# Design Document

E-commerce website

Mateusz Pawlowski, James Porter

## Contents

## 1. Introduction

*Section written by Mateusz Pawlowski G00361162*

As a group of two we decided to do an application based on Creating, Deleting, Updating and Reading objects (CRUD). In order to do this, we decided to create a website that is based around websites like meteor and Carphone Warehouse (buying phones). As an admin you are able to create new phones, giving them all the information needed, deleting a phone, or updating it. Same goes for phone accessories.

We felt this would be an interesting topic to go through, but we wanted to use a language that we barely knew. This is why we picked python and after some investigations, James discovered a framework for python called Django that is specifically used for websites. It is in the topmost used frameworks for python, so we decided to implement this into the project.

## 2. Purpose

*Section written by Mateusz Pawlowski G00361162*

The purpose of this document is to give the reader a clear understanding on the basic structure of the project and outline how the application works. How it came all along and the steps we did in order to create this application. The document will show how an Admin user can create, update, delete and read objects (Phones). We will give a brief description of the language and the framework we used for this project.

## 3. System Requirements

*Section written by Mateusz Pawlowski G00361162*

Tested on Windows mostly. The only differences that may happen through out the other OS systems is, you might need to install different packages.

As of web browsers, you can use (these browsers were tested):

- Google Chrome
- Firefox
- Microsoft Edge
- Safari

## 4. Technology used and why

*Section written by Mateusz Pawlowski G00361162*

### Python

We decided to go with python as both of us had very little experience in it and for this project we wanted to challenge ourselves and learn a new language. Python is a fairly simple language to learn and is one of the most popular out there, meaning there is a lot of tutorials out there showing you how to do certain parts of the code. Python offers a lot of frameworks, and for our project we went with Django. Some of the most top companies use python in their technology stacks, these include, Instagram, Spotify and Disqus.

### SQL Lite

The database chosen was SQL Lite mainly because Django automatically creates an SQL lite database for your project. Django officially supports three other databases and they include Oracle, MySQL, and PostgreSQL however we didn't feel the need to explore the other databases, so we stuck to SQL Lite.

### Django

Django is a high-level python web framework used for rapid development and clean, pragmatic design. It is open- source and the main goal of the framework is take care of a lot of the issues of web development so you can focus on writing an application without starting everything from scratch.

At the beginning of the project we thought of using ruby on rails as the web framework but after some investigation we felt that it was pretty complicating and awkward to set up and use. On the other hand, Django is very simple when used as a beginner and even the more advanced features are fairly okay to figure out after you have played around with the framework for a few days.

It is one of the most popular web frameworks used which gives it a lot of attention, allowing us to find a tutorial about basically anything. The tutorials we mostly used came from stack overflow and YouTube. Django has a built-in admin panel which helps massively. You as the user using the framework don't have to worry about creating an admin panel as Django provides that for you from the beginning. Django is one of the best when it comes to security too. It hides your websites source code and has protection against XSS and CSRF attacks.

### Selenium IDE

Selenium IDE is an extension you can add to your browser like Firefox and Google Chrome. The extension is used to tests websites. We decided it would be good to test the website ourselves before handing it out to be corrected. Selenium is fairly simple to operate, and we learned about it during our time in college, so we had a good idea how to use it.

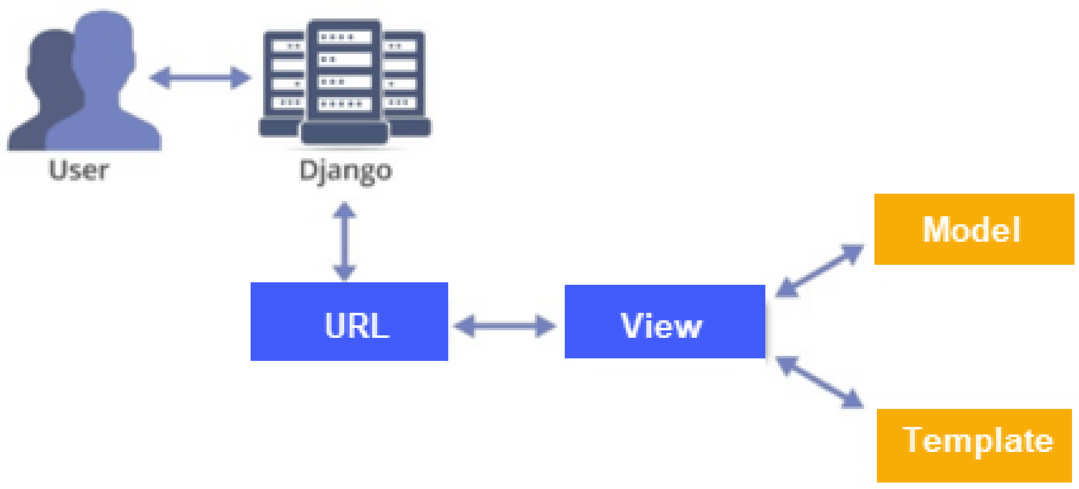
# 5. Architecture of the Solution

*Section written by James Porter G00327095*

## 5.1 Outline of the System Architecture

For the Architecture of the System we used the Python Django Package. It is based on the Model View Template Data Model designed by Django and encourages high re-usability. The MVT (Model View Template) Architecture closely resembles the common architecture in a lot of Front End Frame works MVC(Model View Controller). As shown in base.html template file being used all the other aspects of the webpage by using {% Block Content%}.

The Diagram below by Guru99 explains how the User interacts with the Django Server.



1. The User Connects to the Django Server.
2. The Django Server grabs what it needs from the URLS.py folder
3. The Urls.py is linked to the View which takes in the html information from the template and the Models.py File for its data and database information that will be later displayed on the template. E.g. For Each Phone in the Phones Model display their brand and name.

We decided to use the default Django supported database language which is SQLite. We decided to use the SQLite built in because of Django's inherent ability to work with it at ease.

Django was able to make each table on the database through the models.py folder and uphold the relationships as needed depending on the parameters we entered in the Model.py File.

## 5.2 What is Django?

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

- It is built to be **Ridiculously Fast** despite using Python and does not contain a lot of bloat code or packages like similar Front End Frameworks such as NodeJS + React or Angular.
- Django also has built in cookies that the developer does not need to setup to allow the website to load faster after the user's first interaction with it. This was a great feature for the end user but did cause some issues during development as we had to run our server in incognito browser or delete cookies consistently to see the changes we made in the code on our website. Despite this it was a pleasure to work with in terms of speed when testing.
- It is very **secure** and comes with Tokenization and user authentication and verification built in. To Ensure that the user's information is private, and hackers cannot hijack the packets between the user and the server by using tokenization.
- Django is Exceedingly scalable as each component is separated and can be modified individually. You can see this in our project when we separated the Users, Phones and

base MJ-Phones folders. We did this by running python manage.py createapp <name of folder>.
- Django allows re-use of these components in later projects. This is useful in a business environment as the business can keep a design from a previous products website or be scaled highly after the initial creation of a website. Django names this the “share-nothing” architecture and Django has been used in big industries such as Instagram, Pinterest and Mozilla for years due to it’s very easy and stable scalability properties.

## 5.3 Components of Django

Form:

- Django has a powerful form library which handles rendering forms as HTML. The library helps in validating submitted data and converting it to Python types.

Authentication:

- It handles user accounts, groups, cookie-based user sessions, etc.

Admin:

- It reads metadata in your models to provide a robust interface which can be used to manage content on your site.

Internationalization:

- Django provides support for translating text into various languages, locale-specific formatting of dates, times, numbers, and time zones.

Security:

Django provides safeguard against the following attacks:

- Cross-Site Request Forgery (CSRF)
- Cross-site scripting
- SQL injection
- Clickjacking
- Remote code execution

## 5.4 What is SQLite?

SQLite is a software library that provides a relational database management system. The lite in SQLite means light weight in terms of setup, database administration, and required resource.

SQLite has the following noticeable features: self-contained, serverless, zero-configuration, transactional.

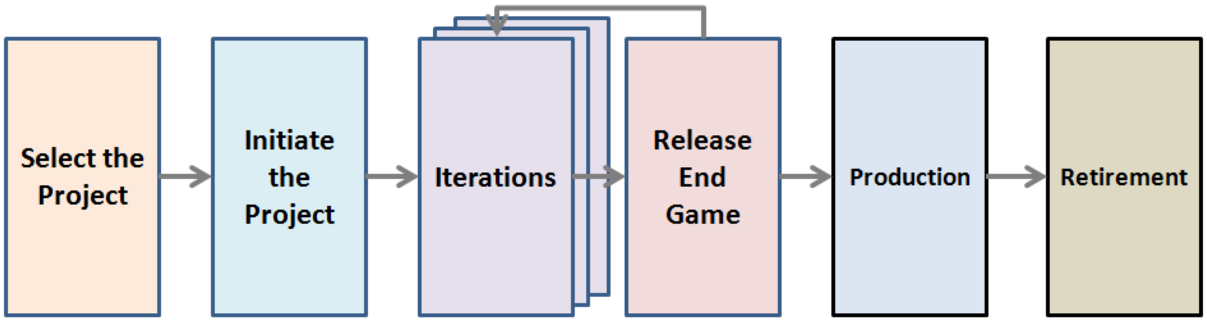
Therefore, it works so well with Django. It can be running without a separate server and does not require any configuration files to be downloaded that are not already provided by Django Framework. It is because of this that the framework is as easy to use as a NOSQL database such as Mongo DB.

SQLite uses dynamic types for tables. It means you can store any value in any column, regardless of the datatype. SQLite also allows a single connection to access multiple databases which means we can do joins and copy multiple databases with a single command.

It also capable of creating in memory databases which are fast and responsive to work with, which can later be saved and stored to a non-volatile memory space such as a hard disk or solid-state drive.

## 6. Architecture of the Solution

*Section written by James Porter G00327095*



Reference : http://www.nwlink.com/~donclark/design/agile_design.png

For our project we went with the **Agile Design Methodology**. We chose this because it is highly used in industry and greatly represented the kind of project progress, we wanted to work around with our lecturer included.

**Agile Methodology** requires the team to **break down a large task into smaller increments**, with minimal long-term planning. The Iterations are completed in a short time frame, **typically between one and four weeks also known as a Sprint**. Agile highly relies on **communication and collaboration to sustain a learning and fast paced development environment**. Knowing we would develop using a framework we had never encountered before in our software development degree, we felt that the learning outcomes we could achieve in the tight time schedules using this methodology would be optimal for completing this project and understanding the Django Framework along the way.

For the Testing Part of this Methodology, we suffered slightly due to the COVID-19 outbreak so we decided to run our own tests and ask family members to test our website and give us their thoughts so that we could further develop and optimize the website. Normally this methodology relies on customers or in this case anonymous testers  who have an insight on our project to be major collaborators in the design and implementation of the code but we were not expecting the impact COVID-19 had on our project so it led to some short comings in many different fields. One of the major ones is testing / user usage which this methodology highly relies on.

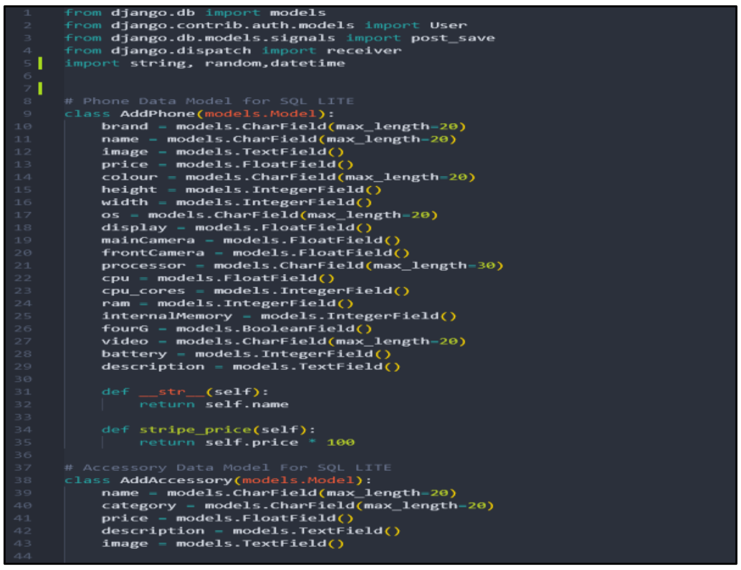
## 7. Features of the Implementation

*Section written by James Porter G00327095*

## 7.1 Database Design

The Database we used was SQLite, which is built into Django natively. To Create Tables for the MJ-Phones Website we had to create models within our applications folders, using Django's provided model.py files.

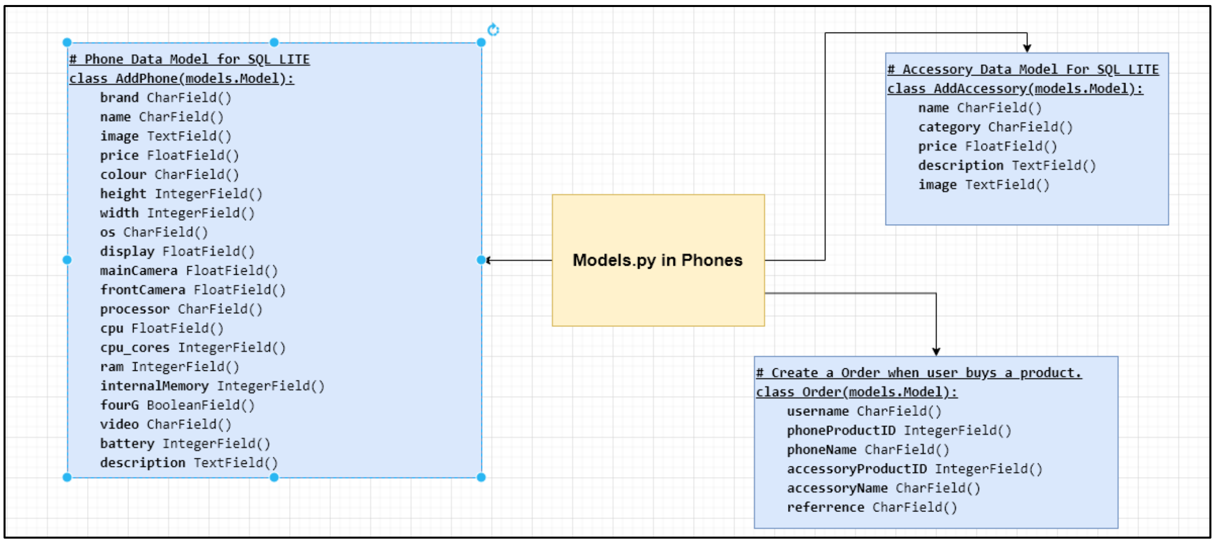
For Example, this is the first section of our models.py file within Phones Folder in MJ-Phones.



```python
from django.db import models
from django.contrib.auth.models import User
from django.db.models.signals import post_save
from django.dispatch import receiver
import string, random,datetime


# Phone Data Model for SQL LITE
class AddPhone(models.Model):
    brand = models.CharField(max_length=20)
    name = models.CharField(max_length=20)
    image = models.TextField()
    price = models.FloatField()
    colour = models.CharField(max_length=20)
    height = models.IntegerField()
    width = models.IntegerField()
    os = models.CharField(max_length=20)
    display = models.FloatField()
    mainCamera = models.FloatField()
    frontCamera = models.FloatField()
    processor = models.CharField(max_length=30)
    cpu = models.FloatField()
    cpu_cores = models.IntegerField()
    ram = models.IntegerField()
    internalMemory = models.IntegerField()
    fourG = models.BooleanField()
    video = models.CharField(max_length=20)
    battery = models.IntegerField()
    description = models.TextField()

    def __str__(self):
        return self.name

    def stripe_price(self):
        return self.price * 100

# Accessory Data Model For SQL LITE
class AddAccessory(models.Model):
    name = models.CharField(max_length=20)
    category = models.CharField(max_length=20)
    price = models.FloatField()
    description = models.TextField()
    image = models.TextField()

```

I will now display each model in the database in a diagram form.

Model.py in Phones Folder, This is the model for the Phones, Accessories and Order Pages.

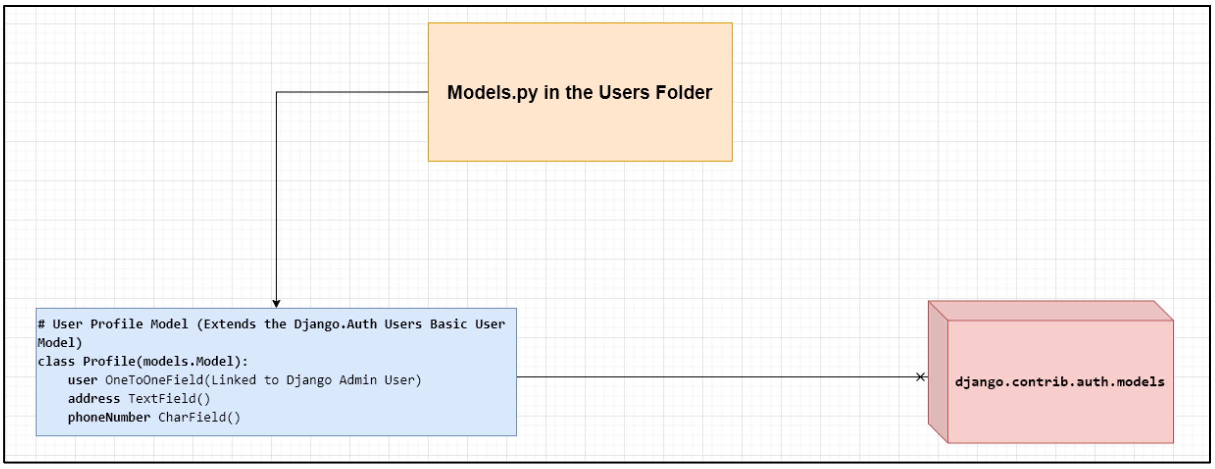


```
# Phone Data Model for SQL LITE
class AddPhone(models.Model):
    brand CharField()
    name CharField()
    image TextField()
    price FloatField()
    colour CharField()
    height IntegerField()
    width IntegerField()
    os CharField()
    display FloatField()
    mainCamera FloatField()
    frontCamera FloatField()
    processor CharField()
    cpu FloatField()
    cpu_cores IntegerField()
    ram IntegerField()
    internalMemory IntegerField()
    fourG BooleanField()
    video CharField()
    battery IntegerField()
    description TextField()
```

```
# Accessory Data Model For SQL LITE
class AddAccessory(models.Model):
    name CharField()
    category CharField()
    price FloatField()
    description TextField()
    image TextField()
```

```
# Create a Order when user buys a product.
class Order(models.Model):
    username CharField()
    phoneProductID IntegerField()
    phoneName CharField()
    accessoryProductID IntegerField()
    accessoryName CharField()
    referrence CharField()
```

Model.py in the Users Folder



```
# User Profile Model (Extends the Django.Auth Users Basic User
Model)
class Profile(models.Model):
    user OneToOneField(Linked to Django Admin User)
    address TextField()
    phoneNumber CharField()
```

The Profile Field in my Users model is a linked extension of the default users profile in Django.auth.contrib.models That Comes with Django at localhost:/admin part of Django. The Default Model did not allow for addresses or phone numbers, so I had to extend it by creating a User's Model and linking it to the Default User Model using a One to One Field.

The reason we remained with SQLite is because we already understood MySQL syntax greatly due to previous course work in our college degree and because of its native implementation to Django along with other reasons such as Speed, Reliability, Ease of Use.

## 7.2 Server Design

**Django is a full stack web framework** in which you can develop the whole web application within Django without linking it back to a Front-End Framework such as React, Vue or Angular. This was very useful as we did not have to work with multiple languages or frameworks to create our website and our learning could remain consistent with the technology we had chosen. The Only languages we would need to worry about were HTML, CSS, JavaScript (with JQuery) and Python along with the various Django Based Terminology.

So, in this use case **we did not need to make a separate** server which held an API to be called to. Everything was completed within Django Itself and was then hosted locally on **Localhost** at Port **8000**

## **7.3 Routes**

Below are the Routes the MJ-Phones Website we created has:

| HTTP Method: | URL path (Route) | Description |
|---|---|---|
| **GET** | / | Display Home Page with Place Holder Fake News and Details on How to Use the Website along with a Welcome Message. |
| **GET** | /admin/ | Open Django Administration Page, To Log In and Modify Models |
| **POST** | /register/ | Register a New User On The Website |

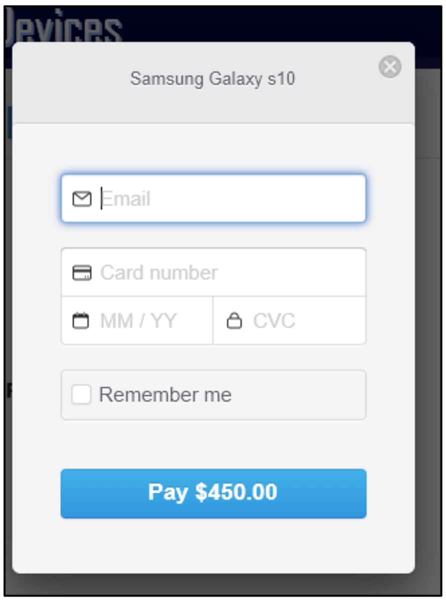
| | | |
|---|---|---|
| POST/GET | /login/ | Enter In Details of User, Post to Server Validate and then if Validated Correctly. Login and GET/Redirect to main page with Navigation Bar Change. |
| GET | /compare/ | Compare Two Phones Specifications/Hardware |
| POST/GET | /charge/<int:phone_id> | Create a Charge for Selected Purchased Phone. And POST information and send it to Stripe API |
| POST/GET | /charge2/<int:accessory_id> | Create a Charge for Selected Purchased Accessory. And POST information and send it to Stripe API |
| GET | /orders/ | Display all of the Currently Logged In Users Orders on the Website. |

### 7.4 Stripe API

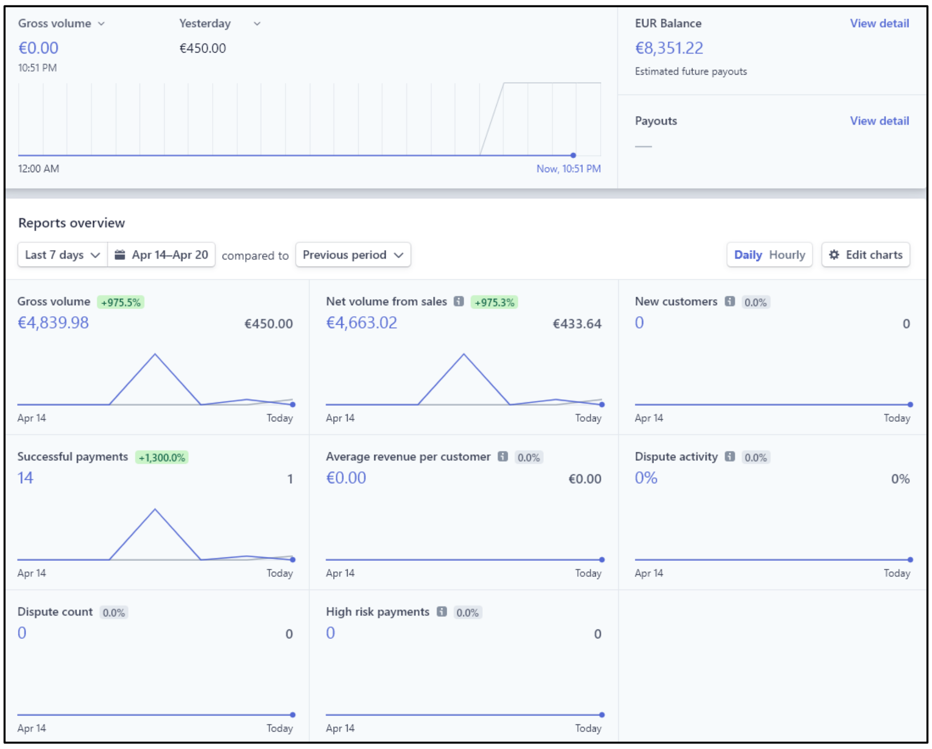
For this Projects Purchasing Model of Phones and Accessories we decided to implement a purchasing API by Stripe.

The Reason we chose this API is because it's easy to implement once you have a Stripe Developer Account and it's used on quite a few platforms and is highly recommended. We were going to originally use PayPal but it's documentation and integration into Django became very difficult to understand and it did not offer as much flexibility as using Stripe which allows the user to use their own Credit/Debit Card Details. By Clicking the Test Mode on Stripe, you can enter in fake card numbers and emails to test if the Stripe API is working on the website. This came in very good use as I could simulate a real e-commerce experience without directly affecting our own personal debit/credit card information.

*What the User Sees:*



*What we see on Stripe Dashboard:*



Gross volume | Yesterday
€0.00 | €450.00
10:51 PM
12:00 AM | Now, 10:51 PM

EUR Balance | View detail
€8,351.22
Estimated future payouts

Payouts | View detail
—

Reports overview

Last 7 days | Apr 14–Apr 20 | compared to | Previous period | Daily Hourly | Edit charts

Gross volume +975.5%
€4,839.98 | €450.00
Apr 14 | Today

Net volume from sales +975.3%
€4,663.02 | €433.64
Apr 14 | Today

New customers 0.0%
0 | 0
Apr 14 | Today

Successful payments +1,300.0%
14 | 1
Apr 14 | Today

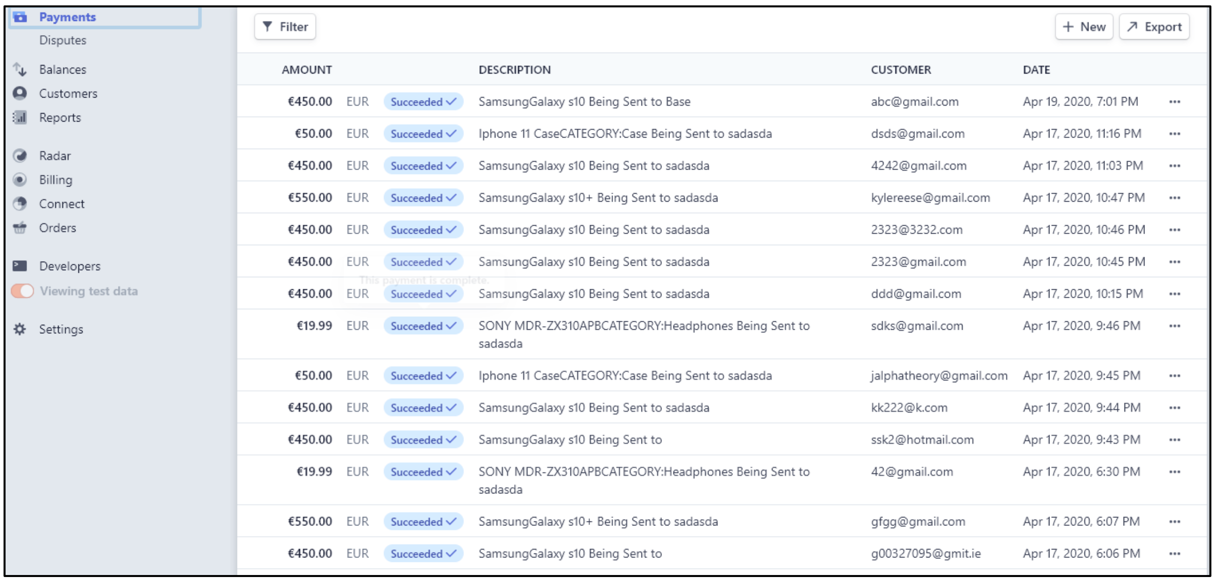
Average revenue per customer 0.0%
€0.00 | €0.00
Apr 14 | Today

Dispute activity 0.0%
0% | 0%
Apr 14 | Today

Dispute count 0.0%
0 | 0
Apr 14 | Today

High risk payments 0.0%
0 | 0
Apr 14 | Today

*Payments on Stripe Dashboard:*



Payments
Disputes
Balances
Customers
Reports
Radar
Billing
Connect
Orders
Developers
Viewing test data
Settings

Filter | + New | Export

| AMOUNT | | | DESCRIPTION | CUSTOMER | DATE | |
|---|---|---|---|---|---|---|
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to Base | abc@gmail.com | Apr 19, 2020, 7:01 PM | ··· |
| €50.00 | EUR | Succeeded ✓ | Iphone 11 CaseCATEGORY:Case Being Sent to sadasda | dsds@gmail.com | Apr 17, 2020, 11:16 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to sadasda | 4242@gmail.com | Apr 17, 2020, 11:03 PM | ··· |
| €550.00 | EUR | Succeeded ✓ | SamsungGalaxy s10+ Being Sent to sadasda | kylereese@gmail.com | Apr 17, 2020, 10:47 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to sadasda | 2323@3232.com | Apr 17, 2020, 10:46 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to sadasda | 2323@gmail.com | Apr 17, 2020, 10:45 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to sadasda | ddd@gmail.com | Apr 17, 2020, 10:15 PM | ··· |
| €19.99 | EUR | Succeeded ✓ | SONY MDR-ZX310APBCATEGORY:Headphones Being Sent to sadasda | sdks@gmail.com | Apr 17, 2020, 9:46 PM | ··· |
| €50.00 | EUR | Succeeded ✓ | Iphone 11 CaseCATEGORY:Case Being Sent to sadasda | jalphatheory@gmail.com | Apr 17, 2020, 9:45 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to sadasda | kk222@k.com | Apr 17, 2020, 9:44 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to | ssk2@hotmail.com | Apr 17, 2020, 9:43 PM | ··· |
| €19.99 | EUR | Succeeded ✓ | SONY MDR-ZX310APBCATEGORY:Headphones Being Sent to sadasda | 42@gmail.com | Apr 17, 2020, 6:30 PM | ··· |
| €550.00 | EUR | Succeeded ✓ | SamsungGalaxy s10+ Being Sent to sadasda | gfgg@gmail.com | Apr 17, 2020, 6:07 PM | ··· |
| €450.00 | EUR | Succeeded ✓ | SamsungGalaxy s10 Being Sent to | g00327095@gmit.ie | Apr 17, 2020, 6:06 PM | ··· |

## 7.5 Bootstrap and JQuery:

### Bootstrap

For some of the templates and web pages we implemented Bootstrap to improve the visual design but both me and my team mate did not know a lot of bootstrap itself and in the future would like to learn more as it is a popular CSS framework for making web pages responsive and aesthetic.

We Implemented minor bits of boot strap into forms and different web pages to try cleaning them up a small amount without over populating our CSS file.

*Example: Dropdown Filter Form in Phones and Accessories, Modal View for Phone's Extra Specification Details.*

### JQuery

When learning how to accomplish filtering in Django, I ran into a issue where every single filter would appear on the URL even if we did not set them to anything. In order to eliminate this issue, I did some research and found a solution using JQuery which would remove any empty filters from the URL.

```
function disableEmptyInputs(form) {
  var controls = form.elements;
  for (var i=0, iLen=controls.length; i<iLen; i++) {
    controls[i].disabled = controls[i].value == '';
  }
}
```

This eliminated the issue entirely and made the URL's much shorter and better to send around after filtering.

***Example:***

***Before ->*** *phones/?brand=""name=""maxPrice=minPrice=2000…etc*

***After JQuery ->*** *phone/?minPrice=2000*

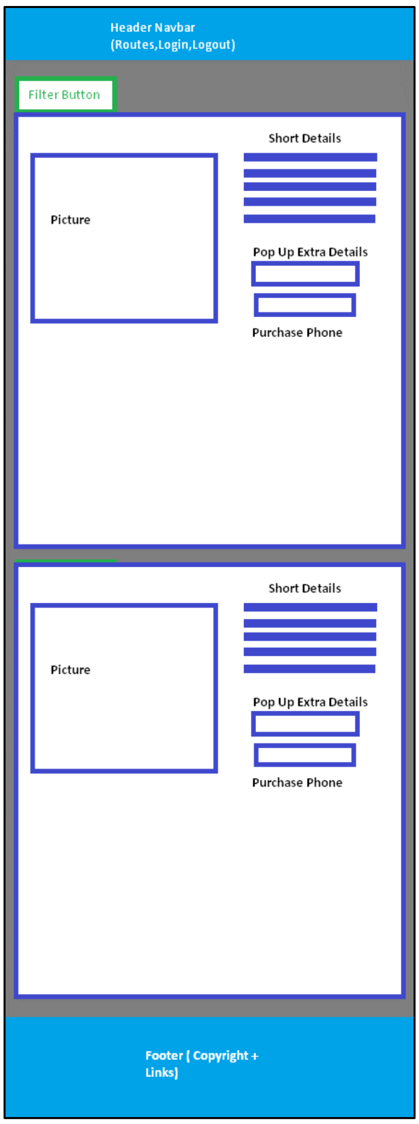
## 7.6 Django:

Django helped us display all the information from its default database language. SQLite and worked as the holding framework and concrete block for the whole project. With it's Model View Template architecture , once we knew how to do something it was a breeze and a lot of the time hours were saved by its innovative features such as Django Administrator, Security, Tokenization, Re-usability, and it's easy to use model integration for databases. It's a very powerful framework and remains quite small in size compared to other frame works. Based off my own observation of react, angular projects on GitHub. The Lines of code tend to be in the 100,000's whilst Django's seemed to stay within the realm of below 10,000. This to me shows how light weight of a framework it is. Everything in our project was linked using it from the html files, to the views controlling those html files which are then linked back to the models run by SQLite's integration into Django.

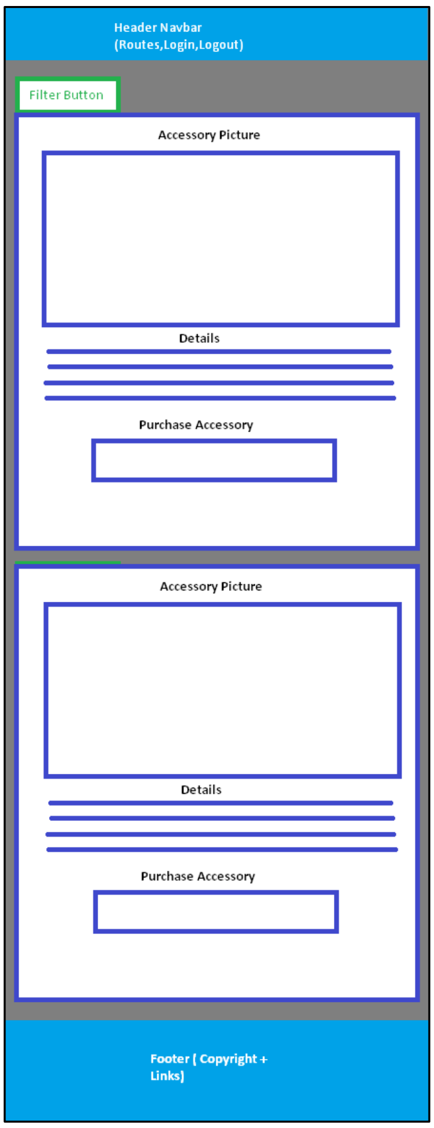
Like it's fellow Web Frameworks you could use commands in the console to modify and create or run the Django Web Framework, all through the manage.py file. For Example: python manage.py runserver or python manage.py makemigrations. Django seemed to support a lightly gripped database model as when we ever had issues with the database , we could easily change it in the models.py file and rather than having to re-enter every entry again, it would seamlessly change and allow us to keep most of our records in the database and we could just change the small details that we added or removed from it within the Django Administrator Section of the Website at /admin/.

## 7.7 Design Mock-Up For the Website:

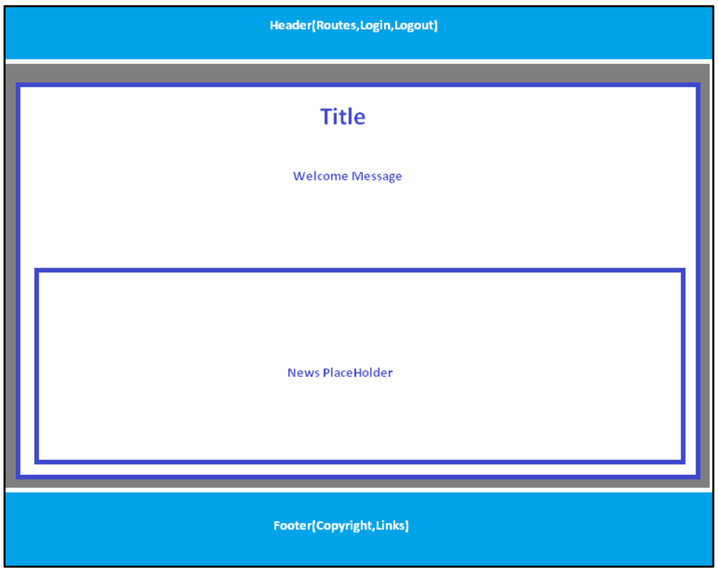
Phones.html



Accessory.html

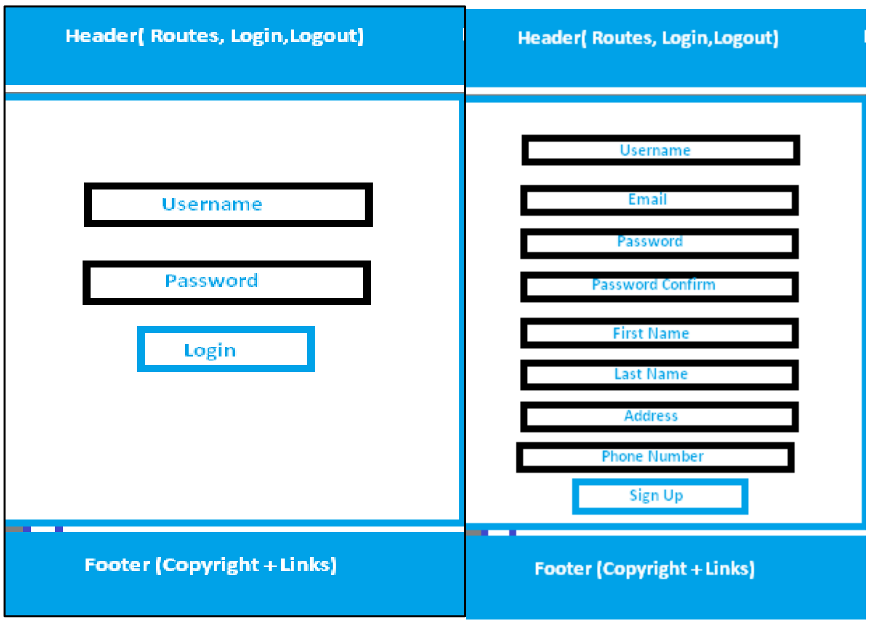


Home.html

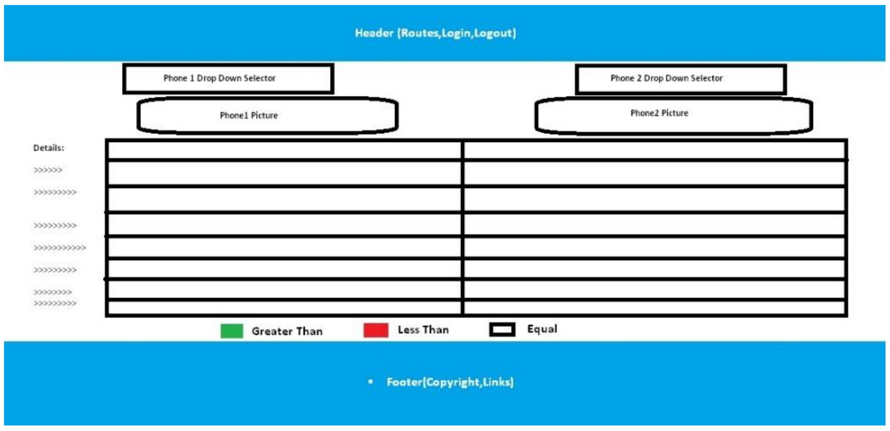


Login.html

Register. html


Header( Routes, Login,Logout)
Username
Password
Login
Footer (Copyright + Links)
Header( Routes, Login,Logout)
Username
Email
Password
Password Confirm
First Name
Last Name
Address
Phone Number
Sign Up
Footer (Copyright + Links)

## Compare.html



This mock-up was the basic idea we had for the website before creating it. This version of the mock up was created in MS Paint as the original mock-up was done on paper and I do not have it on me and do not currently own a scanner.

## Home.html

This is a basic introduction page to the website using basic html and no models linked. It has a placeholder for a future feature we could implement which is the news section. We decided not to implement the news section due to time constraints and difficulty with navigating the pagination features of Django. In the future we would like to add a news section that functions from the database/models and has a paginator as to not clutter the home page. This home page also includes instructions on how to use the website.

## Base.html

This is a boilerplate html template that allows the Django Re-Useable Block Feature. We reference this page on nearly every page of the website using:

```
{% extends "phone/base.html" %}
```

```
{% block content %}
```

The Base.Html contains Stylesheets, Scripts, Plugins, Navbar and the Footer for every single page on the website.

## Phones.html

Retrieves a Query Set of the Phones in the AddPhones Model in model.py. This is connected to the Database and new phones can be created from the python terminal or the /admin/ Django Administration part of the website. Each Phone is displayed on the Phone.html page unless the user specifically uses the drop-down filter box and filters the Query Set, so that only phones of certain parameters are shown.

- Each Phone shown has a Button called Show Product Details. When Pressed this button will show all the phone’s details.
- If the user is logged out, below the Show Product Details Button there will be a red text warning informing the user, that they need to be logged in to purchase a product.

- If the user is logged in, then a Purchase Product Button will show and when pressed the user will enter into the Stripe API and create a charge using their email and credit/debit card details and then redirected to /Charge/ to let them know their order went through. Doing this will also add the product they purchased to the My Orders section of their navbar.
- We Accomplished the ability to show each phone in the database by using a {% for each %} code statement in the template, the {%%} allows the use of python code and thus we iterated over each phone in the database using a normal python for each statement. The Filters are applied through the views.py and will alter the Query Set and therefore the information that comes up in the for each as well.

### Accessories.html

Works Very Similar to Phones.html except it is now retrieving Accessories from the AddAccessory Model in models.py and there is less filter cases as the Accessories model only has Category, Name, Price. There is No Show Product Details button on this page as every bit of information can be displayed on the page without a pop-up due to the fact the accessory's have very few details and don't create clutter on the page.

### Charge.html

This Page is called when a user purchases a product. It informs the user a product from the Phones Page is being sent to their address, it also sends the phone product's details and the users address/name to the Stripe API so that it can be shown on the Stripe dashboard and a website could receive payments and send packages to a user and keep track of their e-commerce trading.

### Charge2.html

This works the same as Charge.html except in this use case we are dealing with accessory purchases and not phone purchases.

### Login.html

Page to login after a user is registered using a username and a password.

### Register.html

Page to register a user using various fields that are important for the website to have when making charges or allowing users to login such as username, password, email, address, phone number.

### Logout.html

A Redirection Route that logs the user out and returns them to the home page

# 8. Limitations

*Section written by Mateusz Pawlowski G00361162*

As everybody should know, at the beginning of 2020 the world has come to a stop. With the whole covid-19 situation that's going on (As of 22nd of April) all the education has been put to stop. No schools are open or third level education sites which made the project harder to complete. As a group we were unable to meet each other in person anymore. Everything had to be done online and in order to contact ourselves we did it via WhatsApp or mail. This made the meetings more difficult to set up because we both had times during the day that did not suit us. This resulted into some parts of the project taking longer to complete than it would in person. The meetings with our tutor were also difficult to set up and eventually we came to the conclusion it will be just easier and handier to email our tutor.

## 9. Problems Encountered

*Section written by Mateusz Pawlowski G00361162*

As this was a language that both of us did not have much experience in and a framework we never heard of until we done some research about it, it is obvious we encountered into some problems.

One of the problems was implementing stripe. It was awkward implementing it and we had to use their sandbox mode in order to check if everything works properly.

Another problem was setting up a database. We had to watch some tutorials in order to get the sql lite database working and find out how to add objects. A great tutorial we found was from a user called: Corey Schafer, on YouTube and he does some great videos about python Django.

https://youtu.be/aHC3uTkT9r8

## 10. Recommendations for Future Development

### 8.1 Make the Site Responsive on all Devices:

I think the next step for this website would be to modify the CSS and Integrate Bootstrap more efficiently into it. Both me and my partner were not very efficient in the use of these programming technologies but if we wanted to bring this out as a real template for E-Commerce Business dealing in phones and phone accessories, it would need to be responsive for all devices and resolutions.

### 8.2 Implement a Model for the Home Page instead of a Place Holder

The home page right now is a placeholder, but it could do with a news section that uses RSS feeds or grabs information from the database in a blog like fashion, It would either be this or a Forum/User Commentary based section for our users to interact.

### 8.3 Ship the Website to a Domain Hosting Service or Heroku

Right now, the Project can only be downloaded from GitHub and ran locally which means it’s still a development build. After finishing off minor adjustments and page additions such as a Add to Basket Function. It would be a good direction to host the website for the general public to use and test, but the issue is this website would only be relative in use for a Phone E-Commerce Business and they will most likely want to rent their own servers outside of Heroku. Heroku would only be good for testing the project in a more real-world scenario outside of local hosting. The End Game would be a business hosting it on their own servers.

# 11. Conclusions

## Teamwork and Agile Development:

Throughout this project we learned how to work with other coders as previously to do this most work we had done was solo and unless you are a freelancer, the industry of software development is usually done within teams and you are given parts of work to do before certain deadlines. At first it was strange for me and my teammate but as time went on, we got used to the regular check-ups on each other and the meetings we had with each other and our lecturer. GitHub and its numerous features did wonders for us through out this project and really helped us as a **version control system.** It helped us keep track of each other's commits, branches and numerous other things. We could setup milestones and assign jobs to each other so that me and my teammate were on the same page of what needed to be done and by who.

We got a good taste of the Agile Development Cycle through out the project, as we worked in constant communication and worked on parts of the project individually and combined them at the end of each of our work loads and checked that the integration of our sections caused no fatal bugs.

It was a great learning experience and we both feel more prepared for the real world team environments and agile development after this project.

## Python and Django:

Coming into this project. Neither me nor my teammate knew what framework to use. We need we wanted to make an e-commerce website based on technology and eventually settled on phones, but we did not know what framework to use. We wanted to go outside the box and really challenge our ability as programmers, so we chose Django. At first, we had no clue how to navigate the framework as Django does not support the same level of Auto Completion that big languages like Node JS, C# and Java would have. The only autocompletion or intelli-sense we could use was within certain parts of the python files but even then, it was not much. With Research through the documentation and on YouTube we quickly had to learn it so we could get started within the 3rd week of the semester and through trial and error over many months and much learning. We eventually arrived at a point me and my partner feel comfortable with Django even though it was all self-taught work and effort. We felt proud coming out of this project and felt we were really tested on our ability to adapt and develop as programmers, as it's common knowledge that the technologies used in the IT industry are always changing so being quick to adapt is of key important. If asked to do this project again, with the knowledge we have now. I feel we could make something much better and in less time.

## Time Management Skills:

We had time management issues coming to the end of the weeks of development of this project and a lesson we learned when we had to rush parts near the end. Is that the consistent work and making goals and milestones to follow by, are very important. In this case we had other projects and are living in times of the pandemic of COVID-19 but there were also times where our work slacked and in the future me and my team mate would like to improve on this so that the next project we have to tackle, we can do much more with.

We learned a bit of time management at the beginning and ending of the project and I feel these lessons in how to manage time and how not to get distracted are important in the industry of software engineering, as you will be hit with constant deadlines like we were and in the work force you have to do your work or risk penalties.

## Overall:

This was a challenging project, but my teammate and I have both agreed it was a great way to learn and test our abilities as programmers. It was frustrating at times but also great fun as we both, hold a passion for technological development and coding. I got to know my team mate better on a emotional and lifestyle scale. There were times when one of us could not do something as planned due to personal reasons or work, but we pulled through and had each other’s back along the way to ensure the success of the project. The knowledge we gathered through developing this will probably stick with us for a long time and we hope that next year we will be more ready for the challenges that may come our way.