

3D editor of custom guitars, web page and an order management system embedded in a local company.



Grado en Ingeniería Informática

Trabajo Fin de Grado

David Leza Pernaut
Alfredo Pina Calafí
Pamplona, 18/06/2018

*Thanks to Mikel and Iñaki for the chance
of developing a project that will be used,
and time invested.
Also thanks to my family and all the
people that somehow encouraged me
to take all of this in English.*

Summary

This project proposes a software solution that results from the interaction between the web page and an Android app we have implemented, and the manufacturing processes of the company. The novelty of this approach is to offer a 3D editor to build a custom guitar.

In the memory of this project the reader will find the decisions we have taken, the technologies and resources we have used, the details of the implementation, as well as all the information about the process of developing a software solution to “computerize” a local company of handmade guitars called “Cayman”.

This memory contains all the technical information and therefore the company will be able to extend in the future the project with any developer if needed. That’s why we will detail how we have used the tools to achieve the objectives, the structure of the software solution and the scope of the achieved objectives.

Index

Introduction.....	4
Tools and resources.....	5
Web development.....	6
Flask.....	6
MySQL.....	6
Three.js.....	7
Bootstrap 4.....	7
jQuery.....	7
Other JavaScript Libraries.....	7
Iframe de Google Maps.....	8
PayPal.....	9
IPN Protocol.....	10
JWT.....	11
Login.....	13
MVC.....	13
UploadCare CDN.....	14
Android app development.....	15
Machinery handling connection.....	15
Orange Pi.....	17
Work-flow / Methodology of work.....	19
GitHub.....	19
Two environments.....	19
Software developed / Implementation of the objectives.....	20
Front end.....	21
Admin-side design with JS.....	21
3D editor of custom guitars.....	24
Back end.....	26
Database.....	26
IPN protocol implementation.....	28
User manual.....	31
Administration manual.....	31
3D editor Manual.....	47
Testing with users.....	49
Conclusions and future lines.....	50
Bibliography.....	52

Introduction

The origin of this project, comes from the necessity of a small local company of handmade guitars. They are just starting in the business, so they thought a web page could give them some presence. Few talks after, we decided that instead of a simple start-up web page, we could build something that offers more value to the company.

As explained in the proposal for the “Final Degree Project”, the main goal of this project is to “computerize” the manufacturing processes and the sales of handmade guitars.

We will develop a web shop where customers can buy their products and order custom guitars. The company will be able to use the web page to manage the orders received on-line as well as the orders they could receive in-person. This way, all the orders are in the same place, so they don't forget or lose track of any order. This will also allow in the future to get some useful business information such as which guitar model is the most or least sold, and more. So, we can define the tasks:

- Build a web shop
- Build an order management system

as more specific objectives to “computerize” the sales of handmade guitars.

To “computerize” the link between the sales of handmade guitars with it's manufacturing processes, we will create an Android app to provide the company an easy solution to check and update the orders, and also be capable of handling the machinery they use.

Then to “computerize” the manufacturing processes with the Android app, there will be a part of the app to handle the milling machine they use to cut the wood into a guitar shape. The Android app will send the commands to a small and headless computer placed next to the milling machine. By removing the necessity of a desktop computer to control the milling machine, we want to ease this manufacturing process of cutting the wood.

So in only one app they can see the incoming orders, and then handle the machinery necessary to start crafting the guitars they have been ordered. Saving time, and also keeping the working environment as clean as possible by removing one desktop computer with it's screen, keyboard and cables connected to the milling machine.

The specific objectives to “computerize” the manufacturing processes are:

- Create an Android app
- Create an interface with the milling machine

In order to stand out from other web pages of guitars, we thought about developing a custom 3D editor of custom guitars. Where the customer can visualize an idea of their custom guitar, and then ask for a budget.

After a small research on the internet and other companies in the guitar business, we didn't find any company that already offered a 3D editor to build custom guitars and then be able to ask for a budget or order of it (as of 5th of June of 2018).

Worth mentioning that there is already a [web page](#) offering a 3D editor of guitars, however there is no option to buy or order a guitar that matches your design.

We found few web pages that allowed 2D editor for guitars, such as [frankmontag](#) or [haloguitars](#).

Almost none of the big companies have any custom guitar editor, not Gibson, not Epiphone, not Gretsch, only [Fender has a custom editor](#) but it's still in 2D.

As the 3D technology is increasing its popularity these days, we thought that we could be one of the first companies offering this service, hence trying to push forward the guitar business in the technology area.

- 3D editor of custom guitars

This way, not only are we offering a unique 3D editor in a web page with shop and an order management system, but also a complete (software) solution that embeds both, in the manufacturing processes and also in the way of working of this company.

As this project is planned to be used day to day by the company, we will run some tests with users, to see whether we need to improve the solution or if we already did a good job.

- Run some tests with users

In summary the specific objectives of the project are:

- To build a web shop
- To build an order management system
- To create an Android app
- To create an interface with the milling machine
- To design and implement a 3D editor of custom guitars
- To run some tests with users

Tools and resources

In order to achieve the objectives, we have taken advantage of using some tools, frameworks and resources, as they allow us to work smarter. Here are the tools and resources grouped by the different environments.

Web development

Flask



For the web development we have been using Flask as a framework. Flask is a micro-framework for Python based on Werkzeug toolkit and Jinja 2 as the template engine.

Flask is called a micro framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.

Along the project we have been adding new extensions that had facilitate our work. Some of the main extensions that have been used are:

- **Flask-JWT**

It was used to secure the RESTful API needed to connect the android app to the server database, where the orders are saved.

- **Flask-Login**

Used to build a strong and secure system of users, that the administrators are going to use to keep up to date their web page. It also comes with some python decorators that made really easy the part of checking if you have authenticated before accessing a route of the web page.

- **Flask-RESTful**

It simplified the process of building our custom RESTful API. By adding a few lines to the models, and adding a resource file declaring the methods allowed with its logic, we were able to made the API that the android app would connect to in the future.

- **Flask-SQLAlchemy**

An ORM (Object-relational mapping), is a programming technique for converting data between incompatible type-systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. This way it was easier to handle the database, as we abstract from the SQL syntax and think of it as objects we wanted to save information.

MySQL



It is an open-source relational database management system. To save the information about the orders, the products available, the options in the 3D editor, we needed a database. The choice was made because it is open-source and it had no cost. It uses SQL (Structured Query Language) to retrieve and save information permanently in the server. To make it more usable we are using also PhpMyAdmin as an administration tool, as it is widely used and we are already familiar with it.

Three.js

three.js

To build the 3D editor, we have used Three.js[1], an API (Application Programming Interface) that have been used to create and display animated 3D scenes in a web browser. It uses WebGL.

One of its main features is that it is a cross-browser JavaScript library. It is vital that is JavaScript, because the JavaScript is executed in the client side (in the visitor's web browser, by their pc, smartphone, laptop...). Not only it allows the visitor to interact faster with the 3D scene, but also liberates the server from tasks.



Bootstrap 4

For mostly all the CSS we have used Bootstrap 4[2]. Bootstrap 4 is a free and open-source library of CSS. It also uses some jQuery in the background. One of the principal features that we liked amongst others, was the ease of making the web design responsive.

jQuery



We have used also some JavaScript animations to make the web page interactive and have a modern feel. For that reason, jQuery[3] is a cross-browser JavaScript library, that allowed us to easily add this animations. Besides, Bootstrap 4 makes use of jQuery for animations, for example in small screens, when the navbar compacts into a “Burguer” icon of menu, if you click that button, the navbar expands.

It also enabled us to use more libraries, as there are libraries and plug-ins built on top of jQuery

Other JavaScript Libraries

It is also worth talking about some other libraries of JavaScript that we used in the project.

Slick JS[4] is a library to easily create a slider of images, based on **jQuery**. Between its features, fully responsive, mouse dragging, infinite looping, accessible with arrow keys navigation... But without any doubt, the best feature is the “Slider syncing”.

This feature create two sliders and sync them to display the same image. This allows you to create the main slider displaying only one image, and just below it a slider displaying more images in smaller size. Then, with the sync option enabled, when you decide to change the image of the smaller slider, it changes automatically the image in the bigger slider, making it look as it is the focused image. This was really useful when creating the page of the products in sale, to show in a professional way the images of it.

SimpleLightBox JS[5] is a library, based on top of **jQuery**, to display images, galleries, videos or custom content and control the lightbox with easy to use api. With **Slick JS** we had sliders to show the images for the product, now we needed to add the option to enlarge the image displayed to allow the visitor to check the image in its full size. It also allows the visitor to navigate through a gallery with the arrow keys, this came in handy when building the image gallery.

Popper JS[6] is a library that doesn't depend on any other library. It allows us to manage the pop-ups of the web page. One of the key points of **Popper JS** is that it keeps the element in their original DOM (Document Object Model) context, if the popper exceeds the limits of the page by the right, it will place the popper in another position to make the popper readable without messing the DOM. It was used to create pop-ups of confirmation to delete items within the admin-side of the web page.

Clickable-tr-jQuery[7], a plug-in based in **jQuery**, as its name suggests it makes the tag “tr” (tag that references to Table Rows) clickable. A small and simple function, yet really useful when dealing with lots of columns in a table. You add the clickable link in each row to redirect you to another page where you can display the information for only each row easier.

UploadCare CDN widget, is a widget to upload images to a service called UploadCare CDN that we will talk later. It is a widget that shows up a modal with options to upload images from your local storage, Instagram, Facebook and another social-networks.

Iframe de Google Maps



In-line frame, the `iframe` tag of `html`, displays another web in your `html` inside a frame. It was really important for the company that it had a clear way to show the visitor how to reach them. Google Maps offers you the option to pin point a place in the map and then embed that into an `iframe` in your web.

PayPal



As we wanted to be able to sell products on-line, we had to find a solution to this. After reviewing many options to process payments on-line, we finally opt for Paypal, as it is one of the only platforms and solutions that didn't charge any monthly fee, and it also provides you the chance to allow credit-cards payments for people that doesn't have a PayPal account. Instead of paying any fee per month, they charge you a percentage between 3% and 4%, this numbers are for small business that don't have a big number of transactions. After talking with the company, we both agreed that it is a nice solution for starting companies.

Then, the next step was to find how to integrate PayPal's payments into our web page. As we want to give the company full control of their web site and also make it easy, we used PayPal buttons. In their Paypal Business account, they will be able to create new buttons to process a predefined transaction for each product. Also from the PayPal page, they will be able to manage the payments they receive and the payments they need to make, such as refund a client.

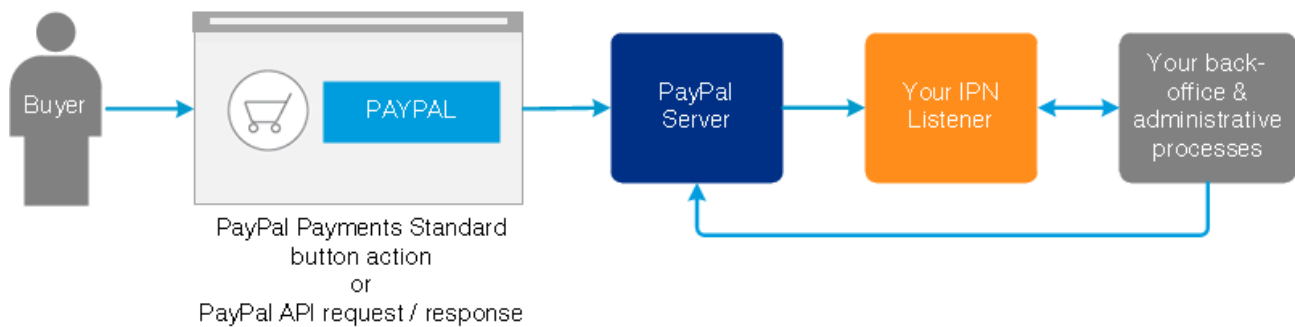
PayPal provides the developers with two options to communicate the state of the transactions. **I**nstant **P**ayment **N**otification and **P**ayment **D**ata **T**ransfer (from now on IPN and PDT). Both IPN[8] and PDT are used to get information about transactions, but they have different goals.

PDT is a service used when you need to show your clients information about the transaction they just made. For example, consider a digital music store. With PDT, this store can let customers download their purchases right away since PDT sends order confirmations immediately. With IPN, such immediate order fulfillment is not possible.

However one of the big advantage of IPN over PDT is that delivery of order confirmation is guaranteed, since IPN resends a confirmation until your site acknowledge receipt. While PDT sends once and only once the confirmation message.

Then we decided to implement IPN listener on our web page. It is worth saying that this is not an excluding decision, you can implement both services in your site. Then you must have in mind that you will receive two confirmation messages for one transaction, one from IPN and other from PDT. Do not process two orders, as only one transaction generated the two messages.

IPN Protocol

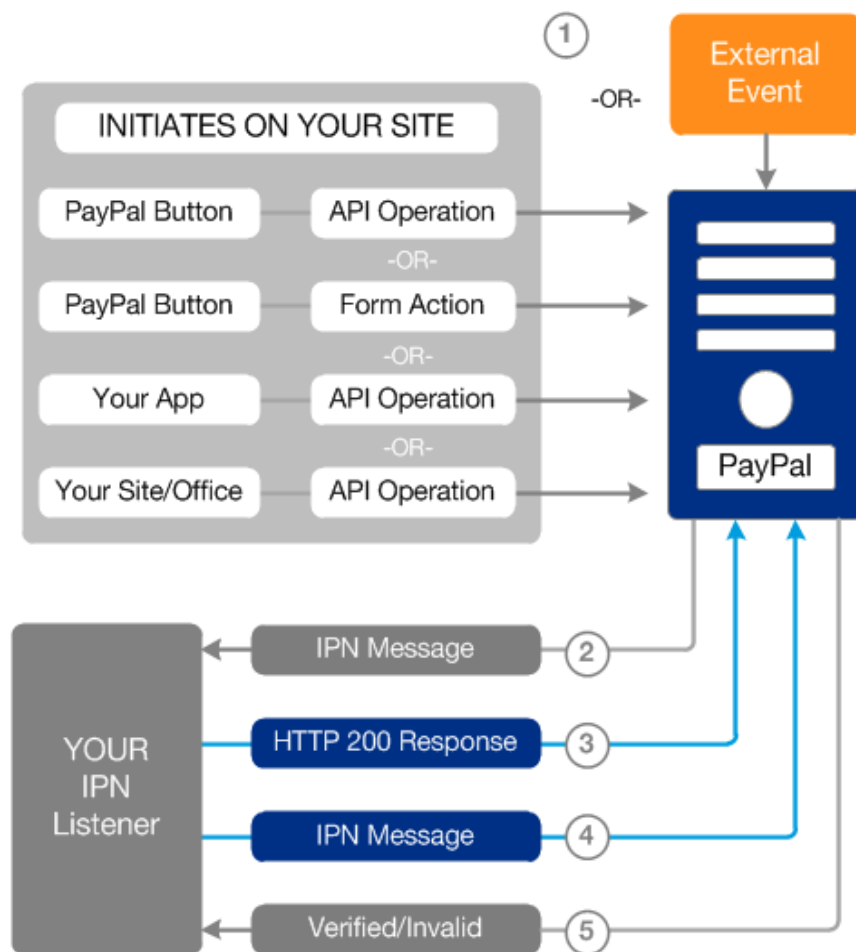


Picture 1: IPN protocol Overview

The IPN protocol goes as follows:

1. Some event such as PayPal button from our web page, or an external event initiates the process.
2. PayPal sends an HTTPS POST request to your listener. This message contains information about the transaction.
3. Your listener response to the previous request should be an HTTP with code 200 success.
4. Your listener makes an HTTPS POST request to PayPal. This request should be the same unaltered message that PayPal sent you first. Same fields in the same order, with the same encoding.
5. Finally PayPal sends a response with only one word in it. It is either “VERIFIED” or “INVALID”, acknowledging you if your message match or not the original.

Further considerations on the protocol. Your listener should always respond to PayPal, independently of you taking action or not. If you don't reply back, PayPal assume that the message have not been receipt, and then sending again the same message with a maximum of 15 retries.



Picture 2: IPN protocol flow

JWT



JWT[9] stands for **J**SON **W**eb **T**oken. It is a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. It allows to secure a RESTful API by authenticating the consumer in each petition.

A JWT token follows this structure, “<header>.<payload>.<signature>”.

The header typically consists of two parts: the type of the token, which is JWT, and the hashing algorithm being used, such as HMAC SHA256 or RSA.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. “exp” stands for expiration time, so the consumer knows when the token will expire, and then ask the server for a new JWT token.

```
{
  "nbf": 1524079769,
  "exp": 1524080069,
  "identity": 1,
  "iat": 1524079769
}
```

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that. The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT token is who it says it is.

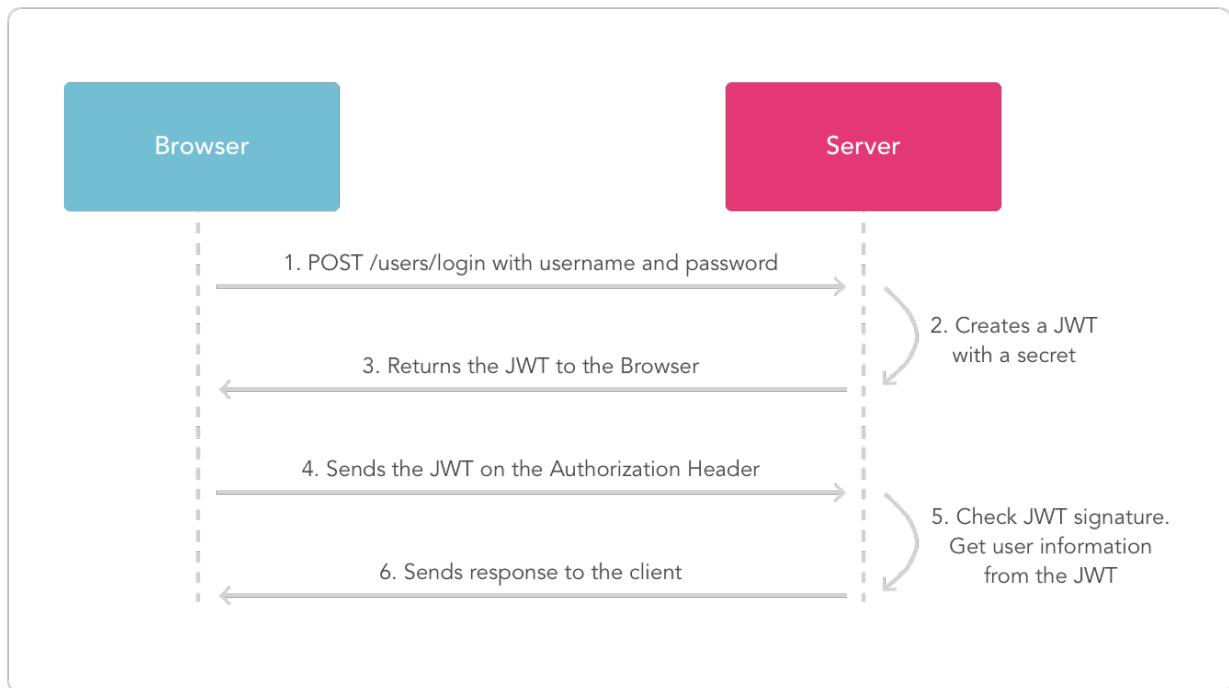
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

All of three encoded in Base64 and combined into a single message separated by dots (“.”).

```
eyJhbGciOiJIUzI1NiJ9.eyJ1bmYiOiE1MjQwNzk3NjksImV4cCI6MTUyNDA4MDA2OSwiaWQiOiE1MjQwNzk3NjksImV4cCI6MTUyNDA3OTc2OX0.H8L-vOBceH8ITqcHW_b5Bwaq1-Xyv7-X9dsgeSEXat4
```

It is vital to have in mind that the purpose of JWT token is to deal with Authentication and Authorization. It is encoded in Base64, so anyone can read it,

however they can not tamper the token. To prevent that anyone steals the token is duty of another tool or certificate, such as SSL for example.



Picture 3: JWT authentication flow

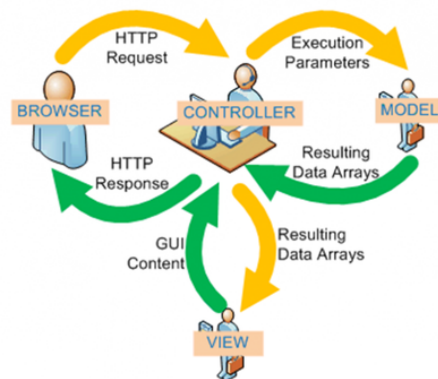
Login

As we only want the administrator to be able to access the admin-side of the web page, we used the previous named Flask-Login. Flask-Login allows us to authenticate and say whether a user is authenticated or not. But things can go wrong, our database can be hacked and exposed to public, making the password of our administrator not safe anymore. So for that reason, it is a good practice and common sense, we are saving in our database only the hashed the password. We hash the password with the SHA256 algorithm released by the NSA in 2001 and widely accepted as strong encryption.

MVC

Model–View–Controller is a design pattern commonly used for developing software that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is

presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.



Picture 4: MVC diagram

UploadCare CDN



Content-Delivery-Network is a geographically distributed network of proxy servers and their data centers. The goal is to distribute service spatially relative to end-users to provide high availability and high performance. CDNs serve a large portion of the Internet content today, including web objects (text, graphics and scripts), downloadable objects (media files, software, documents), applications (e-commerce, portals), live streaming media, on-demand streaming media, and social-networks.

CDNs are a layer in the internet ecosystem. Content owners such as media companies and e-commerce vendors pay CDN operators to deliver their content to their end users. In turn, a CDN pays ISPs, carriers, and network operators for hosting its servers in their data centers.

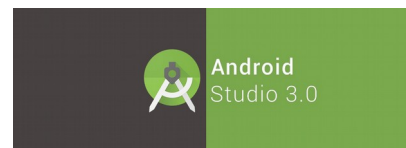
UploadCare[10] is a service of CDN and image processing. Besides the advantages of being a CDN, UploadCare features a simple yet powerful API for image processing. It allows you to request an image by its URL, and by adding to it

some text you can change the image size, crop the image, rotate, add filters, overlay images.

The resize option is really useful when developing the templates. It allows you to quickly change images sizes. This is something you can do with CSS, but the problem is that with CSS, the visitor will download the full image to then resize it, wasting bandwidth, time, and internet data if they are using their smartphones. With UploadCare, you will always be sending the image resize before the client receives it, and also delivering the image faster due to its CDN.

It has a free-plan with limited resources, 500 uploads per month, 500 MB of storage and 5 GB of traffic per month. This is quite convenient when developing a new web page. If you exceeds the resources they offer in the free-plan, you can always stop using the service, resize the images you are using, and upload to the server you are hosting your web page.

Android app development



To develop the Android app we used Android Studio as the IDE (Integrated Development Environment). With Android Studio comes the Android SDK and tools that are almost all you need to develop an easy android app as ours.

It's worth mentioning Volley, an HTTP library that makes networking for Android apps easier and most importantly, faster.

Some of its features are:

- Automatic scheduling of network requests.
- Multiple concurrent network connections.
- Transparent disk and memory response caching with standard HTTP cache coherence.
- Support for request prioritization.
- Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.

- Ease of customization, for example, for retry and backoff.
- Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
- Debugging and tracing tools.



Machinery handling connection

As we wanted to embed our solution into the manufacturing process of the company, we thought that adding this functionality into the Android app was a nice way to have it always available.

The company uses a milling machine by numeric control (a 3 axis drilling machine). After talking with the company, we all agreed that they didn't need to move the milling cutter freely. All they needed was to be able to start and stop some predefined jobs in the milling machine.

The software that was currently handling the milling machine was "Repetier". Originally Repetier is meant to be an application for your 3D printer. It is widely used and has large community, this was useful as it is easier to find help and people that had the same problems before. Even though Repetier is a 3D printer software, they offer the possibility to build your own version of the application based on Repetier. The sellers of the milling machine, have their own version of Repetier that handles the milling machine.

It was quite convenient that Repetier offers a host application and a server application. In their website you can find an API and a guide[11] on how to connect and send commands to the Repetier-Server. The server application starts up a web server that acts as the front-end where you can set up the 3D printer.

There are two ways of communication with the server, Websockets and a REST Api. The Websockets offers you more information about the status and events happening in the 3D printer. As we just wanted the user of the app, to just click a button to start a job in the milling machine, we decided to use the REST Api as it was easier to implement.

First, I read the API and find what commands would be useful to send to the server. Gladly, the Repetier company hosts a demo Repetier-Server page where you can start messing and trying things. The basic structure of the HTTP petitions for the Api is:

```
http://SERVER:PORT/printer/api/<slug>?a=<websocket command>&data=<json object properly url-escaped>&apikey=<API key>>
```

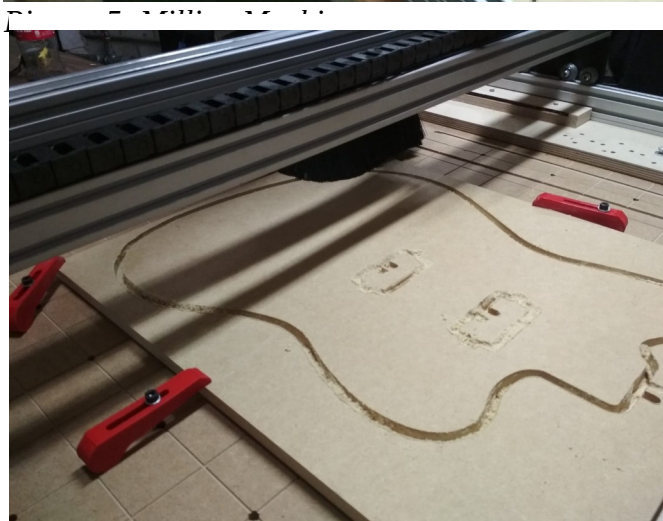
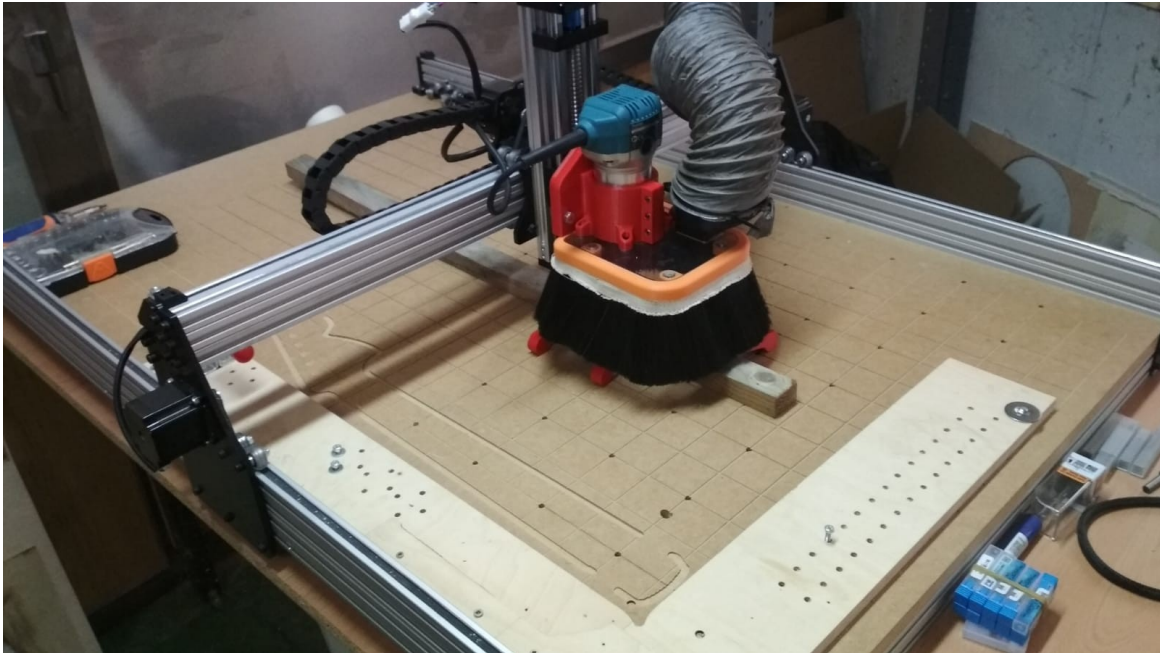
These were the websocket commands that we needed and successfully worked on the demo server :

- listModels. Lists all g-code files for the given printer. The response also contains some statistical data.
- copyModels. Takes as parameter the id of the model you want to copy into the print queue. Copies a g-code to the list of print jobs.
- listJobs. Lists all print jobs.
- startJob. Takes as parameter the id of the job. Starts a print.
- stopJob. Takes as parameter the id of the job. Stops a print.

Afterwards, when tried to make the milling machine work as expected with the commands, it wouldn't move the milling cutter. We finally realized that the milling machine, was indeed connected to the Repetier-Server, but not correctly configured. The problem was that we were using an original version of the Repetier Server (meant to 3D printing), instead of the custom version for the milling machine.

We called the sellers, and asked if they had the Repetier Server version for their milling machine. After installing and setting up the machine with the server, the milling machine was starting the jobs correctly.

To send the petitions to the REST Api, I used the before mentioned Volley library in Android Studio.



Picture 7: Milling Machine working



Picture 6: Example of outcome of the milling machine

Orange Pi



In order to be able to show the company how the web was looking, and get feedback from them as we were developing the project, we used a small single-board

computer similar to the Raspberry Pi. Orange Pi is a cheaper solution than the Raspberry Pi with similar specifications, specifically we utilized an Orange Pi One (H3 Quad-core Cortex A7 for CPU, and 512 MB DDR3 of memory SDRAM). It is able to run Android 4.4, Ubuntu, Debian, Raspbian Image, as well as the banana pi Image.

We installed LAMP (**L**inux-**A**pache-**M**ySQL-**P**ython) model of web service stacks to get our server running. As the web page was a Flask application, we had to install and configure the `mod_wsgi` to work with `apache2`. Furthermore, we had to enable the option to pass the “Authorization” headers to `wsgi`, in the `apache2` configuration. Then Flask was able to process the HTTP petitions that was using a JWT token to access our REST Api.

Even though we weren’t using any important password while developing the project, we still wanted to have a secure and encrypted connection with our server. Therefore, we needed an SSL certificate. We reviewed a few options with the company and finally decided that, at least while we were developing the project, they wanted to save as much money as possible. Let’s Encrypt is a free, automated, and open certificate authority that offers SSL certificates. The best feature of Let’s Encrypt is that they have an automated tool that will renew the certificate when it is about to expire. After installing and adding the certificate to the `apache2` configuration, we wanted to have a domain so that the company could always enter the same address in the web browser, and check the status of the project whenever they wanted.

The same idea applied, not spending money when possible (while developing). They understand that this domain is only for a short period of time, that when we deploy the web page, they will need to buy a domain related to their company name and activity. NoIp is a company that offers free domains with an expiration date of 30 days, if you don’t confirm the domain every 30 days, they will mark your domain as inactive and thus removing it from their system. They also offer a Dynamic Dns Domain application, that every time the server reboots or restart, it updates the ip address of the server to their DNS.

Because it's a free domain, it has some restrictions. We couldn't send emails with that domain, so we didn't install POSTFIX as it wouldn't be useful anyway while developing.

We decided to use another Orange Pi One in the project. With this one, apart from controlling the milling machine working as a Repetier-Server, we wanted to keep the working place free of extra cables and screens near the milling machine. Leaving for the company workers as much space to work with their tools not worrying about hitting anything or unplugging any cable.



Picture 8: Set up of Orange Pi for Milling Machine



Picture 9: Work space gained with the Orange Pi



Picture 10: Set up before Orange Pi

Work-flow / Methodology of work

Even though I have been the only one coding the project, as we decided to have a server running as something we could call pre-production, we needed to work in an organized way. The pre-production server have been really useful all along the project. Whenever I finished an important functionality or made a big change in the style and design, I always sent a message to the company asking them to check the web page. By doing this, we ended with a project that suits the company and is really customized to their taste and necessities.

To keep all possible changes of the project in a clean way, I have used Git as “Version Control”. Then as we had a pre-production server I have been working with 2 environments, using Git as the link between both.

GitHub



GitHub is a web-based hosting service for version control using Git. If you only want an open repository it's free, so that fitted our mentality of saving money when possible. As it is web-based it also was a way of having a backup of the project in case anything went wrong. GitHub also has an Issues management system. This came in handy when defining the tasks for each objective, because I could have a single place with all the To-Do tasks of the project.

Two environments

We wanted to have the server always running to get feedback from the company. As the server needed to be running, we couldn't be developing the project directly into the server. That would translate in the server having functionalities not working properly, or simply with the server not running, due to errors or bugs in the code.

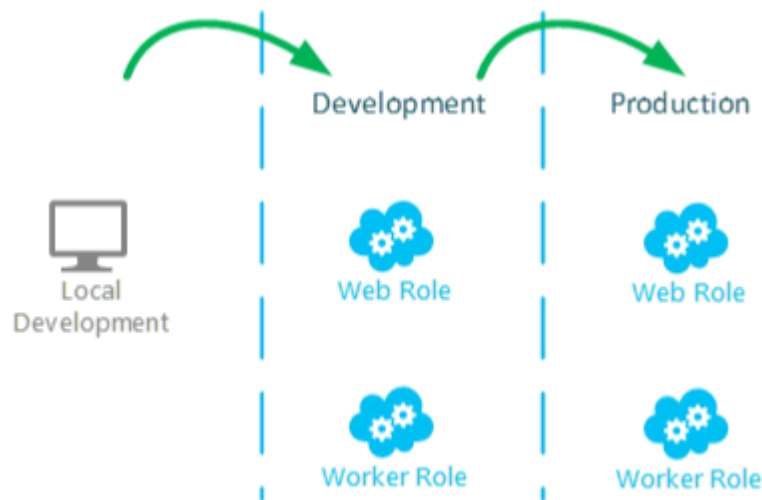
So using two environments seemed a reliable option to have the server running as much as possible. The first environment is the developing workstation, my computer. The second environment is the pre-production server, the Orange Pi One that we prepared previously and is serving the web page all day.

The development environment, had all the tools explained before and a few more. For example I also installed “MySQL WorkBench”, an application that allowed me to see and make operations in the database more visually. The IDE used to code the web was PyCharm, a nice feature of PyCharm is that it's capable of running the web page locally in the development computer, avoiding touching the pre-production server while developing.

The pre-production environment, had SSH installed to allow a remote shell previous authentication. When I finished a task or an issue from GitHub, I would use GitHub instead of FTP to bring the new files or changes to the pre-production environment. Thus not opening more ports than needed in the server. First, from the development

computer I would add the files and commit them to Git locally, then push them to GitHub. Second, I would login to the server with SSH and pull the changes that have been made in the GitHub repository to the server. Lastly, I would restart the apache2 service to make the changes effective, if I considered it was an important change I would ask the company to review the changes.

All in all, this have been our work-flow since the beginning of the project.



Picture 11: Example of multiple environments work-flow

Software developed / Implementation of the objectives

All the tools and resources have been used applying the methodology of work explained before to produce the project as it is today. We can say that all the specific objectives, with exception of the “Interface with the milling machine”, fall somehow under the development of the web server. As it is the biggest part of the project, we will explain some of the most interesting details of the implementation of it.

We can differentiate two sides of the server, front-end and back-end.

Front end

The front-end is the visible part of the web server, the page that the visitors will access with their web browsers. It also includes the private side of the web that is restricted for administrators only. As most of the web is quite standard, we are not going to focus on explaining the basics of any common web page.

As in the company, there also work people not so familiarized with the technology, we wanted to design the admin-side of the web with a similar style of the public side, thus being consistent.

Having said that, I consider that the 3D editor is a unique and distinctive feature that most of other web pages doesn't offer.

As said, we think that this two features are the ones that we it's worth focusing and explaining more detailed.

Admin-side design with JS

Taking into account that we wanted to keep the main style and design from the public side, instead of creating new templates that would like almost identical as the originals, we thought about modifying the existing templates with JavaScript in case that the visitor is authenticated as an administrator.

Then for each page of the public side, we created a JavaScript file adding some elements. For example in the "Guitarras" page, this is the JavaScript file:


```

var tarjeta = document.getElementsByClassName('card-title');

var nombre = document.getElementsByClassName('card-title');

for (i = 0; i < tarjeta.length; i++) {
    var nuevalinea=document.createElement('br');
    tarjeta[i].appendChild(nuevalinea);
    //Añadido enlace a borrar guitarra
    var aTag3 = document.createElement('a');
    //Cargar clase del enlace
    aTag3.setAttribute('class','enlaces');
    aTag3.innerHTML = 'Borrar Guitarra ';
    aTag3.setAttribute('href','borrarguitarra/'+nombre[i].childNodes[0].textContent);
    aTag3.setAttribute('data-toggle','confirmation');
    tarjeta[i].appendChild(aTag3);
}

```



Picture 12: View for non administrators



Picture 13: View for administrators

Then each new link of add a new item or edit an item, shows you a form to fill with the information. All the remove item links have a confirmation pop-up that was implemented with Popper JS.

By doing this, we ended up with a custom Content Management System (CMS) that will not require a lot of effort from the company to understand.

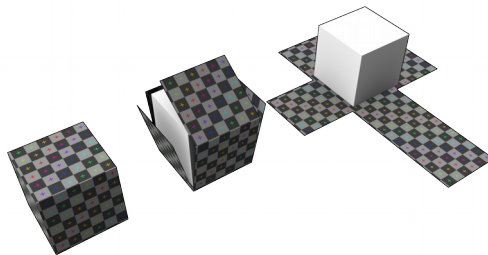
3D editor of custom guitars

One could understand the canvas of the editor page as a virtual universe where you can add objects, change the position from where you are looking, add lights. This is what we called a “scene”, once you have defined the scene with objects, lights and camera (where you are going to be looking from and how), you can render the scene in the canvas. The object is defined by the geometry of it’s shape, and by it’s material. The material has a set of properties, such as the type of the mesh, the colour, the

shininess, the texture, you can imagine it like a “dress”, that will wrap the object’s shape. This is all done with the Three.js.

```
var loadingScreen = {  
  scene: new THREE.Scene(),  
  camera: new THREE.PerspectiveCamera(90, 1280/720, 0.1, 100),  
  sphere: new THREE.Mesh(  
    new THREE.SphereGeometry( 0.15, 0.15, 0.15 ),  
    new THREE.MeshBasicMaterial({ color:0xbbbbbb })  
  )  
};
```

In order to have realistic objects that are accurate to the guitars that the company can offer, we had a multidisciplinary team performing different tasks. There was an industrial engineer (member of the company) in charge of creating the 3D objects in “SolidWorks”. A friend of the industrial engineer, had some knowledge in generating the textures for the objects. After some time the person in charge of the textures taught the industrial engineer the basics of generating textures with a program called “Blender”. Then he left. Blender will load the SolidWorks object, map all of its vertex and unwrap the vertex into a 2D image, that will become the texture later with some help of Photoshop.

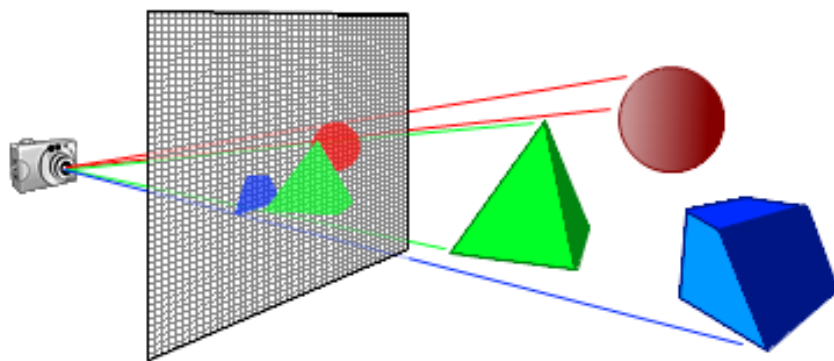


Picture 14: Texture unwrapping

At this point we already have a basic static scene of a guitar in 3D. We thought about giving the guitar a little bit of movement, as if it was in a rotating showcase. As we started to implement this we notice that as each component of the guitar was a different object in the scene, and they all started rotating around it’s own axis, making it look strange. Finally we added each component to a new object that would

be the whole guitar, that will stay hidden until the visitor will press a button. When the visitor click the button, we remove/hide the parts of the guitar and add/show the whole guitar as one object rotating as a single object.

However we still had a single guitar with no options to change. We needed to add the possibility of change some of its components, and we wanted to do it by interacting with the 3D scene. From that necessity, we came up with the idea of double-clicking the component you wanted to change. Ray-casting was the way to go to achieve this goal. We can simplify the concept of ray-casting as thin rope that has one end fixed in the camera position, and the other end goes through the position of the mouse and beyond.



Picture 15: Ray casting representation

Once the mouse is over an object, the ray also intersects with the object. In each object, we defined a callback function triggered when the visitor double-clicked it. This will display a pop-up menu showing the options available for that component.

Now we have to define which options are related to each object. To do that we created some tables in the database with some relations between them, that we will explain later in the Back-end section, for now let's say that for each component, we now which textures are available.

So when the visitor decides to change for example the body of the guitar, we show an empty pop-up. At the same time we are asking the database which options are available for the exact body the visitor double-clicked, and then inflate the pop-up

menu with the options. As all the interactions between the visitor and the server that doesn't refresh the page are handled with JavaScript, we used jQuery to ask for the options of each component, and displaying a loading progress circle.

jQuery comes with some predefined functions to make AJAX (Asynchronous JavaScript And XML) petitions.

```
function ajax1(){
    return $.ajax({
        url: urlBase+"opciones3D/"+pieza,
        dataType: "json", // Work with the response
        crossdomain: true,
        success: function (response) {
            opciones3D = response;
        },
        error: function (response) {
            console.log('ERROR');
        }
    });
}
```

Back end

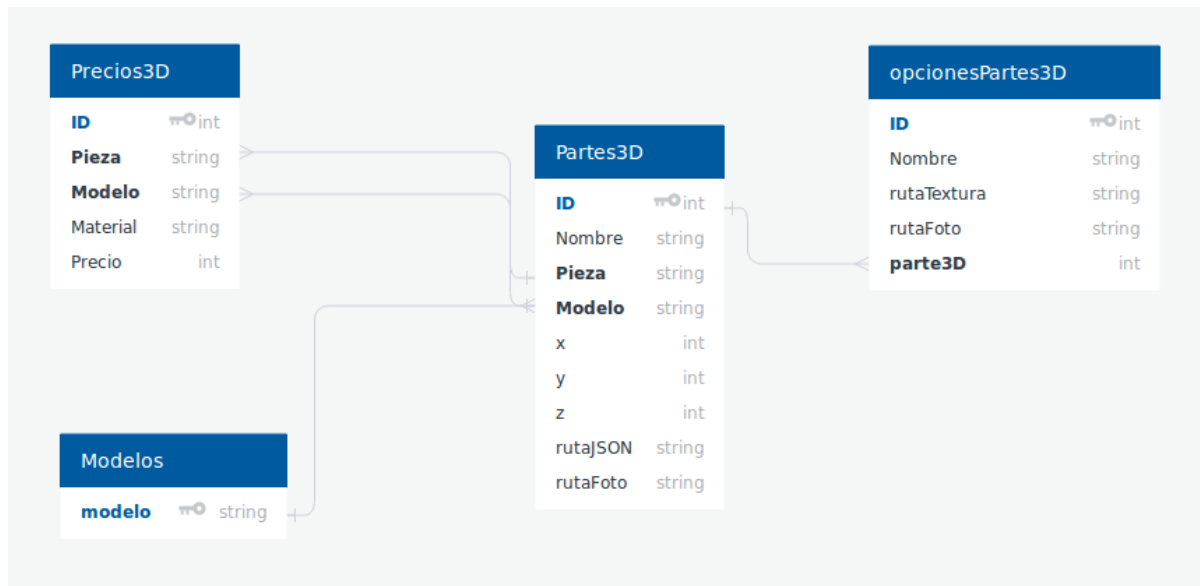
The back-end is the non-visible part of the web server, the interaction with the database, all the processes of authentication and authorization. It can also include some services that the server can be offering, such as a RESTful Api.

As most of the back-end is doing regular stuff of updating the database, we are going to explain the database structure and the more specific details about how we connected the database with the 3D editor and the android app.

Besides the database explanation, we also will explain the implementation of the IPN protocol, to show how we achieved to process the data from a transaction in PayPal.

Database

As we mentioned before in the front-end with the 3D editor, we were loading the options available for each component when double-clicked. We talked about some tables and it's relations. These are those tables in the database:



Picture 16: Schema of editor related tables

So for each model of guitar, there are parts of that model that form the whole guitar. Then for each part, there are options of textures for that same part. Besides, in order to estimate a global price, we have to know the material of some parts such as body, neck, and fretboard.

Now that we know how the tables for the 3D editor are defined and related, we know that the visitor's browser make an AJAX petition to our database. Actually, it doesn't request directly to the database, as the database is only accessible locally. The browser makes the petitions to our RESTful Api. The RESTful Api, is an API that access some resources that have been defined, and returns different information of the resources for different endpoints.

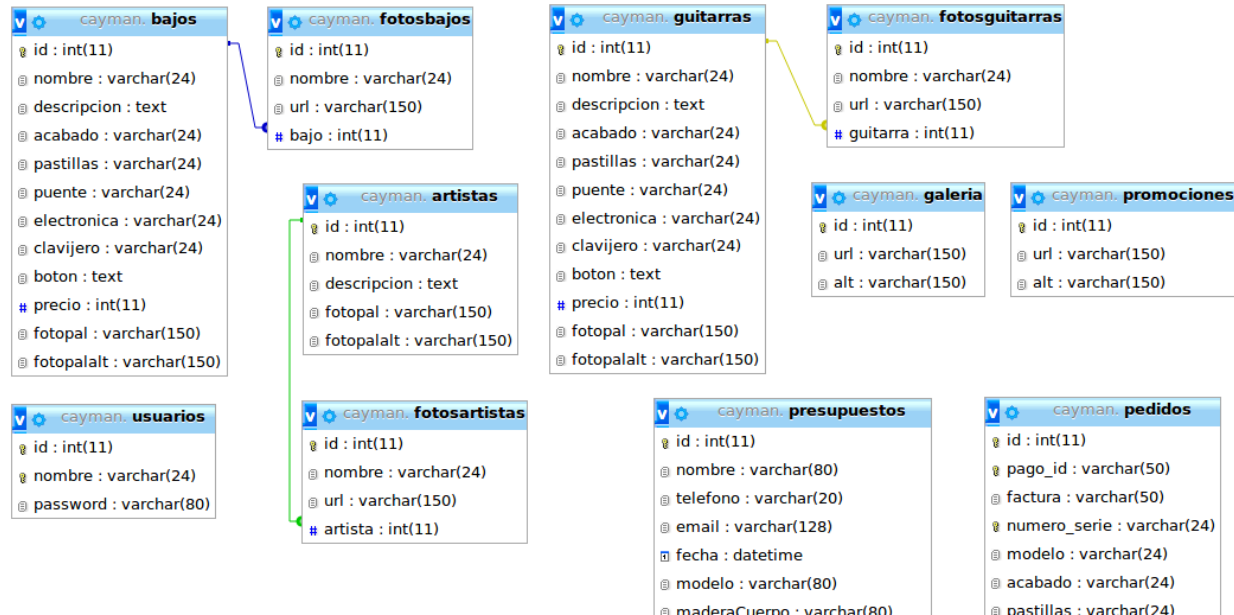
We also knew that if we wanted to access the information of the database from an Android app we needed a RESTful Api. Other solution was to put our database in the internet, and put in the code of the Android app the credentials of the database. That would allowed evil hackers from the internet try to hack the database, and maybe try to make a Denial Of Service (DoS) attack. Furthermore, if someone gets

the apk of the app and de-compile it, they would be able to know the database credentials. For that reasons, developing a RESTful Api was the best option.

We didn't need to secure our Api as long as it didn't return any sensitive data, such as the information for the 3D editor, or the products available in the web page. However, one of the objectives that the Android app wanted to cover, was to be able to CRUD (Create Read Update Delete) the order management system. Then the security was a necessity.

JWT tokens in addition to an SSL certificate, was the security we implanted to our API. With the help of the plug-in Flask-JWT we could add the JWT tokens authentication and authorization in our API.

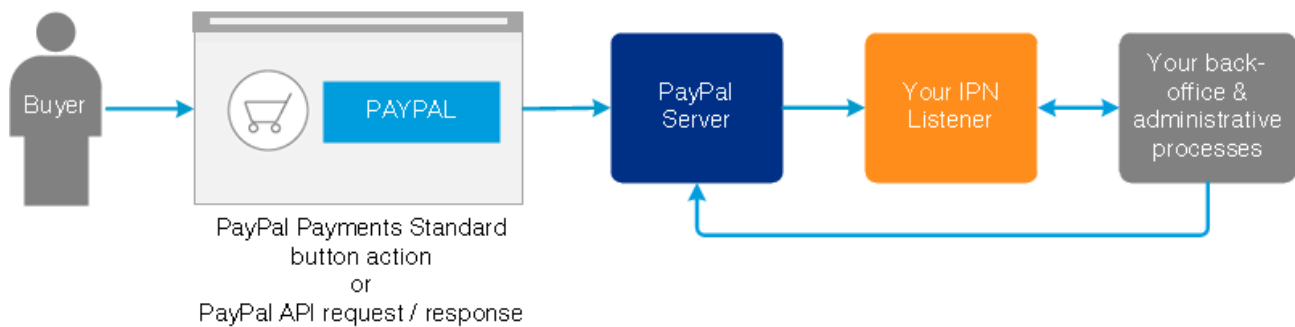
An important tool that was really useful while developing the MySQL database, was SQLAlchemy, an ORM. We created and defined the models for each table that we will later be using when accessing the database with objects in python. This is the rest of the schema of the database:



Picture 17: Schema of the database

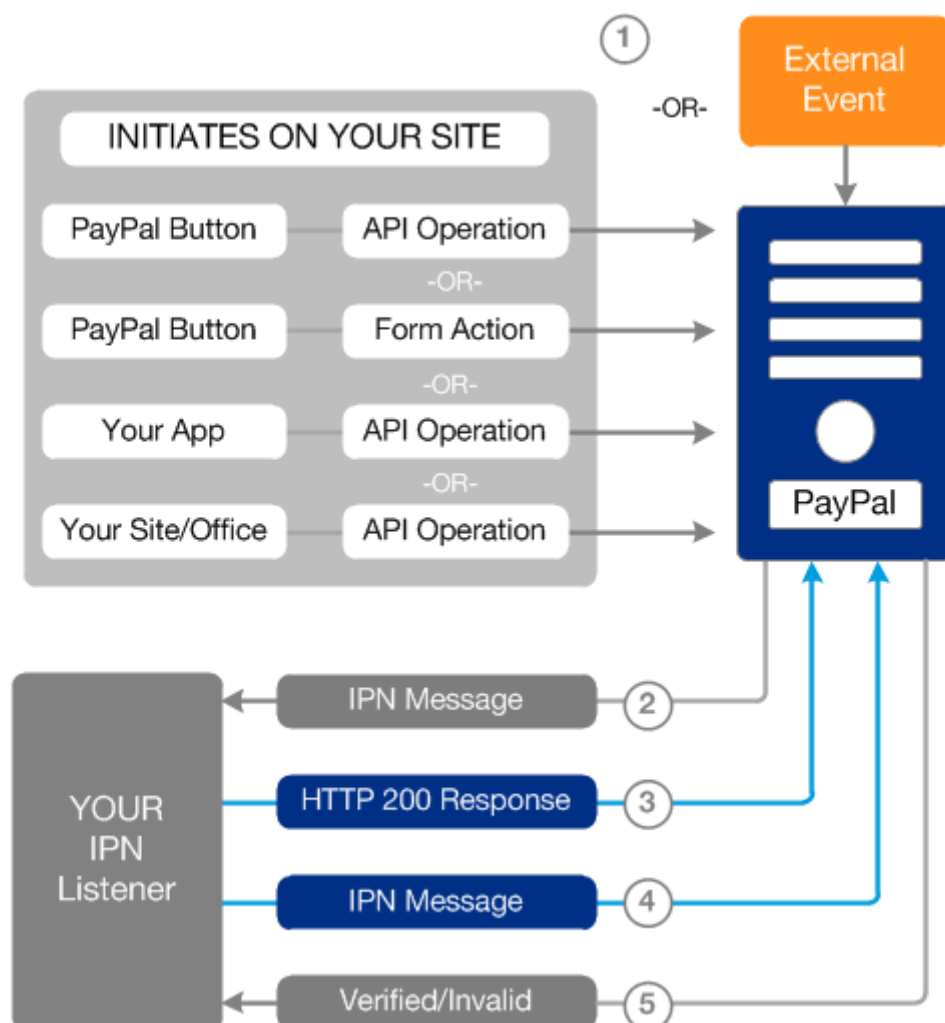
IPN protocol implementation

As we saw in “Tools and Resources”, the IPN protocol flow is like this:



Picture 18: IPN protocol Overview

We can see that there are two main agents involved in this protocol, PayPal server and our IPN listener. The agent that we have to implement is our IPN listener. After having seen the overview of the IPN flow, we can look for the more specific details of the protocol. If we remember from before, the protocol is:



Picture 19: IPN protocol flow

Our listener needs to be able to receive the IPN message from PayPal (2). To do that, we have to login into our PayPal Business account, and set the URL for our IPN listener.

Once that is done, we need to give PayPal a response with code 200 for the IPN message received (3). We can accomplish that by having defined the route for our IPN listener, if the route exists and works, it will send a code 200.

```
@paypal_ipn.route("/paypal_ipn", methods=['POST'])
def paypal_ipn2():
```

The 4th step is to send the same IPN message PayPal send us before, back to PayPal.

To do that we store the arguments of the post petition received in a “ImmutableOrderedMultiDict”, this way we ensure that we won’t change the order of the arguments received. After loading the arguments with the proper encoding, we send it back to the PayPal URL that will validate our message.

```
arg = ""
request.parameter_storage_class = ImmutableOrderedMultiDict
values = request.form

headers = {'content-type': 'application/x-www-form-urlencoded', 'user-agent': 'Python-IPN-Verification-Script'}
for x, y in values.items():
    aux = {x: y}
    arg += '&' + urllib.parse.urlencode(aux)
validate_url = 'https://ipnpb.sandbox.paypal.com/cgi-bin/webscr?cmd=_notify-validate{arg}'.format(arg=arg)
r = requests.post(validate_url, headers=headers)
```

Then we have to check whether the response of the response from PayPal (5) is “VERIFIED” or “INVALID”. If it checks “VERIFIED” then we process the data received and do as needed, for example save the name, phone number, price paid, and more in the database:

```
if r.text == 'VERIFIED':  
    app.logger.warning("VERIFIED")  
    ##ToDo Process the data in the arguments  
    ##Save into db the order  
else:  
    app.logger.warning("INVALID")
```

The company thought about automatically generating the serial number of the products. They already had a system for this.

Example of Serial Number: “0418001”.

04 (Aprl)/ 18 (Year)/ 001 (Number of guitar made that month)

So we implemented that into our IPN listener, when adding orders to the db, as well as when introducing manually an order, either through the admin-side of the web or the Android app. So they didn’t have to change their method of work.

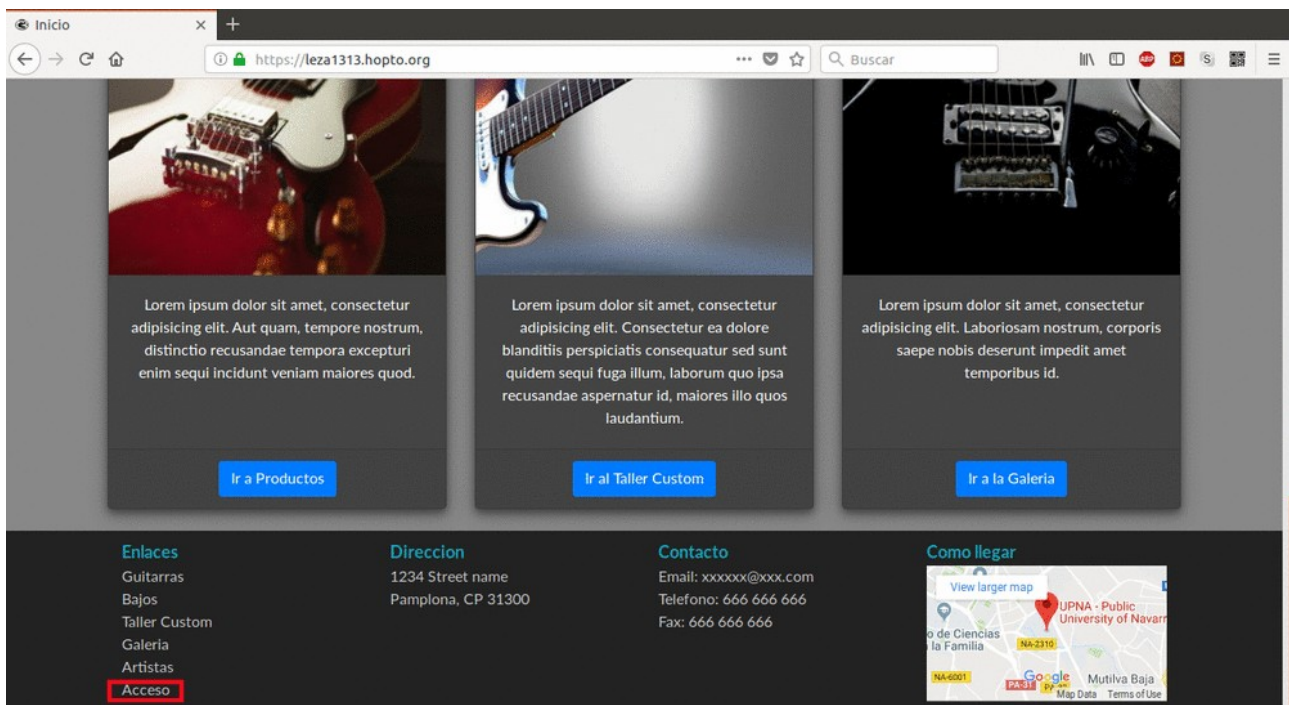
User manual

In this section we are going to see how to execute the tasks the web is designed for. As there is two type of users in the web page that can perform different activities, we are going to see both of them.

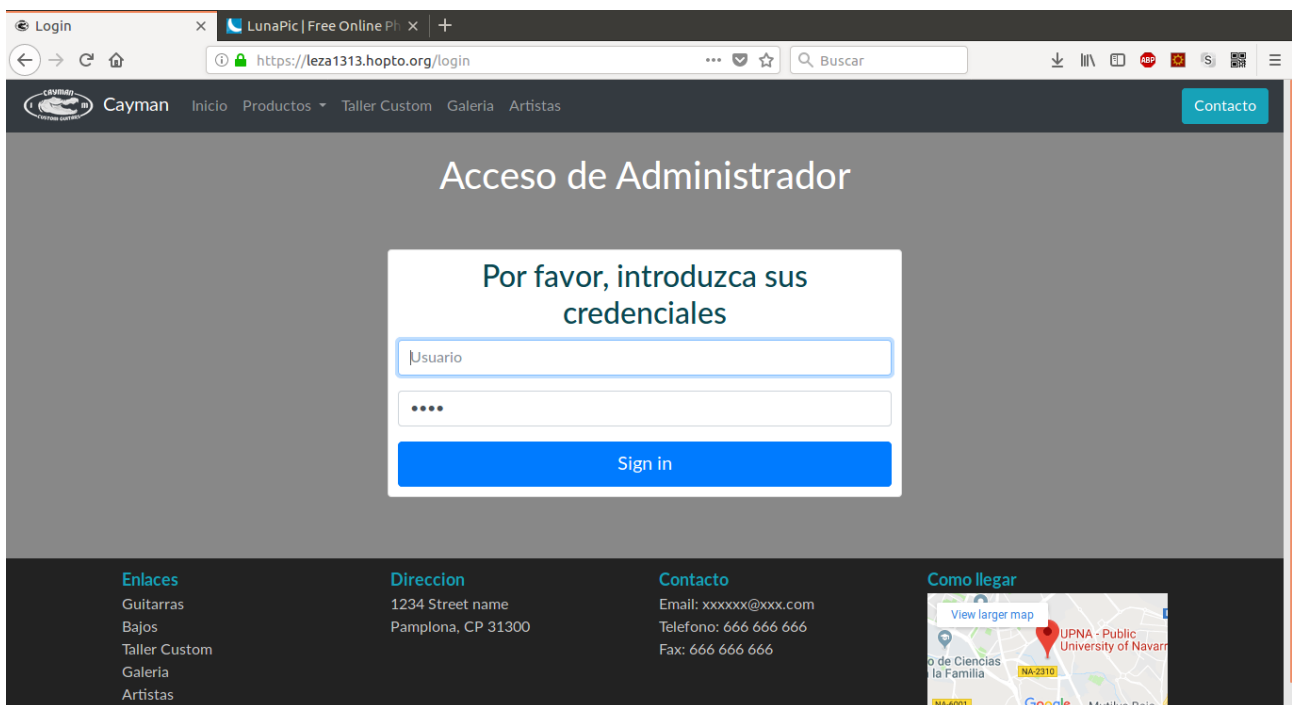
Administration manual

The administration manual is focused on updating and managing the resources the web is displaying.

First of all, you need to be logged in, in order to see the admin actions available. To do that you should head to the footer of the page and click “Acceso”



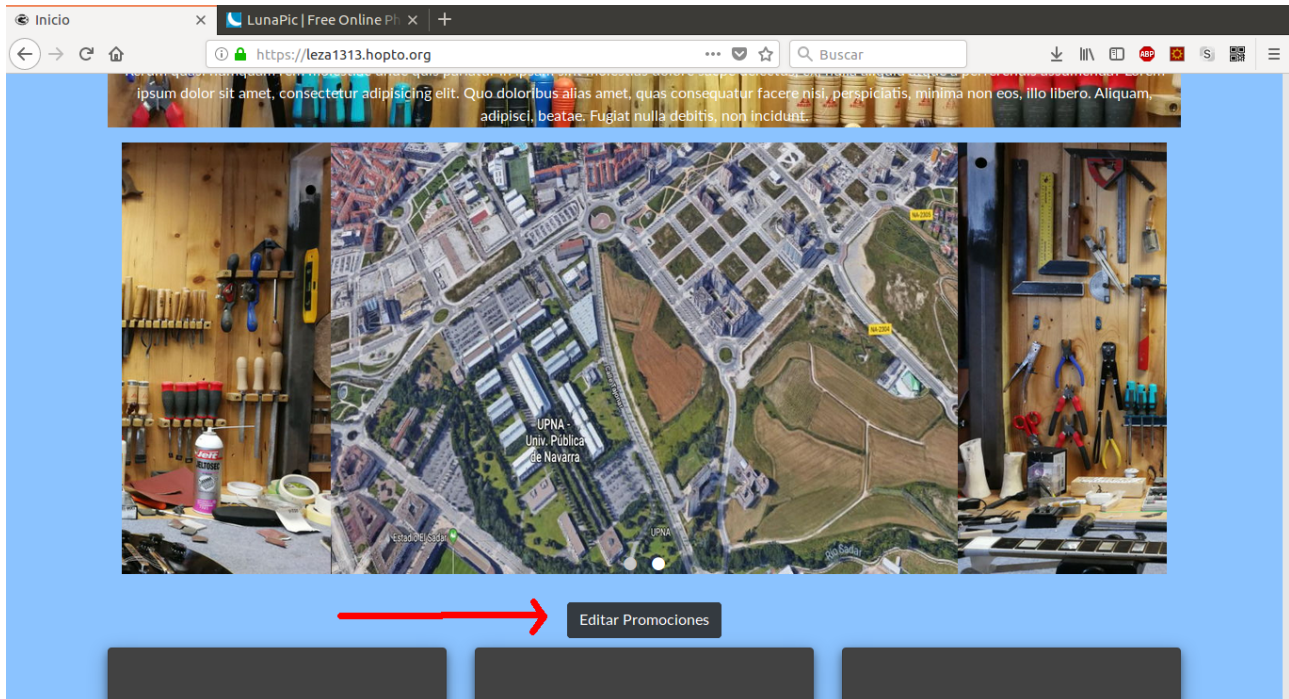
Picture 20: Link to the login page



Picture 21: Login page

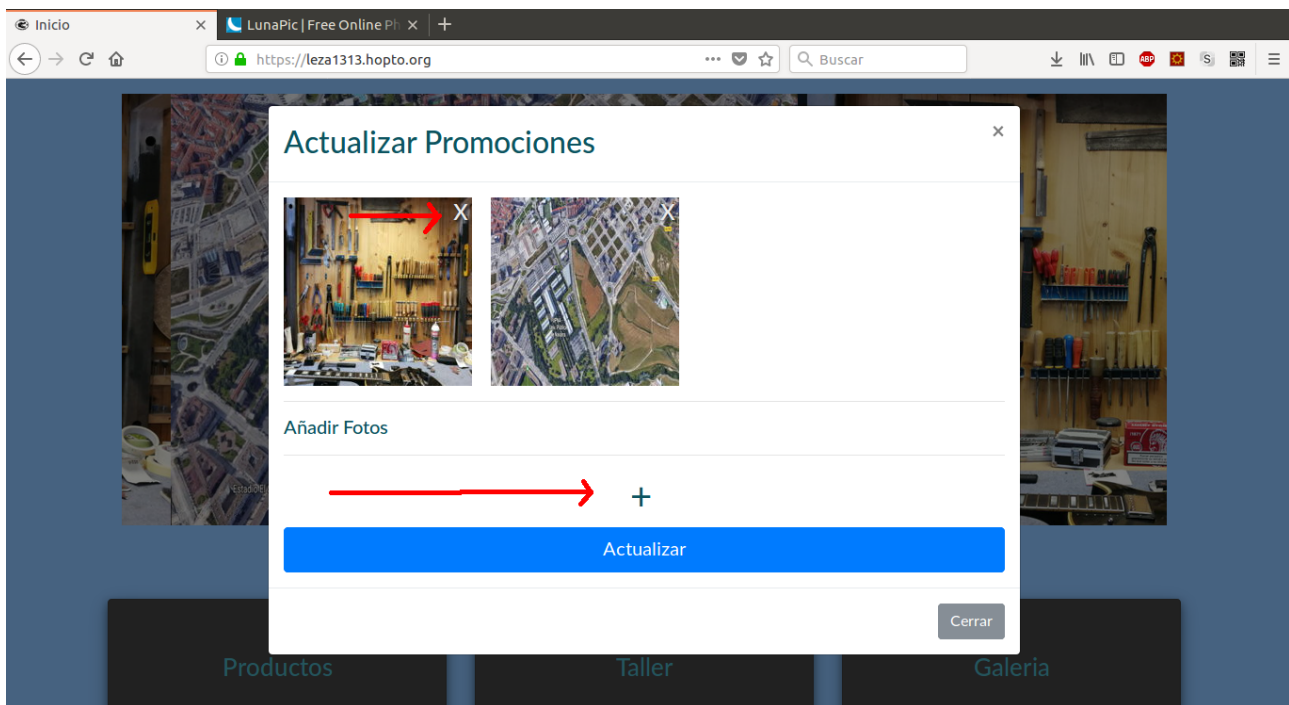
- Change the images of the promotions slider

Once you are logged in, in the homepage, below the slider there is a button that allows you to change the images of it.



Picture 22: Edit Promotions slider

There is two actions to do here. The cross (X) in the top right corner of each image deletes the image. It ask you for confirmation to delete the image. The plus (+) add a field to upload an image.

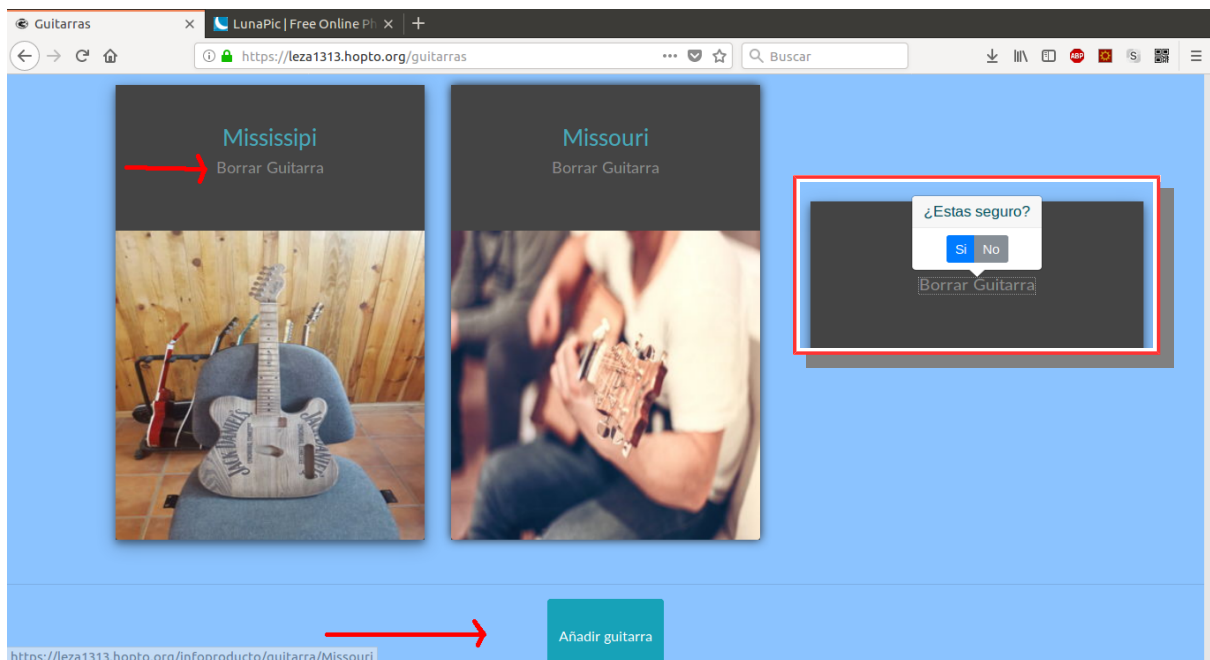


Picture 23: Modal with form to update promotions slider

- Add, Update and Delete Guitars, Bases, and Artists

As guitars, bases and artists are displayed similarly, they also update, delete and add in the same way.

You navigate through the page for the section you want to maintain. Then you see the cards of the items available. In each card, there is a button saying “Borrar <item>” that will show a pop-up to delete the item. In the bottom of the page, after all the cards, there is a button to add new items.



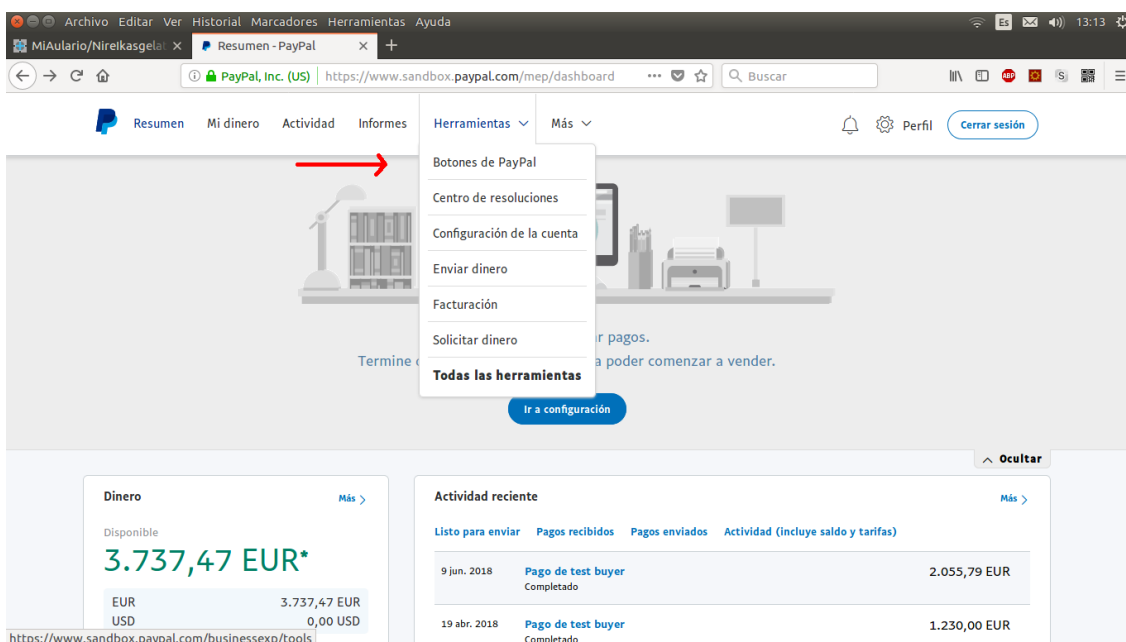
Picture 24: Admin view of guitars

The “Añadir <item>” button takes you to another page with a form. The fields of the form will vary depending of the type of item, but will maintain some structure as the

Picture 25: Form to add a guitar uploading images section.

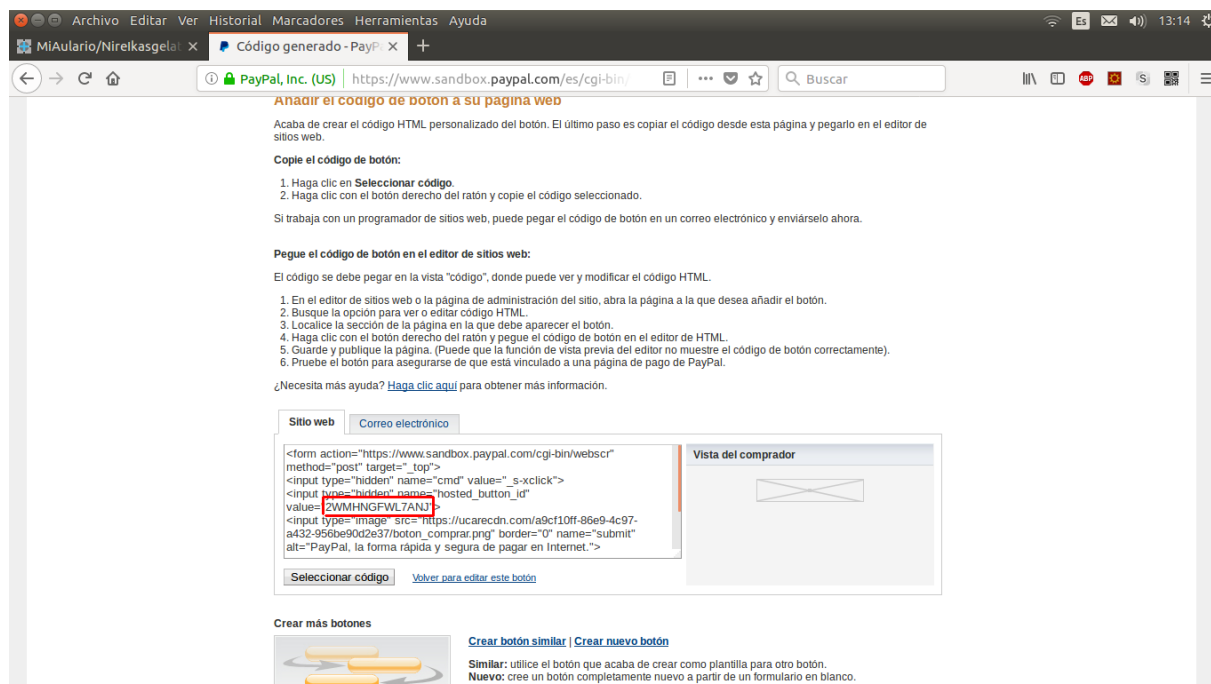
The field of “Boton” is the field where you should paste the code of the PayPal button, in order to sell a product.

To create a PayPal button, you enter in their web page with your account. You need to go to “Tools” and then PayPal buttons.



Picture 26: PayPal Dashboard

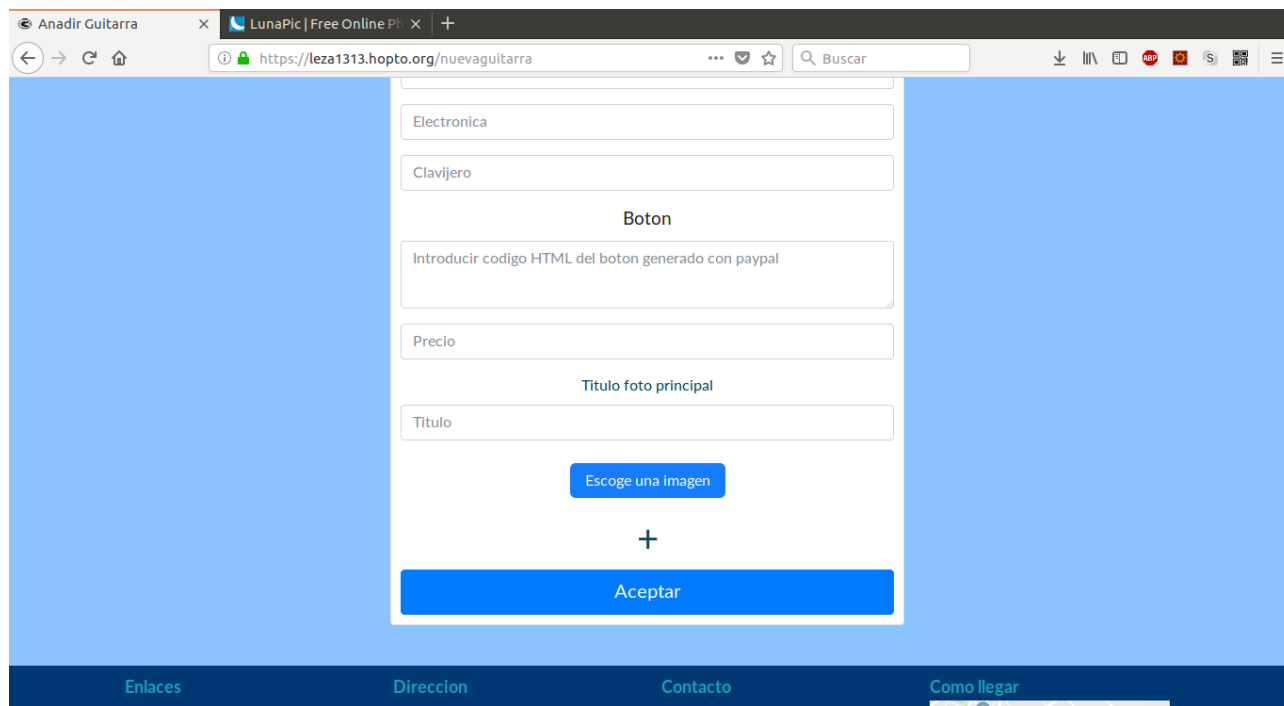
There you will fill the price and name of the product you are selling and finally get to this page.



Picture 27: PayPal button code

Where you will see the code for the button.

Once you pasted the code back in the form of adding a new item, you need to upload some images of the product you want to sell or some pictures of the artist sponsoring the company. It's mandatory to upload at least the principal image.

A screenshot of a web browser showing a form titled "Anadir Guitarra" (Add Guitar). The form is set against a light blue background. It contains several input fields: "Electronica" (Electronics), "Clavijero" (Pickup), "Boton" (Button), "Introducir codigo HTML del boton generado con paypal" (Enter HTML code of the button generated with paypal), "Precio" (Price), "Titulo foto principal" (Main photo title), and "Titulo" (Title). Below these fields is a blue button labeled "Escoge una imagen" (Choose an image). A plus sign (+) is centered below the button. At the bottom of the form is a large blue button labeled "Aceptar" (Accept). The browser's address bar shows the URL "https://leza1313.hopto.org/nuevaguitarra". The browser's top bar shows the tabs "Anadir Guitarra" and "LunaPic | Free Online P...". The browser's bottom bar shows the links "Enlaces", "Direccion", "Contacto", and "Como llegar".

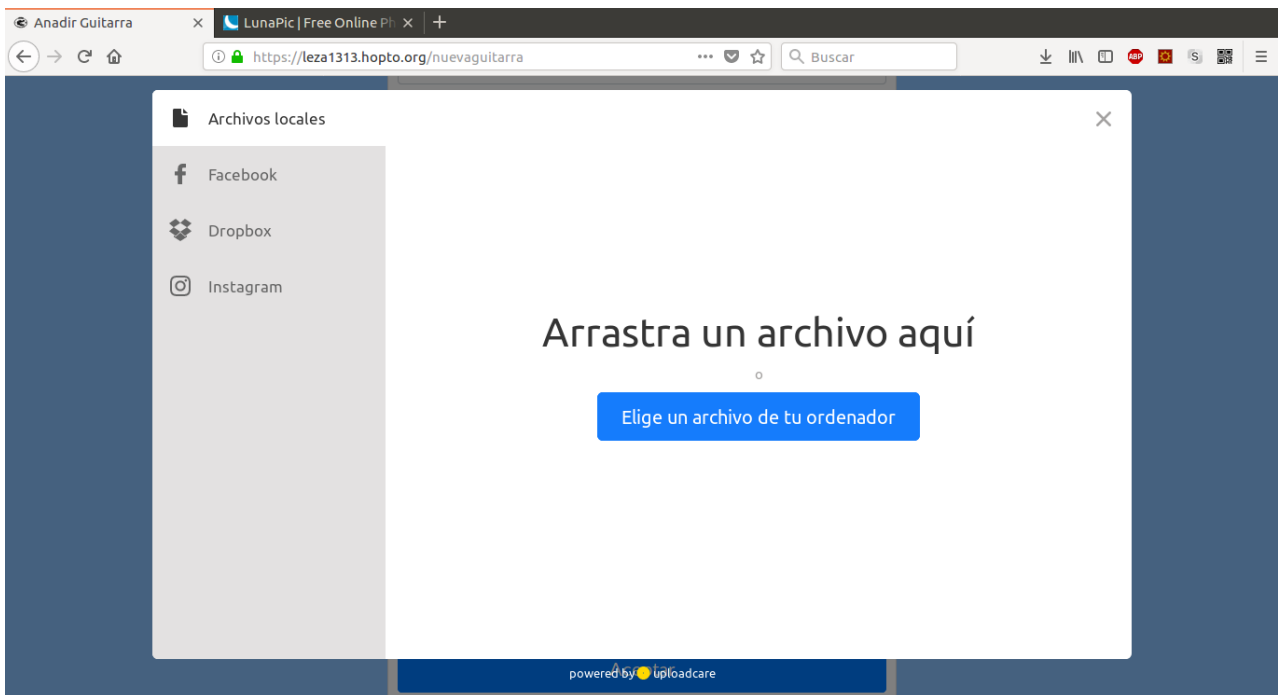
Picture 28: Upload images to new guitar form

If you click the plus (+) it will appear new fields of uploading images. You can add as many images as you want.

A screenshot of a dynamic form for adding more images. It features a title "Titulo para foto n°1:" (Title for photo n°1:). Below the title is a text input field with the placeholder "Titulo". Below the input field is a blue button labeled "Escoge una imagen" (Choose an image). A plus sign (+) is centered below the button.

Picture 29: Dynamicly adding fields to upload more images

When clicking the button “Escoge una imagen”, a big pop-up menu will appear.

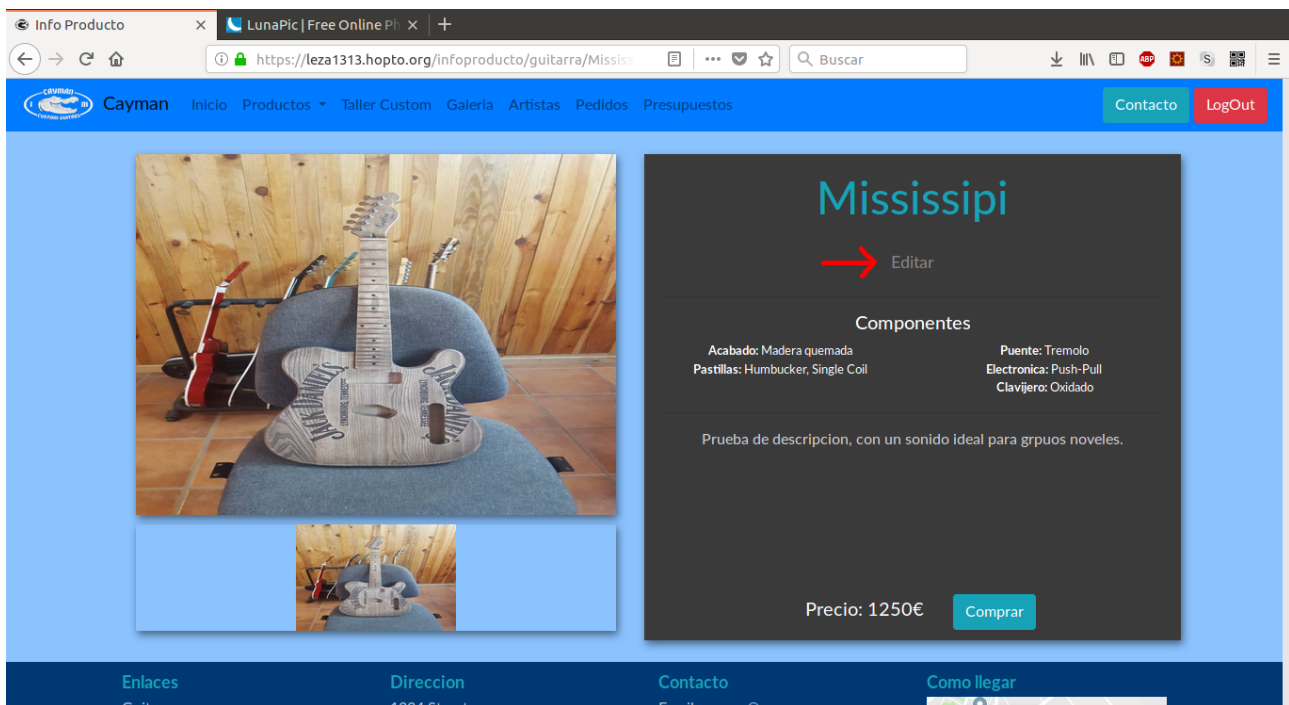


Picture 30: Widget from UploadCare

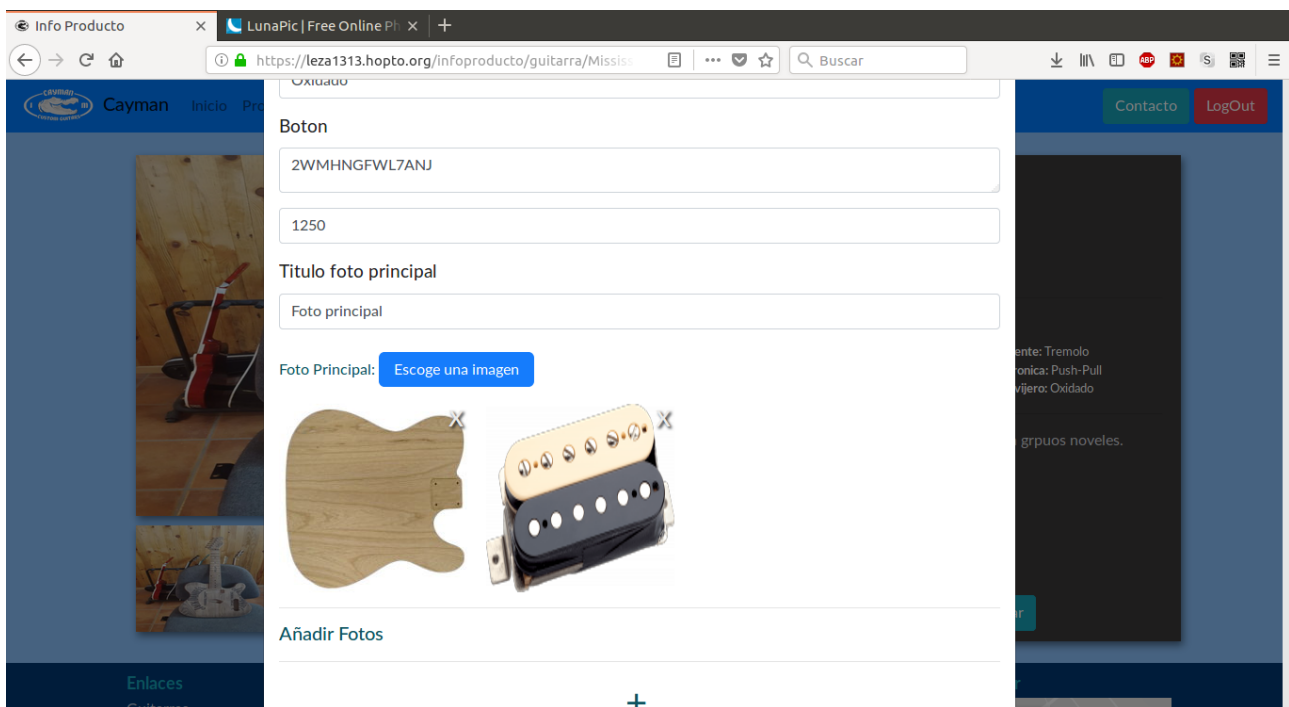
You will select the image from your computer by default, or you can upload images from other 3rd parties.

Finally the last button will submit the form and thus saving the item in the database, and appearing from now on in the page.

To update the item, you go to the item description page. Then you have a button “Editar” right under the name of the item. It will show a menu with the information and images of the item.



Picture 31: Admin view to update guitar

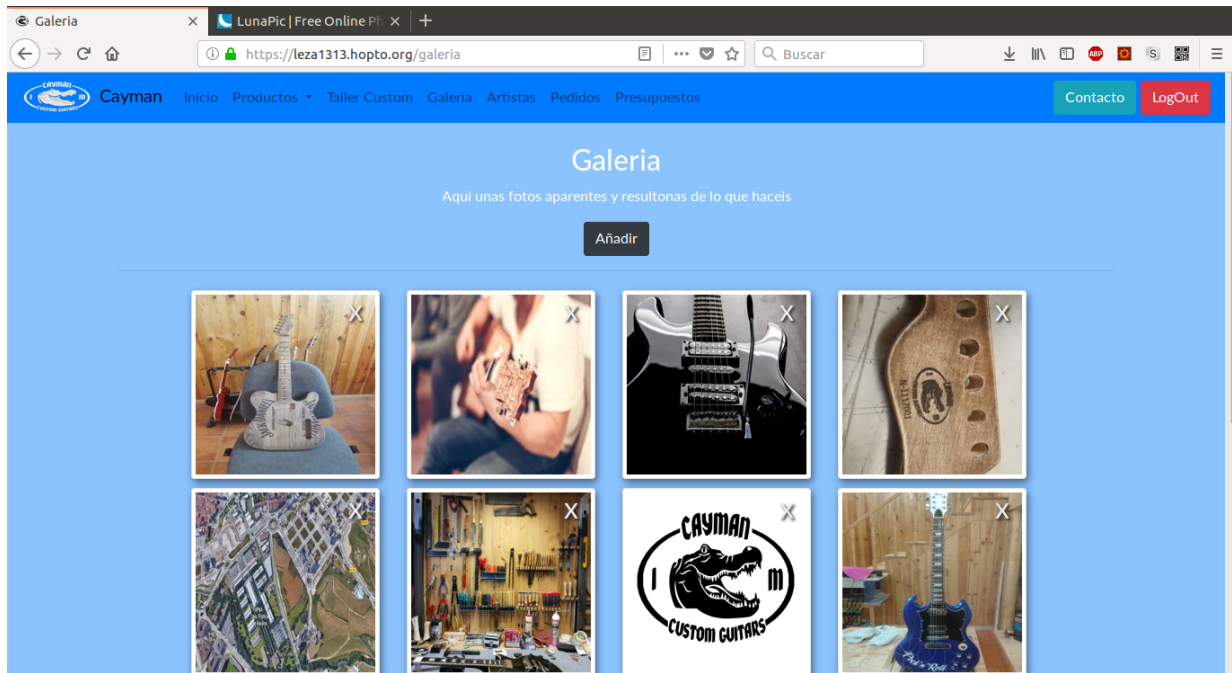


Picture 32: Form to update guitar

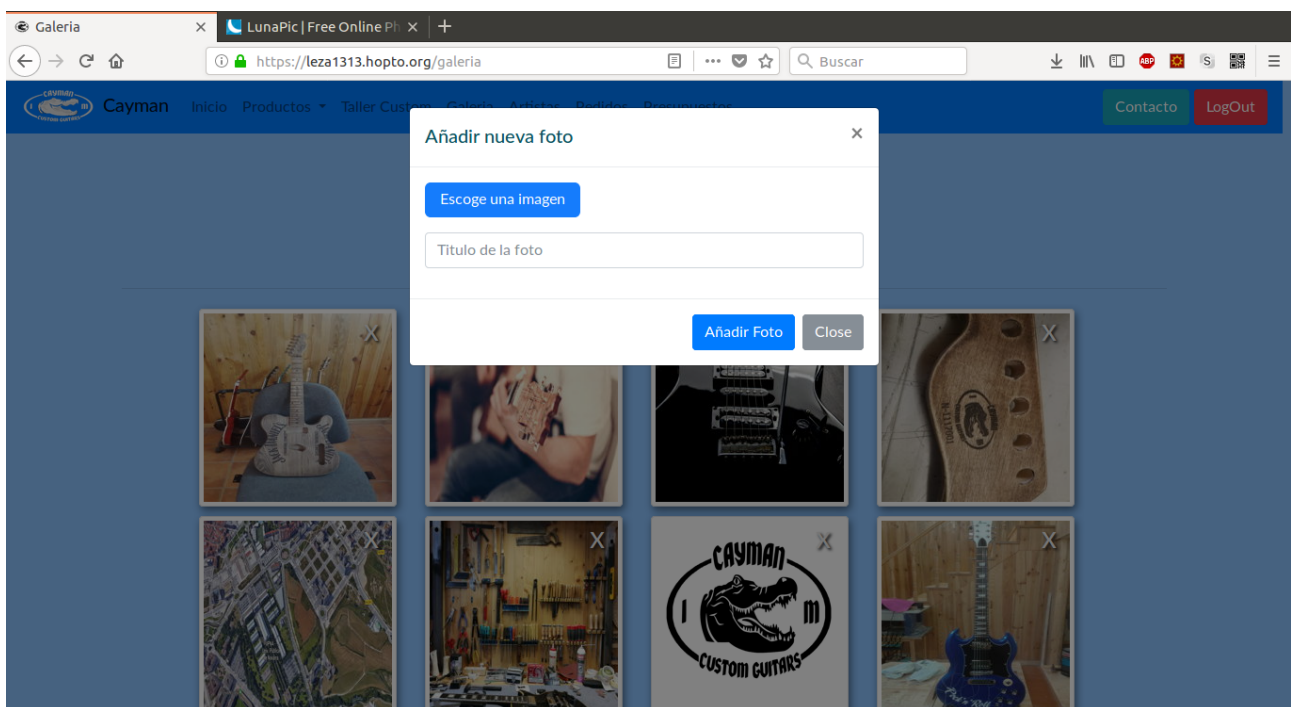
The same as before applies. The cross (X) to delete images, and the plus(+) to add more images.

- Add and delete images from the gallery

Basically it's the same concept of before, with the crosses to delete and a button with a form to upload.



Picture 33: Admin view of Gallery



Picture 34: Upload a new image to Gallery

- Orders management

Orders that have been completed through the web and PayPal will be added automatically, in order to add to the system orders received in-person there is a button to add them to the database.

Pedidos - Mozilla Firefox

MiAulario/Nirelaskgela: Pedidos

https://leza1313.hopto.org/pedidos

Cayman Inicio Productos Taller Custom Galeria Artistas Pedidos Presupuestos Contacto LogOut

Pedidos

Aqui los pedidos realizados

Nuevo pedido

Estado	FACTURA	NUMERO_SERIE	MODELO	ACABADO	PASTILLAS	PUENTE	ELECTRONICA	CLAVIJERO	NOMBRE	DIRECCION	TELEFONO	EMAIL	PRECIO	FECHA	Accion
En proceso	0000	0618004	Tennessee	Cerezo (Heritage Cherry)	Humbucker	Tuneomatic de aluminio	2 tonos y 2 volúmenes	Clavijas de afinación co	test buyer	calle de prueba, n 12, tafalla, 31300, NAVARRA, Spain	906-671-9463	leza1313-buyer@gmail.com	1149.19€	2018-06-10 14:12:33	✕ ✎
En proceso	0000	0618001	Missouri	Lacado sunburst	Single Coil	Standard	Push-Pull1	Cromado Negro	test buyer	calle de prueba, n 12, tafalla, 31300, NAVARRA, Spain	906-671-9463	leza1313-buyer@gmail.com	1640.88€	2018-06-10 13:55:33	✕ ✎
En proceso		0418001	Mississippi	Lacado sunburst	Humbucker	Tremolo	Push-Pull	Cromado	David Leza	Calle Miranda, nº45	600600600	david@leza.com	1289€	2018-04-19 12:21:05	✕ ✎
Entregado	None	0418002	Mississippi	lacado	single coil	tipo1	tipo2	tipo3	Lucio Leza	calle prueba	666777999	lucio@leza.com	889€	2018-04-15 12:37:05	✕ ✎

Enlaces Direccion Contacto Como llegar

Picture 35: View of orders

The crosses (X) as usual, show a pop-up to confirm you want to delete the order. The pencil opens a form to update the order. When set the delivery date of the order, it will change the colour to green and also the status to finished.

Pedidos - Mozilla Firefox

https://leza1313.hopto.org/pedidos

Formulario de Actualización:

- Telefono: 906-671-9463
- Email: leza1313-buyer@gmail.com
- Precio: 1640.88
- Fecha: 2018-06-10 13:55:33
- Fecha Salida: Fecha de entrega o envío al comprador
- Observaciones: Campo para rellenar con observaciones

Tabla de Pedidos:

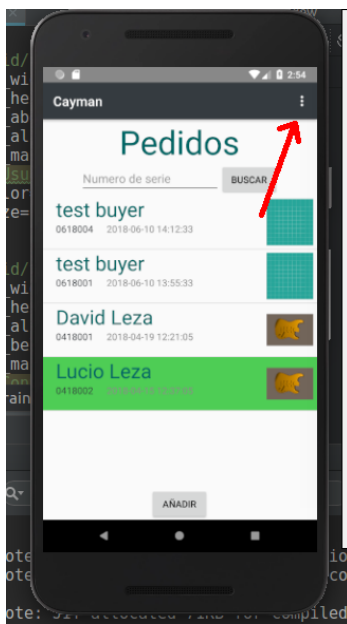
EMAIL	PRECIO	FECHA	Accion
leza1313-buyer@gmail.com	1149.19€	2018-06-10 14:12:33	[X] [Pencil]
leza1313-buyer@gmail.com	1640.88€	2018-06-10 13:55:33	[X] [Pencil]
leza1313-buyer@gmail.com	1289€	2018-04-19 12:21:05	[X] [Pencil]
leza1313-buyer@gmail.com	889€	2018-04-15 12:37:05	[X] [Pencil]

Botones: Contacto, LogOut, Actualizar

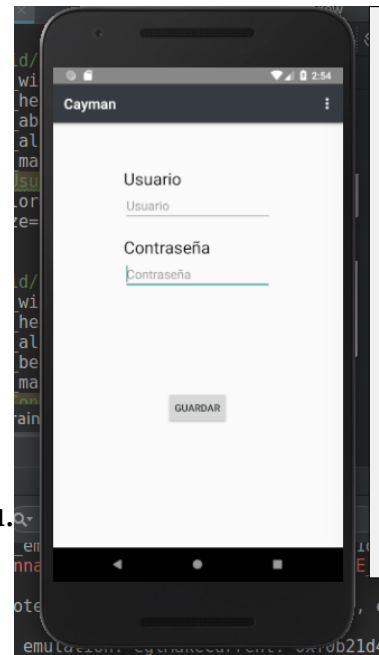
Picture 36: Update order form

You can also manage the orders with the android app, that can be downloaded from the route <https://<server.com>/android>.

Once installed you need to introduce your credentials to retrieve the information from the RESTful api.



Choose "Configuración" in the menu.



Picture 38: Credentials in the android app

Picture 37: Home window of the android app

This is the form to add new order



Picture 39: Add a new order in Android app

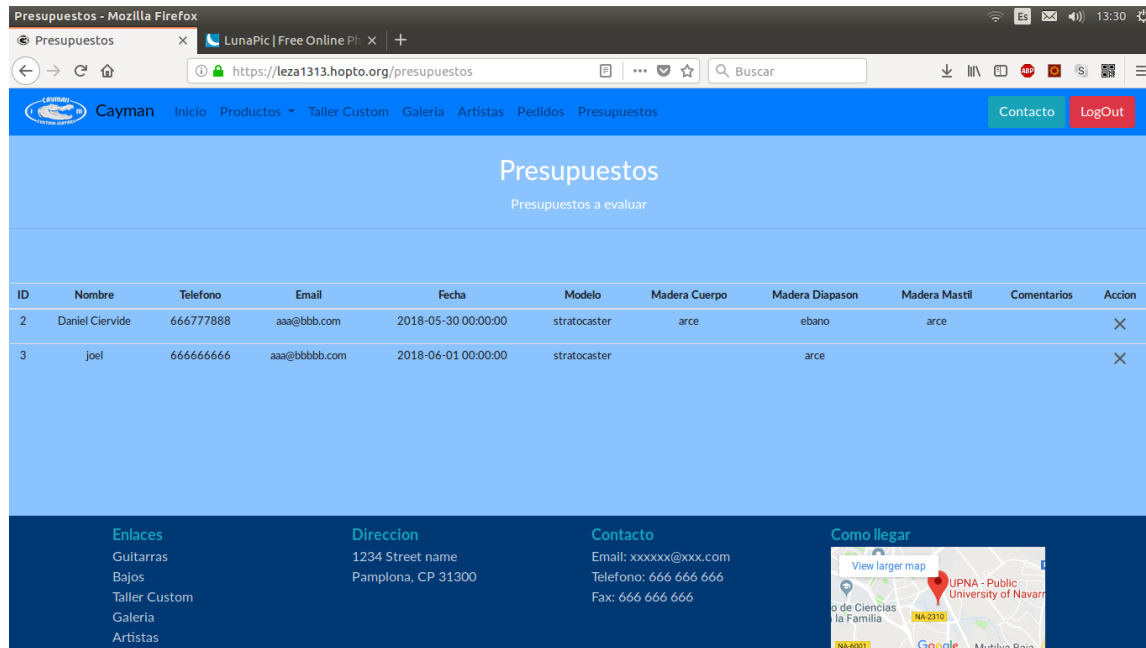
This is the window to see the information of the order



Picture 40: View info about order in Android app

The update window is as the form to add a new one with addition of the delivery date, and filled with the current information.

- Read and delete Budgets



ID	Nombre	Telefono	Email	Fecha	Modelo	Madera Cuerpo	Madera Diapason	Madera Mastil	Comentarios	Accion
2	Daniel Ciervide	666777888	aaa@bbb.com	2018-05-30 00:00:00	stratocaster	arce	ebano	arce		×
3	joel	666666666	aaa@bbbbb.com	2018-06-01 00:00:00	stratocaster		arce			×

Enlaces
Guitarras
Bajos
Taller Custom
Galeria
Artistas

Direccion
1234 Street name
Pamplona, CP 31300

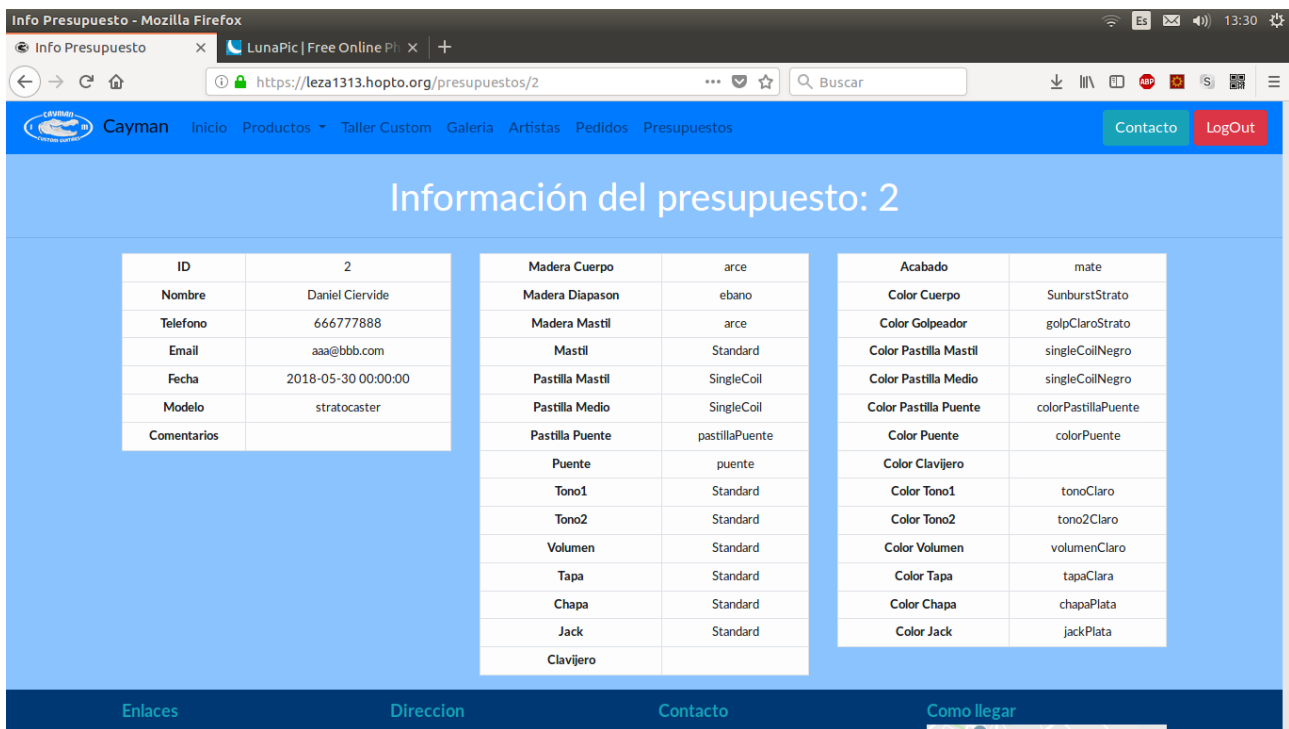
Contacto
Email: xxxxxx@xxx.com
Telefono: 666 666 666
Fax: 666 666 666

Como llegar
View larger map
UPNA - Public University of Navarra
Mutitva Baia

Picture 41: Budgets view

When a visitor build a 3D guitar, he can ask for a budget for that specific guitar and components. Those will appear in the “Presupuestos” window.

As always, the crosses (X) show a pop-up to confirm the removal. If you click the row of a budget, you will be taken to another page showing all the components selected.

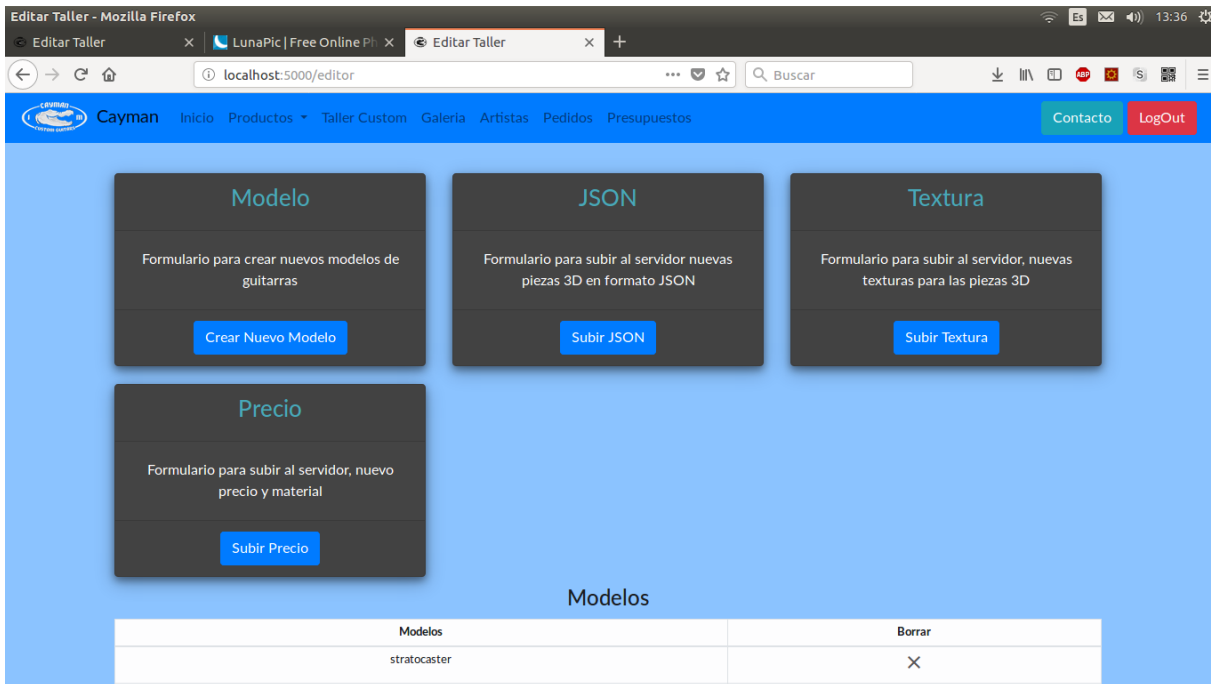


Picture 42: More info about a budget

- Maintenance of 3D editor

As the 3D editor relies heavily in the files made in SolidWorks and Blender, for it's maintenance is necessary to understand how the 3D editor works. Even though not everyone will have this knowledge, here you will see how to add some textures and more.

This is the page you will see.



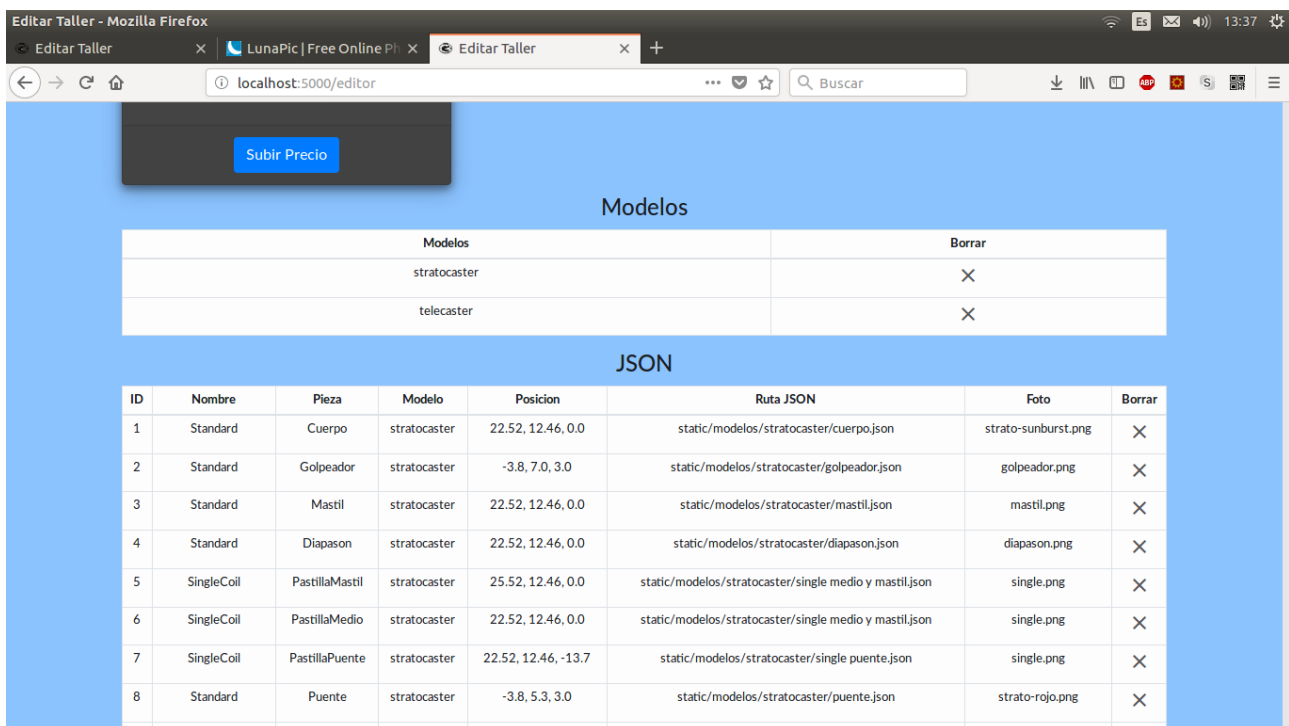
Picture 43: Admin view of the 3D editor

Each card will take you to a new page with a form to add either a new Model of guitar, a new part of the guitar in JSON file, a new Texture for a part, and the price for some parts and material or brands.

Picture 44: Form to upload a new part to the editor

For the upload of a JSON file, you need to specify the coordinates x, y and z. To know which are the coordinates, you can previously mount the guitar part with the rest of the guitar in this [editor](#).

Below the cards for adding items to the editor, there are 4 tables, one for each of the type of items that compound the editor. You can delete any item by clicking the cross (X).



Subir Precio

Modelos

Modelos	Borrar
stratocaster	X
telecaster	X

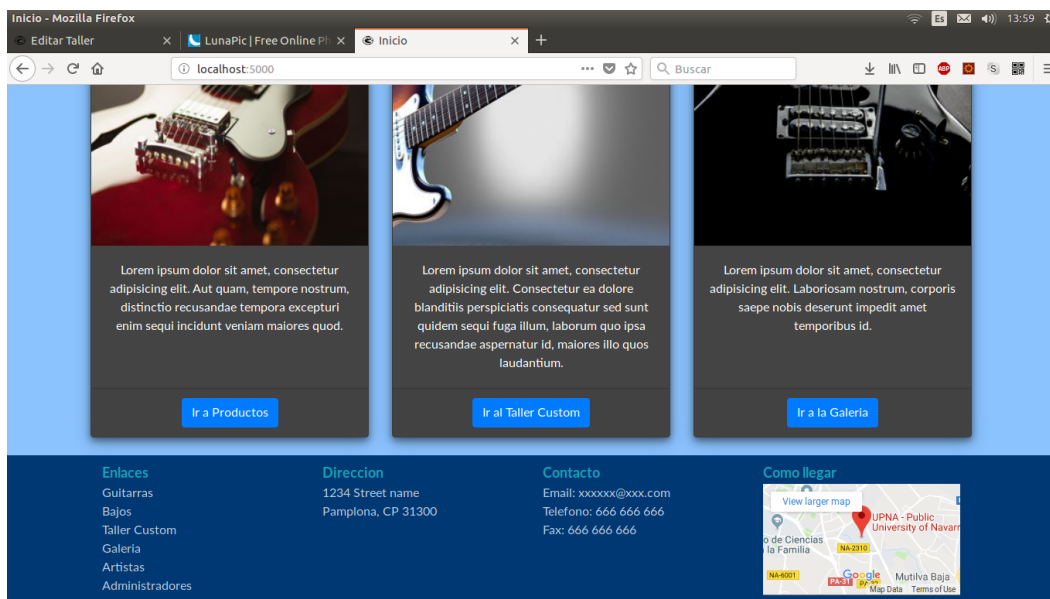
JSON

ID	Nombre	Pieza	Modelo	Posicion	Ruta JSON	Foto	Borrar
1	Standard	Cuerpo	stratocaster	22.52, 12.46, 0.0	static/modelos/stratocaster/cuerpo.json	strato-sunburst.png	X
2	Standard	Golpeador	stratocaster	-3.8, 7.0, 3.0	static/modelos/stratocaster/golpeador.json	golpeador.png	X
3	Standard	Mastil	stratocaster	22.52, 12.46, 0.0	static/modelos/stratocaster/mastil.json	mastil.png	X
4	Standard	Diapason	stratocaster	22.52, 12.46, 0.0	static/modelos/stratocaster/diapason.json	diapason.png	X
5	SingleCoil	PastillaMastil	stratocaster	25.52, 12.46, 0.0	static/modelos/stratocaster/single medio y mastil.json	single.png	X
6	SingleCoil	PastillaMedio	stratocaster	22.52, 12.46, 0.0	static/modelos/stratocaster/single medio y mastil.json	single.png	X
7	SingleCoil	PastillaPuente	stratocaster	22.52, 12.46, -13.7	static/modelos/stratocaster/single puente.json	single.png	X
8	Standard	Puente	stratocaster	-3.8, 5.3, 3.0	static/modelos/stratocaster/puente.json	strato-rojo.png	X

Picture 45: Tables with the items in the editor

- Adding and deleting administrators

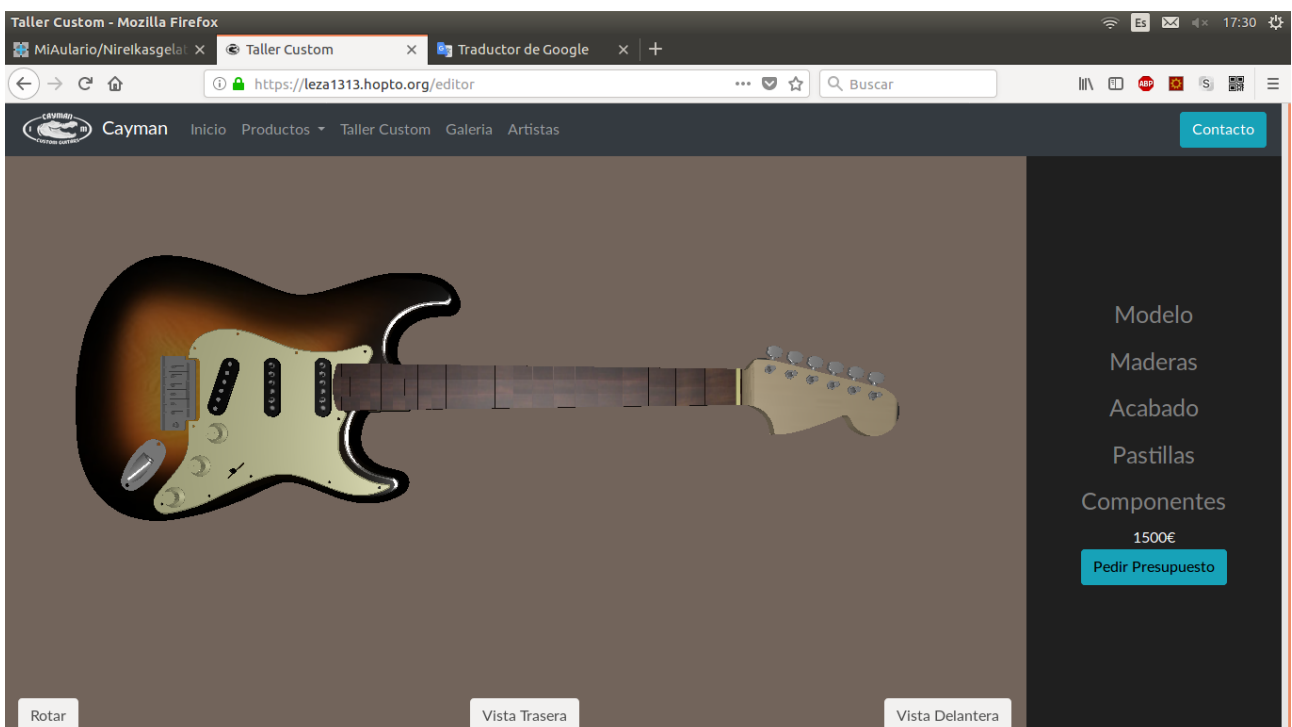
In case you need to add or delete an administrator, you will be able to do it by accessing the page “Administradores” that will appear in the footer when you are logged in.



Picture 46: Admin view of footer

There you will have a form to submit a new administrator. Below a table with the already existing administrators and crosses (X) to delete them.

3D editor Manual



Picture 47: View of the 3D editor

You will have this view of the editor the whole process.

There are 4 interactions possibles with the editor.

1. Change the shape of a component or change a component.
2. Change the finish of a component.
3. Change the position and orientation of the guitar.
4. Change the material or brand of a component.

The editor is divided in two sections, the 3D scene, and the menu of the right.

Changing the finish and the position is made in the 3D scene. While changing the shape, a component and the material or brand is made within the menu of the right.

Actions inside the 3D scene

- Change the position and orientation of the guitar

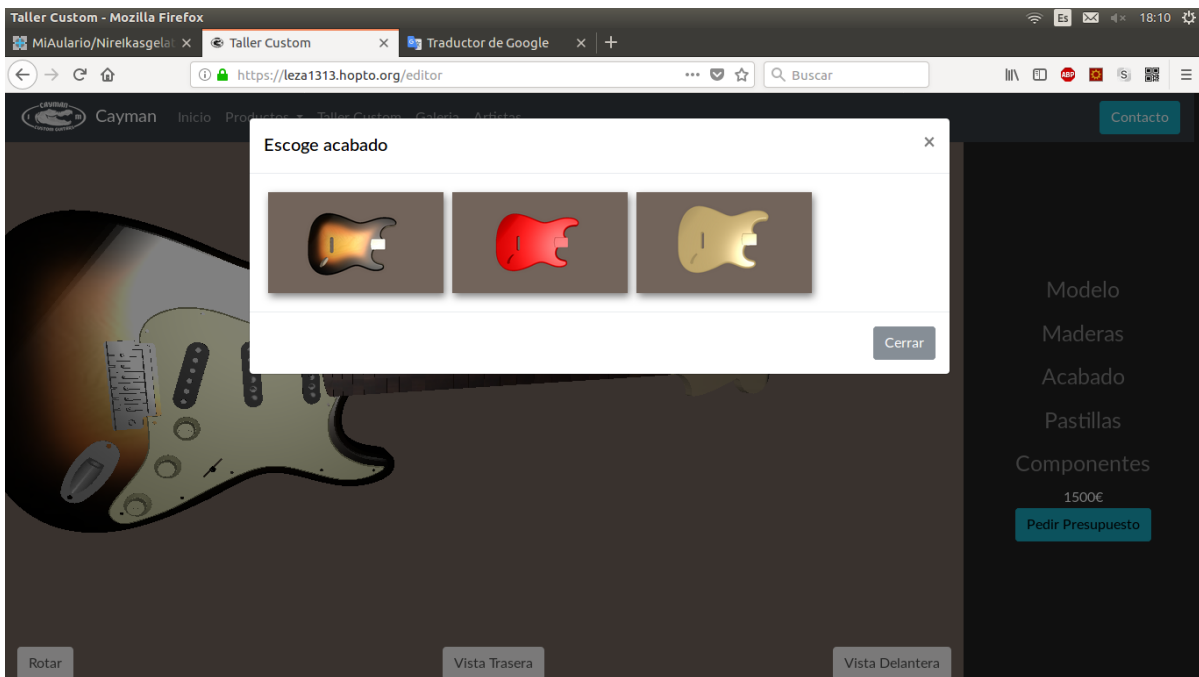
To change the orientation of the guitar, you should left-click and drag the mouse over the 3D scene.

To zoom in and zoom out, you can use the mouse wheel.

To move the guitar, right-click and drag the mouse.

- Change the finish of a components

Hovering the mouse over a component will light it. If you double-click the component lighted will show a menu displaying the options of colour for that component.



Picture 48: Modal to change the colour of the body

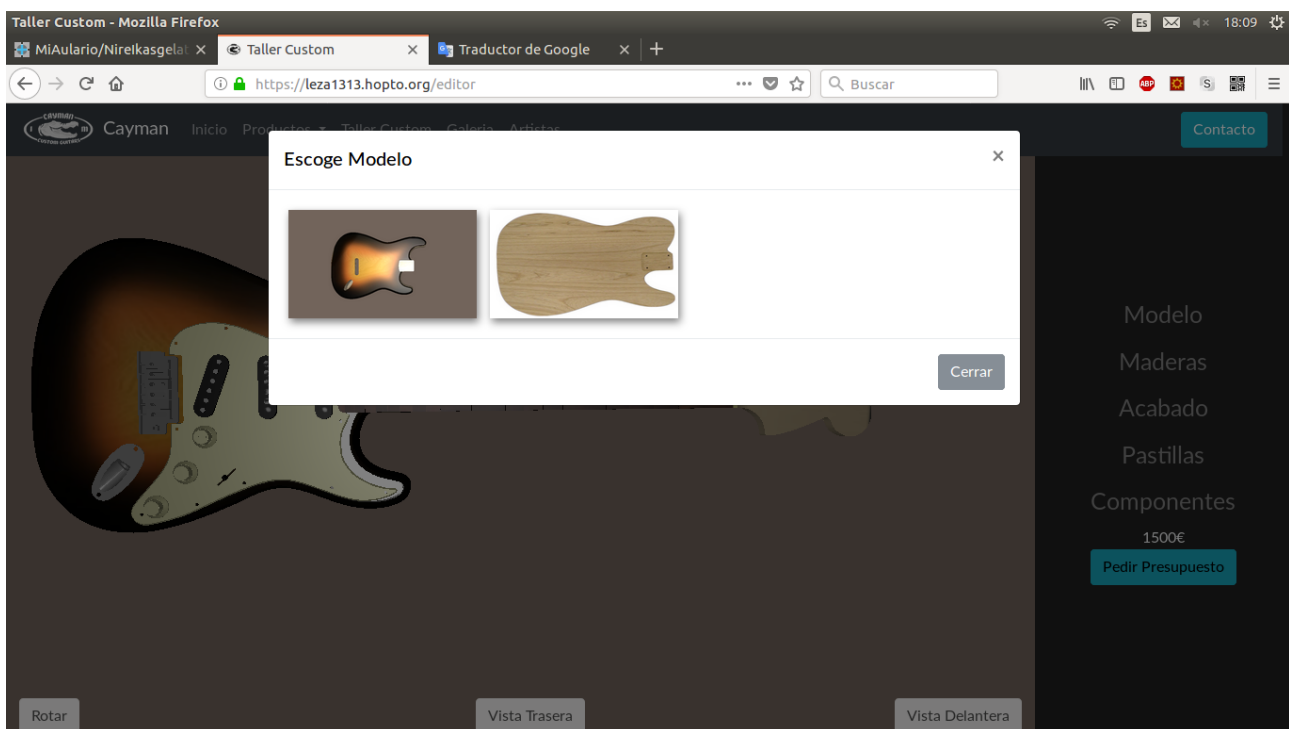
- Automatic rotation and views

In the bottom of the 3D scene there is 3 buttons. The left one rotates the guitar until clicked again. The other two buttons moves the guitar to have a straight view to either the front or the back of the guitar.

Actions outside the 3D scene

The menu of the right is quite self explanatory. If you want to change the model of the guitar, you click the “Modelos” and click the image of the model wanted. Then in “Maderas” you can select the type of wood for few components. In “Pastillas” you can change the type of pickups and the brand.

The only change on the finish that is not made inside the 3D scene is to select if you want a Gloss or Matte finish on the body of the guitar.



Picture 49: Modal to change to model of guitar

Testing with users

We run some test with some users to get some feedback from people outside the project. The tested people used the web page for few minutes to get a brief sense of how the page is organized.

Some of the tests we run were:

- Go to the web page, and buy a “Tennessee” guitar. The PayPal environment is in sandbox so the person didn’t have to worry about money.
- Go to the web page, and build in the 3D editor a Stratocaster guitar, with a Humbucker mastile pickup of brand “Dimarzio”, and with a white finish.

We also run some tests to the company members. This gave us a really good insight of how easy to use really was the web page. The reason for this was that one member of the company is older and not so familiar with the technology, the other member of the company have been more involved in the development of the project. So we had two very different profiles tested with this tests:

- Add a new artist and then update its description

- Add a new order with the Android app. Once the order is added update its delivery date. The username and password is “demo”:”demo”.

Finally we run some System Usability Scale tests on different people with different knowledge in technology but almost everyone with some knowledge in the guitar and bass world.

From each test we received some interesting feedback, we were expecting most of it, but we definitely didn’t expect some problems people found. This are the conclusions we came up after reviewing all the feedback.

- When buying a product, you are redirected to a PayPal page that focus your attention to pay with a PayPal account, even though there is an option to pay with credit card without using an account. This confused some of the users.
- Users didn’t notice that you can click the images in the description of the product or artist, to get a full-screen view of it.
- Most users didn’t notice they could click the cards of artists, in order to read more information about the gear they use and their experience with the company. This problem comes from the amount of web pages that have an artists section, with no further information about them than a picture and their name. This was a surprise for us.
- A decent amount of people found hard to use the 3D editor. However when they realize how to use it, they all enjoyed and thought it is a really good feature. They are looking forward to use the 3D editor with more options available.
- Once learned the process of adding orders with the android app, the older member of the company agreed of the convenience of adding and managing the orders from the smartphone.
- The operator of the milling machine is grateful for the the ease of use and extra space in the work environment, allowing him to supervise the milling machine while working in other stuff at the same time.

- All in all, people agreed that the web page is easy to use (excluding the 3D editor).

Conclusions and future lines

The company and I are really satisfied with the result of the effort we have made. We built a working web shop, so they can start doing some business. We created an order management system, that is compatible with an Android app, to allow them to take in-person orders and entering them to the database.

We are excited that the testers liked the 3D editor of custom guitars, this was a key point about the web, to have something unique that visitors can only find in our web page. Furthermore, the company is able to add more components, options and models dynamically as they design new models.

The interface connected to the milling machine was a simple yet very useful tool that allowed them to work smarter. To this, we can improve the Android app in the future allowing them to add new buttons that launches new jobs in their milling machine.

We both agree that we have “computerize” the company, not only in the sales of handmade guitars but also in the manufacturing processes.

At this point the project is almost in deployment stage. There is few things to improve before that, such as a little pop-up with the instructions for the 3D editor, magnifying glass icon in the images that can be shown in full-screen and minor tweaks.

The next steps for the resources of this project , would be to buy a proper domain, hire a hosting solution, set up a mail server. As for software, we can try to find an easier solution (for the company) to the payments system. Maybe integrate a payment gateway, and make use of some API to avoid going to PayPal to create buttons and then go back to our page to add a product. Some system of discounts could be also interesting.

However, the most interesting upgrade in the future would be to exploit the business information from the database. Adding a section in the web page to show how many guitars of each model they have sold, which one is the most sold, which months

are the ones they sell the most products. With all of this they can make wiser decisions of marketing and promotion.

Bibliography

- 1: Three JS. Ricardo Cabello, 2012, <https://threejs.org/>----- (accessed 15/2/2018)
- 2: Bootstrap4. Mark Otto, Jacob Thornton, 2011, <https://getbootstrap.com/>---- (accessed 3/12/2017)
- 3: jQuery. John Resig, 2006, <https://jquery.com/>----- (accessed 3/12/2017)
- 4: Slick JS. Ken Wheeler, 2017, <http://kenwheeler.github.io/slick/>----- (accessed 15/1/2018)
- 5: Simple lightbox. Damir Brekalo, 2016, <http://dbrekalo.github.io/simpleLightbox/>----- (accessed 17/3/2018)
- 6: Popper.js. Federico Zivolo, 2016, <https://popper.js.org/>----- (accessed 7/5/2018)
- 7: clickable-tr-jquery. DeOldSax, 2015, <https://github.com/DeOldSax/clickable-tr-jquery>-(accessed 25/5/2018)
- 8: PayPal IPN documentation. Ken Howery, Luke Nosek, Max Levchin, Peter Thiel, Elon Musk, 1998, <https://developer.paypal.com/docs/classic/ipn/integration-guide/IPNIntro/?mark=ipn> (accessed 12/4/2018)
- 9: JWT. M.Jones, Microsoft, J.Bradley, Ping Identity, N.sakimura, 2015, <https://jwt.io/>--- (accessed 26/4/2018)
- 10: Uploadcare. Igor Debatur, Dmitry Mukhin, Lev Leviev, Anatoly Chernyakov, 2011, <https://uploadcare.com/>----- (accessed 20/3/2018)
- 11: Repetier Server Guide. Marcus Littwin, Roland Littwin, 2011, <https://www.repetier-server.com/manuals/programming/API/index.html>----- (accessed 2/5/2018)