter, but for now it basically sends some filters, those being date range, schedule range, days of the week, type of data grouping (by minute, hour, day, week, month or year) and lastly the data source (meaning data from the whole *center*, a *zone* or an *access*).

The method functionality consist on getting the *RecordAccess* data that fit into these filters, as said on the “Database” chapter, *RecordAccess* has the data that is sent from each sensor of the enclosure.

At figure 5.12 you can see how the method looked like (and it is not even the full method):

```

@GetMapping(path = "/center/getrecords")
public @ResponseBody List<RecordAccess> getRecords(@RequestParam(value = "center") String name,
    @RequestParam(value = "dateStart") String dateStart, @RequestParam(value = "dateFinish") String dateFinish,
    @RequestParam(value = "agrupacion") String agrupacion, @RequestParam(value = "days") String days,
    @RequestParam(value = "zone") String zone, @RequestParam(value = "idzone") Long idZone) { // Zone = zone or

    Date dateStartServer = null;
    Date dateFinishServer = null;
    try {

        dateStartServer = new SimpleDateFormat("yy-MM-dd HH:mm:ss").parse(dateStart);
        dateFinishServer = new SimpleDateFormat("yy-MM-dd HH:mm:ss").parse(dateFinish);

    } catch (Exception e) {

    }
    ;

    boolean zoneOrAccess = (zoneRepository.existsByName(zone)) ? true : false; // true if zone, false if access

    List<String> daysList = Arrays.asList(days.split(","));
    Set<Integer> daysNum = new HashSet<Integer>();

    for(String d: daysList){
        switch(d){
            case "Domingo": daysNum.add(1); break;
            case "Lunes": daysNum.add(2); break;
            case "Martes": daysNum.add(3); break;
            case "Miércoles": daysNum.add(4); break;
            case "Jueves": daysNum.add(5); break;
            case "Viernes": daysNum.add(6); break;
            case "Sábado": daysNum.add(7); break;
            default:

        }
    }

    List<RecordAccess> records = new LinkedList<RecordAccess>();

    Set<Access> access = new HashSet<Access>();

    if(name.equals(zone)){
        Center center = centerRepository.findByName(name)
            .orElseThrow(() -> new ResourceNotFoundException("Center", "name", name));
        List<Zone> zones = zoneRepository.findByCenterId(center.getId());

        for (Zone z : zones) {
            String query = "";
            switch (agrupacion) {
                case "Hora":
                    records.addAll(recordAccessRepository.findRecordsOfZoneByHour(z, dateStartServer, dateFinishServer, daysNum));
                    break;
                case "Dia":
                    records.addAll(recordAccessRepository.findRecordsOfZoneByDay(z, dateStartServer, dateFinishServer, daysNum));
                    break;
                case "Semana":
                    records.addAll(recordAccessRepository.findRecordsOfZoneByWeek(z, dateStartServer, dateFinishServer, daysNum));
                    break;
                case "Mes":
                    records.addAll(recordAccessRepository.findRecordsOfZoneByMonth(z, dateStartServer, dateFinishServer, daysNum));
                    break;
                case "Año":
                    records.addAll(recordAccessRepository.findRecordsOfZoneByYear(z, dateStartServer, dateFinishServer, daysNum));
                    break;
                default:
                    records.addAll(recordAccessRepository.findRecordsOfZone(z, dateStartServer, dateFinishServer, daysNum));
            }
        }

        return records;
    }

    Zone z = null;
    if (zoneOrAccess) {
        z = zoneRepository.findByName(zone)
            .orElseThrow(() -> new ResourceNotFoundException("Zone", "name", name));
        access.addAll(accessRepository.findByZoneId(z.getId()));
    } else {
        Access a = accessRepository.findByName(zone)
            .orElseThrow(() -> new ResourceNotFoundException("Access", "name", name));
        access.add(a);
    }
}

```

Figure 5.12: Code of the old method `getRecords()` at class `CenterController.java`.

This method uses the *RecordAccessRepository.java* interface functions seen at figure 5.13:

```
public interface RecordAccessRepository extends CrudRepository<RecordAccess, Long> {
    List<RecordAccess> findByAccessId(Long id);

    /* Get Records by Access */

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access=:accessId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecords(@Param("accessId") Access accessId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access=:accessId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsByHour(@Param("accessId") Access accessId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access=:accessId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsByDay(@Param("accessId") Access accessId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access=:accessId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsByWeek(@Param("accessId") Access accessId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access=:accessId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsByMonth(@Param("accessId") Access accessId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access=:accessId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsByYear(@Param("accessId") Access accessId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    /* Get Records by Zone */

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access.zone=:zoneId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsOfZone(@Param("zoneId") Zone zoneId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access.zone=:zoneId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsOfZoneByHour(@Param("zoneId") Zone zoneId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access.zone=:zoneId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsOfZoneByDay(@Param("zoneId") Zone zoneId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access.zone=:zoneId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsOfZoneByWeek(@Param("zoneId") Zone zoneId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access.zone=:zoneId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsOfZoneByMonth(@Param("zoneId") Zone zoneId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);

    @Query("SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine FROM RecordAccess r WHERE r.access.zone=:zoneId AND r.timestamp>=:dateStart AND r.timestamp<=:dateFinish")
    List<RecordAccess> findRecordsOfZoneByYear(@Param("zoneId") Zone zoneId, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum);
}
```

Figure 5.14: Code of the old class *RecordAccessRepository.java*.

Both these methods work, but have obvious glaring clean code problems. Starting with the *RecordAccessRepository.java*, as a Spring Repository it uses either *CrudRepository* or *JpaRepository*, both are interchangeable and use method naming to describe what they do, for example if you create a function *findBy* plus a name of a variable of the class, it will search an object that has that variable name.

This is a pretty simple solution that works with simple queries, the problems comes when you want a complex query, as the interface does not have enough options to fit in the method-naming way. As a solution, the interface lets you create customized queries with the *@Query* annotation, where you can add a SQL-like code that Spring uses for SQL queries. Although this brings other problems, as you can send the filter parameters as the *:parameterName* but you cannot send a parameter that changes part of the query, such as the grouping by time period, so you need to create different methods for each grouping, only changing the part of the method of the *groupBy*. Also the same problem was found when getting data from all *access* of a *zone* or a single *access*.

You cannot send a string and concatenate strings as the *@Query* annotation will not work. But an alternative solution was found and will be explained in a bit.

For the *getRecords()* function it is obviously lacking cleaner code. There are repeated parts of the code that can be put into a private function, code is hardly readable and there are some badly named variables, as well as having to modify the JSON data to fit the different methods of the *recordAccessRepository*, so there is a need to find a way to only have one method on the repository.

All this bad code resulted in performance problems: the request lasted for about thirty or forty seconds depending the filter, this performance is unacceptable, we cannot have the user waiting for this long everytime they want to get information on the influx of people from a determined time.

One huge improvement was changing the server where the app was deployed, more about this will be explained in the deployment chapter, at first it seemed like the problem was solved, as it solved the query in under a second, but as the database grew bigger the query became slower, up to eight or nine seconds to get the data from a day and twenty to thirty seconds to get data from a month prior. The problem was in the code.

There was a testing process to get to the root of the problem, and although the backend code was not perfect it was not it, the time problem came from the query, in concrete the query to get the data from a *zone*, as getting data from a single *access* was fast enough.

The problem was in the way that java persistence queries work, they use an object-oriented system where to get the data from a related object you use the object itself instead of the variable:

```
SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine  
FROM RecordAccess r  
WHERE r.access.zone=:zoneId
```

But relational database as the one we use do not work that way, as the *RecordAccess* table only has the ID of the *access*, same as for the *zone* in the *access* table. So even though the query looks like it only does one select, it actually does two inside the *where* to get the *zone* object. To solve this, instead of passing the parameter *zoneId*, we pass a list of the *access* IDs, and instead of equals it uses the function *IN*.

This simple change provided the app with under a second waiting time of the query.

But there was still the problem of clean code, for that we used the fact that now both the `getRecordsByAccess()` and `getRecordsByZone()` could now be the same method, with the minor change of if it's a single `access` just send a list with only one value of that `access` ID.

For the grouping query part it was not as straightforward. There is a way to instead of using the `@Query` annotation, do the query as a string. First it is needed to create another custom interface for the repository named `CustomRecordAccessRepository.java` where we define the same method without the `@Query` annotation (figure 5.15):

```
package com.example.nexocount.repository;

import ...

public interface CustomRecordAccessRepository {
    /* Get Records */
    List<RecordAccess> findRecords(@Param("accessList") Set<Access> accessList, @Param("dateStart") Date dateStart, @Param("dateFinish") Date dateFinish,
        @Param("days") Set<Integer> daysNum, @Param("grouping") String grouping);
}

```

Figure 5.15: Code of the class `CustomRecordAccessRepository.java`.

Then, we need to implement the interface, here we can create a string of the query and execute it with the `EntityManager` object of the java `Persistence` library. This class will be `CustomRecordAccessRepositoryImpl.java` (figure 5.16):

```
public class CustomRecordAccessRepositoryImpl implements CustomRecordAccessRepository {
    @PersistenceContext
    private EntityManager entityManager;

    public List<RecordAccess> findRecords(@Param("accessList") Set<Access> accessList, @Param("dateStart") Date dateStart,
        @Param("dateFinish") Date dateFinish, @Param("days") Set<Integer> daysNum, @Param("grouping") String grouping) {
        Query query = entityManager.createQuery(
            queryString: "SELECT r, SUM(r.firstLine) as accFirstLine, SUM(r.secondLine) as accSecondLine " +
                "FROM RecordAccess r " +
                "WHERE r.access IN :accessList " +
                "AND r.timestamp >=:dateStart AND r.timestamp <=:dateFinish " +
                "AND DAYOFWEEK(r.timestamp) IN :days " +
                "GROUP BY "+grouping+" ORDER BY r.timestamp ASC");

        query.setParameter( name: "accessList", accessList);
        query.setParameter( name: "dateStart", dateStart);
        query.setParameter( name: "dateFinish", dateFinish);
        query.setParameter( name: "days", daysNum);

        List results = query.getResultList();
        return results;
    }
}

```

Figure 5.16: Code of the class `CustomRecordAccessRepositoryImpl.java`.

As you can, see the parameters have to be set manually, but this allows us to break the string and at the custom grouping part so there is not a need for different methods, we only need to use this one.

Lastly, for the method to be called on the repository, we extend the class with the custom repository. And as we do not need the other methods anymore, we can delete them, becoming a way cleaner and readable class (figure 5.17):

```
package com.example.nexocount.repository;

import ...

public interface RecordAccessRepository extends CrudRepository<RecordAccess, Long>, CustomRecordAccessRepository {
    List<RecordAccess> findByAccessId(Long id);
}
```

Figure 5.17: Code of the class *RecordAccessRepository.java*.

Finally, we change the *getRecords()* method from *CenterController.java* class to fit the new only method and in the process clean up some parts of the code.

The first part of the code changes the JSON data that comes from the frontend to fit java objects, most of this part can be put into different private methods to not have a do-it-all method.

Like the string to java date method at figure 5.18.

```
private Date stringToDate(String dateString) {
    try {
        return new SimpleDateFormat( pattern: "yy-MM-dd HH:mm:ss").parse(dateString);
    } catch (ParseException e) {
        e.printStackTrace();
        return new Date();
    }
}
```

Figure 5.18: Code of method *stringToDate()* of the class *CenterController.java*.

And the method *generateQueryOfDayOfWeek()*, that gets the string of all the days passed through the parameter and changes them to a set of integers where one is sunday until seven is saturday, which is what the SQL method *DAYOFWEEK(timestamp)* returns, as seen at figure 5.19.

```

private Set<Integer> generateQueryOfDayOfWeek(String daysOfWeekString) {
    List<String> daysList = Arrays.asList(daysOfWeekString.split(" "));
    Set<Integer> daysNum = new HashSet<>();

    for(String d: daysList){
        switch(d){
            case "Domingo": daysNum.add(1); break;
            case "Lunes": daysNum.add(2); break;
            case "Martes": daysNum.add(3); break;
            case "Miércoles": daysNum.add(4); break;
            case "Jueves": daysNum.add(5); break;
            case "Viernes": daysNum.add(6); break;
            case "Sábado": daysNum.add(7); break;
            default:
        }
    }
    return daysNum;
}

```

Figure 5.19: Code of method `generateQueryOfDayOfWeek()` of the class `CenterController.java`.

Lastly, we create a method to generate the string part of the query that does the grouping, for the minute there is not a SQL direct method like `hour(timestamp)`, we use instead `date_format(timestamp, '%Y-%m-%d %H:%i:00')` this will group by minute (figure 5.20):

```

private String generateQueryOfGrouping(String groupingFromJSON) {
    String groupingQuery = "";
    switch (groupingFromJSON) {
        case "Hora":
            groupingQuery = "hour ( r.timestamp )";
            break;
        case "Día":
            groupingQuery = "day ( r.timestamp )";
            break;
        case "Semana":
            groupingQuery = "week ( r.timestamp )";
            break;
        case "Mes":
            groupingQuery = "month ( r.timestamp )";
            break;
        case "Año":
            groupingQuery = "year ( r.timestamp )";
            break;
        default:
            groupingQuery = "date_format(r.timestamp, '%Y-%m-%d %H:%i:00')";
    }
    return groupingQuery;
}

```

Figure 5.20: Code of method `generateQueryOfGrouping()` of the class `CenterController.java`.

Then we use all these methods in the function `getRecords()`, seen at figure 5.21:

```

@GetMapping(path = "/center/getRecords")
public @ResponseBody List<RecordAccess> getRecords(@RequestParam(value = "center") String name,
                                                  @RequestParam(value = "dateStart") String dateStart, @RequestParam(value = "dateFinish") String dateFinish,
                                                  @RequestParam(value = "agrupacion") String agrupacion, @RequestParam(value = "days") String days,
                                                  @RequestParam(value = "dataSource") String dataSource, @RequestParam(value = "dataSourceType") Integer dataSourceType) {

    /* change the strings of dates from the JSON to java dates format */
    Date dateStartServer = stringToDate(dateStart);
    Date dateFinishServer = stringToDate(dateFinish);

    /* generate part of the query of the type of grouping */
    String grouping = generateQueryOfGrouping(agrupacion);

    /* generate part of the query of the day of the week filter (it has to be numbers instead of string of the day) */
    Set<Integer> daysNum = generateQueryOfDayOfWeek(days);

```

Figure 5.21: Code of part of the method `getRecords()` of the class `CenterController.java`.

The next part of the code is the one that calls the query of the repository. Before, we splitted the code if we wanted to call the `getRecordsByAccess()` or `getRecordsByZone()` methods depending on if we wanted the data from the whole *center*, a *zone* or an *access*.

Now we can call only one time the query method, also the code is way more readable and the naming of the variable which determined which data we wanted was changed from *zone* to *dataSource*, much more descriptive as this variable can be either the *center*, *zone* or *access*. Also we send the type of *dataSource* so we do not need to check if its a *center*, *zone* or *access* in the backend, as we have this information on the frontend already.

```

switch(dataSourceType) {
    case 0: // dataSource is the Center
        Center center = centerRepository.findByName(name)
            .orElseThrow(() -> new ResourceNotFoundException("Center", "name", name));
        List<Zone> zones = zoneRepository.findByCenterId(center.getId());

        for (Zone z : zones) {
            accessList.addAll(accessRepository.findByZoneId(z.getId()));
        }
        break;
    case 1: // dataSource is a zone
        Zone z = zoneRepository.findByName(dataSource)
            .orElseThrow(() -> new ResourceNotFoundException("Zone", "name", name));
        accessList.addAll(accessRepository.findByZoneId(z.getId()));
        break;
    case 2: // dataSource is a access
        Access a = accessRepository.findByName(dataSource)
            .orElseThrow(() -> new ResourceNotFoundException("Access", "name", name));
        accessList.add(a);
        break;
}

records.addAll(recordAccessRepository.findRecords(accessList, dateStartServer, dateFinishServer, daysNum, grouping));

return records;
}

```

Figure 5.22: Code of part of the method `getRecords()` of the class `CenterController.java`.

The whole function ends up like seen at figure 5.23:

```

@GetMapping(path = "/center/getrecords")
public @ResponseBody List<RecordAccess> getRecords(@RequestParam(value = "center") String name,
                                                  @RequestParam(value = "dateStart") String dateStart, @RequestParam(value = "dateFinish") String dateFinish,
                                                  @RequestParam(value = "agrupacion") String agrupacion, @RequestParam(value = "days") String days,
                                                  @RequestParam(value = "dataSource") String dataSource, @RequestParam(value = "dataSourceType") Integer dataSourceType) {

    /* change the strings of dates from the JSON to java dates format */
    Date dateStartServer = stringToDate(dateStart);
    Date dateFinishServer = stringToDate(dateFinish);

    /* generate part of the query of the type of grouping */
    String grouping = generateQueryOfGrouping(agrupacion);

    /* generate part of the query of the day of the week filter (it has to be numbers instead of string of the day) */
    Set<Integer> daysNum = generateQueryOfDayOfWeek(days);

    List<RecordAccess> records = new LinkedList<>();
    Set<Access> accessList = new HashSet<>(); // List of access for the query recordAccess.accessId IN list of access

    switch(dataSourceType){
        case 0: // dataSource is the Center
            Center center = centerRepository.findByName(name)
                .orElseThrow(() -> new ResourceNotFoundException("Center", "name", name));
            List<Zone> zones = zoneRepository.findById(center.getId());

            for (Zone z : zones) {
                accessList.addAll(accessRepository.findById(z.getId()));
            }
            break;
        case 1: // dataSource is a zone
            Zone z = zoneRepository.findByName(dataSource)
                .orElseThrow(() -> new ResourceNotFoundException("Zone", "name", name));
            accessList.addAll(accessRepository.findById(z.getId()));
            break;
        case 2: // dataSource is a access
            Access a = accessRepository.findByName(dataSource)
                .orElseThrow(() -> new ResourceNotFoundException("Access", "name", name));
            accessList.add(a);
            break;
    }

    records.addAll(recordAccessRepository.findRecords(accessList, dateStartServer, dateFinishServer, daysNum, grouping)); // find RecordAccess that pass this filters

    return records;
}

```

Figure 5.23: Code of method `getRecords()` of the class `CenterController.java`.

Applying all the knowledge from software engineering it was able to create a much shorter, faster and readable code, from 96 lines to 45 lines. It takes less than a second to get the data from the whole center in a day and less than three seconds in a month.

5.4. API

In the backend structure description it is hinted on a basic way how the API of the backend works. In this chapter it is specified the full API before dealing with the frontend structure.

It is a server-side web API, which means a programmatic interface consisting of one or more publicly exposed endpoints to a defined request–response message system, expressed in JSON, which is exposed via the web. It is by means of an HTTP-based web server.

Web 2.0 Web APIs often use machine-based interactions such as REST and SOAP. This is a RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.

The logic will be independent of the API. This will be logicless since the other documents have defined that the application will be who will apply the logic to the data received through the API.

Then the API will serve as a middleware between the database and the application. In order for a user to be able to use the API, it must have linked a token, which will allow the API to manage its permissions automatically. All calls where the opposite is not indicated, will require from this validation. Basically the only that does not require is the sign in.

The goal is to define the calls that will allow the API, its query verb, return objects and their other requirements. In addition, a description of all possible return models is attached. The queries are defined on the controllers. For example:

Query at *CenterController.java* at figure 5.24:

| | | |
|------------|--|--|
| GET | /center/{name}/dangerOccupation | Shows the percentage of dangerous occupation of a center |
| Parameters | name: name of the center | |
| Returns | An integer of the percentage of dangerous occupation | |

```
@GetMapping("/center/{name}/dangerOccupation")
public Integer getCenterDangerOccupation(@PathVariable(value = "name") String name) {
    Center center = centerRepository.findByName(name)
        .orElseThrow(() -> new ResourceNotFoundException("Center", "name", name));

    return center.getDangerOccupation();
}
```

Figure 5.24: Definition of an example of the API and code of method *getCenterDangerOccupation()* of the class *CenterController.java*.

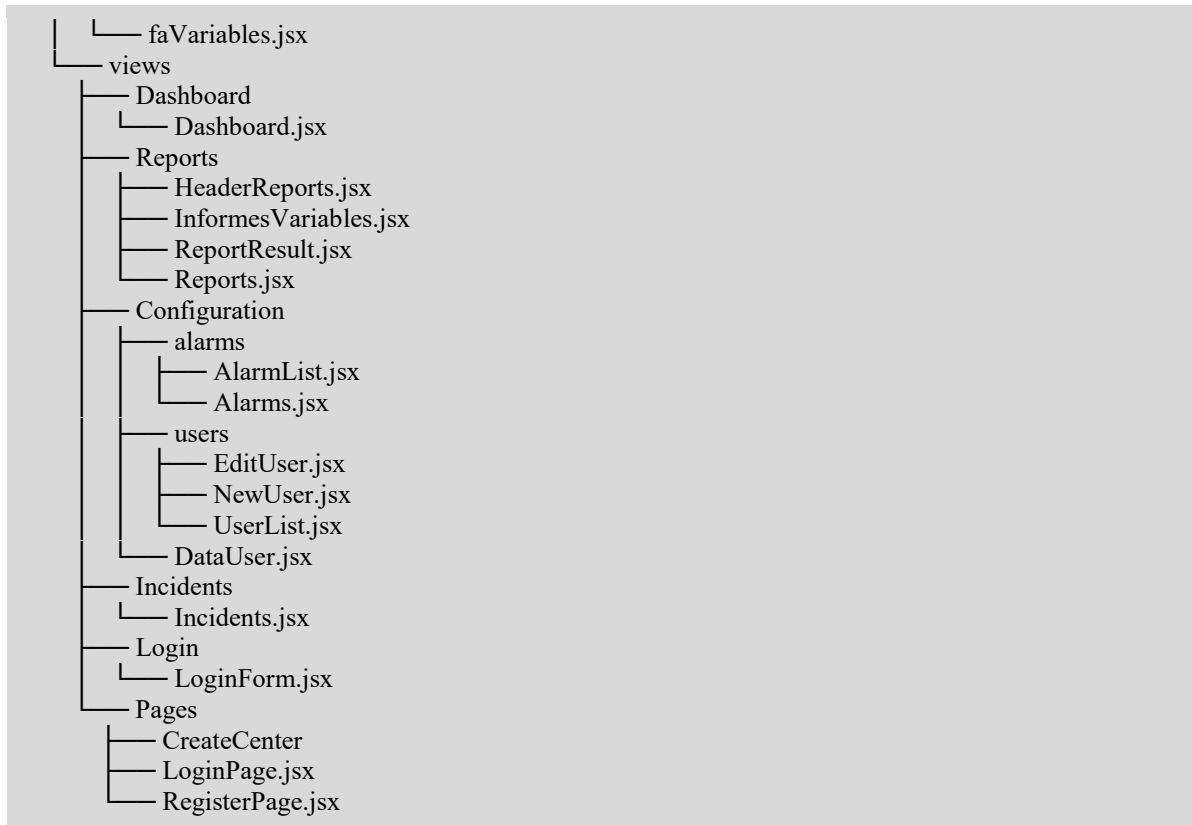
For the complete API queries you can refer to the annex.

5.5. Frontend

The frontend or client is the part of the application that interacts directly with user, therefore it is crucial that it is fast, responsive and intuitive. These principal where in thought when planning both the structure of the code and the user interface.

The frontend code is structured in the following way:

```
nexocount-client
├── CHANGELOG.md
├── README.md
├── package.json
├── Documentation
├── css
├── img
├── tutorial-components.html
├── public
│   ├── nexocount-icon.png
│   ├── favicon.ico
│   └── index.html
├── src
│   ├── app
│   │   ├── App.js
│   │   ├── Index.js
│   │   └── Constants.js
│   ├── assets
│   │   ├── css
│   │   ├── fonts
│   │   ├── img
│   │   └── sass
│   │       ├── lbd
│   │       │   ├── mixins
│   │       │   └── plugins
│   │       ├── light-bootstrap-dashboard.scss
│   │       └── light-bootstrap-dashboard.css
│   ├── components
│   │   ├── customComponents
│   │   ├── Footer.jsx
│   │   ├── Header
│   │   │   ├── Header.jsx
│   │   │   ├── HeaderLinks.jsx
│   │   │   └── PagesHeader.jsx
│   │   └── Sidebar.jsx
│   ├── util
│   │   └── APIUtils.jsx
│   ├── layouts
│   │   ├── Dashboard.jsx
│   │   └── Pages.jsx
│   ├── routes
│   │   ├── index.jsx
│   │   ├── dashboard.jsx
│   │   └── pages.jsx
│   └── variables
│       ├── Variables.jsx
│       └── chartsVariables.jsx
```



The structure was inspired by a React dashboard theme from Creative Tim [31]. So the basic skeleton of the frontend.

The folders are:

5.5.1. React General Classes

These are classes that are created with the *create-react-app* NPM command from Facebook, which offers a build setup.

Apart from the *src* classes, these include the README.md which has the frontend description and explanation of how to install and run it. The package.json with the packages installed from NPM. A documentation folder. And finally, the public folder with the website favicon and more importantly, the index.html that determines the page template.

5.5.2. Source folder structure

The source structure has the JavaScript entry point, which is `app/Index.js`, this class renders the `app/App.js` class. More about this class will be explained later, the other class from the `App` folder contains some constant variables as the name `app/Constants.js` indicates.

The other folders are: the `assets`, `components`, `layout`, `routes`, `variables` and `views`. Before explaining what each folder does, let's take a look at the `App.js`.

5.5.3. App.js

The `App` class will have the logic to check if there is a user logged in, if its logged in it sends the user to the main dashboard, if not the user is redirected to the login page. The code for this to work is explained next.

First the constructor of the class has a state with (figure 5.25):

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      currentUser: null,
      currentCenter: "default",
      isAuthenticated: false,
      isLoading: false
    };
  }
}
```

Figure 5.25: Constructor of class `App.js`.

Before the page is rendered, with the method `componentWillMount()` from React, it calls the a function that loads the current user.


```
loadCurrentUser = () => {
  this.setState({
    isLoading: true
  });
  console.log(localStorage.getItem("center"))

  if(localStorage.getItem("center")!=null){
    getCenter(localStorage.getItem("center"))
      .then(response => {
        console.log(response)
        this.setState({
          currentCenter: response
        });

        getCurrentUser()
          .then(response => {
            console.log(response.currentCenter)
            this.setState({
              currentUser: response,
              isAuthenticated: true,
              isLoading: false
            });
          })
          .catch(error => {
            this.setState({
              isLoading: false
            });
          });
      })
      .catch(error => {});
  } else {
    this.setState({
      isLoading: false
    });
  }
}
```

Figure 5.26: Code of method `loadCurrentUser()` of the class `App.js`.

This code looks at the *localStorage* from the browser, where the *center* variable should be set if the user already logged in in this browser with the *centername* of the user. If it finds the *center* it will load the *currentUser* which calls the API method from the backend that returns the current user via the *token* saved also in the *localStorage* with the user and password information.

This class also contains the methods `handleLogin()` and `handleLogout()` for both login and logout functionality:

```
// Handle Logout, Set currentUser and isAuthenticated state which will be passed to other components
handleLogout(
  redirectTo = "/pages/login",
  notificationType = "success",
  description = "Has cerrado sesión exitosamente."
) {
  localStorage.removeItem(ACCESS_TOKEN);
  localStorage.removeItem("center");

  this.setState({
    currentUser: null,
    isAuthenticated: false
  });

  this.props.history.push(redirectTo);

  notification[notificationType]({
    message: "Nexocount",
    description: description
  });

  this.props.history.push("/pages/login");
}

/*
This method is called by the Login component after successful login
so that we can load the logged-in user details and set the currentUser &
isAuthenticated state, which other components will use to render their JSX
*/
handleLogin() {
  notification.success({
    message: "Nexocount",
    description: "Has iniciado sesión exitosamente."
  });
  this.loadCurrentUser();
  this.props.history.push("/dashboard");
}
```

Figure 5.27: Code of method *handleLogout()* and *handleLogin()* of the class *App.js*.

The *handleLogin()* loads the *currentUser* again as it should already be set on the login page, also sends a notification and redirects the user to the dashboard page. Otherwise, the *handleLogout()* removes the *token* and *center* item from the *localStorage* and sets the *currentUser* variable from the state to null.

```
render() {
  if (this.state.isLoading) {
    return <LoadingIndicator />;
  }
  return (
    <Layout>
      <Content>
        <div>
          <Switch>
            <Route
              path="/pages"
              render={props => (
                <Pages handleLoginForm={this.handleLogin} handleLogoutForm={this.handleLogout} currentUser={this.state.currentUser} {...props} />
              )}
            />
            {indexRoutes.map((prop, key) => {
              return (
                <Route
                  path={prop.path}
                  key={key}
                  render={props =>
                    this.state.isAuthenticated ? (
                      <prop.component
                        handleLogout={this.handleLogout}
                        currentUser={this.state.currentUser}
                        currentCenter={this.state.currentCenter}
                        {...props}
                      />
                    ) : (
                      <Redirect
                        to={{
                          pathname: "/pages/login",
                          state: { from: props.location }
                        }}
                      />
                    )
                  }
                />
              )
            })}
          </Switch>
        </div>
      </Content>
    </Layout>
  );
}
```

Figure 5.28: Code of method `render()` of the class `App.js`.

Finally, the `render` method returns a *switch* routing where all of the pages routes are set, sending the methods and variables of the App state that were explained before. The code also checks with the `isAuthenticated` variable if the user was authenticated, and properly sends the user to either the dashboard page or the login page.

5.5.4. Routing

To organize all the app pages, React offers routing solutions. On this app, as it is observable from the code of `App.js`, there is firstly the `index.js` seen at figure 5.29:

```
import Pages from "layouts/Pages/Pages.jsx";
import Dashboard from "layouts/Dashboard/Dashboard.jsx";

var indexRoutes = [
  { path: "/pages", name: "Pages", component: Pages },
  { path: "/", name: "Home", component: Dashboard }
];

export default indexRoutes;
```

Figure 5.29: Code of the class *index.js*.

This is a simple routing app that paths both the route of *Pages.jsx* and *Dashboard.jsx*. They are divided in pages that follow the layout of the entry of the app, such as the login page, or the layout of the app when the user already logged in, such as the dashboard page.

The *Pages.jsx* route has the login page, register page and then create another center page as they all have the same layout. The *Dashboard.jsx* has all of the main app pages, like the dashboard page, reports page, configuration pages, and incidents page. For example of it is seen at figure 5.30.

```
var dashboardRoutes = [
  {
    path: "/dashboard",
    name: "Panel de Control",
    icon: "pe-7s-graph",
    component: Dashboard
  },
  {
    path: "/Reports",
    name: "Informes",
    icon: "pe-7s-note2",
    component: Reports
  },
  {
    collapse: true,
    path: "/configuration",
    name: "Configuración",
    state: "openConfiguration",
    icon: "pe-7s-edit",
    views: [
      {
        path: "/configuration/config-data-user",
        name: "Datos del Centro",
        mini: " ",
        component: DataUser
      },
      {
        path: "/configuration/edituser",
        name: "EditUser",
        mini: " ",
        component: Users
      }
    ]
  }
];
```

Figure 5.30: Code of the var *dashboardRoutes* of the class *route/Dashboard.jsx*.

5.5.5. Layout

Layout class are handy classes that allow pages that share the same parts of a page to derive from them. The pages are defined on the *views* folder, this view components are routed on either the route/*Pages.jsx* or route/*Dashboard.jsx*, the first for the pages layout, and the later for the dashboard layout.

Mainly the app has two layouts as explained before, the pages layout, which will have the following user interface (figure 5.31).

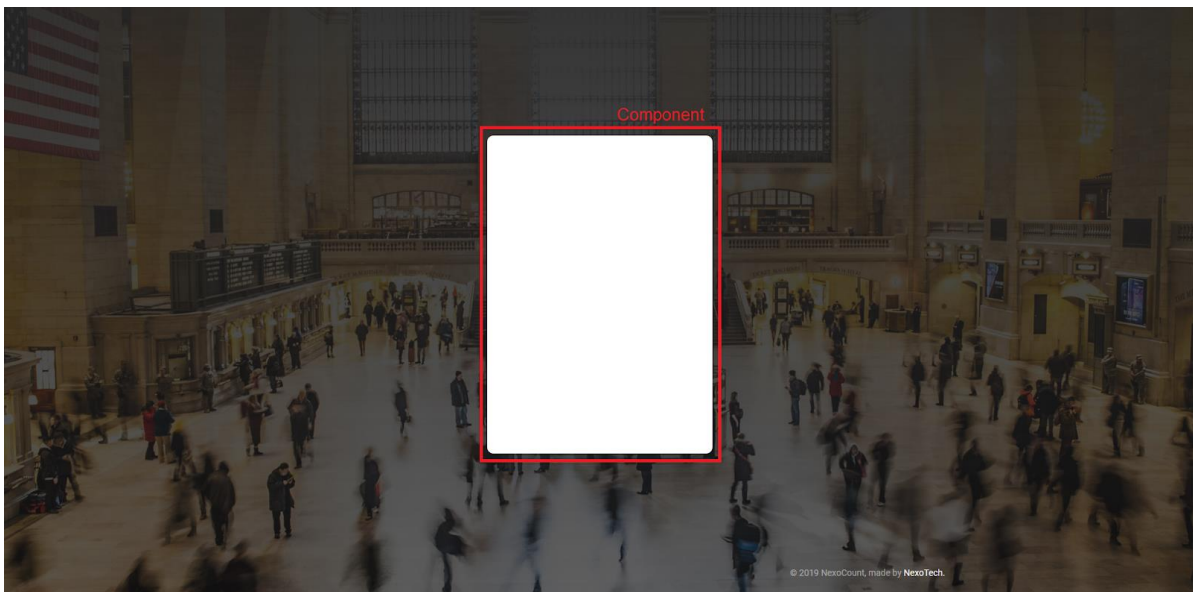


Figure 5.31: Interface of the class *layout/Pages.jsx*.

As you can observe, the view components that use this layout will be inside the red box. Here you can see the render function of this class, where we determine these view components.

```

render() {
  return (
    <div>
      <PagesHeader />
      <div className="wrapper wrapper-full-page">
        <div
          className={"full-page" + this.getPageClass()}
          data-color="black"
          data-image={bgImage}
        >
          <div className="content">
            <Switch>
              <Route path="/pages/login"
                render={(props) => <LoginPage onLogin={this.props.handleLoginForm} {...props} /></Route>
              <Route path="/pages/createNewCenter"
                render={(props) => <CreateCenter onLogout={this.props.handleLogoutForm} currentUser={this.props.currentUser} {...props} /></Route>
            </Switch>
          </div>
          <Footer transparent {...this.props} />
          <div
            className="full-page-background"
            style={{ backgroundImage: "url(" + bgImage + ")" }}
          />
        </div>
      </div>
    </div>
  );
}

```

Figure 5.32: Code of the class *layout/Pages.jsx*.

For the dashboard layout it works the same way:



Figure 5.33: Interface of the class *layout/Dashboard.jsx*.

Although it has a bit more complexity, it takes other component classes like *Header.jsx*, *Sidebar.jsx* and *Footer.jsx*. The render code for it to work is the following at figure 5.34:

```
render() {
  return (
    <div className="wrapper">
      <NotificationSystem ref="notificationSystem" style={style} />
      <Sidebar currentCenter={this.props.currentCenter} currentUser={this.props.currentUser} {...this.props} />
      <Header {...this.props} />
      <Switch>
        {dashboardRoutes.map((prop, key) => {
          return prop.views.map((prop, key) => {
            <Route
              path={prop.path}
              key={key}
              render={props => (
                <prop.component
                  currentUser={this.props.currentUser}
                  currentCenter={this.props.currentCenter}
                  {...props}
                />
              )}
            />
          )}
        )}
      </Switch>
      <Footer fluid {...this.props} />
    </div>
  );
}
```

Figure 5.34: Interface of the class `layout/Dashboard.jsx`.

Wrapping the view component there is the sidebar, header and footer. Now, instead of defining each view component one by one, we take the `dashboardRoutes` that were defined before with all the view components that we want to use with this layout. Note that it could be done the same way with `pagesRoutes` on the pages layout.

5.5.6. Assets

The assets folder contains all the app static images, other images such as the center icon are saved in the cloud. Also it contains some fonts that are not on css. Lastly, the sass folder contains most of the CSS of the webApp in form of *Sassy CSS*.

As explained in a prior chapter, SCSS is a framework of CSS that allows better organizing and implementation of CSS.

For each React component we can easily add CSS to it by creating a SCSS class with the same name and then importing it with the `light-bootstrap-dashboard.scss` class, which works as a router of all SCSS classes, this class will then be compiled to CSS creating a new CSS class equivalent to all of the SCSS classes. This may seem confusing at first, but works wonders with scaling matters.

5.5.7. Components

These are some React components that are not specifically a page by itself, but are used in some pages. Most of them are custom implementation of basic React components such as buttons or checkboxes. But some of them are big components that play a big part in the layout.

These components are mostly used on *DashboardLayout.jsx*. Such as *Header.jsx*, that renders this user interface, with its functionality of showing the page name and a logout button. This button will call the method from the *App.js* class *handleLogout()* that was passed through the props. The user interface of this component can be seen in the prior page.

Also there is the *Sidebar.jsx* component, also used on the dashboard layout, takes the components from the *dashboardRoute.jsx* plus the *CreateCenter.jsx* component and shows them to the user so they can switch between pages. It needs to be said that depending on the restrictions of the user roles some of the page routes will not be shown so they are not able to access them. The sidebar also shows and the name and icon of the center, the username currently logged in and the logo of the app NexoCount.

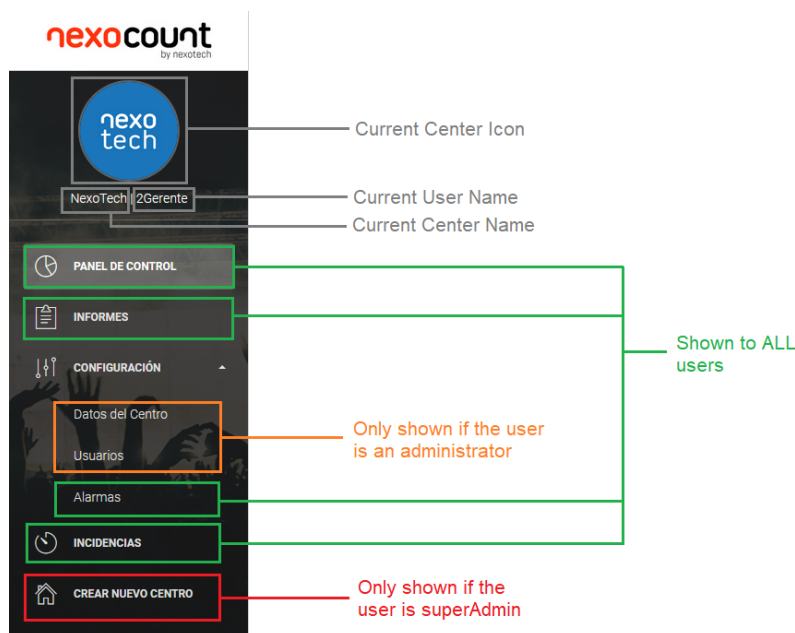


Figure 5.35: Interface explanation of *layout/Dashboard.jsx*.

For the class to know the role of the user it has to ask the backend, then with conditional statements filter depending on the figure shown above.

5.5.8. Login

The login folder contains the login form with the inputs of the center name, username and password and a button to trigger the login method named *handleSubmit()*.

```
handleSubmit(event) {
  this.setState({ loading: true });
  event.preventDefault();
  this.props.form.validateFields((err, values) => {
    if (!err) {
      const loginRequest = Object.assign({}, values);
      login(loginRequest)
        .then(response => {
          localStorage.setItem(
            ACCESS_TOKEN,
            response.accessToken
          );
          localStorage.setItem("center", loginRequest.center);
          getCenter(loginRequest.center)
            .then(response => {
              this.setState({
                currentCenter: response
              });
            })
            .catch(error => {});

          localStorage.setItem("center", loginRequest.center);
          this.props.onLogin();
          this.setState({ loading: false });
        })
        .catch(error => {
          this.setState({ errors: { global: error.message } });
          if (error.status === 401) {
            this.setState({
              errors: {
                global:
                  "Tu usuario o contraseña son incorrectos. Por favor intentalo otra vez!"
              }
            });
          }
          this.setState({ loading: false });
        });
    } else {
      this.setState({ loading: false });
    }
  });
}
```

Figure 5.36: Code of the method *handleSubmit()* of class *loginForm.jsx*.

It validates the fields of the form and if there is not an error such as password length, it sends the *loginRequest* to the backend on the “api/auth/signin” URL with a POST type of request that returns either a *responseEntity* with an error or with the token if correctly logged in. The method then adds both the token and the *centerName* in the *localStorage*, in case of not being successfully logged in it notices the user with the type of error.

It also calls the `onLogin()` method which itself calls the `handleLogin()` method of the `App.js` class sended via props.

5.5.9. API Calls

On some of the frontend code explained, there are some methods which call the API, those method use a fetch function that React has. This function is implemented in the class `util/APIUtils.jsx`.

```
import { API_BASE_URL, ACCESS_TOKEN } from '../app/Constants';

const request = (options) => {
  const headers = new Headers({
    'Content-Type': 'application/json',
  })

  if(localStorage.getItem(ACCESS_TOKEN)) {
    headers.append('Authorization', 'Bearer ' + localStorage.getItem(ACCESS_TOKEN))
  }

  const defaults = {headers: headers};
  options = Object.assign({}, defaults, options);

  return fetch(options.url, options)
    .then(response => (response)
      .json().then(json => {
        if(!response.ok) {
          return Promise.reject(json);
        }
        return json;
      })
    )
    .catch((error) => {
      return Promise.reject(error);
    })
};
```

Figure 5.37: Code of the method `request()` of class `util/APIUtils.jsx`.

All of the methods that are in this class send options of the request to the `request` method, which does the actual fetch and handles the response with React Promises. Noted that anytime the frontend tries to call the backend it has to send the `token` of authentication for the backend to respond.

For each API call there is a method, for example the method that calls the URL “api/auth/login” seen at figure 5.38.

```
export function login(loginRequest) {  
  return request({  
    url: API_BASE_URL + "/auth/signin",  
    method: 'POST',  
    body: JSON.stringify(loginRequest)  
  });  
}
```

Figure 5.38: Code of the method `login()` of class `util/APIUtils.jsx`.

It calls the request method from above with the options specified, which will be the URL of the API, the type of request, in this case POST, and a JSON with the data that the frontend wants to pass to the backend.

5.5.10. Views

Finally, the view components will be divided by the user interface pages. First the views that use the pages layout:

Login

The login page will basically show the `LoginForm.jsx`, there is not a sign up page as if there is a need to add a user, it will be done by a user admin inside the application.

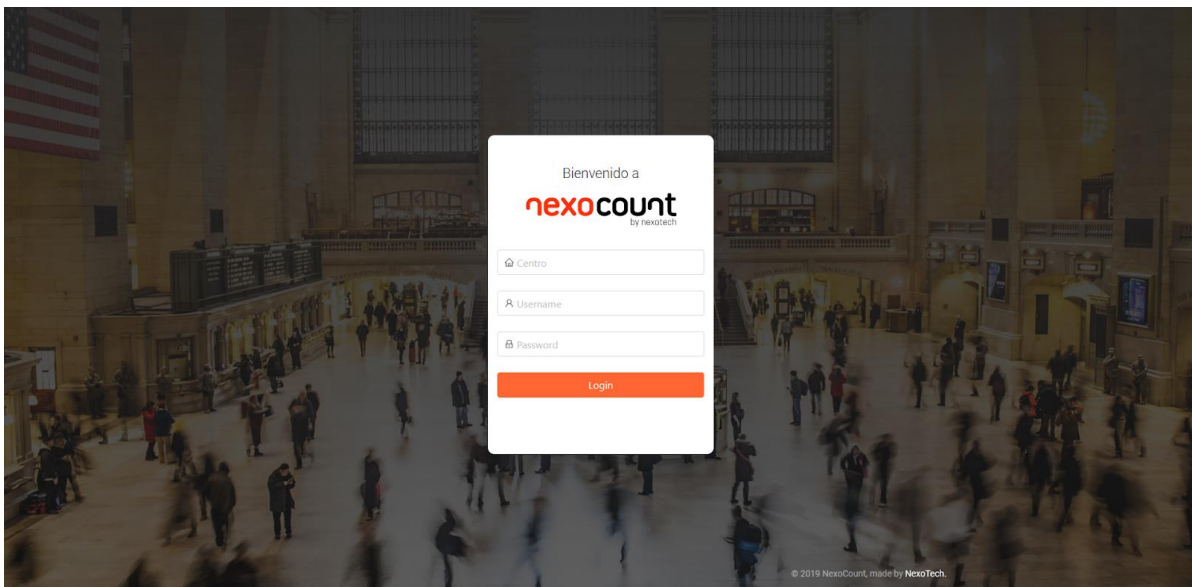


Figure 5.39: Interface of the class `LoginForm.jsx`.

CreateCenter

The *CreateCenter* page will be divided into different steps using a “wizard” form. It uses a React component called *StepZilla*, where you pass the pages as “steps”.

```
render() {
  const steps = [
    { name: "Centro", component: <StepCenter getStore={() => (this.getStore())} updateStore={u => {this.updateStore(u)}} changeLoading={() => (this.changeLoadingState())}/> },
    { name: "Zonas", component: <StepZonas getStore={() => (this.getStore())} updateStore={u => {this.updateStore(u)}} changeLoading={() => (this.changeLoadingState())}/> },
    { name: "Accesos", component: <StepAccess getStore={() => (this.getStore())} updateStore={u => {this.updateStore(u)}} changeLoading={() => (this.changeLoadingState())}/> },
    { name: "Confirmación", component: <StepConfirmation getStore={() => (this.getStore())} updateStore={u => {this.updateStore(u)}} onLogout={this.props.onLogout} /> }
  ];
  return (
    <Grid>
      <Row>
        <Col>
          <Button
            onClick={() => this.props.history.push("/dashboard")}
          >
            <i className={"pe-7s-angle-left-circle"} />
            {""}
            Cancelar
          </Button>
        </Col>
      </Row>
      <Spin spinning={this.state.loading} indicator={antIcon} />
      <Row>
        <Col md={8} mdOffset={2}>
          <Card>
            wizard
            id="wizardCard"
            textCenter
            title="Crear nuevo centro"
            category="Completa los pasos"
            content={
              <StepZilla
                steps={steps}
                stepsNavigation={false}
                nextButtonCls="btn btn-prev btn-info btn-fill pull-right btn-wd"
                backButtonCls="btn btn-next btn-default btn-fill pull-left btn-wd"
                nextButtonText= "Siguiente"
                backButtonText= "Anterior"
              />
            }
          </Card>
        </Col>
      </Row>
    </Spin>
  </Grid>
);
}
```

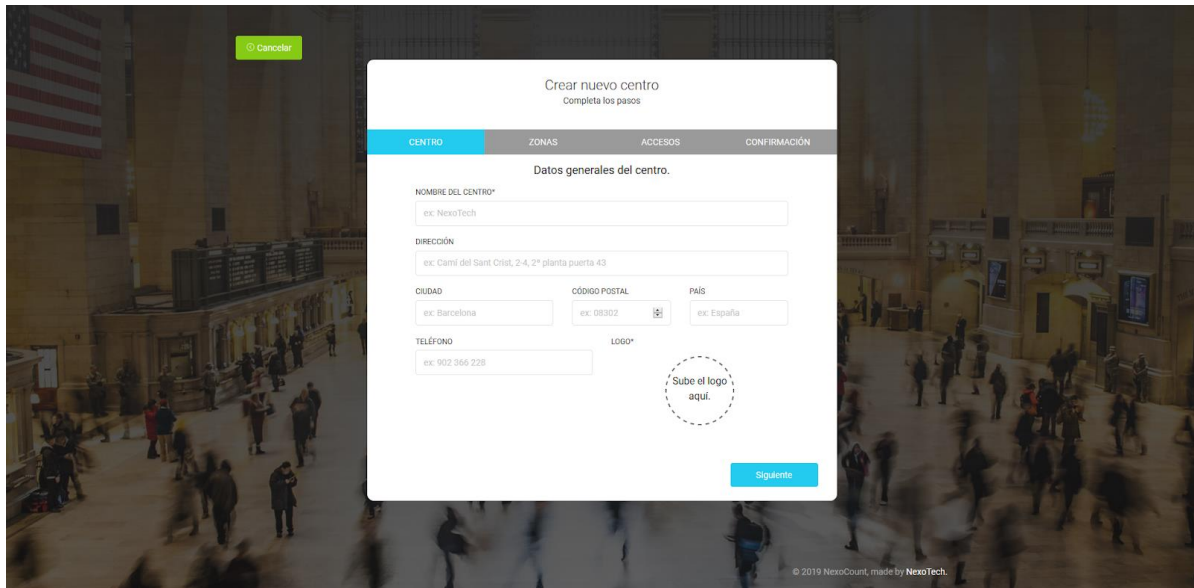
Figure 5.40: Code of the method *render()* of the class *CreateCenter.jsx*.

On these “steps” we pass a function to recollect all the data that the user fills in the variable *store*, so in the last “step” it sends the API request to create the center with this *APIUtils.jsx* method.

```
export function createCenter(centername, direction, phoneNumber, logo, username, zones, access) {
  return request({
    url: encodeURI(API_BASE_URL + "/center/createCenter?centername=" + centername + "&direction=" + direction + "&phoneNumber=" +
      phoneNumber + "&logo=" + logo + "&username=" + username + "&zones=" + zones + "&access=" + access),
    method: 'POST'
  });
}
```

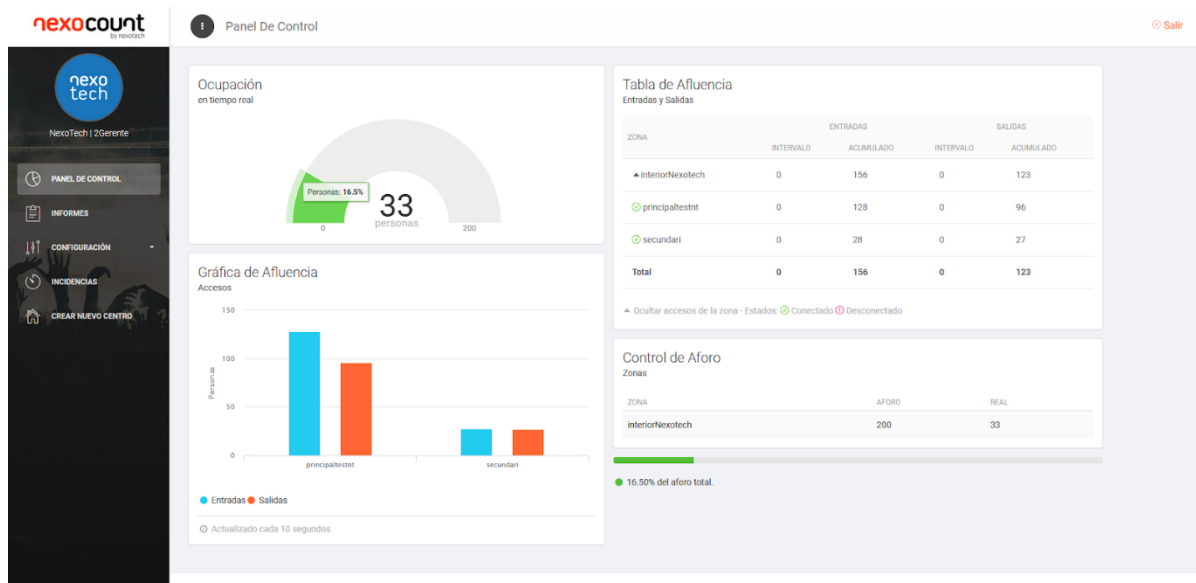
Figure 5.41: Code of the method *createCenter()* of class *util/APIUtils.jsx*.

At figure 5.42 you can see how the user interface looks like:

Figure 5.42: Interface of class *CreateCenter.js*.

Dashboard

Same name as the layout, it is the first page that the user sees when it logs in, it shows real-time data of the occupation of the center. As well as information of the occupation broken down by *zone* or *access*. It also shows in different type of charts the occupation percentage related to the maximum capacity of the enclosure.

Figure 5.43: Interface of class *Dashboard.js*.

A high definition image with guides on what each piece of information is about can be found on the annex.

Reports

The reporting system page brings to the user the ability to get data from the entries, exits and total occupation on the user's center. This data can be filtered by dates, schedule, day of the week and data source (data from all of the center, only a zone or only an access). And it can be grouped by time period (minute, hour, day, week, month or year).

When the users input the filters, the users can then click whenever they want to the generate button. Then, the data that the user asked then shows in two forms, a row chart and a table.

The chart has the number of people on the "x" and the time period on the "y", there are two rows, one for the n° of entries and the other for the n° of exits on each time period. The table will show the same data with the columns being timestamp, entries, exits and occupation.

The page is divided by the set of filters and buttons on the right, and the results on the right.

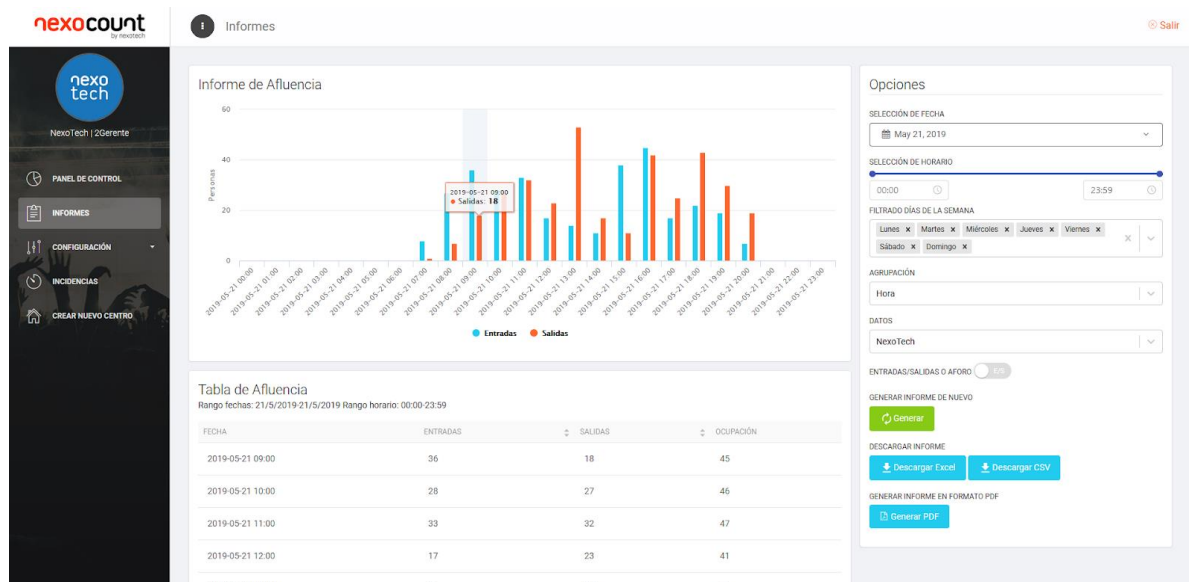


Figure 5.44: Interface of class *Reports.js*.

The data can also be downloaded as a report file in PDF, CSV and spreadsheet. The PDF report will show a preview of it in the app itself.

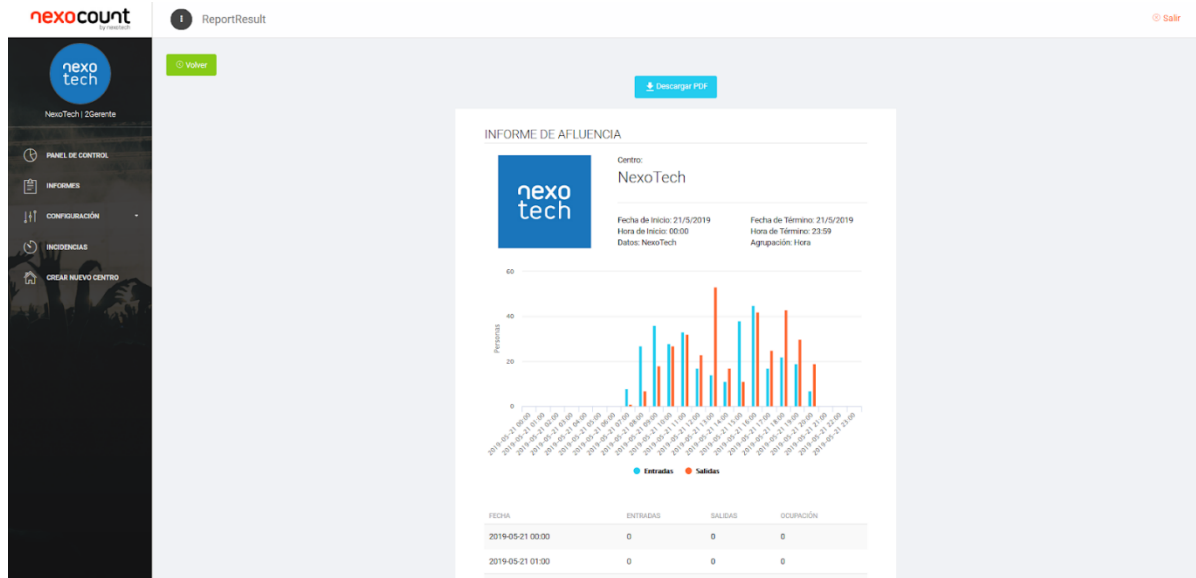


Figure 5.45: Interface of class *ReportResult.js*.

Configuration

The configuration tab is divided by the configuration of general information from the site, the configuration of users and the configuration of alarms.

The general information can only be changed by administrator users of the site, it allows them to change the type of schedule of the site, continuous or partial and the time of the entries and exits reset. The user can also ask for a manual reset that is not on the time of the day reset. They can change the name and maximum occupation of the zones and the name of the access as well. At last, there is also a slider for the warning and danger percentage of occupation.

Figure 5.46: Interface of class *DataUser.js*.

The users configuration can only be accessed by administrators, it shows the component *UsersList.jsx* with the list of users of the site with their name, role and edit or delete options. Finally there is an option to create a new user at the end of the list.

When the edit button of a user is clicked the app sends them to the page with component *EditUser.jsx* that allows the administrator user to change the password, name and role of the other users of the center.

Figure 5.47: Interface of class *EditUser.js*.

The edit page looks the same as the “creating a new user” one mentioned before.

At last, there is the alarm configuration page, available for every user. It is structured the same way as the user configuration page, in this case with the list of the alarms (*AlarmList.jsx*).

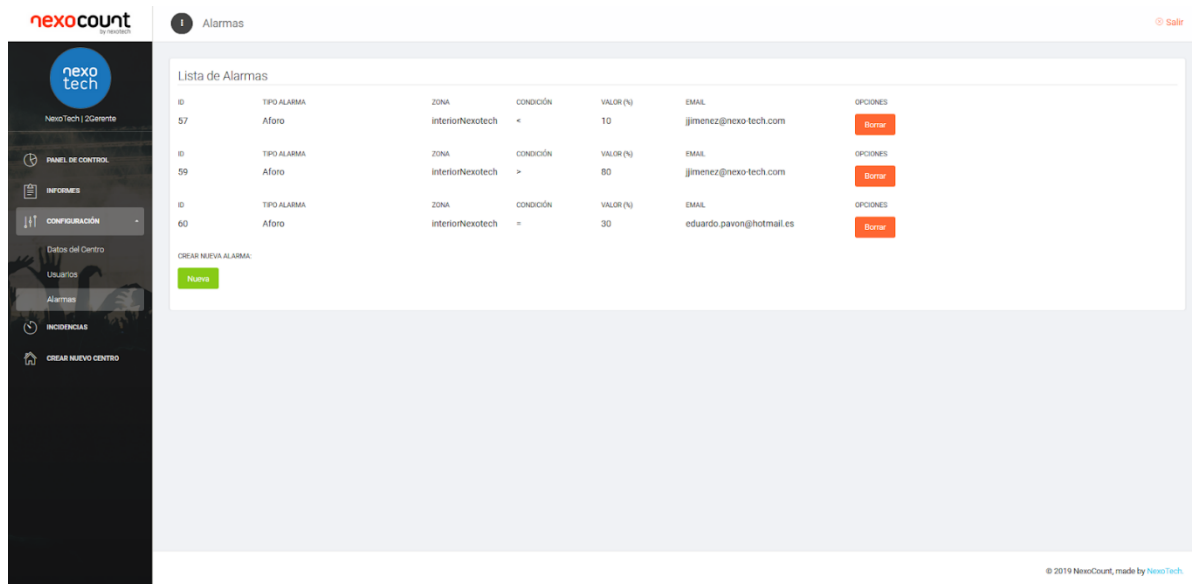


Figure 5.48: Interface of class *UserList.js*.

With a button to create a new alarm, that sends the user to the *NewAlarm.jsx* page.

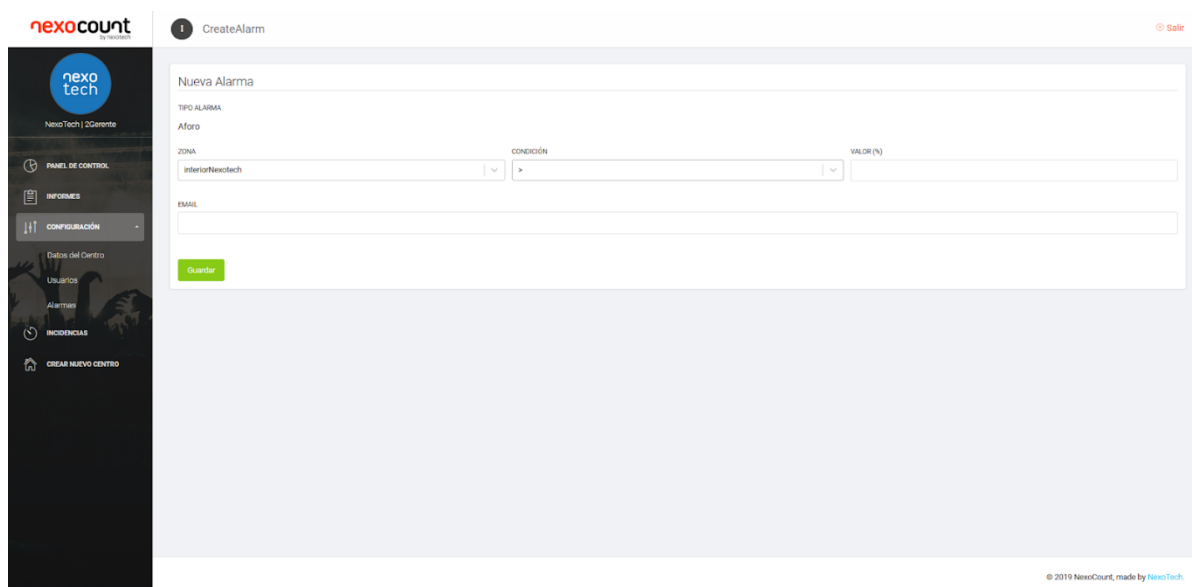


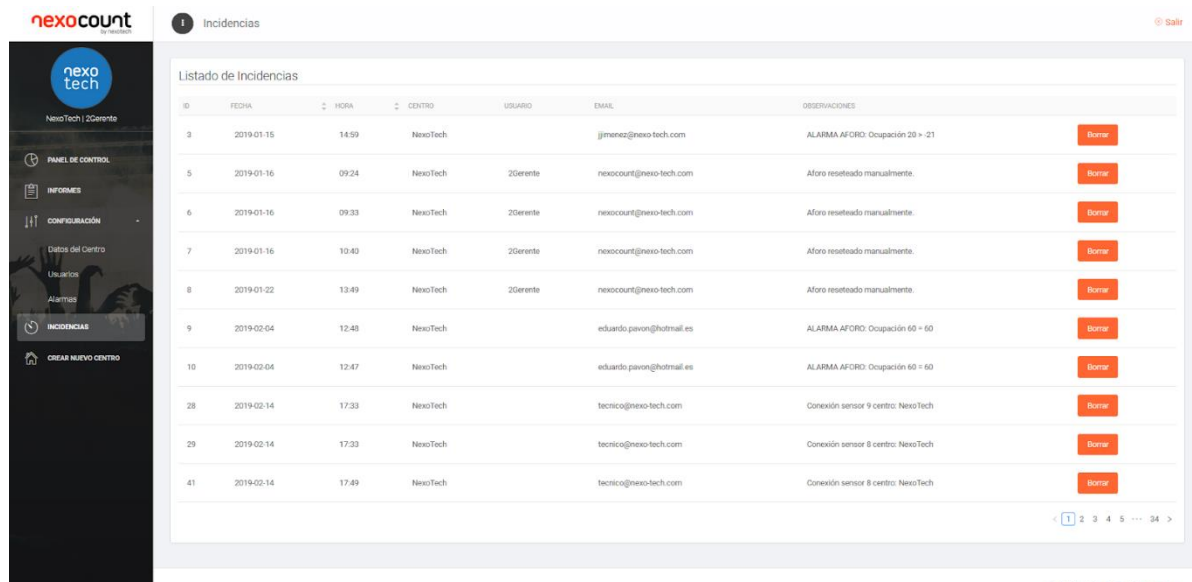
Figure 5.49: Interface of class *CreateAlarm.js*.

New alarm configuration has only the option to create an occupation percentage type of alarm for now (there are plans to allow other type of alarms on the future), which lets the

user input the *zone*, type of equation (less than, more than or equals), the percentage, and the email where a message will be sent when the alarm with the inputted conditions is triggered.

Incidents

The incidents page shows a list of the incidents that occur on the center, this can be alarm triggers, percentage occupation above the warning or danger percentage set on the center configuration page, or if a sensor was disconnected or connected again.



| ID | FECHA | HORA | CENTRO | USUARIO | EMAIL | OBSERVACIONES |
|----|------------|-------|----------|----------|--------------------------|------------------------------------|
| 3 | 2019-01-15 | 14:59 | NexoTech | | jimenez@nexo-tech.com | ALARMA AFORO: Ocupación 20 > 21 |
| 5 | 2019-01-16 | 09:24 | NexoTech | 2Gerente | nexocount@nexo-tech.com | Aforo reseteado manualmente. |
| 6 | 2019-01-16 | 09:33 | NexoTech | 2Gerente | nexocount@nexo-tech.com | Aforo reseteado manualmente. |
| 7 | 2019-01-16 | 10:40 | NexoTech | 2Gerente | nexocount@nexo-tech.com | Aforo reseteado manualmente. |
| 8 | 2019-01-22 | 13:49 | NexoTech | 2Gerente | nexocount@nexo-tech.com | Aforo reseteado manualmente. |
| 9 | 2019-02-04 | 12:48 | NexoTech | | eduardo.pavon@hotmail.es | ALARMA AFORO: Ocupación 60 = 60 |
| 10 | 2019-02-04 | 12:47 | NexoTech | | eduardo.pavon@hotmail.es | ALARMA AFORO: Ocupación 60 = 60 |
| 28 | 2019-02-14 | 17:33 | NexoTech | | tecnico@nexo-tech.com | Conexión sensor 9 centro: NexoTech |
| 29 | 2019-02-14 | 17:33 | NexoTech | | tecnico@nexo-tech.com | Conexión sensor 8 centro: NexoTech |
| 41 | 2019-02-14 | 17:49 | NexoTech | | tecnico@nexo-tech.com | Conexión sensor 8 centro: NexoTech |

Figure 5.50: Interface of class *Incidents.js*.

The columns are the ID of the incident, datetime, center, user (if it was an alarm set by a user), email where the message is sent and observations.

When an incident happens, as said, the application sends an email to the email direction specified on the configuration page.

There are some issues with sending an email at the time of the incident trigger, as a RESTful app only allows one direction communication, the frontend can talk to the backend whenever it wants, but the other way around is not possible. But we want the backend to send the email when an incident happens, not only when a user opens the app on the browser.

There are a handful of solutions to this problem, but the one used in this application consist on having the backend main function open an async executor that executes a function every ten seconds which checks the incidents on the database for new incidents, that is why there is a variable on the table incidents with the name “sent”, if this is not true, the backend sends a mail with the warning and sets this variable to true.

```
package com.example.nexocount;

import ...

@SpringBootApplication
@EntityScan(basePackageClasses = {
    NexocountApplication.class,
    Jsr310JpaConverters.class
})
public class NexocountApplication extends SpringBootServletInitializer {

    @PostConstruct
    void init() { TimeZone.setDefault(TimeZone.getTimeZone("UTC")); }

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(NexocountApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(NexocountApplication.class, args);
    }

    @Bean
    public Executor asyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(2);
        executor.setMaxPoolSize(2);
        executor.setQueueCapacity(500);
        executor.setThreadNamePrefix("DatabaseLookupService-");
        executor.initialize();
        return executor;
    }

    @Bean
    public ConfigurableServletWebServerFactory webServerFactory() {
        TomcatServletWebServerFactory factory = new TomcatServletWebServerFactory();
        factory.addConnectorCustomizers(new TomcatConnectorCustomizer() {
            @Override
            public void customize(Connector connector) { connector.setProperty("relaxedQueryChars", "{(,)}"); }
        });
        return factory;
    }
}
```

Figure 5.51: Code of class *NexocountApplication.java*.

The `@Bean` annotation allows us to this. The `asyncExecutor()` triggers the *DatabaseLookupService.java* service.

```
package com.example.nexocount.util;

import ...

@Service
public class DatabaseLookupService {

    private static final Logger logger = LoggerFactory.getLogger(DatabaseLookupService.class);

    private final RestTemplate restTemplate;

    @Autowired
    private CenterController centerController;

    public DatabaseLookupService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    @Async
    public CompletableFuture<Boolean> checkSendAlarms() throws InterruptedException {
        while(true) {
            try {
                //sending the actual Thread of execution to sleep X milliseconds
                Thread.sleep( millis: 10000);
            } catch(InterruptedException ie) {}
            centerController.sendAllAlarms();
        }
    }
}
```

Figure 5.52: Code of class DatabaseLookupService.js.

This service does in an async way while the backend is running the method *checkSendAlarms()* which triggers the function *sendAllAlarms()* explained before every ten seconds.

5.5.11. Responsiveness

It is important to think about other screen sizes as the app is mostly used by security employees who are on site and moving all the time and can only use it through mobile or tablet. So the application is responsive and tries to clearly show the real-time occupation of the site as the focal visual point. As seen at figure 5.53 and 5.34

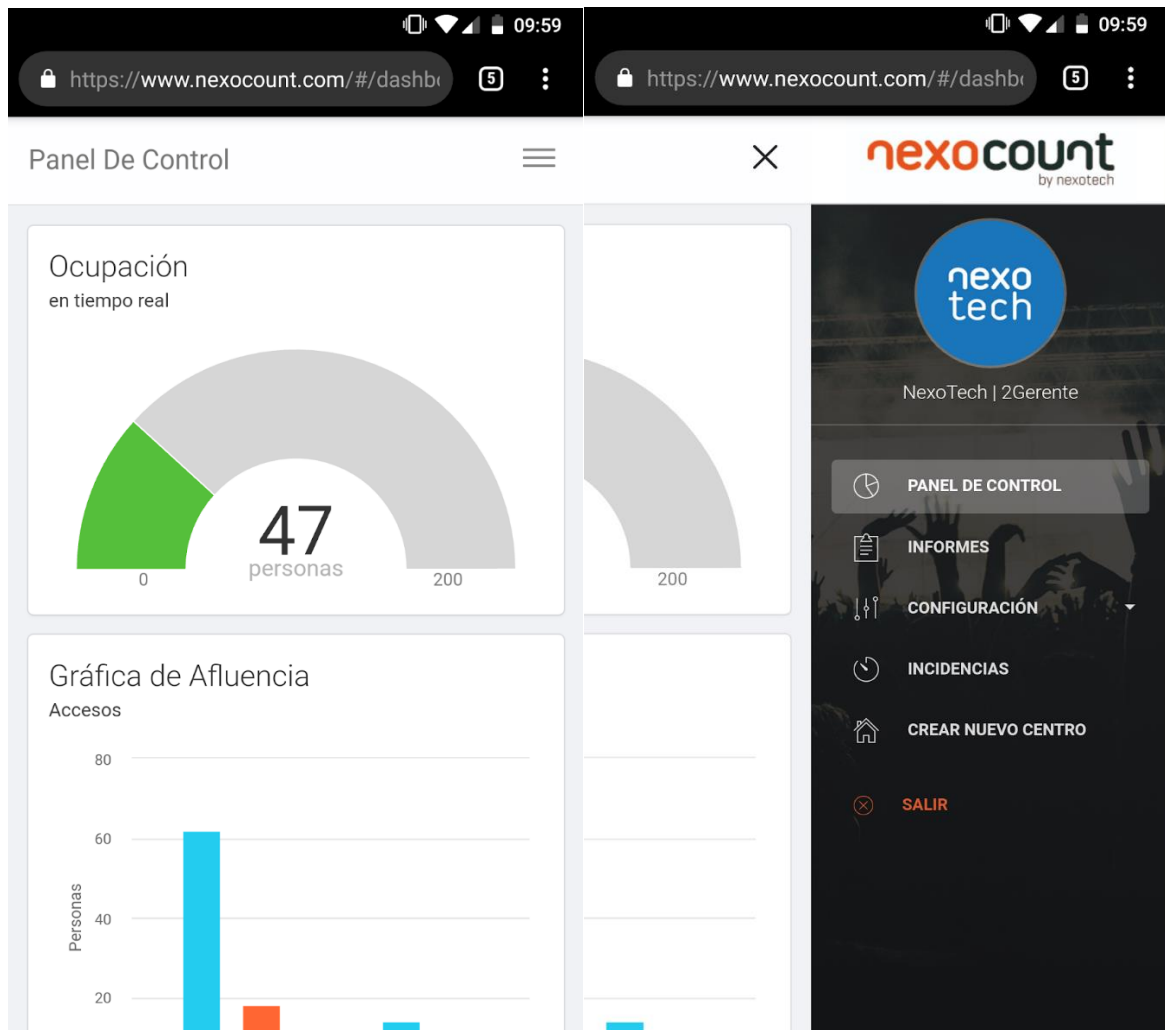


Figure 5.53-5.54: Interface of class Dashboard.js in a mobile-sized screen.

The tab drawer is moved to the right side and opened with a button, the logout button is added to said drawer for spacing reasons.

5.6. Deloyment

Web application deployment involves packaging and running your app on a server. One of the most crucial characteristics of the deployment of this app is that this server cannot be local, as it is not as reliable. Therefore a search for the best cloud solution was had.

At the beginning, a Windows server was used with 2-core CPU and 4GB of RAM with a hosting company. It became clear quickly that this was not enough, as the application run

slowly, the reports took from thirty to forty seconds to generate. As well as having reliability problems where the backend program would randomly shut down with no clear reason.

As a solution, the server was upgraded to a 4-core, 8GB of RAM server, but although it was a bit faster, it was not enough for the report generation to improve, and the reliability issues persisted.

Finally the app was changed from a Windows server to a Linux server, with the same hardware power, and also the hosting provider was changed. The result was drastic, the report generation was down to around eight seconds, which was later improved to less than a second with the query improvements mentioned on the backend section of the development chapter.

The reliability issues were gone as well, no random shut downs or any other problem since the server was changed, and it is been running for two months until this document creation date.

The backend runs on a port of the server while the frontend runs on the DNS itself. The advantage of the Windows server was that the deployment was easier to do, as the Linux server has to be configured through the command line, but with learning time that became a non-issue. Also it runs *Plesk*, which a server control panel that makes the job way easier.

The app is cloned with *Git* on the server, but to add a new feature it need to be shut down for a bit as it pulls the new code pushed. In the future there is the goal of doing continuous integration with *Git* on *Plesk*.

5.7. Gender and age recognition

A gender and age recognizer program is implemented in this project. From a video stream it has the task to recognize the people and predict a gender and age for each person recognized.

The program is a Keras implementation of a CNN for estimating age and gender. Let's see a step by step of the implementation.

5.7.1. Create the training data

To train the neural network, we need a training data that fits the requirements, for that we use the IMDB-WIKI dataset. It takes images of people from both wikipedia.com and imdb.com, it is the largest publicly available dataset of face images with gender and age labels for training.

Secondly, we filter out noise data and serialize images and labels for training into *.mat* file. The file *create_db.py* does that job.

```
def main():
    args = get_args()
    output_path = args.output
    db = args.db
    img_size = args.img_size
    min_score = args.min_score

    root_path = "data/{}_crop/".format(db)
    mat_path = root_path + "{}.mat".format(db)
    full_path, dob, gender, photo_taken, face_score, second_face_score, age = get_meta(mat_path, db)

    out_genders = []
    out_ages = []
    out_imgs = []

    for i in tqdm(range(len(face_score))):
        if face_score[i] < min_score:
            continue

        if (~np.isnan(second_face_score[i])) and second_face_score[i] > 0.0:
            continue

        if ~(0 <= age[i] <= 100):
            continue

        if np.isnan(gender[i]):
            continue

        out_genders.append(int(gender[i]))
        out_ages.append(age[i])
        img = cv2.imread(root_path + str(full_path[i][0]))
        out_imgs.append(cv2.resize(img, (img_size, img_size)))

    output = {"image": np.array(out_imgs), "gender": np.array(out_genders), "age": np.array(out_ages),
             "db": db, "img_size": img_size, "min_score": min_score}
    scipy.io.savemat(output_path, output)
```

Figure 5.55: Code of method *main()* class *create_db.py*.

The file `check_dataset.ipynb` shows that process.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import cv2
from utils import get_meta

db = "wiki"
# db = "imdb"
mat_path = "data/{}_crop/{}.mat".format(db, db)
full_path, dob, gender, photo_taken, face_score, second_face_score, age\
    = get_meta(mat_path, db)

print("#images: {}".format(len(face_score)))
print("#images with inf scores: {}".format(np.isinf(face_score).sum()))

#images: 62328
#images with inf scores: 18016

hist = plt.hist(face_score[face_score>0], bins=np.arange(0, 8, 0.2), color='b')
plt.xlabel("face score")
```

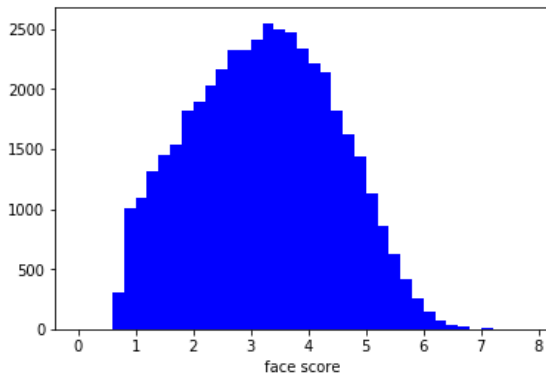


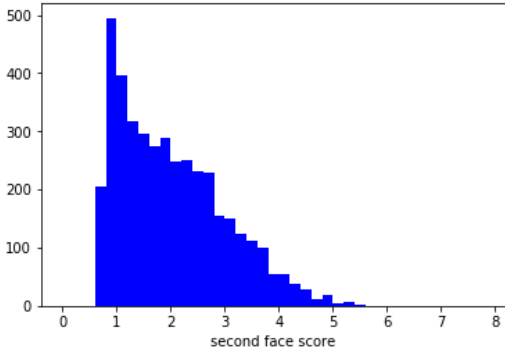
Figure 5.56: Code from `check_dataset.ipynb` that shows the face scores from the database as a graph.

First, we show the total number of images. Then, we show in a graph the “face score” which is given by the dataset.

- **face_score:** detector score (the higher the better). *Inf* implies that no face was found in the image and the `face_location` then just returns the entire image


```
print("#images with second face scores: {}".format(~np.isnan(second_face_score).sum()))
#images with second face scores: 4096

hist = plt.hist(second_face_score[~np.isnan(second_face_score)], bins=np.arange(0, 8, 0.2), color='b')
plt.xlabel("second face score")
<matplotlib.text.Text at 0x124bba748>
```



```
cols, rows = 4, 3
img_num = cols * rows
path_root = "data/{}_crop/".format(db)

def show_imgs(img_paths):
    img_ids = np.random.choice(len(img_paths), img_num, replace=False)

    for i, img_id in enumerate(img_ids):
        plt.subplot(rows, cols, i + 1)
        img = cv2.imread(path_root + str(img_paths[img_id]))
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.axis('off')

    plt.show()

img_paths = []

for i in range(len(face_score)):
    if face_score[i] < 0.0:
        img_paths.append(full_path[i][0])

print("#images with scores lower than 0.0: {}".format(len(img_paths)))
#images with scores lower than 0.0: 18016
```

Figure 5.57: Code from *check_dataset.ipynb* that shows the second face scores from the database as a graph.

The dataset also gives us a “second face score”:

- **second_face_score:** detector score of the face with the second highest score. This is useful to ignore images with more than one face. *second_face_score* is *NaN* if no second face was detected.

We show the plot of the “second face score” of all of the images as well. Then, we check the images with the extreme scores to not use them in the training.

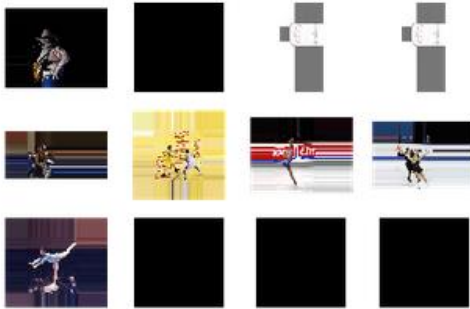
```
img_paths = []

for i in range(len(face_score)):
    if face_score[i] < 0.0:
        img_paths.append(full_path[i][0])

print("#images with scores lower than 0.0: {}".format(len(img_paths)))
```

```
#images with scores lower than 0.0: 18016
```

```
show_imgs(img_paths)
```



```
img_paths = []

for i in range(len(face_score)):
    if 0.0 < face_score[i] < 1.0:
        img_paths.append(full_path[i][0])

print("#images with scores lower than 1.0: {}".format(len(img_paths)))
```

```
#images with scores lower than 1.0: 1319
```

```
show_imgs(img_paths)
```

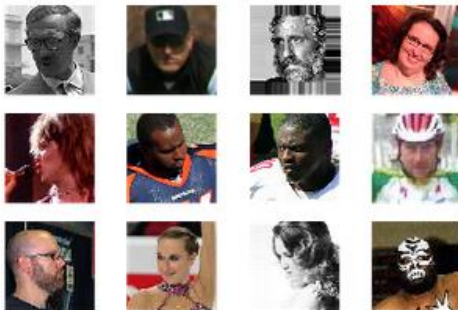


Figura 5.58: Code from *check_dataset.ipynb* that shows the images with extreme face score and normal face score.

For example, first images shown in the figure 5.58 have a score lower than zero so they are not useful images to train as they do not have seeable faces. The second images shown are correct images and we do not need to filter them.

The fact that the dataset provides us with the ability to filter out images that do not fit what we want to train is very useful and provides better results.

We also use another dataset to have a bigger and more diverse set of images. This is the UTKFace dataset. This needs another database creator `create_db_utkface.py` that takes the data and fits it in our database format to use it at the same time as the IMDB-WIKI database.

There is not a need to dive into the explanation of that file as it just changes the code to fit the input format from the UTKFace dataset.

Now that we have the dataset created with all of the images classified, we can proceed to train the neural network.

5.7.2. Training the neural network

Training the neural network is the most crucial and complicated step in order to create the gender and age recognizer. It is important to use the correct variables like the batch size (number of training examples used in one iteration), number of epochs (number of chunks of training examples in one iteration) or the learning rate.

```

logging.debug("Running training...")

data_num = len(X_data)
indexes = np.arange(data_num)
np.random.shuffle(indexes)
X_data = X_data[indexes]
y_data_g = y_data_g[indexes]
y_data_a = y_data_a[indexes]
train_num = int(data_num * (1 - validation_split))
X_train = X_data[:train_num]
X_test = X_data[train_num:]
y_train_g = y_data_g[:train_num]
y_test_g = y_data_g[train_num:]
y_train_a = y_data_a[:train_num]
y_test_a = y_data_a[train_num:]

if use_augmentation:
    datagen = ImageDataGenerator(
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        preprocessing_function=get_random_eraser(v_l=0, v_h=255))
    training_generator = MixupGenerator(X_train, [y_train_g, y_train_a], batch_size=batch_size, alpha=0.2,
                                       datagen=datagen)()
    hist = model.fit_generator(generator=training_generator,
                              steps_per_epoch=train_num // batch_size,
                              validation_data=(X_test, [y_test_g, y_test_a]),
                              epochs=nb_epochs, verbose=1,
                              callbacks=callbacks)
else:
    hist = model.fit(X_train, [y_train_g, y_train_a], batch_size=batch_size, epochs=nb_epochs, callbacks=callbacks,
                    validation_data=(X_test, [y_test_g, y_test_a]))

logging.debug("Saving history...")
pd.DataFrame(hist.history).to_hdf(output_path.joinpath("history_{}_{}.h5".format(depth, k)), "history")

```

Figura 5.59: Code from *train.py*.

We use a Wide Residual Network as the neural network to predict gender and age. A keras implementation of that kind of network from the github user “asmith26” [32]. The variables of the neural network are sixteen for the depth and eight for the width of the hidden layers.

When the network is trained we then save the resulting weights on an HDF5 file.

On figure 5.60 you can see the results of the training, its loss and accuracy.

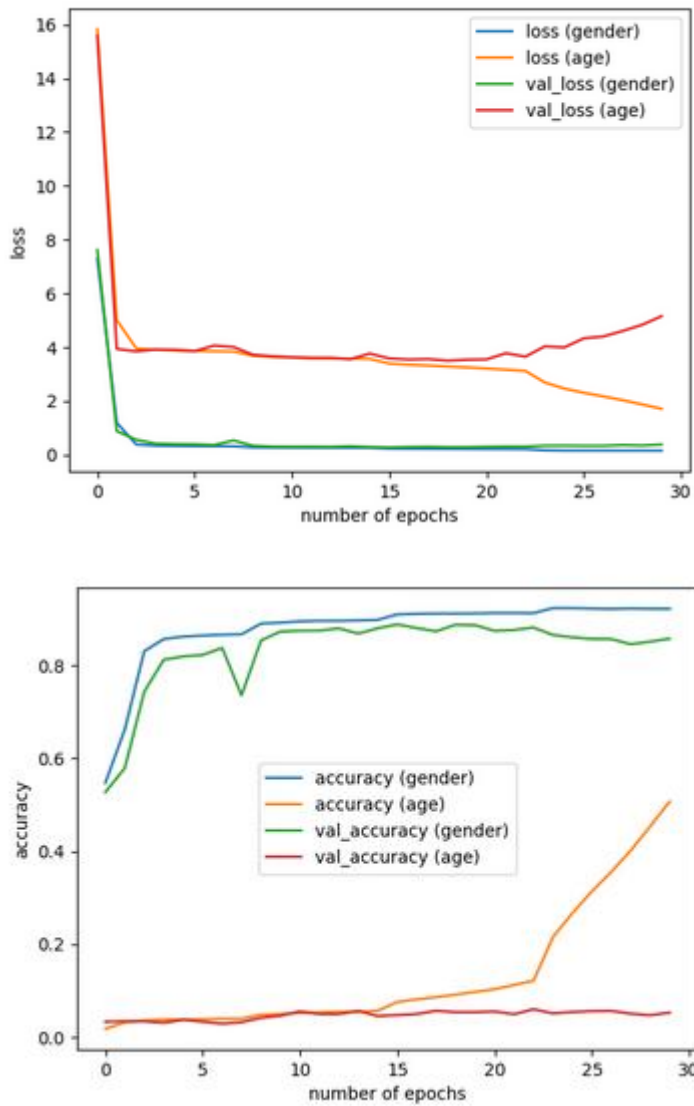


Figure 5.60: Code from *check_dataset.ipynb* that shows the face scores from the database as a graph.

We can see that the accuracy of gender recognition is higher than age recognition as it only has two different output while the age recognition has a 100 range output variance. It goes over 90% accuracy on gender recognition.

5.7.2. Using the trained network

It is time to use the trained network with test data. But first we need to get the faces from the test images, for that we use the Python library *dlib*, which has face detectors.

This library offers two face detecting solutions:

1. *get_frontal_face_detector()*: This face detector is made using the now classic Histogram of Oriented Gradients (HOG)[33] feature combined with a linear classifier, an image pyramid, and sliding window detection scheme. This type of object detector is fairly general and capable of detecting many types of semi-rigid objects in addition to human faces.

It is a fast solution (0.3 seconds on a Windows computer with i52-core Intel processor/8GB of RAM/Intel UHD Graphics 620) but does not work as well on poor lighting conditions and not frontal face positions.

2. *cnn_face_detector()*: This solution loads a pre-trained model and uses it to find faces in images. The CNN model is much more accurate than the HOG based model shown in the *face_detector.py* example, but takes much more computational power to run, and is meant to be executed on a GPU to attain reasonable speed.

Both solutions are tested but ultimately, on testing conditions with poor computer power it is recommended to use the first solution, if the client wants better results, an increase on the price will be made to cover the costs of the computational power.

The *demo.py* file testes the trained neural network, it uses the webcam of the computer running the code.

```
def main():
    args = get_args()
    depth = args.depth
    k = args.width
    weight_file = args.weight_file
    margin = args.margin
    image_dir = args.image_dir

    if not weight_file:
        weight_file = get_file("weights.28-3.73.hdf5", pretrained_model, cache_subdir="pretrained_models",
                               file_hash=modhash, cache_dir=str(Path(__file__).resolve().parent))

    # for face detection
    detector = dlib.get_frontal_face_detector()

    # load model and weights
    img_size = 64
    model = WideResNet(img_size, depth=depth, k=k)()
    model.load_weights(weight_file)

    image_generator = yield_images_from_dir(image_dir) if image_dir else yield_images()
```

Figura 5.61: Code from method *main()* of class *demo.py*.

First we initialize the variables needed for the testing, the weight file that we created as the trained neural network, the face detector, and then we create the neural network model with the *WideResNet* class, and load the pretrained weights that we imported. Lastly we get the images from the frames of the video stream that comes from the webcam.

```

for img in image_generator:
    input_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_h, img_w, _ = np.shape(input_img)

    # detect faces using dlib detector
    detected = detector(input_img, 1)
    faces = np.empty((len(detected), img_size, img_size, 3))

    if len(detected) > 0:
        for i, d in enumerate(detected):
            x1, y1, x2, y2, w, h = d.left(), d.top(), d.right() + 1, d.bottom() + 1, d.width(), d.height()
            xw1 = max(int(x1 - margin * w), 0)
            yw1 = max(int(y1 - margin * h), 0)
            xw2 = min(int(x2 + margin * w), img_w - 1)
            yw2 = min(int(y2 + margin * h), img_h - 1)
            cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 2)
            # cv2.rectangle(img, (xw1, yw1), (xw2, yw2), (255, 0, 0), 2)
            faces[i, :, :, :] = cv2.resize(img[yw1:yw2 + 1, xw1:xw2 + 1, :], (img_size, img_size))

        # predict ages and genders of the detected faces
        results = model.predict(faces)
        predicted_genders = results[0]
        ages = np.arange(0, 101).reshape(101, 1)
        predicted_ages = results[1].dot(ages).flatten()

        # draw results
        for i, d in enumerate(detected):
            label = "{}, {}".format(int(predicted_ages[i]),
                                    "M" if predicted_genders[i][0] < 0.5 else "F")
            draw_label(img, (d.left(), d.top()), label)

    cv2.imshow("result", img)
    key = cv2.waitKey(-1) if image_dir else cv2.waitKey(30)

    if key == 27: # ESC
        break

```

Figura 5.62 : Code from method *main()* of class *demo.py*.

For each frame of the videostream, we first detect the faces on it with the *dlib* face detector, we also draw a rectangle over each face, then with these detected faces we make a prediction with the model.

This prediction returns both the predicted gender (F: female, M: male) and ages (between one and 101), we then draw on top of the rectangle this information.

On figure 5.63 we can see an example on a real-life entrance with blurred out faces for privacy matters.

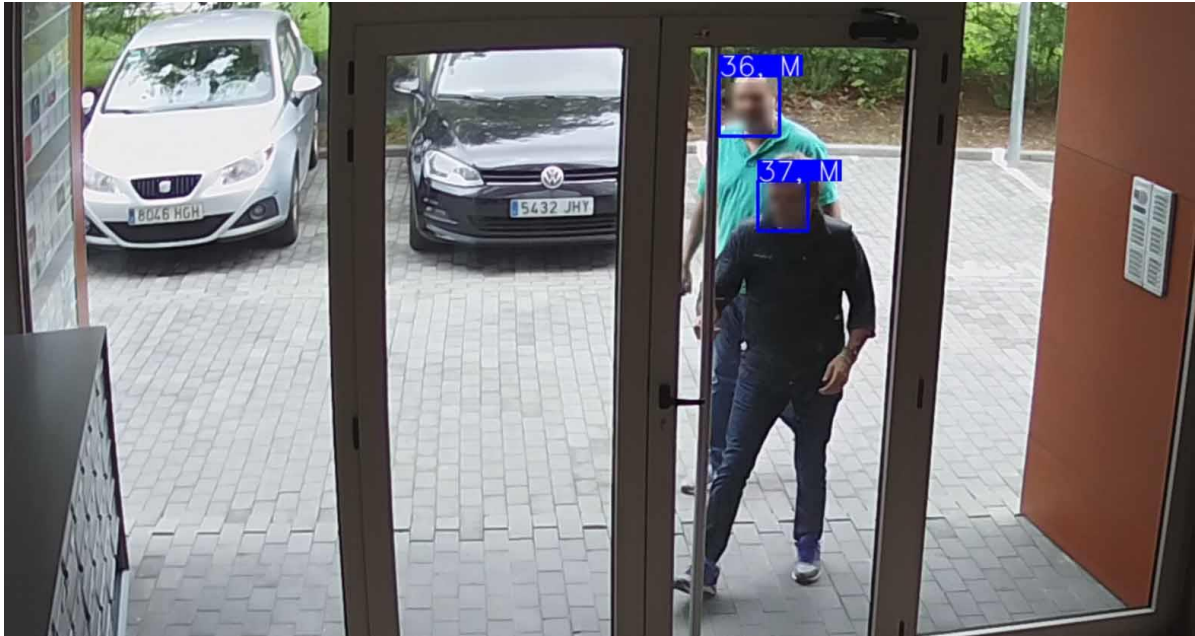


Figure 5.63: Video Frame of a real-life entrance with age and gender recognition. [age, gender]

As you can see, it works quite well even if the faces are small, but it is not enough to create a people counter with it, as the face recognition is not reliable enough, so for the people counting we still use the sensors, and the gender and age recognition will be used for business analytics, with the percentage of male and female that appear on the video stream.

6. Conclusions

This project is, in any measure possible, a success. A product was designed and built which offers its clients a full-on security program for the influx of people in the client's enclosures.

A reliable full-stack web application was built, with particular care on offering excellent user experience.

It is not only a hypothetical project, as it has a real-life purpose and it is used by clients of significant magnitude. On these cases, the product was tested giving incredible results, up to 98% accuracy.

Without much prior knowledge of how to build a full-stack web application, researching and studying the different solutions was crucial, and it shows on the final product.

The code was improved through good software pattern implementations which also increased the app fastness.

The age and gender recognition program was the most difficult part, as learning how machine learning works from the ground-up, finding the perfect dataset that was big enough to have good accuracy, and training a neural network with it to then predict results was definitely challenging. But at the end a solution that worked with good accuracy, particularly the gender recognition, was made.

All the goals set were completed unless the showing of the amount of occupation of each gender with the information obtained in the previously discussed program.

On the future, that goal plus much more will be added. Such as improving the database to fit "big data", adding different languages to the *webApp*, implementing continuous integration or ability to compare reports.

The code was written for it to be scaled, so the application will keep adding features and being improved upon.

7. Bibliography

[1] (2019, March 28). Top 8 Best Backend Frameworks - KeyCDN. Retrieved June 4, 2019, from <https://www.keycdn.com/blog/best-backend-frameworks>.

[2] (2018, February 20). 7 types of retail people counter systems you should be aware of | Ipsos Retrieved June 4, 2019, from <https://www.ipsos-retailperformance.com/resources/blog/people-counter-systems-aware/>

[3] (n.d.). What Is Deep Learning? | How It Works Retrieved June 4, 2019, from <https://www.mathworks.com/discovery/deep-learning.html>

[4] (2016) Ian Goodfellow, Yoshua Bengio & Aaron Courville: *Deep Learning*. MIT Press.

ISBN 10: 0262035618 / ISBN 13: 9780262035613

[5] (2017, February 23). Deep Learning 101 - Part 1: History and Background - Andrew L. Beam. Retrieved June 4, 2019, from https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

[6] (2017) Adam Gibson, Josh Patterson: *Deep Learning*. O'Reilly Media, Inc.

ISBN: 9781491924570

[7] Pandey, Ratnakar: *Deep Learning using Tensor Flow, Keras, Python, Jupyter Notebook* [online] Retrieved January 30, 2019, from https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

[8] Valkov, Venelin: *Credit Card Fraud Detection using Autoencoders in Keras—TensorFlow for Hackers (Part VII)* [online] Retrieved January 30, 2019, from <https://medium.com/@curiously/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd>

[9] Brownlee, Jason: *Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras* [online] Retrieved January 30, 2019, from <https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/>

[10] Brownlee, Jason: *Multivariate Time Series Forecasting with LSTMs in Keras* [online] Retrieved January 30, 2019, from

<https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/>

[11] Seif, George: *Deep Learning for Image Recognition: why it's challenging, where we've been, and what's next* [online] Retrieved January 30, 2019, from <https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef>

[12] Alex Krizhevsky; Ilya Sutskever; Geoffrey E. Hinton (2012): *ImageNet Classification with Deep Convolutional Neural Networks*. University of Toronto.

[13] Karen Simonyan; Andrew Zisserman (2014): *Very Deep Convolutional Networks for Large-scale Image Recognition*. Visual Geometry Group, Department of Engineering Science, University of Oxford.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich (2015): *Going Deeper With Convolutions*. Google Inc.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015): *Going Deeper With Convolutions*. Microsoft Research.

[16] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger (2018): *Densely Connected Convolutional Networks*. Cornell University, Tsinghua University, Facebook AI Research.

[17] Sergey Zagoruyko, Nikos Komodakis (2017). *Wide Residual Network*. Université Paris-Est, École des Ponts, ParisTech, Paris, France.

[18] Büyükyılmaz, Mücahit & Çıbıkdiken, Ali. (2016). *Voice Gender Recognition Using Deep Learning*.

- [19] Amit Dhomne, Ranjit Kumar, Vijay Bhan (2018). *Gender Recognition Through Face Using Deep Learning*. Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Odisha, India.
- [20] Wu, Ming & Yuan, Yubo (2014). Gender Classification Based on Geometry Features of Palm Image. *The Scientific World Journal*, 2014, 1-7.
- [21] Dey, Sangeeta & Kapoor, A. (2015). Sex determination from hand dimensions for forensic identification. *International Journal of Research in Medical Sciences*, , 1466-1472.
- [22] Afifi, Mahmoud (2017). 11K Hands: Gender recognition and biometric identification using a large dataset of hand images.
- [23] Rasmus Rothe, Radu Timofte, Luc Van Gool: *IMDB-WIKI – 500k+ face images with age and gender labels* [online] Retrieved January 31, 2019, from <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>
- [24] Zhang, Zhifei, Song, Yang, and Qi, Hairong: *Age Progression/Regression by Conditional Adversarial Autoencoder* [online] Retrieved May 21, 2019, from <https://susanqq.github.io/UTKFace/>
- [25] Nguyen, Andrew: *Machine Learning by Stanford* [online] Retrieved January 31, 2019, from <https://www.coursera.org/learn/machine-learning>
- [26] (n.d.). Normativa - Control de Aforo – Contador Homologado de Personas. Retrieved June 5, 2019, from <http://www.controldeaforo.org/legislacion.html>
- [27] (n.d.). What is Scrum? - Scrum.org. Retrieved June 5, 2019, from <https://www.scrum.org/resources/what-is-scrum>
- [28] (2011, January 20). Example of a strong and weak entity types - Stack Overflow. Retrieved June 5, 2019, from <https://stackoverflow.com/questions/4741967/example-of-a-strong-and-weak-entity-types>
- [29] n.d.). Power over Ethernet (POE) Explained - Understanding and using POE. Retrieved June 5, 2019, from <http://www.veracityglobal.com/resources/articles-and-white-papers/poe-explained-part-1.aspx>

[30] (n.d.). Introduction to the Java Persistence API - The Java EE 6 Tutorial. Retrieved June 5, 2019, from <https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>

[31] Creative Tim: *Premium Bootstrap Themes and Templates* [online] Retrieved May 21, 2019, from <https://www.creative-tim.com/>

[32] (n.d.). asmith26/wide_resnets_keras: Keras implementation + ... - GitHub. Retrieved June 5, 2019, from https://github.com/asmith26/wide_resnets_keras

[33] (n.d.). Histograms of Oriented Gradients for Human Detection. Retrieved June 5, 2019, from <https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>

Grau en Enginyeria Informàtica de Gestió i Sistemes d'Informació

BUSINESS ANALYTICS PLATFORM OF INFLUX CONTROL OF PEOPLE IN ENCLOSURES

Estudi de la viabilitat

Jan Jiménez Serra
TUTOR: Xavier Font

4rt

Table of Contents

| | |
|---|-----|
| 1. Initial planning | 113 |
| 1.1. List of tasks..... | 113 |
| 1.2. Calendar..... | 118 |
| 2. Budget | 120 |
| 2.1. Resources..... | 121 |
| 3. Viability analysis..... | 122 |
| 3.1. Analysis of technical viability | 122 |
| 3.2. Analysis of economic viability | 122 |
| 3.3. Environmental viability analysis | 123 |
| 3.4. Legal aspects..... | 124 |

1. Initial planning

1.1. List of tasks

The list of tasks contains the description of each task, an ID to facilitate the definition of the dependencies and the time that is predicted to last in a working day, the working days will last 4 hours.

- **Technological Platform and Development Environment**

Everything you need to start the development of the application. From the installation of the workstations (the one of the desktop computer as the personal laptop), installation of the server, in this case a Windows 2012 server, given by the company. Creating the repository in BitBucket and Trello for tracking tasks, and frameworks and technologies already defined as Git for version control. In addition to the IDEs necessary for web development such as Visual Code. Last but not part of the development, a test sensor is installed.

| ID | Task | Dependencies | Time (days) |
|----|---|--------------|-------------|
| A1 | Installation of the work station | | 0 |
| A2 | Installing the server | A1 | 1 |
| A3 | Installation of storage units | A1 | 1 |
| A4 | Installation of Operating Systems | A1 | 1 |
| A5 | Creation of VCS repository | A1,A2,A3 | 1 |
| A6 | Installing IDEs | A1 | 0.5 |
| A7 | Installation of Frameworks of Development | A1 | 0.5 |
| A8 | Testeig sensor installation | | - |
| A | Total | | 5 |

- **Database**

Definition and construction of the application database. In the choice of programming language, the server that is used, in addition to the type of database, is taken into account. Also decisions to organize the tables and the model, the application is influenced by the decisions taken in this section.

| ID | Task | Dependencies | Time (days) |
|----|--------------------------|--------------|-------------|
| B1 | Conceptual Model | A | 5 |
| B2 | Logical-Relational Model | A,B1 | 2 |
| B3 | Physical Model | A,B2 | 1 |
| B4 | Creation of BBDD | A,B3 | 2 |
| B | Total | | 10 |

- **Basic structure of the application**

Choosing the language and frameworks that will be used both in the back-end and in the front-end, in addition to defining the back-end API so they can communicate. User roles architecture and authentication system to access the application is also done in this section.

| ID | Task | Dependencies | Time (days) |
|----|---------------------------------------|--------------|-------------|
| C1 | Back-end architecture | B | 7 |
| C2 | Definition of the API | B | 1 |
| C3 | Front-end architecture | B | 9 |
| C4 | Architecture of roles and permissions | B | 1 |
| C5 | Authentication (Login) | B | 1 |
| C6 | Route architecture | B | 1 |
| C | Total | | 20 |

- **Main dashboard**

A center and user are inserted to test, and the output and input data of the center sensors inserted into the database are sent. The data is taken from the back end, which is sent through the API to the front-end, where they are shown with graphics and tables.

| ID | Task | Dependencies | Time (days) |
|----|--|--------------|-------------|
| D1 | Connection of the sensor with the database | C | 2 |
| D2 | CRUD system to backend of sensor data | C | 2 |
| D3 | Show data with capacity chart | C | 1 |
| D | Total | | 5 |

- **Reports**

System development that meets the report requirements.

| ID | Task | Dependencies | Time (days) |
|----|--|--------------|-------------|
| E1 | Chart of entries and exits | D | 2 |
| E2 | Table of entries and exits | D | 2 |
| E3 | Data filtering system by dates | D | 4 |
| E4 | Data filtering system for hours | D | 4 |
| E5 | Data filtering system for days of the week | D | 4 |
| E6 | Data filtering system for total, <i>zone</i> and <i>access</i> | D | 4 |
| E | Total | | 20 |

- **General configuration**

Develop the interface and logic of the configuration of the center and users. The user must have the possibility of changing the name of the center, the type of schedule, being able to

restore capacity data and, name and maximum capacity of the center and areas. Also be able to create and edit users.

| ID | Task | Dependencies | Time (days) |
|----|---------------------------|--------------|-------------|
| F1 | Center data setup system | E | 5 |
| F2 | Add user system | E | 5 |
| F3 | User editing system | E | 1 |
| F4 | System to eliminate users | E | 1 |
| F5 | Data restoration system | E | 3 |
| F | Total | | 15 |

- **Alarms and Incidents**

Develop the configuration of occupation alarms. List incidences with alarm breaks, sensor disconnection and manual data restorations.

| ID | Task | Dependencies | Time (days) |
|----|--|--------------|-------------|
| G2 | System to add alarms | F | 3 |
| G3 | System to edit alarms | F | 1 |
| G4 | System to eliminate alarms | F | 1 |
| G5 | List of incidents | F | 9 |
| G6 | System to send emails in case of an incident | F | 1 |
| G | Total | | 15 |

- **Extraction of information from images**

Part of the project that involves deep learning. It is sought to develop a system of counting people of each sex and, if possible, divide the capacity in ages. To test the system a normal and thermal camera is needed.

| ID | Task | Dependencies | Time (days) |
|----|---|--------------|-------------|
| H2 | Installing cameras | G | 1 |
| H3 | Identification of people system | G | 30 |
| H4 | Gender recognition system | G | 40 |
| H5 | Age recognition system | G | 10 |
| H6 | Show data on the amount of each gender and age range on the front end | G | 4 |
| H | Total | | 85 |

- **Documentation and Testing**

Tasks that are done throughout the project or at the end of each sprint. It is the documentation of the project, unitary and functional tests of all the code and systems of the web application. In addition to all the necessary research for the project.

| ID | Task | Dependencies | Time (days) |
|----|---------------------------|--------------|-------------|
| I1 | Preliminary documentation | A | - |
| I2 | Memory documentation | A,I1 | - |
| I3 | Final documentation | A-H,I1,I2 | - |
| I4 | Unit backend testing | C-H* | - |

| | | | |
|----|---|------|---|
| I4 | Functional testing at the end of each implementation of a new requirement | C-H* | - |
|----|---|------|---|

1.2. Calendar

In order to prepare the planned calendar, possible critical and problematic paths that may arise during the development are taken into account. For example deprived technologies, or problems with the connection of cameras or equipment.

Each sprint consists of the development tasks of the application, that is, the C to the H. The rest is considered to be done during each sprints or at the end of each one.

Below you can see a calendar in the form of a Gantt diagram, figure 7.1. You can find the full Gantt diagram grouped in days in the annex.



Figure 7.1: Grantt Diagram of the calendar of the project

2. Budget

The costs derived from the closure of agreements and confidentiality contracts associated with the use of technologies protected by the registry and patents office owned by the various hardware and software manufacturers involved are calculated. The details of these costs involves the payment of fees to these manufacturers to obtain the proper certification that allows the company to use the various internal technical resources of each manufacturer as well as obtaining the corresponding certification labels to be able to complete the tasks required in the project and the subsequent commercial exploitation of the resulting products.

With regard to the costs derived from the necessary human resources, it is essential to evaluate the technical training and skills of the personnel involved in the development of the project in order to precisely calculate an overall calculation of the hours necessary to carry out the project. The calculated total hours are segmented according to the Gantt planning of the project, which reflects the percentage implication of each human resource on the total of the project and ready for commercialization at which time the product is prepared to start its amortization process.

The estimation of the budget, therefore, is the following:

1. Practices agreement: 2880 euros, 600 hours:
 1. Cost according to the number of hours planned for the development of the project for the End of Degree Project TFG: € 3,600.00
2. Camera for access control:
 1. BOSCH Flexidome 7000iC 12Mpxl Panoramic Camera, includes installation and feeding kit: € 1,884.10
 2. Fixed-type bullet camera Dinion IP Ultra 8000 of 12Mpxls / 4K of BOSCH, Includes installation and feeding kit: € 1,950.00
 3. Ultra megapixel P-iris varifocal lenses with correction for IR: € 650.00

3. Thermal / thermo-graphic camera:
 1. Thermographic graphic camera FLIR IP, optical 18mm, angle 25°, 320x240 pixels, interior, 30 Hz .: 11,153.70 €
4. SDKs BOSCH An SDK to integrate Bosch video products into the resulting solution. It consists of reusable software components based on COM and ActiveX: € 3,000.00
5. Personal and desktop equipment: € 2,000.00
6. Relevant courses on deep learning: € 200.00
7. Software resources: € 150.00

TOTAL AMOUNT: € 24,587.80

2.1. Resources

The resources required for the project are offered by the company sponsoring the project. Like cameras, thermal cameras, courses and non-free technologies.

3. Viability analysis

Initially, the project's promoters evaluate their investment and financing capacity of the development process, marketing the project, as well as all the necessary means to give continuity in the market with the resulting product.

Considering that we already have the financial, technical and human resources adequate for the development of the project, it is essential to assess the degree of demand of the resulting product. A key aspect will be to have a good knowledge of the needs of the target market in order to plan the best strategic penetration, one that is able to achieve the expected return on time set by management.

3.1. Analysis of technical viability

To carry out the development of the project, state-of-the-art technologies provided by some of the leading hardware and software manufacturers in image capture and processing technology as well as some of the main standards accepted by the global market applied by the principals technology manufacturers in video analysis systems, sensorization and data processing and processing.

The result of the tasks carried out in the framework of the development of the project can be audited by the technical departments of the manufacturers themselves suppliers of the mentioned technical means to verify that the use is the appropriate one as well as to contribute to reinforce the project contributing, permanently , the latest news.

The main technical challenge is with the development and application of gender and age recognition. Although there are functional and feasible solutions, implementing it to an external application is complicated.

3.2. Analysis of economic viability

In the first phase, the financial resources needed to carry out the project development come from two sources (both financed by the company): own resources (60%) and bank

financing (40%). Once the product is already available and has achieved a first phase of penetration in the market, both commercial actions and the continuity of product development will be supported through external investment funds in order to provide monetary resources and, at the time, new instruments to extend the geographic scope of its commercialization.

Currently the market is highly receptive to a solution of the characteristics of the product to be developed. There are a number of factors that draw an excellent scenario for the acceptance of the product in the market. They are the following:

- **Legal framework:** In the last 2 years, it has become a significant regulatory change of multisectoral application that obliges companies, especially those that receive public affairs or leisure activities, to have effective control and security systems. The trigger for this change was caused by the incidents of the Madrid Arena.
- **Market situation:** The target market has reached sufficient maturity to understand the advantages that the use of the technologies employed in the project can bring to improve the development of its business.
- **Competition:** The absence of a consolidated and compatible competition with the main existing sensorization technologies.

The target market has been segmented in order to prioritize and orient commercial actions towards those more receptive markets such as night entertainment, culture and sports and industrial control for the protection of people. Each and every one of them are, at present, areas of great demand for technological solutions aimed at better management of resources, thus increasing the safety of people.

3.3. Environmental viability analysis

All the hardware used has the maximum environmental certifications required, such as the NEMA TS 2-2003 (R2008) published by the National Electrical Manufacturers Association which regulates, in this case, the thresholds of the operating temperatures of the equipment with which it works the project The National Association of Electrical

Manufacturers (NEMA) represents about 325 manufacturers of electrical equipment and medical imaging that manufacture safe, reliable and efficient products and systems for seven main markets:

Construction systems:

- Construction infrastructure
- Illumination systems
- Industrial Products and Systems
- Products and utility systems
- Transportation systems
- Medical images

Combined industries, to which BEMA represents represent 360,000 US jobs in more than 7,000 facilities, cover each state. The industry it regulates produces \$ 106,000 million of shipments of electrical equipment and medical imaging technologies per year with \$ 36 billion in exports.

On the other hand, once the life cycle of the employee hardware has been exhausted, these are recycled in compliance with the legal responsibilities derived from their activity in order to reduce the environmental impact of the processes.

3.4. Legal aspects

The development of the project rigorously respects all the legal prescriptions necessary in order to comply with the regulations in force regarding the protection of industrial intellectual property. This aspect is legally formalized under a contract of confidentiality between the company that promotes the project and the suppliers / manufacturers of the technologies necessary to embody the maximum guarantees of success and sustainability of the project in question.

These contracts are valid until the development of the project lasts or until one of the parties decides the termination of the contract and obliges all those involved in the project to comply with the confidentiality clauses established in the framework of the contract.

In the case of non-compliance with the premises contemplated and accepted by the parties during the established deadlines, both the manufacturers and the company that promotes the project reserve the power to take the proportional measures that it deems appropriate.

Grau en Enginyeria Informàtica de Gestió i Sistemes d'Informació

BUSINESS ANALYTICS PLATFORM OF INFLUX CONTROL OF PEOPLE IN ENCLOSURES

Annexos

Jan Jiménez Serra
TUTOR: Xavier Font

4rt

Table of Contents

| | |
|--------------------------|-----|
| Annex I. CD content..... | 141 |
|--------------------------|-----|

Annex I. CD content

- Project documentation (memòria, viabilitat del projecte i annexos)
- API Documentation
- Code of NexoCount Web Application
- Code of sensor data retrieving program
- Code of Age and Gender Recognition program
- Database design
- Full Gantt diagram of the calendar
- User handbook of NexoCount

