SVTANALYTICS
Seeing is Believing

SOFTWARE ARCHITECTURE DOCUMENT

Author: Gerard Mundo Bosch
Content Owner: SVT Analytics

REVISION HISTORY

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1	v0.5	Monday, May 6

TABLE OF CONTENTS

1. INTRODUCTION

Retail companies are constantly adapting to new technologies. The customer experience is key to succeed and they know it. So they don't hesitate to place more attractive signs, redesign the layout, hire nice employees and use soft colors.

Apart from that, the continuous expansion of the digitalization of the information, combined with cheaper storage and computer infrastructures is directly impacting these companies. Today's stores are equipped with digital signage, free wifi, surveillance cameras, POS directly connected to the company's database, and more. The traditional retail store model is already obsolete in favor of one more adapted to the information technology era.

However, the same ideas behind the scenes apply: attract customer's attention, and build a loyal customer base.

1.1 PURPOSE

1.1.1 PROBLEM DEFINITION

A great way for companies to gain customer loyalty is to learn their likes. There are some ways that store and mall owners already use to that purpose:

- Customer satisfaction and feedback surveys
- Online website reviews
- Sales associates analyzing the retail sales floors
- Data collection from the POS
- Observation of product outflow

But they have problems:

- Customer's time and effort
- Manual collection and analysis of feedback by the employees
- Qualitative vs. quantitative appreciation
- Misinterpretation of the cause of some effects (empty shelves <> likes)

Events that happen in the store must be carefully handled as soon as they happen.

Placing advertisements in stores is difficult too. Reaching the right people for each advertisement is a hard task, and digital signage has helped with that up to some extent. Grabbing consumer's attention with advertisements is essential: it provides brand recognition and highlights popular products.

How to know the effectiveness of the advertisements in the stores, and whether they have or not a significant impact on sales?

Predictions also help the correct company development. They should extend to weeks and months, and not only relay on internal data.

1.1.2 OBJECTIVES

Our system aims to give management a new tool to analyze customer's likes and, in general, understand their behavior inside the store. This will ultimate equip the store with a cutting-edge new technology to improve the customer's experience, increase retail growth and help the development of the company.

With our system in place, the owners will have correct and accurate information to support their vital business decisions.

Through this document, the architecture of the system will be described, as a way that compliments the code and describes what the code itself wouldn't do. This document is intended to describe the architectural decisions which have been made on the system.

1. 2 SCOPE

The project team and developers are working intently on making this project a fully operational and viable technology system for the public market. The vital component was developed some time ago and it's in a quite mature state, but the project as a product is just ending its incubation state to begin its path to become a startup. As any project segues into a product, it is naturally lacking multiple components necessary to be successful in a competitive market, like documentation or project management and schedule.

This document applies to the whole SVT Analytics system except the targeted advertisements feature, which is still in the requirements gathering stage, thus making it hard if not impossible to take it into account in this first release of the document. Also, only a representative selection of use cases and tests is presented and no flow charts are provided due to the lack of previous documents like entities model and insufficient formal and in-depth requirements gathering and discussion about expectations on the system.

In addition to that, the author hasn't had the chance to talk to the other in-house developer to discuss general information about the original component of the system and any decisions made.

Thus, it is highly encouraged at least one future revision of the document to correct inaccuracies, extend it and insert additional feedback and decisions that were made in the past.

I'm curious about whether you are reading the whole document. Please, let me know if you read this.

With these considerations in mind, this document has been made as accurate as possible, and as any developer-made artifact, it naturally contains a few, evident jokes to make the reader smile.

1. 3 GLOSSARY AND ACRONYM LIST

Term list:

- Stakeholder: any person involved or affected, directly or indirectly, by this product.
- Scaffolding: from the Wikipedia «auto-generated code that the application can use to create, read, update and delete database entries, effectively treating the template as a "scaffold" on which to build a more powerful application»
- Javascript: (originally) web-browser interpreted programming language for enhancing web sites in a dynamic way.
- Ruby: dynamic, object-oriented programming language
- SQL injection: the typing of SQL statements in user-input fields with the purpose of messing with the database, gaining access or obtaining private information.

Acronym list:

- SAD: Software Architecture Document.
- POS: Point of Sale, the checkout place at any store.
- REST: Representational State Transfer, web API featuring a state-less client-server infrastructure.
- API: Application Programming Interface, a protocol used as an interface to allow communication between different components.
- MJPEG: Motion JPEG, a video format in which each frame is compressed as a JPEG image.
- CSS: Cascading-Style Sheets, document that describes the appearance of web pages.
- JSON: JavaScript Object Notation, a text-based standard for human-readable data exchange.
- MVC: Model-View-Controller, a software architecture pattern that separates the physical way to store data, the business logic and the appearance to the user.

1. 4 STAKEHOLDERS

Each stakeholder is concerned with different characteristics of the system. Here is a list of the stakeholder roles considered in the development of the architecture described by this SAD.

Name

Software developers
Description
They are the coders of the application
Responsibilities
Write code compliant with the requirements specified by the product owner and customers
Concerns
Security, network, performance, UI, programming language, database, workplace and workstations. And the paycheck!

Name
Software testers
Description
Software developers specialized in testing the application
Responsibilities
Find bugs, security holes and checking the functionality against the requirements
Concerns
Security, platforms, architecture, network, database and Pwn2Own computer hacking contest

Name
Hardware experts
Description
Employees highly informed and with wide experience in devices and components
Responsibilities
Buy and maintain the hardware that supports our application
Concerns
Hardware components prices, new technologies, architecture of the application

Name
Product owner
Description
Conceiver of the original idea
Responsibilities
Lead the developing process, sell the product, listen for product feature proposals and conceive new ideas and features
Concerns
Product success, developer’s happiness ☺

Name
Consumer experience integration companies
Description
Companies that work to enhance customer experience of other companies
Responsibilities
Show truly interest for the product! Sell the product to retail companies and give feedback for developing roadmap and new features
Concerns
Success of their customer companies, find new methods to improve customer experience

Name
Store managers
Description
People holding a managing position at store companies
Responsibilities
Maintain and increase sales and retail growth in general
Concerns
Cutting in expenses, competitors, customers' likes

Name
Store sales associates
Description
Employees working in the open areas of retail stores
Responsibilities
Check items out, place items on shelves, help customers
Concerns
Customers satisfaction, be promoted to store managers.

Name
Store visitors and customers
Description
People interested in buying something or just browsing
Responsibilities
Behave appropriately and be nice. Spend money.

Concerns
Product stock, related products, price, good quality/price relation, good customer attention

1.5 NON-FUNCTIONAL REQUIREMENTS

1. High performance: The system must be able to receive a big number of video streams and be able to process the head count and store it on the database each second.
2. User friendly: the final users will be retail companies' managers, and it should not be assumed that are technology experts at all.
3. Security: all the components must be totally secured, in order to prevent leaks and intrusions that could imply physical security issues inside the stores and data manipulation to harm the company.
4. Failure tolerance: the system should be fail proof and be able to recover and keep working in a matter of seconds.
5. Human errors: humans are the #1 source of involuntary (or voluntary) cause of problems in the systems. The system should always check the user input and, in general, any instruction.
6. Maturity: the system has to run tests every time that a change is made, and new tests have to be built for new features. Both unit and inter-module tests should be done.
7. Changeability: everything is very likely to change, so the system must be able to handle any kind of changes in features.

2. ARCHITECTURE OVERVIEW

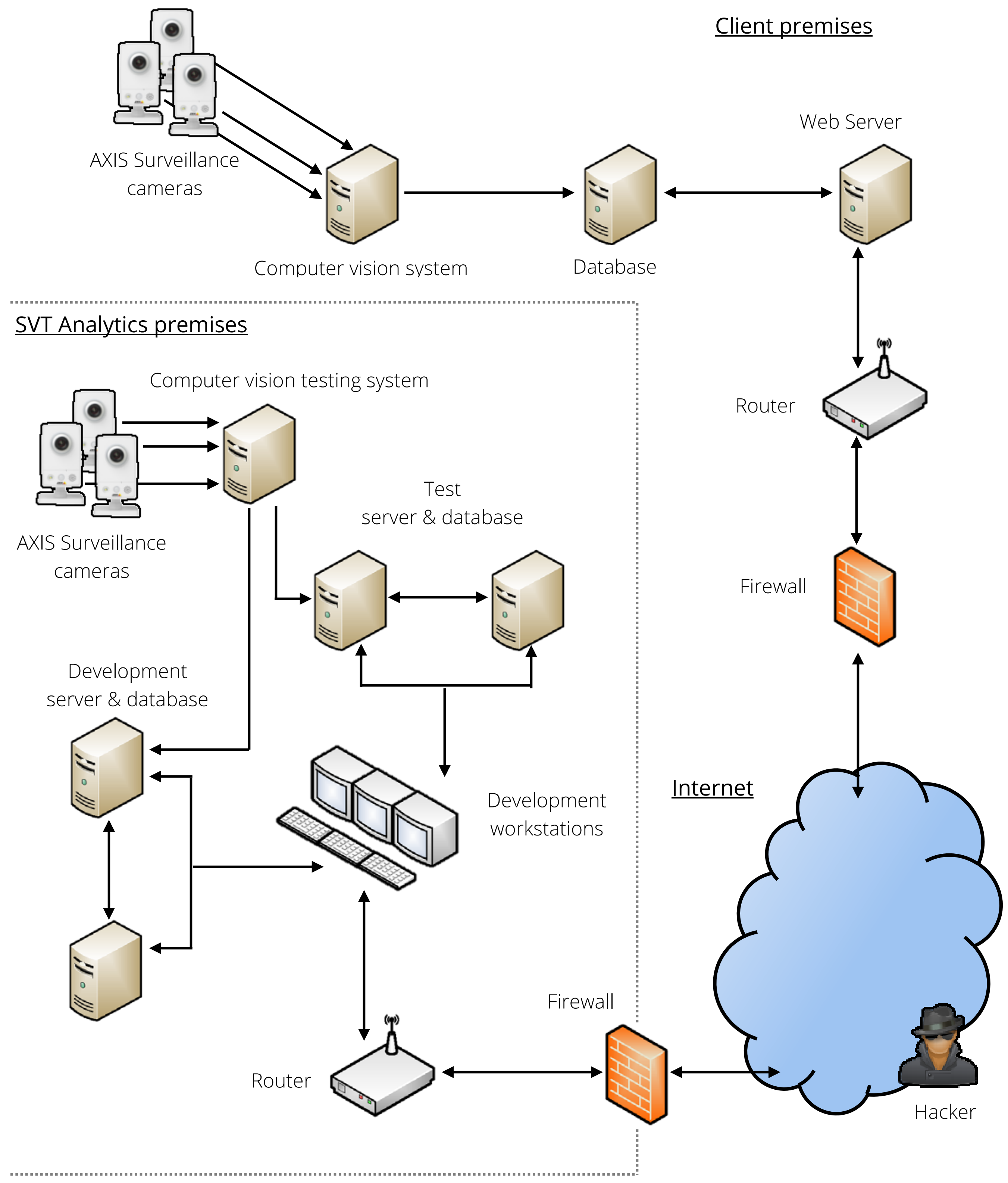
This document is the first approach to present the information of this project in a structured fashion and discuss its architecture. It also provides guidelines for the upcoming half-to-a-year development.

Whenever possible, we make use of existing technology instead of reinventing the wheel and the usability of the system is taken into account as the #1 priority.

The structure that the rest of the document will follow is:

1. A summarized description of the software architecture, including major components and their interactions.
2. Architectural constraints and decisions.
3. A detailed description of each component.
4. System functionality represented by use cases.
5. An outline description of the hardware and software platforms on which the system has been tested so far. Also, where preliminary tests and analysis show they could initially be deployed into.
6. A guide on how to run test cases of the application.

2.1 ARCHITECTURE SUMMARY



The physical system is formed by two main sub-networks: inside the SVT Analytics premises and inside any client premises.

The first one is where the developers have their workstations to work. There is a server and a database for development purposes and a server and a database for testing. There's also a bunch of cameras and a server for computing the headcount on those cameras. This server fills both the development and testing databases.

The workstations are connected to the internet through a router behind a firewall, for security concerns. Inside the company premises there's the same entry configuration.

However, the only machine that can be accessed is the web server. The database is filled by both the web server and the computer vision system. The cameras send a live stream to the computer vision system.

All the network connections are wired, except the surveillance cameras' one, which is wireless. If it's not feasible to have separate network connections between servers that don't need to be connected, it's possible to join them in a single network but the server should be in a separate one that has access to the Internet. Inside the SVT Analytics ideally only the workstations should have access to the Internet.

The current status, however, is another one inside the SVT Analytics premises: the development server and database, and the computer vision developing system are all inside the single development workstation that we have right now. Our testing server and database, and the computer vision testing system are outside the premises, inside the InReality company premises.

Combining systems into a single computer is not recommended, mainly for performance issues.

2.2 CONSTRAINTS AND DECISIONS

About the architecture summary:

- The web server is the only computer accessible from the Internet. This is for enhanced security in case of a break through: no camera's images would be accessed. However, this implies that the computer vision system can't be updated
- The reason to have separate test and development servers and databases is that when developing things can break up very easily, so it's good to always have a testing server with code that works.
- The cameras are AXIS for an unknown reason. The same applies to the chosen model, even though a guess is that their look is nicer and more modest than the typical, pipe-shaped ones.

- If you are wondering why is there a hacker depicted on the Internet, it's because they will be there and will try to break in or mess with our system, so we better have them present.

Concerning the technologies chosen:

- Windows DirectShow Graph Tool: unknown reasons.
- PostgreSQL as the database management system.
  Initially we were using MySQL but after some research, MySQL was clearly not the best open-source solution. Other popular solutions are Oracle and SQL server, both not free solutions.
  The reasons for choosing PostgreSQL are:
  - Free
  - Actively developed
  - Geometry type support included
  - Excels at workloads with a significant write workload (which is our case)
  - Advanced Query Optimizer based on cost, with many plan choices and adaptive statistics
  - 5 different kinds of index
  - Compresses/Decompresses data on the fly, reducing I/O time
  - Asynchronous support

  In the future, it will be interesting to investigate No-SQL solutions, as they are said to be designed for big volumes and high speeds. For now, they have low maturity, are not supported by big companies that can solve any issue in a company environment, and offer few facilities for ad-hoc query and analysis.
- Ruby on Rails as a web framework (and Ruby as a programming language).
  There are several frameworks out there. Initially we were using PHP but a lot of pieces of code would have been hard to write, specially a RESTful API. Another version had been made with the Grails framework (Groovy language), but we never could even run it.
  Ruby on Rails is being constantly updated and a lot of features are being frequently added. This is both good and bad, because this means that things could break with an update. But we'll be using cutting-edge features and tools. The only alternative that we've experience on is CakePHP, and it's a far less useful solution. Other popular solutions are symphony, Zend, Node.js, Django, Pyramid, Sinatra, Jifty, Catalyst, ASP.NET MVC, Struts, Java Server Faces and Spring. The main reasons for choosing Ruby on Rails are:
  - Good reviews that I've received from trusted colleagues.
  - Good reviews on the Internet.
  - Good community support, a lot of manuals, tutorials and examples.
  - Twitter, Groupon, GitHub, Yellow Pages, Shopify, Mingle2, Soundcloud, Kongregate and tons of other sites use it.

    - o Great support for No-SQL databases, and as mentioned before, it will be a good idea to research and eventually switch to them.
    - o Includes scaffolding and creates REST APIs out of the box.
    - o Ruby is good for processing data.
    - o Great tools for developing.
- JavascriptMVC as a front-end web application framework.
  It's overwhelming the amount of web application frameworks that are out there and the good reviews that a lot of them receive. It was hard to make a decision on which one to go for and Backbone.js was really close to being chosen.
  The reasons for choosing JavascriptMVC are:
    - o True MVC architecture
    - o Offers great flexibility
    - o Mature
    - o Packs everything into a single Javascript file
    - o Includes scaffolding
    - o Includes factory testing and documentation modules

  The first three were the determining reasons for choosing it. Other options were AngularJS, Ember.js, Dojo, YUI, Kendo UI, Spine and GWT.

Miscellaneous:

- JSON to transmit data between server and web application, instead of XML: JSON takes up less space, it's much easier to build both manually and from an object in memory and it's human-readable. Also, No-SQL databases quickly output JSON, as it's usually the way things are stored in such kind of engines.
- Storing date and timestamp for each head count of each camera was decided to be separate in two DATE and TIME fields instead of a DATETIME field. Also, a redundant day of the week field has been introduced. The majority of queries will be for certain dates at certain times or all times, instead of a time range, so the reasoning behind this decision is:
    - o Having the day of the week pre-computed will save time when looking for certain days of the week.
    - o Separating date and time fields will allow ignoring the date without any additional processing.

It's a constraint that any component must be able to be runt on a Windows platform, which is a more stable operating system than Linux, and with more support.

3. COMPONENTS

The system is based on five main components. The first two of them can only be described as a black box due to lack of internal information.

3.1 CAMERA SYSTEM

It is a surveillance system with several cameras that are installed at any spot that the manager of the store or company wants to take into account for the analysis. The cameras are AXIS, and there's no way that I can check the exact model right now. It might or not be possible to use other brands or models but this document assumes this specific one.

3.1.1 RESPONSIBILITIES

Its duty is to record video and take pictures, and send them when requested. It also has the responsibility of rejecting unauthorized connections to watch the video stream or capture images.

3.1.2 FEATURES

It features Ethernet and wireless network connections, standard VGA video definition and microphone (I believe – not checkable now).

3.1.3 SETUP

The camera must be connected to a power supply and to a computer via Ethernet. The camera must be then configured following the instructions on the Instructions Manual, making sure to connect it to the wireless network and assign it a unique, static IP address.

Once this is done, both connections can be unplugged, the camera must be fixed to a wall facing the area that must be analyzed and connected to a power supply, making this one last connection the only one permanent (thus, making its setup the most non-intrusive possible for a store).

3.1.4 USAGE

The camera supports different streaming methods, but the one used by the computer vision filter is MJPEG at 320x240.

The usage is as follows:

When the camera receives a petition for the /axis-cgi/mjpg/video.cgi, asks for credentials and, if they are valid ones, it keeps sending JPEG frames until the connection is closed.

3.2 COMPUTER VISION SYSTEM

The computer vision system is a DirectShow filter that can be runt using the Windows DirectShow Graph Tool.

3.2.1 RESPONSIBILITIES

This component must receive a stream of video, compute the head count present in each frame and communicate such count when requested.

3.2.2 FEATURES

The filter receives a stream of JPEG frames, computes the amount of people that appear there and saves in memory the last count made. It can output pictures with a rectangle around any shape recognized as a person.

3.2.3 SETUP

To setup the filter, follow the steps within the document SVT-Analytics Installation Procedure.

3.2.4 USAGE

The filter waits for a UDP connection. Once it’s made, it sends the head count, along with the date, time and camera ID, at the desired timing.

Right now, what it does is to send that data once per connection made.

3.3 DATABASE

The database stores data from the cameras at the interval specified in the code but also stores business-related data: users (credentials), stores, floors, camera location and shape of the area covered, alarms, dashboard configuration, saved chart queries. It also saves information from external services: weather, traffic and events (holidays, celebrations, disasters…). The engine where the database is handled is PostgreSQL.

3.3.1 RESPONSIBILITIES

This component must securely store all the given data fulfilling the ACID specifications that make database transactions reliable: Atomicity (if something fails, nothing is saved rather than saving something that doesn’t make sense), Consistency (do not violate any constraints), Isolation (concurrent queries do not interfere in any ways each other) and Durability (the data will be saved persistently).

It also has to perform as fast as possible and make a good memory management, and support the common SQL language functionality.

3.3.2 FEATURES

Makes usage of B-tree and Hash indexes. It's automatically built and maintained by the web framework. Handles the shapes of the area that each camera covers in geometric types, storing the points efficiently and providing tools to properly operate and modify them.

3.3.3 SETUP

To setup the database engine and structure, follow the steps within the document SVT-Analytics Installation Procedure.

3.3.4 USAGE

The usage is automatically handled by the web framework. However, it can be accessed and used using any PostgreSQL client, like pgAdmin.

3.4 WEB SERVER

The chosen technology is Ruby on Rails, a web developing framework for the Ruby language.

The chosen web server for future deployment is Mongrel. Right now we use WEBrick that comes by default. The reason for the change is that Mongrel is faster, more efficient, and stable. However, we've never used it yet, so the description here is made for WEBrick.

3.4.1 RESPONSIBILITIES

The web server must serve a static, plain webpage along with all the web application components: CSS, images and Javascript.

Then, it must provide a RESTful API to allow the retrieval, insertion, update and deletion of entities in the database.

The RESTful API also needs to provide an initial way of authenticating the user to allow access to the API, and has to handle permissions appropriately.

3.4.2 FEATURES

Full MVC architecture pattern behind the scenes with transparent access to the database using an automatic mapping of the objects to the corresponding tables, providing a REST API that uses a JSON I/O formatting.

Dynamic creation of graphics (heat maps, histograms, pie charts…) via the ImageMagick software.

Protection of privacy-sensible files like floor maps against access from unauthorized parties, checking the permissions to access each private file through file dispatchers instead of direct file mapping.

Process that checks for the conditions that could raise an alarm, and takes any appropriate action if indeed one goes off.

3.4.3 SETUP

To setup the web server and place the code files, follow the steps within the document SVT-Analytics Installation Procedure.

3.4.4 USAGE

To run the server with the code, navigate to the root of the code with a command line and type "rails s". To edit the code, we recommend Aptana Studio.

3.5 WEB APPLICATION

The Web application is runt under JavascriptMVC, a framework for web front-ends written in jQuery (Enhanced Javascript).

It consists on a single-page website (actually, two-page website because the welcome is made in a different one) executing the JavascriptMVC code.

3.5.1 RESPONSIBILITIES

The web application must provide a continuous, interactive and dynamic experience to the user, allowing access to all the information that the user requests and facilitating any task that the user wants to perform over the system.

It must provide comprehensive error messages whenever one happens.

3.5.2 FEATURES

Surprisingly, this front end framework also features a full MVC architecture pattern behind the scenes, all running as jQuery widgets, with models quite compatible with the ones on the server side.

Dashboard customizable with multiple gadgets (charts, alarms' quick review, live streams…) and different layouts (one big gadgets, two side by side gadgets, two small vertical gadgets and a big one, etc…

Configuration of the physical layout of the store (floor names and maps upload, cameras' location, id and shape, etc…).

Step-by-step wizard for the creation of charts and option to save the results.

Separated options and functionality per store.

Alarms section, with options for setting up alarms based on amount of people in a certain area (and others to specify), and with actions to take (send email, send text message). A visual notification when browsing the website is always shown.

Review section to display a history of alarms that went off, periods of time with people on certain areas (or without people in certain areas), and more.

When a request is made and fails, it's retried multiple times and finally a popup is shown asking to retry before continuing.

3.5.3 SETUP

There is no setup for the web application. It's included within the web server files.

3.5.4 USAGE

The code runs within the browser when the page is loaded. However, it's desirable to pack the code in a single file when the version is about to go to the testing server, so that a single request to download the whole file is made.

4. TESTING

4.1 COMPUTER VISION SYSTEM

This section should be better answered by the author of the filter.

The filter can be (manually only?) runt with video from a file.

A way of semi-automatically test the filter is to have a set of videos and a program that connects to the filter and matches the output data at each point of time with a saved file that contains the true data that should be output. This kind of systems are not 100% precise, so rather than a pass/fail response, this program would ideally answer with an accuracy percentage.

What the human would have to do is to load the file on the filter, load the file with the ideal output on the program, and play the filter at the same time that the program connects to it.

Some interesting test cases:

- A video with no one in it, in a room with no objects
- A video with no one in it, in a room with objects
- A video with no one in it but containing some artifacts
- A video with one or more moving objects (or animals?) in a room
- A video with one person entering and leaving the camera angle of vision
- A video with multiple people entering and leaving the camera angle of vision
- A video crowded of people moving

Other interesting cases would be to try videos recorded with different angles, interior designs, light and speed of people walking.

4.2 DATABASE

Writing a command-line script that runs a PostgreSQL CLI client, and checks the output of the following tests:

- Connection to the database engine succeeds
- Database exists
- All the tables exist
- Definition of each table is as expected

It would also be interesting to test insert, updates, deletes and selects encapsulating them in transactions and rolling back at the end, instead of committing.

4.3 WEB SERVER

Ruby on Rails comes with a default Test::Unit framework for testing internal functionality. RSpec allows writing more thorough test suites, but based on the views that it generates, which is not what we are looking for since the views here are at the web application side.

The tests are run using the command line.

Having in mind that the web server offers a RESTful API in mind, some of the tests that could be written are:

- It's possible to log in/out
- For each entity that is exposed to the RESTful API, it's possible to insert new entries, and only update, select and delete the ones that it has access to. For example:
    - Create a new floor
    - Try to view all the floors
    - Try to view all the floors for a given store that the user doesn't have access to
    - Try to view all the floors for a store that the user does have access to
    - Update a floor with an incorrect value
    - Update a floor that doesn't belong to the user
    - Update a floor that does belong to the user
    - Delete a floor that doesn't belong to the user
    - Delete a floor that does belong to the user
- Models (internal functions) should be tested too. For example, for the Cameras model:
    - Check that the to_g function creates a shape that corresponds to the camera's angle of vision area
    - Check that the to_s function of the shape that is returned gives the correct string version of that shape.

  Another example, for the Floors model:
    - Check that the image returned by the heatmap() function is as expected, matching it with one that will be stored.

4.4 WEB APPLICATION

JavascriptMVC comes with FuncUnit, a functional and unit testing framework integrated.

It's based in QUnit for organizing tests and assertions, but it's enhanced to allow to open a webpge, query for elements, simulate user interaction, wait for a condition to be true and get information about the page and run assertions.

So, with FuncUnit you can open the browser and run the tests or you can run the browser automation tools that it comes with, through the command line: Selenium and PhantomJS.

What it fails at is checking the actual visual appearance to the user, the exact layout on the screen; I guess that's something that only humans can do.

Some of the tests that could be run:

- Check that it's possible to log in with a testing account
- Check that it fails to log in with wrong credentials
- Check that the dashboard appears with the desired elements in it
- Check for the account menu list
- Check that the system logs out correctly
- Check for all the menu elements
- Check that they can be clicked and the content loads
- For each content, load its tests. For example, for the floor listing and creation:
    - Try to create a floor without name and without floor map
    - Try to create a floor with a name that includes SQL injection
    - Try to create a floor with an invalid name
    - Try to create a floor with name but without floor map
    - Try to create a floor with floor map but without name
    - Try typing a valid floor name, uploading a floor map and deleting it, then check that the floor can't be created
    - Check that the floors that are listed are correct
    - Check that each floor has the corresponding cameras' area shape on it
    - Check that deleting a cameras' area shape and saving the floor will effectively make the deletion permanent, reloading the floors list.
    - Check that it's possible to draw a camera shape (I DON'T KNOW IF IT'S POSSIBLE TO EMULATE THIS!) and save it.
- Some integration tests could be done too, for example when a shape is drawn on a map, it appears when creating a heat map (but, in this case, how to check an image's content?)