

University College Nordjylland  
AP Degree Final Project  
5<sup>th</sup> Semester



Phaethon

Collaboration with: Means SIA

Participants: Ralfs Zangis, Andrei-Eugen Birta, Adam Blazsek

Supervisor: Jesper Strandgård Mortensen

Submission date: 13-01-2019

Table Of Contents

**1. Introduction..... 4**

    a) Problem Statement.....4

    b) Proposed Solution.....5

**2. Preliminary Study ..... 5**

    a) About the Company .....5

        I. Mission .....5

        II. Vision.....5

    b) ROI Analysis.....5

    c) SWOT Analysis .....6

    d) Stakeholders Analysis .....7

**3. System Development ..... 8**

    a) Development Method .....8

        I. More In Dept Review .....8

            A. Plan Driven Development .....8

            B. Agile .....9

            C. Extreme Programming.....9

            D. Scrum .....9

            E. Kanban .....10

        II. Methodology Selection .....11

    b) Workflow Analysis .....12

        I. Steps of Development.....12

        II. Tasks.....12

    c) Quality Assurance .....13

    d) Working culture .....14

    e) System Design .....15

        I. System Architecture .....15

        II. Domain Model.....19

        III. Database Model .....21

**4. Technology and Programming ..... 22**

    a) Analysis .....22

        I. Framework .....22

            A. Technologies Considered .....22

            B. ASP.NET Web API.....22

            C. WCF (Windows Communication Foundation) .....23

        II. Database .....23

            A. Choices in Databases .....23

---

B.	Relational Databases .....	23
C.	Non-relational/NoSQL .....	24
D.	Database Engines.....	24
III.	Client .....	25
IV.	Middleware .....	28
A.	Clients-System .....	28
B.	System-Database .....	28
<b>b)</b>	<b>Implementation.....</b>	<b>30</b>
I.	Security .....	30
A.	Access to resources.....	30
B.	Resource transport .....	31
C.	HTTP security headers .....	33
D.	Tokens.....	35
II.	Fake Server .....	35
III.	Decorators.....	37
IV.	Multi Language.....	39
<b>c)</b>	<b>Tests.....</b>	<b>42</b>
I.	Unit Tests .....	42
II.	Integration Tests .....	42
III.	Acceptance Tests.....	42
<b>d)</b>	<b>Encountered Difficulties .....</b>	<b>43</b>
<b>5.</b>	<b>Company Feedback.....</b>	<b>44</b>
<b>6.</b>	<b>Conclusion .....</b>	<b>45</b>
<b>7.</b>	<b>References.....</b>	<b>46</b>
<b>8.</b>	<b>Appendix.....</b>	<b>48</b>
a)	Original Latvian Feedback By Means.....	48
b)	Contact Info .....	49
c)	Group Contract .....	49

---

## 1. Introduction

This document summarizes the collaboration of this group, for the Final Academy Profession (AP) Degree Exam of the 5<sup>th</sup> and final Semester.

This group is composed of three individuals, of different nationalities and vast interests. Despite the differences among us, we managed to reach a consensus and create a set of rules upon which we would further conduct our work. All of which can be seen in the section, called “Group Contract”, attached hereto.

The purpose of this project, was the creation and implementation of a software solution, that Means SIA, a PC repair and sale business, based in Tukums, Latvia; would be using as their main resource management tool. Feedback from the company, along with the details of the contact person we have spoken with, can be seen in the “Company Feedback” section of this document.

This project started its development on 3<sup>rd</sup> of November 2018 and was due on 14<sup>th</sup> of January 2019. Means contacted one of our team members, sometime in the middle of October 2018, to enquire about the creation of a software solution which would solve their management and organizational problems. In this report, we will present you our proposed solution and its development process.

### a) Problem Statement

As a company in the business of both private and retail sale of computers, peripheral units, and software for specialized stores, some of the problems they are currently facing are mainly organizational but also include human error related issues which we aim to lower with our solution. Our objectives are to provide a more modern approach to frequent tasks like:

- Managing various stocks (computers, computer parts, etc.)
- Managing sales (including management of invoices)
- Managing suppliers and the items they are supplying
- Managing employees and their salaries
- Managing tasks (repair, maintenance, etc.)

The above is a small list of tasks that the company, currently, finds hard to achieve on a regular basis and the current systems (the classic pen and paper, multiple software solutions) has proven to have major setbacks such as:

- Slow input/output capabilities
- Hard to work with
- Ugly hand writing can cause major problems
- No scalability and reusability
- Heavily relying on human memory to remember various deadlines
- Missing use cases in the current solutions
- Poor data management as data is scattered across multiple databases

Although Means was very open about the final solution, one of the constraints introduced later on was the need to be hosted on a Local Area Network (LAN) and it needed to be as lightweight as possible

without the need of application installation on end user computers. Our current solution can be used by multiple clients, like on mobile where employees can see their progress on the go

By providing an application that the company lacks, we can allow the employees to finish the tasks in a more reliable and timely manner.

### b) Proposed Solution

We aim to achieve our objectives, by creating a distributed software solution, using REST-full Web API as our back end, and an MVC based web client as our main front-end solution. Due to time constraints we decided to only focus on the hosted application but we plan on rolling out several other front-end clients and mobile compatible apps. All to facilitate the company's future needs and ease both their employees and the companies' leader work.

## 2. Preliminary Study

### a) About the Company

"Means SIA", is a small company, based in Tukums, Latvia. It is currently thriving to succeed in the computer sales, assembly in both the private and wholesale business. On top of acting as a computer and computer parts shop, they also provide repair and maintenance services, to customers.

#### I. Mission

The company's mission focuses on the current or short-term goals.

In our case, Means' mission is to provide customers with the cheapest option in Latvia who are looking to repair or buy electronic devices. With over a decade of being in the consumer electronic devices business, Means offers the best option for customers on a budget.

#### II. Vision

The company's vision refers to the long-term objectives.

Means' vision is to expand their current stock and offer a wider variety of items and become the leading online store for consumer electronic devices in Latvia and possibly reaching out the Baltic region.

### b) ROI Analysis

The computer retail company Means, currently has a 20 – 30 thousand Euro monthly income, although they don't expect to see any increase in revenue, they do expect a major reduction in the time it takes to handle their daily tasks. As the application is custom made for Means, we first needed to fully understand where the bottleneck is. From our preliminary investigation we found out that the company is having problems with properly tracking their products, invoices and tasks. As reported, they want a better understanding of their current inventory (information such as the progress of a specific task and who handles it, etc.) and to have a more streamlined experience when it comes to handling their invoices (both incoming and outgoing). The application would also create a better link between the customer and its product, further minimizing the chance that a product is shipped to the wrong customer, which was a request by Means as human error is currently a big issue they are looking to overcome. Means also expects this application to reduce the number of employees needed for a given task.

While we cannot promise Means that our solution will solve all their current problems, we can however focus on analyzing their current business model and tailor the best possible solution to their needs and we assure that the amount of time needed for an inventory search will be reduced greatly. At the time of writing this report we have not yet received the self-reported reduction in time it takes to search for a specific item in the inventory, handling of invoices and item management.

### c) SWOT Analysis

SWOT analysis gives the company (but can also be used for decision making purposes) a strategic perspective to identify its strengths, weaknesses, opportunities, and threats related to its competition. Strengths and weakness are often internally-related, focusing on (but not limited) management and its relationship with its employees, the size of the company etc., while opportunities and threats commonly focus on the external environment, such as the economic outlook or business competitors.

	Strengths	Weaknesses
	1. The shops location 2. High profile customers 3. Long term contracts	1. Disorganization 2. Number of employees 3. Current job market
Opportunities	Use strengths to take advantage of opportunities	Overcome weaknesses by taking advantage of opportunities
1.Brand recognition 2.Online store 3.Networking	Use high profile customers for networking	An online store would require employees from a different background
Threats	Use strengths to avoid threats	Minimize weaknesses and avoid threats
1.Modern computer designs 2.Online only shops 3.Throwaway culture	Long term contracts lock customers to Means even after the computers are not repairable	Embrace the throwaway culture by incentivizing it thus driving the need for more employees

**Strengths:** With a strategically well-placed headquarters Means is located near the city center which is easily approachable by foot. Thanks to several long-term contracts with various high-profile customers they are ahead of the competition both in terms of reputation and brand recognition.

**Weaknesses:** With disorganization currently plaguing the company (not helped by the outdated methods they are using to solve their issues, none appropriate for their use case), it's their number one reason for reaching out to us. Sadly, the current job market does not aid its lack of employees with the number currently reaching four.

While only one major weakness can fully be solved by our solution, we aim to reduce the need for more employees by speeding up the pace at which the company currently operates.

Opportunities: Brand recognition being one that should be capitalized on the most, with proper networking Means could easily expand their current portfolio. As for the private customers, an online store would greatly benefit the sales figures as not just the PC market, but overall shopping trends have changed to the more convenient digital form.

Threats: With competitors offering online accessibility, we certainly think Means is backed into a corner and has no other options but to acquire an online shop to avoid getting beaten by its rivals.

One downside of SWOT is that it only offers a current view of the company and would need to be updated every time the company or its external environment changed, thus its only mainly used as a starting point for identifying areas for improvement. In our case this meant identifying problems possibly not yet recognized by Means but still solvable by our solution presented in the application. We were happy to report that the need for more employees can be aided by our application, but do advise Means to focus on the growing need of implementing an online store which could be an addition using our application's backend as the administrative basis.

#### d) Stakeholders Analysis

In this section we will go into greater details describing the various stakeholders of the product. In total there are four stakeholders: the employees, the manager, the suppliers and the customers.

Stakeholders:

1. Manager - interested in more productive employees
2. Employee - wants to make existing processes more streamlined and less time consuming
3. Customer - requires the same results as before, but isn't interested in how its achieved
4. Business partners - interested in company to succeed and continue cooperating

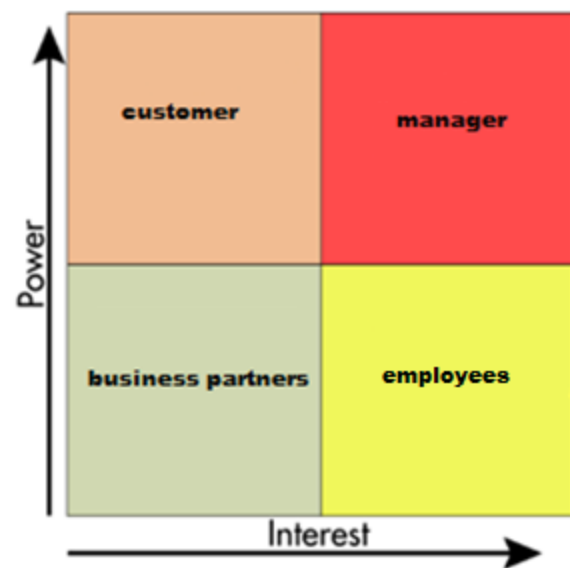


Figure 1 Stakeholders diagram

Due to the fact that the manager and the employees hold the most power and interest we decided to focus only on the two of them:

Employees: making them more productive and reducing the unnecessary work done on day to day basis  
The manager: making management of company/employee easier and furthering the company by increasing their profits. The decision we made to focus on only 2 stakeholders allowed us to make better performing program, which is specifically made for required use cases and let us focus more on quality rather than quantity.

---

### 3. System Development

#### a) Development Method

At a high-level overview the software development process can be separated into two main methodologies:

**Plan Driven:** All the steps are carefully planned, variables such as budget and delivery times are fixed early on in development and overall progress is compared to these predefined criteria.

**Agile:** All the steps are done in small but frequent increments with minimal initial planning and are modified according to the constant feedback and changes in requirements

#### I. More In Depth Review

True to its name, plan driven development requires extensive research with precise planning before beginning the development of the application. The project requirements are researched and defined early on in the first or early phase of the project and the team are expected to strictly adhere to them as it makes following the progress of development easier for the customer. While it sounds like a good choice, requirement changes and issues with development might upset the progress and cause problems due to a fixed budget and deadline and it might cause a degrade in quality of the finished product.

##### A. Plan Driven Development

Two of the most well known plan driven development methods are UP (Unified Process) and the Waterfall method. We will focus on UP more instead of the Waterfall method as it was more relevant to us as it was closer to what we would be comfortable using. The Waterfall method compared to UP is more strict in its process as everything needs to be done sequentially. UP is more free in this regard as it is an iterative and incremental process.

It has been separated into four main phases, each focusing on a specific part of the development: inception, elaboration, construction and finally transition. Each phase is divided into iterations (the number depends on the project size and the size of the team). Each iteration incrementally improves the current version of the system. Compared to the agile development methods UP's iterations last a lot longer (in some cases it can span over a month).

The inception phase is usually the shortest and will most likely have a single iteration. In this phase the preliminary product cost is estimated, a work plan is created and the team investigates the feasibility of the overall project.

UP is tremendously architecture focused, which is one of the main goals of the elaboration phase: to properly determine and approve the architecture of the system. The main goal of the elaboration phase is to establish an architecture baseline, which is the vision of the application that includes all the significant architecture components. To achieve this, use cases are identified and architecturally significant ones are chosen, this process results in a use case diagram being made. In this phase the schedule and cost estimate is defined accurately.

The largest phase is the construction phase, during which the final system is built. The foundation for this phase has already been determined in the elaboration phase. Use cases are more accurately



described and remaining UML diagrams are created, an example would be the activity or a system sequence diagram.

Transition is the final stage of UP development cycle. The system is deployed and the target users begin to use the it.

### *B. Agile*

In stark contrast to the aforementioned Plan Driven methodologies, Agile introduces a more dynamic development environment. In Agile the developers accept that very little requirements will be defined on day one, and the ones that are, will possibly change based on customer/owner feedback and feature priority values. Agile methodologies acknowledge that requirements are only truly fixed for the duration of one sprint.

Receiving constant feedback is key, so short iterations known as sprints are introduced. The sprints are usually a lot shorter than the iterations found in Plan Driven methodologies, typically ranging from one to three weeks depending on the tasks and their complexity.



*Figure 2 Representing various sprints*

### *C. Extreme Programming*

Extreme Programming (XP) is an agile development methodology which shifts its main focus on the overall quality of the software. XP is based on four main values:

1. The first and arguably the most important value is communication. Communication is crucial in any methodology, but especially in Agile as the requirements can change rapidly.
2. Simplicity, the simpler the solution the less chance something fails. Development is to be started with a simple solution and improved with continuous refactoring (also one of the key XP practices).
3. Feedback is the 3rd value which is a must for high product quality.
4. The final value is courage. It encourages making decisions without the need of a formal process. This is the complete opposite with Plan Driven methods like UP where every significant change is accompanied by an abundance of documentation.

XP lists 12 key principles for creating a high quality software. Among the 12 principles is test driven development or TDD for short. TDD requires writing the tests before developing the product. This ensures the number of errors/bugs is minimized in the later stages of production. XP takes the Agile incremental philosophy into programming with small sized releases and continuous integration. The small releases lets end users test the system significantly earlier compared to Plan Driven products, even though they only include part of the intended functionality.

### *D. Scrum*

Scrum is one of the Agile development methods most suitable for small development teams working in sprints ranging from a single week up to a month. At the end of every sprint, value should be presented to the customer by being one step closer to completion. Unlike XP (Extreme Programming), the scrum methodology focuses only on the development process. It defines three roles: Product owner, who represents the product stakeholders (eg. the company receiving the final product). The second role is

---

the scrum master (usually a single person, but it can change over sprints) who is responsible for the scrum process and removing any distractions that the team might encounter. The final role is the team, which represents all of the development team who design, create, implement and test the system (the order is not necessarily way fixed).

Before at the start of each sprint, a sprint planning meeting is held. In this meeting the product owner selects the tasks that should go into the sprint backlog based on the current state of the application and the feedback given by the final users/ customers.

Each sprint is closed with a sprint review where the product owner shows off the accomplishments to the stakeholders. An internal meeting is also held to discuss and improve upon the development process itself.

Scrum also includes daily standup meetings where the members discuss what they have done since the last meeting, what they will be working on and what problems did they run into. These problems are to be resolved by the scrum master.

Besides the usual sprints there is also a special initialization sprint called "sprint zero". Sprint zero is used to set up the work space for the project to begin. Investigate any problems which can potentially slow down the process (such as unknown technology), set up the development environment. Unlike UPs inception and elaboration phases the requirements set in sprint zero are not final and are expected to change over the course of the development.

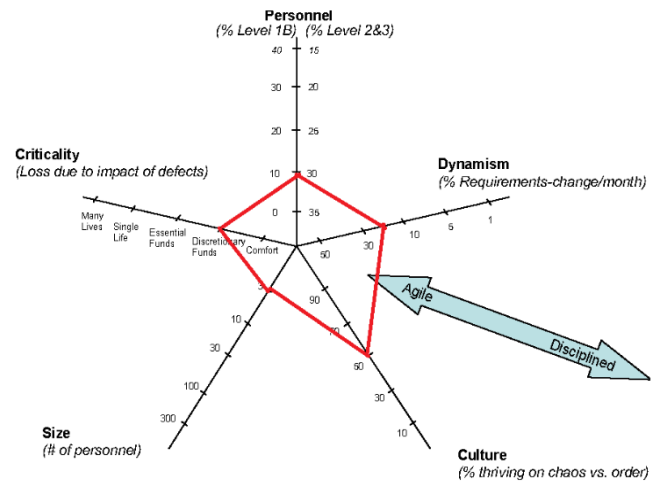
#### *E. Kanban*

As with the previous two agile development methodologies, Kanban aims to work with the chaotic nature of variable requirements and the need to shift focus between priorities quickly. Kanban also focuses on making the development as smooth as possible by making transparency, respect and agreement as its values (there are a more values which are equally as important but these are the ones we aimed to retain when considering the method).

Leadership is an integral part and is encouraged at every level (again circling back to its values) and the changes are smaller, more evolutionary rather than revolutionary. Frequent deliveries in small sizes is a must, and the quality must be up to scratch with the rest of the teams work. The scope and progress of the development work is visualized using a Kanban board, where teams agree to which tasks to choose. If a task happens to last longer then the intended time, it will be put back to the backlog and can be chosen again. Making policies explicit is a must. Policies, such as the definition of "ready", or the definition of "done", should be used rarely for maximum impact. They're used to ensure a common ground across both the team and the stakeholders. Implementing feedback loops to ensure the upmost relevant tasks are implemented, is a good way to minimize time wasted on less important tasks. Those feedback loops include: Strategy Review (Quarterly), Risk Review (Monthly), Replenishment Meeting (Weekly), Kanban Meeting (Daily) and Delivery Planning Meeting (variable)

## II. Methodology Selection

Choosing the right development method is key to a successful product. One must first assess the project, the team, the project scale and the customer of the final product. As our team was small in size (even compared to Agile standards), our requirements changed quite a lot, and we didn't have the time to properly plan out every single move, we alongside the base conception of the Plan Driven methodologies, decided early on we would choose an Agile approach to developing our system.



The following Bohem and Turner chart, resulted us confirming the need to follow an Agile development method.

Figure 3 Bohem and Turner diagram depicting our team

So, we decided to create a unique combination of all the three major Agile development methods. One that would contain the following guidelines:

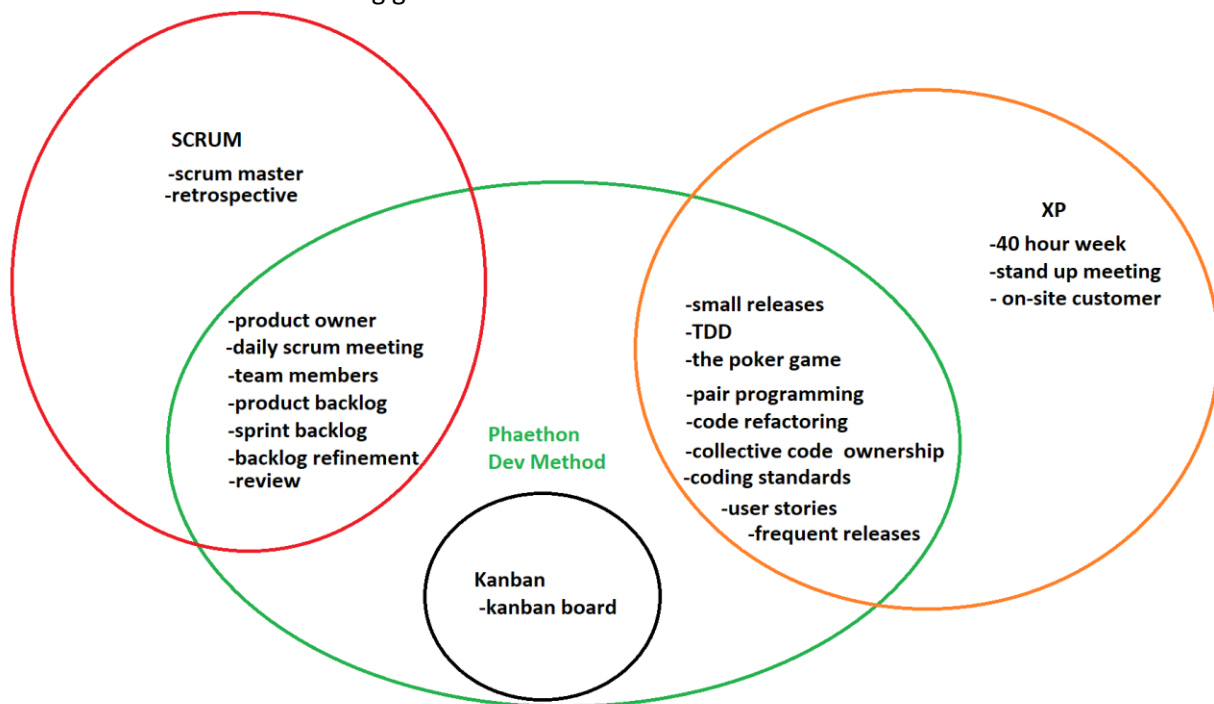


Figure 4 Detailed overview of all elements, of the dev method we used

Our aim was to first build a Minimal Viable Product (MVP). After the MVP has been finished, with small increments we added functionality without braking the core application.

---

We also put a heavy emphasis on communication, as we noticed early on the importance of keeping a shared and up to date understanding of the developments state.

Alongside communication, Pair programming, and TDD were used extensively. As expected TDD slowed us down in the early stages of development, but it proved to be a major contributor to the quality of the overall application and saved us from the headaches of debugging in the later stages where the application was becoming more complex.

### b) Workflow Analysis

After selecting our system development method of choice we almost immediately started working on our task board. The task board contains all the features we intended to implement in to the product. While we aimed to implement most of the features in the backlog we knew (due to time constraints) we would have to leave some features for a later release date or leave it to the next team who will support the product.

The backlog was created in full collaboration with company and prioritized the ones they deemed most important. The product owner stayed in constant communication with Means to ensure we were on the right development track. The product was shown to company many times to receive their feedback and use that feedback to further suite the company's need.

### I. Steps of Development

The consensus was that we would separate development into 6 steps (this decision was based, on our previous experience working with Agile methodologies and internship experiences).

When following a task, it must go through the following steps:

1. **Backlog** – Place where all not started tasks are kept;
2. **To Do** - Tasks for current sprints, with an already assigned working member;
3. **In progress** - Location, where the tasks which are currently being worked on are kept;
4. **In review** - Area where tasks are placed when they are considered to be finished, but still awaiting approval from other members;
5. **To deploy** - List of approved tasks, that are not yet deployed to the customer;
6. **Done** - Region which contains all tasks, which were deployed to the customer;

### II. Tasks

We referred to user stories as tasks most of the time as we never utilized them outside of creating (in sprint zero), implementation and visualization on the task board. We only had a single burndown chart in the end of our first sprint but abandoned it after we reevaluated out priorities and focused on creating an MVP.

To improve the usability of the task board we decided to utilize the following techniques:

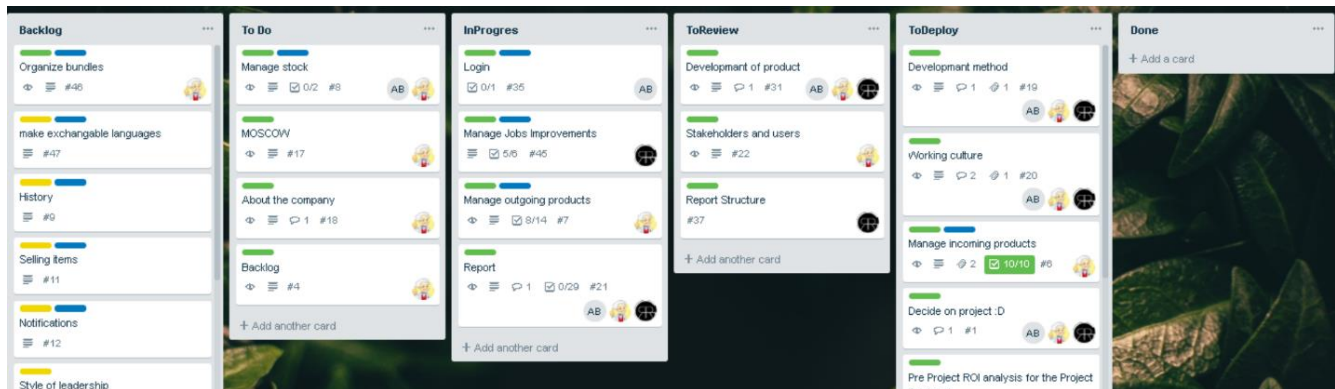


Figure 5 Kanban board, with MOSCOW based coloring

**MoSCoW:** Short for Must, Should, Could, Won't do. It's a task evaluation method used for categorizing the importance of a given task.

We used MoSCoW in the form of colored badges representing the priority:

1. Green - must
2. Yellow - should
3. Orange - could
4. Red – won't

**Task separation:** To make the backlog easier to navigate we decided to add a badge (using a unique colour) which would indicate if it's a programming or other, related task:

1. Blue - programming
2. No colour - other (eg: report related task)

**Descriptions:** Each task had a separate section where a more detailed description complimented the title to remind other members on the team what the task was about.

**Comments:** Knowing that communication is key when it comes to working in groups, it was important to quickly and reliably provide feedback on the state of the task, this was done by the utilization of comments, which allowed us to comment on the quality of the final implementation and possibly even reject it entirely but still provide adequate feedback without actually contacting the member.

**Check list:** When a complex task is presented which might include the need of having multiple parameters or sections which have to be met, a checklist is used to visually inform other users of what has or hasn't been implemented yet. A percentage is automatically shown on the task to indicate the finished progress.

### c) Quality Assurance

In this section we will guide the reader through the steps we took to ensure we handed over a quality final product to Means. While Agile emphasizes the importance of making development more dynamic and adhere more to the needs of the customer/product owner, we mustn't forget about the quality of that product.

---

With quality always in the back of our minds we made sure our tasks were FURPS+ (the plus can represent any extra functionality or parameter the customer or owner wants implemented) approved. Below are the quality attributes we adhered to.

Functionality – what the future users or owner of the application wants it to do.

Functionality represents the main functional requirements in our system, such as registering, managing invoices, viewing a profile, etc. This can also help us in choosing the right software architecture when considering options for not only current but future development. For Means this meant solving the previously mentioned points related to organizational difficulties (such as stock management)

The URPS attributes represent the other non-functional requirements which are architecturally important.

Usability – how effectively can the intended audience use it, focusing on the learning curve both in the user interface and experience

Usability in our case meant ease of navigation and the speed at which the user can access their point of interest such as invoices or upcoming tasks. We aimed for a clean and simple design although considering the target audience of our application (who are mainly computer oriented) we did not need to implement any special design elements to help or ease navigation (not taking languages into consideration as the final product would be a Latvian only version, but we knew that from day one).

Reliability – how robust must the application be when used by the user, also takes into account the up and downtime and the quality of its data.

Due to all customer related information is flowing through the application uptime is at upmost importance, but thanks to the simplicity of the environment in which it will be used in, we do not expect any major failures unless the network suffers downtime.

Performance – defines the parameters that the application must use when it comes to resources and responsiveness.

Performance should again not be an issue as the company has a low number of employees so the overall application is exposed to minimal stress at all times. On top of that, the application would run on a local network, eliminating network latency as a major performance issue.

Supportability – how easily the application can be supported by the current or future developers.

The application has been built with supportability in mind, this being the main reason why API documentation and one of the main reasons why so many tests were created.

#### d) Working culture

This document represents the sum of our “rules” and guidelines that we have set up, to better organize our work.

General rules and guidelines:

- Daily, scrum meetings, organized by slackBOT (although we later changed to Alice, due to slackBOT no longer being available for our communication server, under the same pricing)
- Weekly Refinement meetings are to be held, organized and conducted by the product owner. During these meetings, the team will discuss tasks, what they mean, what must be

- 
- done, how can they be done and ultimately: write a Definition of Done to said task. Only after a task is refined, it can be worked on and moved to its later iteration.
- Mind Slack channel's description and use them to the best of their purpose.
  - Several bots (Trello, Appveyor, Github and ScrumMaster) are set up to reduce the need to constantly check all used applications, notifying all members of new activity, in dedicated slack channels.
  - To keep a level of traceability, always work in new branches, based on the Development branch, with the same name as the task you're currently working on (including the Trello card number) ex: T-cardNumber cardTitle
  - When you believe a task is finished, create a pull request (always write a short description of what you did when creating the pull request), DO NOT MERGE WITH Dev or Master.
  - Once the branch is approved (passes Appveyor checks and gets a positive review), the owner of said branch, will merge with development branch.
  - Merging with master will be done once every sprint or when enough tasks are done.
  - Master Branch is the one we will send the supervisors to review, it is to be considered as ready to go branch.
  - Work from anywhere at any time but finish your tasks on time/as fast, as possible and attend scheduled meetings.
  - Code should be kept minimal as possible.
  - Variables, methods, classes and any other code-related elements, should be named properly and decently, following the CamelCase format.

## e) System Design

### I. System Architecture

Picking the best architecture that would fit our needs, was a pretty long and tedious process, due to the fact that the requirements from the company were not quite clear from the begging and they changed several times, during the development process.

We decided to present the evolution of our architecture through Umlet diagrams, as they would have been seen in different steps in early and later stages of development and finally the current solution.

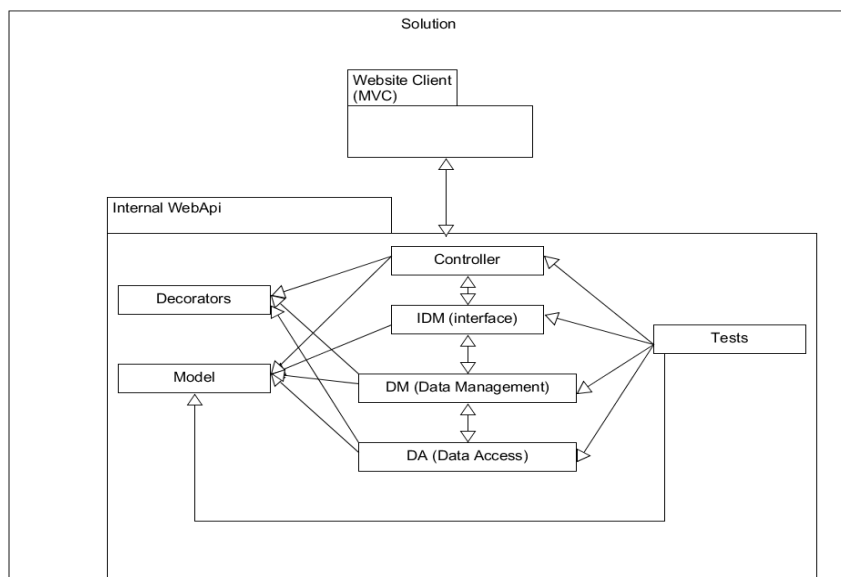


Figure 6 System Architecture diagram v.1

Figure 6, represents our very first drawn system architecture, and you can see that we were planning on a server-client type of system, due to the fact that Means wanted a system that did not require for them to constantly install/uninstall software on their employee's computers.

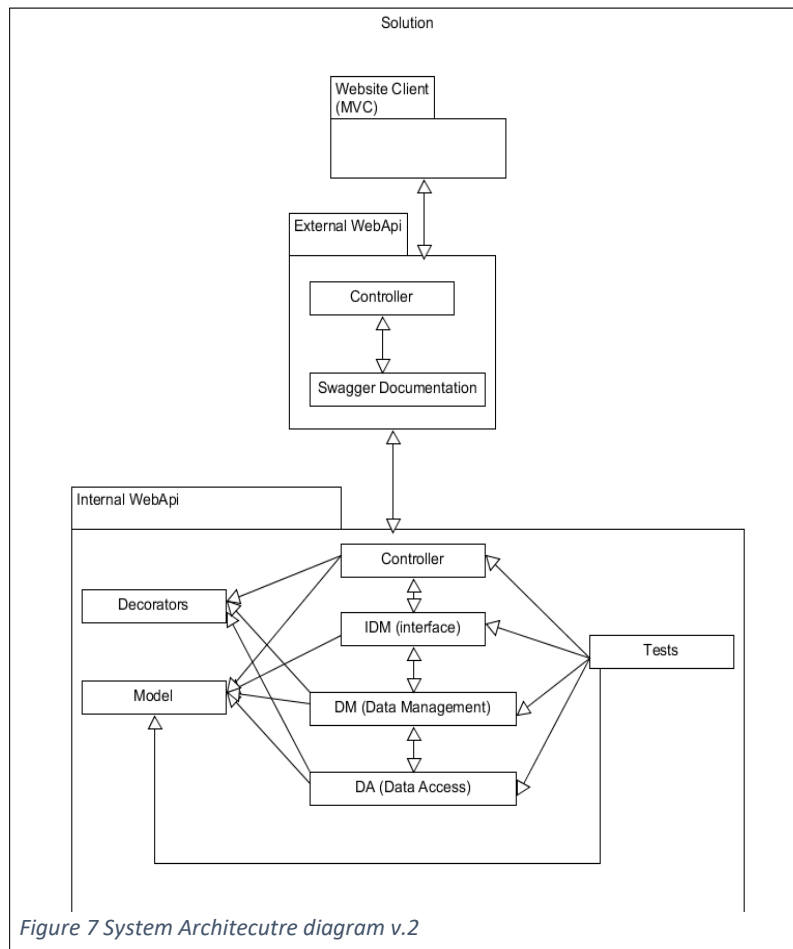
We decided that, in the begging we would have only a web client, built following the MVC framework.

As for the Server, we wanted to make a Web API, composed of several parts:

- Controllers (also known as Endpoints): which would deal with http Requests/Responses, serializing and deserializing and delegating tasks to lower tiers;
- DataManagement (and the affiliated interface): meant to deal with most of the logic of for this solution, by handling cohesive tasks;
- DataAccess: meant to deal with the communication to the database;
- Model: meant to hold all the objects that the system will work with;
- Decorators: meant to decorate actions/classes. One good example being the Logging decorator, which logs all of the actions that a user does, helping in the debugging process, once the solution is deployed;
- Tests: meant to hold all of the test classes we created;

Figure 7 represents version 0.2 of our architecture. The difference from the previous being the new Project called "External API", which got added to our architecture, as we thought of several possible threats and problems that come with the creation of a software solution that uses the internet as part of its base logic.





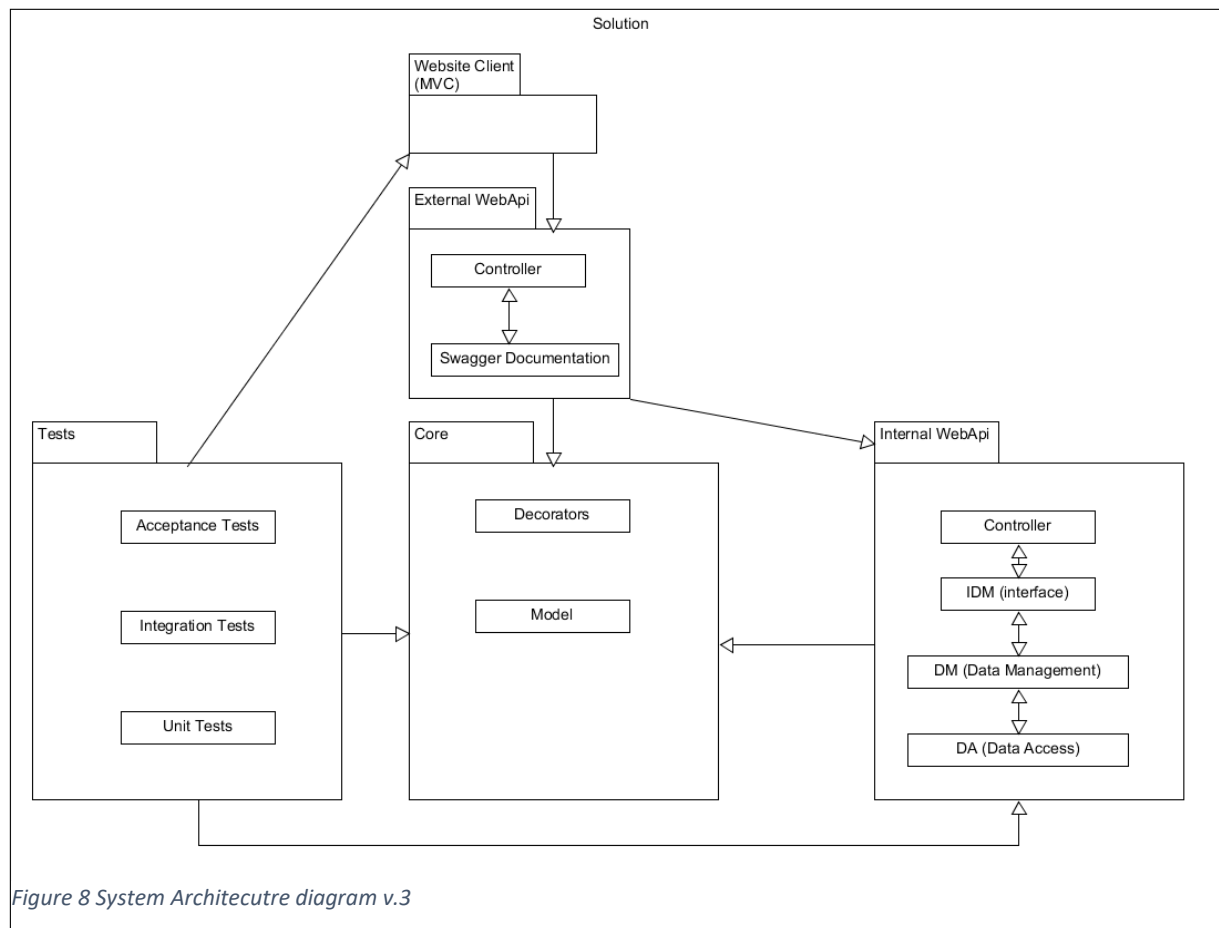
External API, was our response to the problem of future development, for other programming teams. This new project, would aid with development, by adding all the required Documentation, that any Web API should have.

For the documentation, we decided to use Swagger, as it uses the scaffolded XML files, and makes a website that nicely displays our Web API's endpoints, what their parameters and their possible response messages are.

Another function that was attributed to the External WebApi, was security, by acting as a Proxy, between the clients and the actual server. Providing us just another layer of anonymity, from possible directed attacks.

It also allowed Means to switch internet providers or hosting services providers, without having to notify the clients of the changes, as they would always be connected to the proxy, which redirects their calls to the server, instead of directly being connected to the Internal WebApi.

We soon realized that we could reduce the code duplication, improve building times and overall performance, while also increasing our code's cohesion, by simply moving around some of the already existing parts, into dedicated projects. This presented our 3<sup>rd</sup> and "final" architecture, (represented in Figure 8) or so we thought.



We decided to move Models and Decorators into a special Project called “Core” as it would hold information shared by all the other projects. This would allow us to simply change a Core class, without having to rebuild the other projects, as long as it did not affect their functionality.

Tests also got a dedicated project, because we don’t want to rebuild all the tests, every time we made a small change in the rest of our solution. Not only that, but tests became a bit more complex, with the addition of Unit Tests (which would test individual methods), Integration Tests (which would test the Internal Api’s complex functionality), and Acceptance Tests (which would simulate a user’s interaction with the API, through the MVC client, essentially testing the main and most common use cases).

After a good while, and well into the development process, we got new requirements and information from Means, in which they said, among others, that our solution, will not be hosted on the internet, but rather on a local computer/server and will be accessible only from LAN.

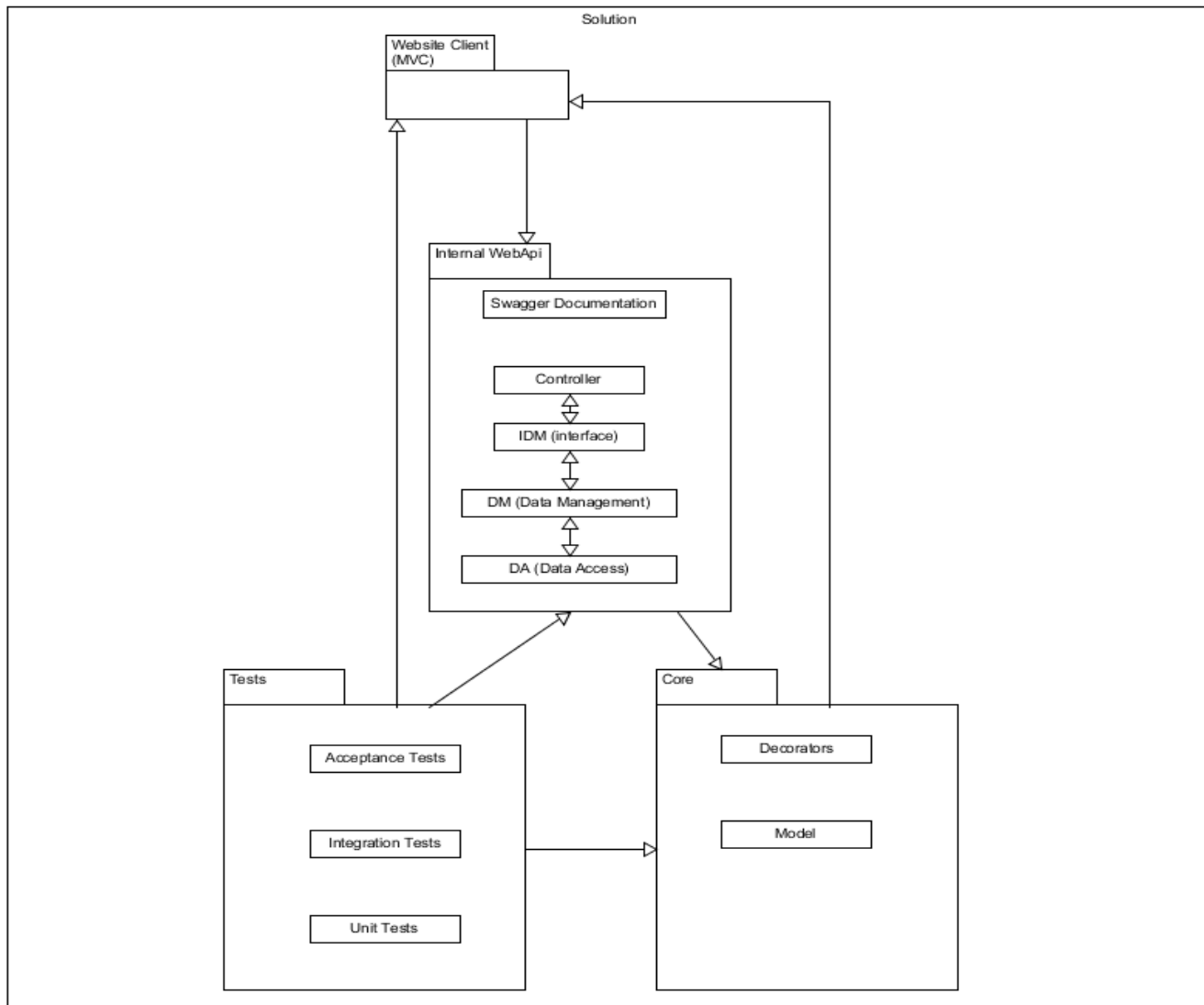


Figure 9 System Architecture diagram v.4

This new information determined us to remove the External API, as it would no longer bring the benefits of having a proxy in the client-server relationship, while bringing all the downsides of having one. Downsides such as: slightly slower responses, and harder to develop new/edit current endpoints.

But we decided to keep the Swagger documentation for possible future dev teams that might work on this solution. And we decided to move it to the Internal Api project.

## II. Domain Model

Before starting work on the product, we wanted to know the functional requirements, the application was to achieve, so that we could make the corresponding model layer and database. After gaining all the necessary information from the company, we created an early version of the domain model, and only after few iterations have we come to the final solution shown in the Figure 10.

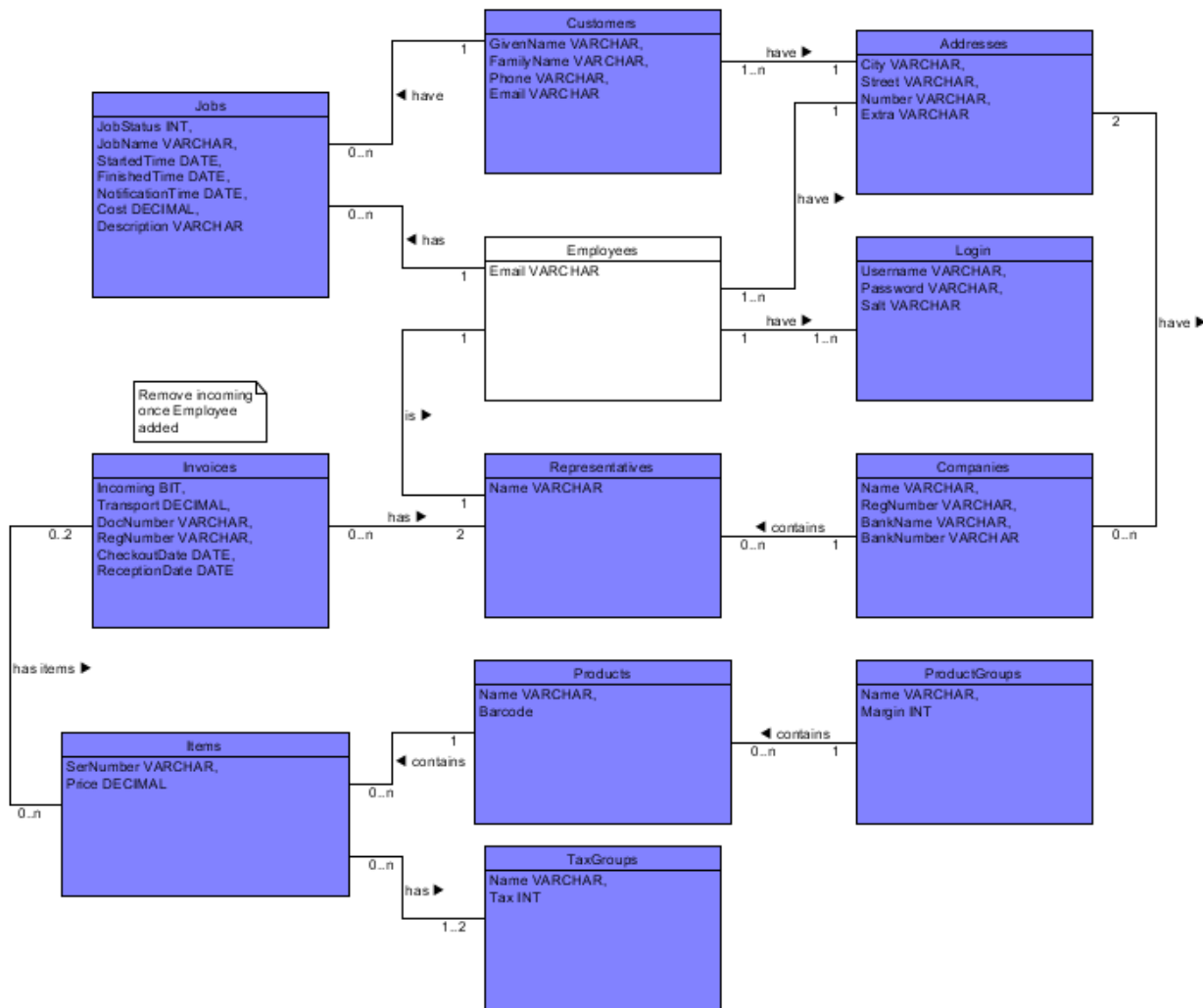


Figure 10 Domain model

On the first look of the resulting domain model, you can notice, that almost all elements are colored blue, except for one. The elements with blue background indicate that they have been created (and are successfully used throughout the project), but elements in white background show what is yet to be done (indicating, that we over estimated how much we can manage to finish).

Explanation of each element:

Address - Holds all the location information.

Customer - Customers of interest and all their corresponding information.

Job - Holds all known information of job (task), which the customer has requested.

Employees - Holds information of the products handler.

Login - Holds login information necessary to access the system.

Companies - Holds information of all the known companies that Means conducts business with.

Representatives - Information, regarding companies' representatives (who specifically represented the company on the occasion).

Invoices - information, regarding the deals between two companies.

Items - Info for unique item (Example: One specific instance of iPhone X)

Products - Information of closely related items (Example: specific iPhone X model)

ProductGroups - Holds details shared by many Products (Examples: iPhone X, Galaxy S9 and Huawei Mate 20, all are phones thus may share information)

TaxGroups - Holds tax specific details shared by many items (Example: iPhone X and the Galaxy S9 come from Germany, where the tax is 19%, so they are in the same tax group, but the Huawei Mate 20 comes from Latvia, where tax is 21%)

### III. Database Model

Our database diagram, just like the domain model early on changed quite a lot, but we managed to get the current solution also early on. As seen in Figure 11, we didn't manage to finish all the needed work, that we intended, having employee tables not being implemented, due to unexpected issues and us wanting to focus more on the quality of the product, rather than quantity.

In Figure 11 you can also see, there is a table named "Logs", where we are saving all the information regarding the requests, which were made to the InternalApi project. This was done, to facilitate easier understanding of actions performed and their results. Not to mention easier support which Means will get.

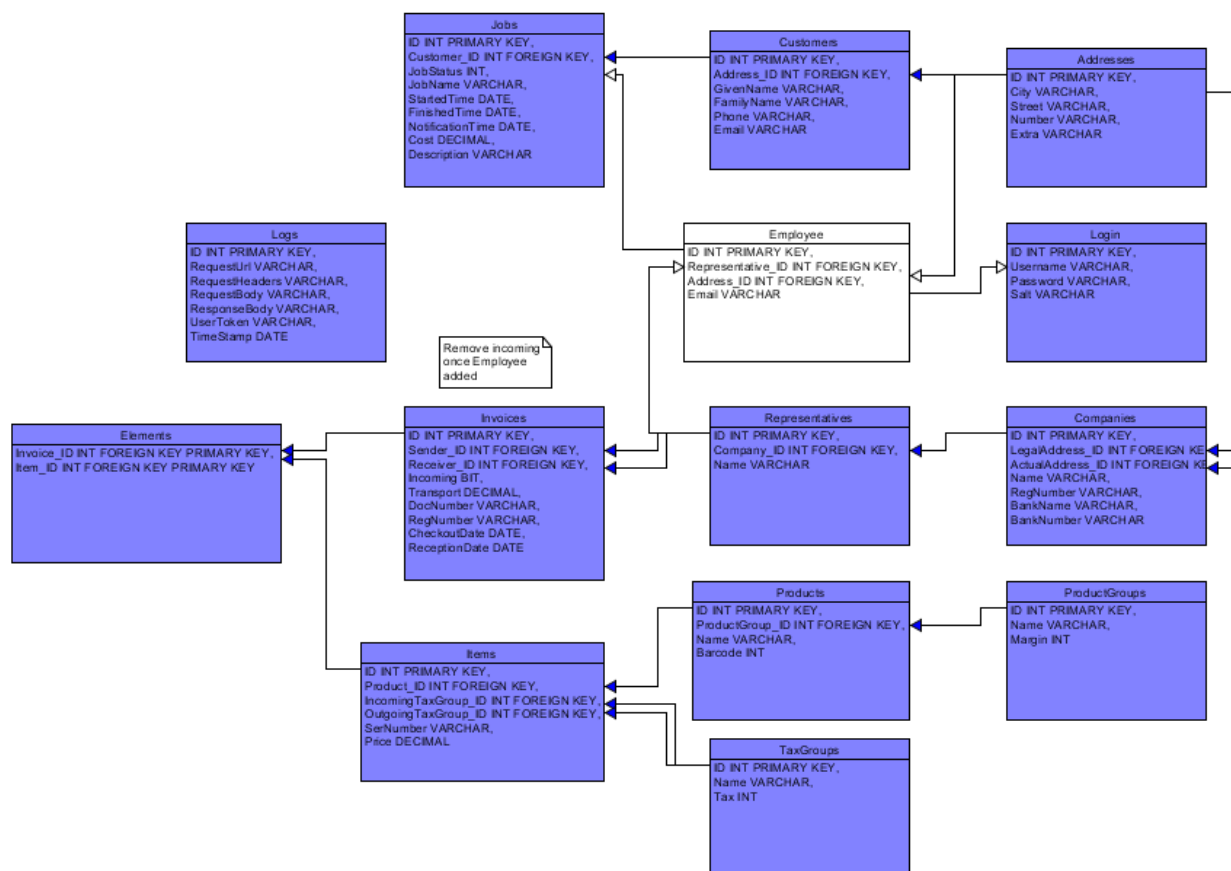


Figure 11 Database diagram

---

## 4. Technology and Programming

### a) Analysis

#### I. Framework

##### A. *Technologies Considered*

Deciding the best technology, to use for the server is a hard and a time-consuming process, so we decided to do more research on this subject and came to a few possible solutions. The options considered will be explained and our final choice will be supported by arguments in the following text.

Here are some definitions worth knowing, before diving deeper into the rest of this subject:

- **Application program interface (API)** is a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a graphical user interface (GUI) by providing all of the necessary building blocks.
- **Web service** is a generic term for an interoperable machine-to-machine software function that is hosted at a network addressable location (All Web Services are APIs but not all APIs are Web Services).
- **Microservices** is an architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable and organized around business capabilities.

When we decided on a framework to follow when creating our API we came to the conclusion, that there are only two frameworks worth considering for the job. Both of the considered technologies are APIs in the .NET Framework meaning, that they are easy to use in Visual Studio and with other .NET technologies. Besides that as it just so happens that some of us have had previous work experience in both of them, experience gained during the internship period:

##### B. *ASP.NET Web API*

Is a framework that makes it easy to build HTTP services that reach a broad range of clients. (Resource oriented)

##### *Pros*

- Easy to make
- Speed (Data is kept to the minimum)
- Many content formats (XML, JSON, CSV and more)
- Supports a wide variety of clients (mobiles, browsers, PC and many more)
- Uses the full feature of HTTP (like URIs, request/response headers, caching, versioning, various content formats)
- Open-source
- Allows for caching, compression, versioning

##### *Cons*

- Only 1 transport protocol (HTTP)
- No support for higher level protocols such as Reliable Messaging or Transactions

---

### C. *WCF (Windows Communication Foundation)*

Is a set of APIs in the .NET Framework for building connected, service-oriented applications.

#### Pros

- Many transport protocols can be used (TCP, UDP, Named pipes and many more).
- Multiple encoding types (Text, MTOM, and Binary).
- Support for higher level protocols (Reliable Messaging or Transactions).
- Improved security.
- Allows also for One way and duplex communication, not limited to request reply.

#### Cons

- A lot of time is spent in configuring WCF services.

After reviewing the Web API and WCF we came to conclusion, that we wanted to have a service which would be usable through multiple devices, that's why we came to conclusion that we should use Web API instead of WCF. Although WCF could be a better solution for a backend project, as it would be easier for us to maintain, test and change the transport protocol (Both projects would be located on the same computer or at a minimum located on computers connected though LAN, where TCP protocol would be a better choice). We determined, that due to time constraints Web API became our main choice. Besides the aforementioned points, during our research, we've encountered many developers warning about WCF and that the configuration could take long time even for professionals.

## II. Database

### A. *Choices in Databases*

When choosing the right database for the project, we had to make multiple choices and one of the biggest ones was choosing whether we should pick relational or non-relational database. We determined that it would be useful, to have a list of pros and cons so we could make an informed decision and not base this on just gut feelings, without seriously considering both options.

There are two types of database technologies: relational databases, which are great at organizing and retrieving structured data; and non-relational databases, which are best used when the data is inconsistent, incomplete or simply massive (like in Big Data projects).

### B. *Relational Databases*

#### Pros

- Relational databases work with structured data.
- They support ACID transactional consistency and support "join" functions.
- They come with built-in data integrity and a large eco-system.
- Relationships in this system have constraints.
- There is limitless indexing. Strong SQL.

---

#### Cons

- Relational databases do not scale out horizontally very well, only vertically.
- Data is normalized, meaning lots of join functions, which greatly affect speed.
- They have problems working with semi-structured data.

#### C. *Non-relational/NoSQL*

##### Pros

- They scale out horizontally and work with unstructured and semi-structured data.
- Schema-free or Schema-on-read options.
- High availability.
- Many NoSQL databases are open source, so they are free of charge.

##### Cons

- Weaker or eventual consistency (BASE) instead of ACID.
- Limited support for join functions.
- Data is denormalized, requiring mass updates.
- Does not have built-in data integrity.
- Limited indexing.
- Requires considerable training, to be used properly.

After reviewing different types of the databases, we determined, that the relational database will be chosen, this was decided because of following factors:

1. We have more experience working with relational databases (less time is wasted getting to know new technology).
2. Support for ACID.
3. Limitless indexing.
4. Relationships have constraints.
5. Data is structured and normalized.
6. Product is made for small number of users and not expected to have more than 4 concurrent users.

#### D. *Database Engines*

As for engines, there are several choices that we considered, for a relational database, some of which are: Oracle Database, SQL Server and MySQL; and since all three of them were using dialects of the same language (SQL), it went down to the very basics when we took the decision on which to use.



---

As a final decision, we chose SQL Server 2017, because of the following: SQL Server executes and commits each instruction, unlike Oracle which requires explicit commands to commit the changes; its included in visual studio 2017; ease of use, since not only were we thought on how to use it, but also compared to Oracle, which gives so many other settings and configurations that can be set to the wrong value; and of course performance.

### III. Client

For the client we reviewed multiple options, in the beginning we thought of creating a dedicated client, but after receiving feedback from the company we got to know that some of their requirements for the client were:

1. Application should work on multiple different platforms (pc, phone...)
2. Should not be needed to install
3. Should be easily changed and updated for future expansions

This forced use to make a web-based client, which would not only communicate with the API, but would also be interactive, understandable and would allow us to make use of various languages.

Since we were new to web development, having only little experience from the 3<sup>rd</sup> semester exam project where we developed a dedicated client and a web client with limited functionality, we decided to review this subject and decide on what technology is best to use. We limited our choice to ASP.NET solutions, since we had decided to work in C#.

Some of our options which we reviewed were:

1. MVC
2. Web forms
3. Web pages

After reviewing these choices, we determined that web forms aren't for us, although they are easy to create, we wanted to learn something more complex than just dragging and dropping, besides web forms are being depreciated and other options are more relevant for what our project must do. After comparing the options which were left with, we determined that MVC is the best choice, as it is frequently being used in actual businesses, its Ideal for developing complex but lightweight applications and it was the technology which we had the most knowledge in, so less time would be spent learning a new way of doing things, instead focusing on developing a solution.

To improve the created client, we decided to use various technologies such as:

1. Ajax
2. JQuery
3. Bootstrap
4. ASPX
5. Session
6. Cookies

And languages associated to them:

1. JavaScript
2. CSS
3. Html
4. Razor

### Type of client:

When deciding on the type of client, we would be making, we first had to review the requirements given to us by the company. After reading the request and thinking of possible solutions we came to two possible types of clients we could make, to achieve the goals:

**Standalone program** - Single application which would be given to the client and allow them to do manage the database (depicted in Figure 12).

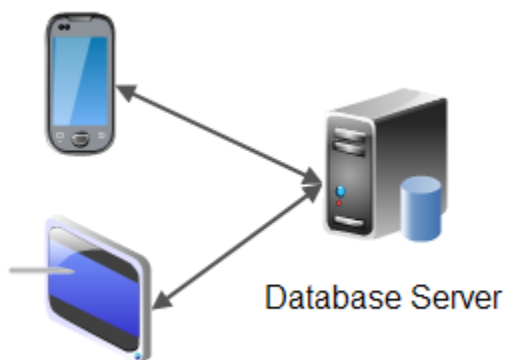


Figure 12

#### Pros:

- Easy to make
- Security (All of the work is done within the customer's computer)
- Great performance is achievable

#### Cons:

- Requires customer to install an application
- If there is a new update, client would be required to get the newest version (without updating the client wouldn't be able to experience newest features or the users applications might not work)
- Client system and hardware dependent

**Client-Server application** - Simple client which deals with interacting with the customer and communicates to a server or multiple servers, that make changes to database (depicted in Figure 13).

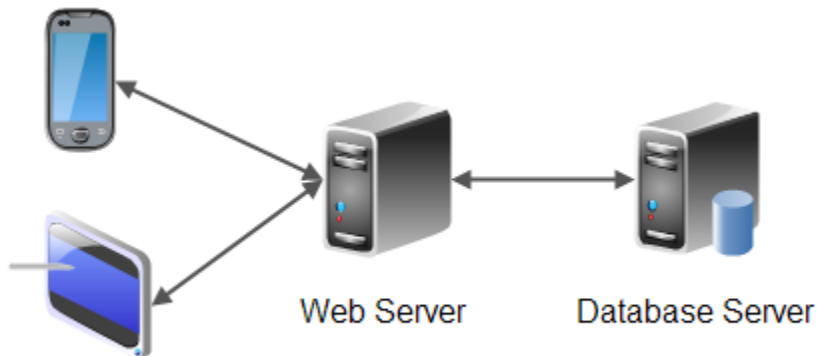


Figure 13

**Pros:**

- If there is update to the product all the functionality is automatically accessible to the user
- Easy to expand to more clients
- Not dependent on the user's hardware capabilities

**Cons:**

- Harder to develop than the dedicated client
- Worse performance due to extra steps (communicating with server slows down response)

After reviewing the pros and the cons, we determined that the Client-Server application better suits our needs, as it would be easy to work with (for the user), has possibility to be easily expanded to new types of clients (such as mobile) and functionality updates automatically effect all clients not requiring them to get the newest release of the application.

Meanwhile the cons of having a distributed application are rather minor, for example sacrificing a bit more time in development of the application for easier expansion in future and minor performance decreases due to the fact that the business logic being located on dedicated server.

#### IV. Middleware

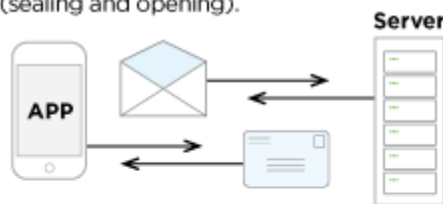
##### A. Clients-System

Accessing the web services can be done through two different ways:

### SOAP vs. REST APIs

SOAP is like using an envelope

Extra overhead, more bandwidth required, more work on both ends (sealing and opening).



REST is like a postcard

Lightweight, can be cached, easier to update.

REST (Representational State Transfer) - Is a relative newcomer and as an architecture style which does not require processing, is lightweight and is naturally more flexible than SOAP, requesting the user to just use URLs and can get a response in multiple different formats, while SOAP is restricted to only using XML.

SOAP (Simple Object Access Protocol) - Old but still relevant solutions to the data exchange problem, SOAP uses a more rigid set of messaging patterns than REST. SOAP relies exclusively on XML to provide messaging services, the XML can become extremely complex and in some programming languages, programmers need to build those requests manually, which becomes problematic because SOAP is intolerant of errors, but when working with .NET languages, developers don't have to even see the XML message.

Figure 14

##### B. System-Database

#### Technologies for data access:

When deciding on which data access technology to use, we came to the 3 most popular technologies on the internet right now, about to application development:

1. ADO.NET
2. LINQ to SQL
3. ADO.NET Entity Framework

After we obtained knowledge of the most popular technologies (we chose to follow in the footsteps of most popular products, as they most likely have the best solution and also, they have great support for them), we started comparing them.

ADO.NET was one of our initial vision in creating the application, although its easier to use in difficult scenarios, we determined, that its extensions - LINQ to SQL and Entity Framework are easier to use in casual scenarios and is faster to develop for and easier to maintain than ADO.NET. After reviewing and comparing the last two options we had, we came to the conclusion, that Entity Framework is easier to maintain and more powerful than LINQ to SQL, also as of the release of .NET 4.0, LINQ to SQL is often considered by many to be an obsolete framework.

When starting work in Entity Framework, we must decide which of the following methods to use for our project:

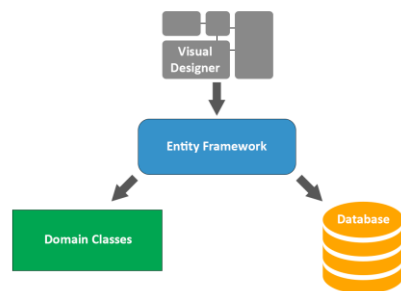


Figure 15(Visual diagram used by the Framework to autogenerate the Database SQL script and the Data Model source code files)

### Model-First

#### Pros:

1. Great when there is need to have a visualization of the database
2. Easy to use and understand when dealing with large data structures
3. Models can be updated accordingly, without data loss

#### Cons:

1. Autogenerated SQL scripts can lead to data loss in case of updates
2. Hard to have precise control over generated model classes

### Database-First

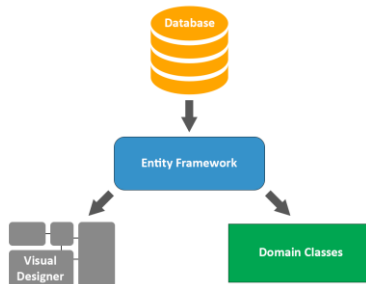


Figure 16 SQL script used to create the Database and from that corresponding Data Model and diagram is generated

#### Pros:

1. Easy to use existing database for creating models
2. Change will be always performed on the database, so no data loss can occur

#### Cons:

1. Can be hard to update database when dealing with multiple instances
2. Less control over generated model classes, even more so than the Model-First approach

### Code-First

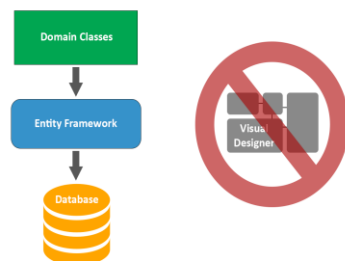


Figure 17 Model classes from project used in Entity Framework generate the database accordingly to it

#### Pros:

- 1.No need for diagrams
- 2.Easy to maintain and develop code
- 3.Ideal for small-to-medium sized projects
- 4.Saves development time
- 5.Good control over generated database

#### Cons:

- 1.Requires knowledge of object-relational mapping
- 2.Maintaining the database can be tricky without suffering data loss (Mostly overcome by using Migrations, added in EF 4.3)

---

After reviewing options, we decided to go with the Code-First approach as we did not want to waste too much of our time on making the database work, since we don't have a lot of time to begin with. Besides the already mentioned reasons, this solution by many is considered the best for small-to-medium sized projects, we wanted to gain more experience, have an easily maintainable code and there was no need for diagrams. In the end the actual cons of Code-First were rendered useless, since they don't affect us at all, such as data loss to give an example.

## b) Implementation

### I. Security

To increase the security of our final application we decided to follow the already well established standards and practices which would greatly reduce the risk of sensitive information being accessible by an unauthorized person.

#### A. Access to resources

When talking about resource access it is important to know what the difference between authorization and authentication is:

- **Authentication** is the process of verifying who you are. When you log in with a user name and password you are getting authenticated.
- **Authorization** is the process of granting access to a desired resource. Gaining access to a specific or desired resource because the validated values were approved, and the user was granted access.

If a user wants to use our application, we decided to restrict their initial access and first must be authenticated in order to enter deeper into the application. To authenticate a user must fill in their login details (in our case being the correct username and password combination).

Instead of saving the password as they were entered and expose our customers sensitive details, we decided to take necessary steps and save them encrypted. We decided to use SHA256 to encrypt passwords, but only after adding salt (a unique random string) to the password, which would ensure, that even if two same passwords are entered, they would still have different hash values, making them harder to crack or confuse. Doing this also ensures, that if someone manages to get values in the database, they will not be able to make much use of it, as they would still have to brute force each of the passwords open, which could potentially take a very long time, allowing us or Means to react if noticed.


				
Password	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz
Salt	-	-	et52ed	ye5sf8
Hash	f4c31aa	f4c31aa	1vn49sa	z32i6t0

Figure 18 Data encrypted using SHA vs data encrypted using SHA + Salt

Authorization was ensured on the server side by the usage of tokens (making this process more secure), this was done by checking if user is logged in, only if they were, would they be able to access the functionality of the product, otherwise he would not be able to access any other page other than the login page. In the future we plan on implementing different access levels for different users, but this was not done right now, as we had a limited time.

#### B. Resource transport

Http (Hypertext transfer protocol) is commonly used when two machines are communicating over the internet, but it's not secure and if the information would be received by a malicious person who should not have it, he would likely be able to easily read and use it for his own benefits. HTTP can easily be exploited by using a in man-in-the-middle attack, where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other.

To improve the security of the messages sent over internet, we decided to encourage the company to use HTTPS, for all the pages (having it only for login is not secure enough), which is an extension of the HTTP, but with secure communication integrated. In HTTPS,

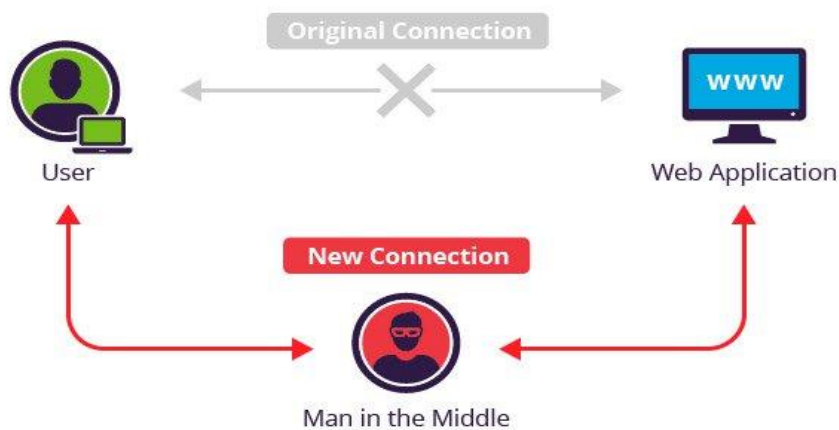


Figure 19 Representation of a man in middle attack

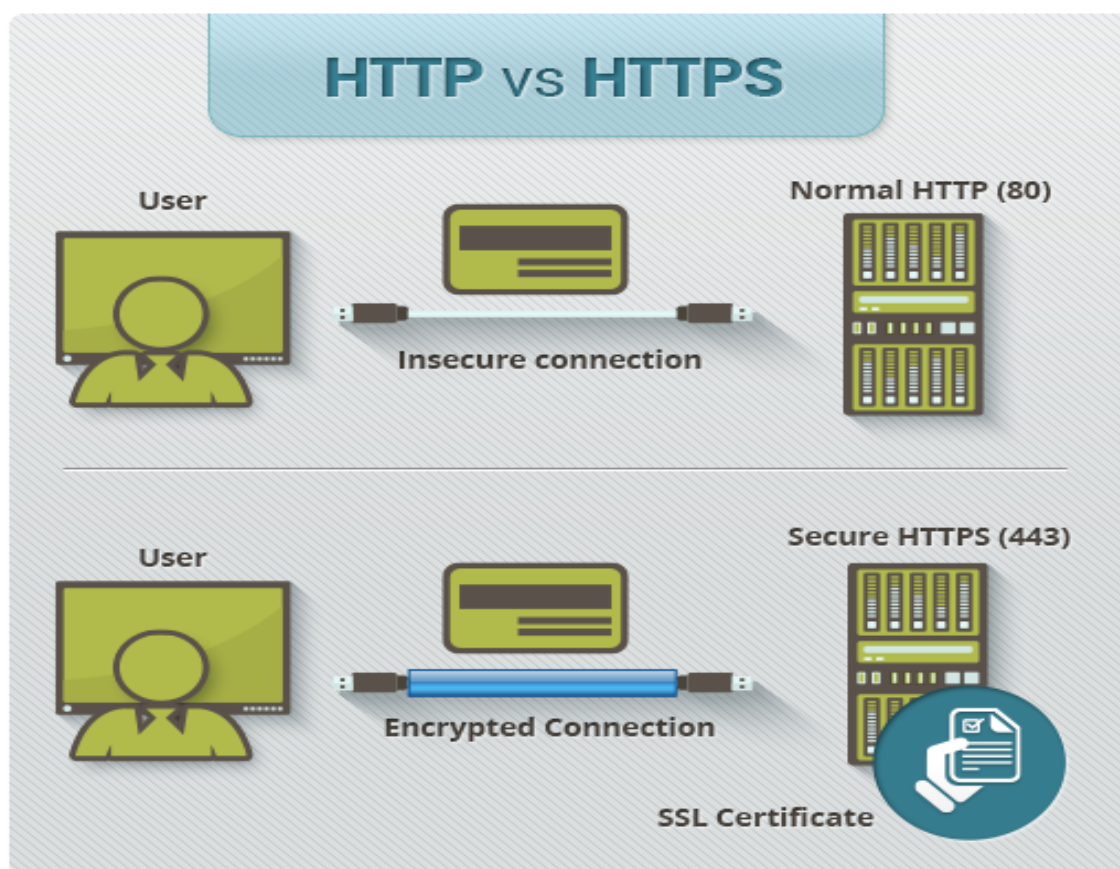


Figure 20 Http vs HTTPS

the communication protocol is encrypted using Transport Layer Security (TLS) or Secure Sockets Layer (SSL). The protocol is therefore also often referred to as HTTP over TLS or HTTP over SSL.

As said before there are 2 common transport layer security protocols for communication over the internet, SSL and TLS.

SSL although being one of the most popular ones, is now considered to be obsolete, due to it having multiple vulnerabilities (e.g. POODLE, DROWN), but it provides better performance speed wise than TLS and it is easier to implement for testing purposes, as IIS Express (Inbuilt in Visual Studio) comes with possibility of easily implementing it. This is the reason why we decided to test the product using SSL, but when deploying we would advise the company to use the more secure TLS protocol. Although it is unlikely that an attack could take place, as the company is relatively small and the attacker would have to be on site to carry out the attack (as application would be hosted only on LAN), it is still possible, that there is a chance for it happening and making it as hard as possible to provide him with the opportunity to do what he wants could discourage him from carrying out the attack.



### C. HTTP security headers

Security headers instruct your browser exactly how to behave when it handles your website's content and data. By setting up your security headers correctly, not only you help protect your site, but your users as well. This will also help you cut down on security flaws and working hours invested in tracking and fixing them.

List of the most popular headers:

- **HTTP Strict Transport Security (HSTS)** - prevents any communication happening over HTTP.
- **Cross Site Scripting Protection (X-XSS)** - stop loading the page when a cross-site scripting attack is detected.
- **Clickjacking Prevention (X-Frame-Options)** - so hackers don't "iframe" your site, to trick other users into clicking links which you never intended to (especially bad, since cookies, cache and all saved data by the browser, is loaded and used, when framing a web page).
- **Browser Sniffing Protection (X-Content-Type-Options)** - makes it harder for hackers to guess the right mime type, by inspecting the content.
- **Referring Settings (Referrer-Policy)** - prevents linked website, to be able to see where the users are coming from.
- **Content Security Policy (CSP)**- helps to prevent code injection attacks like cross-site scripting and clickjacking, by telling the browser which dynamic resources can load.
- **HTTP Public Key Pinning (HPKP)**- tells a web client to associate a specific cryptographic public key with a certain web server to decrease the risk of [MITM](#) attacks with forged certificates.

In the following text we are going to explain, why we picked the select headers, also do more in depth investigation for one header, to explain how it works and why it was made like that. In Figure 21, you can see our implementation of the headers and their assigned values and by observing the Figure 22 reader can also see Response headers, as seen by user's browser.

```
<httpProtocol>
  <customHeaders>
    <add name="Strict-Transport-Security" value="max-age=31536000; includeSubDomains; preload"/>
    <add name="X-Xss-Protection" value="1; mode=block" />
    <add name="X-Frame-Options" value="DENY" />
    <add name="X-Content-Type-Options" value="nosniff" />
    <add name="Referrer-Policy" value="no-referrer" />
  </customHeaders>
</httpProtocol>
```

Figure 19 Implementation of security headers in web.config

▼ Response Headers [view source](#)

Cache-Control: private  
Content-Length: 7857  
Content-Type: text/html; charset=utf-8  
Date: Thu, 10 Jan 2019 11:37:21 GMT  
Referrer-Policy: no-referrer  
Server: Microsoft-IIS/10.0  
Strict-Transport-Security: max-age=31536000  
X-AspNet-Version: 4.0.30319  
X-AspNetMvc-Version: 5.2  
X-Content-Type-Options: nosniff  
X-Frame-Options: DENY  
X-Powered-By: ASP.NET  
X-SourceFiles: =?UTF-8?B?QzpcVXNlcncYnV:  
X-Xss-Protection: 1; mode=block

Figure 20 (Response headers, as seen by user's browser)

The reason we added Cross Site Scripting Protection, is that we wouldn't want scripts, which doesn't belong to our application, to be run, as they could compromise security and ruin the end user experience.

Browser Sniffing Protection was added, although this attack is unlikely, it can happen and adding this doesn't impact ordinary user.

Reason we added Strict Transport Security is simple, as we didn't want user communicating with server in unsecure manner.

Referring Settings was added since the users' privacy is an important matter nowadays and not disabling it could pose problems, as attacker could extract sensitive information in your URLs, which we don't want to forward to other domains.

Adding of Clickjacking Prevention was a straight forward decision, since we wouldn't want users of the product clicking unsafe links, exposing themselves and our application to possible attack.

Content Security Policy wasn't implemented, since although it is useful, it is hard to configure properly and could cause multiple issues, which would mean reduced development time and reduce final quality.

The header we decided also to not implement was Public Key Pinning, as there was no such need for it in application run on LAN.

Now we are going to dive deeper in one of the implemented headers, to explore what was done and why, we decided to review most complex one, that being Strict Transport Security.

How it was implemented:

**<add name="Strict-Transport-Security" value="max-age=31536000; includeSubDomains; preload"/>**

To understand it better let's dissect it into smaller parts:

1. **max-age:** Indicates how long this header should be applied for on the site, so users will automatically call the site by using HTTPS instead of standard HTTP but setting it for too long could also create issues. If for example there are changes and you expect to communicate over HTTP not HTTPS. We decided to set to value of 31536000, which is one year in length, as we don't plan on using anything else than HTTPS for all pages.
2. **includeSubDomains:** makes all present and future subdomains accessible only through HTTPS.

3. **Preload:** removes the opportunity for the attacker to intercept and tamper with redirects when user first communicates with site, as they take place over HTTP.

#### D. Tokens

Part of the security measure we've taken against malicious use of our system, is a token-based authentication system.

A token looks like the following string:

"162b34975bf333604040d150c7e536db690ebe7fb6b26b837da4dc9209905b2854" and enciphered within it, lay 3 important pieces of information: the login ID, which can be found in the database, salt, and the associated Username. As the tokens are enciphered using a Caesar cipher with a rotation equivalent to the user ID, then encrypted using SHA-256 together with the unique salt, the likelihood of an outside malicious user, to forge the token is quite small, in fact, the only way for someone to do it, is if they already have access to information within the database, making our system relatively safe.

The way tokens are generated and distributed to legitimate clients, is via the use of a "Token Generator", that is called in the "Login" endpoint (see fig 23), only after a successful authentication is

```
[Route("Login")]
[HttpPost]
0 references | RaidenRabit, 2 hours ago | 3 authors, 6 changes | 2 work items | 0 requests | 0 exceptions
public async Task<HttpResponseMessage> Login()
{
    var requestContent = await Request.Content.ReadAsStringAsync();
    Login login = JsonConvert.DeserializeObject<Login>(requestContent);
    int loginID = _loginManagement.Login(login.Username, login.Password);
    string userToken = loginID + UtilityMethods.ComputeSha256Hash(UtilityMethods.Encipher(login.Username, loginID));
    if (loginID != 0)
    {
        return Request.CreateResponse(HttpStatusCode.OK, userToken);
    }
    else
    {
        return Request.CreateResponse(HttpStatusCode.BadRequest, "Incorrect login credentials");
    }
}
```

Figure 21 (Token generator usage)

completed, using a legitimate username + password combination. The result of which, is a Response message containing the token, that must be sent as a header, with every request to the API.

## II. Fake Server

As this project is based on Web API technology, running tests proved to be a rather long and tedious process, as it constantly required us to re-run the Web API server, every time a new test/ new code implementation would be carried out. And in order to better automate this entire process, we decided to implement a fake server, that would automatically host the Internal Api project in IIS Express, every time a test would be scheduled to run. This solution proved to be extra helpful as it allowed for our continuous integration solution (called Appveyor), to automatically run the tests as well.

The way we implemented this entire automation, is slightly different depending on the type of tests, we were running.

For Integration tests, we have decided to create a base class for all the tests, that would instantiate an InternalApiFakeServer class, which in a relatively short method (see figure 24), would start up a server and create a HttpClient, which the tests, would later use to call the API. At the end of each test, the base class, in its Tear Down function, would dispose of the server, client and all the other objects which were used, only to start everything again from scratch, for every single test that is scheduled to run. This was done in order to ensure each test gets a fresh start and to make sure that they were independent from one another.

```
1 reference | Ralf, 2 days ago | 1 author, 1 change | 1 work item | 0 exceptions
public void StartServer()
{
    var httpConfiguration = new HttpConfiguration();
    config.Filters.Add(new Core.Decorators.ExceptionFilter());
    config.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling
    = ReferenceLoopHandling.Ignore;
    config.Formatters.JsonFormatter.SerializerSettings.NullValueHandling
    = NullValueHandling.Ignore;
    InternalApi.WebApiConfig.Register(config);

    _internalServer = new HttpServer(config);
    _internalClient = new HttpClient(_internalServer);
    _internalClient.BaseAddress = new Uri("http://localhost:64007/");
    _internalClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
}
```

Figure 22 (Internal Api fake server implementation)

For the Acceptance tests, we, at first tried to have the same architecture, however, due to inbuilt limitations of the Selenium web drivers, we had to take a slightly different approach, in terms of implementation. Concept wise, this automation is exactly the same as the one for the integration tests

```
1 reference | RaidenRabit, 14 days ago | 1 author, 5 changes | 1 work item | 0 exceptions
public void StartServer()
{
    var webClientApplicationPath = GetApplicationPath("WebClient");
    var internalApiApplicationPath = GetApplicationPath("InternalApi");

    KillAllIIS();
    _iisProcess = IISProcess(49873, webClientApplicationPath);
    _iisProcess2 = IISProcess(64007, internalApiApplicationPath);
}
```

Figure 23 (Acceptance test Fake Server start-up method)

```
2 references | RaidenRabit, 14 days ago | 1 author, 2 changes | 1 work item | 0 exceptions
private Process IISProcess(int iisPort, string path)
{
    var programFiles = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles);
    var iisProcess = new Process();
    iisProcess.StartInfo.FileName = programFiles + @"\IIS Express\iisexpress.exe";
    iisProcess.StartInfo.Arguments = string.Format("/path:{0} /port:{1}", path, iisPort);
    iisProcess.StartInfo.CreateNoWindow = true;
    iisProcess.StartInfo.RedirectStandardOutput = true;
    iisProcess.StartInfo.UseShellExecute = false;
    iisProcess.Start();
    return iisProcess;
}
```

Figure 24 (implementation of IIS process start-up)

(see figure 25), an Acceptance Test Base class, among other things, instantiates a Web Client Fake Server class, which hosts both the Internal Api and the Web Client, in two different IIS Express processes (see figure 26), processes set-up with a specific port and specific project path, that is obtained from a brief machine search, making the automation true and requiring no manual changes from the user (us). The IIS Express processes are saved, in order to be stopped, once the Acceptance Test is finished. And the entire process starts over again, once a new Acceptance Test is scheduled to run, again to make sure that they are independent from one another.

But before any of this starts, a KillAllIIS command is issued (see figure 27), in order to make sure that no remnant IIS Express process, were accidentally left running (ex: when manually stopping a debugging session of an acceptance test).

```
1 reference | RaidenRabbit, 14 days ago | 1 author, 1 change | 1 work item | 0 exceptions
private void KillAllIIS()
{
    ...Process[] ps = Process.GetProcessesByName("iisexpress");

    ...foreach (Process p in ps)
    ...{
    ...    try
    ...    {
    ...        ...if (!p.HasExited)
    ...        {
    ...            ...p.Kill();
    ...        }
    ...    }
    ...    catch (Exception ex)
    ...    {
    ...        ...Console.WriteLine(String.Format("Unable to kill process {0}, exception: {1}", p.ToString(), ex.ToString()));
    ...    }
    ...}
}
```

Figure 25

### III. Decorators

Global decorators and Filters are an important part of our Internal Api project, as they provide vital functionality, while keeping the code clean, and helping keep up the high cohesion, low coupling, and no code duplication standards, that we established to follow early on.

There are three main parts where we have employed the help of Global decorators and Filters, in our project:

- Authentication and Authorization
- Exception handling
- Action and error Logging

The filters are added to the Internal Api, through a couple of simple code lines, in the configuration file (see figure 28)

```
0 references | RaidenRabit, 6 hours ago | 3 authors, 5 changes | 3 work items | 0 exceptions
protected void Application_Start()
{
    // ...
    GlobalConfiguration.Configure(WebApiConfig.Register);
    GlobalConfiguration.Configuration.Filters.Add(new Core.Decorators.ExceptionFilter());
    GlobalConfiguration.Configuration.Filters.Add(new Core.Decorators.AuthFilter());
    GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling
    = ReferenceLoopHandling.Ignore;
    GlobalConfiguration.Configuration.Formatters.JsonFormatter.SerializerSettings.NullValueHandling
    = NullValueHandling.Ignore;
    // ...
}
```

Figure 26

The exception filter (figure 29 for implementation), is a simple and small one, as its only functionality is to return an "Internal Server Error" response to the client, in case an unhandled exception occurs in the system, or in case an error that cannot be fixed by the user, by simply slightly changing their request parameters. As can be seen, when the "OnException" method is fired, an "internalServerError" response message, is attributed to the actionExecutedContext.

```
2 references | Ralf, 59 days ago | 1 author, 1 change
public class ExceptionFilter: ExceptionFilterAttribute
{
    // ...
    0 references | Ralf, 59 days ago | 1 author, 1 change | 0 exceptions
    public override void OnException(HttpActionExecutedContext actionExecutedContext)
    {
        // ...
        actionExecutedContext.Response = new HttpResponseMessage(HttpStatusCode.InternalServerError);
        // ...
    }
}
```

Figure 27

Similar to the Exception Filter, the Authentication and Authorization filter (figure 30), uses the actionContext, among other things, to do their jobs.

```
0 references | RaidenRabit, 1 hour ago | 1 author, 5 changes | 1 work item | 0 exceptions
public override void OnActionExecuting(HttpActionContext actionContext)
{
    try
    {
        base.OnActionExecuting(actionContext);
        string requestUrl = actionContext.Request.RequestUri.ToString();

        if (requestUrl.Contains("Login/CreateOrUpdate"))
            return;
        if (requestUrl.Contains("Login/Login"))
            return;

        var userToken = actionContext.Request.Headers.GetValues("UserToken").FirstOrDefault().ToString();
        if (!UtilityMethods.EvaluateToken(userToken))
            throw new Exception("Not logged in");

        string id = userToken.Remove(userToken.Length - 64);
        actionContext.Request.Headers.Remove("UserToken");
        actionContext.Request.Headers.Add("UserToken", id);
    }
    catch (Exception)
    {
        actionContext.Response = new HttpResponseMessage(HttpStatusCode.Forbidden);
    }
}
```

Figure 28

Unlike the Exception filter, the AuthFilter is driven by a “OnActionExecuting” event, that gets fired every time a call is made to the API. As a request reaches the API, the filter checks the request’s URL, in order to make sure that specific actions (such as Login) are not hindered by the lack of a UserToken.

After making sure that the Request is not intended for authenticating a user, the filter checks for the existence of a “UserToken” header, within the request. The lack of such a header, or if the header’s value is considered to not be legitimate, a “Forbidden” response message is assigned to the actionContext, preventing the client from accessing information to which they do not have access to.

The way a token is evaluated by the filter, is by querying the same information from the database and employing the use of a token generator, to obtain the correct token for this specific potential user, then comparing the 2 tokens.

The Logging part of this project is done through the help of a Delegating Handler, that, once a Request reaches the API, the handler would save the Request’s body, headers, URL, alongside with a times stamp generated on the spot, and the UserToken, if any is sent. All the information being saved in a specific table, in the database.

This information will later be used in order to determine which user did what, and for maintenance purposes, helping us to better understand what was done, when and how, in case any errors occur, during the daily use of this software solution.

As can be seen in Figure 31, the handler makes sure to not alter the request, but simply acts as an observer. We also decided to exclude all actions done via the Swagger documentation from being logged, to not overly pollute the database.

#### IV. Multi Language

One of the more interesting bits of code, in this project is the

Multi Language system, that we developed for the web client. This feature was one of the more important ones, as the users of this program are native speakers of Latvian and may find themselves lost, in the English version of the website.

```
0 references | RaidenRabbit, Less than 5 minutes ago | 1 author, 1 change | 1 work item | 0 exceptions
protected override async Task<HttpResponseMessage> SendAsync(
    HttpRequestMessage request, CancellationToken cancellationToken)
{
    Logs log = new Logs();
    // log request body
    log.RequestBody = await request.Content.ReadAsStringAsync();

    // log request headers
    log.RequestHeaders = request.Headers.ToString();

    // log request URL
    log.RequestUrl = request.RequestUri.ToString();

    // log UserToken
    try
    {
        log.UserToken = request.Headers.GetValues("UserToken").FirstOrDefault();
    }
    catch (Exception)
    {
        log.UserToken = "";
    }

    // let other handlers process the request
    var result = await base.SendAsync(request, cancellationToken);

    if (result.Content != null)
    {
        // once response body is ready, log it
        log.ResponseBody = await result.Content.ReadAsStringAsync();
    }

    if (!request.RequestUri.ToString().Contains("swagger"))
    {
        log.Timestamp = DateTime.Now;
        LogToDb(log);
    }

    return result;
}
```

Figure 29

Figure 32, shows how the interactive and intuitive UI elements, help the user to easily change the website's global language.

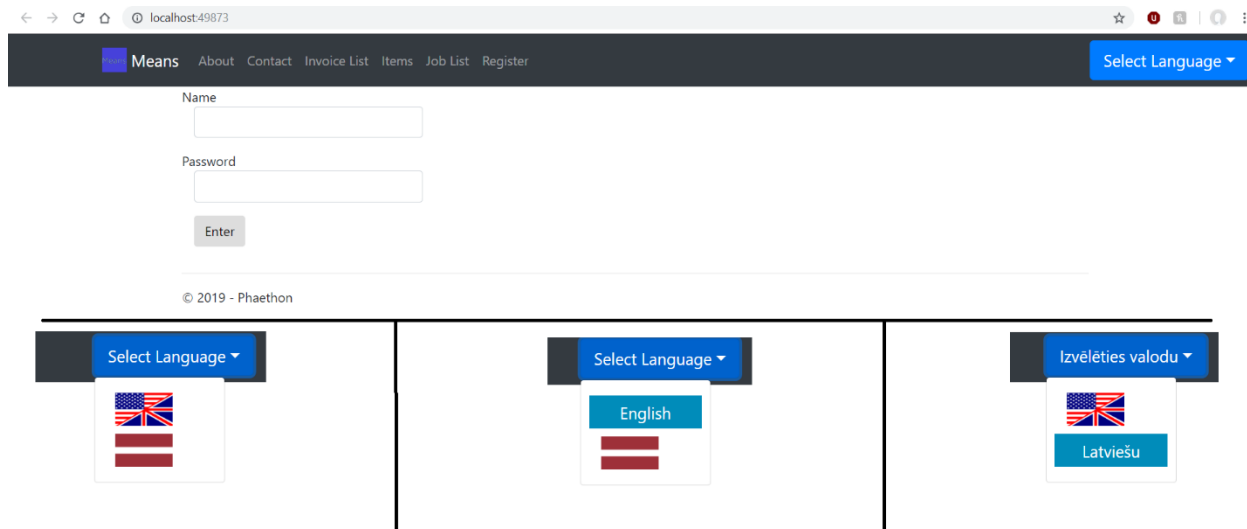


Figure 30(Changing language, UI)

This feature was implemented with the help of a user-sided element, that saves the language code, for the desired language which the user had selected.

Figure 33 shows the action behind the scenes, how every time the users selects a language, a new cookie is created and its assigned to hold the international abbreviation for that specific language. Later on, at every change, or page update, a global method checks the cookies (see figure 34) and depending

```
0 references | RaidenRabit, 31 days ago | 1 author, 2 changes | 1 work item | 0 requests | 0 exceptions
public void ChangeLanguage(string LanguageAbbreviation)
{
    if (!LanguageAbbreviation.IsNullOrEmpty())
    {
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(LanguageAbbreviation);
        Thread.CurrentThread.CurrentUICulture = new CultureInfo(LanguageAbbreviation);
    }

    HttpCookie cookie = new HttpCookie("Language");
    cookie.Value = LanguageAbbreviation;
    Response.Cookies.Add(cookie);
}
```

Figure 31

```
0 references | RaidenRabit, 31 days ago | 1 author, 1 change | 1 work item | 0 exceptions
protected void Application_BeginRequest(object sender, EventArgs e)
{
    HttpCookie cookie = HttpContext.Current.Request.Cookies["Language"];
    if (cookie != null && cookie.Value != null)
    {
        System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo(cookie.Value);
        System.Threading.Thread.CurrentThread.CurrentUICulture = new CultureInfo(cookie.Value);
    }
    else
    {
        System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("en");
        System.Threading.Thread.CurrentThread.CurrentUICulture = new CultureInfo("en");
    }
}
```

Figure 32



Figure 33 (Language packs)

Name	Value
About	About
AboutText	Use this area to provide additional information.
Action	Action
ActualAddress	Actual address
AddressDetails	Address Details
AddressID	Address ID
Back	Go back
BankName	Bank name
BankNumber	Bank Number
Cancel	Cancel
CheckoutDate	Checkout date
City	City
CityAndCountry	Aalborg, Denmark
CleanInfo	Clear information
Companies	Companies
Company	Company
Contact	Contact
Cost	Cost
CustomerDetails	Customer Details
CustomerID	Customer ID
Date	Date

English

Name	Value
About	Par mums
AboutText	Izmantojiet šo apgabalu, lai sniegtu papildu informāciju.
Action	Darība
ActualAddress	Patiesā adrese
AddressDetails	Adrešes Detālas
AddressID	Adrešes ID
Back	Atgriezties
BankName	Banka
BankNumber	Bankas numurs
Cancel	Atcelt
CheckoutDate	Izrakstīšanās
City	Pilsēta
CityAndCountry	Aalborg, Denmark
CleanInfo	Notīrīt informāciju
Companies	Kompānijas
Company	Kompānija
Contact	Kontakts
Cost	Izmaksas
CustomerDetails	Klientu informācija
CustomerID	Klienta ID
Date	Datums

Latvian

on the value it holds, decides to pick the correct language pack. In case of a faulty abbreviation, the code falls back and picks the default language, English.

As briefly hinted before, all the text presented on this web client, is held inside language packs (see figure 35). Text documents, that hold both an ID and a Text Value, that is displayed on screen.

Figure 36, depicts how the Language Packs are imported in every view, and how the values are called upon, in the appropriate places.

```

1 using WebClient.Resources.Language_Files
2 model.Core.Model.Login
3 @ViewBag.Title = "Login";
4
5 using (Html.BeginForm())
6 {
7     @Html.AntiForgeryToken()
8
9     <div class="row">
10         <div class="col-sm-12">
11             <div class="form-group">
12                 @LanguagePack.Name
13                 <div class="col-md-10">
14                     @Html.EditorFor(model => model.Username, new { htmlAttributes = new { @class = "form-control" } })
15                     @Html.ValidationMessageFor(model => model.Username, "", new { @class = "text-danger" })
16                 </div>
17             </div>
18
19             <div class="form-group">
20                 @LanguagePack.Password
21                 <div class="col-md-10">
22                     @Html.EditorFor(model => model.Password, new { htmlAttributes = new { @class = "form-control" } })
23                     @Html.ValidationMessageFor(model => model.Password, "", new { @class = "text-danger" })
24                     @Html.ValidationSummary(true, LanguagePack.Error2, new { @style = "color: red" })
25                 </div>
26             </div>
27         </div>
28     </div>

```

Figure 34 Showing how language pack is implemented in code

---

### c) Tests

Testing is an integral part of developing an application. It can determine if the logic behind the software's designer is correct or not, it cannot however identify all the faults within the software. Testing in its simplest form means running an application with the intention of finding bugs and errors that the user might encounter during normal or abnormal usage, this can be done as soon as an executable application is present. The number of tests which can be run is near infinite so it's important to only select ones that are feasible within the given constraints (such as most importantly time).

In order to ensure that the development process goes as smoothly as possible, we decided to build several tests for all of the functions of our code. We decided to separate them in three main groups:

- I. Unit tests;
- II. Integration tests;
- III. Acceptance tests;

Although, programmatically similar, concept wise, they would serve a different purpose. And once put together, would ensure that our product works as intended. These tests are also part of the way we're ensuring the final quality of our product has lived up to our and Means's expectations.

#### I. Unit Tests

Unit tests are our 1<sup>st</sup> line of defense against bugs and unintended code behavior. Our unit tests are focused on testing separating individual methods and testing them under as many scenarios as possible. The tests separation being achieved by "Mocking" or "Substituting" unnecessary elements for the specific scenario the test is supposed to verify, such as constructor parameters, objects, or database elements and responses.

#### II. Integration Tests

Our integration tests focus on testing entire backend user stories, correctly initializing all required elements, just as they would be initialized in a normal run of the program. Due to the nature of our project (Web API), in order to correctly test out all of the backend functionality, including Web Api specifics (such as: Routing, Filters, and Decorators), we had to create a parent class, that would run a "FakeServer". The "FakeServer" would come with several other perks, such as: enabling us to run the Integration Tests without forcing us to host it in IIS beforehand, enabling our continuous integration service (Appveyor) to build and run our integration tests, and allowing us to debug the entire Web Api, without having to attach the Visual Studio debugger to the IIS process. All these perks, drastically improving development time and the programming experience, by providing ease of use.

#### III. Acceptance Tests

The Acceptance tests are meant to test use cases as they would happen in a real-world environment (sending requests from the MVC client and testing out visual elements in a browser window), essentially simulating a real person's manouvers just as it would be sitting in front of a computer. This is achieved with the help of the use of web drivers, run by Selenium, a portable software-testing framework for web applications. We chose to use this framework, instead of many others like: Squish, Tricentis, or TestCraft, due to its:

- 
- Compatibility across Operating Systems;
  - Support for multiple scripting languages (including C#, which we are most familiar with and are using for this project);
  - Support for running tests on multiple browsers;
  - Large and open community (in case we needed assistance);
  - Extended documentation (in case we needed assistance);
  - Open Source (free to use and a large community behind it);

Although these tests are quite important, we decided to test only the most common successful scenarios of the most common functionality, that a user might use, due to the fact that the Acceptance Tests take quite a while to run, develop, and ultimately, debug. Another small down side of our Acceptance Tests is that our continuous integration service of choice (Appveyor), is not able to run them, due to the fact that the Acceptance Tests require a browser to be installed, while Appveyor uses fresh images for their nodes, for every build.

#### d) Encountered Difficulties

Just as every project this one, did not go without a little bit of struggle. As we started to implement our solution, we have encountered “few” issues related to Entity Framework. Issues, which sometimes returned peculiar error messages such as: “cannot attach to \*.mdf” or “the underlying provider, failed to start”. It was found out later, that this issue could be solved in various ways, for example by not calling the database, whose location wasn’t specified from two different projects, as it would be trying to look for this database in the wrong folder. Or it can also be solved, by just providing a specific location where the database should be, so all interaction with the database would be in one single place.

Another issue we encountered involved the Visual Studio Compiler, that would at times indicate errors or warnings, where none would in fact be (sometimes tests would also not be started). Issues which we no longer encountered after we decided to use the Resharper plugin. This plugin improved our programming experience by providing a better UI interface for things like auto completion, scaffolding and importing references and Nuget packages, into the project, but all this came at the cost of a lot slower Visual Studio response times.

Besides already mentioned issues, one of our team members encountered a problem in testing, where the back-end project would not give responses to requests, saying “Object reference not set to an instance of an object.”, indicating, that the response message to the request couldn’t be made. This issue was not fully fixed by the time of hand in, but the team member found workaround, by debugging and reviewing the response that he would be receiving on his own machine (manually looking at value, which would be retrieved), while testing if the test passes on Appveyor.

Other issues included mismatched Nuget versions which caused compilation and various Migration errors.

---

## 5. Company Feedback

Sia Means  
Kurzemes iela 29  
Tukums, LV-3101  
Arnis@means.lv  
20. December, 2018

Back in the early 2018, Means embarked on a project unlike any other in the company's history. Means decided to upgrade their resource managing system, so it would be possible, to further expand the company and its operations. To achieve this goal we contacted several students, introducing them to our company and its needs, and after some time we received offer, saying, that they would be willing to cooperate with us.

We at Means needed a resource management system, which would provide all functionality the company needs in one single software solution. At the time, the solutions we used were slow and with limited amount of use cases. Meaning that there is no single place, where all the necessary information is saved, essentially making the company inefficient in dealing with data. The students, proposed a software solution that would handle all of our uses cases, while allowing for future expansion of the system, as the company would grow and extend, both in the employee count and working stores.

Work on the product started in late October. In the beginning there were many inquiries from one of the student regarding more in-depth information about what the product should achieve. Later in early December we received an early release of product, showing working invoice, task and item man agent parts. The release we received was also having other features which would improve use of the solutions, solution, features like: multi lingual website and the automation of some use cases, so there would be as little time wasted doing day to day tasks. Even though product did what was required, there were some issues with how product was working and it visual style, but after the team was notified about them they quickly fixed the issues, making use of the feedback they had received.

The current solution, that we have received has all the required functionality, which we expected from this product, as well as already mentioned extras, which are welcome additions. Other than the use cases which were made, there was more we would have wanted to have in the product, but just as they previously warned us, the product could not have been finished in such a short amount of time and would require a lot more work, in order to be done with all use cases, but this is a great starting point.

In total we are satisfied with the product and the students we worked with and the level of professionalism they have shown, as they showed great interest in developing a solution, which would not only fulfil our needs, but they also went further and worked on improving the usability of the product, by making it as automated and understandable as possible, while using the feedback we have given them for their work.

---

## 6. Conclusion

In conclusion, during this semester we managed to, not only gain knowledge about various web technologies, especially the Web API, and a few new programming languages (such as javascript, razor, CSS), but also strengthened our ability to collaborate with a company, when working on a project they are interested in. Experience and knowledge, which we believe, holds a great value, in regard to our future carriers.

Our project turned out to be quite close to what we imagined when we were just pitching the idea to the company. Our list of features includes:

- Manage invoices – invoices contain information about the incoming orders from other businesses, these information parameters involve: quantity, serial number, name, barcode, price, tax group of the product or products in question. They can be added/deleted/modified in the invoice list section
- Manage items – Items (also referred to as Products) can be added/deleted/modified in the Items section. Items have parameters such as: serial number, name, barcode, tax group in both incoming and outgoing form (incoming from other countries, outgoing in Latvia)
- Manage Jobs – Jobs are more complex as they have: name, start/end time, cost, status and a description. With jobs needing a customer they have the following attributes: name (a separate input for given and family name) email and a phone number along with address details. The jobs can be added/deleted/modified in the Job List section
- Users can be added/deleted/modified under the Registration/Edit Current User section, the users attributes include a username and a password
- The already mentioned language selection which currently includes English and Latvian
- Logging/tracking of employees' activities to provide troubleshooting capabilities
- Email notification implementation using Gmail API
- Encrypted user details with enhanced browsing safety thanks to HTTPS

We learned a lot of interesting facts regarding web technologies and how they work, while researching various problems we encountered. We also strengthened the knowledge gained during the internship period, knowledge which had a great impact on the way we worked on this project, especially regarding the “Working Culture” and “Development Method”.

Although there is room for improvement, for example: more features, enhanced API documentation and additional clients; both the company and we are equally satisfied with what we have achieved.

We suspect due to Hand-in website limits, the entire repository cannot be uploaded. To see how we worked and what files we created, one has to follow the link, which will take you to our GitHub repository: <https://github.com/RaidenRabit/Phaethon>

As an ending note, we would like to thank all the readers, who invested their time in reading this paper, also the guiding teacher, who helped and guided us throughout the entire process.

---

## 7. References

During the development process, we used the following applications and environments:

- Kanban board: <https://trello.com/b/2RX6l8nl/sem5project> (Trello)
- Official means of communication: <https://sem5project.slack.com/> (Slack) and weekly real-life meetings.
- Version control solution: <https://github.com/RaidenRabit/Sem5Project> (GitHub)
- Deployment testing environment: <https://ci.appveyor.com/project/RaidenRabit/sem5project> (appveyor)
- IDE: Visual Studio, with ReSharper plugin.

[https://www.tutorialspoint.com/mvc\\_framework/mvc\\_framework\\_introduction.htm](https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm)

<https://docs.microsoft.com/en-us/aspnet/web-forms/what-is-web-forms>

[https://www.w3schools.com/asp/webpages\\_intro.asp](https://www.w3schools.com/asp/webpages_intro.asp)

<https://stackoverflow.com/questions/2698151/entity-framework-vs-linq-to-sql-vs-ado-net-with-stored-procedures>

<https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/>

<https://www.dotcominfoway.com/linq-to-sql-vs-ado-net-entity-framework/>

<http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/>

<https://martinfowler.com/eaCatalog/domainModel.html>

<https://www.asp.net/web-api>

<https://www.saksoft.com/advantages-of-web-api-over-wcf/>

<https://www.c-sharpcorner.com/UploadFile/1d42da/web-service-basics/>

<https://social.msdn.microsoft.com/Forums/en-US/435f43a9-ee17-4700-8c9d-d9c3ba57b5ef/advantages-amp-disadvantages-of-webservices?forum=asmxandxml>

<https://raygun.com/blog/soap-vs-rest-vs-json/>

<https://docs.microsoft.com/en-us/dotnet/framework/wcf/wcf-and-aspnet-web-api>

<https://www.webopedia.com/TERM/A/API.html>

<https://serverfault.com/questions/57077/what-is-the-difference-between-authentication-and-authorization>

[https://en.wikipedia.org/wiki/Man-in-the-middle\\_attack](https://en.wikipedia.org/wiki/Man-in-the-middle_attack)

<https://www.ssl.com/article/pros-and-cons-of-ssl-https-tls/>

<https://en.wikipedia.org/wiki/HTTPS>

<https://www.globaldots.com/8-http-security-headers-best-practices/>

[https://www.owasp.org/index.php/HTTP\\_Strict\\_Transport\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet)

<https://www.globalsign.com/en/blog/ssl-vs-tls-difference/>

<https://blog.elmah.io/improving-security-in-asp-net-mvc-using-custom-headers/>

[https://developer.mozilla.org/en-US/docs/Web/HTTP/Public\\_Key\\_Pinning](https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning)

<https://scotthelme.co.uk/hsts-preloading/>

[https://msdn.microsoft.com/en-us/library/vstudio/cc716760\(v=vs.100\).aspx](https://msdn.microsoft.com/en-us/library/vstudio/cc716760(v=vs.100).aspx)

<http://www.informit.com/ARTICLES/ARTICLE.ASPX?P=375542&SEQNUM=3>

SCOTT KENDALL (2002) THE UNIFIED PROCESS EXPLAINED

KENT BECK: EXTREME PROGRAMMING EXPLAINED: EMBRACE CHANGE, ADDISON-WESLEY

<https://www.scrum.org/resources/plan-driven-vs-value-driven-development>

<https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/WHAT-IS-FURPS/>

---

<https://www.scaledagileframework.com/NONFUNCTIONAL-REQUIREMENTS/>

<https://www.scrumalliance.org/learn-about-scrum>

BALANCING AGILITY AND DISCIPLINE: A GUIDE FOR THE PERPLEXED - BOOK BY BARRY BOEHM AND RICHARD TURNER

<http://www.craigarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf>

## 8. Appendix

### a) Original Latvian Feedback By Means

Sia Means  
Kurzemes iela 29  
Tukums, LV-3101  
Arnis@means.lv  
20. Decembris, 2018

#### Interesentiem:

2018. gada sākumā SIA Means sāka jaunu projektu, kas atšķiras no jebkura cita uzņēmuma vēsturē. Tika nolemts uzlabot kompānijas resursu pārvaldību, turpināt attīstīt uzņēmumu un efektīgāk vadīt procesus. Lai to darītu, uzrunājām Datorzinātnes fakultātes studentu, iepazīstinot viņus ar uzņēmumu un tā vēlmēm. Saņēmām piedāvājumu veidot atbalsta produktu universitātes projekta laikā.

Means bija nepieciešama resursu pārvaldības sistēma, kas nodrošinātu visu funkcionalitāti uzņēmuma vajadzībām vienā produktā, jo sistēma, kas tolaik tika izmantota, ir lēna un ar ierobežotu funkciju skaitu. Šajā sakarā uzņēmumam ir vairāki risinājumi, kurus izmanto dažādām vajadzībām kā rezultātā nav vienotas nepieciešamās informācijas uzglabāšanas sistēmas, padarot uzņēmumu neefektīvu datu apstrādē.

Produkta izveide sākās oktobra beigās, kad studenti ieguva specifisku informāciju par produkta funkcionalitāti, izskatu un produkta uzdevumu. Vēlāk decembra sākumā mēs saņēmām priekšlaicīgu produkta izlaidumu, parādot rekvizītu, uzdevumu un priekšmetu pārvaldības daļas, kā arī citas funkcijas, kas uzlabo risinājuma izmantošanu, piemēram: iespēja strādāt multilingvistiskā vidē un automatizētu lauku lietojums, jo mēs vēlējamies maksimāli ekonomēt laiku ikdienas uzdevumu veikšanai, taču bija dažas problēmas, par kurām studentu komanda tika informēta, un viņiem izdevās tās veiksmīgi atrisināt, lai padarītu produktu atbilstošu uzņēmumu vajadzībām.

Galīgajam risinājumam, ko esam saņēmuši, ir visas nepieciešamās funkcijas, ko mēs sagaidām no šī produkta, kā arī jau minētie papildinājumi, kas ir apsveicami. Neskaitot galvenos lietošanas gadījumus, mēs gribējām vairāk funkcionalitāti, bet, kā izrādījās, un studenti mums paziņoja, produkts nevarēja tikt pabeigts tik īsā laikā un prasītu daudz vairāk darba, lai to izdarītu ar visiem lietošanas gadījumiem, bet šis ir lielisks sākumpunkts.

Kopumā mēs esam apmierināti ar produktu, studentiem un profesionalitāti, kuru viņi izradīja. Viņi izradīja lielu interesi izstrādāt risinājumu, kas ne tikai atbilstu mūsu vajadzībām, bet arī turpināja darbu un atbildīgi strādāja, lai uzlabotu produkta lietojamību, padarot to par daļēji automatizētu un saprotamu, izmantojot mūsu atsaukumi par padarīto, ko mēs viņiem esam devuši sava darba tālākai pilnveidošanai.



---

b) Contact Info

- Phone: +371 63182071
- Website: <https://www.means.lv/>
- Registration number: 40003532375
- Legal address: Latvia, Tukuma nov., Tukums, Kurzemes iela 29, LV-3101

c) Group Contract

**I Definitions:**

PROJECT – Semester 5, final AP Degree in Computer Science, project. The official name of the project being: Phaethon.

MEMBER – Refers to everyone who signed the contract.

**Official means of communication: Slack group, Physical meetings, Provided email.**

**II Contract Terms:**

This Temporary Collaboration Contract states the terms and conditions that govern the contractual agreement between members of Group who agree to be bound by this Contract.

Whereas, The Group is engaged in the PROJECT; and

Whereas, The Group desires to employ and retain the services of MEMBER on a temporary basis according to the terms and conditions herein.

Now, therefore, in consideration of the mutual covenants and promises made by the parties hereto, The Group and MEMBER covenant agree as follows:

- TERM. The term of this Temporary Collaboration Contract shall commence on 01.11.2018 and continue until the completion of the PROJECT.
- TERMINATION. MEMBER agrees and acknowledges that, just as they have the right to terminate their collaboration with The Group at any time for any reason, The Group has the same right, and may terminate their collaboration with The Group at any time for any reason
- Group 7 shall employ MEMBER as Project Worker. MEMBER accepts collaboration with Group 7 on the terms and conditions set forth in this Temporary Collaboration Contract and agrees to devote their full time and attention to the performance of their duties under this Agreement, as stated on Exhibit B attached hereto. In general, MEMBER shall perform all the duties as described on Exhibit A attached hereto.

- 
- **RETURN OF PROPERTY.** Within Seven (7) days of the termination of this Temporary Collaboration Contract, whether by expiration or otherwise, MEMBER agrees to return to The Group, all products, samples, or models, and all documents, retaining no copies or notes, relating to the Company's business including, but not limited to, UML Models, ID numbers, Code lines, Contracts, in any form or measure, obtained by MEMBER during its representation of The Group.
  - **GROUP'S PROCEDURES.** MEMBER agrees and acknowledges that he or she shall comply with The Group's established disciplinary code as well as any other rules, policies, and procedures that may be introduced from time to time. Copies of such documents are available upon request.
  - **NO MODIFICATION UNLESS IN WRITING.** No modification of this Agreement shall be valid unless in writing and agreed upon by all parties.
  - **WORKING CONDITIONS.** All the employees of The Group must strictly follow the code stated on Exhibit C, attached hereto. Any violations of the code will immediately lead to the consequences stated on Exhibit B attached hereto.
  - **APPLICABLE LAW.** This Temporary Collaboration Contract and the interpretation of its terms shall be governed by and construed in accordance with the laws of the Danish State and subject to the exclusive jurisdiction of the federal and state courts located in Denmark, unless specified in the contract.

IN WITNESS WHEREOF each of the parties has executed this Temporary Collaboration Contract, both parties by its duly authorized officer, as of the day and year set forth below.

Exhibit A – Duties

In general, the duties of the position to be filled by the employee shall encompass the following:

[see PROJECT requirements]

Exhibit B – Working Performance

In general, the employee shall fulfill his duties, as stated on Exhibit A attached herein. In case one of the employee is not able to fulfill his duties, he must announce at least one of the other employees of The Group which can be seen on The Group participants, attached hereto, in case the employer who is not capable of fulfilling his job and does not announce at least one of the other employees of The Group, or has no valid reason for doing so, the other employers of The Group get to decide which consequences he must suffer, decision made by voting. For a vote to pass it requires at least 51% of the votes to be positive. In case on or more of the employers are not present for the voting, they must contact The Group and clearly state their vote, otherwise it will be considered that they voted for immediate termination of the contract. Each employer has the write to propose one consequence. Immediate termination of the contract is always an available consequence that must be voted over.

---

## Exhibit C – Working Code

Working place and hours are decided on a weekly basis, unless otherwise established through vote with a positive percentage of at least 51%, through **official means of communication**. Clothing and Language are to be kept under normal social rules. Any aggression shown towards another worker is strictly forbidden.

- Participate to the daily stand-up meetings, held in Slack and conducted by the GeekBot;
- Participate to the Refinement and Retrospective meetings, regularly held;
- In order to keep a level of traceability, please always work in new branches, based on the Development branch, with the same name as the task you're currently working on (including the trello card number) ex: #CardNumber-card-title;
- Commits should be named as follows: #CardNumber commit message;
- When you believe a task is finished, create a pull request (always write a short description of what you did when creating the pull request), DO NOT MERGE WITH Dev or Master;
- Once the branch is approved (passes appveyor checks and gets a positive review), the owner of said branch, will merge with development branch;
- Merging with master will be done at the end of each sprint or when enough tasks are done;
- Master Branch is the one we will send teachers to review, it is to be considered as ready to go branch;
- Work from anywhere at any time, but finish your tasks on time/as fast as possible, and attend scheduled meetings;

## The Group Participants

---

### • Ralf Zangis

-Date of birth: 08.03.1997

-Contact: 1062012@ucn.dk

---

### • Andrei Eugen Birta

-Date of birth: 11.02.1998

-Contact: 1062021@ucn.dk

---

### • Adam Blázquez

-Date of birth: 25.03.1997

-Contact: 1062084@ucn.dk