Tampere University **Unit of Computing Sciences**
TIEA4 Project Work (City centre campus)
TIETS19 Software Project Management Practice (City centre campus)

Ryhmä 8 - Pallas
Snowledge Digitalized

Final Report

88018, Sakari Eskelinen
432222, Kaisa Kauhanen
424492, Arttu Lakkala
428786, Aarre Leinonen
93003, Markku Nirkkonen
K382694, Johanna Pekki

Version history

Version	Date	Author	Description
0.1	12.1.2021	Team / Sakari Eskelinen	First version of Final Report
1.0	26.1.2021	Sakari Eskelinen	Final review version, added statistics and minor updates

Contents

1 Introduction

1.1 Purpose of the report

Purpose of this document is to provide an all-around picture of the project, its constraints, risks and results. Based upon the original project plan which was modified alongside the project progress, this is implicitly the story of outcome and report about the experience of completing the Pallas project, or application *Snowledge*, as we had it called by its workname.

1.2 Product and environment

Project goal is to create an online application – a web page, in essence – that can provide information about the snow situation around Pallastunturi area. The software is made for ski school Pallaksen Pöllöt and their target audience will be mainly the ski school customers and teachers – but also other casual visitors on the zone could use it for their benefit during their presumed stay in Lapland hotels.

Functionality is implemented as a website, which contains a map populated by landscape dividing segments, of which the snow status information can be viewed, as long as a local operator has updated the status after having actually ventured into the wild to see how it is at the moment. Data can be accessed from any web browsing capable platform without any credentials like any other web page, however, adding snow data requires credentials to log in. In the current version there are two roles, admin and operator, with the difference that an admin can also administrate segments and users.

The saved data shows up on segments by a highlight colour on the map, and also by an icon and text if the segment is actually selected. In addition, the site provides concurrent weather data of temperature and wind from an on-location sensor, and there is also avalanche warning informed about on a specific region of Pallastunturi.

The project source code can be found in GitHub:
https://github.com/Zachax/TUNI-Lumitietous-sovellusprojekti

The test server version exists still at the beginning of 2021:
http://itc-pallas.rd.tuni.fi/

Final production address of the application will be found here:
http://lumitiedot.pallaksenpollot.com/

1.3 Definitions, abbreviations and acronyms

Admin: Person who updates website with admin privileges End-users: Admins and users.

API: Application programming interface

Back end: Data access layer of a software that runs on the background of an application (mostly invisible to a user).

CSS: Cascading Style Sheets, the general style sheet language used for setting external looks and layouts of a web page.

Developer: Team member who is developing the software.

Front end: Presentation layer of a software which contains UI (mostly visible to a user).

HTML: Hypertext Markup Language, the standard markup language for creating web page frame bases.

JS: JavaScript, a programming language used mainly on web page functionalities.

MVP: Minimum Viable Product – a state of a product (eg. software) which has been assembled into usable condition with minimal features and refining. Comparable to a functional prototype.

Node.js: JavaScript runtime environment which allows running JavaScript code outside conventional web browser container, enabling the web based scripting language to work as more traditional back end suitable programming language.

Open-source: A type of a computer software in which the source code is released to public and its copyright license allows unrestricted use for any purpose.

React: An open-source JavaScript library for front end building.

Source code: Raw original format of a computer program in human readable code of programming language.

MMT: Metrics monitoring tool provided by university for project management.

UI: User interface

User: Persons who use website to get information about snow situation.

2 Project organisation

2.1 Group

Sakari Eskelinen, sakari.t.eskelinen@tuni.fi 0456755873. First timer as a project manager. Previously acted as a developer with few programming languages such as Java, Javascript and C# (Unity). Studying part-time, alongside a “80% full-time" job in IT field; daytimes are mostly for work, however, working hours are flexible.

Kaisa Kauhanen, kaisa.kauhanen@tuni.fi, 0440982507. Experiences in UI design, programming and quality assurance. Flexible working hours.

Arttu Lakkala, arttu.lakkala@tuni.fi 0443065061. Main experience in programming with multiple languages. Also, some limited experience in data visualisation.

Aarre Leinonen, aarre.leinonen@tuni.fi, 0505624033. Main experiences in programming, mainly Java, little experience in Python and web development. Flexible working hours.

Markku Nirkkonen, markku.nirkkonen@tuni.fi, 0402171939. Main experiences in programming with mainly Java. Some experience of web development as well. Also very interested in UI/UX design. Working hours have to be fitted to other studies, but mainly flexible.

Johanna Pekki, johanna.m.pekki@tuni.fi, 0442544525. Main experiences in information processing systems of organisations and planning.

2.2 Customer

Juuso Holstein, juuso.holstein@laplandhotels.com, representative of Pallaksen Pöllöt
Arto Hämäläinen, arto.hamalainen@harte.fi, representative of Pallaksen Pöllöt

2.3 Other stakeholders

Timo Poranen, timo.poranen@tuni.fi, project course teacher responsible
Pekka Mäkiaho, pekka.makiaho@tuni.fi, project group’s instructor

2.4 Related organizations

University IT department provided a test server for developing.

End-users will be members of Pallaksen Pöllöt, and everyone in need of information about snow attributes around Pallastunturi area.

Production server is rented by a 3rd party service provider by the customer organization and application itself requires license fees from Google for using their Maps API.

3 Project implementation

3.1 Communication

The project team held an online Teams meeting once a week separately but usually the same day (Wednesday) both with the customer and among themselves. Exception to this was during couple weeks of Christmas holidays in late December. In addition to weekly meetings, the team had separate workshops for collaborate remote working and other extra meetings when needed. For workshops and other extra meetings it was not always expected that every member would join them.

Otherwise, the main everyday communication medium has been Slack, in which we had our main bulk of textual communication. Traditional emails were obviously also used especially if sharing non-vocal information with the customer or stakeholders. Phone/Whatsapp connections were available for backup, but those were in practice used mainly few times in the beginning in order to confirm gathering up before other communication means were properly established.

The team never met as a whole on location during the Autumn 2020, due to ongoing COVID-19 pandemic. Majority of the team managed to schedule physical meetings at the university early in the Autumn before unnecessary contacts were once again urged to be subdued.

3.2 Tools and technologies

The following tools and technologies were utilized and examined during the project:

- Tools used for communications and sharing data: Teams, Slack, Sharepoint, Trello, email and phone (WhatsApp).
- Project management tools were MMT (university's hour signing and statistics application) and Trello. Trello is a platform for Kanban-style task listing.
- Version control and source code is saved in GitHub.
- Primary Development tool was agreed to be Visual Studio Code, however, since most of the code was written individually, other code editors might have been applied to.
- Webpage was originally intended as simply a stack of HTML-CSS-JavaScript on front-end, however React eventually replaced the raw JS and part of CSS.
- Google Maps is used as the map base.
- Interface design was started with InVision program. However, InVision was deemed ill suitable for proper a UI design, yet it worked fine as a shared drawable planning site. The actual UI mockup was eventually made with JustInMind instead. Also Indesign was used while designing.
- Node.js is used for backend application.
- The application uses a database powered up by MySQL.
- Apache web server was tested, however, it was soon deemed unnecessary overhead as the Node.js could provide all backend needed.
- Vagrant was used for creating virtual machines for providing development environment while creating new or experimental features. This ensured that it was possible to keep most of the time a functional test (or QA, Quality Assurance) environment representing the latest mostly stable version shared by the whole team and demoed while needed.
- Ubuntu Linux was used on the server as operating system.

3.3 Sprints and methodology

The project was implemented by an applied free-form agile practices that were influenced by Scrum method. This means we had our 5 sprints (or iterations, as they were commonly also

called) with backlog planned for each. Sprint progress was monitored by frequent messaging and regular meetings, during which MML statistics were monitored and more importantly Trello based Kanban board & backlog was reviewed in order to get the larger picture about how the project has proceeded, what are the team members doing right now and how is it to progress further.

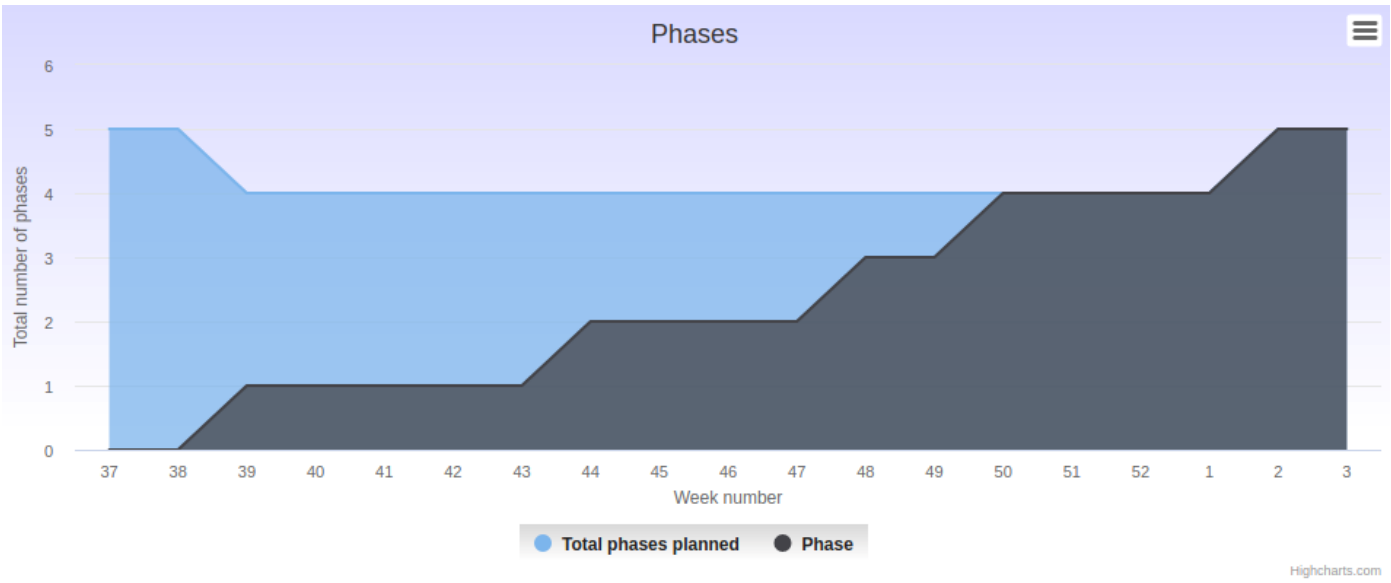
Intention and for most part the actual practice as well of the team was to flexibly communicate and adapt to the situation and requests by the customers instead of strictly sticking to the plan or any specific method. By point of view of the overall management, it could be argued that the team succeeded fairly well to keep knowledge of the current project situation shared not only by the team but also the customer by regular communication. Main points were shared in meetings, with further details discussed when needed and otherwise noted in documentation.

Sprint scheduling was based upon agreed review meetings with the group instructor, to which the whole group and the customer representatives participated. In addition of 5 original sprints (with numbering started from 0, as a slight jesting gesture to common programming language indexing practice) it ended up being necessary to have a slightly extended final 6th sprint, since despite the application itself being approximately finished and even transferred to the production environment by the final deadline of 13th January 2021, documentation and reporting for the project were not yet sufficiently completed by then.

The project schedule is below:

Project schedule (0-4 original plan, additional 5 for final execution):

Sprint no	Schedule (start/end date) High level content.	Deadline
0	Hello World, project plan	17.9.2020
1	Basic functionality, UI mock-up, process plan	28.10.2020
2	Core functionality, Usable UI	23.11.2020
3	Additional functionality and all features, Improved UI based on feedback	9.12.2020
4	Final bug fixes, final usability updates	13.1.2021
5	Extended finalization, delivering the missing product features and documentation.	29.1.2021



3.4 Deliverables and outcomes

Like mentioned in Chapter 1, source code with additional documentation, commenting and instructions can be found from GitHub: https://github.com/Zachax/TUNI-Lumitietous-sovelluspro-jekti
Implementation of the web page can be found from production customer address: http://lumitiedot.pallaksenpollot.com/

Other “external” (ie. shareable outside the team) documentation produced: Final report (this documentation), Test report, Test plan, Database plan, Process flowchart, Use case scenarios, Schedule plan chart and Workflow chart.

In addition, the team made internal documentation of meeting notes, hourly bookkeeping (including extra spreadsheet for calculating in further detail how much is needed weekly) and, obviously, the Trello Kanban board backlog. Both of which were available for use in case needed, and there were cases where they proved out useful during some backtracking.

The final application consists of approximately:

- Up to 6000 lines of comment equipped code...
- ...of which around 4500 lines (~75%) is some form of JavaScript.
- ...several hundred lines (~10%) of HTML and CSS.
- ...couple hundred lines (<5%) of database SQL creation lines and data inputs.
- The application has two main views with a handful of adaptable sections for modifiable data.
- The whole project source and main documentation is divided into approximately 100 separate files, of which actual code is estimated more than two thirds.
- Since JavaScript is not a traditionally classed programming language, separate files could be called close comparables.
- Over 110 functional methods within the code.
- Estimated total amount of self-coded lines: 60%.
- Estimated total amount of partially reused code lines: 35%.
- Estimated total amount of totally reused code lines: 5%.

3.5 Restrictions and limitations

Due to time constrains several features were left out, mainly due to time constrains:

- Proper usability testing with a pilot group.
- Feature to add new sources of weather data.
- Possibility to do reporting or forecasts based upon historical data.
- Possibility to upload pictures from a segment in the nature.
- System logging for maintenance and possible debugging.
- Background map application was first thought to be interchangeable but building such proved out too hard in the given time span. In order to change the used map application, the whole front end should be largely rewritten. However, back end could be recycled if necessary.
- 3D view of the landscape as an alternate to the normal 2D map.

Also certain ways of implementation were noted that are not optimal:

- There is no implemented login persistency (eg. cookie to keep session). Therefore, the logged user is logged out if the page is refreshed.
- Segment management was left quite rudimentary; original thought for usability would have been to be able to modify segments directly on the map, now it only works by adding numeric coordinates.
- User administration was toned down: only two roles (admin and operator), with admins cannot be deleted in order to avoid accidental lockup of the system.
- Historical data of segment snow situation entries is being saved into the database, however, there is no implementation of actually viewing this historical data in the UI.
- Weather data is not added to entries by any automation. We were unable to implement a good way to get historical weather data by specific time scope through the API, and direct copying of the weather situation during the data input was deemed a bad idea, since the data would not necessarily be added concurrently according to the actual sighting.

3.6 Third party components, licenses and IPRs

Most of the third-party components/material used are either open-source systems or public domain data, such as weather data through Finnish Meteorological Institute's API and some modified React UI samples. Various more or less generic code examples or snippets were gathered up here and there into a mesh, where they are practically indistinguishable from other code. Therefore, even if the source code in GitHub is not 100% original (like in practice no code is), there should be no Intellectual Property Rights conflicts or infrictions in utilization and publishing as they are now.

Map of the application is provided by Google Maps, for which the customer had to order a paid license. Testing the application was done by trial time to the API. Also, production server is provided by a paid 3rd party server provider.

4 Working hours

Expected working hours per developer was 115 hours and 160 hours for the project manager. All but one members of the team essentially reached at least the minimum required the working hours by the end of the project. Since the numbers are taken out before the final review, it is likely that some members have forgotten to register some of their last hours.

Realized working hours in person-hours by person and task by 26.1.2021.

	Sakari	Aarre	Arttu	Kaisa	Markku	Johanna	Total
Documentation	40,5	3,25	6,75	10,25	3	1,5	65,25
Requirements	33,25	0,5	0	0	0	0	33,75
Design	9	11,5	7	12,5	15	5,75	60,75
Implementation	3,5	34,5	45,75	12	62,25	0	158
Testing	4,5	3	0	15,75	1,2	0	24,75
Meetings	39	33,75	37,75	42,5	34,75	29,5	217,25
Studying	19,5	21	11,25	21	14,25	9	96
Other	2,5	1	1	0	0	0	4,5
Lectures	9,25	9,25	4,5	9,25	7,25	8,25	47,75
Total	161	117,25	114	123,25	138	54	708

Realized working hours (weekly totals / project).

	Weekly total
Week 36	26
Week 37	26
Week 38	42
Week 39	33
Week 40	41
Week 41	35
Week 42	25
Week 43	25
Week 44	38
Week 45	38
Week 46	36
Week 47	34
Week 48	38
Week 49	32
Week 50	50
Week 51	34
Week 52	9
Week 53	Failed report
Week 1	42
Week 2	63

	Weekly total
Week 3	23
Week 4	5 (incomplete)
Total	708

5 Quality assurance

5.1 General description of testing

Testing of the application was largely proceeded by do and use circulation, so that when somebody had implemented something, somebody else would test it. We had continuously a test server running, which was updated after new versions came available, and in addition it was naturally possible for developers to have their own Git branches running locally before merging back to the main. We had repeated demo sessions both internally and to the customer, so whenever the weekly progress had brought some new feature, we could immediately test it and demonstrate it.

In addition, we made a more systematic test plan in form of a checklist for features, to which we could sign up if they already (or still!) worked for the ending iteration round. There was also a thought of arranging a more thorough pilot group testing by the end user side, unfortunately this plan was scrapped due to lack of time in the end. Changes and progress were taken into notes by internal memo, Trello card notes and sometimes also in code remarks.

5.2 Bug reporting

If bugs were detected and it was not immediately fixed, a practice was to open a new Trello card about it and toss into backlog. Most of these cases got fixed up in passing, and some cases even ended up getting fixed without a clear cause determined. Overall number of obvious bugs was most of the time fairly low – obviously some things did not work as expected while still clearly unfinished, so here it is referred to when a new feature was actually implemented.

There is not exact count upon how many bugs were actually found and squashed. In our Trello board there are only 3 dedicated bug cards. For example, at least one of them was quite severe yet not something one would detect instantly if simply testing the software: all the map segments would vanish if the server was left running overnight. This was due to database connection timing out if being unused for many hours.

For what is remaining, there are few known features that could be classified as bugs if the point of view is adjusted accordingly, but also which are essentially not going to get fixed, such as: attempting to remove admin user does nothing (error note was not implemented – that admins can’t be deleted is actually intentional), refreshing browser logs the user out and there are some other actions which probably should notify with an error notification, but it is just not implemented.

Of course, the odds are that there would still be some other bugs that we are simply unaware of. It is very rare for any software to be completely bug free, but nothing too severe is there for what we have found about.

5.3 Conclusions on product’s quality

Since we were regularly showing the current state of the application to the customer (and it was also available for testing on their own at later phases), we could also get feedback about whether some implementation was good, acceptable or lacking from their point of view. We found it unfortunate that we were unable to deliver all the features originally intended, however, we also got this understanding that the end result would also be at least good enough.

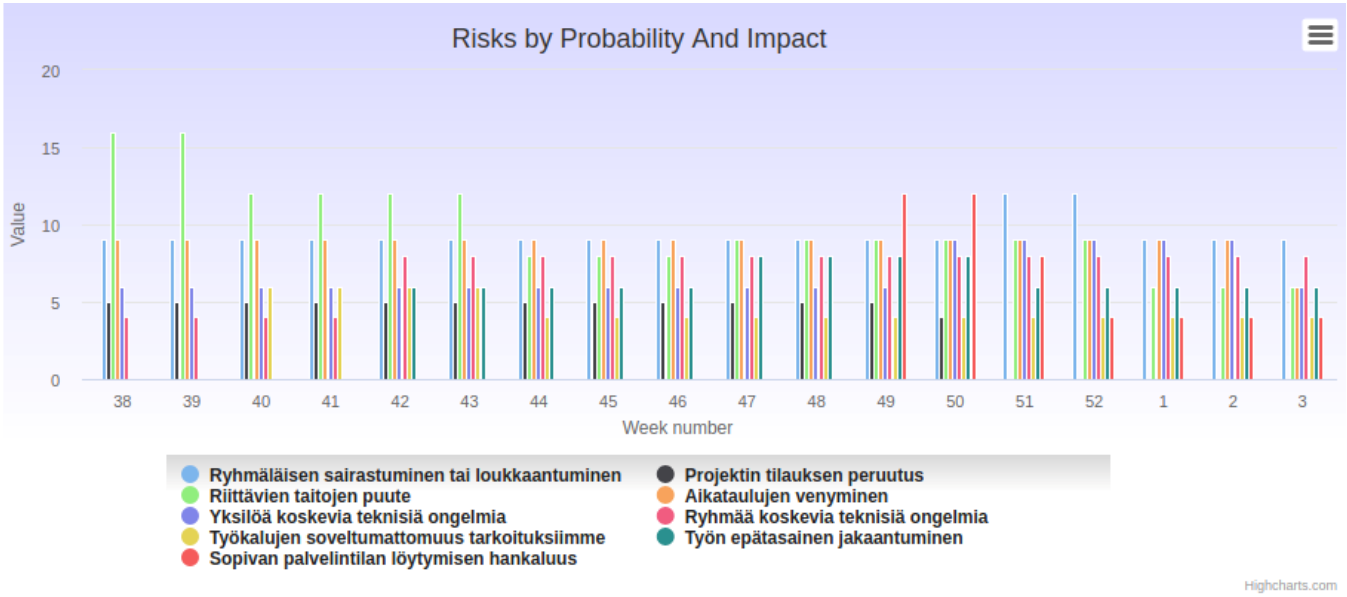
There are few quirks left in the design (and hence it can be debatable if those count as bugs or not), which might end up causing issues while using the application, if attempting to do some things, but this is to be mitigated by documented instructions. Nevertheless, we believe the application is sufficiently error-free to be taken into production, despite its flaws that are mostly about unfinished features. In fact, there also should not be many errors of the kind that would allow actually breaking the application from user interface itself. Also, the customer is to be provided with some quick help documentation to for instance reference how to restart the application, in case for whatever reason it would happen that the application or even the whole server would temporarily crash and need restarting.

6 Risks and problems

6.1 Foreseen risks

Following risks were anticipated and encountered (or near missed):

- Technical problems with individual members: for instance, one member's computer broke down.
- Technical problems concerning the group: getting and setting up servers both for testing and production were not as straight-forward as hoped for. As for a more amusing incident though, we also managed to lock ourselves up from the test server at start while learning to use Linux.
- Physical accidents or diseases: one member was in a car accident (no person injuries, luckily!).
- Lack of sufficient skills: this was probably situation in the beginning for almost everyone. Required a lot to learn!
- Tools being ill suitable for our needs: few systems were tested but rejected, as they would not fit to our needs well enough.
- Inequal division of work: certain parts of the project required substantially more work than others, mainly UI development and front end overall, but it proved out difficult for multiple developers to work with it at the same time. Fortunately, our main UI developer apparently also expressed to enjoy the work being done, at least most of the time and probably more when the project manager was not nagging about missing comments in code.
- Related to previous, one member was also unable to find enough time to participate very much for the project starting from later Autumn.
- Being unable to keep up with schedules...fair enough, this should actually be a result of a realized risk, not exactly a risk itself. Check the next paragraph.



6.2 Risks not foreseen

There were two major risks that were unforeseen and came true, both of which are time related: 1. initial misinterpretation of the required working hours and 2. duration of the sprints. 1st issue

was that initially the obtained requirement number was 100 signed hours per team member, which led to quite lax booking in the start due to concern of overbooking time. The team got to hear after around one third of project time was already gone that the actual required hours are 115 h per developer and 160 per manager - a substantial add to the original plan.

2nd issue came up with not paying attention early enough that the third sprint (no. 2) was in fact set as only around half of the length of previous two sprints (two weeks vs four weeks). This combined to the fact that almost half of December went with holidays, the project schedule by the end of the project was essentially one sprint behind the intended. However, this was partially mitigated due to original assumption in planning that “final deadline is always too soon” and as the project went well in schedule for the first half of the project.

Of course, various other minor issues occurred, all of which not being mentioned here. One example would be MMT being unable to handle properly with an incomplete week during new year change. This lead issues with reporting during the end phase of the project, including Charts section not working as expected and one weekly report getting completely left out.

6.3 Communication challenges

There were some comments among the group that certain things would have likely worked out better if we had not been forced to work remotely all the time. Such as showing some things about server setting on spot and so forth. It naturally takes an extra distancing layer if everything is represented through the computer. This is somewhat a question of what one has gotten used to, but it is no denying that several things could have been easier to deal with on-location.

On the other hand, for most parts most team members were well available through Slack and Teams, so if there was something to ask or comment about, people usually were able to react fairly swiftly even while not pre-agreed upon. In addition, for some members it was probably easier to work with as there was no need to physically travel to the university every now and then, especially for those who were doing their other work alongside studies. Also, it could have happened that people would have not paid as much attention to providing written documentation if traditional on-spot locations would have been a regular convention.

6.3.1 Communication practices and tools

Communication tools are largely described in chapters 3. and 3.1. Like stated before, the team had regular meetings of which a short memo would be written up. People would chat in Slack and leave notes in Trello when needed. It was expected usually to give some kind of a summary in Slack after something was done: “ok, I just finished this API feature X and it’s in Git – feel free to implement it to UI”. Also, it was expected that people would tell if they’re for instance unable to participate some meeting, for which cases there should be that memo to check what was talked about.

6.3.2 Communication related challenges

Taking into account the situation, most times communication went alright. However, various small issues of course happened every now and then. Small issues such as technical problems when someone having a bad connection, issues with microphone, computer malfunctioning at the wrong moment or accidentally clicking Teams to shut down instead of sharing screen. On few occasions at the beginning of a meeting nobody knew for sure where is someone, after

which we'd might hear a bit later that the person had gotten something unexpected such as traffic accident, having fallen asleep for too long during a nap or some other unexpected overwhelming cases that can happen to anyone. Most of the time the team could gather up without many issues though, or at least people were able to report beforehand that they'd be unable to attend this time.

Of course, it took a moment for the team to get warmed up. When we started, it was a bunch of mutually unknown people, in which situation it most likely feels more or less awkward to start just working with those people. So, at the very start early in Autumn our communication was not necessarily very solid and regular, and there were many new applications to use without having used to, which most likely did not exactly help. Slack was probably easier to get adopted to, but using Trello's Kanban board and especially updating cards there took a while to get along with.

6.3.3 Strategies and solutions

Like already mentioned, we had our memos and task boards, so that if someone missed something during a meeting, there should still be notes around to pick later about what is going on. Also, like mentioned we attempted to tell as much as possible (without going on too excessive narration) about what we are doing, so that there would not be too badly such situations that people are constantly in a void without awareness about other people and their doings. Of course, we had no Big Brother show going on, so rarely nobody knew about others too accurately all the time.

After a month or two things started to work better when the team members started to know each other better, which naturally made things easier. Everyone was given their share of stating comments regularly every meeting that were kept regularly, and usually everyone had something to say. It's hard to say if it was exactly part of a strategy, but the team seemed rather motivated during the project duration, which no doubt helped people being available and responding relatively well most of the time. People were also encouraged to keep in touch with each other also outside preagreed meetings and without manager involvement, but it is difficult to estimate if such practices had much practical effect or if it just happened that the team members got along.

Nevertheless, if something did not work or if communications were insufficient for some part, it was usually talked about and given this modest hope of improving on that field. Such as writing more comments on the code was directly asked for. Trello was another thing that was taken as a thing to look after during the weekly meetings as a group, which most likely improved the group's awareness of the others work in more concrete fashion and also reminded about actually using the cards themselves as well.

7 Not implemented in this project

7.1 Rejected ideas

Chapter 3.5 deals more accurately about features that were intended to be implemented but never were. These were mostly due to lack of time resources in a long run for the project course duration – of course it could have been possible to have another project worth of features implemented, like with pretty much every software it is possible to continue work till no end. Some ideas such as easily swappable map application was also rejected due to that having probably been too difficult for us to implement.

For many technologies there would have been alternate options to be used, and decision for which of the options would be selected was decided usually either quite intuitively or pragmatically. Therefore for instance Node.js become the backend programming language instead of Python based Django because more developers were already more familiar with JavaScript than Python, and since the web page would contain JavaScript anyway, it was estimated to increase odds of success in the end. SQL vs no-SQL database solutions had somewhat similar results. Alternatively some design tools were tested and others simply felt more comfortable to use, and regarding to the base map application we could not really find a replacement for Google Maps that would have been as easily available to adapt to our purposes, so it was to remain.

7.2 Further development

Had we continued further on the software project, we'd likely implemented or at least investigated yet the following features and practices:

- Historical data would be available to be browsed through UI if wanted. Also, it would be possible to take a log file report about old updates. Historical data is already stored in the database so it's retrieval should not present massive issue.
- Weather (wind & temperature) data could be added to the status updates according to a desired time stamp with further impelmentation of Ilmatieteenlaitos data.
- 2D map view could be alternated with 3D Earth representation view.
- Segments could be edited by dragging the coordinates on the map view if wanted instead of plain typing coordinates.
- There could be third user role: superaddmin, which could not be replicated nor deleted, and which could delete admin roles.
- There'd been kept a proper pilot group testing the application usability, after which the application features would be yet reviewed.
- Status updates could have pictures attached to give a visual data instance of the situation. In addition, easier use of external links could be added
- The application could keep some logged notes about processed events, in order to be able to monitor the application usage and possible debugging.
- There could have been more responsivity within the application, such as telling more clearly if some attempted action was invalid instead of simply not doing it.
- Login design is not very robust, so it could have been implemented differently – including that page reloading would not cause user to log off.
- Segment visibility by multi selection, so that the interface would show up and/or select all segments which are defined currently by this type of snow status.

Lots of the ideas above came up while talking with the customer and listening to their needs, such as historical data reporting. Some other thoughts, such as map based editing, came amidst the team thinking that this could be implemented better than it is now. Also the customer had few existing example applications such as Fatmap, from which some ideas might have gotten adapted from.

8 Lessons learnt

No doubt every member of the team has learned a thing or three, but it also most likely has differentiated between members what was learned. Everyone probably learned something about project running practices and agile methods while at least developers surely learned about Git, programming and things like server/Linux use. Several if not majority of tools and technologies used during the project were new for at least some of the members, so those have required learning new skills.

The team also got to learn how to see better through the eyes of the customer and maybe something about Nordic mountains as well, in order to be able to understand how to better provide features they want and maybe even more importantly need. Respectively, we got this impression that also our customers learned about software development process, in such sense that they would have become more capable of expressing what they want, need and to be better aware about what can be done.

Other things learned or remarked:

- Going through the Trello board cases during weekly meetings and moving cards to Done/Rejected while the team is around seemed like a good idea. First, they were updated and moved whenever finished, which lead to a situation where some developers forgot that cards existed, and others might not notice what some other members had done.
- Time will always run out too early. Even if in the original plan there is explicit intention that the application is largely finished by the end of the 4th sprint and that during the 5th sprint the team should only do finishing and reviewing.
- The team had good and positive spirit overall.
- We felt that our contact and communication with the customer was good.

Things that could have done differently:

- Plans and their implementations could have been more explicit and systematic. For instance, in the end for a moment it was not all clear to the customer or even to the team how much we would be able to implement of our intended features in the end, when we realized that time would run out.
- Likewise and related, minimum acceptance criteria were not exactly set in stone, at least not early on.
- Certain mandatory things such as review preparations could have worked upon earlier than now happened. Even while in some cases it would have been hard to finish things earlier, at least could have checked related instructions and dates earlier, and have them getting reminded about well ahead.
- At least somewhat related to previous, there was a major misnotion about work time remaining realized at around last third or fourth of the project scheduled time. This was partially mitigated, however, by original plan which laid significant emphasis upon “slow landing” and getting MVP done as soon as possible.
- Delegating different tasks among developers might have worked out by sharing the load better.

9 Comments about the course

Positive things:

- The team was most likely lucky to have had such nice people to work with.
- Overall, at least the team felt like this was a successful project in the end, despite few shortcomings with results and anticipated difficulties in the beginning.

Negative things:

- The team got initially incorrect information about number of hours required for course completion. Substantial amount of time had already passed when the correct amount was provided by the team instructor and increase in requirements was significant.

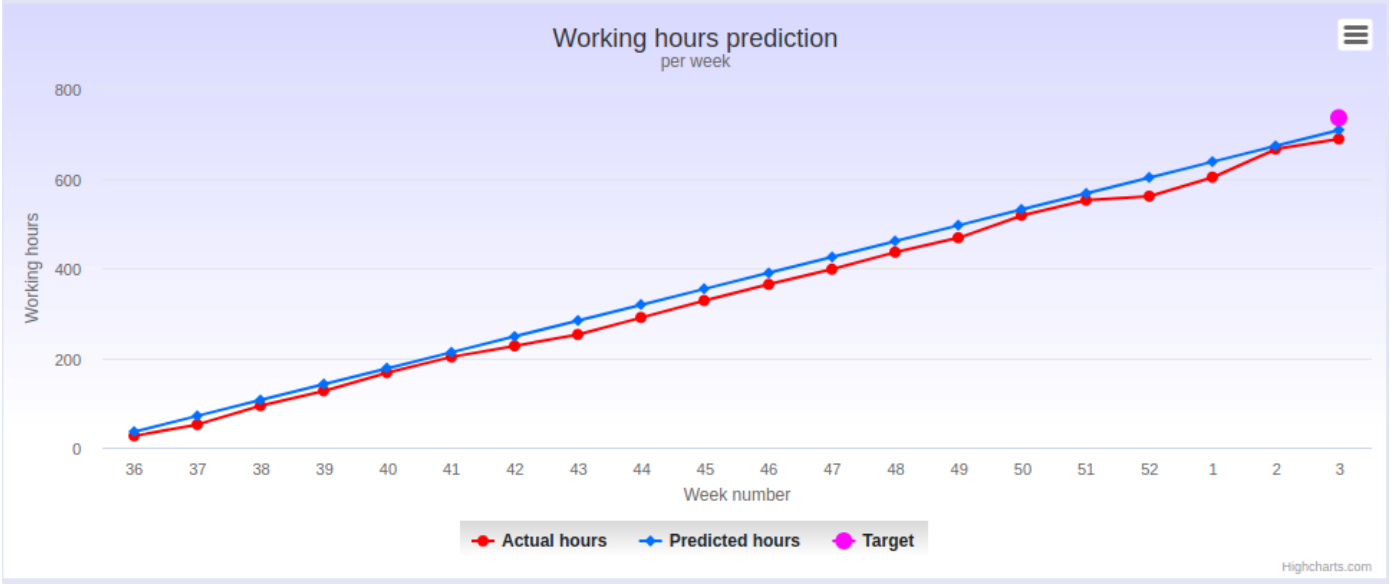
Other things:

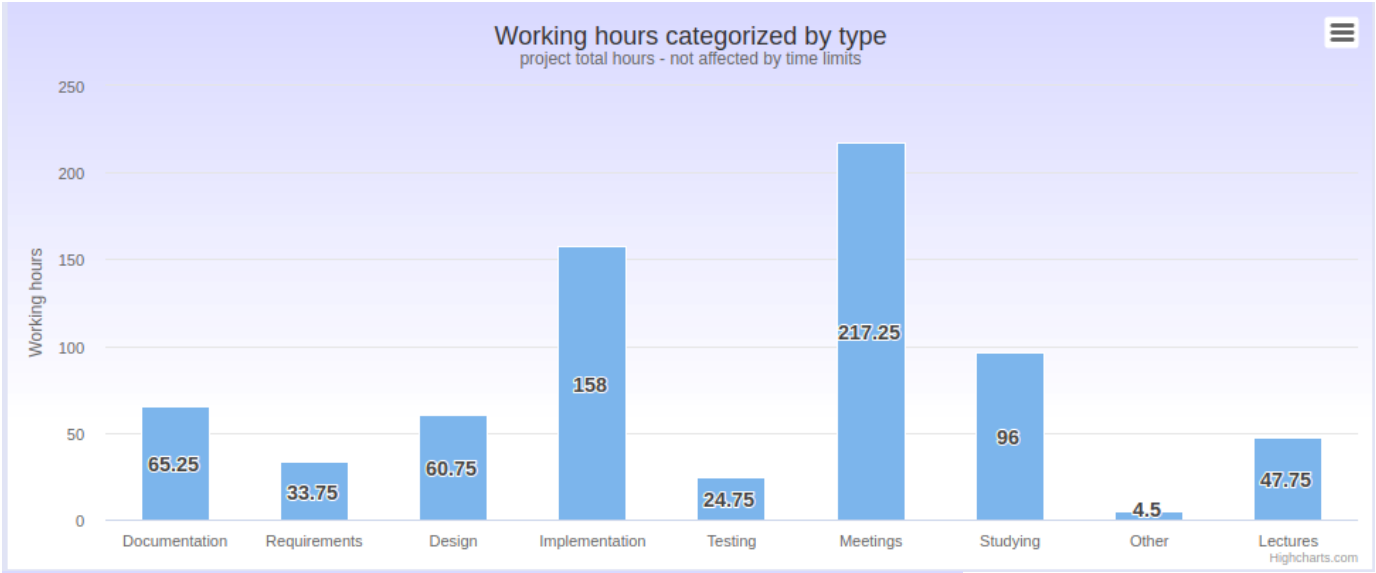
- Course reviewing felt perhaps a bit distanced (pun not intended) due to remote teaching during COVID-19 season.
- The team never met as whole in person, also due to the pandemic.

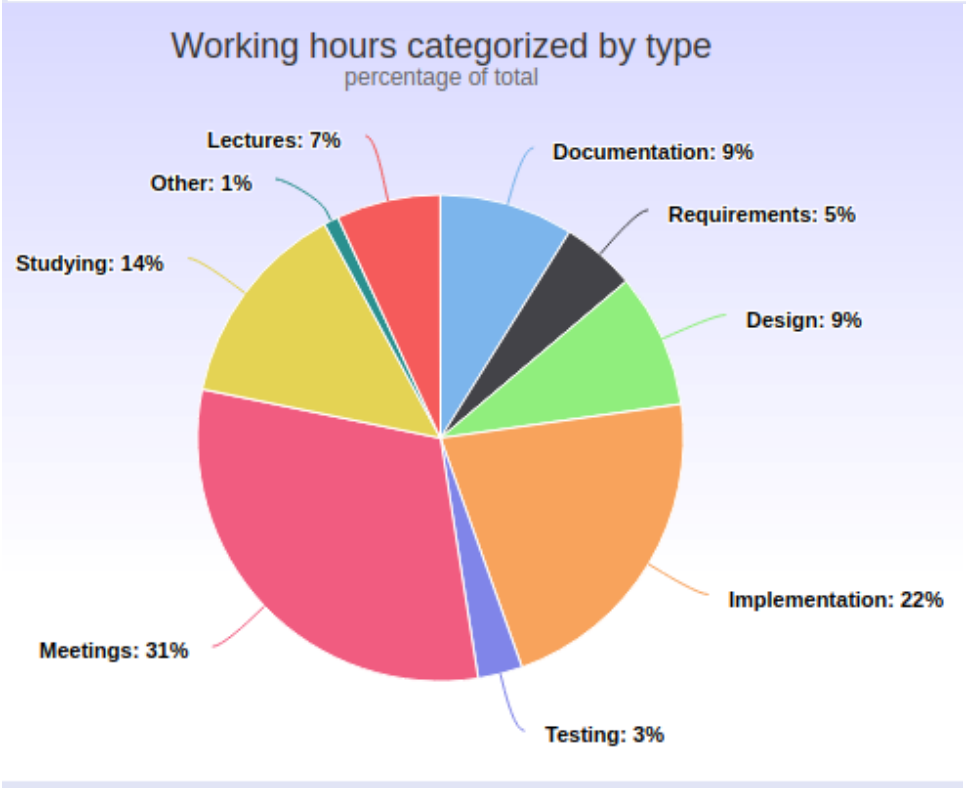
10 Statistics

This final chapter counts largely as appendix section with project management statistics that were available as precise graphs. The date of taking records was 26.1.2021, while the final review date is scheduled for 29.1.2021. Some of the statistical information is rather amidst other reporting text, since they seemed a bit orphaned in the end and it did not seem relevant to repeat the same graphs and numbers here again in same form.

	Name	Project Role	Working hours	Last seen	Actions
	Pekka (supervisor) Mäkiaho	supervisor			View
	Sakari Eskelinen	manager	161 / 160	26.01.2021	View Edit Delete
	Aarre Leinonen	developer	117.75 / 115	25.01.2021	View
	Arttu Lakkala	developer	114 / 115	16.01.2021	View
	Kaisa Kauhanen	developer	123.25 / 115	22.01.2021	View
	Markku Nirkkonen	developer	138 / 115	20.01.2021	View
	Johanna Pekki	developer	54 / 115	13.01.2021	View
	Total		**708 / 735**		




Working hours categorized by type
project total hours - not affected by time limits
Working hours
250
200
150
100
50
0
65.25
33.75
60.75
158
24.75
217.25
96
4.5
47.75
Documentation
Requirements
Design
Implementation
Testing
Meetings
Studying
Other
Lectures
Highcharts.com


Working hours categorized by type
percentage of total
Lectures: 7%
Documentation: 9%
Other: 1%
Requirements: 5%
Studying: 14%
Design: 9%
Implementation: 22%
Meetings: 31%
Testing: 3%

Total hours

Note that there is a reporting error with statistics application for week 53 as it is missing. The reporting tool would not properly handle the incomplete week during the New Year.

Requirements

Regarding to requirements it can be mentioned that the statistics do not take Week 4 into account anymore, since weekly reports are generated by the end of a week. At least most of remaining Sprint Backlog items are booked out for good by 27.1.2021 the latest – a day after this report submission.