Universitatea Tehnică a Moldovei

Sistem de gestionare a camerelor DSLR

DSLR camera controlling system

Student: **gr. FAF-161**
Terman, Emil

Coordonator: **Schidu, Vasile**
Lector universitar

Chișinău, 2020

MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII
Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Admis la susținere
Șef de departament:
Fiodorov I. dr., conf.univ.

„___” *mai*_____ 2020

Sistem de gestionare a camerelor DSLR

Proiect de licență

Student: _____ **Terman Emil, FAF-161**

Coordonator: _____ **Schidu Vasile, lect.univ.**

Consultant: _____ **Prodan-Șestacova Liubovi, dr., conf.univ**

Consultant: _____ **Catruc Mariana, lect.univ.**

Chișinău, 2020

Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică
Programul de studii Tehnologii Informaționale

Aprob
șef de departament
Fiodorov Ion, dr.conf.univ.

_____

„ 30” octombrie 2019

CAIET DE SARCINI
pentru proiectul de licență al studentului

Terman Emil
(numele și prenumele studentului)

1. Tema proiectului de licență *Sistem de gestionare a camerelor DSLR* .

confirmată prin hotărârea Consiliului facultății nr. *1* **din** „ *30*” *octombrie 2019*

2. Termenul limită de prezentare a proiectului de licență *14.05.2020*

3. Date inițiale pentru elaborarea proiectului de licență *Sarcina pentru elaborarea proiectului de diplomă.*

4. Conținutul memoriului explicativ
Introducere
1 Analiza domeniului de studiu
2 Modelarea și proiectarea sistemului
3 Realizarea sistemului
4 Documentarea produsului realizat
5 Evaluarea economică a proiectului
Concluzii

5. Conținutul părții grafice a proiectului de licență

6. Lista consultanților:

Consultant	Capitol	Confirmarea realizării activității	
		Semnătura consultantului (data)	Semnătura studentului
L. Prodan-Șestacova	*Argumentarea economică*		
M. Catruc	*Standarde tehnologice, Controlul calității*		

7. Data înmânării caietului de sarcini *02.09.2019*

Coordonator *Schidu Vasile* _____ _____
semnătura

Sarcina a fost luată pentru a fi executată de către studentul *Terman, Emil*
02.05.2020
semnătura, data

PLAN CALENDARISTIC

Nr. crt.	Denumirea etapelor de proiectare	Termenul de realizare a etapelor	Nota
1	*Elaborarea sarcinii, primirea datelor pentru sarcină*	*02.09.19– 30.09.19*	*10%*
2	*Analiza domeniului de studiu*	*01.10.19– 30.11.19*	*20%*
3	*Proiectarea sistemului*	*01.12.19 – 25.12.19*	*20%*
4	*Realizarea sistemului*	*16.01.20 – 14.03.20*	*25%*
5	*Descrierea sistemului*	*15.03.20– 10.04.20*	*10%*
6	*Testarea sistemului*	*11.04.20– 20.04.20*	*10%*
7	*Finisarea proiectului*	*21.04.20– 08.05.20*	*5%*

Student *Terman Emil ( )*

Coordonator de proiect de licență *Schidu Vasile ( )*

Declarația studentului

Studentul Terman Emil-Sergiu declară pe propia răspundere că lucrarea de față este rezultatul muncii mele pe baza propriilor cercetări și pe baza informațiilor obținute din surse care au fost citate și indicate, conform normelor etice, în note și în bibliografie.

Declar că lucrarea nu a mai fost prezentată sub această formă la nici o instituție de învățământ superior în vederea obținerii unui grad sau titlu științific ori didactic

Emil Terman

Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică
Programul de studii Tehnologii Informaționale

AVIZ

Teza de licenţă

Tema: DSLR camera controlling system
Studentul: Terman Emil, FAF-161

1. **Caracteristica tezei de licență:** Teza a fost elaborată în conformitate cu toate cerințele și standardele în vigoare; reprezintă o analiză detaliată a implementărilor existente, depășind domeniul imediat al implementării soluției propuse și analizînd scenarii similare în alte domenii. Teza reprezintă un vestigiu coerent al lucrului intelectual efectuat de către student.
2. **Estimarea rezultatelor obținute:** Rezultatele obținute sunt deja aplicabile; aplicația e bine documentată, iar codul este deloc criptic, bine organizat și elaborat cu responsabilitate.
3. **Corectitudinea materialului expus:** Referințele externe au fost luate din surse respectabile, iar sinteza informației obținute în rezultatul lucrului și a evaluării surselor externe este concludentă și logică. Teza corespunde standardelor tehnice, inclusiv analiza economică a proiectului și diagramele utilizate.
4. **Calitatea materialului grafic:** Figurile prezentate în lucrarea dată descriu si analizeaza în detaliu sistemul curent. Diagramele UML corespund standartului 2.0. Materialul grafic prezentat în lucrare îşi îndeplineşte scopul de a îmbunătăţi descrierea sistemului impementat.
5. **Observații și recomandări:** Această teză se face evidentă, în primul rînd, datorită caracterului practic al problemei pe care o abordează. Este recomandat de a extinde sistemul curent cu noi funcţionalităţi.
6. **Caracteristica studentului și titlul conferit:** Studentul a demonstrat o asiduitate deosebită în studierea independentă a unei teme netriviale și o abordare inginerească în soluționarea problemei specificate.
   Rezultatele obținute în cadrul tezei îmi permit să recomand admiterea tezei de licență d-lui Terman Emil spre susținere și să o apreciez cu nota maximă.

Conducătorul tezei de licență
Schidu Vasile

Abstract

Thesis **DSLR camera controlling system** presented by Terman Emil, FAF-161 was written in English. It has 25 figures, 11 listings, 8 tables, and 15 references. The report consists of introduction, 4 chapters, and conclusions.

The thesis aims to develop a tool to remotely configure and control DSLR cameras. It is a platform that mainly focuses on solving the problem of taking long time-lapses (one - two years or more).

The application was build using Python programming language. The tool consists from two components: camera communication and the client. The technologies used are Django framework for building the web api, SQL Server for storing the data, GPhoto2 for communicating with the cameras and Ykush for hard resetting.

DSLR Camera Controller represents a tool for photographers that want to take reliable long term time-lapses. It has the right tools to handle the camera settings, file transfer, camera failure strategy and time-lapse scheduling.

Rezumat

Lucrarea **DSLR camera controlling system** prezentată de Terman Emil a fost scrisă a fost scrisă în engleză. Ea constă din 25 de figuri, 11 secvențe de cod, 8 tabele și 15 referințe. Lucrarea face parte din introducere, 4 capitole și concluzii.

Teza are ca scop sa dezvolte un instrument pentru a configura si controla de la distanta camerile DSLR. Aceasta este of platform care in mare parte se focuseaza pentru a rezolva problema de luare repetata a fotografiilor intr-o perioada lunga de timp (unul-doi ani sau mai mult).

Applicatia a fost dezvoltată utilizând limbajul de programare Python. Instrumentul consită din doua parti: comunicarea cu cameri si clientul. Tehnologiile utilizate sunt frameworkul Django pentru a dezvolta API-ul web, SQL Server pentru a stoca datele, GPhoto2 pentru comunicarea cu cameri si Ykush pentru a reseta connectiunea cu cameri.

DSLR Camera Controller reprezintă un instrument pentru fotografi care vreau sa capteze fotografii time-lapsuri intr-o manera foarte sigura. Acesta ofera optiunele de a modifica setarile camerilor, de a configura un transferul de imagini, de a alege o strategie in cazul in care camerile intalnesc o eroare si programarea time-lapsurilor.

Table of contents

List of Figures

Listings

Abbreviations

DSLR – Digital single-lens reflex camera
HTTP – HyperText Transfer Protocol
HTTPS – Secure HyperText Transfer Protocol
TLS – Transport Layer Security
SSL – Secure Sockets Layer
PTP – Picture Transfer Protocol
SSH – Secure Shell
MTP – Media Transfer Protocol
OSX – Mac **OS X** operating system
HTML – HyperText Markup Language
JSON – Java Script Object Notation
REST – Representational state transfer
DCOM – Distributed Component Object Model
CORBA – Common Object Request Broker Architecture
RMI – Remote method invocation
XSD – XML Schema Definition
SMTP – Simple Mail Transfer Protocol
CRUD – Create Read Update Delete
URI – Uniform Resource Identifier
URL – Uniform Resource Locator
RFC – Request for Comments
API – Application Programming Interface
CERN – European Organization for Nuclear Research
CAR – Computer Assigned Reporting
TCP – Transmission Control Protocol
UX – User Experience
AJAX – Asynchronous JavaScript and XML
DOM – Document Object Model
UI – User Interface
MVC – Model View Controller
RDBMS – Relational database management system
SQL – Structured Query Language
CAP – Consistency Availability Network Partition tolerance
DBMS – Database Management System
ORM – Object Relational Mapping
DSL – Domain-Specific Language
USB – Universal Serial Bus

Introduction

Time-lapses are used in various domains. Some people want to study a natural phenomena, some want to have an overview on how a city has evolved over the years and others simply want to see how something was built. Some of the more advanced cases are when scientists want to take a picture of a particle at a very high speed, which emits light for a very short time, so a combination of time-lapse with a high exposure is used to create a sketch of what it looks like, to at least help the scientists in the early stage.

No matter the case, the quality requirements are usually extremely high, therefore a DSLR Camera is usually used for this purpose. But these cameras are usually very expensive, so nobody is really thinking about leaving them with no supervision for more than a day, even less people are thinking about making a tool to manage year long time-lapses and even less people are thinking how to easily scale, allowing the photographer to manage hundreds of time-lapses at once with no headache.

Setting up a time-lapse is not too hard. It is usually required to connect the camera to a computer, install the camera specific software, configure it and start taking photos. At the end of the time-lapse, the user usually drag and drops the photos in a special editor and it is done. This works perfectly for 1-2 day long time-lapses, where the user can sit near the camera an manage everything, or when the cost of failure is very low. But this gets extremely tedious when the same software is used for year long time-lapses. Different problems start to appear, like storage failure, camera disconnection, electricity cut off and more. This only gets worse when the user needs a few dozens of such time-lapses to run at the same time.

The goal of this thesis is to create a tool where users can set up a time-lapse and forget about its existence for the next few years, only getting occasional emails when something fails. Why it is important to have such a product on the market? Because there isn't one present. Or those that exist support only a specific model of camera, or offer no API for automation.

1 Project analysis and System requirements

1.1 Problem Definition

A photographer tasked with year long time-lapse usually spends its friday checking on the time-lapses. He would usually use an application like TeamViewer or another Remote Desktop application to connect to the computer that has the cameras connected to. He would have to manually check if the desktop application is still running and if it had any failures. If he is lucky enough, he might have written a script that opens the application on startup and program the mouse to start the time-lapse again from the last index, only if no additional pop-ups appear in the meantime. On the other hand, if the time-lapse is very expensive, say it's a very expensive experiment that requires a few weeks time-lapse, then the photographer would have for each camera an open remote desktop to see if it is still working.

It becomes exponentially difficult to automate this entire process for different models of cameras. In big enterprises usually the photographer is given a set o company owned cameras, which often come from different brands: Canon, Nikon, Sony, Pentax, Olympus, etc. Each brand usually comes with its own specific software, so it becomes next to impossible to write an all purpose reliable script to automate a time-lapse composed of multiple different cameras.

Taking continuous high quality time-lapses is one of the most challenging work that can be done by a photographer.

1.2 Project Analysis

DSLR Camera Controller is a tool which aims to provide easy access and management of cameras, supporting a wide range of models, through a web browser. The main features are:

a) Live preview;
b) Take a picture;
c) Camera configuration;
d) Failure strategy:
   - Send email;
   - Hard reset;
   - Reboot system after N failures;
e) Time-lapse scheduling;
f) File transfer;
g) Photo name patterns:
   - Time (timestamp, hour, minute, day, year, etc);
   - Capture index;
   - Camera serial number;
   - Camera name;
   - Time-lapse name;

1.2.1 Failure strategy

A camera can fail because of a variety of reasons. Some models, automatically go to into sleep mode when the connected computer does not send any commands for an hour. This happens very often, as some time-lapses usually happen during the working days. Another common case is when the local workers simply unplug the device, or trip on the cable. Other times the power is simply cut off for a brief period of time, causing some confusion on some models. So choosing a failure strategy is very important to reduce the time the user has to spend on the app.

One of the main problems the photographers have with time-lapses is that it can fail silently, meaning that it can encounter a failure and stop working, without the notice of the user. This is a critical problem for long time-lapses, because in most cases, it is unacceptable to have gaps in the time-lapse, especially gaps for weeks or months. To allow the user to be notified of any failures, he can enable failure email notifications, which will send notifications to the specified emails with data about the failure.

Getting failure emails every minute may not be quite the best solution when the user simply has to re-plug the camera. Re-plugging the camera solves the problem most of the time, so it is worth investing in a tool that does just that. Ykush is a device that allows automatic re-plugging of USB devices, therefore the application supports this option as well, in case the user has one connected. Enabling hard resetting on failure, will trigger any Ykush boards but it will also try to reconnect to the cameras internally, which for some cameras it works too.

Having a general solution that will work almost for sure for all models is very hard to come by, and even harder to keep it up to date, so in case the user does not want to purchase any extra external devices, a system reboot might be satisfactory The user can configure an automatic system reboot in case the camera fails N times in a row.

1.2.2 Time-lapse scheduling

It is not an easy task to create a tool that will combine all kinds of scheduling the user can come up with. So a solution to this problem is to use the Cron utility.

Cron is a time-based software utility for job scheduling in Unix like systems. It allows to express schedules to run periodically at fixed times, dates or intervals. While it will allow the user to express almost any kind of schedule he wants, he will have to learn a bit about cron scheduling, but attaching some of the most common cases to help them out should solve this problem.

1.2.3 Photo name patterns

File name collision might be a nightmare in some applications. In one case, the person in charge of time-lapses had to use a software, that did not support naming patterns, so the files were coming out appended with an index. This only worked until the application was closed. So he had to manually check what was the last index so that the files won't overwrite. In another case, he forgot to change the index, so a 2 week time-lapse was simply overwritten without his notice.

Allowing the user to create a custom file pattern is important for making sure that nothing gets erased. It would also allow the user to easily create batch time-lapses every month/week.

1.2.4 Web Client

The biggest problem with the existing applications is that it only offers a desktop version. Meaning, that the only way to remotely control a camera from home, is to use an application like Remote Desktop to control the whole computer. This is still a satisfactory solution. But for some cases it becomes absolutely impossible, as it requires a relatively high bandwidth for relatively little data. For some experiments, in tunnels with no internet coverage modems are a pretty viable solution. But using remote desktop every week through modems that are barely able to connect to the internet is not the most realistic solution. Therefore, a web client would probably be the best way to approach this problem. The client side will represent a web application, a minimalist layer used for communication with the user. The web infrastructure makes the application easy to access. The requirement to run the client side application is to have a modern web browser, for instance Google Chrome, Firefox etc.

1.3 Theoretical Analysis

Considering the complexity of the application, a right amount of research is required in order to construct a workable tool. The final product represents a workable application focused mainly on a small niche for remote long term time-lapses. Further will follow a more detailed description of various aspects of the platform, regarding the technologies best suited for building the application, the concepts behind different tools used in the application and means for solving specific problems.

1.4 Web Services

Building complex systems is not a simple task and usually it consist of smaller logical parts that communicates trough interfaces. To allow further extension of the product, it would be wise to expose an API so that other applications can use it. It is easier to understand how a system works once it is decoupled in multiple independent modules. A small logical unit can be understood faster and better and once it breaks down it is easier to fix it. In the end the point is that the applications should be able to communicate efficiently over the web. Various software are built in different programming languages, are running on diverse operating systems, hence a transparent communication model is needed and at the same time is language agnostic. That is how the web services protocols came to existence. During the time they have evolved into a set of communication standards that offered developers the opportunity to construct decoupled systems.

In order to define the standards, a set of rules are needed to be defined, such as:

- How can a software perform a request to another system;
- What is the set of parameters that should be set in the request;
- What should be format of the request looks like;
- What are the logical parts that the request consists of;
- How should the response be represented;
- How should the errors be described.

As a result, on the market usually persist two main approaches of constructing web services, SOAP and REST. Each approach have their strong points and weaknesses and both heavily relies on HTTP proto-

col, in case of SOAP it also supports other transport protocols.

SOAP is a messaging protocol that have the entire architecture wrapped around XML data representation. In a nutshell, it is a method of communication between two applications. An example of SOAP communication is represented in figure 1.1 The protocol specifies how exactly the HTTP headers should be encoded. A SOAP provider comes in hand with a WSDL file which represents the description of the web service. Things like the possible parameters and their formats, the structure of the message, what is the response format, how it can be correctly accessed. The communication via SOAP protocol is also done using XML formatted files. The structure of the the request and response is documented in the WSDL file and it is validated with the help of XSD schema.

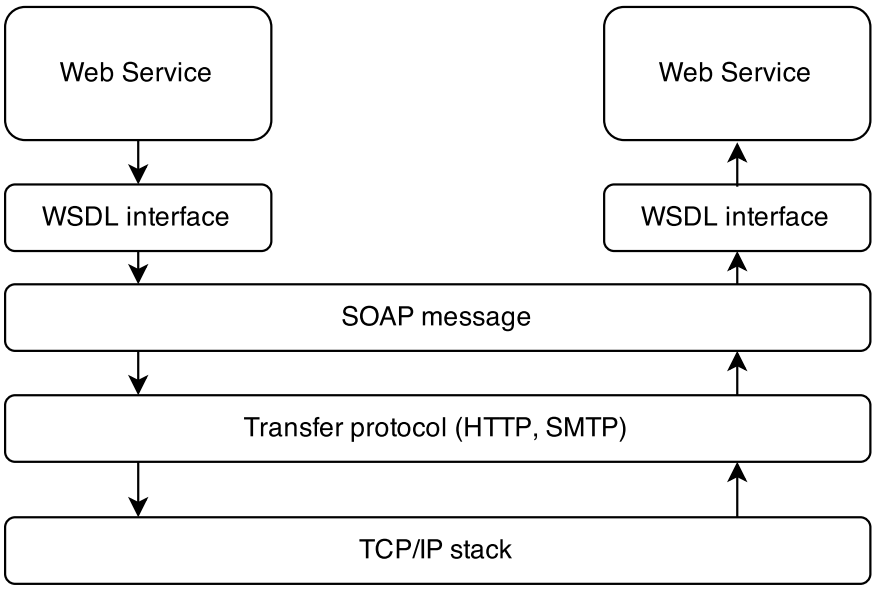

Figure 1.1 – An example of SOAP communication

SOAP represents the next evolving stage between computer communication at he application layer. It was able to replace RPC technologies such as DCOM, CORBA, Java RMI. This reinstatement was highly needed because RPC technologies were brining a complex coupling to the programming language, which is definitely not a good thing. On the other hand the SOAP calls are much slower comparing to native RPC applications. There tends to be firewall latency due to the fact that the firewall is analyzing the HTTP transport. SOAP calls are much more likely to get through firewall servers, since HTTP is typically Port 80 compliant, where other calls may be blocked for security reasons. Since HTTP requests are usually allowed through firewalls, programs using SOAP to communicate can be sure that the program can communicate with programs anywhere. SOAP focuses on exposing pieces of application logic (not data) as services, platform operations. It aims for accessing named operations, each implement some business logic through different interfaces. An advantage offered by SOAP is the WS-Security which adds some enterprise security features. Supports identity through intermediaries, not just point to point (SSL). It also provides a standard for integrity.

REST is a simple stateless architecture that generally runs over HTTPS/TLS protocols. The flexibility is given by assigning resources their URI. The neat part is that it heavily relies on URLs. The REST philosophy is deeply entangled with HTTP protocol implementation, for instance the HTTP verbs GET, POST, PUT, DELETE, PATCH etc, are a part of REST RFC. Although it is a set of guidelines and best

practices, many might understand it in their own way. Therefore, what usually happens is that the whole application is filled only with ”GET” and ”POST” requests. In some cases, some may even use the ”GET” endpoints for creating resources, which will only confuse the reader. So this is a double-edged sword. Hence a lot of debates and discussions are still happening even today. The good part is that it doesn't need tedious descriptor files such as WSDL in order to describe a REST application. Modern frameworks usually offer a way to automatically create documentation, such as ”Swagger”. REST did not only impact the web architecture but it has also affected how we think about software in general. Striving to write stateless code has forever changed our mindsets, because there were times that some would think that it is absolutely fine to have methods with side effects. REST gained a lot of popularity as being a simpler alternative to SOAP and WSDL-based web services. And the most viable example is the implementation of the entire Word Wide Web. One strong thing wielded by REST applications is that the message and response content can be delivered in any format. The most used are XML based format such as HTML, and for API platforms JSON is the most common and handy format, and it has lots of advantages against XML, such as readability, payload size, easy integrable with dynamic languages, it is data oriented.

Nowadays JSON is becoming the preferred format especially for RESTful APIs. Because of the format simplicity sometimes it gets harder to define the communication structure. Which is why JSON community is working now on an elegant format called JSON-api. It simplifies a lot of things in terms of message structure. It resembles to WSDL only it is less restrictive and more intuitive. Another alternative for structuring the message format is HATEOAS. The purpose it aims is defining application state using hypertext.

1.4.1 Data Storage

The whole idea of computer science is wrapped around of ways of manipulating data. From the very start engineers had issues with finding ways to store data. During the time, the hardware evolved and nowadays the disk space does not represent a problem anymore. The actual challenge is how to make interaction with data as efficiently as possible. Interaction represents means of reading, and querying data, effectively saving it on the disk, keeping it consistent and avoid data loss. What if data is related to other data types. How to implement the relationship between the data. How to make possible for multiple users to read and write at the same time. These are the actual problems which are confronted in computer science.

The classical solution to this problem are the RDBMS approaches. It is a common choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personal data, and other applications since the 1980s. Relational databases have often replaced legacy hierarchical databases and network databases because they are easier to understand and use. The relational databases rely on SQL which is a special-purpose language designed for managing data. It is used to query, insert, update and modify data. RDBMS were and still are an irreplaceable solution for managing efficiently relatively small amounts of data. The RDBMS philosophy is built around ACID principle, Atomicity, Consistency, Isolation and Durability. The combination of this four principles has granted such a big success to relational databases.

As mentioned above RDBMS is widely used for lots of applications and successfully solves problems and there hasn't been a better alternative on the market. The competition is applied to different implemen-

tations of databases, such as PosgreSQL, MySQL, OracleSQL, MsSQL, MariaDB. All of them are quite similar, and each has its strong and weak points. The actual problem appears when the Big Data started to get more an more popular. Unfortunately the classical database approach was not enough for the constantly data increase. RDBMS enforces a well defined schema, as a result it gets slower and unmanageable when the amount of data gets bigger.

What developers decided was to loosen up one of the ACID principle and create new brand of databases which have a different structure and would allow storing and working with big amounts of data. This is how the term BASE principal came to life. Basically Available, Eventually Consistent. BASE concept supports the idea of network partitioning, which means that the database will always have a response disregarding the amount of requests at the given time. The catch is what kind of response should it have in case of multiple access to the database. Two solutions were proposed. First one is that the database should always return a result even though it is not up to date. The second solutions is to inform the database user that the service is not available for now. Both solutions have their own applications. Choosing which one to use depends entirely on what it is better suited for the business. The idea of choosing the database model is also presented by CAP theorem. CAP states that when choosing a database you can choose only two of the three features. These are Consistency, Availability and network Partition tolerance. In figure 1.2 is illustrated in more details the CAP and databases categorized based on theorem.

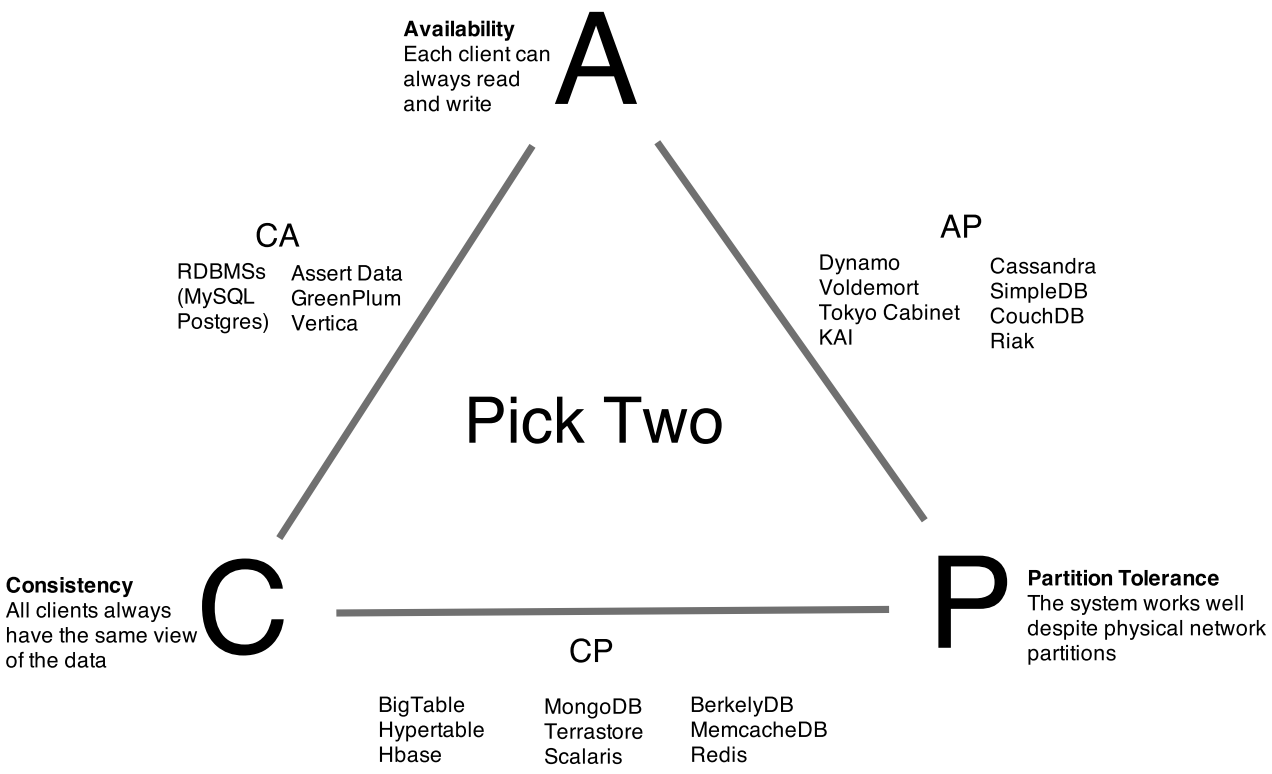

Figure 1.2 – An illustration of CAP Theorem

Along with the implementation of conceptual new database the term NoSQL started to spread. The term came from the fact that the new databases were not using SQL for data management. In reality there are more types of conceptual database, and covering all of them under the same umbrella seems too ambiguous. The most widely used types of databases, not considering RDBMS, are described bellow.

Document Based Database is a new approach of database management. It is used in usual English sense of a group of data that encodes some sort of user-readable information. This contrasts with the value in the key-value store, which is assumed to be opaque data. The basic concept that makes a database document-oriented as opposed to key-value is the idea that the documents include internal structure, or metadata, that the database engine can use to further automate the storage and provide more value.

Document databases contrast strongly with the traditional relational database (RDBMS). Relational databases are strongly typed during database creation, and store repeated data in separate tables that are defined by the programmer. In an RDBMS, every instance of data has the same format as every other, and changing that format is generally difficult. Document databases get their type information from the data itself, normally store all related information together, and allow every instance of data to be different from any other. This makes them more flexible in dealing with change and optional values, maps more easily into program objects, and often reduces database size. This makes them attractive for programming modern web applications, which are subject to continual change in place, and speed of deployment is an important issue. The current most popular implementation of such type of database is MongoDB and CouchDB.

Column Based Database is a database management system that stores data tables as sections of columns of data rather than as rows of data. In comparison, most relational DBMSs store data in rows. This column-oriented DBMS has advantages for data warehouses, customer relationship management systems, and library card catalogs, and other ad hoc inquiry systems where aggregates are computed over large numbers of similar data items.

It is possible to achieve some of the benefits of column-oriented and row-oriented organization with any DBMSs. Denoting one as column-oriented refers to both the ease of expression of a column-oriented structure and the focus on optimizations for column-oriented workloads. This approach is in contrast to row-oriented or row store databases and with correlation databases, which use a value-based storage structure. Such type of database implementations are BigTable, Casandra.

Key Value Database use the associative array (also known as a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection. The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented on top of it. The key-value model can be extended to an ordered model that maintains keys in lexicographic order. This extension is powerful, in that it can efficiently process key ranges. Example of such type of database implementations are Redis, Memcache, Voldemort.

1.4.2 Modern Web Application

Web applications are heavily using HTTP protocol as means of transporting data. HTTP is a stateless protocol. For every request made by a client a TCP socket is opened. The HTTP server receives a request that is handled by the application layer. When the response is sent back to the client, the TCP socket is closed and the transaction ends. The whole chain of events is repeated basically at every user interaction. The result is that a simple web application has a stateless behavior. The application layer of a web applications aims to get rid of statelessness. In 2004 the concept of web 2.0 surfaced. Javascript started to become more popular because it gave the power to animate the pages and create a more humane UX. The magic was behind the

AJAX technology. The concept introduced by AJAX was making a web page run asynchronous requests and make live partially DOM changes. Developers could create web applications which did not require full page reload at while interacting with a web page. Successfully implementation of this concepts are Facebook, Gmail, Twitter etc. AJAX, JQuery and other Java Script technologies brought web applications one step closer to the desktop applications experience.

Nowadays the single page applications are becoming a hot topic. The main reason is that they are able to offer more native application like experience to the user. This is hard to do with other approaches. Supporting rich interactions with multiple components on a page means that those components have many more intermediate states. Server side rendering is hard to implement for all the intermediate states. Small view states do not map well to URLs.

Single page applications are distinguished by their ability to redraw any part of the UI without requiring a server round trip to retrieve HTML. This is achieved by separating the data from the presentation of data by having a model layer that handles data and a view layer that reads from the models. Interaction with the single page application often involves dynamic communication with the web server behind the scenes.

Here are enumerated a set of technologies that helps of building single page applications:

– Ember;
– Angular;
– React;
– Meteor;
– Marionette.

Due to the fact that single page applications have a rich functionality, they also include a complex architecture. For instance in figure 1.3 is illustrated the conceptual structure of Ember framework. It is hard to wrap the head around the structure, but once there is a basic understanding of the logical layers, building applications is a joy for a developer.

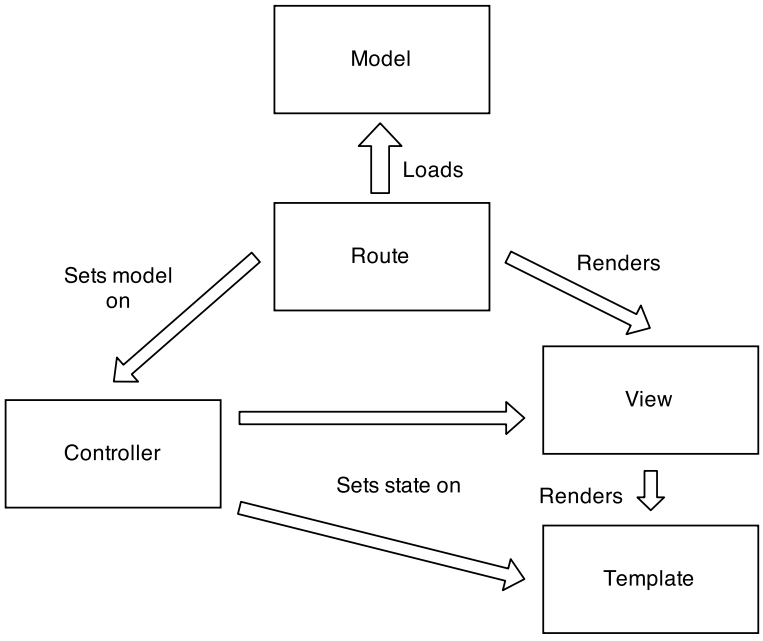

Figure 1.3 – Ember framework architecture

But for simple applications that do not require these features, Angular or React for example might be an overkill. As new updates come out, supporting the UI might get more and more expensive. For an application with just a few buttons that the user should only interact a few times a year, simple HTML and Javascript should be just enough.

Until now everything discussed was related to building the front part of a modern web application. But a web application usually consist also from the backed part. The HTTP application that listens to client requests. For building one there are a lot of frameworks which allows to scaffold a prototype. MVC based frameworks are powerful and provides lots of functionalities out of the box and the good thing is that the majority of frameworks are mature and stable. Here are a list of frequently used solutions:

- Django;
- ASP.NET Core;
- Ruby on Rails;

The mentioned technologies have huge stacks that sometimes are not needed when building a smaller application, or scalable one. Besides for building modern web applications where the client application is developed in a Javascript framework, means that the "V" (view) part from MVC is not needed anymore. Plus there are already on the market lightweight web technologies such as Sinatra, Flask, Node (in combination with express library). This type of application can serve just as good. In case if new module is required by the application, it can be easily added to the micro-framework stack. In ruby this is done by using gemfiles (gems are libraries in Ruby language) were gems can be easily added and installed effortlessly.

1.4.3 Web Sockets

WebSocket is a protocol providing full-duplex communication channels over a single TCP connection. The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011, and the WebSocket API in Web IDL is being standardized by the W3C. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. The WebSocket protocol makes more interaction between a browser and a website possible, facilitating live content and the creation of real-time games. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way a two-way (bi-directional) ongoing conversation can take place between a browser and the server. The communications are done over TCP port number 80, which is of benefit for those environments which block non-web Internet connections using a firewall. Similar two-way browser-server communications have been achieved in non-standardized ways using stop-gap technologies such as Comet. In DSLR Camera Controller web sockets are used to transfer the live preview from the server.

The WebSocket protocol is currently supported in most major browsers including Google Chrome, Internet Explorer, Firefox, Safari and Opera. WebSocket also requires web applications on the server to support it. WebSocket reduces latency. For example, unlike polling, WebSocket makes a single request. The server does not need to wait for a request from the client. Similarly, the client can send messages to the server at any time. This single request greatly reduces latency over polling, which sends a request at intervals, regardless of whether messages are available. WebSocket makes real-time communication

much more efficient. Polling can always be used (and sometimes even streaming) over HTTP to receive notifications over HTTP. However, WebSocket saves bandwidth, CPU power, and latency. WebSocket is an innovation in performance. It is also is an underlying network protocol that enables to build other standard protocols on top of it. It is also a part of an effort to provide advanced capabilities to HTML5 apps in order to compete with other platforms.

2 Software Design

In the current chapter is represented and described the architecture of the DSLR camera controlling system. It contains a set of relevant diagrams modeled in UML language. The diagrams provide a fundamental documentation an description of the system structure and behavior.

The aspect that should be defined is how the user will interact with the application. Therefore a use case diagram was modeled to show the set of available actions offered at the user's disposal. The client part of the application represents a browser web page. There are six main actions a user can perform. The operations can be seen in figure 2.1. When the application is opened, the user can view/manage the connected cameras. The seconds main action is to manage a selected camera's settings. The app should detect all available configurations supported by the camera, if it has any. Next, the user can take a photo and get a live preview as well. Additionally, the user is allowed to configure a failure strategy. And finally, the main action is to allow the user to configure a time-lapse.

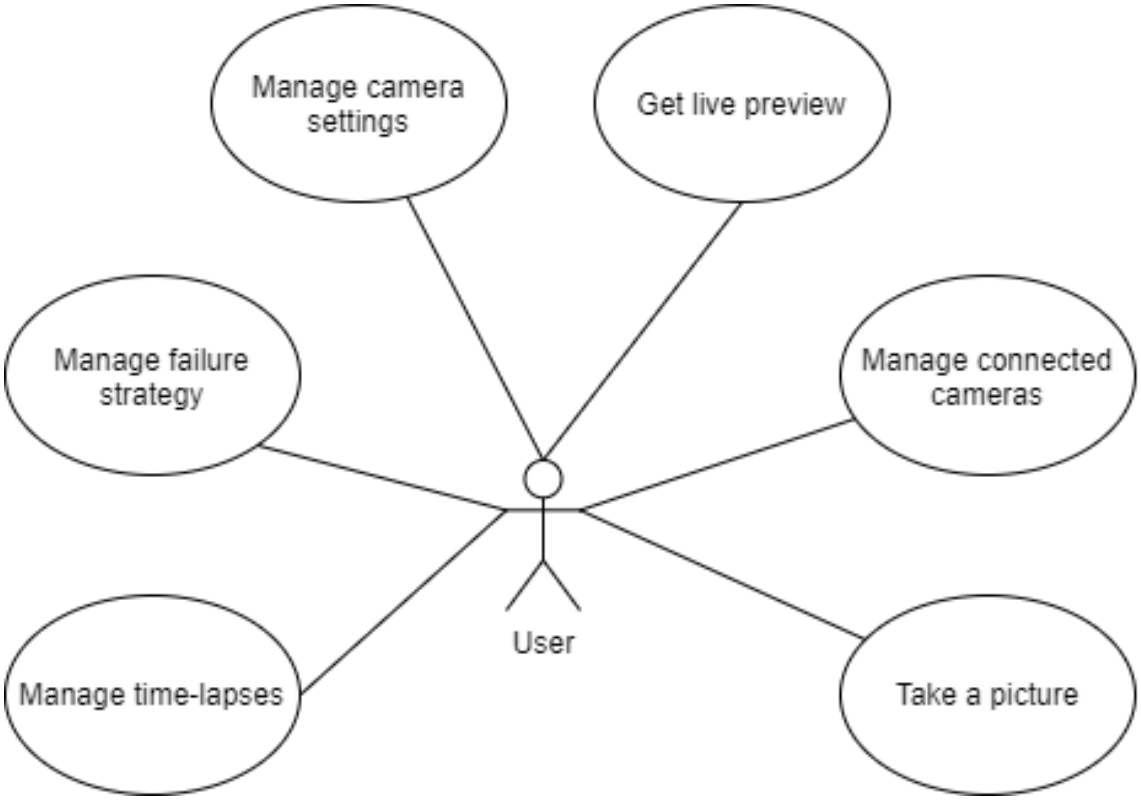

Figure 2.1 – User use case diagram

To offer a detailed overview of how the entire system works, a set of sequence diagrams are provided. They show key parts of the platform and the way they interact. Moreover it is crucial to depict the depth of chain of events happening in the background. For instance the sequence diagram, shown in figure 2.2, specifies what happens when a user simply asks for camera settings. First of all the client browser does an HTTP request in order to get the application page. Now the user is able to interact with the platform. When the client requests the settings on a detected camera, the application first has to ask the selected camera's available configurations. The camera, if it supports this feature, will return a list of settings. These settings also include read-only configurations, like serial number or camera power, so the application filters the

known read-only configurations and returns them to the user.

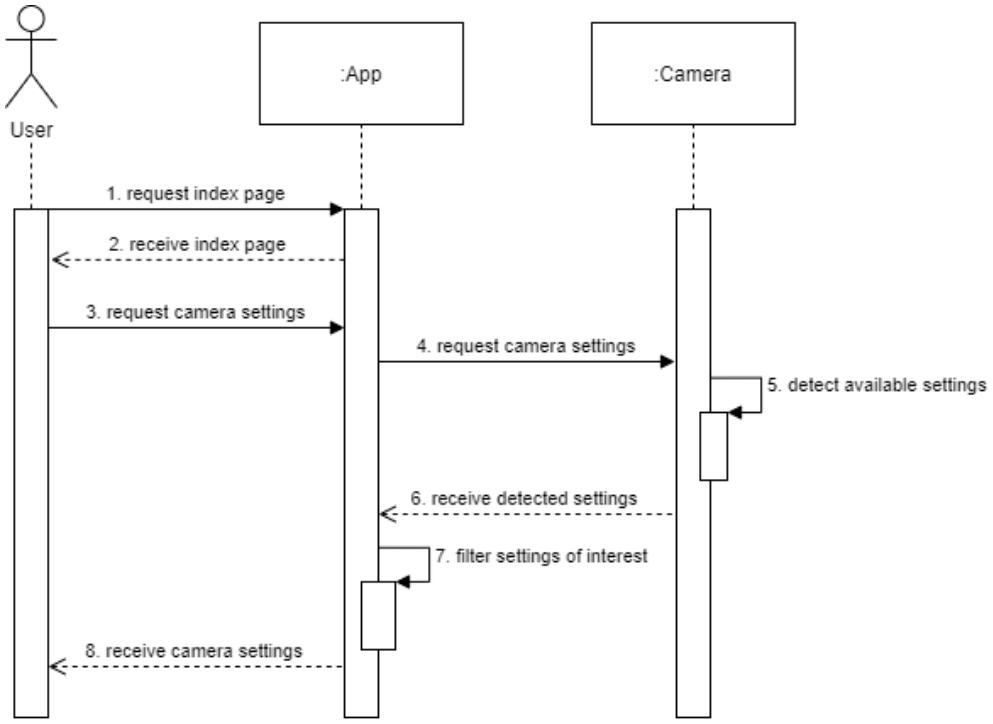

Figure 2.2 – Camera settings sequence diagram

For time-lapses we can distinguish two main steps. Each one is described in the sequence diagrams illustrated below. Their behavior looks a bit similar, nevertheless each has unique characteristics worth point out.

The first step of a time-lapse is the process of scheduling one, that is depicted in figure 2.3. First, the user requests a new schedule. The most important data that the user has to provide is the **cron** expression, which is used for scheduling. The request is then processed in the application, so that it is configured to require the currently connected cameras. If a camera is missing during a capture event, the platform will consider it as a failure and will take the appropriate measures (send email notifications, hard reset, etc.). Next, a special worker that runs separately from the main thread receives the request. The worker then registers the new time-lapse into a persistence mechanism and computes its next run based on all available time-lapses. Reading all the active time-lapses from the database is not the most optimal method, but considering that there will usually be only one active time-lapse, it can be considered satisfactory. At the end, the user receives a success message with the newly created schedule.

The sequence diagram, illustrated in figure 2.4, represents the next step after the user has successfully registered a time-lapse. When the Scheduler triggers a capture event it runs an abstract command composed by the application. This command goes through the list of required cameras and tries to take a picture. After the picture was taken, the scheduler recomputes when it should run the next time. Afterwards, an event that the picture was successfully taken is then published. The application has a special handler for this event which transfers the file if it was configured to do so, then it logs the successful capture and transfer.

The time-lapse is the most crucial feature of the application. So, to better understand how it works, in the figure 2.5 is described through an Activity Diagram a single iteration of the time-lapse. It starts by rescheduling its next run. When a capture event is published, the application requests the time-lapse's

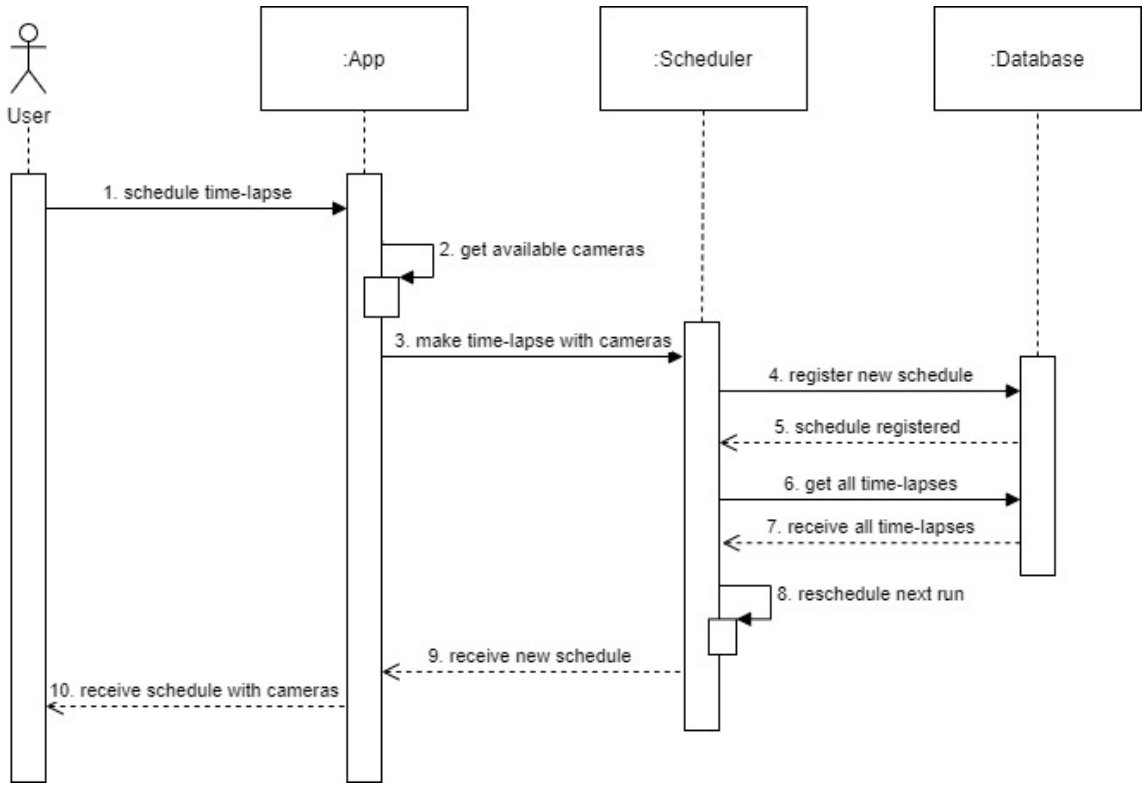

Figure 2.3 – Time-lapse requesting sequence diagram

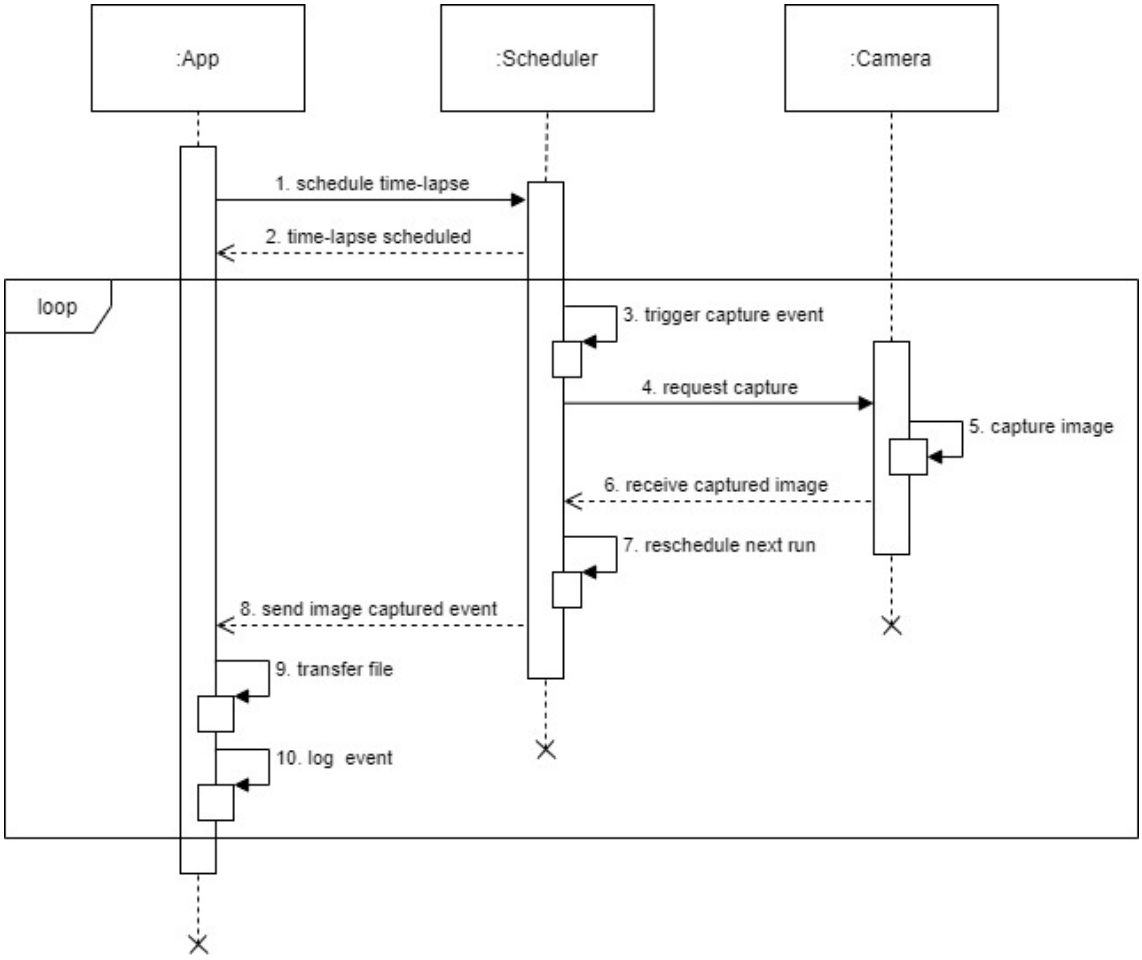

Figure 2.4 – Schedule capture trigger sequence diagram

cameras to take a picture. If a camera is missing, or one of the cameras returned an error, or some error happened, the application registers it as an error. The details of the failure are registered in a special logging system that the user can later query from the web. Afterwards the failure process starts. Otherwise, if there were no errors while taking photos, the photo is logged and then the file transfer routine starts, independent of photo capturing routines. Again, if an error occurs in this step, it is also logged in the system.

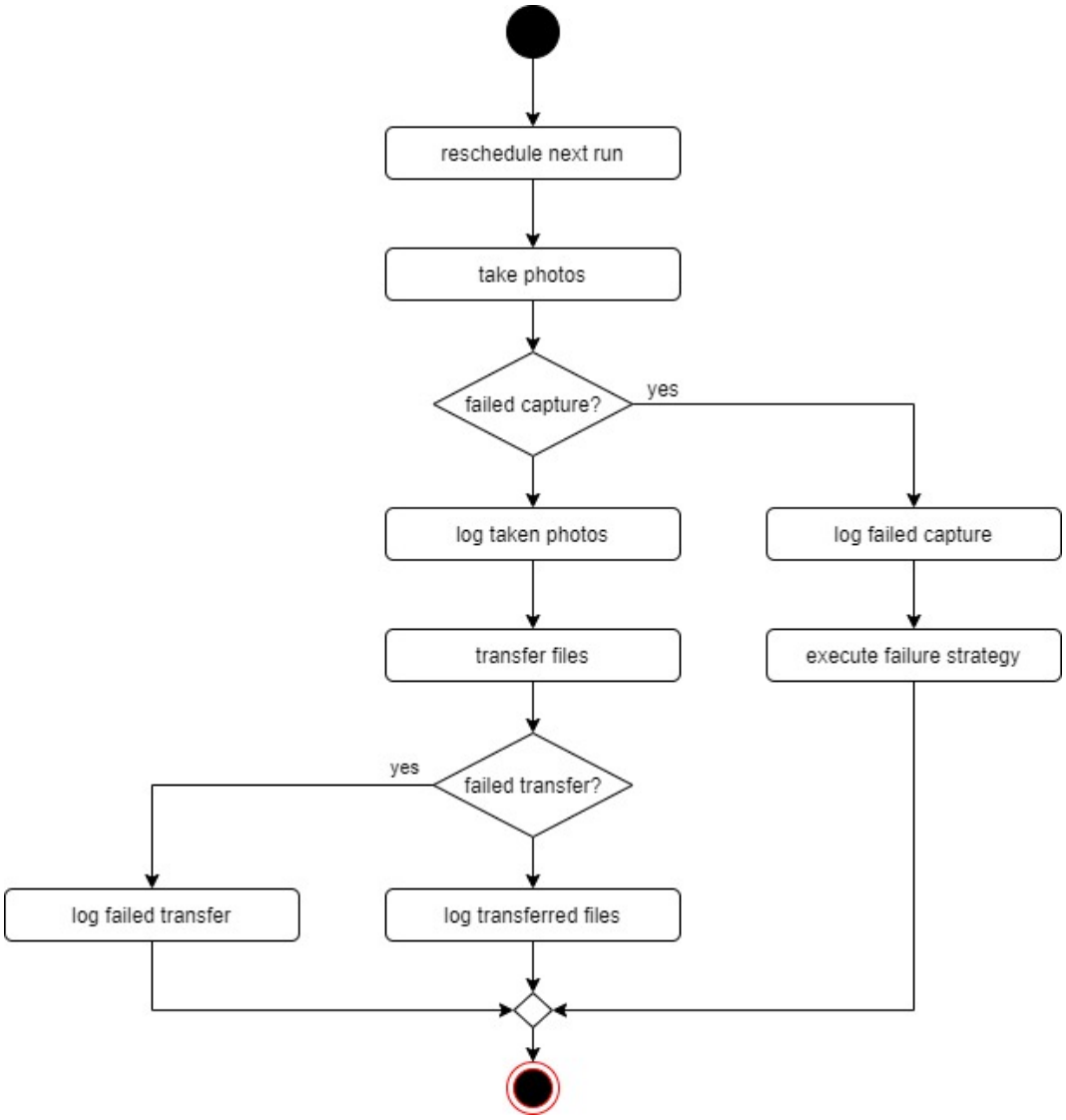

Figure 2.5 – Time-lapse iteration activity diagram

The processing of the capture failures is a key feature of this application as well. Coming up with a few practical solutions that would work across multiple camera models was not an easy task. Nevertheless, in the figure 2.6 is described the activity diagram of how the failures are being handled. This is an extension of the diagram from the figure 2.5. First, if the user has enabled the email notification feature, on failure, the application will fetch all the emails registered for notifications and send them the error message. Then, if hard resetting is enabled, the application will first try to programmatically reconnect to the cameras. In some cases this is good enough, as some camera models simply require a wake-up call. Then, if an Ykush device is detected to be connected to the system, then it activates it as well, which basically re-plugs the cameras. Re-plugging almost always solves any problems it might have encountered. And finally, if system

rebooting is enabled, the program will reboot the system after a consecutive number of failures. This allows the user to reset the USB connections at the expense of processing time. But considering that it is cheaper than an external device for reconnecting the USB connections, this may be viable enough option as well.

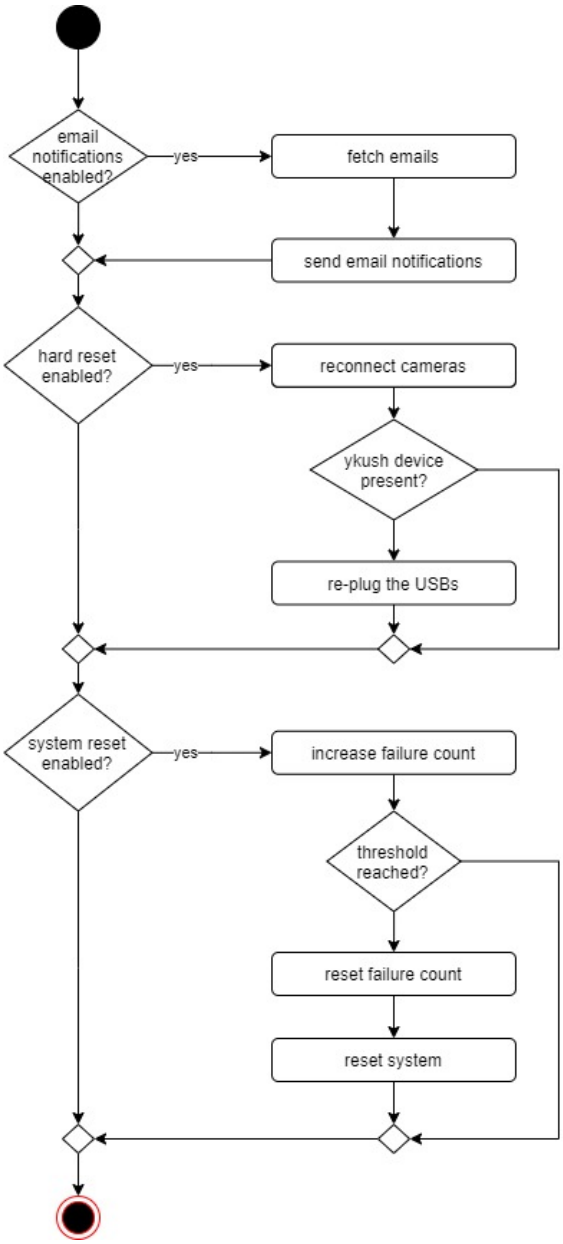

Figure 2.6 – Time-lapse failure activity diagram

In the following part of this chapter is described the most important classes of the DSLR Camera Controller application through a few class diagrams. Most of them are related to the camera and time-lapse management. Information about the system components interaction is not enough for understanding the tool. The class diagrams deliver information under a higher level of granularity, hence the system becomes more easy to comprehend.

First in the figure 2.7 is described how the application is using the cameras. The concept of camera is completely abstracted from the core. The whole application simply communicates with an abstract camera that is known to have a few functionalities like capturing an image, setting a configuration, etc.. Binding the application to the fact that the cameras are connected through some ports or that it is using a special library

to communicate with the camera would transform the maintenance of the application into a nightmare. That's why the implementation of the abstract camera and camera manager sits in the infrastructure level of the application. To allow testing of some functionalities without any cameras connected, some fake cameras were introduced with a few predefined configurations like output image, available setting, etc.. CameraManager works with cameras, but the implementations are aware what kind of cameras they are working with, so it is not quite the Bridge design pattern implemented here.

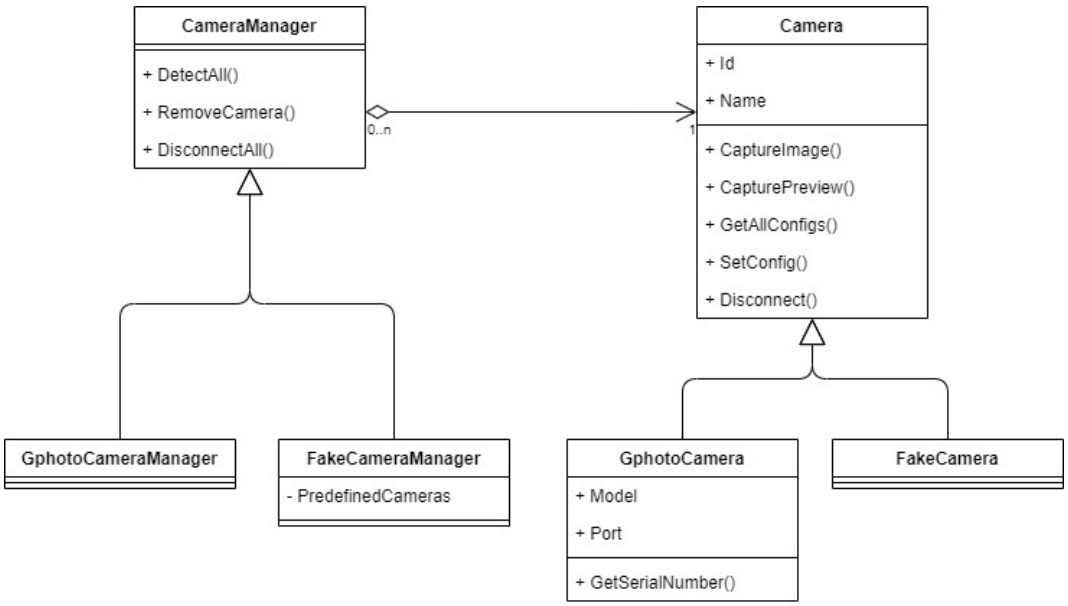

Figure 2.7 – Camera management class diagram

Next follows a class diagram on how a time-lapse is persisted on the platform. In the figure 2.8 is represented what data is required for a time-lapse. It is composed of a nullable schedule and has reference to the required cameras. If the schedule is null, then it means that the time-lapse has been put on hold. The separation of the time-lapse from the schedule offers the possibility to reuse the scheduling logic for other features. One of the requested feature was to email a monthly report on the progress of the application.

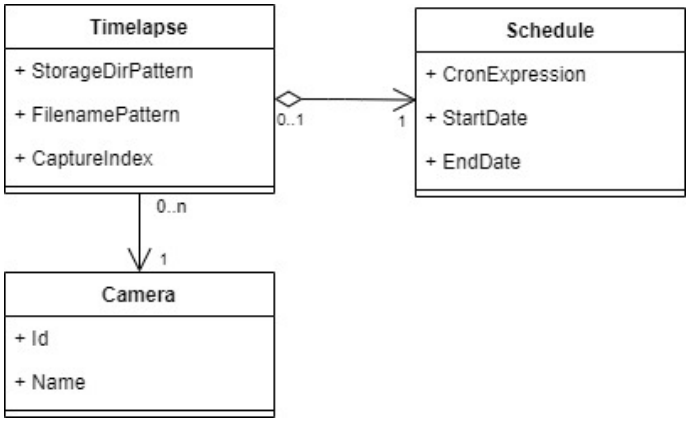

Figure 2.8 – Time-lapse data class diagram

3 Implementation Part

In the previous chapters were discussed the concepts covered by DSLR camera controlling project. The research of the ideas is crucial given that the applications aims a niche that is relatively new in the market. To be sure that the platform development start from the right foot, a thorough architecture design, modeled in UML language, was provided in the previous chapter. What follows now is the description of the implementation part. An exhaustive description of every step will be given, including code snippets, the technologies used and the reason of their choice.

3.1 Used Tools

In order to achieve a better performance in this application development, there were choose specific tools that are offering the possibility to implement and extend the application's functionality.

PyCharm

PyCharm brings you a smart coding assistance for Python languages, Javascript, HTML and CSS. Enjoy code completion, powerful navigation features, on-the-fly error detection, and refactoring assistance for all of these languages. PyCharm provides advanced coding assistance for Django, Angular, Javascript and more. All in one IDE. The IDE analyzes your project to provide the best code completion results for all supported languages. Hundreds of built-in inspections report any possible issues right as you type and suggest quick-fix options. PyCharm helps you get around your code more efficiently and save time when working with large projects. Jump to a method, function or variable definition in just one click, or search for the usages.It also supports advanced features like remote SHH project development and a lot more.

PyCharm provides powerful built-in tools for debugging, testing and tracing your server-side and client-side applications. With minimum configuration required and thoughtful integration into the IDE, these tasks are much easier with PyCharm. Place the breakpoints, step through the code, and evaluate expressions – all without leaving the IDE.

PyCharm integrates with popular command line tools for web development, providing you with a productive, streamlined development experience without using the command line. PyCharm (community edition) is built on top of the open-source IntelliJ Platform, which JetBrains have been developing and perfecting for over 15 years. Enjoy the fine-tuned, yet highly customizable experience it provides to fit your development workflow. PyCharm provides a unified UI for working with many popular Version Control Systems, ensuring a consistent user experience across git, GitHub, SVN, Mercurial, and Perforce. PyCharm is extremely customizable. Adjust it to perfectly suit your coding style, from shortcuts, fonts and visual themes to tool windows and editor layout. [1]

gPhoto

gPhoto is a set of software applications and libraries for use in digital photography. gPhoto supports not just retrieving of images from camera devices, but also upload and remote controlled configuration and capture, depending on whether the camera supports those features. Released under the GNU Lesser General Public License, gPhoto is free software.

gPhoto supports more than 2500 cameras as of June 2019. It is cross-platform, running under Linux, FreeBSD, NetBSD and other Unix-like operating systems.

gPhoto has support for the Picture Transfer Protocol (PTP) and will also connect to devices that use the Media Transfer Protocol (MTP). Many cameras are not supported by gPhoto, but have support for the USB mass storage device class, which is well-supported under Linux.

gPhoto supports camera tethering control, preview, viewfinder in PTP or camera specific protocols on numerous cameras. [2]

SSH

For a quick, cheap and secure connection with the system that has the cameras connected, SSH brings all the required tools to ease the use and management of the DSLR camera controlling system .

Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line, login, and remote command execution, but any network service can be secured with SSH.

SSH provides a secure channel over an unsecured network by using a client–server architecture, connecting an SSH client application with an SSH server. The protocol specification distinguishes between two major versions, referred to as SSH-1 and SSH-2. The standard TCP port for SSH is 22. SSH is generally used to access Unix-like operating systems, but it can also be used on Microsoft Windows. Windows 10 uses OpenSSH as its default SSH client and SSH server.

SSH was designed as a replacement for Telnet and for unsecured remote shell protocols such as the Berkeley rsh and the related rlogin and rexec protocols. Those protocols send information, notably passwords, in plaintext, rendering them susceptible to interception and disclosure using packet analysis. The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet, although files leaked by Edward Snowden indicate that the National Security Agency can sometimes decrypt SSH, allowing them to read, modify and selectively suppress the contents of SSH sessions. [3]

YKUSH Yepkit

YKUSH boards allow the user to selectively switch ON and OFF the power of each of the USB devices connected to the Hub downstream ports. The control is done using an application in the host system (e.g., PC to which the YKUSH board is connected. Switching ON/OFF a YKUSH downstream port with a device connected to it has the same effect as to physically Connecting/Disconnecting a Device to a port of a typical USB hub.

The switching control is performed by sending commands to the on-board micro-controller (YKUSH board control unit), which is seen by host as a HID USB device not requiring the installation of specific drivers. The communication between the host and the board control unit is based on a simple message protocol. The communication protocol is documented and detailed in the Product Manual available for download bellow, in the Documents and Resources section. [4]

3.2 Camera abstractions

Dependency Inversion principle is as simple as it is important: High-level modules, which provide complex logic, should be easily reusable and unaffected by changes in low-level modules, which provide utility features. From Bob Martin's definition, it mentions that abstractions should not depend on details.

Details should depend on abstractions [5] [6]. An important detail of this definition is, that **high-level** and **low-level** modules depend on the abstraction.

Following this principle, it was important to distinguish which parts of this application should be treated as details. It may not be obvious at first, but after a through investigation the camera and all its management can be treated as an abstract concept. Following the same principle, we can distinguish a few abstractions worth noting:

- Camera & Camera management (gPhoto);
- Camera reset logic (Ykush);
- Scheduling (Django scheduling);
- File transfer (rsync);

From an architectural point of view, a DSLR camera is a detail. The fact that gPhoto tool is used to communicate with cameras is also a detail. To avoid tightly coupling the **Business** to such details, making it hard to change, it's important to abstract the core logic from it. In the **listing 1** it is represented the general interface of a camera. The 2 most important functionalities are: get/set config and capture image. It also has preview capturing for getting a quick lightweight preview of the camera, useful for live previewing.

```
class Camera:
    @property def id(self):
    @property def name(self):
    @property def summary(self):

    def disconnect(self):

    def list_configs(self) -> Iterable[str]:
    def get_config(self) -> CameraConfig:
    def set_config(self, config_fields: Iterable[CameraConfigField]):

    def capture_preview(self) -> memoryview:
    def capture_img(self, storage_dir, filename_prefix) -> str:
```

Listing 1 – Camera Interface

To facilitate the communication with all the connected cameras, a Camera Manager is introduced. It is designed to abstract the orchestration of cameras in a way that the user would not need to know anything about the synchronization problems gPhoto has. In the **listing 2** it is represented a rough summary of what this manager is expected to be able to do.

```
class CameraManager:
    @property def cameras(self) -> Iterable[Camera]:

    def detect_all_cameras(self):
    def get_camera(self, camera_id) -> Camera:
    def remove_camera(self, camera_id):
```

```
7       def disconnect_all(self):
```

Listing 2 – Camera Manager Interface

Inspecting the requirements, the client was in a need for a small but powerful functionality. He would need to quickly detect all connected cameras. Therefore, a *detect all cameras* functionality is requested from the manager. After the cameras are detected, the *cameras* property should be filled with *Camera* instances. This would allow the user to iterate through all the detected cameras an run the required functionality.

To have a serializable method of communicating between the user and the system, a unique Id is required. The concrete cameras are expected to expose a unique identifier which would allow the user to query a specific camera from the manager. The user would then be able to remove a camera from the manager if required.

3.3 Camera implementations

Being able to test the application without any connected cameras brings a large range of benefits, some of them would be:

- Independent development of the client side;
- Stress testing;
- Less budget consumption for camera renting;

This brings us to the conclusion that a fake camera would perfectly fit for the development of this application. A more technical name for such a concept would be a *test stub*. In advanced polymorphism computer science, test stubs are programs that simulate the behaviours of software components that a module undergoing tests depends on. Test stubs are mainly used in incremental testing's top-down approach [7]. Additionally, these stubs will also help in testing if the right abstractions were made, helping in developing a more robust and reliable architecture which would drastically help in further development.

In the **listing 3** is represented a piece of code of how these stubs work. Mainly, the constructor receives all the required information, otherwise it auto-generates predefined data. For example, when a preview or image is requested, it simply draws two triangles using the *pillow* library for drawings.

```
1  def capture_img(self, storage_dir, filename_prefix) -> str:
2      if not os.path.isdir(storage_dir):
3          raise Exception('Path: "{0}" does not exist'.format(storage_dir))
4
5      img = self._stub_img()
6      filename = '{0}.jpeg'.format(filename_prefix)
7      file_path = os.path.join(storage_dir, filename)
8
9      img.save(file_path, format='JPEG')
10     return file_path
11
12 @staticmethod
```

```
def _stub_img():
    # Draw 2 triangles
    img = Image.new('RGB', (255, 255))
    draw = ImageDraw.Draw(img)
    draw.polygon([(20, 10), (200, 200), (100, 20)], fill=(255, 0, 0))
    draw.polygon([(200, 10), (200, 200), (150, 50)], fill='yellow')

    return img
```

Listing 3 – Stub image capture

Next, in the **listing 4** it is represented how *gPhoto* tool is used to detect all detected cameras.

```
def detect_all_cameras(self):
    self.disconnect_all()

    with self._gp_lock:
        cameras_name_and_port = gp.check_result(gp.gp_camera_autodetect())

        port_info_list = gp.PortInfoList()
        port_info_list.load()

        for name, port in cameras_name_and_port:
            gp_camera = gp.Camera()
            idx = port_info_list.lookup_path(port)
            port_info = port_info_list[idx]
            gp_camera.set_port_info(port_info)

            camera = GpCamera(name, port, gp_camera)
            self._cameras_dict[camera.id] = camera
```

Listing 4 – gPhoto detect all cameras

gPhoto is a very fragile library. It may sometimes work on OSX but fail to run some simple commands on the Linux environment. To avoid any unforeseen bugs and to simply ease the development process, a synchronization system was introduced. This system would only allow a single caller to communicate with the cameras, otherwise the library usually fails (this is in regard to *_gp_lock*).

Detecting all cameras is quite simple using *gPhoto*. The first step is to ensure that all cameras are disconnected. Next, it gets all the camera names and ports. Afterwards, information about the available ports are loaded. Finally, depending on the port and the loaded information, the *GpCamera*s are created. *GpCamera* is a concrete implementation of *Camera*. Now, the user can simply iterate through the cameras and execute the required functionality.

Now, the most important functionality is represented in the **listing 5**. The entire application relies on this specific functionality, therefore it became slightly sophisticated.

```
def capture_img(self, storage_dir, filename_prefix) -> str:
```

```
    with self._gp_lock:
        with self.GpConnection(camera=self):
            if not os.path.isdir(storage_dir):
                raise Exception('Path: "{0}" does not exist'.format(storage_dir))

            file_device_path = self._gp_camera.capture(gp.GP_CAPTURE_IMAGE)

            _, file_extension = os.path.splitext(file_device_path.name)
            file_extension = file_extension[1:]
            filename = '{0}.{1}'.format(filename_prefix, file_extension)
            file_path = os.path.join(storage_dir, filename)

            camera_file = self._gp_camera.file_get(
                file_device_path.folder,
                file_device_path.name,
                gp.GP_FILE_TYPE_NORMAL)

            camera_file.save(file_path)

    return file_path
```

Listing 5 – gPhoto capture image

Before starting any communication with the camera, a lock is put in place to prevent any other threads from interfering. Then, a new connection is open. Next, the *gPhoto* library captures an image with the preset settings and saves it on the local storage. Then the next step is to move it on the current system, which is also done through *gPhoto*. When the file is moved, it is given a new name which follows the complex file naming requirements described earlier. And finally, the full path of the new image on the local system is returned.

While image capturing might seem a big overwhelming at first, preview capturing on the other hand is much more simple. In the **listing 6** it is represented how this process is taken care of.

```
def capture_preview(self) -> memoryview:
    with self._gp_lock:
        with self.GpConnection(camera=self):
            camera_file = gp.gp_camera_capture_preview(self._gp_camera)
            file_data = gp.gp_file_get_data_and_size(camera_file)
    return memoryview(file_data)
```

Listing 6 – gPhoto capture preview

Since the file is lightweight and short living, it is read in memory and returned, therefore no additional tasks are required. This makes live previewing a lot faster than trying to store it on the local storage.

3.4 Scheduling

Creating a schedule was never an easy task. Initially the client required a very specific use case: ”A schedule that triggers camera capture every minute or so from Monday to Friday, from 8 AM until 8 PM”. Using timers for this use case would easily solve the problem. But then, the client may want a whole different type of schedule. For example, he may require a schedule that would trigger the capture event every second day, or a schedule with lunch breaks, etc.. This would create a whole lot of additional unnecessary complications. For every new type of schedule, the scheduling part would need to be modified.

To not reinvent the wheel, the best tool for this job became *cron*. This software utility is a time-based job scheduler in Unix-like computer operating systems. Users that set up and maintain software environments use cron to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals. It typically automates system maintenance or administration—though its general-purpose nature makes it useful for things like downloading files from the Internet and downloading email at regular intervals. The origin of the name cron is from the Greek word for time chronos. Cron is most suitable for scheduling repetitive tasks. Scheduling one-time tasks can be accomplished using the associated *at* utility [8].

Using the *apscheduler* python library allows the use of this very tool to create schedules. The **listing 7** contains the definition of a cron schedule used in this application.

```
class CronSchedule:
    def __init__(
        self,
        name: str = 'CronSchedule',

        start_date: datetime = None,
        end_date: datetime = None,

        year: str = '*',
        month: str = '*',
        day: str = '*',
        week: str = '*',
        day_of_week: str = '*',
        hour: str = '*',
        minute: str = '*',
        second: str = '*'):
    ...
```

Listing 7 – Cron schedule

Again, to allow the use of different tools other than *apscheduler*, the business logic defines an abstraction over the scheduling process which is represented in the **listing 8**.

```
class Scheduler:
    def start(self):
    def add_job(self, func, func_kwargs, cron_schedule: CronSchedule):
    def modify(self, cron_schedule: CronSchedule):
```

```
    def delete(self, cron_schedule_id: int):
```

Listing 8 – Cron scheduler

The start method would be used for starting all the existing schedules. For example *apscheduler* works by reading and writing to a predefined persistence mechanism. In DSLR camera controlling system it is configured to use a lightweight SQLite database. This allows persisting the jobs even when the application is forcefully stopped or a power outage happens, which is a critical sale point for this application. As described earlier, the existing applications had trouble automating this part. In the **listing 8** it is described the implementation of the *apscheduler*.

```
class ApsScheduler(Scheduler):
    def __init__(self):
        self._scheduler = BackgroundScheduler()

    def start(self):
        self._scheduler.add_jobstore(DjangoJobStore(), 'default')
        self._scheduler.start()

    def add_job(self, func, func_kwargs, cron_schedule: CronSchedule) -> str:
        job = self._scheduler.add_job(
            func=func,
            kwargs=func_kwargs,
            trigger='cron',
            start_date=cron_schedule.start_date,
            end_date=cron_schedule.end_date,
            year=cron_schedule.year,
            ...)

        return job.id

    def modify(self, cron_schedule: CronSchedule):
        ...

    def delete(self, cron_schedule_id: int):
        for job_id in self._get_all_affected_job_ids(cron_schedule_id):
            self._scheduler.remove_job(job_id)
```

Listing 9 – Aps Scheduler implementation

At the start, it configures to use the *DjangoJobStore* which simply tells the scheduler to use the same persistence mechanism as defined in Django. Basically, instead of directly referencing the library from the core, the core application defines an interface which is latter implemented by a third party library. In other words *ApsScheduler* is a wrapper around library's own scheduler.

Using such a tool brings its own risks and complications though. The user would need to learn a bit how to define cron expressions. For this reason, this part must be thoroughly documented so that it's much easier to get started.

3.5 Web API

For the web part, the application uses the Django framework. It is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secures websites. Django is a Python-based free and open-source web framework that follows the model-template-view (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an American independent organization as a non-profit organization [9].

Each *view* represents a specific use case of the application. Each such use case is separated in its own class making development independent of other views. For resolving the required dependencies, the Dependency Injection principle is used. In software engineering, dependency injection is a technique in which an object receives other objects that it depends on. These other objects are called dependencies. In the typical ”using” relationship the receiving object is called a client and the passed (that is, ”injected”) object is called a service. The code that passes the service to the client can be many kinds of things and is called the injector. Instead of the client specifying which service it will use, the injector tells the client what service to use. The ”injection” refers to the passing of a dependency (a service) into the object (a client) that would use it [10]. For dependency injection, the application uses *pinject* tool which is a is a dependency injection library for python [11].

The **listing 10** represents how Django uses the application's core functionality to expose an API. The routing for this view is latter configured in a Django setting file.

```
class CameraCaptureImgAndDownload(View):
    camera_ctrl_service = obj_graph().provide(CameraCtrlService)

    def get(self, request, *args, **kwargs):
        camera_id = kwargs['camera_id']

        try:
            capture_dto = self\
                .camera_ctrl_service\
                .camera_capture_img_and_download(camera_id=camera_id)

            return FileResponse(
                open(capture_dto.real_file_path, 'rb'),
                filename=capture_dto.download_filename)

        except CameraNotFoundException:
            return CameraNotFoundApiException(camera_id=camera_id)

        except CameraException as e:
            return HttpResponseServerError(content=str(e))
```

Listing 10 – Capture image View

First, this view resolves the *CameraCtrlService* from the object graph (*pinject*). Then it tries to call the capture image functionality from the service, returning a FileResponse which is basically a downloadable file for the end user. In case of a failure, like camera is not found or an internal camera error, a respective

error is returned to the user.

For live previewing a slightly different approach was taken. Instead of simply returning the current preview, a *StreamingResponse* is returned, which allows endlessly streaming camera previews. The **listing 11** represents the view for the live preview source.

```
class CameraPreviewSource(View):
    camera_ctrl_service = obj_graph().provide(CameraCtrlService)

    def get(self, request, *args, **kwargs):
        camera_id = kwargs['camera_id']

        return StreamingHttpResponse(
            self._preview_generator(camera_id=camera_id),
            content_type="multipart/x-mixed-replace;boundary=frame")

    def _preview_generator(self, camera_id):
        try:
            while True:
                frame = self\
                    .camera_ctrl_service\
                    .camera_capture_preview(camera_id=camera_id)

                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n'
                       + frame.tobytes()
                       + b'\r\n')

        except CameraNotFoundException:
            return CameraNotFoundApiException(camera_id=camera_id)

        except CameraException as e:
            return HttpResponseServerError(content=str(e))
```

Listing 11 – Live preview View

Sending a big file in Django would best be to stream it in chunks. Similarly, instead of streaming chunks of a file, it streams camera previews in an endless loop, which is automatically canceled on user's request. As a result, the HTML for this functionality becomes very simple and transparent, which can be seen in the **listing 12**.

```
<img src="{% url 'api/camera_preview_source' camera_id=camera.id %}">
```

Listing 12 – Live preview HTML

4 Testing and documenting the system

Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time. There is nothing better than a strongly tested software. Imagine that you spent a lot of time developing something that is not gone used by real users because it's buggy. Since we assume that our work may have mistakes, hence we all need to check our own work.

However some mistakes come from bad assumptions, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done. One of the most important things to do if you want to have a successful release and to promote an application and make it popular is to permanently monitor the crash rate of the application and keep it as small as it's possible. Because nobody likes when the application is crashing or when something is not working as it's supposed.But how to keep the crash rate of an application small? Ideally, we should get someone else to check our work because another person is more likely to spot the flaws. [12]

But usually, in startups, there is not enough money to pay for human resource and to have quality engineers that will test the application manually and will assure it's quality. That's why a good solution would be to have a crash-free application by writing unit tests, instrumentation tests, snapshot tests which were done for the system and will be described below.

Application's views

In this section it will be explained how the application works and what the user is able to do with this application. It will be explained all the use cases, starting from camera listing to time-lapse management.

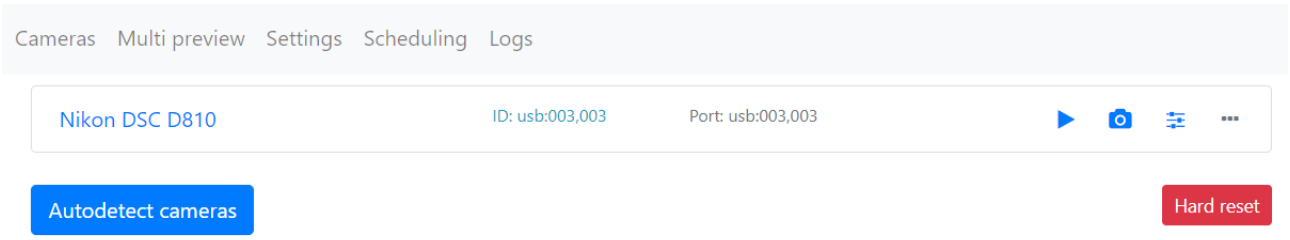

Figure 4.1 – Application's main page

In the Figure 4.1 it is represented the first screen the user sees. On the application start-up, the system automatically auto-detects all the cameras, so the user should see a list of the detected cameras. Each listing comes with its own set of buttons, like live preview, image capture, camera settings and camera reconnect. It also displays its Name, ID and the connected port. From this page, the user can re-detect the cameras or hard reset the ports. The hard resetting logic works by triggering the Ykush board if available. This board would then reset the USB ports which effectively re-plugs the USB ports.

Figure 4.2 shows the page containing the live preview of a camera. On the right side is a list with all the favorite settings. These are settings which the user may choose to display on the live preview page.

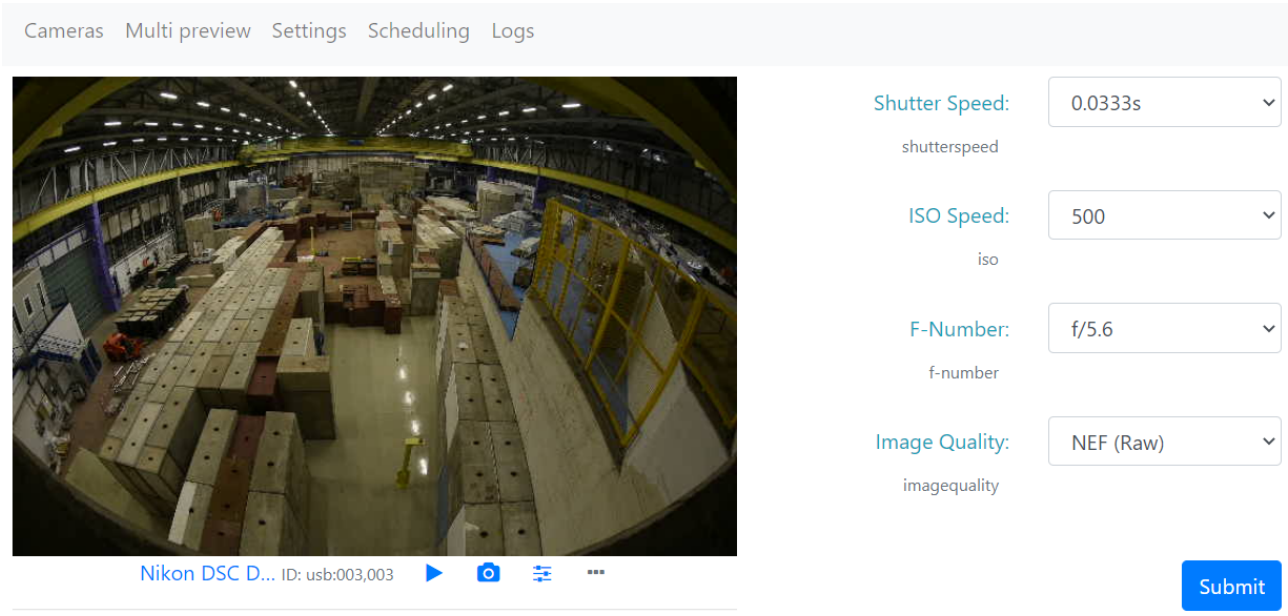

Figure 4.2 – Camera live preview

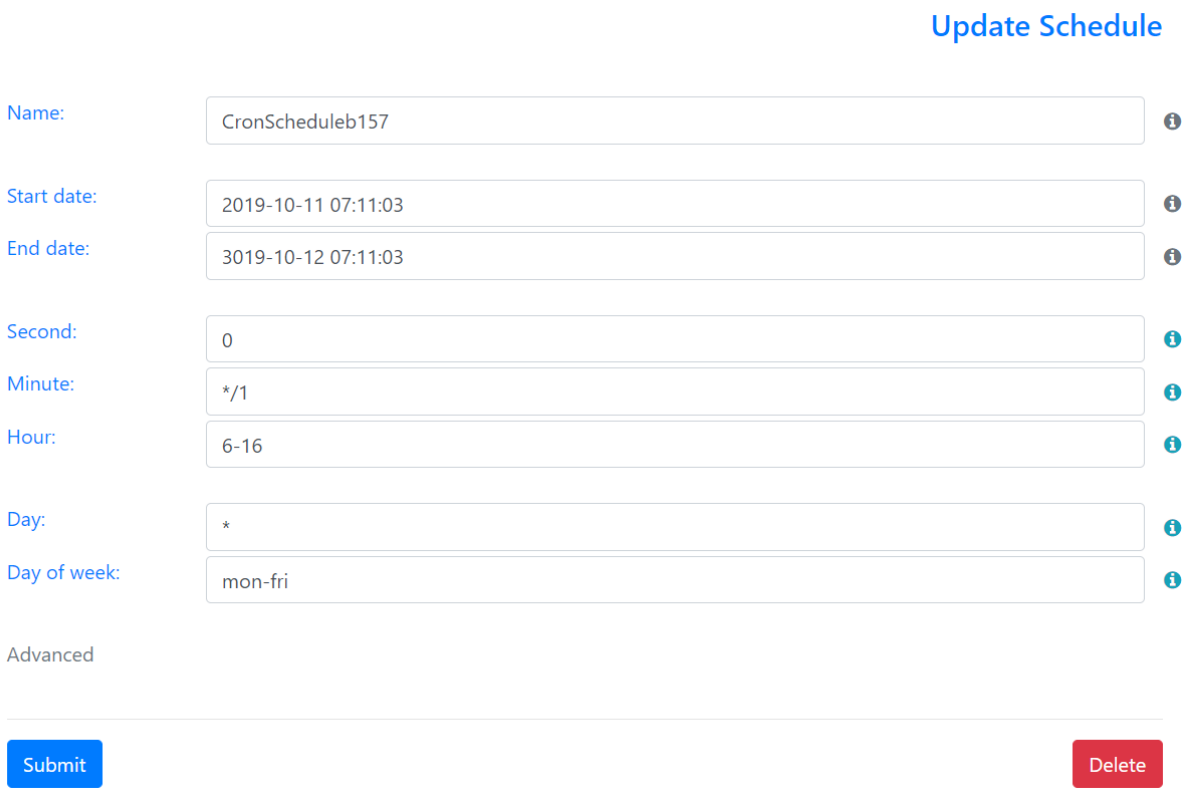

Figure 4.3 – Cron scheduling

In the Figure 4.3 it is represented the page where the user can configure a schedule. It's not seen in this figure, but below the Submit and Delete buttons the user can find documentation on how to correctly create or modify a schedule.

In the Figure 4.4 it is represented the page for Time-lapse configuration. The user can choose the storage directory format and the filename format. Not included in this figure, but if the user scrolls down,

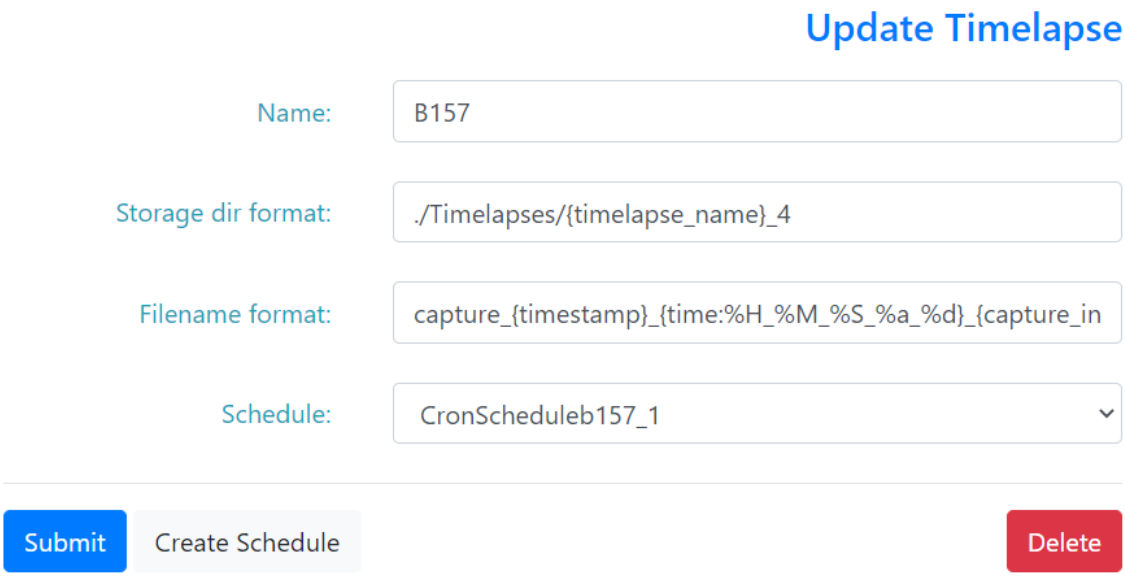

Figure 4.4 – Time-lapse scheduling

he would find some documentation on how to use these features. In this figure for example, the user chose to store the time-lapse photos with a filename that will contain the timestamp, the current time (in the specified format) and capture index. This format produces images in the following form:

```
B157_4/capture_1591376340024355_16_59_00_Fri_05_94639.nef
```

In the Figure 4.5 it is represented the page containing the stream of logs. Here the user can search and filter specific logs. For example he may choose to display all the errors (which is usually the case). This allows the user to see the progress of how the application is doing.

And finally, in the Figure 4.6 the user can configure any of the available settings. Since *gPhoto* works on a large variety of camera models, some of the displayed fields may be only read-only, that's why when a setting fails to change, an alert next to that setting appears.

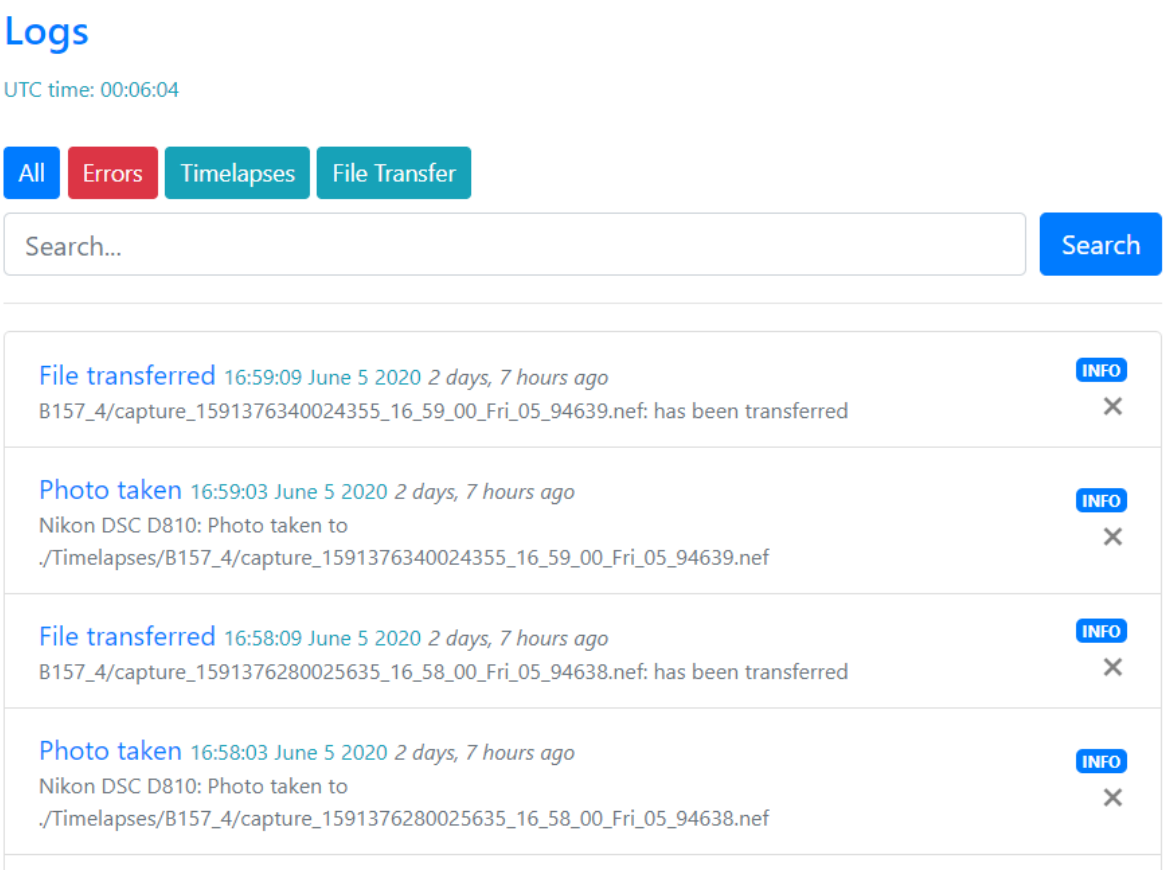

Figure 4.5 – Logs stream

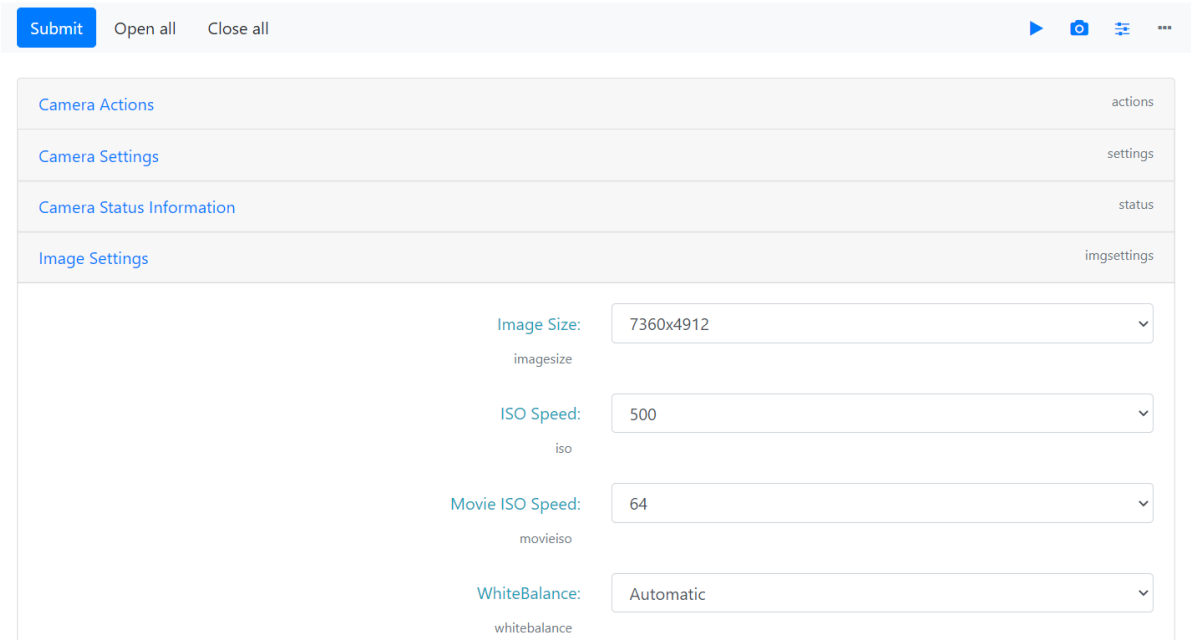

Figure 4.6 – Camera settings

5 Economic Analysis

5.1 Project description

This section covers the analysis of the developed product from an economic point of view. This product is a web application which whose aim is to facilitate the process of automatically taking long term time-lapses with DSLR Cameras. This section describes aspects like the usefulness of the product, market research, project and time schedule as well as the profitability of the implemented solution.

5.2 Project time schedule

The success of the project depends on the right establishing of time constrains of involved activities as well as resources involved in those activities. For the accomplishment of a project it is necessary to establish a schedule. For the development of the DSLR Camera Controller application, Agile project management is applied to offer flexible and iterative method of designing the application. It goes in 6 stages: requirements, design, develop, test, deploy and feedback. These steps are represented in the figure 5.1. It is a set of repetitive actions which were proven to have a substantial effect on the development of a product.

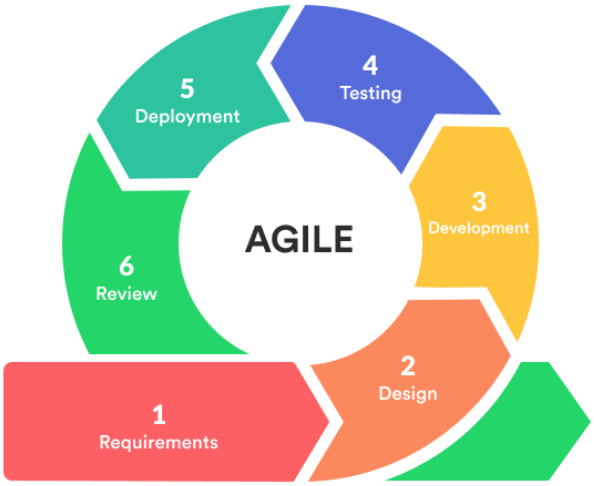

Figure 5.1 – Agile development

5.2.1 Objective determination

The main objective of the following project is to provide a complete and functioning application for photographers. Otherwise without a finished product there can be no profit. More to that, it is important to market the application and get exposed to a large audience in need. This can be done by targeting first the company that has originally proposed this project.

As it was originally a project requested by the photographic department of CERN (the European Organization for Nuclear Research), this company will prove to be the best first target. Additionally, CERN is still struggling with the automation of taking photos in general, so it can prove to be a client that may offer other projects in the same field in the future. For example, there's a problem in automation of the

process of generating a 3D model for VR of the Large Hadron Collider. A project was proposed to create a robot which would take photos of the entire accelerator, then through photographic technologies generate a 3D model. Considering the similarities, it is a project worth considering for future development.

5.2.2 Time schedule establishment

Time management is an important factor in determining the success of a product. So, as mentioned above, the project will iterate over 6 steps every sprint. Because the required team is not very big, to enure the maximum performance, the sprints will be divided into one week each. Naturally, the first few sprints will be mostly composed of research tasks, to analyze the available tools currently available. But normally the sprint will start by deciding on a set of stories that the team should manage to finish until the end of the week. This includes the requirements and design stages. Next follows the development stage followed by the testing stage. Once the stories are finished, they are tested on a development environment. Depending on how successful was the testing part, the application is then deployed on a staging environment where the client can interact with the application and leave feedback. Additionally, the team will also make brief morning Standups just to ensure that everyone is on the same page. Depending on how well the team performs, new ceremonies like sprint review and sprint retrospective can be done once in a while. Total duration of the project is computed using (5.1).

$$D_T = D_F - D_S + T_R, \tag{5.1}$$

where D_T is the duration, D_F – the finish date, D_S – the start date and T_R – reserve time. In table 5.1 is presented the first iteration of the project schedule. It uses the following notations:

- PM – project manager
- SA – system architect
- SM – sales manager
- D – developer

Table 5.1 describes the activities that will occur during project development, who is involved into each process and how much time does it take to accomplish a task. Total amount of time spent on the following project is estimated to be 135 days.

5.3 Economic motivation

The following section describes the evaluation of the project from the economic point of view. That includes the total profit, number of potential clients, salaries that have to be paid to employees, revenues that the company gets by commercializing the product. All the costs and prices are given in MDL (Moldavian lei) currency. Tangible and intangible assets, indirect expenses will also be taken into account. Wear and depression in regard to final product will also be computed. It should be mentioned that DSLR Camera Controller is an open source project posted publicly on Github. In production and development, open source as a development model promotes a universal access via a free license to a product's design or blueprint, and universal redistribution of that design or blueprint, including subsequent improvements to it by anyone. Before the phrase "open source" became widely adopted, developers and producers used a variety of other terms. Open source gained hold with the rise of the Internet, and the attendant need for massive retooling

Table 5.1 – Time schedule

Nr	Activity Name	Duration (days)	People involved	Comments
1	Define the project concept and objectives	5	PM, SA, SM, D	It is a common task
2	Perform market analysis	10	PM, SA	Will results into a document describing market analysis
3	Analysis of the domain	15	SA, D	Research of the recognition algorithms
4	Write down requirements and specifications	5	PM, SA, D	
5	System design (UML)	10	PM, SA, D	
6	Database design	5	PM, SA, D	Development database and end-user database schemes
7	Preprocessing and learning part of the implementation	30	PM, SA, D	
8	End-user application development	20	PM, SA, D, SM	
9	Validation of results	10	PM, SA, D, SM	
10	Documentation	5	D	
11	Deployment and testing	10	PM, SA, D	
12	Active marketing	10	SM	
13	Total time to finish the system	135		

of the computing source code. Opening the source code enabled a self-enhancing diversity of production models, communication paths, and interactive communities. The open-source software movement arose to clarify the environment that the new copyright, licensing, domain, and consumer issues created. The entire economical part is done on the presumption that the software will have payed licenses. Either way it is a curios approach to compute all the necessary resources and indexes for developing a project. It opens managerial insights over entrepreneurial ideas.

5.3.1 Tangible and intangible asset expenses

The budget of a project is most of the times what shapes the future development process. Depending on the budget, some additional features can be implemented or otherwise, can be dropped because of insufficient funding. In this section, the budget will be defined and computed so that managing it would be easier in the future. In Table 5.2 are presented all the tangible assets used for developing this product. Tangible assets are defined as a set of assets that have a physical form.

Table 5.2 – Tangible asset expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Mac Book pro	retina display i5	Unit	23000	1	23000
Camera	Sony Alpha-A3000	Unit	8000	1	8000
Camera	Nikon Z6	Unit	32000	1	8000
				Total	63000

The opposite of tangible assets are intangible assets, which are considered nonphysical investments. These assets are presented in the Table 5.3.

Table 5.3 – Intangible asset expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
License	Enterprise Architect Desktop Edition License	Unit	1900	3	5700
License	PyCharm Commercial License	Month	160	5	800
				Total	6500

Below, in Table 5.4 are presented the direct expenses which appeared during this project. These expenses appeared during project development and cannot be included in any of the previous tables, because their value can not be added directly into the budget.

So the total amount of direct expenses in MDL is

$$T_e = 63000 + 6500 + 870 = 70370 \tag{5.2}$$

5.3.2 Salary expenses

This section is concerned about the salaries to employees and various funds that should be paid. The distribution of salaries is the following: project manager - 400MDL, system architect - 450 MDL, sales manager - 300 MDL, developer - 380 MDL. The Table 5.5 presents more thoroughly what would be the expenses for paying the salaries.

Now by having computed all the salaries for the employees, it is time to compute how much to be paid to social services fund, medical insurance fund and the total work expenses by summing up all previous expenses.

This year the social service fund is approved to be 23%, therefore the salary expenses are computed according to the relation (5.3).

$$\begin{aligned} FS &= F_{re} \cdot T_{fs} \\ &= 148700 \cdot 23\% \\ &= 34201, \end{aligned} \tag{5.3}$$

where FS is the salary expense, F_{re} is the salary expense fund and T_{fs} is the social service tax approved each year. The medical insurance fund is computed as

$$\begin{aligned} MI &= F_{re} \cdot T_{mi} \\ &= 148700 \cdot 4.5\% \\ &= 5948, \end{aligned} \tag{5.4}$$

where T_{mi} is the mandatory medical insurance tax approved each year by law of medical insurance and this year it is 3.5%.

So now having computed social service tax and medical insurance tax, it is possible to compute total work expense fund as follows

Table 5.4 – Direct expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Whiteboard	Universal Dry Erase Board	Unit	500	1	500
Paper	A4	500 sheets	60	2	120
Marker	Whiteboard marker	Unit	15	10	150
Pen	Blue pen	Unit	5	20	100
				Total	870

Table 5.5 – Salary expenses

Employee	**Work fund (days)**	**Salary per day (MDL)**	**Salary fund (MDL)**
Project Manager	105	400	42000
System Architect	110	450	49500
Sales Manager	45	300	13500
Developer	115	380	43700
		Total	148700

$$\begin{aligned} WEF &= F_{re} + FS + MI \\ &= 148700 + 34201 + 5948 \\ &= 188849, \end{aligned} \tag{5.5}$$

where WEF is the work expense fund, FS is the social fund and MI is the medical insurance fund. In that way the total work expense fund was computed.

5.4 Individual person salary

Along with total work expense fund, it is necessary to compute the annual salary for the developer. Considering that the developer has a salary of 380 MDL per day and there are totally 250 working days in the year, so the gross salary that the developer gets is

$$GS = 380 \cdot 250 = 95000, \tag{5.6}$$

where GS is the gross salary computed in MDL.

Social fund tax this year represents 6%, so the amount that should be tax paid in MDL represents

$$SF = 95000 \cdot 6\% = 5700. \tag{5.7}$$

Medical insurance tax represents 4.5% and gives the following result

$$MIF = 95000 \cdot 4.5\% = 4725. \tag{5.8}$$

In order to proceed with income tax computations, it is necessary to calculate the amount of taxed salary.

$$\begin{aligned} TS &= GS - SF - MIF - PE \\ &= 95000 - 5700 - 4725 - 10128 \\ &= 74447, \end{aligned} \tag{5.9}$$

where TS is the taxed salary, GS – gross salary, SF – social fund, PE – personal exemption, which this year is approved to be 10128.

The last but not the least thing to be computed is the total income tax, which is 7% for income under 29640 MDL and 18% for income over 29640 MDL.

$$\begin{aligned} IT &= TS - ST \\ &= 29640 \cdot 7\% + (74447 - 29640) \cdot 18\% \\ &= 2074.8 + 8065.3 = 10140.1, \end{aligned} \tag{5.10}$$

where IT is the income tax, TS – the taxed salary and ST – the salary tax. With all this now it is possible to find out what's going to be the net income.

$$\begin{aligned} NS &= GS - IT - SF - MIF \\ &= 95000 - 10140.1 - 5700 - 4725 \\ &= 74434.9, \end{aligned} \tag{5.11}$$

where NS is the net salary, GS – gross salary, IT – income tax, SF – social fund, MIF – medical insurance fund.

5.4.1 Indirect expenses

The indirect expenses are things like electricity, Internet traffic, water, etc. Those will be presented in Table 5.6.

Table 5.6 – Indirect expenses

Material	**Specification**	**Measurement unit**	**Price per unit (MDL)**	**Quantity**	**Sum (MDL)**
Internet	Moldtelecom	Pack	200.00	3	600
Transport	Public bus	Trip	3.00	132	396
Phone	Moldtelecom	Pack	30.00	3	90
Electricity	Union Fenosa	KWh	1.58	250	395
				Total	1481

5.4.2 Wear and depreciation

Another important part of economic analysis is the computation of wear and depreciation. It is a well known fact that any product decreases its value with time. Depression will be computed uniformly for the whole project duration, so that there are no accountancy issues. In other words, if a product is planned for

3 years, it should be divided into 3 uniform parts according to each year.

Straight line depreciation will be applied. Normally wear is computed regarding to the type of asset. The notebook and single-board computer are usable for a period of 3 years. Licenses will last for a single year. At first tangible and intangible assets are summed up and then the salvage costs of each of the items at the end of their period of use has to be subtracted:

$$\begin{aligned} TAV &= \sum (AC - SV) \\ &= (23000 - 1000) + (8000 - 1000) + (5700 - 1000) + (8400 - 1000) \\ &= 41100, \end{aligned} \tag{5.12}$$

where TAV is the total assets value, AC – assets cost, SV – salvage value. In order to get the yearly wear, divide total asset value by the period of use of assets, being 3 years.

$$\begin{aligned} W_y &= TAV/T_{use} \\ &= 41100/3 \\ &= 13700, \end{aligned} \tag{5.13}$$

where W_y is the wear per year, TAV – total assets value, T_{use} – period of use. Relation (5.13) included tangible assets which will last for 3 years and intangible assets which last only one year. The initial value of assets in MDL was

$$\begin{aligned} W &= W_y/D_y \cdot T_p \\ &= 13700/365 \cdot 135 \\ &= 5067, \end{aligned} \tag{5.14}$$

5.4.3 Product cost

With all the project expenses computed, it is easy to compute the product cost which includes direct and indirect expenses, salary expenses and wear expenses as shown in Table 5.7.

5.4.4 Economic indicators and results

With the expenses computed, it is now time to calculate the possible price for each copy of the application. The total product cost is very high, consequently there are 2 strategies that can be applied – whether sell less with a high price or sell more with a lower price. It is not possible to add a percentage to the product cost that will represent the profit. It is assumed that the expected profit represents 20% of the total product cost and the expected number of sold copies to be 500.

$$\begin{aligned} GP &= C_{total}/N_{cs} + P_p \\ &= 199737/500 + 20\% \\ &= 480, \end{aligned} \tag{5.15}$$

Table 5.7 – Total Product Cost

Expense type	**Sum (MDL)**	**Percentage (%)**
Direct expenses	70370	26.47
Indirect expenses	1481	0.55
Asset wear expenses	5067	1.90
Medical insurance tax	5948	2.23
Social service tax	34201	12.86
Salary expenses	148700	55.95
Total product cost	**265767**	**100**

where GP is the gross price, C_{total} – total product cost, N_{cs} – number of copies sold, P_p – chosen profit percentage. This is not the price of the end product, since it is necessary to add sales tax (VAT), which represents 20% and is added to the gross price.

$$\begin{aligned} P_{sale} &= GP + TX_{sales} \\ &= 480 + 20\% \\ &= 576, \end{aligned} \tag{5.16}$$

where P_{sale} is the sale prices including VAT, GP – gross price, TX_{sales} – sales tax. The net income is computed by multiplying gross price and the number of expected copies to be sold, which will be

$$\begin{aligned} I_{net} &= GP \cdot N_{cs} \\ &= 480 \cdot 500 \\ &= 240000, \end{aligned} \tag{5.17}$$

where I_{net} is the net income, GP – gross price, N_{cs} – number of copies sold. Moreover it is necessary to compute the gross and net profit. The indicators are GPr – gross profit and NPr – net profit.

$$\begin{aligned} GPr &= I_{net} - C_{production} \\ &= 240000 - 199737 \\ &= 40263 \\ NPr &= GPr - 12\% \\ &= 40263 - 12\% \\ &= 35431.44, \end{aligned} \tag{5.18}$$

where I_{net} is the net income, $C_{production}$ – cost of production. The profitability indicators are C_{profit} – cost profitability, S_{profit} – sales profitability computed in MDL.

$$\begin{aligned} C_{profit} &= GPr/C_{production} \cdot 100\% \\ &= 40263/199737 \cdot 100\% \\ &= 20.15\% \\ S_{profit} &= GPr/I_{net} \cdot 100\% \\ &= 40263/240000 \cdot 100\% \\ &= 16.77\%. \end{aligned} \tag{5.19}$$

5.5 Marketing Plan

Concept of Marketing derived from the word market. Marketing - economical activities that guide flow of goods and services from producer to consumer. Marketing is a system of economical activities about price setting, promotion and distribution of products and services to satisfy current and potential consumers requests. Marketing is the science and art of exploring, creating, and delivering value to satisfy the needs of a target market at a profit.

Functions of Marketing:

- Analyzing of external environment;
- Analyzing consumers behavior;
- Development of product;
- Development of distribution;
- Development of promotion;
- Price setting;
- Social responsibility;
- Management marketing.

To make people use a new application is not so easy because it needs time and investment to make it popular and well known. First of all the application will be easy to use so that an ordinary browser user will be able to intuitively use the application.

Market research stages:

- Identifying the problem;
- Developing program of research and gathering information;
- Establishing specific information (internal, external);
- Establishing methods for collecting data;
- Performance of research;
- Information analysis, drawing conclusions.

Introduction stage - This stage of the cycle could be the most expensive for a company launching a new product. The size of the market for the product is small, although they will be increasing. On the other hand, the cost of things like research and development, consumer testing, and the marketing needed to launch the product can be very high, especially if it's a competitive sector.

Strategy - Screaming, massive penetration The growth stage is typically characterized by a strong growth in sales and profits, and because the company can start to benefit from economies of scale in production, the profit margins, as well as the overall amount of profit, will increase. This makes it possible for businesses to invest more money in the promotional activity to maximize the potential of this growth stage.

Maturity Stage - During the maturity stage, the product is established and the aim for the manufacturer is now to maintain the market share they have built up. This is probably the most competitive time for most products and businesses need to invest wisely in any marketing they undertake. They also need to consider any product modifications or improvements to the production process which might give them a competitive advantage.

Declining stage - the market for a product will start to shrink, and this is what's known as the decline stage. This shrinkage could be due to the market becoming saturated (i.e. all the customers who will buy the product have already purchased it), or because the consumers are switching to a different type of product.

5.6 Economic conclusions

DSLR Camera Controller project was analyzed from the economic point of view. It was computed the production cost, different profit and profitability indicators, various types of expenses involved, including direct, indirect, salary and taxes. The whole analysis is worth to understand if the product will be successful and if it's worth investing money in it. The biggest expense represents the intellectual equity, since it is critical to have a reliable product, which is based on extensive research and professional development techniques. The price of the application can become a blocker, therefore it's price might be dropped. In such scenario other means of profit can exist.

The commercialization of the product is not an easy task. Especially when the product is open sourced. Nevertheless high-quality service and customer support can be provided only to institutions and users that bought the product. The success of the product highly depends on financial strategy and solid economic analysis, which was presented in this chapter.

Conclusions

DSLR camera controlling system is the outcome of this thesis work. It is a tool for automating the work with cameras, but specifically, it mostly focuses on automating the process of reliably taking long term time-lapses. The uniqueness of this application is that the targeted users requires no additional input once the application has been set, as all the tasks have been completely automated. An instance of this application is currently running since Summer 2019 at a warehouse at CERN, with the only required user input being that the client had to re-plug the camera because someone has tripped over the wires. No additional actions were needed for this system to continue working. On the other hand, the applications the client was using were extremely hard to automate. Those applications were extremely prone to stopping because of different reasons: power outage, someone would trip over the cables and re-plug them back but the applications would stop working, someone would simply power off the camera because of a private event which would also break the existing applications. Additionally the client had to manually do health checks every week and check if the photos were successfully transferred on the server. On top of that, since the existing applications were desktop-only, the user had to use a tool like Remote Desktop to connect to the system running the applications. This was extremely painful as the internet speed is not very high in that warehouse.

But the DSLR camera controlling system solves all these problems. Instead of using a Remote Desktop application, the user can access the cameras with a few clicks from a browser. With a few additional SSH tricks, the client can access its cameras from anywhere with no additional costs. In case of a failure, email notifications are dispatched and a few configurable failure strategies can be set up. A full log of all the events is accessible within the application, making it a lot easier to investigate and solve the problems, or to simply check the application's progress.

A quintessential part for the future development of the DSLR camera controlling system is to make the communication with the cameras even more reliable to prevent any possible capture failures with any supported camera. The application currently experiences minor problems after every weekend, because the schedule is configured to not take any pictures during this time, making the cameras fail 2-3 times every Monday.

Another future implementation would be to include authentication and authorization. Currently the application did not require any additional security protections because the photos taken are free to view for everyone. Even if there were any security requirements, it could be solved using SSH tunneling. But an authentication mechanism would be much simpler for the user and for its management.

References

1 – PyCharm - Wikipedia (Accessed on 23/05/2020)
https://en.wikipedia.org/wiki/PyCharm.

2 – gPhoto tool for camera control - Wikipedia (Accessed on 26/05/2020)
https://en.wikipedia.org/wiki/GPhoto.

3 – The SSH protocol - SSH Academy (Accessed on 26/05/2020)
https://www.ssh.com/ssh.

4 – Yepkit USB Switchable Hub - Yepkit store (Accessed on 26/05/2020)
https://www.yepkit.com/products/ykush.

5 – The SOLID Principles - Wikipedia (Accessed on 26/05/2020)
https://en.wikipedia.org/wiki/SOLID.

6 – The Dependency Inversion Principle - The Clean Architecture book by Bob Martin. (Accessed 15/04/2020)

7 – Stub object for testing - Wikipedia (Accessed on 29/05/2020)
https://en.wikipedia.org/wiki/Test_stub.

8 – Cron tool for scheduling repetitive tasks - Wikipedia (Accessed on 14/11/2019)
https://en.wikipedia.org/wiki/Cron.

9 – Django Python Web Framework - Wikipedia (Accessed on 26/05/2020)
https://en.wikipedia.org/wiki/Django_(web_framework).

10 – Inversion of Control Containers and the Dependency Injection pattern - by Martin Fowler (Accessed on 26/05/2020)
https://martinfowler.com/articles/injection.html.

11 – Pinject tool for dependency injection in Python - Github (Accessed on 14/11/2019)
https://github.com/google/pinject.

12 – Unit Testing - Software Testing Fundamentals (Accessed on 26/05/2020)
http://softwaretestingfundamentals.com/unit-testing/.

13 – SWOT Analysis (Strengths, Weaknesses, Opportunities, Threats) (Accessed on 26/05/2020)
https://www.mindtools.com/pages/article/newTMC_05.htm.