University College Nordjylland

AP Degree Final Project

5th Semester



Phaethon

Collaboration with: Means S/A

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

Supervisor: Jesper Strandgård Mortensen

Submission date: xx-xx-xxxx

# Table Of Contents

[Table Of Contents 2](#_Toc534716425)

[1. Introduction 4](#_Toc534716426)

[a) Problem Statement 4](#_Toc534716427)

[b) Proposed Solution 4](#_Toc534716428)

[2. Preliminary Study 5](#_Toc534716429)

[a) About the Company 5](#_Toc534716430)

[b) ROI analysis 5](#_Toc534716431)

[c) SWOT 5](#_Toc534716432)

[d) Stakeholders analysis 6](#_Toc534716433)

[e) Group Contract 7](#_Toc534716434)

[f) MOSKOW analysis 10](#_Toc534716435)

[3. System Development 11](#_Toc534716436)

[a) Development Method of Choice 11](#_Toc534716437)

[I. Method Selection 14](#_Toc534716438)

[i. Pros 15](#_Toc534716439)

[ii. Cons 15](#_Toc534716440)

[b) Quality Assurance 15](#_Toc534716441)

[FURPS  15](#_Toc534716442)

[c) System Design 16](#_Toc534716443)

[I. System Architecture 16](#_Toc534716444)

[II. Database Model 21](#_Toc534716445)

[III. Domain Model 21](#_Toc534716446)

[d) Working culture 23](#_Toc534716447)

[4. Technology and Programming 24](#_Toc534716448)

[a) Analysis 24](#_Toc534716449)

[I. Framework 24](#_Toc534716450)

[II. Database 26](#_Toc534716451)

[III. Client 30](#_Toc534716452)

[Decision: 30](#_Toc534716453)

[IV. Middleware 33](#_Toc534716454)

[1. System-Database 33](#_Toc534716455)

[2. Clients-System 33](#_Toc534716456)

[b) Implementation 33](#_Toc534716457)

[I. Security 33](#_Toc534716458)

[HTTP security headers 37](#_Toc534716459)

[Tokens 39](#_Toc534716460)

[II. Fake Server 40](#_Toc534716461)

[III. Decorators 42](#_Toc534716462)

[IV. Multi language 45](#_Toc534716463)

[c) Tests 47](#_Toc534716464)

[I. Unit Tests 47](#_Toc534716465)

[II. Integration Tests 48](#_Toc534716466)

[III. Acceptance Tests 48](#_Toc534716467)

[d) Encountered Difficulties 48](#_Toc534716468)

[5. Company Feedback 48](#_Toc534716469)

[6. Conclusion 49](#_Toc534716470)

[7. References 49](#_Toc534716471)

# Introduction

“Means SIA”, is a small company, based in Tukums Latvia. It is currently thriving to succeed in the computer repair, assembly in both the private and wholesale business. Means contacted one of our team members at the end of 2018’s summer, to enquire for the creation of a software solution that would solve their management and organizational problems. In this report, we will present you our proposed solution, that would benefit both us and company by giving us a chance to work on a real problem for real customers. We aim to successfully develop an application which would provide a good solution for Means and learn to overcome the challenges which will arise along the way.

The project is due mid-January 2019, by the time which we intend to have Means utilize our software solution.

## Problem Statement

As a company in the business of both perivate 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 providing
* 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 system (the classic pen and paper) 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 relining on human memory to remember deadlines

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

## Proposed Solution

We aim to achieve our objectives, by creating a distributed software solution, using REST-full WebApi as our back end, and MVC as our main front-end solution. Due to time constrains we will only focus on the hosted application but we plan on rolling out several other front-end clients, like a dedicated windows app and mobile compatible apps. All to facilitate the company's future needs and ease both their employees and the companies' leaders work. We also plan on comparing sales figures pre-software launch and with the help of the company give post launch numbers to further investigate future problem areas and improve their return of investment.

# Preliminary Study

## About the Company

This section refers to the values of the company Means. The company’s mission focuses on the current or short-term goals while the vision refers to the long-term objectives.

In our case, Mean’s 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 field, Means offers the best option for customers on a budget.

Means’s 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.

## ROI analysis

The online computer retail company Means currently has a 20 - 30k 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 what are the most popular items, what is in stock, 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 it’s 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 which we offer by having a minimalistic approach to the UI, avoiding the need of spending hours learning the interface. While Means does not expect an increase in revenue, we recon by helping cutting down the number of errors and speeding up the time it takes to ship a given product Means will be able to see a slight increase in monthly sales thus an increase in revenue.

## SWOT

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

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.

S: They have a strategic advantage in their HQ city, long-term contract, high profile customers,

W: Disorganization, the number of employees, hard to find employees

O: Brand recognition, online store, networking from high profile customers

T: Not repairable computers, loss of modularity, online shops

## Stakeholders analysis

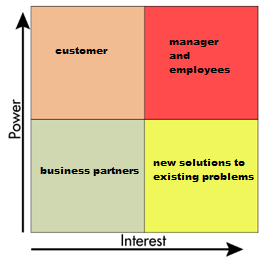
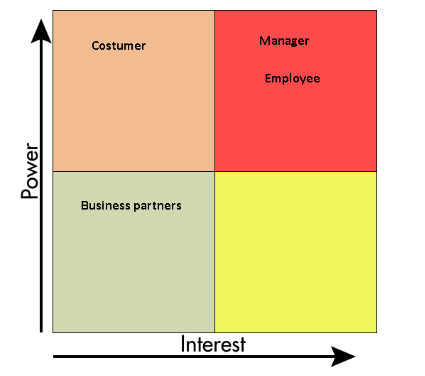


Figure (stakeholders diagram)

In this section we will go into greater details describing about stake holders of this product. In total there are 4 stake holders: employee, manager, supplier and customer, but due to limited time we decided to focus more on two of them: employees- making them more productive and reducing the unnecessary work done on day to day basis and to lesser degree for manager of company- making management of company/employee easier and furthering the company by increasing profits. The decision we made to focus on only 2 users allowed use to make better preforming program, which is specifically made for required use cases and let us focus more on quality rather than quantity.

Stakeholders:

1. Manager- interested in more productive employees

2. Employee- wants to make existing processes more streamlined and less time consuming

3. Costumer- 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

## 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](https://www.facebook.com/staniek.rajmund)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](https://www.facebook.com/cervinka.marek)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](https://www.facebook.com/cervinka.marek) 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](https://www.facebook.com/cervinka.marek) as Project Worker. [MEMBER](https://www.facebook.com/cervinka.marek) 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](https://www.facebook.com/cervinka.marek) 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](https://www.facebook.com/cervinka.marek) 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](https://www.facebook.com/cervinka.marek) during its representation of The Group.
* GROUP'S PROCEDURES. [MEMBER](https://www.facebook.com/cervinka.marek) 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 appveyour 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](mailto:1062012@ucn.dk)

**• Andrei Eugen Birta**

-Date of birth: 11.02.1998

-Contact: [1062021@ucn.dk](mailto:1062021@ucn.dk)

**• Adam Blázsek**

-Date of birth: 25.03.1997

-Contact: 1062084@ucn.dk

## MOSKOW analysis

After selecting our system development method of choice we almost immediately started on our 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 onwer stayed in constant communication with Means to ensure we were on the right development track. The product was shown to company after every sprint to receive their feedback and use that feedback to further suite the company’s needs.

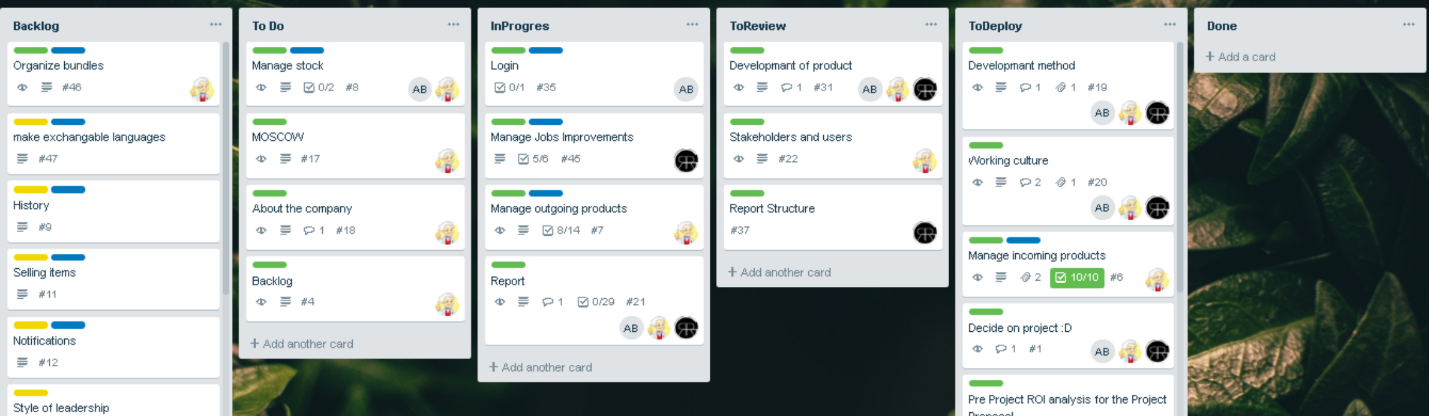


Figure (Kanban board, with MOSKOW coloring)

### Steps of Development:

We decided that there would be need for 6 steps of development, this decision was made, since we had already worked using agile method and based on experience gained we decided to improve our previous board, by using knowledge from our previous projects and from internship period. When product goes from idea to reality, it has to go through all of the following steps:

1. **Backlog** – Place where all not started tasks are kept;
2. **To Do**- Task for current sprint, with already assigned working member;
3. **In progress**- Location, where 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, deployed to the customer;

### Tasks:

To improve the readability of task we decided to integrate new features into task board:

MoSCoW Decision to add it was made, to allow product owner and group realise the importance of the task, so group wouldn’t drift from the main idea of project too far, trying to achieve something that’s not required.

Moscow model can be seen by looking at the item and labels colour assigned to it:

1. Green- must
2. Yellow- should
3. Orange- could
4. Red- wont

**Task separation**- we also wanted to make backlog more easily readable, by adding another label which would indicate if task is in programming.

1. Blue- programming
2. No colour- others

1. **Comments**- Our group decided that each task should have description added to it, so when it is assigned to a group member, he wouldn’t have to guess what he has to do, and he wouldn’t waste his time making something which was not required by task.

1. **Check list**- To further the goal of allowing user to understand what’s required by task and to reduce the possibility of forgetting something to do it was enforced and expected that a checklist, representing definition of done is to be added to each item in board.

# System Development

## Development Method of Choice

At a high-level software process models can be put into two categories:

Plan driven: All steps are carefully planned, variables such as budget and deadlines are fixed in advance and progress is measured against these criterias.

Agile: All the steps are done in small increments with minimal pre planning and are modified accoring to feedback and changes in requirements

## More indept review

True to its name, plan driven development requires extensive research with precise planning before beginning the development of the application. The plan requirements are researched early on in the beggining 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 chocie, 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.

### 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 Watefall method compared to UP is more strict in its process as everything needs to be done sequentally. UP is more free in this reagard as its an iterative and incremental process. It has been devided into four phases: inception, elaboration, construction and finally transition. Each phase is devided into interations (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 menthods 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. Primary goal of this phase is to establish an architecture baseline, which is vision of the application that includes all 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.

### Agile

In stark contrast to the aforementioned plan driven approach, agile introduces more feedback and value driven development. In Agile the developers accept that not all requirements will be clear from day one, and the ones that are will possibly change based on customer/owner feedback and feature priorities might significatly change. To receive reliable and fast feedback, short iterations called sprints are introduced. The sprints usually last from one to a few weeks depending on the tasks and their complexity, but these iterations are still shorter compared to the ones found in Plan driven methodologies.

Figure representing various sprints

Scrum is one of the agile development methods 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 feeback given by the final users/cutomers. All agile methodologies acknowledges that requirements may not be known in advance and therefore unlike plan-driven approaches, requirements are only fixed for the duration of one sprint.

Each sprint is closed witha 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 workspace 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.

Extreme Programming (XP) is another agile development methodology which focuses more on the quality of the software. XP is based on four main values. The first and arguably the most important value is communication. Communication is crutial in any methodology, but especially in agile as the requirements can change rapidly. Simplicity, the simpler the solution the less chance something fails. Development is to be started with a simple solution and improved with continous refactoring (also one of the key XP practices). Feedback is the 3rd value which is crucial for product quality. The last value is courage which encourages making decisions without the need of a formal process. This is the complete opposite in contrast 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. Amongst 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.

As with the previous two agile development methodologies, Kanban aims to work with the caothic nature of variable requirements and the need to shift focus between priorities quickly. Kanban also focuses on making the development as smooth as possible with making transparency, respect and agreement as its values (there are a lor more but these are the ones we aimed to retain when considering the method). Leadership is key and is encouraged at every level (again circling back to its values) and the changes are evolutionary rather then 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 visualised 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 minimaize 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)

### Methodology Selection

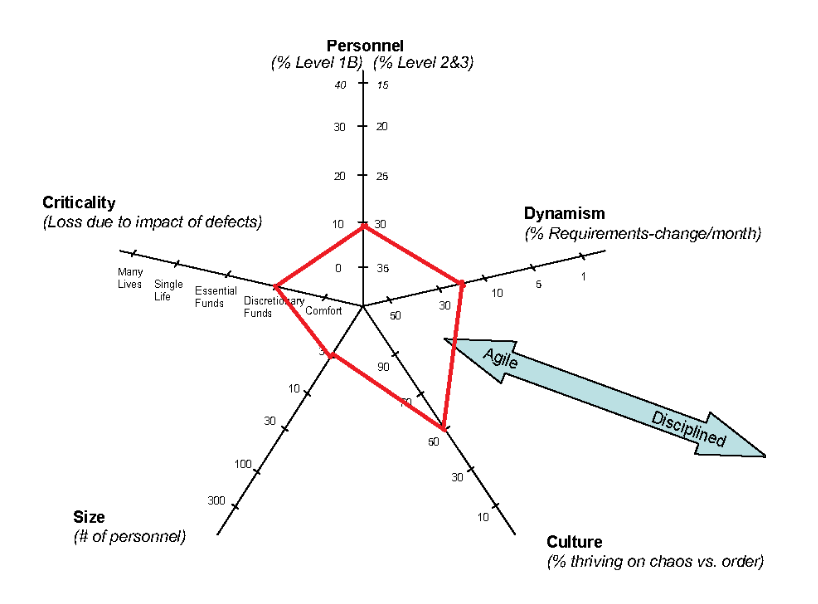


Figure (Bohem and Turner diagram)

Choosing the right development method is key to a succesful product. One must first asses the project, the team, scale and 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 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.

So we decided to create a unique combination of all 3 major agile development methods. One that would contain the following guidelines:

* 40 hour week
* Planning Poker estimation
* Daily scrum meetings (managed by bot)
* Product owner
* Sprint based deployment
* Product backlog
* Sprint backlog
* Weekly standup meetings (organizedand conducted by product owner)

While we aimed to keep using our unique combination of agile methodologies but ended up defaulting to XP and aiming to first build a Minimal Viable Product (MVP). After the MVP has finished, we gradually added functionality without braking the application.

Our focus was 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 was also used extensively to help speed up the early stages of development and reach a presentable and error free product as fast as possible.

## Quality Assurance

### FURPS

With agile putting emphasis on the programming instead of planning, it’s a good idea that developers ensure their stories are FURPS+ approved. FURPS (later the plus was added for additional requirements the customer might impose) is a set of quality attributes that raises the quality of the application by making the developers adhere to them.

Functionality – what the customer wants it to do

Functionality represents the main functional requirements in the 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 categories represent the non-functional requirements which are architecturally important. Knowing these requirements helps us ask the customer the right questions.

Usability – how effectively can the target audience use it, focusing on the UI

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.

Reliability – how reliable its data must be, how low the downtime must be and accuracy of the retrieved information.

 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 – how fast it must be, how resource intensive it must be

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.

Supportability – how configurable, editable, expendable it must be

The application has been built with supportability in mind which will be discussed later in the document.

## System Design

### System Architecture

Pick the best architecture that would fit our needs, was a pretty long 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 represent the evolution of our architecture through Umlet diagrams, as they would be seen in different steps in early and late stages of the development and the final solution.

Figure 5, 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.

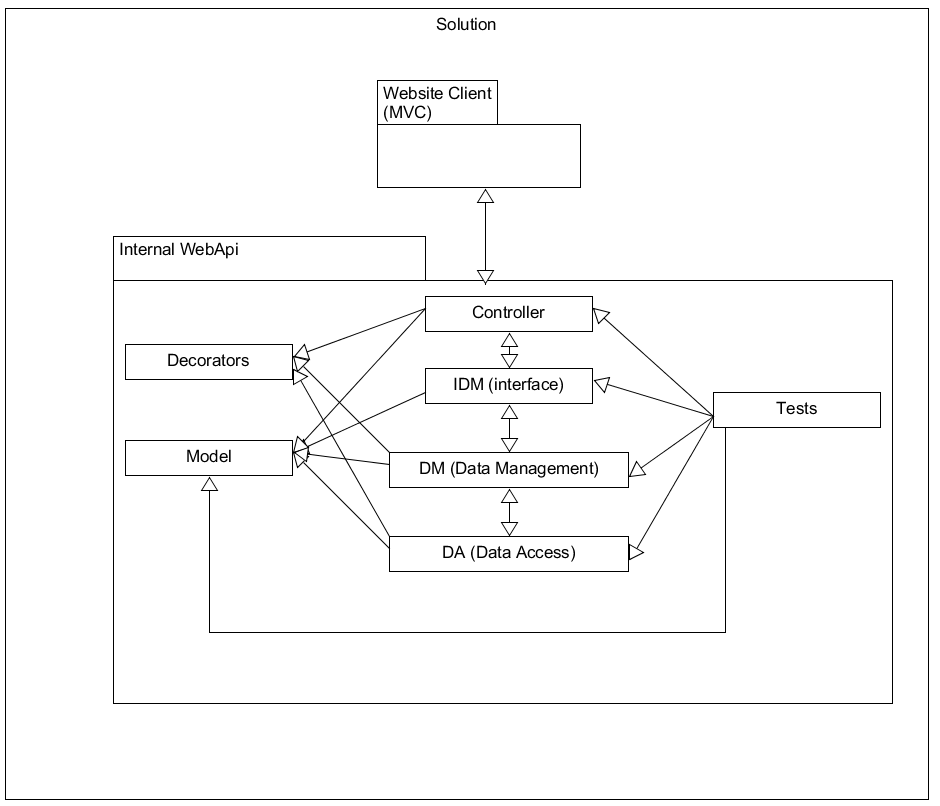


Figure (System Architecutre diagram v.1)

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 WebApi, composed of several parts:

Controllers (also known as Endpoints): which would deal with httpRequests/Responses, serializing and deserializing and other communication parts:

* DataManagement (and the affiliated interface): meant to deal with most of the logic of for this solution;
* DataAccess: meant to deal with the communication to the database;
* Model: meant to hold all of 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 will create;

Figure 5 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 help with that, by adding all the required Documentation, that any WebApi should have.

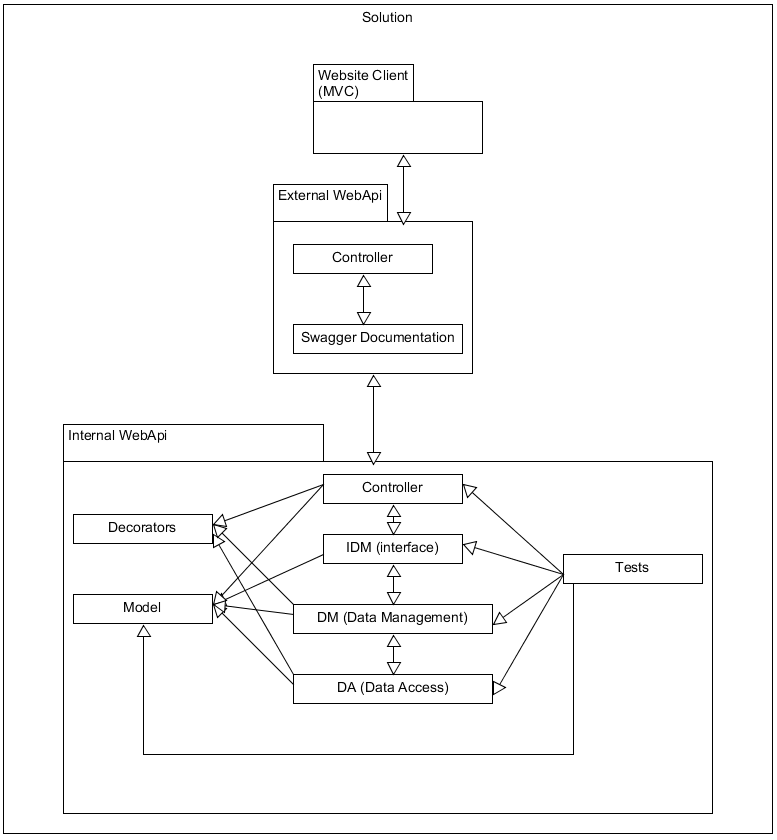


Figure (System Architecutre diagram v.2)

For the documentation, we decided to use Swagger, as it uses the scaffolded XML files, and makes a website that fancily displays our webapi’s endpoints, what their parameters and their possible response messages.

Another function that was attributed to the External Api, 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 realised that we could reduce the code duplication, improve building times and overall performance, while also increasing code’s cohesion, by simply moving around some of the already existing parts, into dedicated projects. This gave presented our 3rd and “final” architecture, (represented in Figure 7) 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 other projects. This would also allow us to simply change a Core class, without having to rebuild the other projects, as long as it did not affect their functionality.

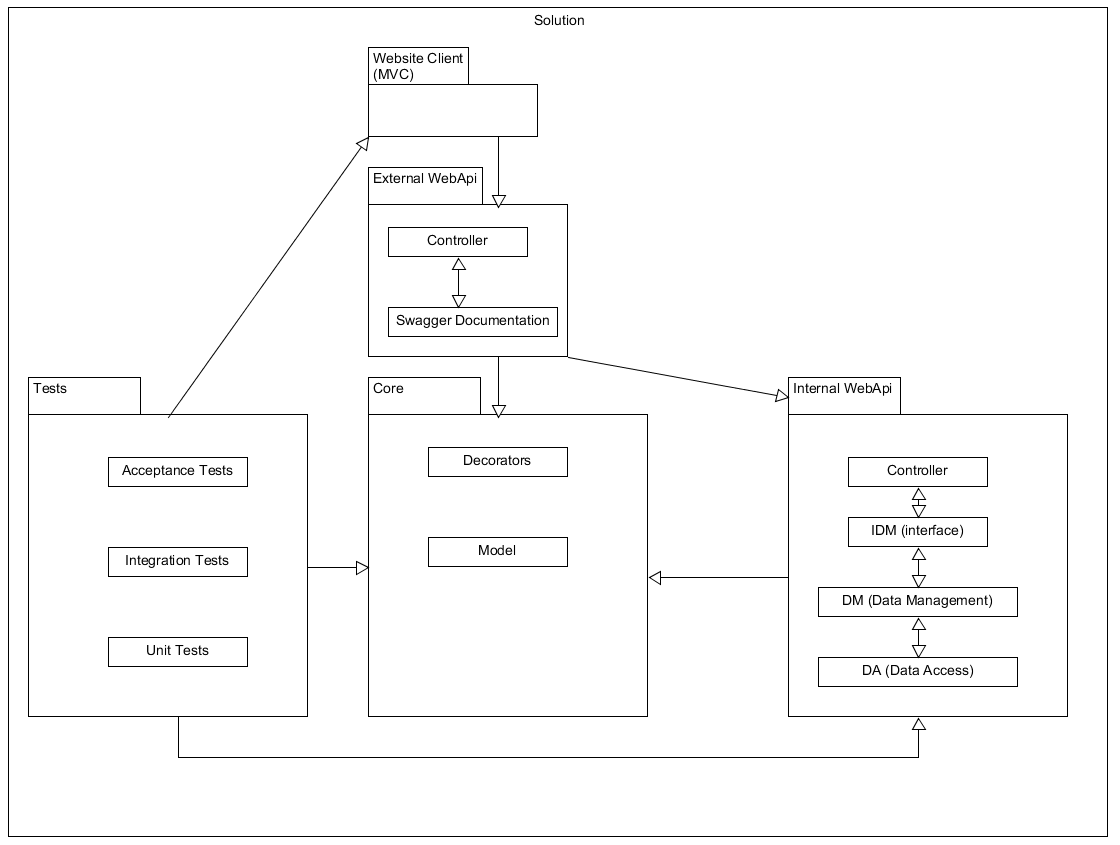


Figure (System Architecutre diagram v.3)

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

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 of the downsides of having one. Downsides such as: slightly slower responses, and harder to develop new/edit endpoints.

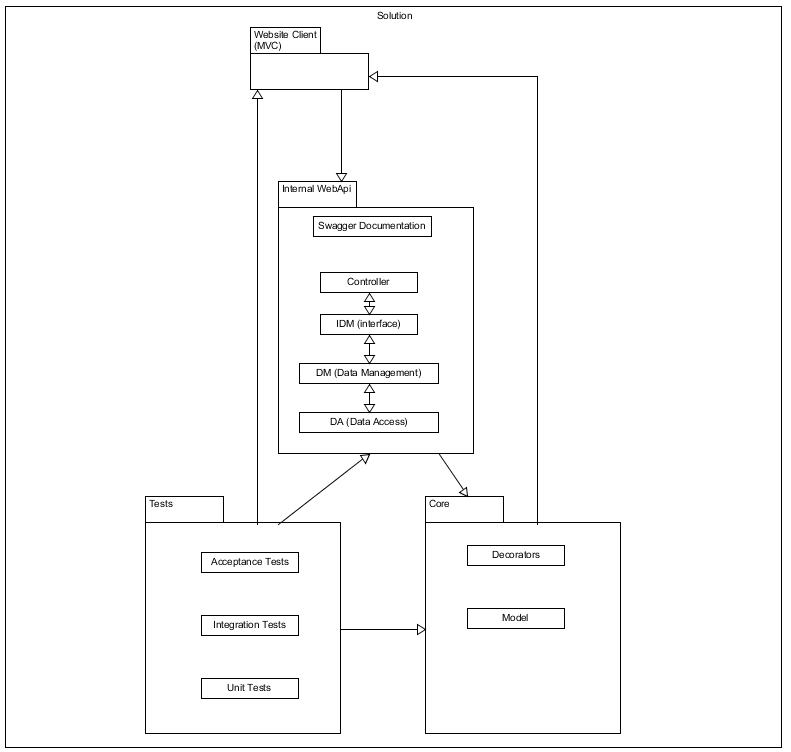


Figure (System Architecutre diagram v.4)

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.

### Database Model

Our database diagram, just like domain model early on changed quite a lot, but we managed to get to final solution (which we used through project) also early on. As seen in Figure 8, its visibly like domain model and just like explained there, we didn’t manage to finish all work, that we intended, having employee not being implemented, due to unexpected issues and us wanting to focus more on quality of product, rather than quantity.

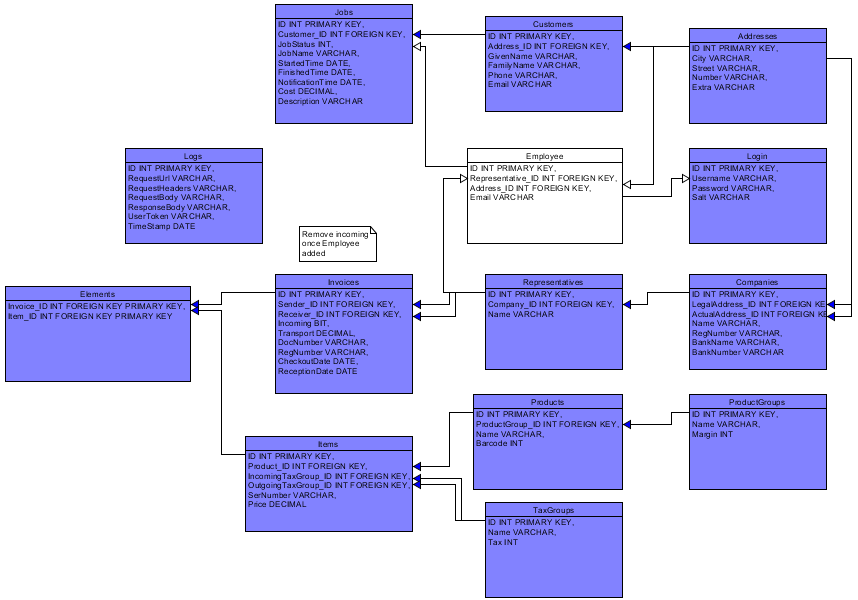


Figure (database diagram)

In Figure 9 you can also notice, that there is table called “Logs”, where we are saving all of 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.

### Domain Model

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

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).

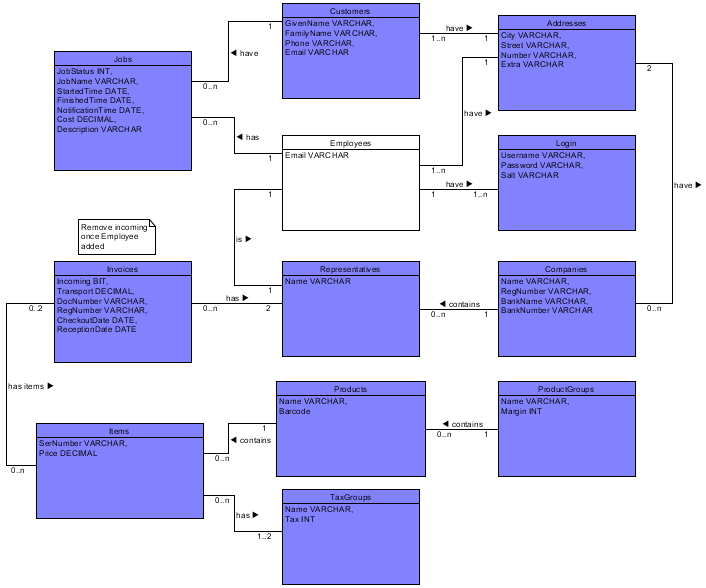


Figure (Domain model)

Explanation of each element:

Address- Holds all location information.

Customer- Information of interest companies costumer.

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

Employees- Holds information of the products user.

Login- Holds registered users details, to access the system.

Companies- Holds information of all the known companies.

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

Invoices- information, regarding the deals between two companies.

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

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

ProductGroups- Holds details shared by many Products (Example: iPhone X, Galaxy S9 and Huawei mate 20 all are phones)

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

### Working culture

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

Kanban board: <https://trello.com/b/2RX6I8nl/sem5project> (trello)

Official means of communication: <https://sem5project.slack.com/> (slack) and real life meetings.

Version control solution: <https://github.com/RaidenRabit/Sem5Project> (github)

Deployment testing environment: <https://ci.appveyor.com/project/RaidenRabit/sem5project> (appveyour)

IDE: Visual Studio, with ReSharper plugin.

General rules and guidelines:

* Daily, scrum meetings, organized by slackBOT
* Weekly Refinment 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 be done and ultimately: write a Definition of Done to said task. Only after a task is refined, it can be worked on.
* Mind Slack channels description and use them to the best of their purpose
* Several bots (trello, appveyour, 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 channel
* In order 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 apveyour 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 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
* 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.

# Technology and Programming

## Analysis

### Framework

**Technologies considered:**

After deciding how client would work, we had to decide what technology to use for client communicating with the server. After some research we came to few possible solutions, these options will be explained and our final choice will be supported by arguments in the following text.

When talking about client-server reader should know, that all Web Services are API but not all APIs are Web Services. To get more information about this reader is encouraged to read following definitions:

* **Application program interface (API)** is a set of [routines](https://www.webopedia.com/TERM/R/routine.html), [protocols](http://www.webopedia.com/TERM/P/protocol.html), and tools for building [software applications](https://www.webopedia.com/TERM/A/application.html). A good API makes it easier to develop a graphical user interface ([GUI](https://www.webopedia.com/TERM/G/Graphical_User_Interface_GUI.html)) by providing all the building blocks.
* **Web service** is a generic term for an interoperable machine-to-machine software function that is hosted at a network addressable location.
* **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.

**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
* Many content formats (XML, JSON, CSV and more)
* Supports wide variety of clients (mobiles, browsers, PC and many more)
* 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

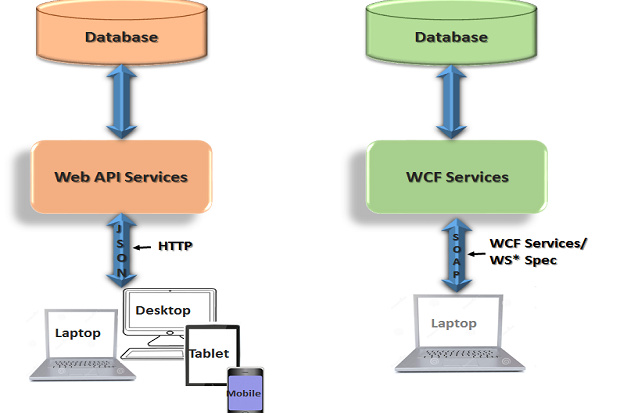
**WCF (Windows Communication Foundation)** is a set of APIs in the .NET Framework for building connected, service-oriented applications. (Service oriented)

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:

* Allows only XML content format
* A lot of time is spent in configuring services

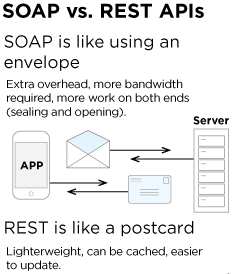


After reviewing the Web API and WCF we came to conclusion, that we wanted first of all, to have a service which would be usable though multiple devices, that’s why we came to conclusion that we should use Web API instead of WCF. Although WCF could be better solution for project called InternalApi, as it would be easier for us to maintain, test and change the transport (Both projects would be located on the same computer or at a minimum located on computers connected though LAN, where TCP protocol would be better), we determined, that we could learn a lot and save some time for developing solution, using only Web API, as there are many developers warning about the WCF and that configuration could take long time even for professionals.

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

REST- Is relative newcomer and as an architecture style does not require processing, is lightweight and is naturally more flexible than SOAP, requesting user to just use URL and can get response in multiple different formats, while SOAP is restricted to XML.

SOAP- Old but still relevant solutions to data exchange problem, it is 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, you need to build those requests manually, which becomes problematic because SOAP is intolerant of errors, but when working with .NET languages, you don’t have to even see the XML.



### Database

**Choosing right database:**

When choosing the right database for project, we had to make multiple choices and one of the biggest ones were choosing, whether we shouldrelational or Non-relational database. To decide we concluded that it would be useful, to have a list of pros and cons so we could make informed decision, not doing it just on gut feeling, 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.

**Relational Databases**

**Pros**

* Relational databases work with structured data.
* They support ACID transactional consistency and support “joins.”
* 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 joins, which affects speed.
* They have problems working with semi-structured data.

**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 and so “free”.

**Cons**

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

**Decision:**

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

1. We have more experience (less time spent on 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.

**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 command 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 give so many other settings and configurations that can be set to the wrong value; and performance.

**Technologies for data access:**

When deciding on which data access technology to use we came to 3 most popular technologies on internet right now, for 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 most popular products, as they most likely are the best and also, they have great support for them), we started comparing them.

ADO.NET was one of our initial ideas and in the beginning, we thought of it as the main one, but 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 and easier to maintain than ado.net. After reviewing and comparing the last two options we had, we came to 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 technology.

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

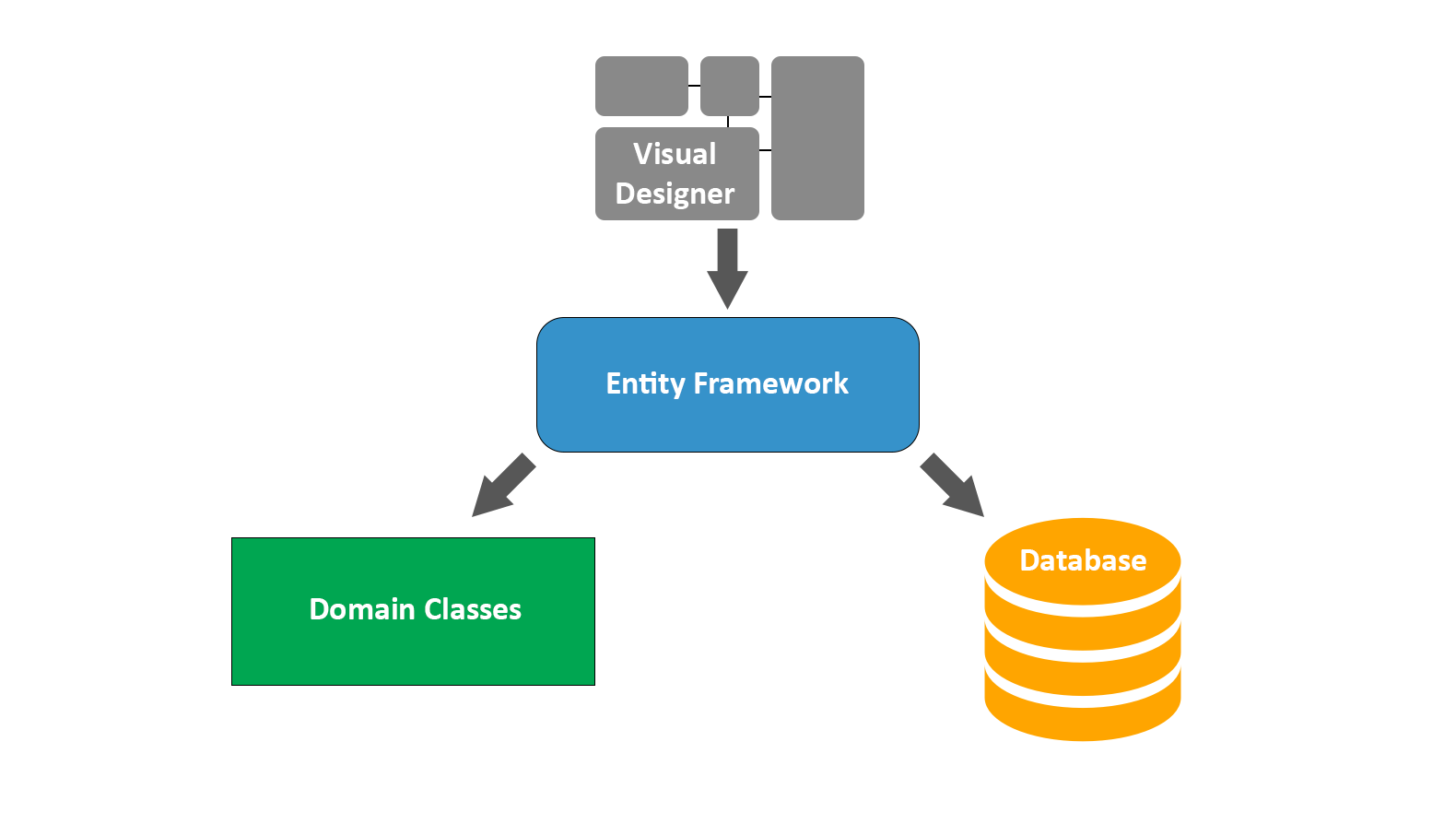


Figure (Visual diagram used by the Framework to autogenerate the Database SQL script and the Data Model source code files)

* **Model-First**

**Pros:**

1. Great when there is need to have visualization of database
2. Easy to use and understand for large data structures
3. Model 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**

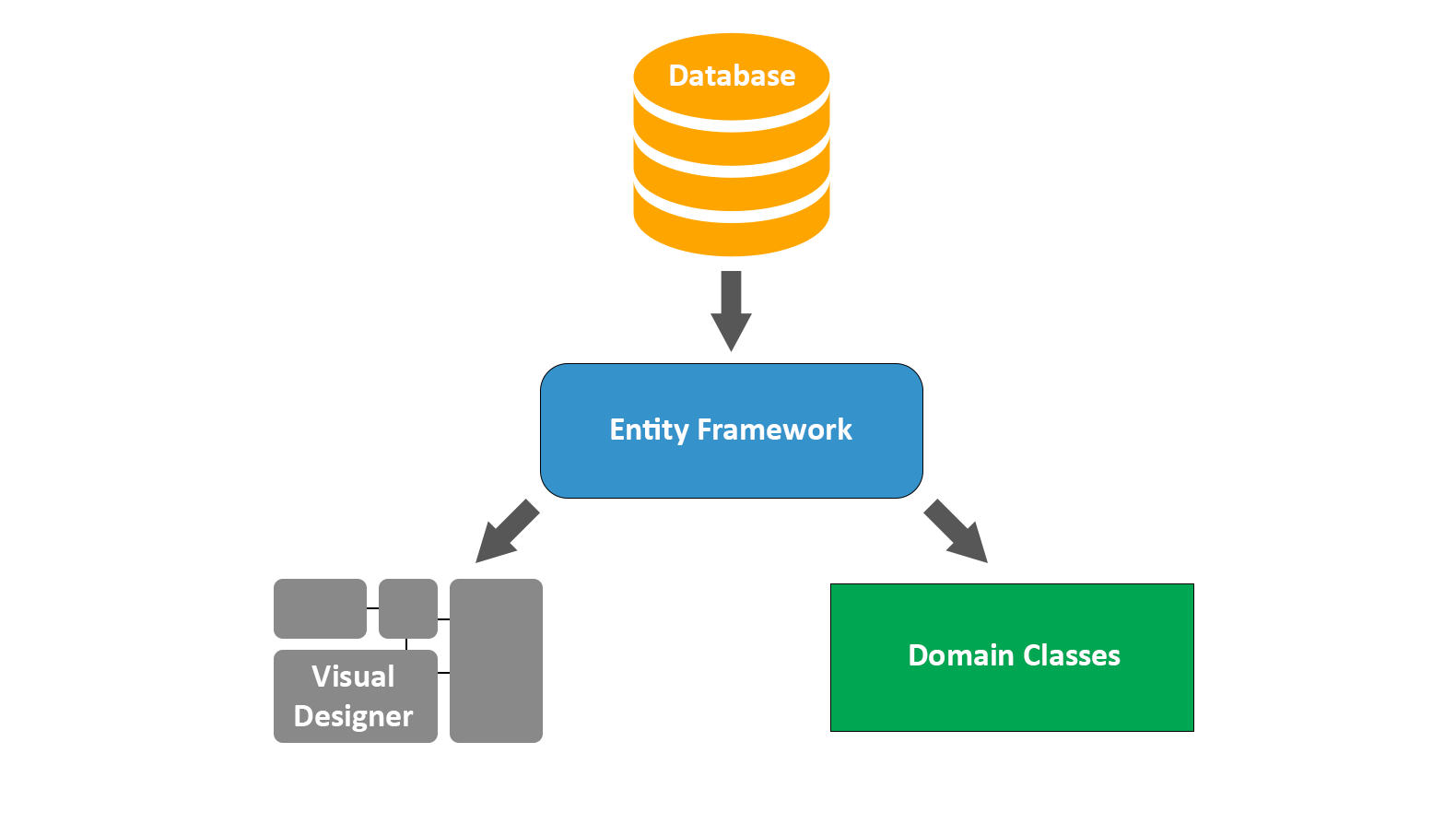


Figure (SQL script used to create the Database and from that corresponding Data Model and diagram is generated)

**Pros:**

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

**Cons:**

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

* **Code-First**

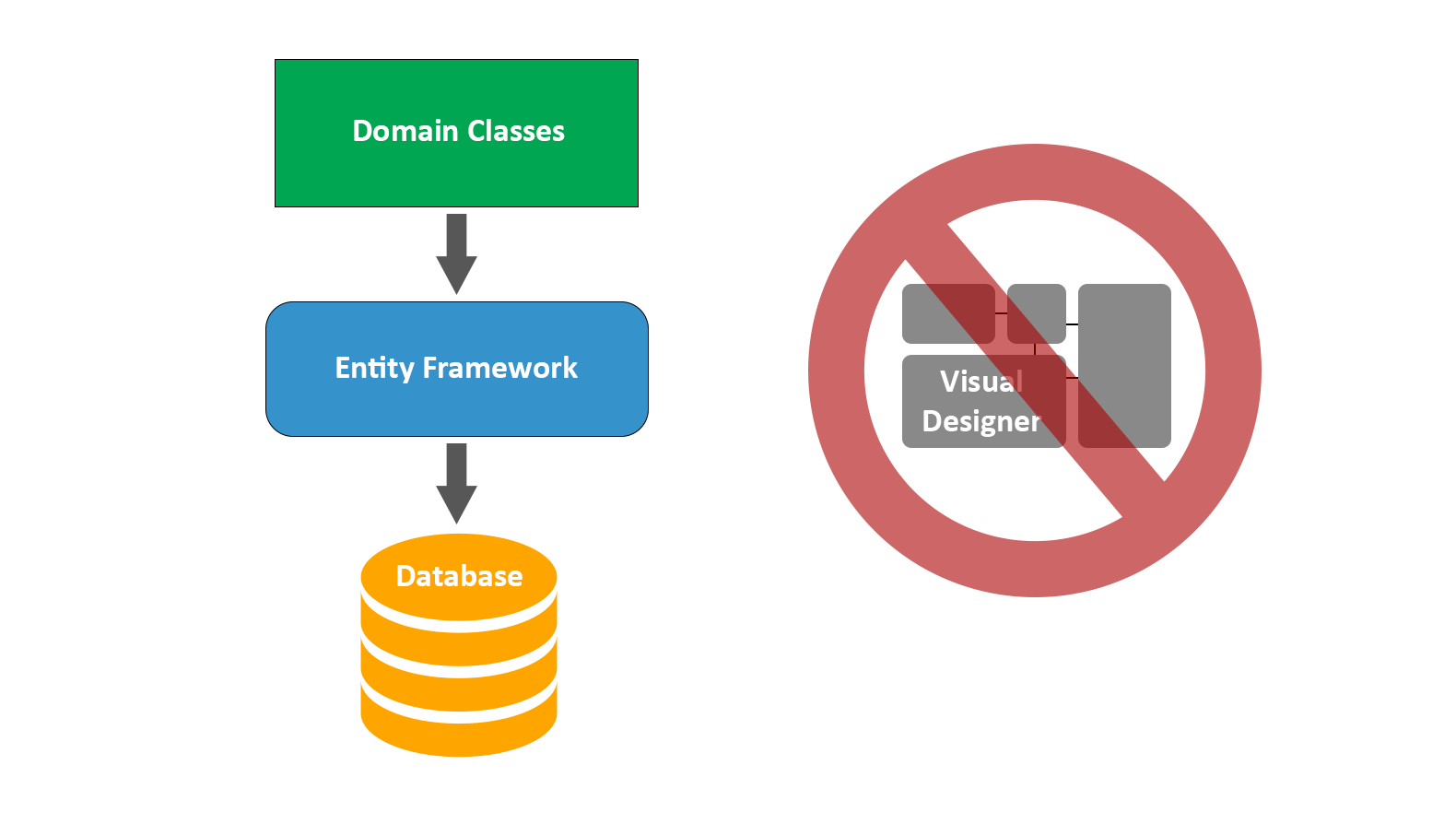


Figure (Model classes from project used in Entity Framework generate the database accordingly to it)

**Pros:**

1. No need for diagrams
2. Easy to maintain code 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 Code-First as we did not want to waste too much of our time on making database work, since we don’t have a lot of time to begin with. Besides already mentioned reason, this solution by many is considered the best for small-to-medium sized project, we wanted to gain more experience, have 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 or are insignificant, in case of data loss for example.

### Client

For client we reviewed multiple options, in the beginning thinking of creating dedicated client, but after communicating with company we got to know that their requirements for client to fulfill were:

1. Application should work on multiple different platforms (pc, phone…)
2. Shouldn’t be needed to be installed
3. Should be easily changed and updated

This forced use to make Web client, which would not only communicate with API, but would be also interactive, understandable and would allow for use of various languages.

## Decision:

Since we were new to web development, having only minor experience from 3rd semester project where we developed dedicated client and 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, as although they are easy to make, we wanted to learn something more than just dragging and dropping, besides web forms are being depreciated and other options are more relevant for what project must do. After comparing the options which were left we determined that MVC is the best one of them, as it is frequently being used in actual businesses, its Ideal for developing complex but lightweight applications and it was technology which we had the most knowledge in, so less time would be spent on learning new way of doing things, instead focusing on developing solution.

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

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

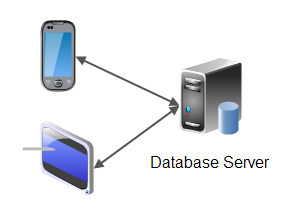
And languages associated to them:

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 client and allow him to do manage database (depicted in Figure 13).



Figure

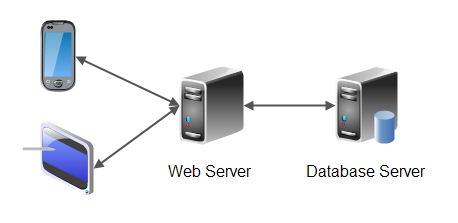
**Pros:**

* Easy to make
* Security (All of the work is done within costumer’s computer)
* Great performance

**Cons:**

* Requires costumer to get application
* If there is new update, client would be required to get the newest version (without updating client wouldn’t be able to experience newest features or his applications might not work)
* Client system and hardware dependent

**Client-Server application**- Simple client which deals with interacting with costumer and communicates to server, who makes changes to database (depicted in Figure 14).



Figure

**Pros:**

* If there is update to product all functionality is automatically accessible to client
* Easy to expand to more clients
* Not dependent on client’s hardware capabilities

**Cons:**

* Harder to develop than dedicated client
* Worse performance due to extra step (communicating with server)

After reviewing the pros and the cons, we determined that Client-Server application better suits our needs, as it would be easy to work with (for client), has possibility to be easily expandable to new types of clients and functionality updates automatically effect all clients not requiring them to get the newest release of application. Meanwhile the cons of having distributed application are rather minor, for example sacrificing a bit more time in development of application for easier expansion in future and minor performance decrease due to doing the business logic being located in dedicated server.

### Middleware

#### System-Database

#### Clients-System

## Implementation

### Security

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

**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 on with a user name and password you are authenticating.
* **Authorization** is the process of verifying that you have access to something. Gaining access to a resource because the permissions configured on it allow you access is authorization.

If a user wants to use our application, we decided to restrict his access and first must be must authenticated to enter. To authenticate, a user must fill in his or her login details (username and password).

Instead of saving the password as it was entered and expose our customers details, we decided to take a step forward 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 the same passwords are entered, they would still have different hash values, making them harder to crack. Doing this also ensures, that if someone manages to get values in 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 take very long time, allowing us to react if noticed.

Authorizations was ensured on the server side, making it more secure, it was done by checking if user is logged in, only if he was, he would be able to access the functionally of product, otherwise he would not be able to access any other page than login. In future we plan on implementing different access for different users, but this was not done right now, as we had limited time and none of the features implemented should be locked for ordinary user.

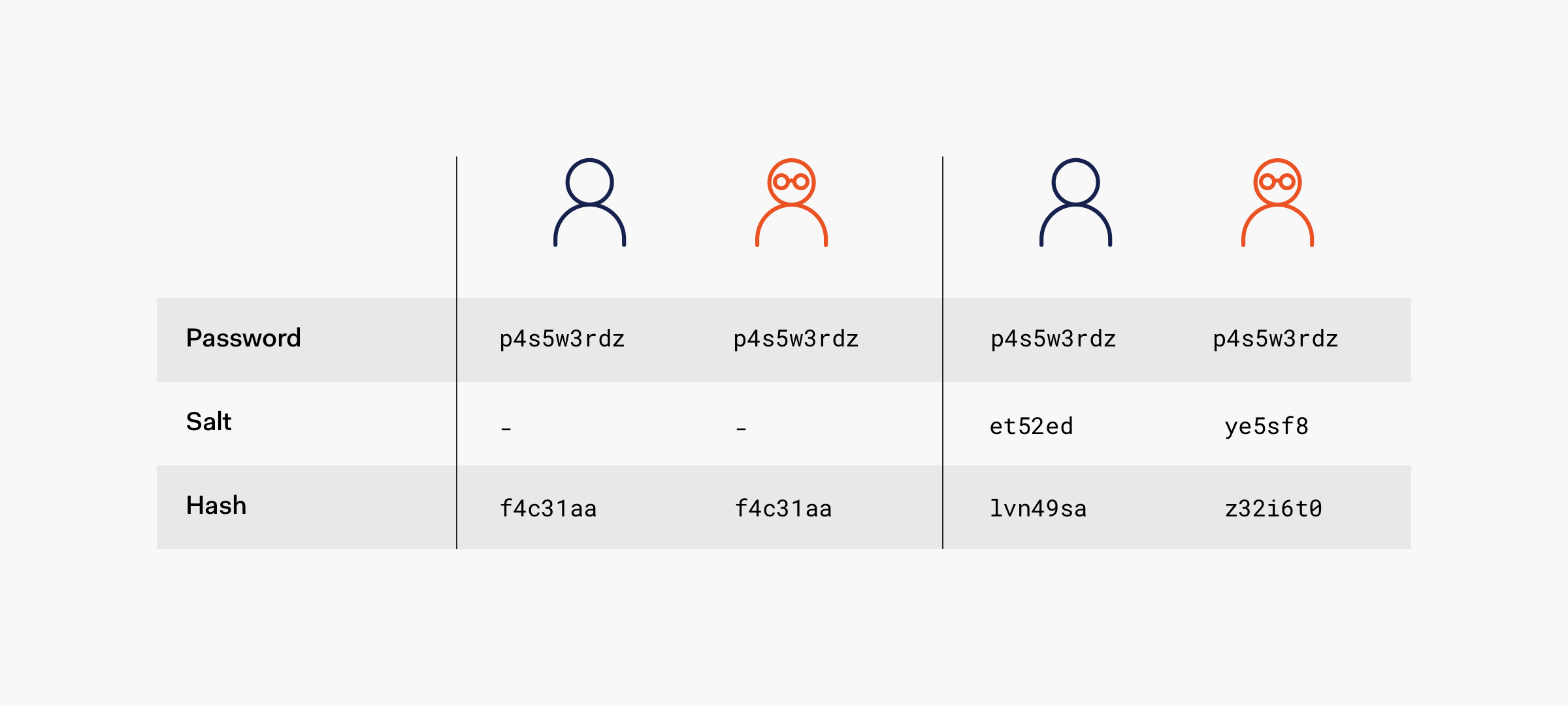


Figure (Data encrypted using SHA vs data encrypted using SHA + Salt)

**Resource transport:**

Http (Hypertext transfer protocol) is commonly used when two machines are communicating over internet, but it’s not secure and if the information would be received by person who should not have it, he would be able to easily read and use it for his own benefit. HTTP can easily be used 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 message sent over internet, we decided to encourage the company to use HTTPS, for all pages (having it only for login is not secure), which is an extension of the HTTP, but with secure communication integrated. In HTTPS, the [communication protocol](https://en.wikipedia.org/wiki/Communication_protocol) is [encrypted](https://en.wikipedia.org/wiki/Encrypted) using [Transport Layer Security](https://en.wikipedia.org/wiki/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.

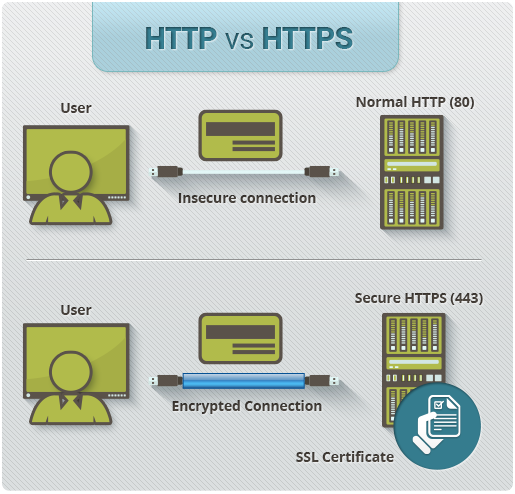


Figure (Http vs HTTPS)

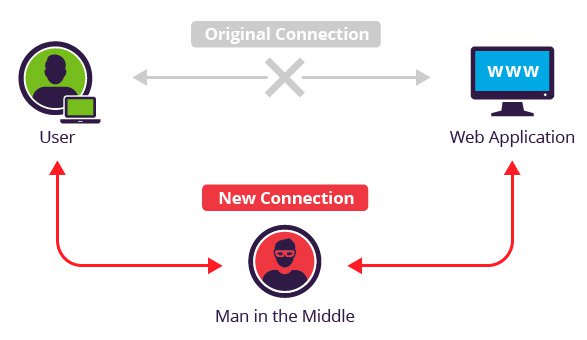


Figure (Representation of a man in middle attack)

**TLS or SSL:**

As said before there are 2 most common transport layer security protocols for communication over internet, SSL and TLS. SSL although being one of the most popular one now is considered obsolete, due to it having multiple vulnerabilities (e.g. [POODLE](https://www.globalsign.com/en/blog/poodle-vulnerability-in-ssl-30/), [DROWN](https://www.globalsign.com/en/blog/drown-attack-sslv2/)), but it is better performing 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 product using SSL, but when deploying we would advise the company to use TLS, due to greater security. Although it is unlikely that an attack could take place, because the company is relatively small and the attacker would have to be on the site to attack (as application would be hosted only in LAN), it is still possible, that it can happen and making it as hard as possible to do what he wants could discourage him from attacking, if the possibility of being caught wouldn’t make him reconsider.

#### 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 (Used in our solution indicated by use of bold):

* **Cross Site Scripting Protection (X-XSS)-** stop loading the page when a cross-site scripting attack is detected.
* Content Security Policy (CSP)- helps to prevent code injection attacks like cross-site scripting and clickjacking, by telling the browser which dynamic resources that can load.
* **Browser Sniffing Protection (X-Content-Type-Options)**- makes it harder for hackers to guess the right mime type, by inspecting the content.
* **Clickjacking Prevention (X-Frame-Options)**- so hackers don't iframe your site, to trick you into clicking links which you never intended to.
* **HTTP Strict Transport Security (HSTS)**- prevents any communication happening over HTTP.
* 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](https://developer.mozilla.org/en-US/docs/Glossary/MITM) attacks with forged certificates.
* **Referring Settings (Referrer-Policy)-**  prevents linked website, to be able to see where the users are coming from.

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.

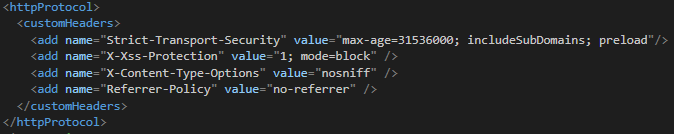


Figure (Implementation of security headers in web.config)

**Headers implemented:**

In Figure 18, you can see our implementation of the headers, one of the first things that can be noticed is that Clickjacking Prevention is not present here, the reason for that is, if you are using ASP.NET MVC 5 or newer (which we are), this header is added automatically as seen in Figure 19, when looking at the headers assigned to a network element.

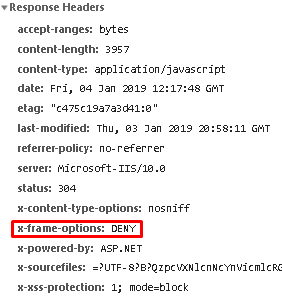


Figure (Response headers, as seen by user's browser)

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 user experience.

Browser Sniffing Protection was added, as although this action 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 user privacy is actual topic nowadays and not disabling it could pose problems, as attacker could extract sensitive information in your URLs, which you don't want to forward to other domains.

**Headers not implemented:**

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 much more.

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.

**Implementation explained:**

Now we are going to look deeper in one of the implemented headers, to explore what was done there and why, to do it, 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 the site, so user will automatically call 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 be communicated with HTTP not HTTPs. We decided to set to value of 31536000, which is one year, 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 attacker to intercept and tamper with redirects when user first communicates with site, as they take place over HTTP.

#### Tokens

Part of the security measure we’ve taken against any 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, that 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 20), only after a successful authentication is done, using 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.



Figure (Token generator usage)

### 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 to re-run the web api server, every time a new test/ new code implementation would be made. 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 IISExpress, every time a test would be scheduled for running. This solution proved to be extra helpful as it allowed for our Continuous Integration solution (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 are running.

For Integration tests, we have decided to create a base class for all of the tests, that would instantiate an InternalApiFakeServer class, which in a relatively short method (see figure 21), 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 other objects that were used, only to start everything again, for every single test that is scheduled to run. This was done in order to ensure that each test gets a fresh start and to make sure that they are independent from one another.

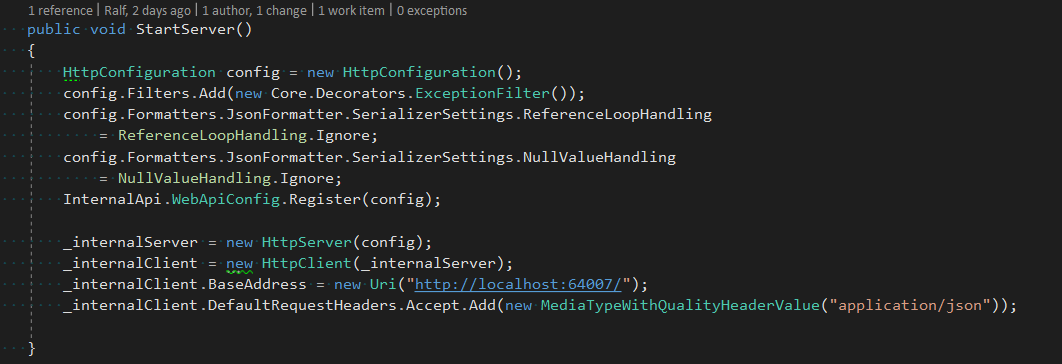
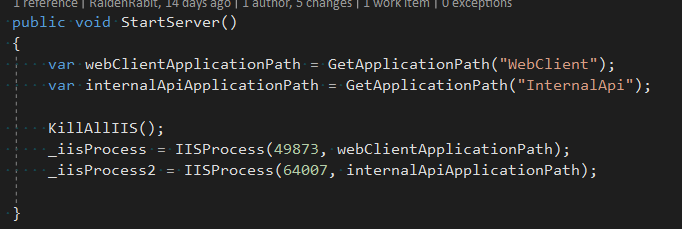
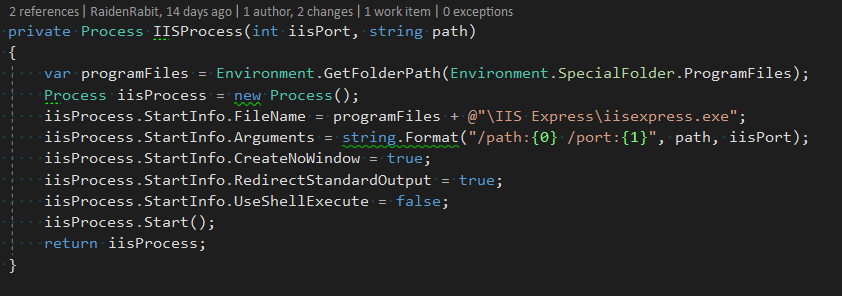


Figure (Internal Api fake server implementation)

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(see figure 22), 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 IISExpress processes (see figure 23), 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 IISExpress processes are saved, in order to be killed, once the acceptance test is finished. And the entire process starts again, once a new Acceptance Test is scheduled to run, again to make sure that they are independent from one another.

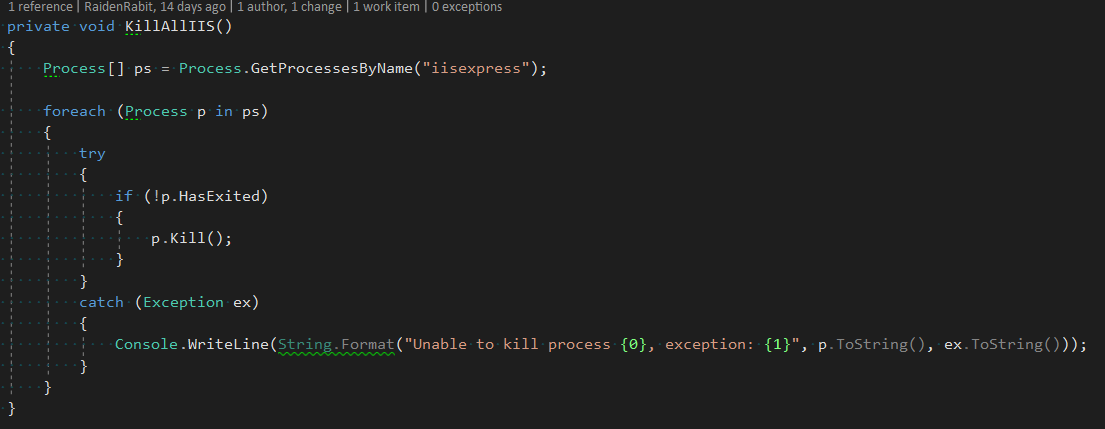


Figure



Figure

But before any of this starts, a KillAlliis command is issued(see figure 24), in order to make sure that no remnant IISExpress process, were accidentally left running (ex: when manually stopping a debugging session of an acceptance test).



Figure

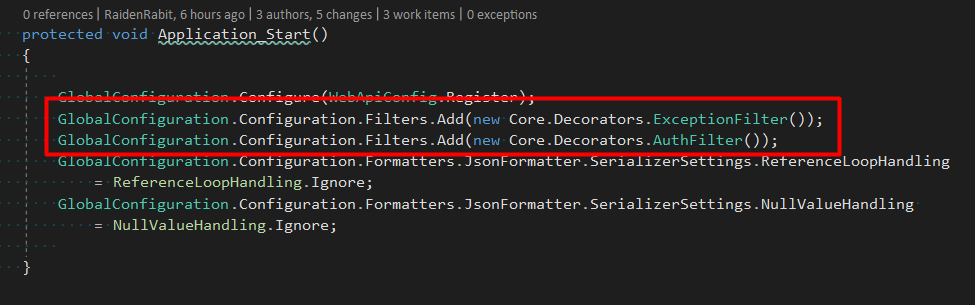
### Decorators

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 the high cohesion, low coupling, and no code duplication standards, that we established to follow.

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

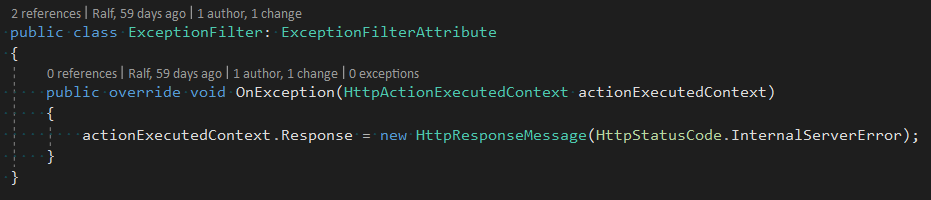
* Authentication and Authorization
* Exception handling
* Action and error Logging

The filters are added to the Internal Api, through a couple of simple lines, in the configuration files (see figure 25)



Figure

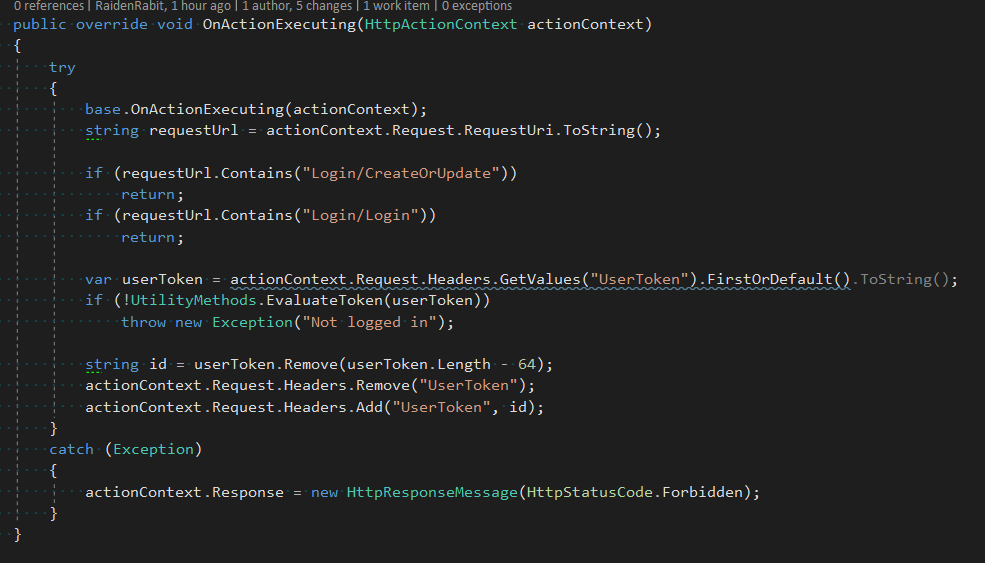
The exception filter (figure 26 for implemantion), is a simple and small one, as its only functionality is to return an “Internal Server Error” response to the client, in case an unhandled exception occurs in the system, or in case an error that cannot be fixed by the user, by simply slightly changing their request parameters. As can be seen, when the “OnException” method is fired, an “internalServerError” response message, is attributed to the actionExecutedContext.



Figure

Similar to the Exception Filter, the Authentication and Authorization filter (figure 27), uses the actionContext, among other things, to do it’s job.

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 (Register and Login) are not hindered by the lack of a UserToken.



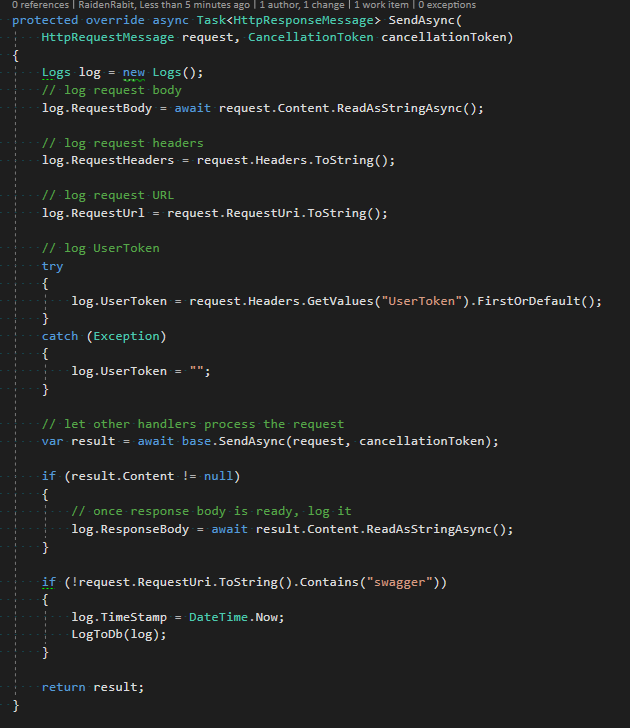
Figure

After making sure that the Request is not intended for registering a new user or authenticating one, 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 accesing information to which they do not have access.

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 specifc table, in the database.

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



Figure

As can be seen in Figure 28, the handler makes sure to not alter the request, but simply act as an observer. We also decided to exclude all actions done via the swagger documentation from being logged, to not pollute the database.

### 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.

Figure 29, shows how the interactive and intuitive UI elements, help the user to easily change the website’s language.

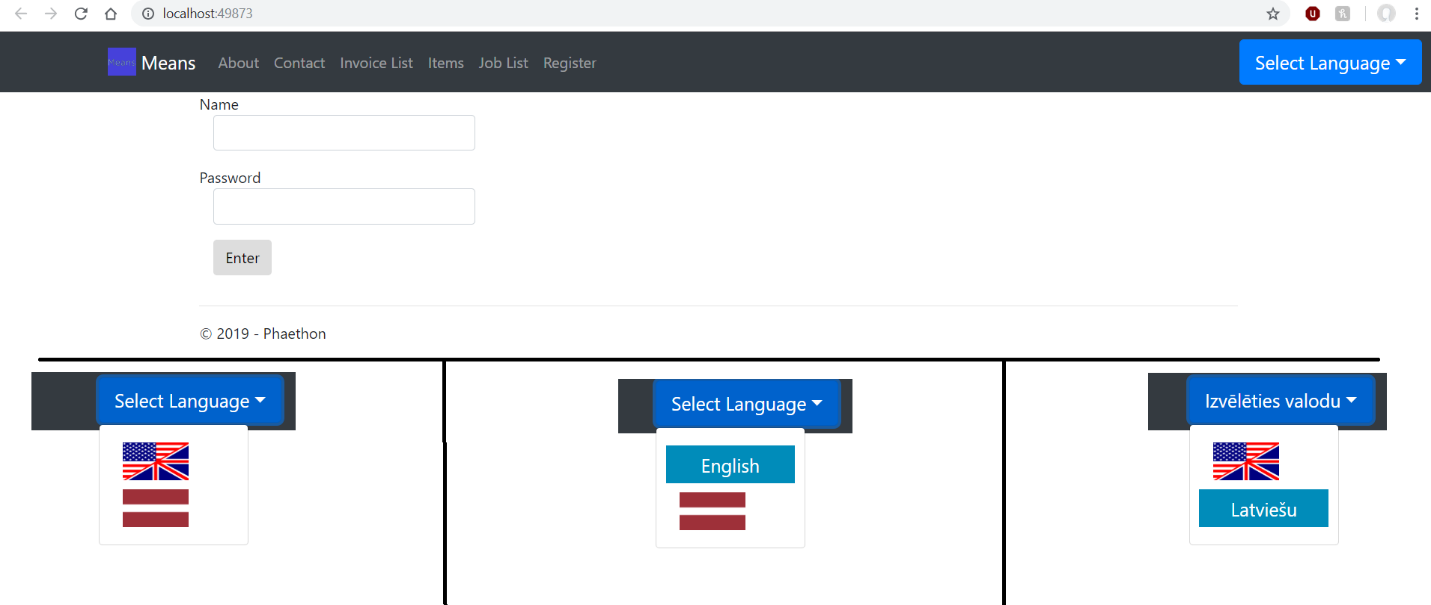
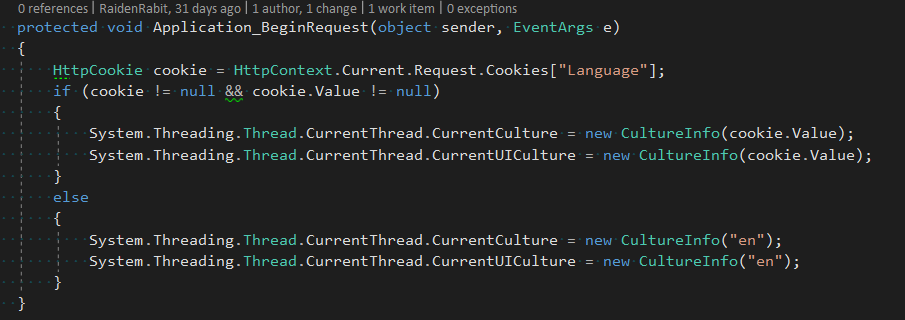
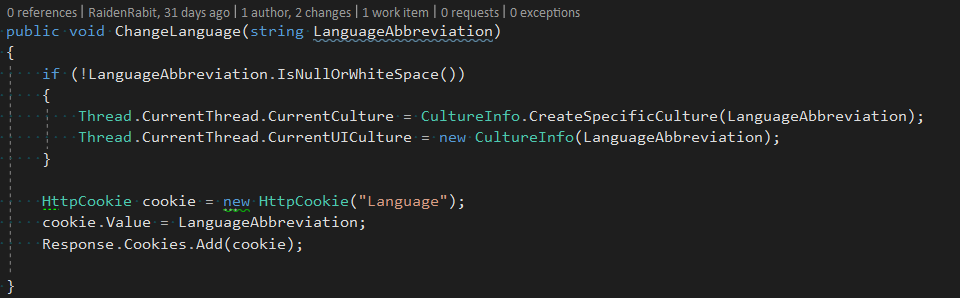
This feature was implemented with the help of a user-sided element, that saves the language code, for the language the user selected.

Figure (Changing language, UI)

Figure 30, 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 31) and depending 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.



Figure



Figure

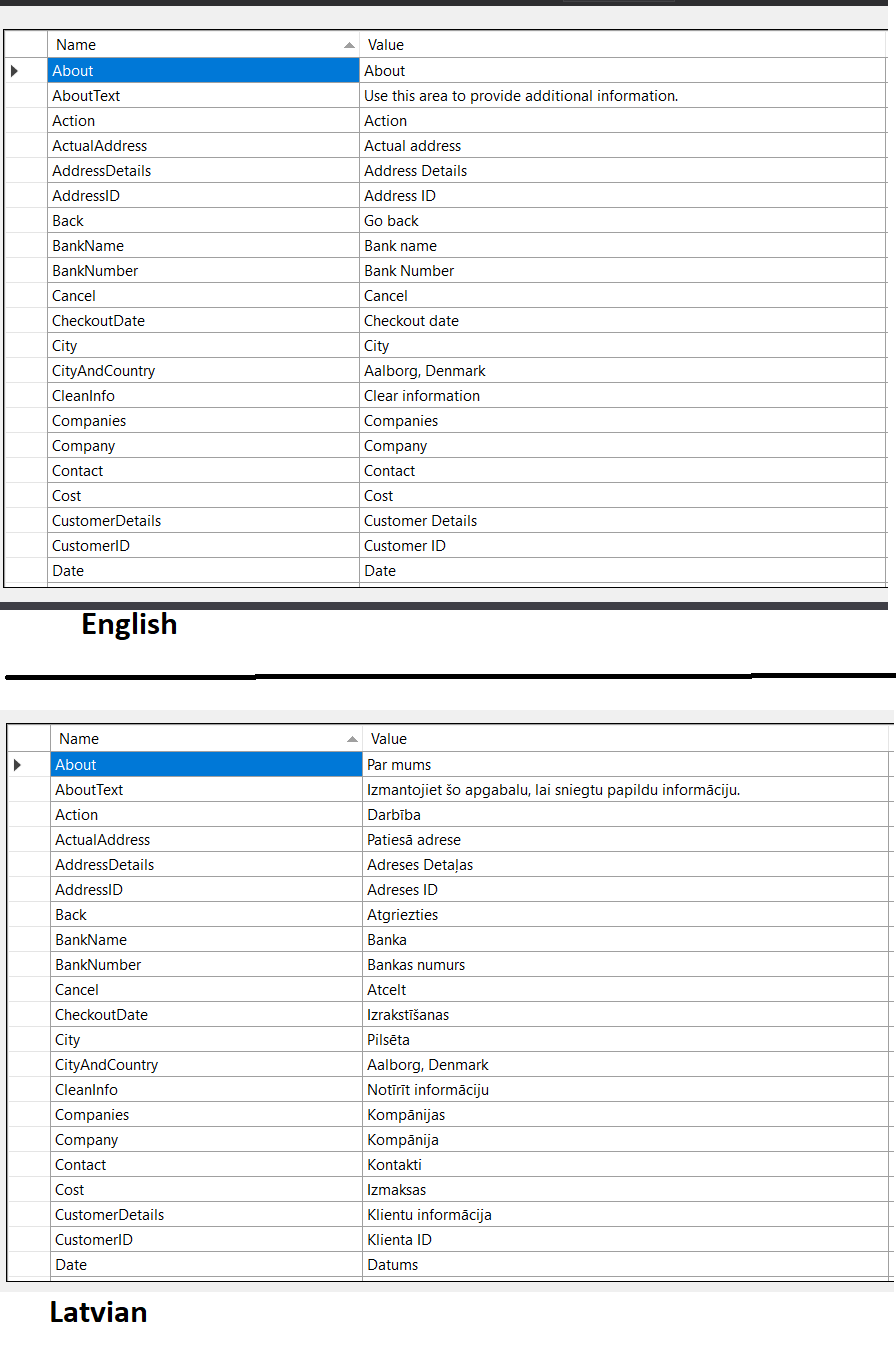


Figure (Language packs)

As briefly hinted before, all the text presented on this web client, is held inside language packs (see figure 32). Meer text documents, that hold both an ID and a Text Value, that is displayed on screen.

Figure 33, depicts how the Language Pack and 34 shows how its imported in every view, and how the values are called upon, in the right places.

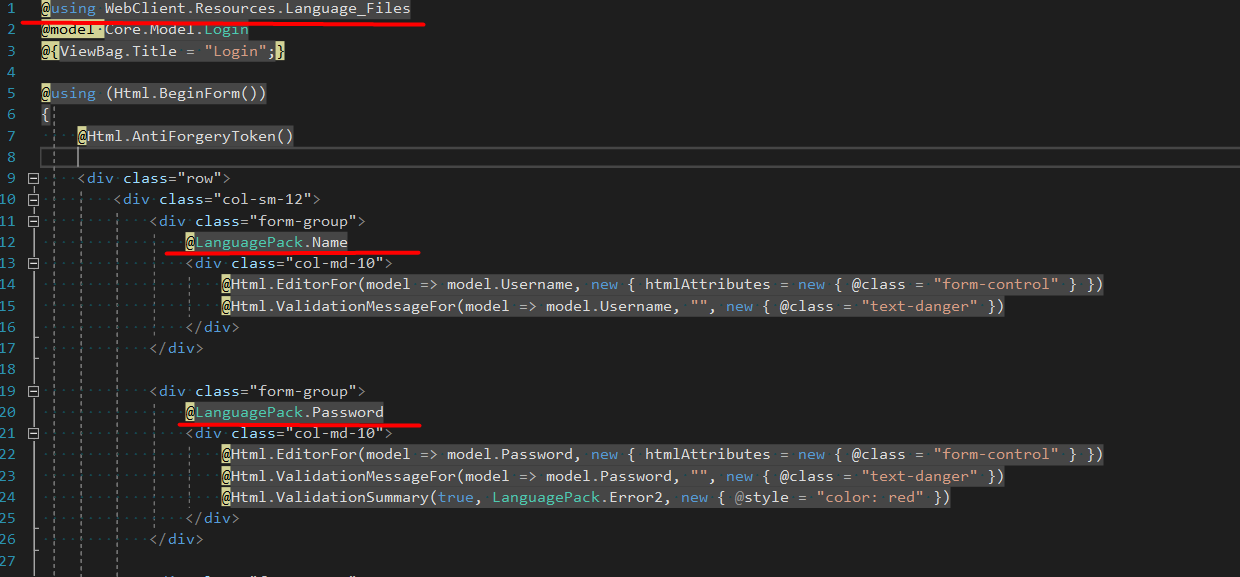


Figure Showing how language pack is implemented in code

## Tests

Testing is an integral part of developing an application. It can determine if the logic behind the software’s designer is correct, it cannot however identify all the faults with the software. Testing in its simplest form means running an application with the intention of finding bugs and errors that the user might encounter, this can be done as soon as an executable application is present. The number of test 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:

1. Unit tests;

2. Integration tests;

3. 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 quality of our product.

### Unit Tests

Unit tests are our 1 st 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.

### Integration Tests

Our integration tests focus on testing entire backend use cases, 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 (WebAPI), in order to correctly test out all of the backend functionality, including WebApi 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 WebApi, 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.

### 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, sitting in front of a computer, actions. This is achieved with the help 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);

Support for running tests on multiple browsers;

Large and open community (in case we need assistance);

Extended documentation (in case we needed assistance);

Open Source (free to use );

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.

## Encountered Difficulties

Insert EF jokes

# Company Feedback

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 goals 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.

# Conclusion

Insert how it was a very smart thing to add … and how we underestimated the size of the project in sprint zero

# References

Andrei please add your URLs  
Ralfs, I can add yours if you want but you can also add them

Adam add your URLs