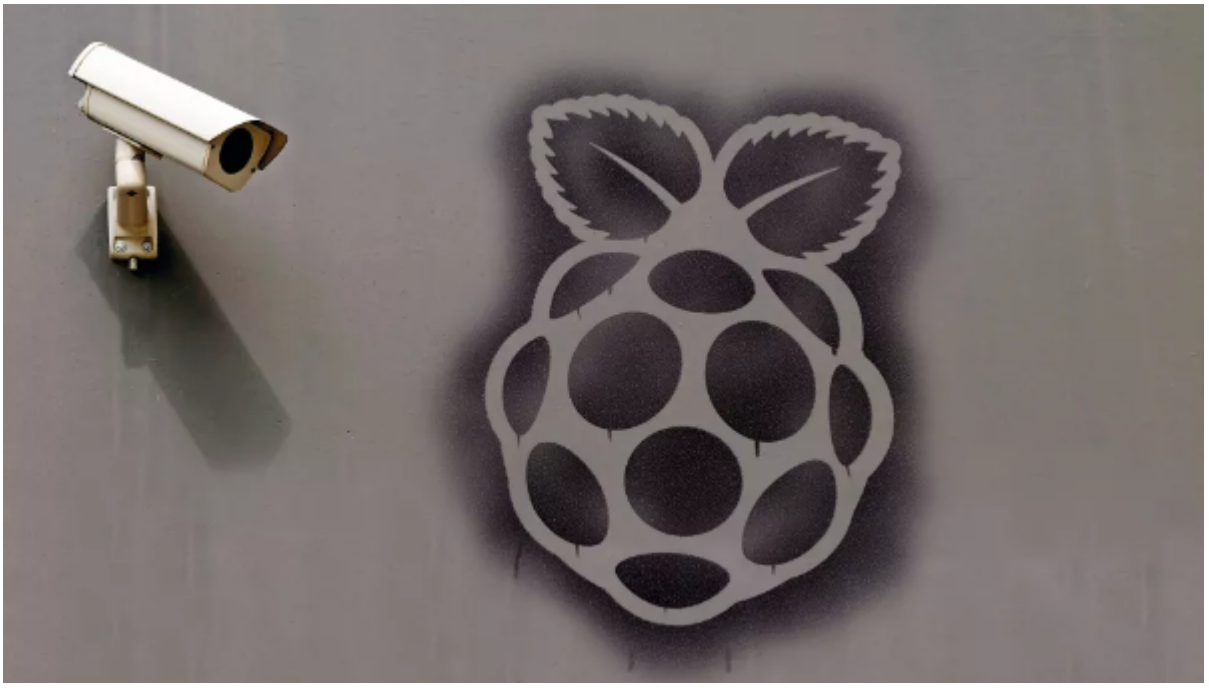
3rd Semester Project Report
Security Camera - Zealand



Team members:
Maria Nijar - mari85tf@edu.zealand.dk
Kengo Reimers Kato - keng0007@edu.zealand.dk
Kacper Jakub Bytnar - kacp0294@edu.zealand.dk
Olegs Marcenuks - oleg0195@edu.zealand.dk

Abstract **1**
Project manifest
Extreme Programming 2
Product Description 6
Corollary practices 6
Mandatory Practices and Roles 6

Introduction **8**
Problem definition 8
Solution approaches: 9
Technologies: 9
Methodologies 9
Development phase: Extreme programming 9
Project planning 10
Limitations 10

Development process **10**
Inception Deck 10
Risk Analysis for inception deck 13
Business Process Model 14
Stories 14

Domain Model **15**

Class Diagram **16**

Database **17**

System architecture **18**

GUI Design **19**
Gestalt law 20
Heuristics 21

Extreme programming process **21**
Week 1 21
Week 2 22
Sprint 1 (Week 3) 23
Sprint 2 (Week 4) 25
Sprint 3 (week 5) 27

Quality assurance **29**
Unit Tests 29
System Testing 30
Acceptance Testing 30

Abstract - Kengo

After a semester of learning a broad variety of new skills within Systems Development, Programming, and Technology; we as a group were tasked to put all skills learnt during the semester into a relevant project that can show both the use of teamwork, and work proficiency that we have individually gained. Along our progression throughout the past weeks of working on Security Camera - Zealand, we have worked with a raspberry pi and different components within (Motion Sensor, Camera, Monitor), C# staying under the .NET Framework using a REST web API, python for as a TCP broadcaster and receiver, and Azure databases as a final location to store the images taken on the camera. All of this was worked on using an Agile XP methodology for maximum efficiency and results. With all of this combined, the ultimate result of the project includes a front-end website which is able to display the images taken through the raspberry pi when the motion sensor is activated, therefore allowing the user of the product to see the security camera images on their laptop.

Project manifest - Maria

Extreme Programming

Extreme Programming (XP) is an agile software development framework that aims to create higher-quality software and better quality of life for the development team. XP is the most specific of the agile frameworks that deal with appropriate engineering practices for software development.

There are 5 XP values i.e., communication, simplicity, feedback, courage, and respect. As a team, it is important to communicate with each other to transfer knowledge from one team member to everyone else on the team. It reduces misunderstandings and errors between members of the project. In our team, we are choosing to directly communicate with each other by discussing face-to-face with the aid of a whiteboard or other drawing mechanisms.

Simplicity means “what is the simplest thing that will work?” As a team, we should avoid waste and do only the absolutely necessary things such as keeping the design as simple as possible. It will make things easier like maintenance, support and revision.

Feedback also supports simple design. As a team, we are building something. That is why we should gather feedback on our design and implementation, and then adjust our product going forward. Through constant feedback about our previous efforts, as a team, we can identify areas for improvement.

As a team, we need the courage to stop doing something that doesn’t work and try something else, to accept and act on feedback, even when it’s difficult to accept.

Respect the members of the team in order to communicate with each other. As a team, we should provide and accept feedback and we should work together to identify simple designs and solutions.

We use these XP engineering practices because it has been shown to have a significant impact on the productivity of the project team. We believe that adapting these methods will have a positive impact on our team. Applying these methods together introduces management, customer and development practices and values to form a unified and disciplined team.

Requirement Analysis

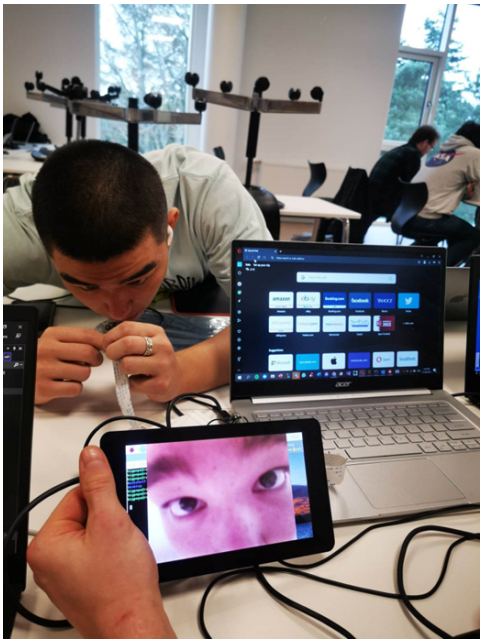
Requirement Analysis is a process of determining the needs and expectations of a new product. It entails frequent communication with the product’s stakeholders and end-users to aspects of planning, managing disagreements, and documenting all essential components.

<u>Identify Product Owners and End-Users</u>
In our case, the product owners and the end-users are the same people. They will use it for their own purpose.

<u>Capture Requirements</u>

Build Prototype: During the last sprint, we made a prototype of how the Front-End of the product will look to the product owners. With that, we received feedback from them and we could work on finalising the product to how the product owners wanted.

Meeting together: During school hours, we usually sit together with the project owners. I helped our communication with each other, answering questions and avoiding misunderstandings that can arise during the project.



Requirements reviews/inspections: Mostly, our User Stories are done with pairs. When a User Story is finished, the other pair is reviewing the code and gives feedback for improvement or if something is missing. The inspection is finished when everyone in the team agrees that the User Story has all the requirements and officially we label the User Story as done.

Test-case generation: When a User Story for a certain case is done, we assign someone from our group to start testing the code. We use test cases to see if the different aspects within a system are functioning properly and to verify that the system has met all related standards, guidelines and customer requirements. The process of creating a test case can also aid in the discovery of errors or defects in the system.

Stories: As a team, we drive user stories from a scenario, which will provide more information to developers to help them design the product's features. The user stories were then prioritised on planning poker using the Moscow method, which is a four-step approach to prioritising the user stories. This method helps easily clarify and prioritise what should be done first during the project. Afterwards, the prioritised user stories were then placed in the product backlog.

<u>Categorize Requirements</u>

Explanations on why we prioritise the User Stories in this order.

Must-Have

We have in total of 4 User Stories in the Must-Have priority. These User Stories prioritise not only the fundamentals of the project but also the basic functionalities for the system to work. These User Stories are an absolute MUST. There is no way out and there is no shortcut for the development team.

On the side of the user of the system, it is very important to use the items in a database. Without the database, we can't transfer data to the user interface. The UI has only a list of items. The database also needs the it

em to identify them on the customer side. By only using the barcodes, we make the system easier to use for the user of the system as well as the customer.

On the side of the customer, it would be very important that they can start the session and me knowing the scanning went successful, the scanner has to make a noise to alert the customer.

Should-Have

We have in total of 3 User Stories in the Should-Have priority. These User Stories priorities are completing the fundamentals of the project. We create a simulation of the supermarket story. With these functionalities, we can sell this system as a product for supermarkets or other businesses. They make the system that we are building complete for the user as well as for the customer. These User Stories are essential but not vital.

On the user side of the system, it would be handy to see only items that the user is searching for in the UI. The UI is getting extra features here.

On the side of the customer, we add a finishing process so we can simulate it as a supermarket where there are different customers and each has a different amount of items and total prices. In these situations, we also make it easier for the customer to delete a product if they accidentally scanned it twice.

Could-Have

We have in total of 5 User Stories in the Could-Have priority. These User Stories priorities are little details that make the project look complete. When we have all the User Stories from this priority included in the system then the project will have a finishing look and can be used as a full-functioning system for the outside world. These User Stories are not a problem if it's left out but still are of significance.

On the side of the user of the system, it would be good for the UI to see more details of the displayed items and have statistics and/or graphs to use for helping businesses economically. Pictures would be a more convenient feature for users rather than using only the name of the items. On the side of the customer, we added a screen for the total price so the customer can use it to see how much they have to pay. With this, we can use it to add it to the UI for statistical purposes.

Won't-Have (but nice to have)

We have only 1 User Story in the Wont-Have priority. These User Story's priority is irrelevant, but the development team can work on them if they have time left.

On the side of the customer, it would be easy to add details of the scanned product like a shopping ticket.

Planning

Our definition of done: First, all the tasks of the user stories should be finished. That means that all the user stories are fully functional and implemented. Everything was documented through Swagger. The code doesn't need any refactoring and it was unit tested where possible. Afterwards, when we receive feedback from the product owners and apply

the feedback. When the whole team accepts that our product has the quality and the product owners consider the project work to be complete, then we are done.

The project is ‘definitely’ done when we obtain the product owner’s signoff.

Risk analysis: It enabled our team to assess the risks that we faced. With this, we decided whether or not to process a certain decision. At the end of our project, we could prevent some risks that we foresaw, but not all of them. During our project, there were some unexpected risks that we didn’t foresee. These risks will be written in the root cause analysis and will be used in future projects.

Weekly Cycle: In total, we had 3 sprints in our project, each with a duration of 1 week. The Sprint Planning occurred always on the first day of a new sprint. Some User Stories needed more than one sprint duration.

How will we work

Informative, visual Workspace: We use Google Docs to collect all the documents in a folder. Here we can easily look at each other's work, edit together and see how far we come in our project.
We use Trello to easily make ‘todo’ cards for the whole team. Here we can choose which ones we want to work on, see where the others are and which ones are finished. It also makes it easier to put comments inside the ‘todo’ cards so we can improve on what we are working on.
We use Discord as a way to not only communicate with each other but also with the project owners. If we have any questions, we have a way to communicate with them as quickly as possible.
We use Excel for visualising the sprint burndown chart. With this, we always know the progress of our project work.

Pair Programming: Whenever there is a task where you have to code, we did it with two people. It makes it easier to find coding mistakes and twice as fast to receive working code. Afterwards, as a team, we looked together at the code and gave feedback and reviews, and if necessary we refactored it.

Shared Code: All the code that we are working on is published on GitHub. We have all access to and are all collaborators. With this, we always know where the rest of the group is currently on and how far we are in our project. It is also an easy way when we use pair programming. We only need to pull the code and commit it when we are finished.

Sit Together: We decided that we would use the school hours to work on our project. If we feel that we need more time, we will sit together after class. We also used outside class hours to meet with each other at school. If there is someone that can’t come to school then

we use weekends to meet each other in discord. Discord is an easy way to communicate with each other and share screens at the same time.

Sitting together prevented misunderstandings and miscommunications that could arise during our work progress. It is also the easiest option to communicate with each other and you don't need to wait for an answer. Sometimes we used other classrooms to share our screens. This led to easier following what is happening during our project and faster working on our project.

Product Description - Kacper

The product that we have made with our project is a multi-functional security camera that allows the owner of the product to implement more security within their life. The reason for why our product is worthy of purchasing is because it is a low costing product since the raspberry pi was revolutionary for its pricing, therefore making it an efficient cost worthy product within the security market. The product will have a lightweight but sleek look to it allowing it to be placed in areas without seeming "out of place" and it will have different features to make sure it is kept in place. The camera has the possibility to be placed in different areas since it can be used in many different ways, ultimately making it the perfect tool for a variety of people, anywhere from parents at home to security equipment in much larger buildings.

Corollary practices - Maria

Mandatory Practices and Roles

- On site Customer
- "Planning Game"
- Pair Programming
- Informative, visual workspace
- Shared code
- Single code base
- Test First/Driven Development

Definition of Done

- Fully functional User Stories
- Everything documented through Swagger
- Code doesn't need any refactoring
- Code was Unit tested where possible
- Applied the feedback
- Team accepted the quality of the product
- Product owners consider the project work to be complete

Root Cause Analysis

RCA is a broad word that refers to a variety of approaches, tools and procedures used to identify the root causes of problems. Some RCA approaches are more focused on discovering genuine root causes than others, while others are more general problem-solving procedures. Still, others simply provide support for the basic activity of root cause analysis.

Types

Physical Causes: When a physical item fails, this is referred to as a physical cause. In our case, we physical causes are the problems that we experienced with the raspberry pi like:

Camera: On the last sprint, the camera stopped displaying the images. The output messages that we received from the raspberry pi were purely a problem in the connection. The solution that we came up with is to disconnect and reconnect everything from the raspberry pi and we also reset the sunny chip.

Sensor: The biggest issue here was that the sensor is very sensitive to movement, so we tested for what it caused and for a possible solution. We learned that the sensor has pins that can reduce the sensitivity from the sensor but unfortunately it didn't help in our case. The solution that we came to in the end was that we put in the code that the camera has to wait 45 seconds each time there is a movement so we don't have a lot of unnecessary pictures.

TCP: It is not fully a physical cause, but we can say that the average space that a picture is taking inside the raspberry pi is around 120 bytes. A TCP connection can only store data of around 65 bytes. The problem was that it couldn't send an uncompressed jpeg file without having bugs. The solution that we came up with was that we need to split the file into two parts and send them as a zipped file.

Human Causes: This type of root cause occurs when one or more team members do something poorly. Human error will frequently result in a physical cause, such as:

Class Library: We choose a first class library because it will be easier to correct errors in the future because the class library is separated from the API and it won't cause any problems on the API part. Afterwards, we chose that our class library would go inside the API because the group had some problems with the DLL file. Our first solution was to have on each computer the DLL file, delete the reference that was already set from GitHub and add the new reference. With this, the group could use the library in the API. In the end, we choose the easy option of having the library inside the API. With this, when we update the API, we automatically update the class library. We also have the class library together with the API in one repository, so we don't need to search for it individually.

For future projects, we would choose Model over Class Library, because there is no need to put a class library inside the REST service. With the model, we don't need to have a reference to a DLL file.

Class Library is better than Model for debugging/refactoring and re-use purposes. If we have to solve a coding error, it would stay only inside the library and won't cause any problems to the other parts of the system.

The main advantage of a model class is that you don't need to use references to another file. It is already inside the REST service and only the REST service can use it

From UDP to TCP: We switched from UDP to TCP because we wanted to have a reliable connection where the packages are coming in the order we send them.

Displaying pictures: Because of a lack of knowledge, we spend a lot of time on how to convert a byte array to a picture. In the end, our solution was to link each picture, using the id of the picture. We used a function to read the bytes and convert them into an image.

Continuous Integration: We received an error message by pushing the code with CI. DevOps couldn't find the Secrets File because we didn't display it for outsiders. After a long discussion with everyone from our group, we came to the conclusion that we don't push the code to DevOps. We only stayed on publishing the code in Azure and pushing the code on GitHub.

Organisational Causes: An organisational root cause occurs when a system or process used by an organisation to perform its functions fails. During our project, we didn't have any organisational causes.

Introduction - Kengo

Problem definition

The problem definition around our project is clearly the factor that there is an increased need for home security due to multiple break-ins happening in Denmark and around the world. We used the initial given architecture with a minute detail change of using TCP instead of UDP since the use of UDP will require a division of packets which leads to an increase in chance of losing the order of the packets.

Problem - Solution:

- How we are going to get an image taken on the raspberry pi into an azure database so that we are able to view it on a website - Solution was to use one laptop acting as a server that will receive the images from the pi through TCP (after compression on the pi) so that it can then be uploaded to the azure database staying as a byte-string array

- How can we use components of the raspberry pi (e.g. sensor, camera, monitor) in order to achieve what we want - Solution to this was to do research on the different python libraries we needed to use within the raspberry pi (software) and how to connect them together (hardware).

- How are we going to turn the raspberry pi physically into a functional camera in the future as it is the desired product in the end - Some possible solution to this that we thought of as a team consists of a 3D printer (economically demanding procedure), or hand craft a case for it to hold things in place (e.g. cardboard, paper).

Solution approaches:

During the entire process of this project, we clearly had no choice but to think of how we are going to practically place the camera and how it will look when we were to get it to its fully functional environment. The fundamental base theory of our product is to bring a device to our customers that will be able to function as a security camera and triggering a camera to take an image when the motion sensor is activated. Our multiple solutions consisted of:

- One solution, was having the security camera placed by the front door of a house, so that when someone was to be on the property of someone else who owns this camera, the motion sensor will be activated causing a picture of whomever this is to be taken and be visible to the owner of the home and the camera.

- Another solution that we had in mind was that we were to just simply have the security camera in an inside area so that it can keep patrol on for example a room inside the house. This opens up a lot more options since people can use it for things such as baby monitors. This ultimately opens up more doors for sales opportunities since the customer base can increase. It is also more efficient since the product can cost less to create due to no need of making it waterproof as it will be indoors removing the risk of getting the camera damaged from vandalism too.

Technologies:

Languages used: C#, .NET framework, javascript, html, css, bootstrap, vue.js, python
Applications: Console Application, ASP.NET REST WebAPI, Class library
Hardware: Raspberry pi, PIR Motion Sensor, Raspberry Pi Camera Module 2, A laptop, Monitor for raspberry pi
Technology: TCP, Postman, Swagger, Selenium, Socket Testing, PuTTy, Visual Studio Enterprise, Azure, Pycharm, Nano

Methodologies

The specific methodology that was used for our product is XP. As you may know XP consists of five valuable principles: Communication, Simplicity, Feedback, Courage, and Respect. We as a team aimed to meet all five principles when working throughout our project.

Development phase: Extreme programming

Since we decided as a team that we were to go with using the agile XP methodology, we would have to say that overall, it benefited us when working with our product. If we were to take a closer look at what the different components of XP were that we used, it would include: Test-first development, Refactoring, small releases, simple design, on-site customer, sustainable pace, pair programming, collective ownership, incremental planning, and continuous integration. Following so, some examples of where we showed use of the five principles include:

1. Communication - Used online communication between the group and software such as trello and google drive to have live editing that helped us with working together more as a team.
2. Simplicity - Made the functions as simplistic as possible by using less code if possible and writing comments to make sure that everything was understood by others. Refactoring and simplistic GUI design.
3. Feedback - Had a discord server with our product owner so that we were constantly in touch with each other. We also got feedback from them through in person communication too.
4. Courage - We showed evidence of courage when we had to make split decisions on whether we should keep working on a particular area such as the broken sensor or not being able to upload to devops.
5. Respect - We showed evidence of respecting each other when we had planned meetings online or in person and we all showed up.

Project planning

For this project we had several dates to consider. Code freeze deadline was on December 15th and the final report hand-in on December 22nd. Taking this into account we planned our sprint as follows: we had 3 sprints x 5 days each with 5.5 to 6 working hours per day. The rest of the days after the code freeze we spent on finishing the report.

Limitations

Some struggles that we had faced during the project are lack of knowledge in some areas for the hardware we were using, and having faulty hardware in the end. How we approached the first problem listed above, was by taking time to research the individual parts and learn how to circuit the raspberry pi as it would have been a huge problem if we were to short circuit it. The second limitation was the most severe, it ultimately ended up causing issues with the final product. The research that we did for the first limitation was very useful as we were able to create multiple tests for the different components to check whether the hardware was fully broken or not.

Development process - Maria

Inception Deck

Project Name: Security Camera

Why are we here?

To design and develop a solution that tracks moving objects and takes a picture (and eventually also a video) and sends it to the receiver using distributed systems and technologies.

Elevator Pitch:

For Product owners
Who wants to secure a building
Is a web application
That helps to know if someone unauthorised is inside a building
Unlike the current situation where there are no cameras that notify if there is some movement detection
Our product will address the needs of people that want to secure their property and reduce the stress of the unseen situation and not knowing the identity of the thief.

Product Box:

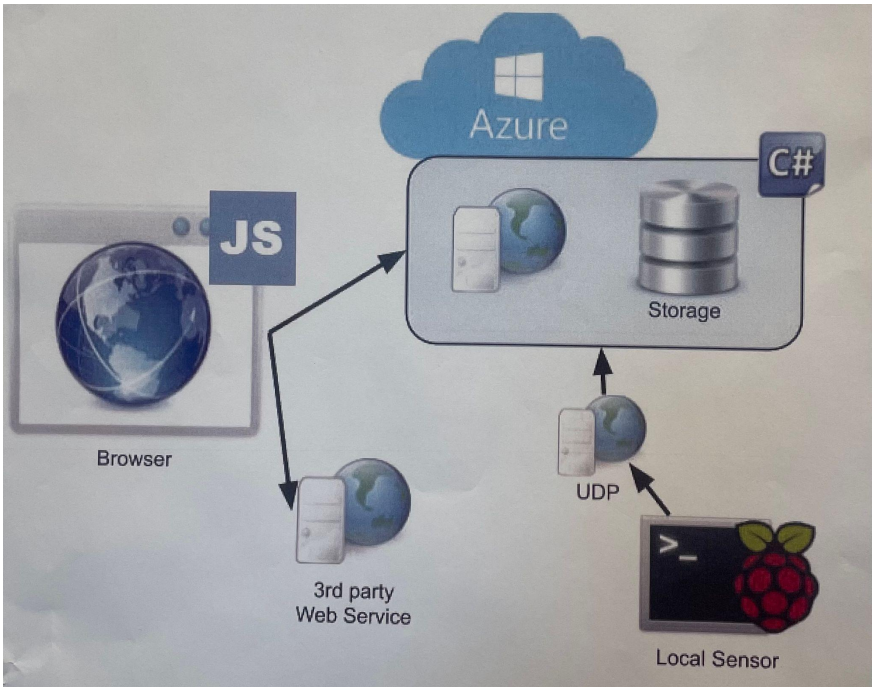
A NOT List:

In scope	Out of scope
- Raspberry Pi camera taking pictures on movement detection - Active motion detection - Raspberry sending data via TCP - GUI displaying data from REST API - REST API services connected to database - Filtering pictures on website	- Sending a notification on movement detection with access to live camera preview - Live camera preview on website - Admin panel for remote cameras control - Storage of pictures on raspberry PI

Meet the Neighbors:

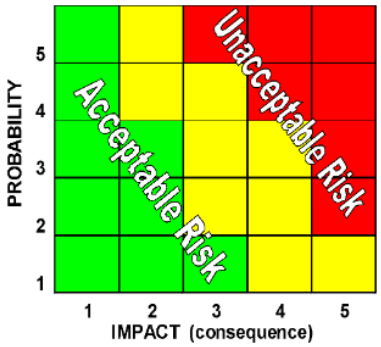
- The students doing the project
- The product owners assigning us the project
- The people inside a building/property where the cameras are installed
- The people that receive the pictures/videos of the cameras

Show the Solution:



Later we replaced UDP with TCP.

Risk Analysis for inception deck



Value consecuences

1 A minor issue in the project that creates irritation
2
3 Issue that cause delays, necessitating a revision of the plan and budget
4
5 Serious issue that prevents the project implementation, and consistency can be closure of the project

Prevent when High Probability and Minor Consequence
Mitigate when High Consequence and Minor Probability (Plan B)

				Define action taken	
Item	Probability	Consequence	P*C (Risk value)	Prevent	Mitigate
Risk assesment for our project.					
Completing the project on time	2	5	10	Do whatever it takes to deliver it on time	Do the most important parts of the project first
Failure of raspberry components	3	3	9	Test before actual show	Check for spare parts / order if necessary
Bugs in a code	5	3	15	Testing, pair programming, consultations	same as in prevent + refactoring
Lack of experience in Python / JS / API	3	5	15	practice	more practice, ask teachers for help, look in internet
Lack of experience in TDD and Pair Programming	2	4	8	practice	more practice, ask teachers for help, look in internet
Sickness of team members	2	4	8	Can't be prevented	Work online, takeover responsibilities of sick person
Report writing	2	5	10	Document all steps of the work	Focus on report after something is done

What are the risks evolving our project:

- Theft of the sensors and raspberry pi
- Wi-Fi connection for the raspberry pi
- Changing the raspberry pi
- Parts breaking of the raspberry pi
- Sensitivity of the raspberry pi
- Raspberry pi screen will not work
- Completing the project on time

- Bugs during coding
- Lack of experience in Python / JS /API
- Lack of experience in TDD and Pair Programming
- Sickness of team members
- Report writing

Size it Up

Week 1	**Week 2**	**Week 3**	**Week 4**	**Week 5**	**4 days**
Brainstorming ideas, preparation, design, architecture	Coding	Coding	Coding	Coding & writing report	Writing report and hand-in

What's it going to give?

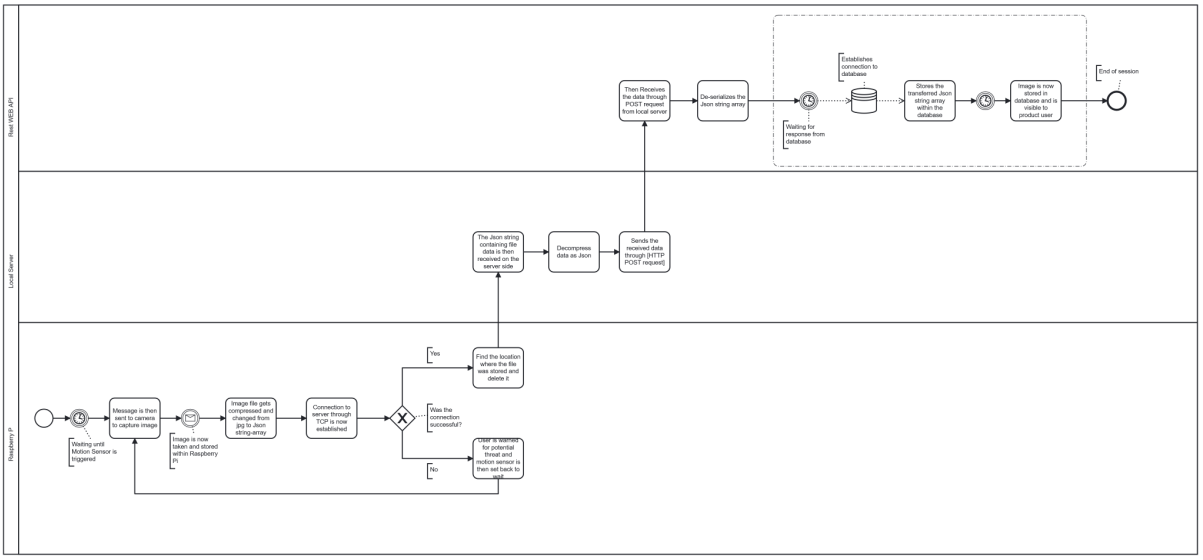
- Scope

What's it going to take?

- 21 days, approximately 6 hours daily, 4 developers.
- Approximately 126 hours/person, 480 team hours total.

Business Process Model

The business process model will represent how the operations of our project are carried out to achieve the intended goals. It is a good method to make it more comprehensible because it shows the many devices and events within the various applications as a timeline.



When the sensor detects a movement in the location of the security camera, the process starts with an event on the Raspberry Pi. When it is false, there is no movement, when it is true, there is movement and it will send TCP packet(s) from the Raspberry Pi to another device that is running the TCP receiver program. The data sent to the API is JSON formatted. The TCP receiver sends an HTTP POST request to the WebApi application after receiving the data. The controller will look at the CameraId for the given PictureId and generate a new Camera object, timestamp it, and deliver it to the manager, who will then transfer it to the data access layer for the database.

Stories

Class Library

S1.1 Create a Class Library
S1.2 Test the Class Library

WebAPI

S2.1 Create a Manager in the WebAPI
S2.2 Create a Controller in the WebAPI
S2.3 Add update sensor data to the controller and manager
S2.4 API documentation – more comments on our camera API
S2.5 Test the manager of the API via Unit Testing
S2.6 Test the controller of the API via Postman

ConsoleApp

S3.1 Create UDP receiver (later TCP)
S3.2 Test UDP receiver (later TCP)
S3.3 Refactor and comment UDP receiver (later TCP)

WebApp

S4.1 Create a Vue.js app
S4.2 Functionality to get data from the API
S4.3 Find API and implement functionality to get data from the weather API

Database

S5.1 Create a database in Azure
S5.2 Create Model Classes
S5.3 Add the relationships in the database
S5.4 Design the database

Sensor app

S6.1 Create a Python app to receive the data from the Raspberry Pi
S6.2 Create functionality for the camera
S6.3 Create functionality for the sensor
S6.4 Add functionality to send the sensor data via TCP

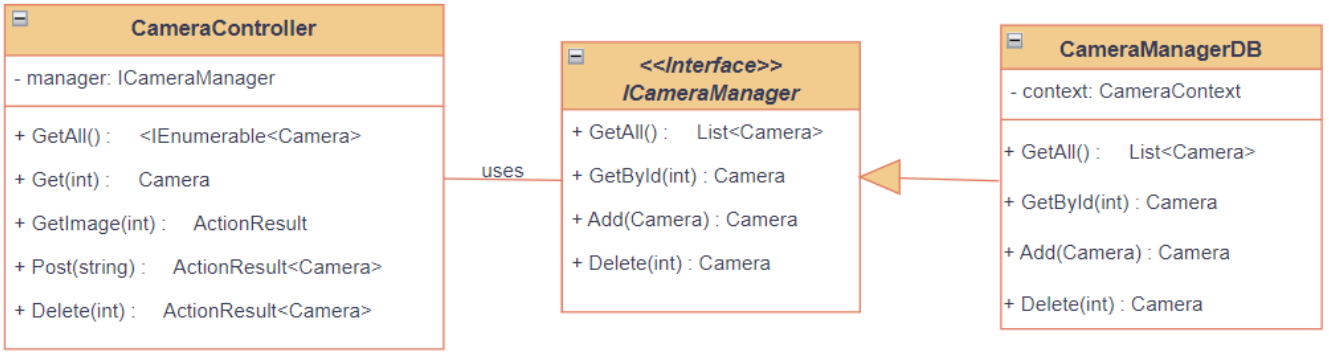
Prototype

S7.1 Create a Front-End prototype

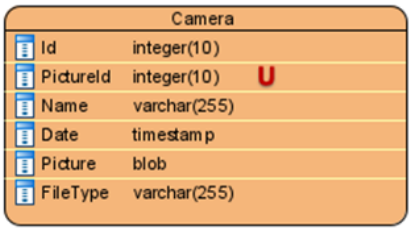
Domain Model - Kacper

A domain model, as opposed to software or databases, is a graphical depiction of real-world ideas or flow. Our project has one security camera and one sensor, so the relationship here is one-to-one. A room where the camera is installed has either movement or not. When someone enters the room, the sensor sends the security camera data to the TCP receiver.

Class Diagram - Kacper & Maria



A class diagram is a graphical depiction of a class object in a model system that is organised by class type. In our case, every time the status of the moving object changes, the Camera class holds the Id of which camera the sensor sensed a moving object, the incremented id of the picture with the incremented name, makes a timestamp and sends the taken picture with the type of the file to the API. The data from the UDP receiver is received by the API and stored in the Azure database.



Information about the picture from the camera. The picture in itself is a byte array. Afterwards, it will automatically convert it into a picture using the FileType. The picture receives a timestamp with an auto-generated name and id.

The CameraManager has a DbContext object, which allows access to the database whenever the methods in the Camera class are invoked. The ICameraManager interface is inherited and implemented by the manager. The managers class's goal is to handle data persistence, whether it is a static List, a Json File, a database, or something else.

The interface is injected into the CameraController so that the controller's HTTP methods can access the managers' methods. The controller is responsible for managing HTTP-related operations.

Finally, anytime the web application makes a request, the controller class's HTTP methods are invoked.

Database - Maria

Camera					
Id (PK)	PictureId (PK)	Name (PK)	Date (PK)	Picture	FileType
1	1	Pic01	18-12-2022 09:46:15		jpeg
1	2	Pic02	18-12-2022 10:01:48		jpeg

Our database consists of a lot of things. It is composed of a table where we store the Id of each camera and also the id of the picture as a primary key. With this, it will make it easier where the picture was taken, if in the future there will be more than one camera.

Each picture also receives a name that is auto-generated and timestamps it when the camera takes a picture. With the date, we can filter down when we search for a specific date or duration. It would also be better to see only the pictures that are only taken today and discard pictures that are staying too long in the database. With this, we can prevent the database from having space shortage in the future. With the name, we can filter down if we want to find one specific picture.

We also store the picture in our database that will read the bytes and convert it into a picture using the file type that the database receives from the camera. With this, we can show the picture on the website instead of showing the bytes.

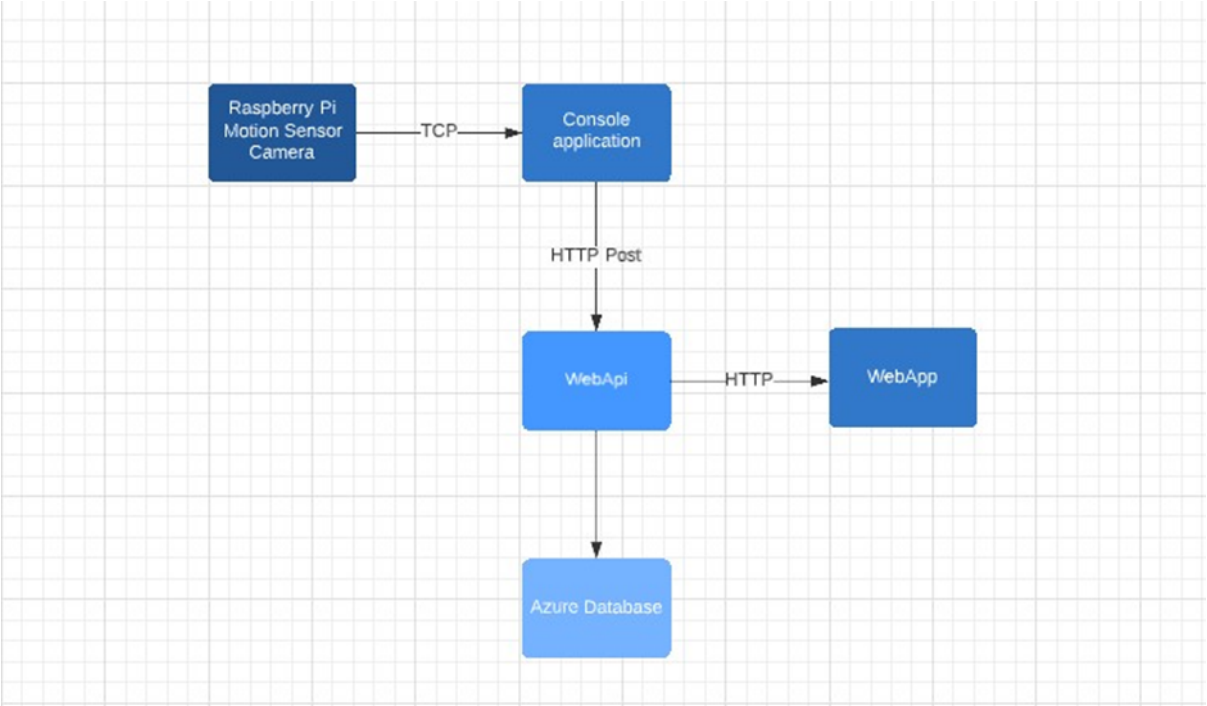
To establish the connection between API and Azure database we used DbContext instance which acts as a bridge between our classes and database. It represents a database connection and a group of tables in general. A table is represented by DbSet.

The ConnectionString of our Azure database is in the Secrets.cs file. We store there our authorization and authentication of the data. Because our API is published on GitHub, we use the gitignore file to not display the Secrets.cs file. Not only is our REST service ready to use the database, but it is also secure from outsiders.

We created the DB manager to implement the CRUD methods. We use SaveChanges to update our database so that the data would stay there and not disappear.

We decided to choose Id, PictureId, Name and Date as composite keys, so that each time the sensor has detected movement and the data is submitted to the database, it has a unique identity. This is achievable because the name is never the same and the timestamp is accurate to the millisecond.

System architecture - Oleg



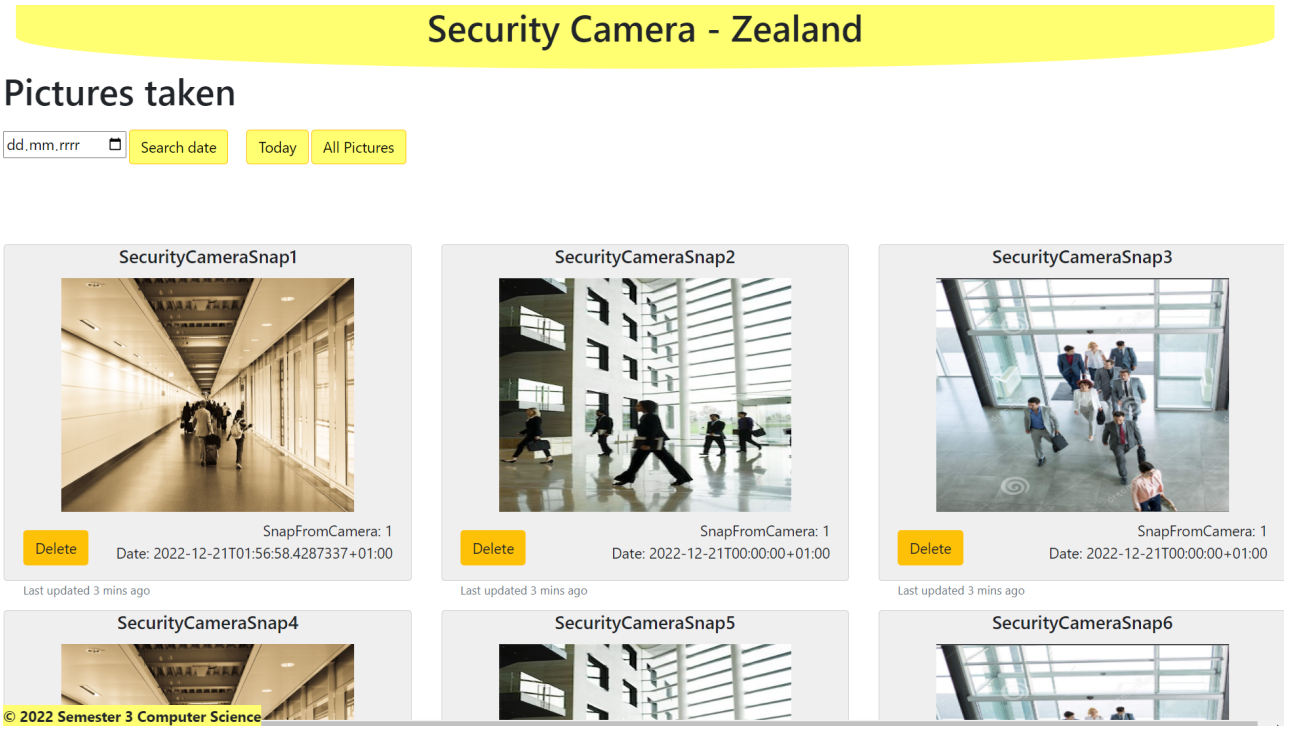
Architecture of security camera flow looks as follows:

Both sensor and camera are connected to Raspberry Pi. When the status of the motion sensor changes, meaning the sensor intercepts heat signatures of the person or object passing by the sensor, which triggers the camera to activate and take a photo. Further, the photo taken by camera is converted into JSON format and is sent to the application via TCP connection. The photo is automatically deleted after it is sent. This approach will allow us to avoid sending the same photos to the database and do not overload raspberry pi memory. The process execution on Raspberry Pi is made using Python.

We had to slightly deviate from the original project requirements, since the consistency and integrity of the transmitted and received data was important to receive full and correct photos, we decided to use TCP connection instead of UDP.

Data received by application is posted to the Rest API using an HTTP client. When the API receives photos from the application it posts and stores them in Azure database. Our web application is connected to the Rest API, and it calls the API via HTTP methods to get data.

GUI Design - Kacper



Graphical user interface (GUI) was designed together during a brainstorm in sprint 3 by drawing the design on paper sheets. Everybody from our team has agreed on a minimalistic design focused on providing the core functionalities in a user-friendly way.

Pictures were displayed using bootstrap, more precisely a “card-deck” class. The top of the card contains the image with the title. At the left-bottom there is a “Delete” button which allows you to delete a particular picture from the database. The right-bottom shows the crucial data about the photo: the ID of the camera that took the picture and date of taking the picture.
We decided to place filtering buttons on the top panel so the user could easily access live filtering based on his criteria.

For base colours of the graphical interface at the initial stage, we decided to use a lot of yellow colour RGB (255,255,113) which resembles Zealand’s colour palette. We decided on a fully white background (255,255,255) and light grey card’s background rgb(240, 239, 239).

To provide a better user experience our GUI should follow the design heuristics and apply Gestalt laws.

Banner && top panel buttons && footer:

rgb(255,255,113)

Cards-background:
rgb(240, 239, 239)

Delete buttons
rgb(255, 193, 7)

:

Website’s background

Gestalt law

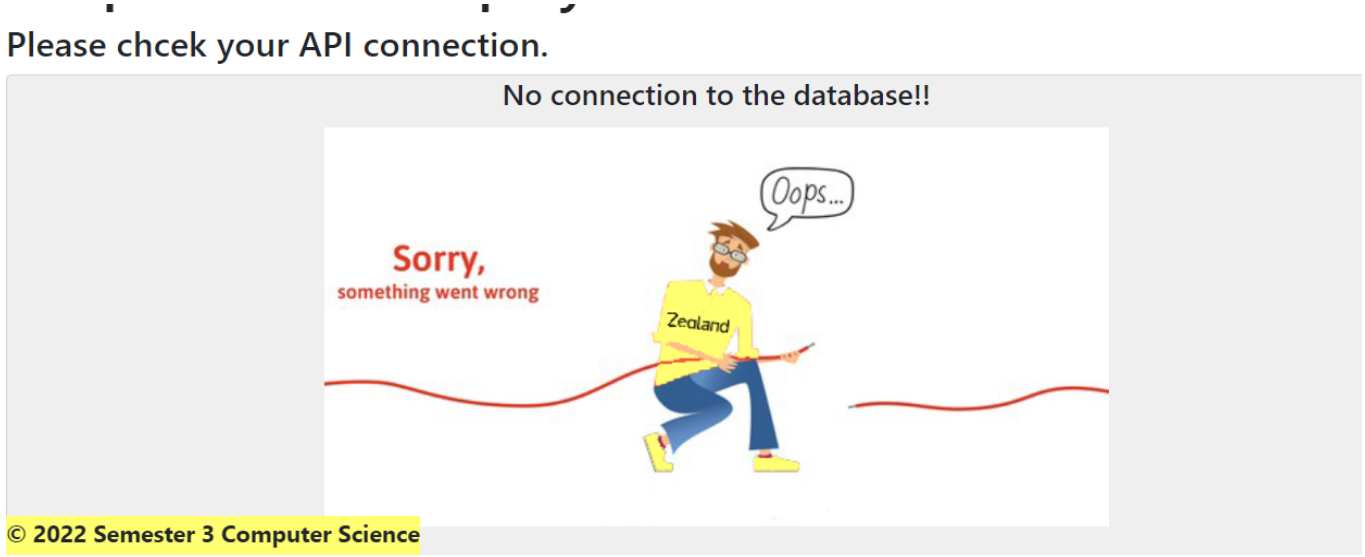
While designing the GUI we took into consideration several gestalt law principles. Similarity and proximity, symetry principles are used in button shapes, cards, and colours. E.g, All the main filtering buttons share the same size, shape, and colour so it’s easy to assume that these are clickable buttons that provide similar functionalities. Main data displaying cards are also similar in their format, size, colours, and shapes so users can quickly distinguish pictures and see crucial data about them.

We made sure to follow the C principle from CRAP, which is the contrast that was applied in a balanced way to make sure there is not too much contrast so we would avoid a confusing or visually jarring design.

Heuristics

In this project, it was very easy to apply heuristics, especially error prevention since the only user input is the Date field for filtering which is clickable so there is no possibility for the user to make any errors. We were able to achieve all core system functionalities while keeping an aesthetic and minimalistic design by staying focused on the essentials.

In case our system couldn't reach API services there is a sufficient picture prepared so we could keep the user informed about the current system status.



Extreme programming process - Oleg

Week 1

Our team is sitting together face to face with the product owners. Inside our team, we brainstormed together 3 ideas for another team for the final project.

Our ideas were:

- security camera - to create an application with a security camera and sensor which would be able to capture pictures and make short videos as soon as somebody is passing by the sensor.
- card reader - to create a web application where we can see how many people are present in a classroom. The students need to scan every time they enter and leave a classroom.
- rain sensor - to create a web application with the locations where we have a rain sensor. It displays which location is raining in real time.

After a short discussion our team decided to choose the project about security cameras. All team members agreed that this is the most interesting and challenging idea.

Our team as a product owner during the first meeting had to brainstorm and prepare 3 ideas for another team for the final project.

Our ideas were:

- A postbox with a sensor which can count incoming mail(newspapers, magazines etc.) and notify the owner if there is something in a postbox.
- An application which uses a barcode scanner which is able to identify the item by the barcode and show all the information about that item.
- An application which is using sensors to track and count the amount of students which were late for the classes and create statistical reports for a specific period of time.

Week 2

Sprint planning

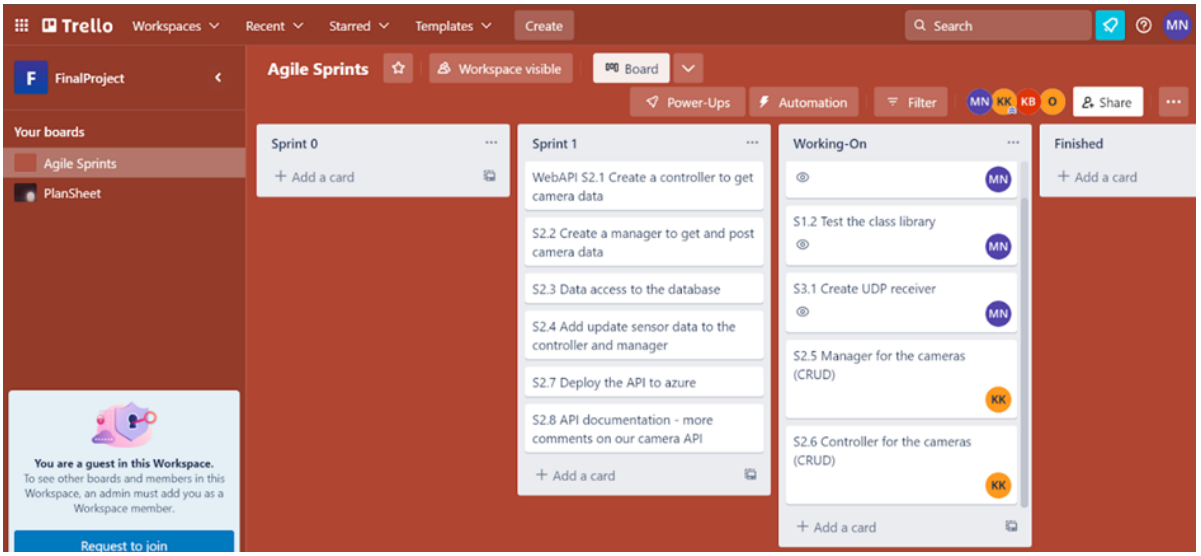
This week, we sat together with the product owner team.They told us which idea they want to work for and visa versa. After the meeting, Kengo created a folder in google drive where we as a team can put our works in one place. It is also an easy place to edit or write real time documents where everyone in the team has access to it. We also created a messenger group to contact each other outside working hours, if we have any questions. Together with the product owner team, we created a discord group where we can share documents like for example a product backlog, or in case we have a question. We exchanged our product backlogs, discussed it inside our team wherever we should change something or/and if we have some questions and afterwards we confirmed it together with the product owners

At the very beginning we tried to plan the overall process of creation and implementation of the project. Afterwards, we brainstormed user stories for the other team. In total we got 13 User Stories for the other team. For prioritising the user stories, we used planning poker. We implemented Moscow-principle, because it is easier for the other team to see what user stories we definitely want to be implemented. Then we wrote the acceptance criteria for each user story.

Our team also divided tasks for everyone. Some tasks can be done during the whole duration of the project, some need to be finished before the next meeting with the product owners. We decided that Maria will write the product manifest during the project, Oleg the introduction and Kengo the product description.

Kengo created a PlanSheet in Trello where we can create different tasks. He created tasks that we have to do before the next meeting with the product-backlog. Since there are not a lot of tasks, only Kengo wrote the explanation for how we came out with the ideas and Maria wrote the product backlog and the explanation for why we gave the user stories a particular priority.

Afterwards, we discussed in detail how we can split the user stories up in tasks and how we are going to successfully finish this project. We created in Trello a sprint planning with the tasks that we will do for sprint 1. Kengo set up a sprint planning in Trello with tasks we have to do for sprint 1.



For time estimation for user stories our team used a three-point estimation system where every team member gave their personal time estimates for each user story which gave us an approximate overview of how fast we would be able / or should finish each user story.

Initially it was important to create the basis of the project on which we would be able to expand the project further. Furthermore, at the very beginning we had to familiarise ourselves and test all our Raspberry Pi components, such as PIR motion sensor, Raspberry Pi Camera, and display.

Sprint 1 (Week 3)

As a group, we work on the tasks that are assigned in Sprint 1 on Trello.The Class Library for the camera was created by Maria. With this, Kengo could start writing the manager and the controller for the cameras using the CRUD-operations as well as working together with Kacper and Oleg on the Raspberry Pi. Together they looked at how the Raspberry works and how to use the camera and the video using bash scripting.

Afterwards Kacper and Maria looked for how to connect the Raspberry Pi with the client using Putty and Filezilla.

The UDP-receiver was coded by Maria and was tested by the whole team if it connected the Raspberry Pi with the receiver.

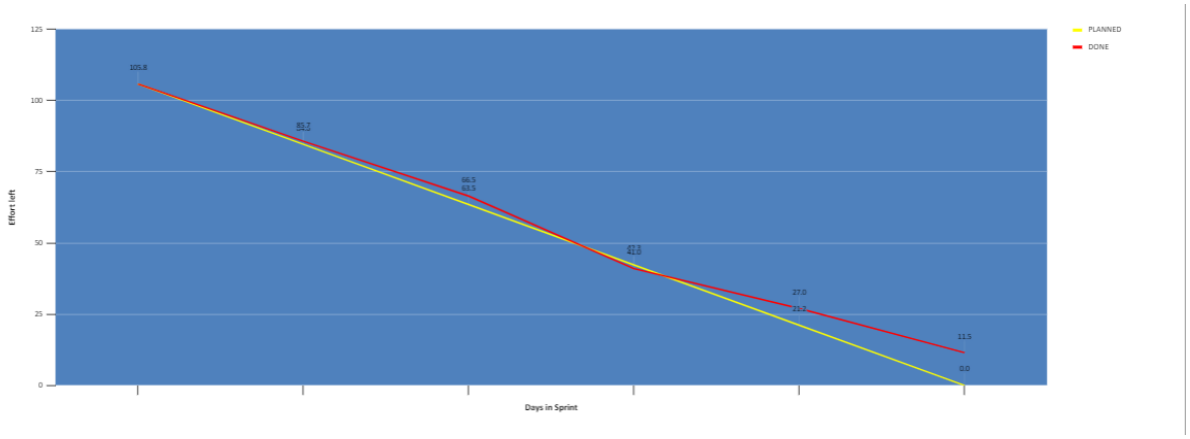
As a group, we created a github where we can share code in one place. It makes it also easier to Pair Program all the code we make together.

	Sprint 1			Effort "Day 0" calculated using three time estimates (or PERT)					105.8
Story ID / Task ID	Stories / Tasks / Planned	No. of team member per task	Total amount of work	Read note i cells for calculations					No. of Team members: 4
				Estimate effort in hours					Daily working hours: 5.5
				A	B	C	M	Sd	No. of Sprint Days: 5
			0				0	0	Available hours of work (resources): 110
S1.1	S1.1 Create a Class Library	2	5.4	2	3	2.5	2.7	0.1	
S1.2	S1.2 Test the Class Library	3	8.1	2	3	2.5	2.7	0.1	
S2.1	S2.1 Create a Manager in the WebAPI	2	5.8	2.5	3	3	2.9	0.1	
S2.2	S2.2 Create a Controller in the WebAPI	2	5.6	2	3	3	2.8	0.2	
S2.3	S2.3 Add update sensor data to the controller and manager	2	3.8	1.5	2	2	1.9	0.1	
S2.4	S2.4 API documentation – more comments on our camera API	2	4	2	2	2	2	0	
S3.5	S3.1 Create UDP receiver	2	8.6	4	4.5	4	4.3	0	
S3.2	S3.2 Test UDP receiver	2	5.2	3	2.5	2.5	2.6	-0.1	
S3.3	S3.3 Refactor and comment UDP receiver	2	3.8	1.5	2	2	1.9	0.1	
	Repository creation	3	1.5	0.5	0.5	0.5	0.5	0	
			0				0	0	
	Connect camera and motion to Raspberry Pi	2	2	1	1	1	1	0	
	Sensor and camera testing	2	4.4	2	2	3	2.2	0.2	
	Raspberry Pi testing with Putty and FileZilla	4	11.6	3	3	2.5	2.9	-0.1	
			0				0	0	
	Sprint planning	4	16	4	4	4	4	0	
	Tutorials and reading materials	4	20	5	5	5	5	0	

During this sprint we tried to stick to the plan and we managed to complete almost everything planned.

Sprint 1 velocity	
Story	Estimation
S1.1 Create a Class Library	2
S1.2 Test the Class Library	1
S2.1 Create a Manager in the WebAPI	2
S2.2 Create a Controller in the WebAPI	2
S2.3 Add update sensor data to the controller and manager	1
S2.4 API documentation – more comments on our camera API	1
S3.1 Create UDP receiver	5
S3.2 Test UDP receiver	2
S3.3 Refactor and comment UDP receiver	1
Total Estimation	17
Actual Velocity	14

There were some moments that we were not sure about and for which we needed more time to study, go through the tutorials and practice to complete. Thus, some of the tasks were not done during Sprint 1 so they were moved to Sprint 2.



Sprint 2 (Week 4)

During this week we had a couple of meetings where we continued to work on previously assigned tasks and user stories. Oleg and Kengo decided to focus more on Raspberry Pi components and Python code to establish a connection, take photos and convert them into JSON and send it to API while Kacper and Maria continued to work on API, Web Application and Database.

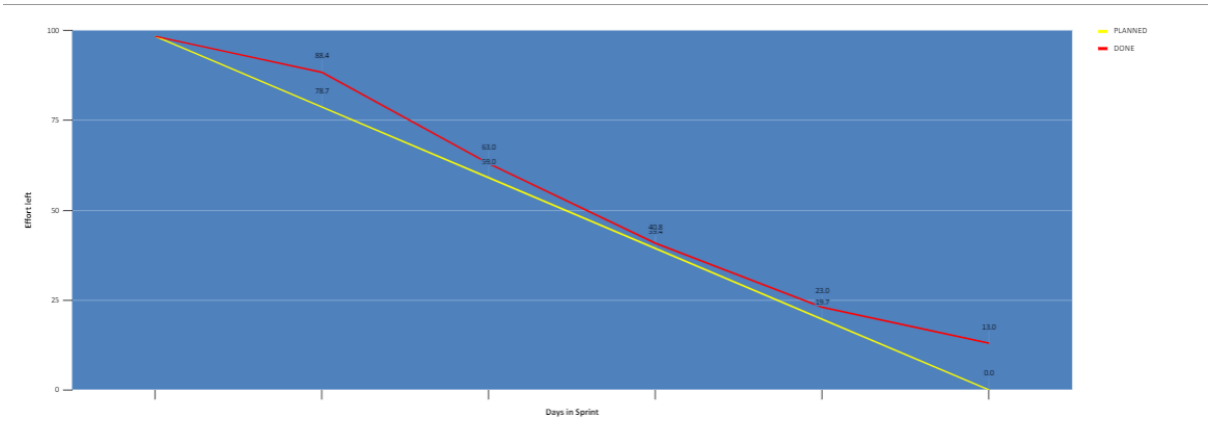
At one point we noticed a strange behaviour of our PIR Motion Sensor as it started to give a lot of false triggers. After extensive investigation, reading of many tutorials and topics on that matter, and tests, we concluded that our sensor is defective, and we would not be able to fix it ourselves as it would require soldering which none of us had any knowledge of. We decided to order a new sensor and continue to work as planned.

After consultation with Morten, we managed to successfully convert photos taken by our camera to JSON file, send, receive, and decode it back to photo. Due to the large size of JSON received after photo encoding, and the fact that TCP has built-in systems to check for errors and to guarantee data will be delivered in the order it was sent, we decided to use a TCP connection between Raspberry Pi and server, which would send data further to application, instead of UDP as we could not afford to lose any data during the transmissions of packets so we can get the same photo which was sent.

Sprint 2				Effort "Day 0" calculated using three time estimates (or PERT)					98.4
Story ID / Task ID	Stories / Tasks / Planned	No. of team member per task	Total amount of work	Read note i cells for calculations					No. of Team members: 4
				Estimate effort in hours					Daily working hours: 5.5
				A	B	C	M	Sd	No. of Sprint Days: 5
			0				0	0	Available hours of work (resources): 110
S4.1	S4.1 Create a Vue.js app	2	5.4	2	3	2.5	2.7	0.1	
S4.2	S4.2 Functionality to get data from the API	2	5.4	2	3	2.5	2.7	0.1	
S4.3	S4.3 Find API and implement functionality to get data from the weather API	2	7.2	3	4	3	3.6	0	
S3.1	S3.1 Create TCP receiver	2	5.6	2	3	3	2.8	0.2	
S3.2	S3.2 Test TCP receiver	2	3.8	1.5	2	2	1.9	0.1	
S5.1	S5.1 Create a database in Azure	2	4	2	2	2	2	0	
S5.2	S5.2 Create Model Classes	2	2	1	1	1	1	0	
S5.3	S5.3 Add the relationships in the database	2	4.8	2	2.5	2.5	2.4	0.1	
S5.4	S5.4 Design the database	2	3	2	1.5	1	1.5	-0.2	
			0				0	0	
	Python code to convert photo to JSON	2	8.6	5	4	4.5	4.3	-0.1	
	Testing JSON	2	4.6	2	2.5	2	2.3	0	
	Sensor troubleshooting	4	16	4	4	4	4	0	
	Tutorials and reading materials	4	12	3	3	3	3	0	
	Sprint planning	4	16	4	4	4	4	0	

We had to deviate from the plan and spend more time to move from previously designed UDP receiver to create and test TCP receiver and to figure out how to encode photos to JSON format. For a number of the above reasons, we spent more time on some tasks than originally planned and some tasks were moved to the next sprint.

Sprint 2	
STORY	ESTIMATION
S4.1 Create a Vue.js app	5
S4.2 Functionality to get data from the API	1
S4.3 Find API and implement functionality to get data from the weather API	2
S3.1 Create TCP receiver	1
S3.2 Test TCP receiver	1
S5.1 Create a database in Azure	1
S5.2 Create Model Classes	2
S5.3 Add the relationships in the database	1
S5.4 Design the database	5
Total Estimation	19
Actual Velocity	15

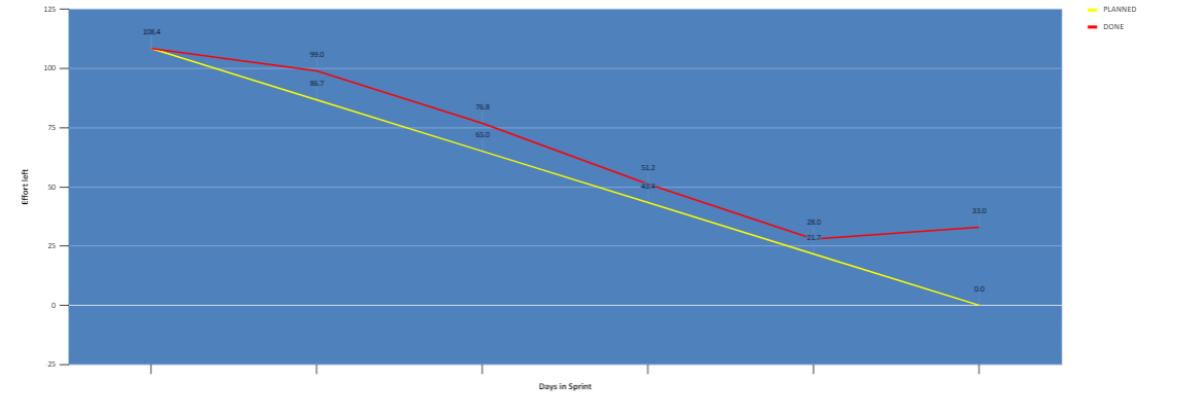


Sprint 3 (week 5)

During this week we continued to work on not yet finished tasks and errors from previous sprints.

Sprint 3									Effort "Day 0" calculated using three time estimates (or PERT)		108.4
Story ID / Task ID	Stories / Tasks	Planned	No. of team member per task	Total amount of work	Read note i cells for calculations						
					Estimate effort in hours					No. of Team members:	4
					A	B	C	M	Sd	Daily working hours:	5.5
										No. of Sprint Days:	5
				0				0	0	Available hours of work (resources):	110
S6.1	Create a Python app to receive the data from the Raspberry Pi		2	5.4	2	3	2.5	2.7	0.1		
S6.2	Create functionality for the camera		2	5.4	2	3	2.5	2.7	0.1		
S6.3	Create functionality for the sensor		2	7.2	3	4	3	3.6	0		
S6.4	Add functionality to send the sensor data via TCP		2	5.6	2	3	3	2.8	0.2		
S2.5	Test the manager of the API via Unit Testing		2	3.8	1.5	2	2	1.9	0.1		
S2.6	Test the controller of the API via Postman		2	4	2	2	2	2	0		
S7.1	Create a Front-End prototype		2	4.4	2	2	3	2.2	0.2		
				0				0	0		
	Sensor troubleshooting		4	20	5	5	5	5	0		
	Camera troubleshooting		4	20	5	5	5	5	0		
	Compress JSON to reduce size		2	8.6	5	4	4.5	4.3	-0.1		
	Database connection troubleshooting		4	24	6	6	6	6	0		
				0				0	0		

2



Sprint 3	
STORY	ESTIMATION
S6.1 Create a Python app to receive the data from the Raspberry Pi	7
S6.2 Create functionality for the camera	5
S6.3 Create functionality for the sensor	5

S6.4 Add functionality to send the sensor data via TCP	1
S2.5 Test the manager of the API via Unit Testing	1
S2.6 Test the controller of the API via Postman	1
S7.1 Create a Front-End prototype	2
Total Estimation	22
Actual Velocity	22

Quality assurance - Maria

We use Quality Assurance because it is a very good process to verify that our project satisfies the product owner's stated standards. It is a must for us to implement it, if we are working according to our Definition of Done. To assure quality, as a team, we have to agree that we can't refactor the code, the code is working as we wanted, the product owners gave feedback and it is tested.

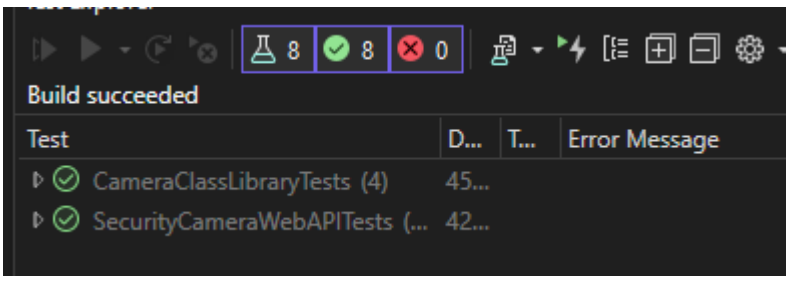
Refactoring is a technique of rearranging code without affecting its original functionality. Its purpose is to improve internal code by making many modest changes that do not affect the code's exterior.

During our project, before making any changes or adding new features to old code, we considered refactoring. Going back and cleaning up the current code before adding new programming will not only increase the project's quality but will also make it easier for future developers to build on the original code. We used refactoring in some parts of the code but not everywhere where there was no need. But when we refactored, we are only finished when the code is readable for everyone and it still functions like before the refactoring.

The feedback helped us determine the priorities and the design and hence improve the final project. With this, we didn't have fast misunderstood assignments and helped us regularly identify areas for improvement. It also increases the productivity inside the group and we could give a satisfactory product to the product owners.

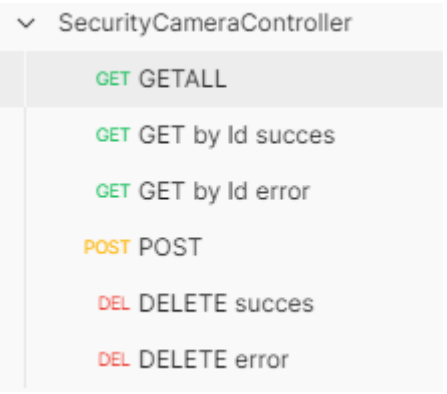
Unit Tests

A unit test is a method of testing a unit, which is the smallest amount of code in a



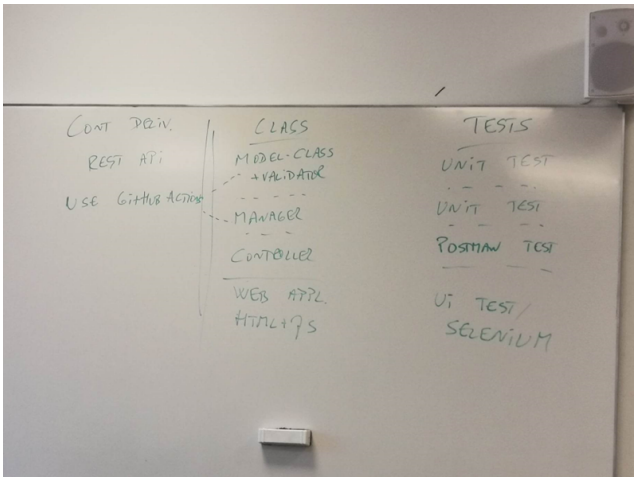
system that can be logically separated. For this, we Unit Tested the Validate-methods of the Class Library. Not only are we ensuring the code meets quality standards before it's deployed to the API but the validate tests guarantee that a system performs as expected by running all of its functions and collecting measurable results.

The manager class was also Unit Tested. It is important to know that the HTTP methods are correctly working and we can deploy the class to other parts of the API.



The controller class was tested in Postman. Here, we could see if the controller can interact with the manager class. We also test what happens during exceptions, different status codes and what it should return in order to fulfil the HTTP protocol's response status codes. You can also test the controller with Swagger, which is a very easy testing method that has an interacting user interface.

The web application was tested using Selenium. The test cases were mostly the buttons we created in the frontend. We looked if the button delete worked or displayed pictures of only today. Overall, we didn't cover all code for the web application.



One of the most significant advantages that we found is that it may be executed whenever a piece of code is modified, allowing bugs to be fixed as soon as feasible.

System Testing

It is the process where our group assessed how the various components of an application interact with one another in the overall, integrated system. After implementing and testing each component separately, we performed a system test in which the entire system was tested as a unit. We assessed how the various components of the overall system interacted with one another. Azure was used to hosting the database and the API. The remaining components were linked to a local network (mainly the router in the classroom) that provided internet access. As a group, we followed the procedures outlined in the test case table. We completed all processes successfully and with the expected outcomes.

Acceptance Testing

It determines whether the system is ready for release. During our project, requirements changes can be misunderstood in ways that do not fulfil the product owner's intended demands. During this final phase, the product owners will test the system to see whether the

application meets their needs of the product owners. After this process is done and the software has been approved, the program will be delivered to production.

As you can see, the depth of these tests is yet another reason why bringing in software testers early is critical. When a program is thoroughly tested, a greater number of flaws are discovered, resulting in higher-quality software.

Conclusion - Kengo

Overall, after careful analysis of how we executed our planned methodology of working around agile Extreme Programming, we would have to agree that it was a good option to go with for a project such as this. Evidence of this can be shown throughout the report, especially when talking about things such as the development process and sprint planning. Not only that, but keeping with the sprint planning of splitting our workloads into separate sprints also helped us as a team to keep up with the work whether we physically met or not and not lose the tempo of getting the work done on time. Also, the architecture that we were first given was a good architecture to base our work on, although we had to change minor details such as using TCP instead of UDP, it still gave us a very good understanding of where we should start and what way we should follow to accomplish our goals. Ultimately, we were able to further develop our knowledge about the XP methodology as we were working within it and the project can be seen as very valuable since it allowed us to not only recover all the things that we had learned during the semester but also further deepen our knowledge in areas that we struggled on or had to learn for the creation of this specific product.