Surveillance Camera



Salahdin Belghiti

28 de febrero de 2020

[VERSION 1.0]

Content

[1. Front-End 2](#_Toc33612202)

[1.1. Web pages 2](#_Toc33612203)

[1.2. Style sheets 2](#_Toc33612204)

[1.3. Libraries and programming languages 2](#_Toc33612205)

[1.4. Pages descriptions 2](#_Toc33612206)

[1.4.1. INDEX.html 2](#_Toc33612207)

[1.4.2. RESETPASSWORD.html 3](#_Toc33612208)

[1.4.3. MAINPAGE.html 3](#_Toc33612209)

[1.4.4. STATPAGE1 y 2.html 3](#_Toc33612210)

[1.5. Statistics programming 3](#_Toc33612211)

[2. BACK-END 4](#_Toc33612212)

[2.1. Login page (URL: ‘/’) 4](#_Toc33612213)

[2.2. Login page – Recover password (modal) 4](#_Toc33612214)

[2.3. Reset password page (URL: ‘/resetpassword?id=XX’) 5](#_Toc33612215)

[2.3.1. GET Reset password 5](#_Toc33612216)

[2.3.2. POST Reset password page - "/forgotpassword" 5](#_Toc33612217)

[2.4. Main page (URL: ‘/main’) 6](#_Toc33612218)

[2.5. Charts page (URL: ‘ /charts’, ‘ /charts2’) 6](#_Toc33612219)

[2.6. Modal views 6](#_Toc33612220)

[2.6.1. POST Change password (modal) – “/updatepassword” 6](#_Toc33612221)

[2.6.2. POST Change name (modal) – “/updatename” 6](#_Toc33612222)

[2.6.3. POST Change email (modal) – “/updateemail” 6](#_Toc33612223)

[2.7. Camera Connector and Camera Connector Controller 6](#_Toc33612224)

[2.8. Queries 7](#_Toc33612225)

[2.9. Cookies 7](#_Toc33612226)

[2.10. Messages 7](#_Toc33612227)

[3. Client – Python language 8](#_Toc33612228)

[4. Database 9](#_Toc33612229)

[4.1. Users table 9](#_Toc33612230)

[4.2. Records table 9](#_Toc33612231)

# Front-End

The color’s combinations of the webpage is result of multiple color palettes. The final purpose is to make a webpage with the colors of the product/service, accesible menus and a simple desing.

## Webpages

All web pages are responsive, that is, their appearance adapts to the device that visits it, such as mobiles, tablets, computers, etc. The project consists of the following pages:

* Index
* Mainpage
* Resetpassword
* Statpage1
* Statpage2

## Style sheets

Each style sheet is associated with your HTML page. Contains the formatting, colors, and layout that each item on the page should have.

* Main\_styles
* Stats\_styles
* Styles

## Libraries y Programming languages

The following libraries and languages are used to make the code simpler and easier to handle/maintain.

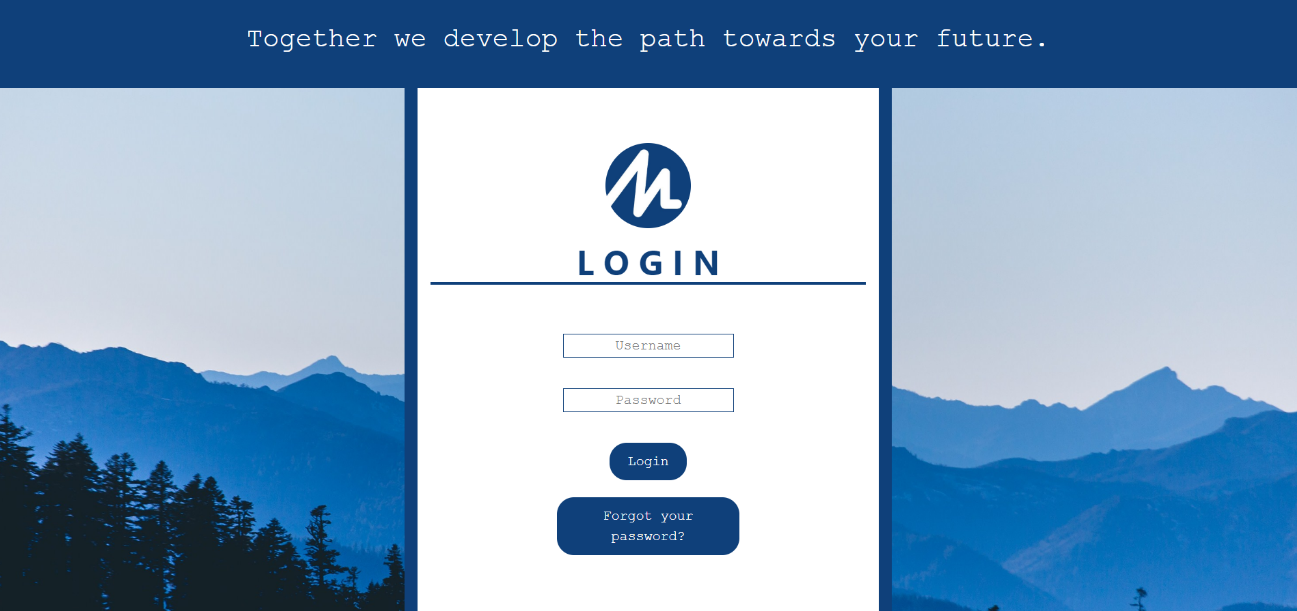
* CanvasJS
* Bootstrap
* JavaScript

## Webpages description

### **INDEX.html**

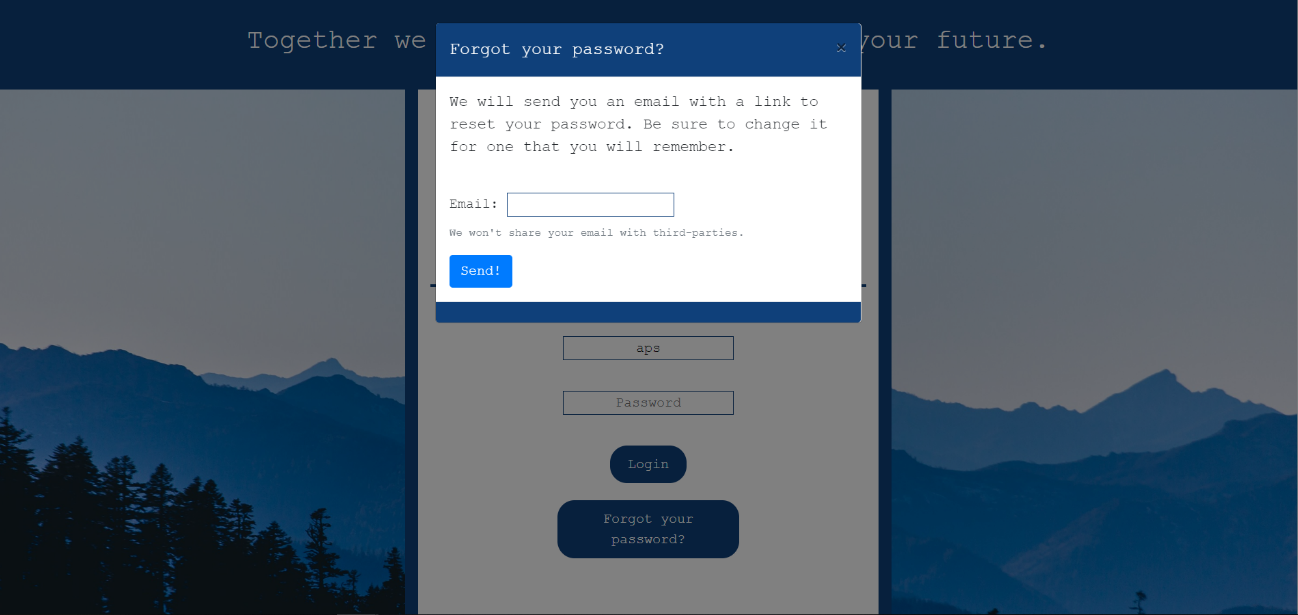
The login page, in which data is entered in the corresponding fields and after a pass through conditions (whether the field is not empty or without meeting requirements) and that obviously the fields match some previously created account, the user will be able to access the main page.

With Bootstrap (toast), errors or any useful information is displayed to in a small text (e.g., "wrong password" or "email sent").



### **RESETPASSWORD.html**

Same layout as the "index.html" page but this time we have two fields for two passwords since the only functionality of this page is to change the user's password (the link to this page is mailed from the home page).

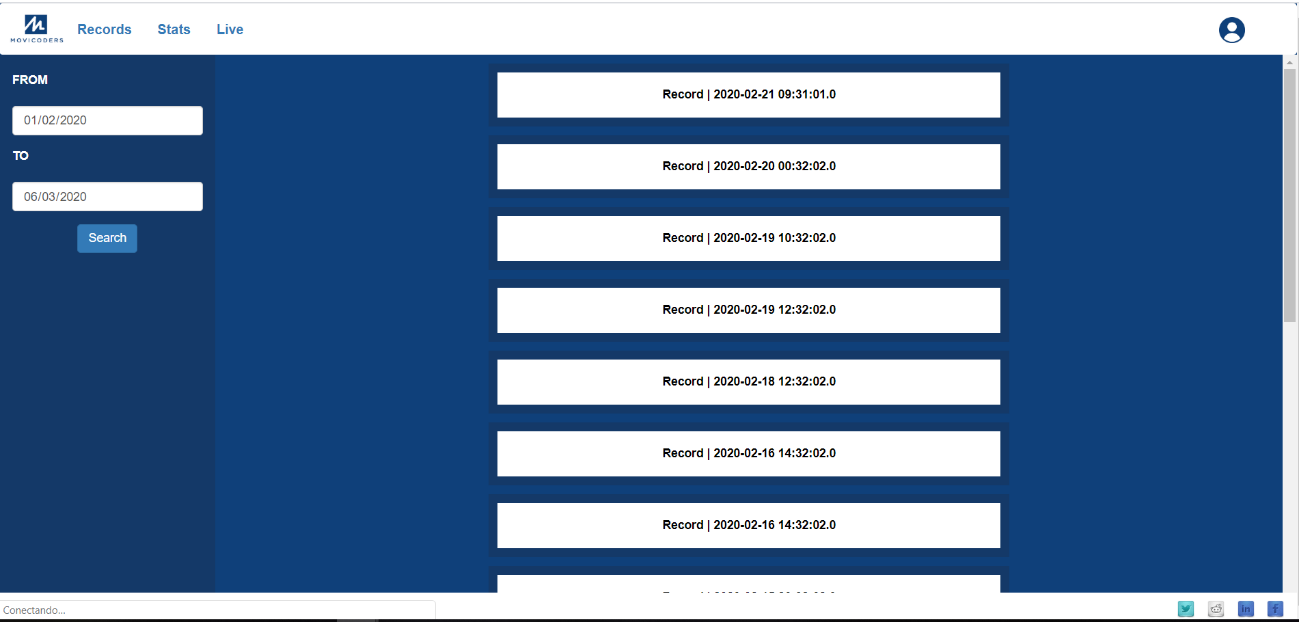


### **MAINPAGE.html**

Once we access the login page, we get to the main page, which by default will be the section of recordings. Starting with the navigation bar, there are 3 options depending on where the user wants to go ("recordings", "stats" and "live"). In "mainpage.html", we will have a filter and an accordion type list (this is why Bootstrap is used) of all recordings, for each recording, we will have access to all their information: date and time, with which camera has made it and a video of that particular recording. Using the filter, it is possible to view all recordings of the selected date or leave it by default, which would be the recordings of that week.

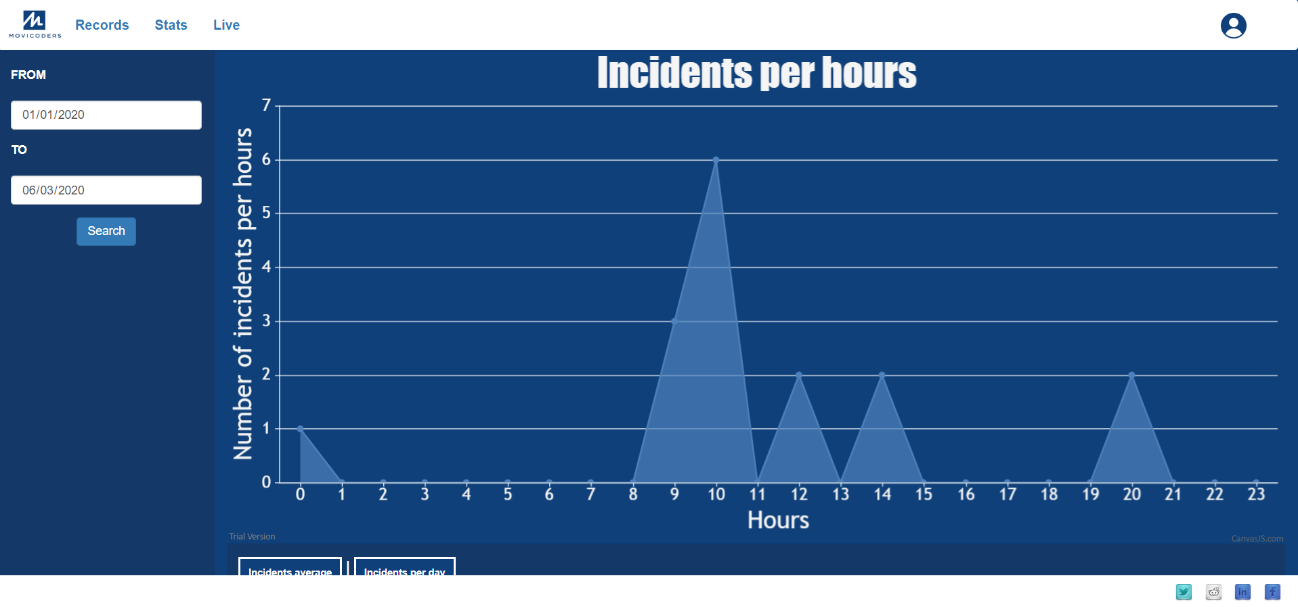
Finally, we have a button at the top right for independently change the user's name, password or mail quickly (a model type window of Bootstrap). There is also to log out. This button in turn when we switch to "mobile mode" will be removed, merging the previously spoken options with those of the navigation menu ("recordings", "statistics" and "direct") bringing everything together in one button that is more accessible for a mobile user.

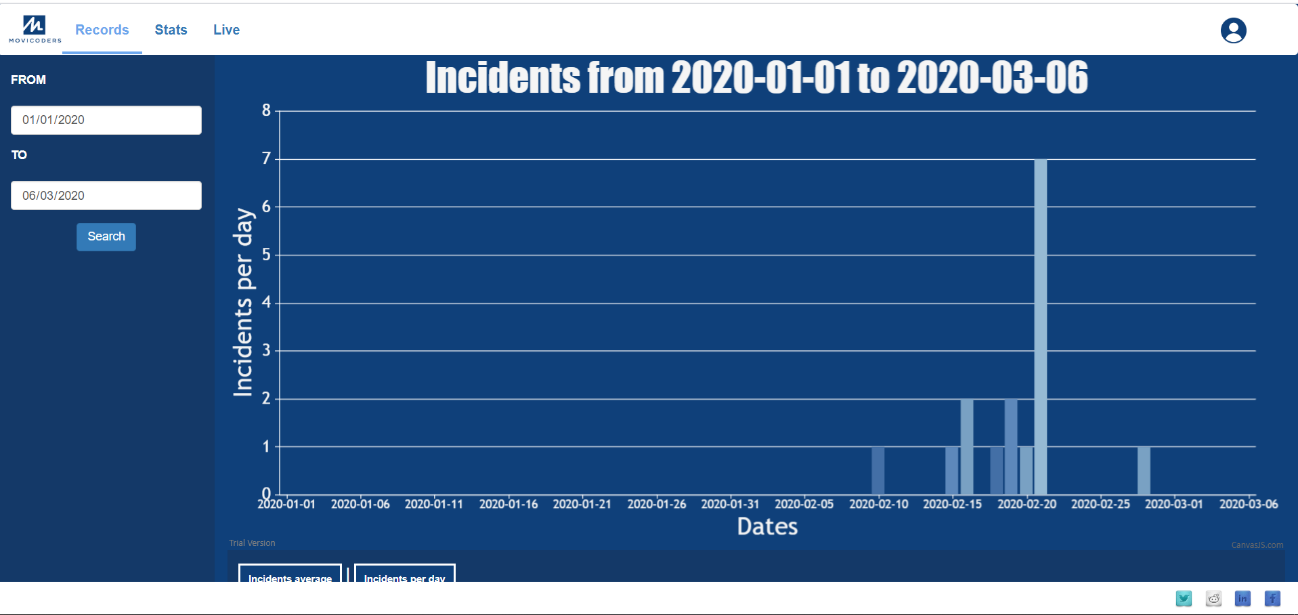
In the footer we can access the different social networks.



### **STATPAGE1 y 2.html**

Using the same filter, footer and navigation bar of mainpage.html, in the center of the page web have statistics tables with the information to display.





## Statistics webpages

Statistics are programmed with Javascript, and CanvasJS. There are two types of statistics, which show the number of incidents per day, filtered by start date, and end date, which the user selects, and the one that shows the hourly incidents, which indicates the most (and least) hours active, also filtered by date of start and end.

The day-by-day incident statistic is bar-type, and hour-type statistics are line-type. The javascript code is composed of the method that draws the graph, and the methods that treat the data, which are the following: - getArray(), this method is responsible for dividing the date and time, and uses one of the two data, depending on the type of graph. It is stored in an array, for later use. - getReps(), is responsible for counting the number of times the date/time is displayed in the array (and therefore the number of incidents per day/hour) and returns it in a collection of type Map. - format(), is responsible for adapting each data pair (key, value) to the CanvasJS format, consisting of the number of incidents (y) and the corresponding date (label), the format is as follows: { y: 12, label: "01/01/2020"} - formatMap() is used to apply the format() method in each entry of the map, with the forEach() method.

Finally there is the CanvasJS method, where you can choose the theme, animation, title, texts, etc.

# BACK-END

## Login page (URL: ‘/’)

The login page will control access to the application by checking the user (username + password) against the data stored in the database. This control is based on the following steps:

Once the user enters the requested data and selects the login button, the controller class (UserController) will check whether all fields are populated by an object of the BindingResult class and the Valid annotation. If this object has any errors (.hasErrors()), the page will be reloaded with the corresponding inwarnings (although this check is done with JS in the view, this double-check is performed in the backend). As soon as all fields are correct, the driver checks whether the (user + password) exists in the database. This verification is done through the DAO class (UserDaoImpl), responsible for sending the specific query for searching in the DDBB. In the event that the (user + password) exists, the controller creates an object of the Cookie class and adds the user's username, in addition to setting its storage time to 15 minutes and redirecting to the home page (URL: '/main/'). Otherwise, a 'USER\_ERROR' message attribute is sent to the view model for show it on the page.

## Login page – Recover password (modal)

In order to retrieve the password, the login page (URL: '/') opens a modal window that will ask the user for an e-mail to which a permission link will be sent. This e-mail must be in the database. Once the client requests the change, the server will do this check, in addition to double-checking if the typed e-mail pattern agrees with the pattern of an email (matcher class .find() method). Verification is performed by the method within the UserDaoImpl class, which executes the query query to the DDBB.

Possible failures that can occur in this action are handled by sending attributes to the view model so that they are displayed to the user via the addFlashAttribute method of the RedirectAttributes class. This method adds the model we want to the view before redirecting the user to a desired page. For the password recovery method, the following is verified:

* If the user does not enter the email correctly or does not fill in the e-mail field (which for some reason the need of the html does not work), the 'EMAIL\_ERROR' attribute will be sent to the view, after redirecting to the login page.
* If the e-mail entered is not in the BBDD, the 'USER\_ERROR' attribute will be sent to the view, after redirecting to the login page.
* In case none of these conditions occur, it will send the'EMAIL\_OK'attribute to the viewand the mail will be sent via the sendEmail method.
* sendEmail method: This method receives as a parameter the object of the User class. This instance is exactly the user that contains the e-mail embedded in the modal, because once the mail is found in the BBDD, the driver makes a query query for the user to return.
* To send mail, the SendMailService class method (.sendMail() is used. Before calling this method, we define the subject, the body of the e-mail (with the tags and styles you want) and the link that we want to be inside the body.
* Class SendMailService: This class will be responsible for sending mail to the user. The attributes defined in the sendEmail method will be the parameters used in this class. The Spring Framework's email support – JavaMailSender – is used in conjunction with mimeMessageHelper class, responsible for generating the content in HTML (with links, images and/or attachments).
* Link: The link to be sent in the body of the e-mail will be composed of: protocol + domain + port + path + query + parameters ('http://localhost:8080/resetpassword?id=XXX'). The id is obtained with the user parameter passed to the sendEmail function.

## Reset password page (URL: ‘/resetpassword?id=XX’)

### **GET Reset password**

The GET method of this URL receives the user's id through the @RequestParam. Once it gets it, the driver checks to see if a user with this ID exists. If it exists, it sends the user object to the view, otherwise it redirects to the login page.

### **POST Reset password page - "/forgotpassword"**

The POST method receives from the view the parameters entered by the user (the password and its confirmation) with the annotation @RequestParam. Initially checks whether the passwords entered in the fields are the same. If they are, it will modify the password and update in the BBDD with the update controller method and redirect to the login page. If the passwords are not the same, the 'FIELD\_ERROR' attribute will be sent to the view after redirecting to the same reset page.

## Main page (URL: ‘/main’)

We use a filter (similar to the charts page) that will show us the recordings made within the selected dates. By default this filter will be set in the last week from the day we are.

## Charts page (URL: ‘ /charts’, ‘ /charts2’)

As on Main page, these two pages use a filter that queries the database, but only retrieves the dates of the recordings so that they can work with Javascript and display the graphs.

## Ventanas Modales

### **POST Change password (modal) – “/updatepassword”**

The POST method receives the current password and the two parameters of the new password sent by the view through the @RequestParam. The function of this method is simply to handle possible errors and/or successes. First, check if the RequestParam password variable (currentpassword) is the same as the user (obtained from cookies) that is in the BBDD, if they do not correspond, it sends the parameter PASSWORD\_ERROR to the view. If applicable, check if the two password entries match. If yes, the driver will update the password and send the message to the view PASSWORD\_OK, otherwise it will send a model with an error message to view ("PASSWORD\_ERROR FIELD\_ERROR").

### **POST Change name (modal) – “/updatename”**

The POST method receives the modified entry from the view name. If the field is not empty, the driver updates the user object with a query and sends a message to the "NAME\_OK" view.

### **POST Change email (modal) – “/updateemail”**

As well as the 'Change password' method, the post receives the current password and the two parameters of the new mail sent by the view through the @RequestParam. First it checks if the RequestParam password variable (currentpassword) is the same as the user (obtained from cookies) that is in the BBDD, if they do not correspond, it sends the parameter PASSWORD\_ERROR to the view. Once the fields of the two email entries match, check if this email already exists in the BBDD. When all conditions are favorable, the User object is updated.

## Camera Connector y Camera Connector Controller

These methods are responsible for connecting the server to the video surveillance camera.

From the camera we can receive two types of notifications, that of sending an email and that of recording finished. In the first we call the SendEmail method, which is responsible for iterating all users and sending an email warning of new movements in the camera sensor. When the recording is finished, we receive the notice that is responsible for saving the recordings in the database, receiving date, duration, camera and location of the video within the raspberry in order to recover it in the front windows.

## Queries

Queries are used to find and update user attributes with different queries. Different input parameters are used, as well as different object returns.

## Cookies

Cookie-related methods will be shared across all pages of the application and, so that they are not repeated in all classes, they are grouped into a single – SharedMethods.

For the use of cookies, one of the necessary methods is the method to raise them. The application creates a cookie with the username key and its value. This data is obtained once the user logs in to the application. For future use of these cookies, the other pages use a method of recovery (findCookies).

Once the LogOut is selected, the initially created cookie is deleted with the deleteCookies method.

## Messages

PASSWORD\_OK = "Password changed"

PASSWORD\_ERROR = "Wrong current password"

USER\_ERROR = "User not found"

EMAIL\_OK = "Email sent"

EMAIL\_ERROR = "Email field is empty or format is incorrect"

EMAIL\_ERROR1 = “Email already exist. Choose another one”

EMAIL\_ERROR2 = “Email no registrado”

FIELD\_ERROR = "The fields do not match or there are required fields empty"

NAME\_OK = "Name changed"

NAME\_ERROR = "Required field empty"

# Client – Python language

## Software:

The program is divided into 5 code files and one configuration file.

### **Communication module**:

### It handles communication between Python and the Java server. It tries to connect to the server with a timeout of 10 seconds, if it does not succeed, the program will start anyway, but warning that it is not connected to the server. You can connect when the server is up and running. It notifies the server when it has to send an email, and sends a String with the necessary data to query the DDBB. It uses the socket library.

### **Stream module:**

### It takes care of the PiCamera and streaming. Primarily, it uses the picamera, socketserver, threading, and server libraries. In this module there are variables that configure stream and recording. Streaming is always running on the corresponding thread, and controls the camera through splitter\_port 1 (port 1). Recordings are controlled by the record() and stopRecording() methods. The record() method creates the file name, starts recording, and prompts the server (using the communication file method) to send the email. And finally, the stopRecording() method stops the record, and notifies the server to save it to the DDBB.

### **Main module:**

### Its where the main threads are. It consists of 6 types of thread, gui.refresh, gui.checkConnection, motion, timer, main and stream. The motion thread is responsible for controlling the infrared sensor. Timer threads control the duration of recordings, and control the size of files with an expandable limit of 30 seconds. One is created with each recording, and they are automatically cleaned when you reach 10 used threads. The stream is responsible for controlling streaming and recordings. The thread gui.refresh is responsible for updating the data in the graphical part of the program. The thread gui.checkConnection periodically checks that the connection to the server is successful. The main thread is responsible for connecting the thread stream, the thread movement, and the timer thread.

### **Gui module:**

### It takes care of the graphical part of the program. It updates visually and in real time, the status of the connection to the server, the status of the streaming, and the status of the recording, as well as the duration of the recording. It is also responsible for applying user settings, and data persistence, using the pickle library.

### **Exec module**:

### Contains the main flow of execution. Calls the main.init() method, which in turn takes care of starting all threads and variables in the main module. It also contains the main thread, stream.window.mainloop(), which starts the graphical part. The config.pickle file saves the user's settings.

## Graphics interface:

### **Status Panel:**

* Server connection status:

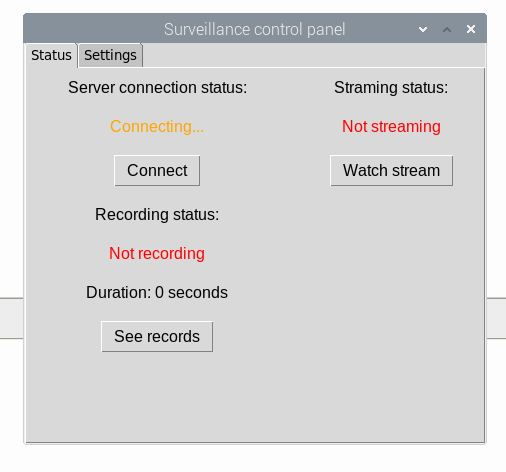
Displays the status of the connection to the server, and allows you to connect or disconnect with the button.

* Streaming status:

Displays the status of the stream and the Watch Stream button opens the stream web page.

* Recording status:

Displays the status of the camera to indicate whether or not you are recording, and if you are recording, displays the length of the video. The see records button opens the folder that stores the recordings.



### **Settings Panel:**

* Recording resolution:

Selects the resolution of the recordings.

* Recording fps:

Selects the frames per second of the recordings.

* Recording limit:

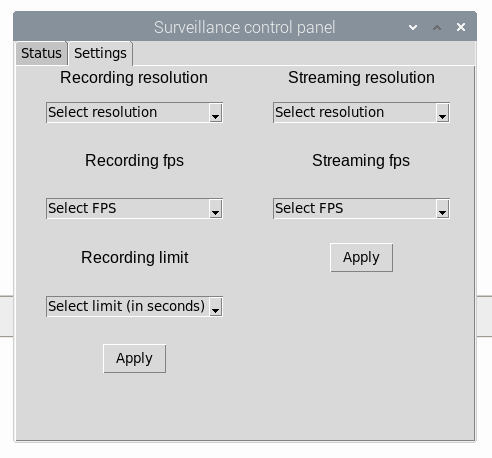
Select the duration limit of the videos.

* Streaming resolution:

Select the resolution ofthe stream.

* Streaming fps:

Selects the frames per second of the stream.



# Database

We use a MySQL database composed of two tables that are not related to each other.

## Tabla users

It consists of 5 fields:

* ‘Id’ (int, long 11, primary key)
* ‘username’ (varchar, long 16, unique key)
* ‘password’ (varchar, long 16)
* ‘email’ (varcha r, largo 50, unique key)
* ‘name’ (varchar, largo 50)

## Table records

It consists of 5 fields:

* ‘id’(int, largo 11, primary key)
* ‘date’ (Datetime)
* ‘duration’ (int, long 11)
* ‘camera’ (int, long 11)
* ‘video\_location’ (text)